Corso di Crittografia UNIMI

Crittografia (Università degli Studi di Milano)

Crittografia

Ilaria Salbe

salbeilaria@gmail.com

Appunti per il CdL Magistrale:
Sicurezza Informatica

Università degli Studi di Milano
Anno Accademico 2019/2020

Indice

1 Introduzione

La crittografia o criptografia (dal greco $\kappa\rho\upsilon\pi\tau o\zeta$ [kryptós], "nascosto", e $\gamma\rho\alpha\phi\iota\alpha$ [graphía], "scrittura") è la branca della crittologia che tratta delle "scritture nascoste", ovvero dei metodi per rendere un messaggio "offuscato" in modo da non essere comprensibile/intelligibile a persone non autorizzate a leggerlo. La crittologia è la scienza che si occupa della comunicazione in forma sicura e di solito segreta. Da un lato comprende l'ideazione di metodi sempre più sicuri per occultare il reale significato di determinati segni (crittografia), dall'altro riguarda la decifrazione di testi occultati senza conoscerne a priori il metodo usato (crittoanalisi). Quindi, la crittoanalisi è la scienza di risolvere i crittosistemi per recuperare l'informazione nascosta. Crittografia e crittoanalisi sono le due facce della stessa medaglia, una medaglia che nel corso della storia ha dato più importanza all'una o all'altra, alternativamente.

Fino dall'antichità l'uomo ha sentito l'esigenza di trasmettere messaggi segreti; infatti i primissimi esempi di crittografia sono stati scoperti in alcuni geroglifici egiziani risalenti a più di 4500 anni fa. Nella città di Menet Khufu, che sorge sulle sponde del Nilo, è stata ritrovata una delle più antiche testimonianze di alterazione di una scrittura: si tratta di una incisione funeraria di circa 4000 anni fa realizzata sulla tomba di un nobile. In questo caso, l'intento della trasformazione consisteva semplicemente nel conferire dignità e onoreficenza alla persona defunta. Nello stesso periodo, tuttavia, cominciarono a comparire anche trasformazioni aventi lo scopo di rendere misterioso ed arcano il significato delle parole.

Fonti preziose di esempi di scritture segrete sono i testi sacri. Nel Vecchio Testamento gli storici hanno evidenziato 3 tipi di trasformazioni: l'Atbash, l'Albam e l'Atbah. La prima è universalmente accettata; le seconde sono maggiormente discusse. L'Atbash ebraico, è una tecnica di trasformazione ad alfabeto capovolto: il primo carattere dell'alfabeto viene sostituito con l'ultimo dell'alfabeto, il secondo carattere viene sostituito con il penultimo, e così via. Il nome deriva dalla regola di trasformazione. Infatti la prima lettera dell'alfabeto ebraico è Aleph, l'ultima è Taw, la seconda è Beth e la penultima è Shin. Concatenando le iniziali di queste lettere nell'ordine si ottiene la parola Atbash. L'Atbash viene utilizzato nel libro del profeta Geremia per cifrare il nome della città di Babilonia. L'Albam, invece, richiede che l'alfabeto venga diviso in due parti e che ogni lettera venga sostituita con la corrispondente dell'altra metà. Infine, l'Atbah richiede che la sostituzione soddisfi una relazione di tipo numerico. Le prime nove lettere dell'alfabeto vengono sostituite in modo tale che la somma della lettera da sostituire e della lettera sostituente risulti uguale a 10. Quindi, per esempio, Aleph (prima lettera dell'alfabeto) viene sostituita con Teth (nona lettera dell'alfabeto). Per le restanti lettere dell'alfabeto deve valere una regola simile con somma pari a 28 in decimale (per esempio, la 13-esima lettera viene sostituita con la 15-esima, etc.).

In India, invece, forme di crittografia furono concretamente praticate. In diversi testi indiani, infatti, sono presenti riferimenti a forme di scritture segrete. Nell'Artha-Sastra, un testo classico sugli affari di stato, si sottolinea l'importanza delle scritture segrete nei servizi di spionaggio, mentre nel Latila-Vistara, un libro che esalta le virtù di Budda, si narra di come questi stupisse il proprio insegnante parlando di scritture perpendicolari, disordinate, etc.. Nel Kama-Sutra, invece, tra le 64 arti (yogas) che la donna deve conoscere e praticare c'è l'arte delle scrittura segreta. La 44-esima e, in particolare, la 45-esima arte (mlecchita-vikalpa) trattano di regole di trasformazione delle parole basate essenzialmente sulla sostituzione dei caratteri.

Nessun esempio dimostra le potenziali conseguenze della crittoanalisi in modo più drammatico del processo a Maria di Scozia, il cui esito dipese esclusivamente dallo scontro tra i suoi cifratori e i decrittatori di Elisabetta I. Nel 1586 Maria Stuarda, regina di Scozia, fu condannata a morte per aver cospirato contro la cugina Elisabetta. La congiura, organizzata da Anthony Babington, prevedeva:

- La liberazione di Maria dalla prigionia in Inghilterra;
- L'uccisione di Elisabetta;

- Una ribellione alla religione protestante.

Sir Francis Walsingham, segretario di stato, provò che Maria aveva preso parte alla congiura. Maria e Babington comunicavano grazie a

- Un corriere (Gilbert Gifford)
- Un birraio, che nascondeva i messaggi dentro lo zipolo delle botti di birra
- Un cifrario, costituito da:
  - 23 simboli che sostituivano le lettere;
  - 35 simboli, che sostituivano parole o frasi;
  - 4 nulle e un simbolo per le doppie.

Gifford consegnava a Walsingham tutti i messaggi, che venivano decifrati da Thomas Phelippes. Maria firmò la sua condanna a morte rispondendo alla lettera di Babington. Babington e complici furono arrestati e squartati vivi. Maria fu decapitata l'8 febbraio 1587.

La ricerca di nuovi sistemi crittografici diede un grande impulso alla crittografia durante il periodo antecedente la Seconda Guerra Mondiale. Già dall'inizio del XX sec, infatti, stava nascendo l'esigenza di poter usufruire di una crittografia sicura e, soprattutto, veloce e facilmente utilizzabile: per questo nacquero le prime macchine cifranti. Fin dalla fine del XIX sec, lo sviluppo della crittografia rese necessaria una progressiva automatizzazione dei metodi di cifratura e decifratura, le macchine cifranti nacquero quindi non tanto per rendere i sistemi crittografici più sicuri, ma semmai per velocizzarli. La più famosa di queste macchine è sicuramente Enigma, brevettata dall'ingegnere tedesco Arthur Scherbius nel 1918 e adottata dell'esercito e dalla marina tedesca durante la Seconda Guerra Mondiale. Nell'agosto del 1939 i Britannici costituirono la scuola dei codici e dei cifrari a Bletchley Park, dove reclutarono i migliori crittoanalisti, matematici e scienziati. Sfruttando anche le conoscenze raggiunte dagli alleati polacchi, durante la guerra, gli inglesi continuarono a forzare sistematicamente i messaggi cifrati con Enigma e dal 1941 anche quelli cifrati con la più sofisticata macchina Lorenz. La crittografia giocò quindi un ruolo di fondamentale importanza durante tutta la durata della guerra e, ad esempio, fu fondamentale per lo Sbarco in Normandia. Infatti Eisenhower e Montgomery erano in grado di leggere tutti i messaggi degli alti comandi tedeschi, che usavano la macchina Lorenz; ebbero così conferma che Hitler aveva creduto alla falsa notizia di un imminente sbarco alleato nei pressi di Calais, e aveva concentrato le sue migliori truppe in quella zona. Poterono quindi ordinare lo sbarco in Normandia sicuri che avrebbe incontrato ben poca resistenza.

La crittografia trova spazio anche in letteratura. Un esempio è Il codice Da Vinci di Dan Brown.

La crittografia moderna si differenzia notevolmente dalla crittografia di cui si è parlato fin ora; l'avvento dei computer infatti ha rivoluzionato profondamente sia i sistemi crittografici sia il modo di vedere e utilizzare la crittografia. Molti sistemi crittografici analizzati in precedenza e considerati ragionevolmente sicuri fino al XIX secolo, possono oggi essere forzati in tempi brevissimi grazie alla velocità di elaborazione del computer. Nell'era dei computer la crittografia è "uscita dai campi di battaglia" e viene utilizzata da ogni persona, più o meno consapevolmente, nella vita di tutti i giorni: per prelevare soldi con un bancomat, nell'effettuare acquisti su internet o, semplicemente, chiamando con un telefono cellulare. La crittografia è diventata uno strumento di massa, atto a proteggere i segreti di stato tanto quanto i dati che noi vogliamo, o almeno vorremmo, rimanessero privati.

La crittografia moderna si occupa del progetto e della valutazione di metodi e tecniche per la protezione dell'informazione, non solo segretezza delle comunicazioni. É fondata su solide basi matematiche.

Possiamo distinguere tre fasi principali nella storia della crittografia:

- Primo stadio: dalle prime civiltà fino al secolo scorso. Gli algoritmi sono sviluppati e implementati "a mano".
- Secondo stadio: seconda guerra mondiale con l'apparizione delle prime macchine cifranti.
- Terzo stadio: ultimi 50 anni. Si sviluppa l'utilizzo di computer e fondamenti matematici.

Nei documenti fisici, a differenza dei documenti digitali, la copia è distinguibile dall'originale; l'alterazione lascia tracce e la prova di autenticità si basa su caratteristiche fisiche (firma, ceralacca, ...). La firma digitale può essere un metodo per provare l'autenticità di un documento digitale.

Introduciamo ora il concetto di Romantic Cryptography noto anche come "Matching without embarrassment". E se desideri esprimere il tuo amore a qualcuno, ma temere le conseguenze se lui o lei non restituisce i tuoi sentimenti? É un problema secolare: quanti di noi conoscono qualcuno che chiede a un amico di parlare con lui/lei per "vedere se le/gli piaccio?" Mostriamo come Alice e Bob possono stabilire se si amano a vicenda, ma senza l'imbarazzo di rivelare che lo fanno se l'altra parte non condivide i propri sentimenti. Se sono interessati devono mettere prima l'asso rosso e poi quello nero, se no al contrario, poi mettono le carte nel mazzo e shiftano le carte. Se compaiono 3 carte rosse di seguito, sarà un sì. Una terza persona se vede che esce un no, non capisce chi ha detto di no. Il problema è che se un partecipante avvia il protocollo è perché è interessato. La soluzione è quella di fare incontri casuali.

Un protocollo o schema definisce le interazioni fra le parti per ottenere le proprietà di sicurezza desiderate, dove le parti sono le entità coinvolte nello schema, mentre le proprietà di sicurezza sono gli obiettivi di sicurezza che si vogliono ottenere con lo schema. Quello che si vuole ottenere è un tubo, ovvero un canale sicuro per lo scambio di messaggi. Questo purtroppo è lo scenario ideale, infatti nella realtà ci sono attaccanti che cercano di violare questo canale sicuro con intenzioni malevole. Addirittura è possibile che uno dei due interlocutori voglia agire con finalità malevoli. I nemici possono essere esterni o interni. Inoltre, i nemici, oltre che "umani", possono essere hardware o software.

I protocolli sono pensati per essere eseguiti in determinati ambienti e situazioni, ma gli ambienti possono diventare "ostili": le condizioni di esecuzione possono mutare (ad esempio, dopo l'iniezione di un virus worm). Nella pratica alcuni aspetti tralasciati diventano importanti: ad esempoi il cellulare si attiva ad ogni chiamata, ma pubblica automaticamente la posizione e altre informazioni.

Un protocollo, inoltre, specifica quali tecniche adottare:

- Tecniche di cifratura
  - Cifrari simmetrici o a chiave privata (cifrari a blocchi, stream cipher)
  - Cifrari asimmetrici o a chiave pubblica
- Tecniche per autenticazione ed integrità
  - Funzioni Hash
  - MAC
  - Firme digitali
- Scambio e condivisione di chiavi. In crittografia, il principio di Kerckhoffs (noto anche come assunzione, assioma o legge di Kerckhoffs), fu enunciato dal crittografo olandese Auguste Kerckhoffs alla fine del 1880. Esso afferma che la sicurezza di un crittosistema non deve dipendere dal tenere celato l'algoritmo crittografico ma solo dal tenere celata la chiave.
- Generazione di numeri pseudo-casuale

Gli obiettivi delle diverse tecniche di crittografia sono:

- Confidenzialità: le informazioni trasmesse/memorizzate sono accessibili in lettura solo da chi è autorizzato.
- Autenticazione: essere sicuri della persona con cui si entra in comunicazione. L'autenticazione può essere fatta a diversi livelli:
  - Entità (sistema biometrico o login)
  - Temporale (timestamp)
- Integrità: solo chi è autorizzato può modificare l'attività di un sistema o le informazioni trasmesse. Con il termine modifica si intende: scrittura, cambiamenti, cancellazione, creazione, ritardi, replay e riordino di messaggi, ...
- Non-ripudio: Chi invia/riceve non può negare la trasmissione o la ricezione del messaggio
- Anonimia: protezione dell'identità o del servizio utilizzato. Internet non garantisce l'anonimato: ogni computer connesso ad Internet ha un indirizzo IP unico. É possibile associare all'indirizzo IP l'identità di una persona o di un'azienda. Analizzando il traffico si possono ricavare le preferenze dell'utente. Da chi si desidera privacy:
  - Amministratori Web server
  - Amministratori locali del sito del client
  - Internet Provider
  - Chiunque possa intercettare il traffico della rete

  Gli utenti possono voler proteggere:
  - Identità (nome, e-mail,...)
  - Locazione (indirizzo IP, nome dominio, locazione fisica)
  - Nome dei siti visitati
  - Richieste a motori di ricerca
  - Abitudini sul Web (acquisti,...)
  - Oggetti salvati (script, applet, imagini, file HTML,...)

  Alcune tecniche per mantenere una certa anonimia:
  - Remailers: invio di posta elettronica con mittente anonimo
  - Crowds: accesso anonimo a siti WEB
  - Anonymizer: accesso anonimo a siti WEB
  - Autenticazione Anonima: accesso anonimo a risorse

2 Crittografia Classica

Dall'antichità fino a pochi anni fa con la parola Crittografia si intendeva la capacità di sviluppare comunicazioni segrete, per lo più utilizzata per usi militari e diplomatici. Oggi, con la parola crittografia si intende lo studio di tecniche ed applicazioni per la protezione dell'informazione basata sull'esistenza di problemi difficili.

Le comunicazioni segrete possono essere un successo, sia grazie alla crittografia, ma anche grazie alla steganografia.

La crittografia è la scienza che studia tecniche e metodologie per cifrare (codificare) un testo in chiaro (in inglese plaintext), al fine di produrre un testo cifrato (ciphertext) comprensibile solo ad un ricevente legittimo (receiver), il quale possiede l'informa- zione sufficiente (detta chiave) per decifrarlo, recuperando il testo in chiaro. La crittografia fa parte, insieme alla crittoanalisi, della crittologia. La crittoanalisi studia come decrittare un testo cifrato per ottenere il testo in chiaro: il verbo decrittare indica l'azione compiuta da un'entità che non possiede la chiave per recuperare, in modo legittimo, il testo in chiaro. In letteratura, suddetta entità viene indicata con il termine ascoltatore (eavesdropper) oppure avversario o anche nemico. Lo scopo della crittografia è di produrre un messaggio cifrato m (che viaggerà poi a partire da un mittente, in inglese sender, fino ad un receiver), in modo che nessun avversario sia in grado di decrittare il contenuto di m.

Il termine steganografia è composto da due parole di origine greca, stèganos (che significa nascosto) e gràfein (che significa scrivere). Tale termine indica quindi "la scrittura nascosta" o, per meglio dire, l'insieme dei metodi e delle tecniche che permettono a due o più entità (siano esse persone o macchine) di comunicare in modo tale da occultare agli occhi di un eventuale ascoltatore, non tanto il contenuto (come nel caso della crittografia), ma la stessa esistenza di una comunicazione riservata. In altre parole, la steganografia è l'arte di nascondere un messaggio (che si vuole rimanga segreto) all'interno di un'altro messaggio contenitore (che si vuole sia pubblico) di aspetto diverso, spesso totalmente diverso, e non sospetto.

Già in epoche remote, sono state escogitate tecniche per nascondere la presenza di un canale di comunicazione tra parti distinte. Erodoto (486 – 425 A.C.) racconta la storia di un nobile persiano che fece tagliare a zero i capelli di uno schiavo fidato al fine di poter tatuare un messaggio sul cranio di quest'ultimo; una volta che i capelli furono ricresciuti allo schiavo, questi fu inviato a destinazione, con la sola istruzione di tagliare nuovamente i propri capelli. Sempre nelle storie di Erodoto si racconta dell'espediente della tavoletta di cera. Demerato, in esilio, avvisa gli spartani del progetto di invasione da parte di Serse, re dei Persiani. Demarato inviò a Sparta un messaggio con la notizia della guerra imminente su una tavoletta di legno, che ricoprì completamente di cera per nascondere il testo al messaggero e agli altri Persiani che avessero intercettato la tavoletta lungo il viaggio. Erodoto racconta che fu la regina Gorgo, moglie di Leonida I, a capire per prima che sotto quella che sembrava una tavoletta di cera ancora da scrivere si celava un messaggio nascosto.

Alcune tecniche di steganografia sono:

- Disposizione delle lettere/parole in un messaggio innocuo
- Contrassegno dei caratteri (Ripasso a matita o Fori sulle lettere).
- Inchiostro invisibile

La steganografia moderna è entrata nel mondo nel 1985 con l'avvento di personal computer applicati a problemi di steganografia classica. Alcuni esempi di steganografia moderna sono:

- Nascondere messaggi all'interno dei bit meno significativi di immagini o audio.
- Nascondere dati all'interno di dati crittografati o all'interno di dati casuali.
- Immagini incorporate nel materiale video (opzionalmente riprodotte a velocità più bassa o più veloce).
- Watermarking: inclusione di informazioni all'interno di un file multimediale o di altro genere, che può essere successivamente rilevato o estratto per trarre informazioni sulla sua origine e provenienza. In italiano può essere tradotto come filigrana elettronica.

Riassumendo, a differenza della crittografia, il cui obiettivo è quello di rendere protetto un dato canale di comunicazione da un eventuale ascoltatore non autorizzato, impedendo a quest'ultimo di accedere al contenuto del canale, le tecniche steganografiche si sforzano di nascondere

la mera presenza del canale protetto, utilizzando un canale pubblico (il cui contenuto è accessibile ad un ascoltatore malizioso) come veicolo per lo scambio di messaggi che devono rimanere riservati. Il fine ultimo della crittografia è quello di proteggere il contenuto di un messaggio, preservando, attraverso un metodo di cifratura, la confidenzialità di tale messaggio. Anche se un testo cifrato è perfettamente riconoscibile come tale, è fondamentale che un estraneo, che venga in possesso del messaggio, non possa in alcun modo accedere alle informazioni contenute in esso o poterle modificare. Lo scopo della steganografia, invece, è quello di impedire che un estraneo abbia il seppur minimo indizio che stia avvenendo un trasferimento di dati confidenziali. Nessuno deve poter sospettare che si voglia trasmettere materiale riservato: la steganografia fallisce nel momento in cui la trasmissione viene scoperta, anche se non si è in grado di decrittarne il contenuto. Il vantaggio della steganografia è che le parti possono nascondere di essere in comunicazione, ma se il corriere è attentamente perquisito o una lettera è attentamente esaminata (Figura 1), il messaggio può essere scoperto, perdendo così la segretezza.

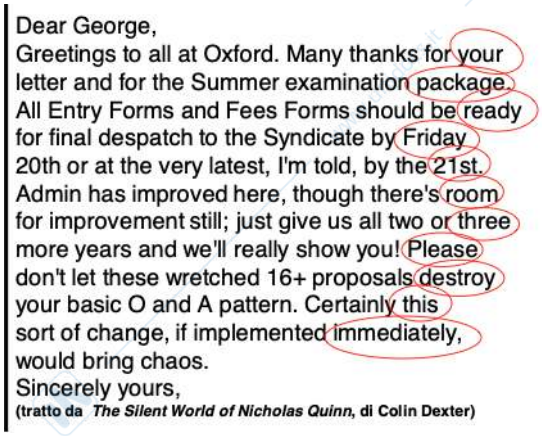

Dear George,
Greetings to all at Oxford. Many thanks for your letter and for the Summer examination package. All Entry Forms and Fees Forms should be ready for final despatch to the Syndicate by Friday 20th or at the very latest, I'm told, by the 21st. Admin has improved here, though there's room for improvement still; just give us all two or three more years and we'll really show you! Please don't let these wretched 16+ proposals destroy your basic O and A pattern. Certainly this sort of change, if implemented immediately, would bring chaos.
Sincerely yours,
(tratto da *The Silent World of Nicholas Quinn*, di Colin Dexter)

Figura 1: Esempio di messaggio scoperto

Un esempio di metodi antichi di cifratura è La scacchiera di Polibio, nota anche come quadrato di Polibio, un sistema crittografico inventato dallo storico greco Polibio verso il 150 a.C.. Si basava sul frazionamento dei caratteri del messaggio in chiaro così che potessero essere rappresentati utilizzando un più piccolo insieme di simboli. La scacchiera originale è costituita da una griglia composta da 25 caselle ordinate in cinque righe ed altrettante colonne. Le lettere dell'alfabeto vengono inserite da sinistra a destra e dall'alto in basso. Le righe e le colonne sono numerate: tali numeri sono gli indici o "coordinate" delle lettere costituenti il messaggio in chiaro. La trasposizione avviene sostituendo ad ogni lettera del messaggio un numero le cui cifre rappresentano il numero di riga e di colonna della sua posizione nella scacchiera. Ad esempio, operando sulla parola CASA: la lettera W si trova nella 1° riga e 3° colonna, per cui il suo codice sarà 13; la lettera A si trova sulla 1° riga e 1° colonna, per cui il suo codice sarà 11. Proseguendo con questo metodo il risultato finale per CASA sarà 13 11 43 11.

Un altro esempio è il cifrario di Cesare, uno dei più antichi algoritmi crittografici di cui si abbia traccia storica. É un cifrario a sostituzione monoalfabetica in cui ogni lettera del testo in chiaro è sostituita nel testo cifrato dalla lettera che si trova un certo numero di posizioni dopo nell'alfabeto. In particolare, Cesare utilizzava uno spostamento di 3 posizioni (la chiave era dunque 3), Figura 2.

Testo in chiaro	a	b	c	d	e	f	g	h	i	k	l	m	n	o	p	q	r	s	t	v	x	y	z
Testo cifrato	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	V	X	Y	Z	A	B	C

Figura 2: Cifrario di Cesare

Per cifrare un messaggio, basta prendere ogni lettera del testo in chiaro e sostituirla con la corrispondente lettera della riga testo cifrato. Per decifrare, viceversa. Ecco un semplice esempio italiano:

OMNIA	GALLIA	EST	DIVISA	IN	PARTES	TRES
RPQLD	JDOOLD	HVW	GLYLVD	LQ	SDUWHV	WUHV

2.1 Cifrari Simmetrici

Con crittografia simmetrica, o crittografia a chiave privata, si intende una tecnica di cifratura. Rappresenta un metodo semplice per cifrare testo in chiaro dove la chiave di crittazione è la stessa chiave di decrittazione, rendendo l'algoritmo molto performante e semplice da implementare. Tuttavia presuppone che le due parti siano già in possesso delle chiavi, richiesta che non rende possibile uno scambio di chiavi con questo genere di algoritmi. Lo scambio avviene attraverso algoritmi a chiave asimmetrica o pubblica, generalmente più complessi sia da implementare che da eseguire ma che permettono questo scambio in modo sicuro. Dopodiché la comunicazione verrà crittata usando solo algoritmi a chiave simmetrica per garantire una comunicazione sicura ma veloce.

La prima cosa da mettere in luce è la consuetudine di classificarli in due categorie. Per capire su cosa si basa la classificazione, consideriamo una coppia di utenti A e B, indichiamo con AB la chiave segreta che hanno concordato di usare e supponiamo che A sia il mittente e B il destinatario di un messaggio riservato. Sia infine m la stringa di bit in chiaro cifrata da E_k e decifrata da D_k. Le attività che A e B devono svolgere sono:

1. A: calcola $c = E_{AB}(m)$ e trasmette c.
2. B: riceve c e calcola $D_{AB}(c) = D_{AB}(E_{AB}(m)) = m$

In generale m è solo una parte del testo in chiaro M che A intende inviare a B: le attività 1. e 2. devono dunque essere ripetute per M/m volte. La dimensione di m ed il modello cui si ispira il meccanismo sono i parametri che hanno portato a distinguere due differenti tipi di cifrario:

- Il Cifrario a flusso (stream cipher) si ispira a One-time pad, trasforma pochi bit alla volta (tipicamente un solo bit od un byte) e risulta particolarmente adatto quando occorre proteggere singoli dati, generati uno dopo l'altro (trasmissione seriale di caratteri ASCII, comunicazione fonica digitalizzata, ecc.). Al flusso di dati input corrisponde un flusso di chiavi (keystream).
- Il Cifrario a blocchi (block cipher) trasforma blocchi formati da molti bit (64, o 128, o più) e risulta particolarmente adatto quando occorre proteggere strutture di dati (trasmissione di pacchetti, archiviazioni di file su hard disk, ecc.).

Tecnica di sostituzione e tecnica di trasposizione sono i metodi fondamentali per codificare il messaggio in chiaro per acquisire il rispettivo testo cifrato. Questi due metodi sono gli elementi costitutivi fondamentali delle tecniche di crittografia e possono anche essere utilizzati insieme, il cosiddetto codice del prodotto. La differenza essenziale tra la tecnica di sostituzione e la tecnica trasposizionale è che la tecnica di sostituzione sostituisce le lettere del testo in chiaro con altre

lettere, numeri e simboli. D'altra parte, le tecniche di trasposizione non sostituiscono la lettera, invece cambia la posizione del simbolo. Nella sostituzione, ciascun elemento viene mappato su un altro elemento. Mentre, nella trasposizione, gli elementi in chiaro vengono scambiati di posto. Il requisito fondamentale per poter applicare le seguenti operazioni è che le informazione non devono essere perse, ovvero le operazioni devono essere reversibili.

Una generalizzazione del cifrario di Cesare è data dai cifrari con shift, nei quali anziché fissare lo shift a tre posizioni, lo scelgo come chiave dell'algoritmo, ossia:

- K rappresenta la chiave
- $c = E_k(m) = (m + k) mod(26)$ (cifratura)
- $m = D_k(c) = (c - k) mod(26)$ (decifratura)

Si hanno solamente 26 possibili chiavi: la lettere A può essere mappata in A,B,..Z. Escludendo $k = 0$, per cui avremmo il semplice alfabeto, le chiavi possibili sono 25.

Dato un testo cifrato Y deve essere difficile risalire alla chiave usata K. Una volta individuata la chiave, conoscendo D_K sarà possibile decrittografare tutti i messaggi. Mediante l'attacco a forza bruta si tenta ogni possibile chiave fino ad ottenere un risultato. Per avere un risultato, in media, bisogna provare la metà delle chiavi. Ad esempio, per quanto riguarda il cifrario di Cesare (o, in generale, il cifrario con shift) si potrebbe semplicemente provare una chiave alla volta con un attacco di forza bruta. Il problema è riconoscere quando abbiamo trovato il plaintext, spesso non è così semplice (Figura 3).

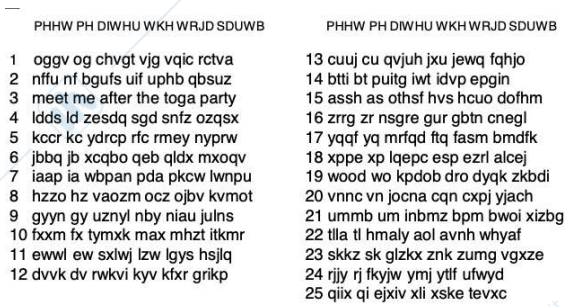


Figura 3: Forza bruta su shift

Ogni schema di cifratura deve avere uno spazio delle chiavi che non è vulnerabile ad una ricerca esaustiva (o attacco a forza bruta). Questa è una condizione necessaria ma non sufficiente!

Nei cifrari monoalfabetici, anziché semplicemente shiftare le lettere, si potrebbe effettuare una permutazione arbitraria delle lettere dell'alfabeto. Ogni lettera del plaintext si mappa ad un'altra lettera random del ciphertext. Quindi la chiave è costituita da una stringa di 26 lettere. In pratica, al posto di una lettera dell'alfabeto in chiaro metti quella corrispondente dell'alfabeto cifrante. Rispetto allo shift della Figura 4, partendo dal testo in chiaro: C A S A, si ottiene il testo cifrato: T O P O.

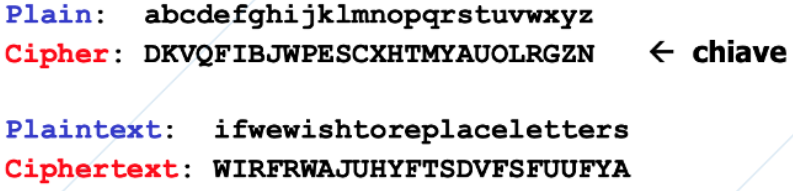


Figura 4: Cifrari monoalfabetici

Quante chiavi sono possibili? Con un alfabeto di 21 lettere, circa 50 miliardi di miliardi di possibilità. Con un alfabeto di 26 lettere, si ottengono $26! = 4 \times 1026$ (circa 288). Potremmo pensare di aver raggiunto un adeguato livello di sicurezza e invece il problema è legato alle caratteristiche del linguaggio naturale che rendono possibili attacchi alternativi.

Dimensione Chiave	Numero Chiavi	Tempo per 1critt/μs	Tempo per 10^6critt/μs
32	$2^{32}=4{,}3\times10^{9}$	2^{31}μs 35,8 minuti	2,15 ms
56	$2^{56}=7{,}2\times10^{16}$	2^{56}μs 1142 anni	10,01 h
128	$2^{128}=3{,}4\times10^{38}$	2^{127}μs $5{,}4\times10^{24}$anni	$5{,}4\times10^{18}$anni
168	$2^{168}=3{,}7\times10^{50}$	2^{167}μs $5{,}9\times10^{36}$anni	$5{,}9\times10^{30}$anni
26car	$26!=4\times10^{26}$	2×10^{26}μs $6{,}4\times10^{12}$anni	$6{,}4\times10^{6}$anni

Figura 5: Velocità per attacco a forza bruta

La tabella rappresentata nella Figura 5 rappresenta i tentativi per risolvere un sistema di cifratura. 2^{80} è un limite ragionevole per la tecnologia moderna.

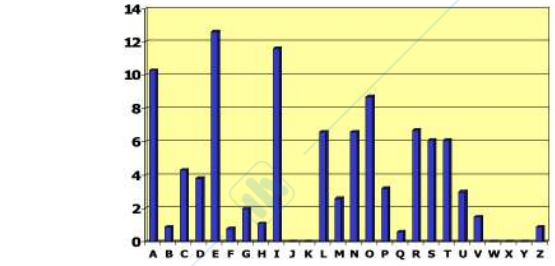
Con il termine unconditionally secure si intende che indipendentemente dal tempo e dalle risorse a disposizione è impossibile decrittografare il testo cifrato. Conosciamo solo un sistema così sicuro! Mentre con il termine computationally secure, si intende che il tempo richiesto per violare la cifratura è grande e comunque superiore alla vita utile delle informazioni contenute.

L'analisi crittografica è basata sulla natura dell'algoritmo e sfrutta conoscenza su: testo in chiaro e algoritmo. Con il metodo della forza bruta, invece, si tenta ogni possibile chiave fino ad ottenere un risultato. In media bisogna provare la metà delle chiavi.

Vediamo un esempio di analisi crittografica sui cifrari monoalfabetici di prima. Il concetto chiave è che la sostituzione monoalfabetica non cambia la frequenza relativa delle lettere, quindi:

- Si calcola la frequenza delle lettere nel ciphertext;
- Si comparano i dati ottenuti con le tabelle della lingua usata;
- Si parte dai picchi, facendo delle ipotesi, e poi via via con delle prove fintanto che si arriva al testo in chiaro.

Esistono anche delle tabelle con le frequenze di doppie e triple. La Figura 6 rappresenta le frequenze delle occorrenze delle lettere in una determinata lingua.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
italiano	10,3	0,9	4,3	3,8	12,6	0,8	2,0	1,1	11,6	0,0	0,0	6,6	2,6	6,6	8,7	3,2	0,6	6,7	6,1	6,1	3,0	1,5	0,0	0,0	0,0	0,9
inglese	7,3	1,3	3,5	4,3	12,8	3,0	2,0	3,5	7,8	0,3	0,5	3,7	2,8	7,8	7,5	2,8	0,5	8,5	6,0	9,3	3,0	1,5	1,5	0,5	2,3	0,3
francese	8,3	1,3	3,3	3,8	17,8	1,3	1,3	1,3	7,3	0,8	0,0	5,8	3,2	7,2	5,7	3,7	1,2	7,3	8,3	7,2	6,3	1,8	0,0	0,0	0,8	0,0

Figura 6: Frequenze occorrenza lettere

Alcune contromisure a quest'analisi sono:

- Omofoni: Molti simboli per cifrare singoli caratteri frequenti. Ad esempio:

testo in chiaro: E

testo cifrato: $\tilde{O}$ $\tilde{N}$ ®

Così facendo, si abbassano le frequenze dei simboli del testo cifrato: da 12.6 per la lettera E a 3.15 per i simboli random.

- Nulle: Aggiungere simboli meno frequenti in posizioni da non alterare il significato.

  testo in chiaro: QUELQRAMODELQLAGO...

  testo cifrato: ...

  Ne consegue un aumento frequenze dei corrispondenti simboli.

- Nomenclatore: in aggiunta all'alfabeto cifrante si usa un insieme di parole in codice. Gli svantaggi che derivano sono:
  - Compilazione e trasporto del repertorio
  - Se cade in mani ostili, ripetizione della distribuzione

2.1.1 Cifrario Affine

I cifrari affine sono un caso particolare di cifrario a sostituzione. Per trovare la sostituzione si utilizza un'equazione, detta affine: $c_i = E(p_i) = (k_1 p_1 + k_2) \pmod{26}$. La chiave è data da tue costanti: $k_1, k_2 \in Z_{26}$. La decrittazione invece seguirà questa formula: $p_i = D(c_i) = (c_i - k_2) * k_1^{-1} \pmod{26}$, dove il k_1^{-1} sta a significare che devo prendere l'inverso in modulo 26 del valore k_1. L'inverso di un numero x in un'aritmetica modulare modulo N è quel numero y per il quale risulta $xy = 1 \pmod{N}$. Ad esempio, se $k_1 = 7$ e $k_2 = 3$ la funzione di cifratura è: $c = E(p) = (7p + 3) \pmod{26}$. Mentre, la funzione di decifratura è: $p = D(c) = 7^{-1}(c - 3) \pmod{26} = 15(c - 3) \pmod{26}$. Quindi, prendendo come esempio di testo in chiaro la parola "hot", la cui rappresentazione numerica è 7 14 19. Essa viene cifrata, applicando c, in 0 23 6, la cui rappresentazione in stringa è: "axg". Si noti che affinché la decifratura sia possibile deve esistere k^{-1}, cioè che l'equazione $k_1 x = c - k_2$ sia risolvibile. É possibile dimostrare che ciò è vero se e solo se: $gcd(k_1, 26) = 1$. In generale l'equazione $ax = b \pmod{m}$ ha una unica soluzione x se e solo se $gcd(a, m) = 1$, mentre k_2 può prendere qualsiasi valore tra 0 e 25. Il cifrario affine è comunque monoalfabetico.

Infatti, i cifrari a shift e a sostituzione sono detti monoalfabetici: una volta scelta la chiave ogni carattere è mappato ad un unico carattere cifrato. Per complicare le cose è possibile utilizzare due approcci:

- Utilizzo di più alfabeti cifranti
- Utilizzo di cifrature di più lettere per volta (Sillabe, Digrammi).

2.1.2 Cifrario Alberti

Nel 1404, Leon Battista Alberti, un pittore, musicista e filosofo del tempo, propose di utilizzare più alfabeti cifranti e di sostituirli durante la cifratura. Alberti inventò un metodo per generare messaggi criptati con l'aiuto di un apparecchio, il disco cifrante, il primo sistema di cifratura polialfabetica. L'apparecchio si compone di due dischi concentrici, rotanti uno rispetto all'altro, contenenti un alfabeto ordinato per il testo in chiaro (testo da cifrare) e un alfabeto disordinato per il testo cifrato (testo risultante). Permette la sostituzione polialfabetica con periodo irregolare. Lo scorrimento degli alfabeti avviene per mezzo di lettere chiave inserite nel corpo del crittogramma. Usa come indice una lettera scelta nel cerchio interno (mobile), nel nostro

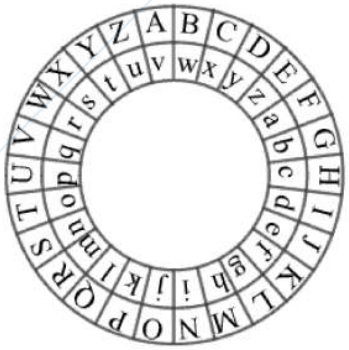


Figura 7: Disco di Alberti

esempio la v. Stabilita la v come lettera indice e avendola giustapposta alla A del cerchio esterno (stabile o fisso), lo sviluppo dei due alfabeti è il seguente (vedi Figura 7).

Supponendo di voler cifrare la parola DISCO, otteremo YCLUF, ruotando in senso orario il disco dopo ogni lettera cifrata.

2.1.3 Cifrario Porta

Giovanni Gattista Porta, invece, di cifrare le lettere cifra i digrammi, cioè gruppi di due lettere per volta. Si tratta del primo cifrario per digrammi (1563). Ogni digramma è poi assegnato ad un numero, e la chiave consiste nella tabella, rappresentata nella Figura 8. Ad esempio, se abbiamo "DOMANI" come testo in chiaro, otterremo il seguente testo cifrato: 92 312 346. La chiave è una permutazione arbitraria di: numeri del cifrato e lettere su righe e colonne. É possibile usare caratteri speciali, anziché i numeri, per alfabeto testo cifrato. La chiave rimane sempre la tabella, composta da 26 x 26 = 676 simboli.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
B	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
C	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77
D	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103
E	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129
F	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155
G	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181
H	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
I	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233
J	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259
K	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285
L	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311
M	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337
N	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363
O	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389
P	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415
Q	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441
R	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467
S	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493
T	494	495	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519
U	520	521	522	523	524	525	526	527	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544	545
V	546	547	548	549	550	551	552	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567	568	569	570	571
W	572	573	574	575	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591	592	593	594	595	596	597
X	598	599	600	601	602	603	604	605	606	607	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623
Y	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639	640	641	642	643	644	645	646	647	648	649
Z	650	651	652	653	654	655	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671	672	673	674	675

Figura 8: Cifrario di Porta

2.1.4 Cifrario Playfair

Il cifrario Playfair o quadrato di Playfair è una tecnica di cifratura simmetrica manuale basata su un cifrario monoalfabetico a due lettere. Lo schema fu inventato nel 1854 dal fisico inglese Sir Charles Wheatstone ma prende il nome del suo amico Lord Playfair Barone di St. Andrews, che cercò di divulgarne l'uso. La tecnica cifra coppie di lettere (digrafi). L'analisi delle frequenze può essere ancora intrapreso, ma sono possibili 600 digrafi, invece di 26 monografi. L'analisi della frequenza dei digrafi è possibile, ma considerevolmente più difficile. Inoltre le frequenze relative delle singole lettere hanno un intervallo molto più ampio di quello dei digrafi, rendendo l'analisi delle frequenze ulteriormente complicata. Il cifrario Playfair si basa sull'uso di una matrice 5×5 contenente una parola chiave. La memorizzazione della chiave e 4 semplici regole

sono tutto ciò che è richiesto per creare la tabella 5 per 5 e usare il codice. La tabella è costruita introducendo le lettere della parola chiave (eliminando le lettere duplicate), e poi riempiendo gli spazi rimanenti con le lettere non utilizzate dell'alfabeto, in ordine. Essendo 26 le lettere dell'alfabeto inglese e 25 gli spazi nella matrice, occorre escludere una lettera: generalmente viene esclusa la "Q", ma alcune versioni mettono la "I" e la "J" nello stesso spazio mentre altre escludono la "W", che se necessario può essere cifrata con una doppia "V". La chiave può essere scritta a partire dalla prima riga della tabella, da sinistra a destra, o con un altro percorso, per esempio a spirale iniziando dall'angolo in alto a sinistra e finendo nel centro. La parola chiave insieme alla convenzione per riempire la tabella 5 per 5 formano la chiave di cifratura. Per cifrare un messaggio, si deve dividere il messaggio in digrafi (gruppi di 2 lettere) in modo che, per esempio "Domani" diventi "DO MA NI". Le regole da applicare per ogni coppia di lettere del testo in chiaro sono 4:

- Se entrambe le lettere sono le stesse nel digrafo (o se la lettera è da sola), si aggiunga un "X" dopo la prima lettera. Cifrare la nuova coppia di lettere e continuare. Alcune varianti usano la "Q" al posto della "X", ma ogni lettera poco comune andrebbe bene. Ad esempio, Balloon diventa Balxloon (BA LX LO ON)
- Se le lettere appaiono nella stessa riga della tabella, vengono codificate con le lettere alla propria destra (considerando la tabella ciclica).
- Se le lettere appaiono nella stessa colonna della tabella, vengono codificate con le lettere immediatamente sotto (considerando la tabella ciclica).
- Se le lettere non sono nella stessa riga o colonna, permettono di individuare un rettangolo nella tabella che ha per vertici opposti le due lettere. A questo punto, le lettere si codificano con le lettere nelle stesse righe rispettivamente ma negli angoli opposti del rettangolo definito dalla coppia originale. L'ordine è importante, la prima lettera della coppia codificata è quella che appartiene alla stessa riga della prima lettera nel messaggio in chiaro.

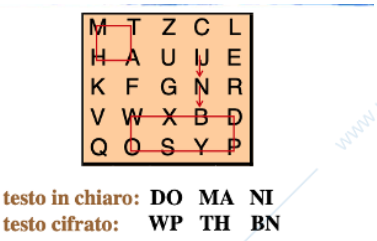


Figura 9: Cifrario di Playfair

Un esempio di cifratura è dato dalla Figura 9. Per decifrare, si usano le inverse di queste quattro regole e eliminando ogni "X" (o "Q") non necessaria nel messaggio finale. Come dicevamo precedentemente, il cifrario di Playfair ha una sicurezza maggiore della cifratura mono alfabetica (676 digrammi), ma la struttura del testo rimane. L'analisi viene condotta in base alla frequenze dei digrammi più comuni nella lingua (ad esempio: es, er, on, re, el, er, de).

2.1.5 Cifrario di Hill

Nella crittografia classica, il Cifrario di Hill è un cifrario a sostituzione polialfabetica basato sull'algebra lineare. Ideato da Lester S. Hill nel 1929, è stato il primo cifrario polialfabetico in cui era possibile nella pratica (anche se con difficoltà) operare con più di 3 simboli alla volta. Nella cifratura, ogni lettera è per prima cosa codificata in numero. Lo schema usato più di

frequente è semplicemente: $A = 0, B = 1, ..., Z = 25$, ma questa non è una caratteristica essenziale del cifrario. Un blocco n di lettere è quindi considerato come uno spazio vettoriale di dimensione n, e moltiplicato per una matrice nxn, modulo 26. Quindi, m lettere in chiaro sono sostituite con m lettere cifrate secondo m equazioni lineari. L'intera matrice è considerata la chiave del cifrario e deve essere casuale, a patto che sia invertibile in $\mathbb{Z}_{26}^n$ (per assicurare che la decrittazione sia possibile). Considerando il messaggio 'TUO' e la seguente chiave (che in lettere sarebbe Y Y P R Z W I Z J): $\begin{pmatrix} 24 & 24 & 15 \\ 17 & 25 & 22 \\ 08 & 25 & 09 \end{pmatrix}$

Considerando che 'T' è '19', 'U' è '20' e 'O' è '14', il messaggio è il vettore:

$\begin{pmatrix} 19 \\ 20 \\ 14 \end{pmatrix}$

Conseguentemente il vettore cifrato è dato da:

$$\begin{pmatrix} 24 & 24 & 15 \\ 17 & 25 & 22 \\ 08 & 25 & 09 \end{pmatrix} \begin{pmatrix} 19 \\ 20 \\ 14 \end{pmatrix} = \begin{pmatrix} 1146 \\ 1131 \\ 778 \end{pmatrix} \pmod{26} \equiv \begin{pmatrix} 02 \\ 13 \\ 24 \end{pmatrix}$$

che corrisponde al testo crittato 'CNY'. Il calcolo è dato dalla moltiplicazione matrice $\times$ testo in chiaro. Dopo aver effettuato questo calcolo, al risultato si toglie il modulo (in questo caso 26) tante volte quante ne servono per avere come differenza un numero inferiore del modulo stesso e che dia, quindi, in risultato una lettera ben definita.

$C \leftarrow 02 = 24 \cdot 19 + 24 \cdot 20 + 15 \cdot 14 \pmod{26}$

$N \leftarrow 13 = 17 \cdot 19 + 25 \cdot 20 + 22 \cdot 14 \pmod{26}$

$Y \leftarrow 24 = 08 \cdot 19 + 25 \cdot 20 + 09 \cdot 14 \pmod{26}$

Per decifrare riscriviamo il testo codificato come vettore, che poi moltiplicheremo semplicemente per la matrice inversa della matrice chiave (IFKVIVVMI in lettere). Ci sono metodi standard per calcolare la matrice inversa; si veda matrice invertibile per i dettagli. Si trova che la matrice inversa in

$\mathbb{Z}_{26}^n$ di quella dell'esempio precedente è:

$\begin{pmatrix} 13 & 03 & 23 \\ 23 & 18 & 13 \\ 17 & 08 & 10 \end{pmatrix}$

E quindi:

$$\begin{pmatrix} 13 & 03 & 23 \\ 23 & 18 & 13 \\ 17 & 08 & 10 \end{pmatrix} \begin{pmatrix} 02 \\ 13 \\ 24 \end{pmatrix} = \begin{pmatrix} 617 \\ 592 \\ 378 \end{pmatrix} = \begin{pmatrix} 19 \\ 20 \\ 14 \end{pmatrix} =$$

che ci fa tornare a 'TUO', come si voleva.

C'è ancora da discutere una complicazione presente nella scelta della matrice di codifica: non tutte le matrici sono invertibili. Una matrice ha un inverso se e solo se il suo determinante non è zero e non ha fattori in comune con il modulo base. Perciò, se si lavora modulo 26 come sopra, il determinante non deve essere zero e non deve essere divisibile per 2 o 13. Se il determinante è zero, o ha fattori in comune con il modulo base, allora la matrice non può essere usata nel cifrario di Hill e deve essere scelta un'altra matrice (altrimenti non sarebbe possibile decrittare). Fortunatamente, matrici che soddisfano le condizioni per essere usate nel cifrario di Hill sono piuttosto comuni.

Riferendosi all'esempio fatto, il determinante è:

$$\begin{vmatrix} 06 & 24 & 01 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{vmatrix} \equiv 6(16 \cdot 15 - 10 \cdot 17) - 24(13 \cdot 15 - 10 \cdot 20) + 1(13 \cdot 17 - 16 \cdot 20) \equiv 441 \equiv 25 \pmod{26}$$

Così, modulo 26, il determinante è 25. Poiché questo non ha fattori comuni con 26, la matrice può essere usata nel cifrario di Hill. Il rischio del determinante di avere fattori comuni con i moduli può essere eliminato utilizzando come modulo un numero primo. Di conseguenza

un'utile variante del cifrario aggiunge tre simboli ulteriori (ad esempio spazio, punto e punto di domanda) per portare il modulo a 29.

2.1.6 Cifrario di Vigenère

Il cifrario di Vigenère è il più semplice dei cifrari polialfabetici. Si basa sull'uso di un versetto per controllare l'alternanza degli alfabeti di sostituzione, concetto introdotto per la prima volta da Giovan Battista Bellaso ne La cifra del Sig. Giovan Battista Belaso del 1553.

Il metodo si può considerare una generalizzazione del cifrario di Cesare; invece di spostare sempre dello stesso numero di posti la lettera da cifrare, questa viene spostata di un numero di posti variabile ma ripetuto, determinato in base ad una parola chiave, da concordarsi tra mittente e destinatario, e da scrivere ripetutamente sotto il messaggio, carattere per carattere; la chiave era detta anche verme, per il motivo che, essendo in genere molto più corta del messaggio, deve essere ripetuta molte volte sotto questo, come nel seguente esempio (Figura 10.

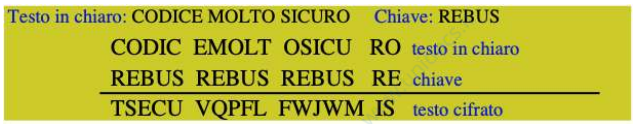


Figura 10: Cifrario di Vigenère

Il testo cifrato si ottiene spostando la lettera chiara di un numero fisso di caratteri, pari al numero ordinale della lettera corrispondente del verme. Di fatto si esegue una somma aritmetica tra l'ordinale del chiaro (A = 0, B = 1, C = 2...) e quello del verme; se si supera l'ultima lettera, la Z, si ricomincia dalla A, secondo la logica delle aritmetiche finite.

Il vantaggio rispetto ai cifrari monoalfabetici (come il cifrario di Cesare o quelli per sostituzione delle lettere con simboli/altre lettere) è evidente: il testo è cifrato con n alfabeti cifranti. In questo modo, la stessa lettera viene cifrata (se ripetuta consecutivamente) n volte; ciò rende quindi più complessa la crittoanalisi del testo cifrato.

Per semplificare la cifratura, Vigenère propose l'uso della seguente "matrice" quadrata, composta da alfabeti ordinati e spostati. Se si vuole cifrare, con la chiave dell'esempio precedente (10), la lettera "C" della parola codice basterà trovare la lettera "C" nella prima riga, individuando la colonna relativa. Basterà poi trovare la "R" di "rebus" nella prima colonna per trovare la riga, individuando, tramite l'incrocio, la lettera corretta da usare.

Figura 11: Quadrato di Vigenère

Come si è detto questo cifrario ha goduto per tre secoli la fama di essere un cifrario inattaccabile; nel 1863 il colonnello prussiano Friedrich Kasiski pubblicò un primo metodo di decrittazione. Il numero possibile di chiavi è 26^t, dove t rappresenta la lunghezza della chiave. É possibile rompere questo algoritmo di crittazione, utilizzando indice di coincidenza ed indice mutuo di coincidenza.

Il cifrario di Vigenère resiste all'analisi delle frequenze, dal momento che una lettera cifrata corrisponde a più simboli in chiaro ed esiste un numero grande di possibili chiavi. La crittoanalisi si basa sulla determinazione delle ripetizioni. A partire dalle ripetizioni si stima la lunghezza della chiave. Si analizzano le frequenze delle lettere in ognuno degli alfabeti cifranti corrispondenti alle lettere della chiave.

2.1.7 Cifrari a trasposizione

Come dicevamo precedentemente, le operazioni di trasformazione utilizzabili per crittare i dati sono:

- Sostituzione: ciascun elemento del testo viene mappato su un altro elemento
- Trasposizione: elementi del testo in chiaro vengono scambiati di posto

Vediamo ora alcuni esempi di cifrari a trasposizione

La scitala o scitale è considerato tradizionalmente un messaggio cifrato e segreto che veniva inviato dagli efori, i cinque supremi magistrati di Sparta, ai generali e ai navarchi impegnati nelle spedizioni militari. Si tratta di uno dei più antichi metodi di crittografia per trasposizione conosciuti: il meccanismo di codifica permetteva, nel caso la scitala fosse stata intercettata dal nemico, di mantenere segreto il contenuto del messaggio e, nello stesso tempo, consentiva al ricevente di verificarne l'autenticità, in quanto solo chi era dotato di una bacchetta identica a quella utilizzata dal mittente per preparare la scitala, poteva decifrare e leggere il messaggio. Plutarco descrive accuratamente il funzionamento della scitala nella Vita di Lisandro, dove precisa che con questo termine si intendeva sia la pergamena col messaggio che la bacchetta che veniva utilizzata per la sua scrittura e decifrazione. Prima di scrivere il messaggio, gli efori preparavano una striscia di pergamena lunga e stretta e la avvolgevano a spirale attorno ad una bacchetta, che era esattamente uguale, come lunghezza e diametro, ad un'altra bacchetta che i magistrati avevano precedentemente fornito al destinatario. Dopo aver fatto aderire la pergamena alla bacchetta, facendo attenzione a non lasciare nessuno spazio nel quale il legno fosse visibile e nello stesso tempo evitando di sovrapporre diversi lembi della pergamena stessa, gli efori provvedevano a scrivere il messaggio. Quindi, gli efori srotolavano la striscia dalla bacchetta e la inviavano al loro emissario tramite un messaggero. Se questi veniva intercettato durante il suo viaggio, il messaggio sarebbe stato incomprensibile in quanto composto da lettere non collegabili tra loro. Solo il destinatario, invece, avendo a disposizione una bacchetta identica a quella in mano agli efori, poteva riavvolgere la pergamena attorno ad essa e ricostruire la posizione originaria delle lettere e capire il contenuto del messaggio.

Il cifrario a staccionata (noto anche come cifrario a zigzag) è un tipo di cifrario a trasposizione che deve il suo nome al modo in cui il testo in chiaro viene cifrato: esso viene trascritto lettera per lettera su righe ideali, diagonalmente verso il basso e poi risalendo una volta arrivati alla riga più bassa e viceversa arrivati alla riga più in alto, disegnando ipotetiche traverse di un'immaginaria staccionata oppure di uno zigzag. Il messaggio cifrato si ottiene, alla fine, leggendo le lettere così posizionate riga per riga. Ad esempio, utilizzando 3 righe ed il messaggio "PIANTARE IL CAMPO DIETRO LA COLLINA", la trasposizione sarà:

```
P . . . T . . . I . . . M . . . I . . . O
. . . O . . . N .
. I . N . A . E . L . A . P . D . E . R .
L . C . L . I . A
. . A . . . R . . . C . . . O . . . T . .
. A . . . L . . .
```

Figura 12: Rail Fence

Che sarà cifrato come:
PTIMI OONIN AELAP DERLC LIAAR COTAL
Il cifratore ha diviso il testo cifrato in blocchi di 5 lettere così da ridurre gli errori di trascrizione.I cifrari a staccionata non sono molto robusti: il numero di parole chiavi utilizzabili è abbastanza piccolo tanto che un crittanalista le può provare tutte praticamente a mano.

Nella trasposizione colonnare (Cifrario a colonne) il messaggio è scritto lungo le righe di una griglia di dimensioni prefissate e poi letto lungo le colonne, secondo un ordine particolare delle stesse. Sia la lunghezza delle righe che la permutazione delle colonne sono definite da una parola chiave. Ad esempio, la parola VETRINA è lunga 7 lettere, così le righe saranno anch'esse di lunghezza 7, e la permutazione è definita dall'ordine alfabetico delle lettere della parola chiave. In questo caso l'ordine sarebbe "7-2-6-5-3-4-1" perché l'ordine alfabetico delle lettere di VETRINA è A-E-I-N-R-T-V, e le loro posizioni sono quindi A=1, E=2, I=3, N=4, R=5, T=6 e V=7, per cui V-E-T-R-I-N-A corrisponde proprio a 7-2-6-5-3-4-1. Nei cifrari a trasposizione colonnare regolari le posizioni vuote alla fine dell'ultima riga vengono riempite con caratteri casuali, mentre in quelli irregolari sono lasciati bianchi. Alla fine, il messaggio è letto sulle colonne secondo l'ordine specificato dalla parola chiave. Ad esempio, supponendo di usare la parola chiave VETRINA e il messaggio PIANTARE IL CAMPO DIETRO LA COLLINA, in una trasposizione colonnare regolare avremmo una griglia così composta:

V	E	T	R	I	N	A
7	2	6	5	3	4	1
P	I	A	N	T	A	R
E	I	L	C	A	M	P
O	D	I	E	T	R	O
L	A	C	O	L	L	I
N	A	D	V	Y	I	Q

Figura 13: Cifrario a colonne

Con le ultime cinque lettere insignificanti (DVYIQ). Il testo cifrato è così letto: RPOIQ IIDAA TATLY AMRLI NCEOV ALICD PEOLN. Per decifrarlo, il destinatario risale alla lunghezza delle colonne dividendo la lunghezza del messaggio per quella della parola chiave. Poi può scrivere il messaggio riorganizzandolo in colonne e poi riordinare le colonne in base alla parola chiave.

In crittografia un cifrario a trasposizione è un metodo di cifratura in cui le posizioni occupate dalle unità di testo in chiaro (che in genere sono lettere o gruppi di esse) sono cambiate secondo un determinato schema, così che il testo cifrato costituisca una permutazione del testo in chiaro. Matematicamente parlando viene utilizzata una funzione di corrispondenza biunivoca sulle posizioni dei caratteri durante l'operazione di codifica e una funzione inversa durante quella di decodifica. Riassumendo, è basato sulle permutazioni. Divido il mio testo in chiaro in blocchi di n caratteri. Poi, stabilisco una funzione biettiva per cui ognuno degli n caratteri finisce in una e una sola posizione tra gli n, e viceversa. Se la parola in chiaro è "gatto", e stabilisco che:

$1 \to 2$

$2 \to 1$

$3 \to 5$

$4 \to 4$

$5 \to 3$

allora avrei che la parola crittata è AGOTT.

La sicurezza, per quanto riguarda le tecniche di trasposizione, è poca quando si tratta di messaggi brevi (ad esempio, con un messaggio di 3 lettere si possonoa avere 6 anagrammi diversi).

2.1.8 Crittogrammi Beale

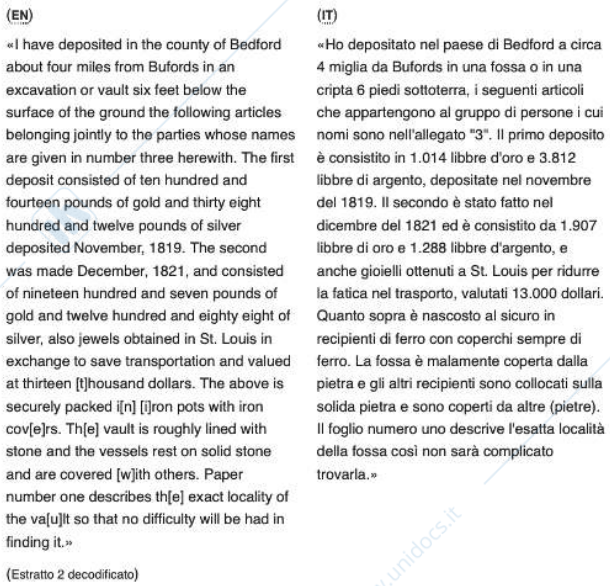
Il cifrario Beale viene considerato come uno dei grandi enigmi crittografici ancora irrisolti. Si compone di una serie di tre messaggi lasciati nel 1822 da Thomas J. Beale ad un amico, con l'impegno di leggerli solo se non fosse tornato, che condurrebbero ad un favoloso tesoro sepolto nella Contea di Bedford, in Virginia (Stati Uniti d'America). Il primo messaggio indicherebbe il luogo del tesoro, il secondo (Figura 14) la descrizione ed il terzo i nomi dei compagni di Beale. Morris tentò per 20 anni di decifrare i crittogrammi, senza successo. Nel 1862 mostrò i crittogrammi ad un amico che, dopo aver decifrato il secondo, pubblicò un opuscolo nel 1885.

115, 73, 24, 807, 37, 52, 49, 17, 31, 62, 647, 22, 7, 15, 140, 47, 29, 107, 79, 84, 56,
239, 10, 26, 811, 5, 196, 308, 85, 52, 160, 136, 59, 211, 36, 9, 46, 316, 554, 122,
106, 95, 53, 58, 2, 42, 7, 35, 122, 53, 31, 82, 77, 250, 196, 56, 96, 118, 71, 140,
287, 28, 353, 37, 1005, 65, 147, 807, 24, 3, 8, 12, 47, 43, 59, 807, 45, 316, 101, 41,
78, 154, 1005, 122, 138, 191, 16, 77, 49, 102, 57, 72, 34, 73, 85, 35, 371, 59, 196,
81, 92, 191, 106, 273, 60, 394, 620, 270, 220, 106, 388, 287, 63, 3, 191, 122, 43,
234, 400, 106, 290, 314, 47, 48, 81, 96, 26, 115, 92, 158, 191, 110, 77, 85, 197, 46,
10, 113, 140, 353, 48, 120, 106, 2, 607, 61, 420, 811, 29, 125, 14, 20, 37, 105, 28,
248, 16, 159, 7, 35, 19, 301, 125, 110, 486, 287, 98, 117, 511, 62, 51, 220, 37, 113,
140, 807, 138, 540, 8, 44, 287, 388, 117, 18, 79, 344, 34, 20, 59, 511, 548, 107,
603, 220, 7, 66, 154, 41, 20, 50, 6, 575, 122, 154, 248, 110, 61, 52, 33, 30, 5, 38, 8,
14, 84, 57, 540, 217, 115, 71, 29, 84, 63, 43, 131, 29, 138, 47, 73, 239, 540, 52, 53,
79, 118, 51, 44, 63, 196, 12, 239, 112, 3, 49, 79, 353, 105, 56, 371, 557, 211, 515,
125, 360, 133, 143, 101, 15, 284, 540, 252, 14, 205, 140, 344, 26, 811, 138, 115,
48, 73, 34, 205, 316, 607, 63, 220, 7, 52, 150, 44, 52, 16, 40, 37, 158, 807, 37, 121,
12, 95, 10, 15, 35, 12, 131, 62, 115, 102, 807, 49, 53, 135, 138, 30, 31, 62, 67, 41,
85, 63, 10, 106, 807, 138, 8, 113, 20, 32, 33, 37, 353, 287, 140, 47, 85, 50, 37, 49,
47, 64, 6, 7, 71, 33, 4, 43, 47, 63, 1, 27, 600, 208, 230, 15, 191, 246, 85, 94, 511, 2
270, 20, 39, 7, 33, 44, 22, 40, 7, 10, 3, 811, 106, 44, 486, 230, 353, 211, 200, 31,
10, 38, 140, 297, 61, 603, 320, 302, 666, 287, 2, 44, 33, 32, 511, 548, 10, 6, 250,
557, 246, 53, 37, 52, 83, 47, 320, 38, 33, 807, 7, 44, 30, 31, 250, 10, 15, 35, 106,
160, 113, 31, 102, 406, 230, 540, 320, 29, 66, 33, 101, 807, 138, 301, 316, 353,
320, 220, 37, 52, 28, 540, 320, 33, 8, 48, 107, 50, 811, 7, 2, 113, 73, 16, 125, 11,
110, 67, 102, 807, 33, 59, 81, 158, 38, 43, 581, 138, 19, 85, 400, 38, 43, 77, 14, 27,
8, 47, 138, 63, 140, 44, 35, 22, 177, 106, 250, 314, 217, 2, 10, 7, 1005, 4, 20, 25,
44, 48, 7, 26, 46, 110, 230, 807, 191, 34, 112, 147, 44, 110, 121, 125, 96, 41, 51,
50, 140, 56, 47, 152, 540, 63, 807, 28, 42, 250, 138, 582, 98, 643, 32, 107, 140,
112, 26, 85, 138, 540, 53, 20, 125, 371, 38, 36, 10, 52, 118, 136, 102, 420, 150,
112, 71, 14, 20, 7, 24, 18, 12, 807, 37, 67, 110, 62, 33, 21, 95, 220, 511, 102, 811,
30, 83, 84, 305, 620, 15, 2, 108, 220, 106, 353, 105, 106, 60, 275, 72, 8, 50, 205,
185, 112, 125, 540, 65, 106, 807, 188, 96, 110, 16, 73, 33, 807, 150, 409, 400, 50,
154, 285, 96, 106, 316, 270, 205, 101, 811, 400, 8, 44, 37, 52, 40, 241, 34, 205,
38, 16, 46, 47, 85, 24, 44, 15, 64, 73, 138, 807, 85, 78, 110, 33, 420, 505, 53, 37,
38, 22, 31, 10, 110, 106, 101, 140, 15, 38, 3, 5, 44, 7, 98, 287, 135, 150, 96, 33, 84,
125, 807, 191, 96, 511, 118, 440, 370, 643, 466, 106, 41, 107, 603, 220, 275, 30,
150, 105, 49, 53, 287, 250, 208, 134, 7, 53, 12, 47, 85, 63, 138, 110, 21, 112, 140,
485, 486, 505, 14, 73, 84, 575, 1005, 150, 200, 16, 42, 5, 4, 25, 42, 8, 16, 811,
125, 160, 32, 205, 603, 807, 81, 96, 405, 41, 600, 136, 14, 20, 28, 26, 353, 302,
246, 8, 131, 160, 140, 84, 440, 42, 16, 811, 40, 67, 101, 102, 194, 138, 205, 51,
63, 241, 540, 122, 8, 10, 63, 140, 47, 48, 140, 288.

Figura 14: Secondo crittogramma Beale

I messaggi sono composti da una sequenza di numeri e si è compreso che la chiave di lettura è posta in tre libri. I numeri in questione indicano pagine e posizione delle lettere da trovare, una volta unite tutte le singole lettere si ha il messaggio completo. Finora è stato interpretato solo il secondo, a partire dalla Dichiarazione di indipendenza degli Stati Uniti (Figura 15).

Il primo e il terzo crittogramma di Beale sono inviolati da più di un secolo. I crittogrammi potrebbero essere stati alterati dall'autore dell'opuscolo per impedirne la decifratura. E il tesoro? Qualcuno potrebbe averlo trovato utilizzando gli indizi dell'unico crittogramma decifrato. L'intera storia potrebbe essere una montatura, ispirata dal romanzo di Edgar Allan Poe. Ancora oggi crittoanalisti e cacciatori di tesori sono affascinati dalla vicenda.



(EN)

«I have deposited in the county of Bedford about four miles from Bufords in an excavation or vault six feet below the surface of the ground the following articles belonging jointly to the parties whose names are given in number three herewith. The first deposit consisted of ten hundred and fourteen pounds of gold and thirty eight hundred and twelve pounds of silver deposited November, 1819. The second was made December, 1821, and consisted of nineteen hundred and seven pounds of gold and twelve hundred and eighty eight of silver, also jewels obtained in St. Louis in exchange to save transportation and valued at thirteen [t]housand dollars. The above is securely packed i[n] [i]ron pots with iron cov[e]rs. Th[e] vault is roughly lined with stone and the vessels rest on solid stone and are covered [w]ith others. Paper number one describes th[e] exact locality of the va[u]lt so that no difficulty will be had in finding it.»

(IT)

«Ho depositato nel paese di Bedford a circa 4 miglia da Bufords in una fossa o in una cripta 6 piedi sottoterra, i seguenti articoli che appartengono al gruppo di persone i cui nomi sono nell'allegato "3". Il primo deposito è consistito in 1.014 libbre d'oro e 3.812 libbre di argento, depositate nel novembre del 1819. Il secondo è stato fatto nel dicembre del 1821 ed è consistito da 1.907 libbre di oro e 1.288 libbre d'argento, e anche gioielli ottenuti a St. Louis per ridurre la fatica nel trasporto, valutati 13.000 dollari. Quanto sopra è nascosto al sicuro in recipienti di ferro con coperchi sempre di ferro. La fossa è malamente coperta dalla pietra e gli altri recipienti sono collocati sulla solida pietra e sono coperti da altre (pietre). Il foglio numero uno descrive l'esatta località della fossa così non sarà complicato trovarla.»

(Estratto 2 decodificato)

Figura 15: Secondo crittogramma Beale risolto

2.2 Macchine cifranti

Una delle prime descrizione dei cilindri cifranti risale al 1605 grazie a Francis Bacon. Successivamente, alla fine del XVIII secolo, Thomas Jefferson inventò una macchina cifrante, detta cilindro di Jefferson che è rimasta in uso fino alla seconda guerra mondiale. Essa consiste in un cilindro montato su un asse e diviso in 36 dischi uguali che possono ruotare liberamente attorno all'asse. Sul bordo esterno di ciascun disco sono scritte le 26 lettere dell'alfabeto, equispaziate l'una dall'altra. L'ordine con cui sono scritte le lettere non è quello naturale e varia da ruota a ruota. Il numero possibile di ordinamenti dei dischi equivale a 36!, ovvero a $3,72 \cdot 10^{41}$.

A partire dal 1919, si andava affermando una classe di macchine cifranti basate su principi meccanici o elettromeccanici, le cosiddette macchine a rotori, che rimasero per anni le dominatrici incontrastate della scena crittografica. Il funzionamento di una macchina a rotori è concettualmente molto semplice: esso si basa su una catena di dischi mobili (i rotori, appunto) affacciati l'uno sull'altro, ciascuno dei quali costruito di materiale isolante e dotato su entrambe le facce di 26 contatti elettrici. Ogni contatto di una faccia corrisponde ad uno ed un solo contatto sull'altra faccia, ma la corrispondenza non è diretta, bensı "incrociata" in qualche modo. Ad ogni contatto o combinazione di contatti, corrisponde un carattere dell'alfabeto utilizzato per i messaggi. La decifratura è semplice perché basta inserire il messaggio cifrato, carattere per carattere, dal primo rotore, per riottenere il testo in chiaro. Supponiamo che i dischi siano tre in successione. Applicando una tensione al contatto 1 del primo disco, questi la trasferisce ad un differente contatto del secondo disco per via del suo collegamento elettrico interno, il secondo fa lo stesso con il terzo cosı che alla fine il contatto alimentato in uscita corrisponde ad una lettera differente da quella applicata all'ingresso. Ora i tre dischi possono ruotare a passi discreti attorno al proprio asse e sono collegati tra loro da un meccanismo, l'odometro, grazie al quale, non appena viene creata la codifica di una lettera del testo in chiaro, l'ultimo disco ruota di $\frac{1}{26}$ di giro; ció ha l'effetto di modificare la struttura cifrante della macchina e quindi, in pratica, le permette di applicare al prossimo carattere una trasformazione differente da quella applicata al carattere precedente. Quando l'ultimo disco ha compiuto un'intera rotazione, quello alla sua sinistra avanza di un passo ed il ciclo riprende, ma con una struttura che non si ripete mai come prima. In pratica dunque, la macchina modifica il suo cifrario ad ogni carattere e non riapplica la stessa trasformazione se non dopo un numero straordinariamente alto di caratteri cifrati.

Figura 16: Rotori

Quindi, il componente principale è un insieme di rotori, anche denominati ruote o tamburi, i quali ruotano i dischi con una serie di contatti elettrici su entrambi i lati. Il cablaggio tra i contatti implementa una sostituzione fissa di lettere, sostituendoli in qualche modo complesso. Dopo la crittografia di ogni lettera, i rotori avanzano posizioni, cambiando lo schema di sostituzione. In questo modo, una macchina rotore produce un complesso cifrario a sostituzione polialfabetica, che cambia con ogni pressione del tasto.

2.2.1 Macchina Enigma

Tra le varie macchine del genere, costruite tra le due guerre mondiali, la tedesca ENIGMA è quella che ha avuto un ruolo di primaria importanza nella cifratura e decifratura di messaggi militari nel corso della seconda guerra mondiale. Enigma fu un dispositivo elettromeccanico per cifrare e decifrare messaggi. Nata da un tentativo di commercializzazione poi fallito, fu ampiamente utilizzata dal servizio delle forze armate tedesche durante il periodo nazista e della seconda guerra mondiale. La macchina Enigma fu sviluppata da Arthur Scherbius in varie versioni a partire dal 1918 quando ottenne il brevetto, ispirandosi al disco cifrante di Leon Battista Alberti. La prima versione misurava appena $34 \times 28 \times 15$ cm ma aveva un peso vicino ai 12 kg. Come dicevamo, in linea di puro principio, Enigma può essere considerata come un'estensione del metodo del cifrario di Vigenère munita di disco di Leon Battista Alberti. La differenza principale sta nel fatto che i dischi cifranti sono più di uno, posti fra loro "in cascata", e che manca qui la chiave, detta anche verme, che invece era elemento essenziale nella cifratura di Vigenère.

La macchina Enigma applica ripetutamente sostituzioni e permutazione di tutte le lettere del messaggio. Le sostituzioni sono operate da connessioni elettriche tra coppie di contatti che corrispondono a caratteri. La macchina ha al suo interno un certo numero di rotori, il cui funzionamento è il seguente: il primo rotore scatta ogni volta che una lettera è stata cifrata, quando ha compiuto un giro completo scatta anche il secondo rotore e così via per gli altri rotori.

Enigma è una macchina simmetrica, nel che se la lettera A è cifrata con la I in una certa posizione del testo, allora nella stessa posizione la I sarà cifrata con la lettera A, questo significa che può essere utilizzata si in fase di cifratura che in quella di decifratura, una comodità pagata in termini di debolezza crittografica.

Quando viene premuto un pulsante, il segnale elettrico attraversa i rotrori fino a giungere ad un ultimo rotore chiamato riflettore, che fa tornare indietro il segnale fino ad illuminare una lettera che è il carattere cifrato, permettendo all'operatore di copiare a mano il testo cifrato.

I tedeschi erano convinti che Enigma fosse inattaccabile, tuttavia già dai prima anni '30 un gruppo di matematici polacchi era riuscito a decifrare i crittogrammi. Anche il gruppo di crittoanalisiti inglese, cui partecipava Alan Turing, erano in grado di forzare l'Enigma sin

dall'inizio della guerra sfruttando le sue debolezze. La macchina Enigma consiste, semplificando di tre componenti principali collegati elettricamente:

- una tastiera, per dare in input le lettere del testo in chiaro
- un rotore, che cifra ogni lettera trasformandola nel corrispondente carattere del crittogramma
- un visore con alcune lampadine, che accendendosi indicano la lettera da inserire nel testo cifrato.

Il rotore, o scambiatore, è il componente più importante e consiste in uno spesso disco di gomma attraversato da una fitta rete di fili provenienti dalla tastiera. Questi fili entrano nello scambiatore e dopo un percorso formato da vari gomiti escono dalla parte opposta. Lo schema interno del rotore permette al congegno di operare una semplice cifratura a sostituzione monoalfabetica. Quando l'operatore preme il tasto corrispondente ad una lettera da cifrare, l'impulso elettrico che passa nei fili elettrici attraversa il rotore e va ad illuminare il visore in modo da cacendere la lettera cifrata corrispondente. Il rotore, quindi, crea una corrispondenza tra lettere in chiaro e quelle da cifrare. Dopo aver cifrato ogni lettera, il disco scambiatore ruota di $\frac{1}{26}$ di giro, quindi ogni lettera viene sostituita diversamente ad ogni passo. In questo modo il rotore definisce 26 diversi alfabeti cifranti permettendo ad Enigma di effettuare una sostituzione polialfabetica. Dopo 26 pressioni sulla tastiera il rotore torna nella posizione iniziale, tuttavia introducendo un altro scambiatore, mentre il primo ruota il secondo rimane fermo fin quando il primo ha completato il giro. Il secondo rotore quindi avanza di una posizione grazie ad una dentatura. Quindi ognuno dei rotori si orienta in 26 modi diversi. In questo modo, si ritorna al punto di partenza dopo 26 giri del primo scambiatore, cioè dopo la cifratura di $26 \times 26 = 676$ lettere. Aggiungendo un terzo rotore, il numero di sostituzione diventa $26 \times 26 \times 26 = 17576$. Il periodo è tale da evitare lo studio delle frequenze: sarebbero necessarie almeno 50 lettere cifrate per ciascuna sostituzione, equivalenti ad un enorme mole di testo. La macchina enigma ha un altro componente chiamato riflessore, che consiste in un disco di gomma con circuiti elettrici simili a quelli del rotore. A differenza del rotore, il riflessore non ruota e i fili entrano ed escono dallo stesso lato. In questo modo, quando si da in input una lettera, il segnale elettrico passa attraverso i tre rotori, arriva al riflessore, torna indietro e passa di nuovo nei rotori, ma percorrendo un percorso diverso.

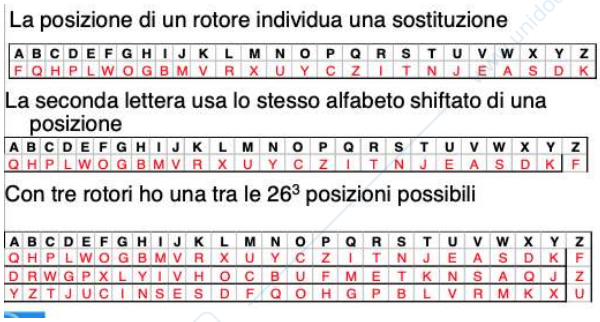


Figura 17: Sostituzioni di Enigma

Per aumentare il numero di chiavi, sono stati usati rotori removibili e sostituibili, questo significa che tre rotori potevano essere permutati in sei modi diversi. Inoltre, tra la tastiera e il primo rotore fu inserito il pannello a prese multiple, un componente che permetteva di scambiare due lettere prima della loro immissione nel rotore attraverso dei cavi con spinotto, in modo tale da far percorrere all'impulso di una lettera il percorso di un'altra. Inizialmente, vi erano a disposizione sei cavi che permettevano di scambiare sei coppie di lettere contemporaneamente.

Quindi, i tre rotori potevano combinarsi in $26 \times 26 \times 26 = 17576$ posizioni diverse e potevano essere messi in 6 posizioni reciproche diverse. Per quanto riguarda il pannello a prese multiple

il numero di combinazioni di $2 \times 6 = 12$ lettere su 26 sono $\frac{26!}{6!(26-12)!2^6} = 100.391.791.500$. Il numero totale di chiavi è $17576 \times 6 \times 100.391.791.500 = 1,05869e16$. Il mittente e il destinatario dovevano accordarsi sull'inizializzazione della macchina. La chiave, quindi, conteneva informazioni sull'assetto del pannello a prese multiple, sulla scelta e sull'ordine dei rotori e sul loro orientamento rispetto all'asse di rotazione. Inoltre, era necessario comunicare la scelta del riflettore. Enigma veniva quotidianamente settata con impostazioni giornaliere. Ogni mese gli operatori ricevevano un cifrario con scritte le chiavi giornaliere. Riassumendo, quindi, ogni giorno i settaggi della macchina Enigma riguardavano:

- la scelta dei rotori da utilizzare e del loro ordine (Walzenlage),
- la scelta della posizione dell'anello di ciascun rotore (Ringstellung),
- la scelta delle posizioni iniziali in cui porre i rotori (Grundstellung),
- infine la scelta delle coppie di lettere da scambiare tramite i cavi del pannello a prese multiple (Steckerverbin- dungen).

Il pannello fa una semplice sostituzione monoalfabetica, ma con un grande numero di chiavi. Le sostituzioni non cambiano durante la cifratura, quindi si rompe con l'analisi delle frequenze. I rotori hanno un piccolo numero di chiavi, ma l'assetto cambia continuamente, ciò gli permette di resistere all'analisi delle frequenze. Per rende il sistema di cifratura più sicuro, la chiave giornaliera veniva usata per trasmettere al destinatario una seconda chiave, detta "chiave di messaggio", che veniva usata per cifrare il testo in chiaro. La chiave di messaggio differiva dalla chiave giornaliera per l'orientamento dei rotori. Per evitare errori, la chiave di messaggio veniva trasmessa due volte consecutive. In questo modo la chiave giornaliera veniva usata per cifrare meno informazioni.

L'analisi crittografica è basata sulla natura dell'algoritmo e sfrutta la conoscenza su testo in chiaro e algoritmo. Con il metodo della forza bruta si tenta ogni possibile chiave fino ad ottenere un risultato, in media bisogna provare la metà delle chiavi.

Nel settembre 1932 l'ufficio cifra polacco chiamò tre matematici, guidati da Marian Rejewski, con il compito di cercare di forzare la macchina Enigma usata dai tedeschi. Come dicevamo precedentemente, la chiave di messaggio veniva ripetuta due volte all'inizio del messaggio stesso per "maggiore sicurezza". Questa idea di trasmettere due volte di seguito i tre caratteri segreti si rivelò essere un grosso appiglio per il crittanalista. Dal punto di vista matematico l'Enigma si riduce a un prodotto di permutazioni. Poiché il primo e il quarto carattere di ogni messaggio sono uguali (chiave di messaggio ripetuta) Rejewski potè ricostruire la permutazione tra il primo e il quarto carattere del messaggio nel giro di pochi giorni, avendo a disposizione ogni giorno molte decine di messaggi cifrati con Enigma. Indichiamo con ABCDEF i 6 caratteri iniziali di un messaggio. Partendo da un qualsiasi messaggio è possibile, possedendo sufficienti messaggi, creare un algoritmo che fornisca sotto forma di ciclo la codifica della prima lettera come composizione della permutazione che porta x in A e x in D. Per ottenere tale ciclo si prende un messaggio di partenza dato, successivamente si ricerca un messaggio con A uguale alla D del primo messaggio, è necessario iterare questo procedimento fino a quando non si ha un'uguaglianza fra la D del messaggio attuale e la A di quello di partenza. Ripetendo il procedimento con messaggi di partenza le cui A non appartengono ai cicli già determinati, otteniamo la permutazione A - D come prodotto di cicli. Si prendano in considerazioni le seguenti chiavi di messaggio:

- DMX VBJ
- VBJ PMK
- PUK FNY
- FNY IFW

- IFW XGX
- XGA GQC
- GQC ZDB
- ZDB YZD
- YZD OJF
- OJF DUA

É evidente dal primo messaggio che D vada in V e dal secondo che V venga mandato in P, ora si possiede un ciclo parziale dato da (DVP...). Per completare il ciclo si prosegue in modo del tutto analogo fino al messaggio 10, grazie al quale si chiude il ciclo ottenendo: (DBPFIXGZYO). Iterando l' algoritmo su sufficienti messaggi, qui non riportati, si ottiene un prodotto di cicli disgiunti che forniscono la permutazione A - D. A - D = (DBPFIXGZYO)(EIJMUNGLHT)(BC)(RW)(A)(S). Proseguendo in maniera del tutto analoga si ricavano:

B - E = (UNFGQDZJ)(AXYWV)(ELOT)(CKI)(MB)(R)(S)

C - F = (GILOQZTMERSUVNP)(XJKYW)(ACBDF)

Il numero di collegamenti nelle concatenazioni dipende solo dagli scambiatori, cioè solo da 6 (assetti) $\times 17576 = 105456$ combinazioni. Rejewski classificò l'orientamento iniziale tramite un repertorio delle combinazioni, individuata la chiave bisognava ricostruire l'assetto dei pannelli. Si procedette così alla creazione di "Bombe", ossia una macchina calcolatrice. Una volta ricostruita la struttura logica di Enigma usando la teoria delle permutazioni, Rejewski progettò un dispositivo elettromeccanico, il "ciclometro", che permetteva di ricostruire velocemente la posizione iniziale dei rotori; in seguito venne usata anche la tecnica dei "fogli perforati", ed infine venne costruita la "bomba crittologica", un rudimentale calcolatore basato sul metodo forza bruta. La "Bomba" era una macchina composta da molti moduli: ciascun modulo consisteva di uno scaffale di ferro largo 2,10 metri, alto 1,90, profondo 60 centimetri, e pesante circa una tonnellata. Ogni modulo metteva in movimento 108 rotori (più tre di controllo) in gruppi di 12 per fila, che eseguivano gradualmente la decodifica dei messaggi.

Le incredibili capacità di Rejewski e del suo team si scontrarono nel 1939 con due modifiche sostanziali della macchina. Furono introdotti 2 nuovi rotori da poter inserire nei 3 box predisposti, ciò aumentava notevolmente il numero di chiavi, in quanto dalle 6 possibili configurazioni dei rotori si passava a 60 e di conseguenza le possibili permutazioni dovute ai rotori a 1.054.560. Inoltre, furono introdotti altri 6 collegamenti nel pannello a prese multiple fornendo ai tedeschi 159 miliardi di miliardi di possibili chiavi. La complessità computazionale era divenuta troppo grande per i mezzi polacchi.

La macchina presentava ancora alcune debolezze:

- Nessuna lettera cifra se stessa
- Una lettera non cifra lettere contigue
- Utilizzo di cillies, chiavi semplici (qweqwe)
- Se L_1 cifra L_2 allora L_2 cifra L_1
- Invio della posizione iniziale rotori

Allo scoppio del conflitto il governo inglese decise di spostare la Stanza 40, ovvero il centro della crittoanalisi governativa, nel più ampio sito di Bletchley Park che poteva ospitare nettamente più personale. I crittoanalisti di Bletchley Park assimilarono il metodo polacco e cercarono di elaborare strategie per ridurre i tempi di decifrazione, che dopo le modifiche tedesche si erano dilatati. Il personale e i mezzi maggiori rispetto ai crittoanalisti polacchi permisero agli inglesi di possedere sufficiente materiale per osservare che una debolezza risiedeva negli operatori. Infatti nonostante le indicazioni fossero quelle di inserire chiavi casuali per ogni messaggio, molti addetti inserivano terne di lettere consecutive sulla tastiera come ASD o ZXC; queste chiavi banali venivano chiamate *cillies*. Questi non erano punti deboli del sistema crittografico in sé, ma di come gli addetti lo utilizzavano. Inoltre, i responsabili della scelta della chiave giornaliera decisero di non riposizionare per due giorni consecutivi un rotore nella medesima posizione. Ovvero nominando i 5 rotori con N, M, P, Q e L se un giorno l'assetto è N - M - L il giorno seguente sicuramente N non può essere in prima posizione, M non può essere in seconda e L non può essere in terza; quindi si passa da 60 possibili assetti a 31. Il pannello a prese multiple viene configurato con un ragionamento simile. Infatti sono vietati gli scambi fra lettere consecutive; ciò non fornisce nessun vantaggio alla resistenza del cifrario, ma al contrario esclude molteplici possibili assetti del pannello ai decifratori. Sfruttando i cillies e questi accorgimenti si riusciva a decifrare alcuni messaggi di Enigma, però la comunicazione usando la macchina nonostante ciò rimaneva sufficientemente sicura. Ad Alan Turing e agli analisti di Bletchley Park furono forniti fondi illimitati. Turing osservò che i messaggi trasmessi dall'esercito tedesco avevano una struttura prevedibile. Ad esempio intorno alle 6:00 di ogni mattina veniva trasmesso un bollettino meteo con una struttura molto rigida, per cui era logico supporre che la parola wetter (tempo atmosferico) comparisse in una determinata posizione; l'esperienza accumulata a Bletchley Park forniva diversi indizi di questo tipo; queste supposizioni furono chiamate *cribs*. Quindi, crib è un frammento di testo in chiaro che può essere dedotto in base a considerazioni non crittoanalitiche. Questo rivoluzionò il mondo della crittografia, per la prima volta un attacco a un sistema crittografico viene condotto non solo fondandosi sul testo cifrato, perché non più sufficiente, ma fondandosi sul testo cifrato e testi precedenti di cui si possiede testo cifrato e testo in chiaro. Nella crittoanalisi moderna questo tipo di attacco viene chiamato "known plaintext".

Turing pensò che i cribs rappresentassero la chiave per la risoluzione di Enigma e la supposizione si rivelò esatta. Il procedimento di Turing si rivelò assolutamente geniale: individuato un potenziale crib si chiama e una configurazione di Enigma che fornisce la cifratura della prima lettera in chiaro a con A. Si ricerca nel crib la lettera in chiaro coincidente con A e questa dista k lettere da a quindi ha assetto $e+k$. Si procede in modo analogo fino trovare una lettera cifrata coincidente con a e si indica con n il numero di lettere necessarie a tornare ad a così da formare un ciclo. Collegando n macchine Enigma in modo che esse abbiano collegamenti tra lettere cifrate e le lettere del crib che si sono supposte correlate, si esclude il pannello a prese multiple in modo non banale. Nella pratica Turing collegò 3 macchine Enigma, trovato un potenziale crib del tipo descritto e di lunghezza 3, osservò che a subisce una modifica dovuta al pannello a prese multiple che sarà indicata con L1 prima dei rotori. Successivamente, subirà una codifica sempre dal pannello prima dell'illuminazione sul display che si denoterà con L2, questa sarà la stessa subita dalla seconda lettera in chiaro. Quindi la seconda lettera subirà anch'essa L2 in ingresso e una nuova codifica dovuta al pannello indicata con L3 prima di essere visualizzata. La terza e ultima lettera del ciclo in entrata subisce L3 ma poichè deve risultare uguale ad a in uscita deve subire L1; per la precisione tutte le Ln in uscita sarebbero le inverse di quelle in entrata ma essendo trasposizioni coincidono. Così facendo si ottiene che la cifratura dipende solo dai rotori ed è sufficiente controllare le 17.576 possibili impostazioni compatibili. Quindi, il procedimento di verifica automatizzato consisteva nel collegare l'output del primo gruppo di scambiatori con l'input del secondo gruppo in corrispondenza di L1. Poiché questo valore non era noto era necessario collegare le 26 uscite del primo gruppo con i 26 ingressi del secondo formando 26 circuiti, ciascuno dotato di una lampadina per evidenziarne la chiusura. Quando si scopriva il

giusto orientamento degli scambiatori, uno dei circuiti si chiudeva causando l'accensione della lampadina.

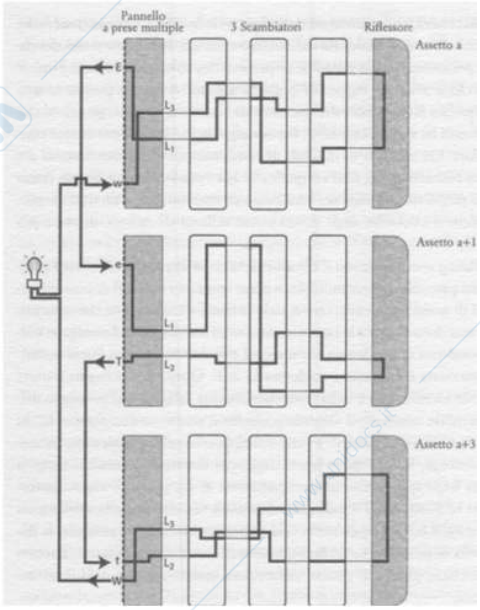


Figura 18: Crittoanalisi di Turing

La concatenazione di 3 macchine Enigma che ruotavano ogni secondo la propria posizione nel peggiore dei casi avrebbe impiegato 5 ore per trovare l'impostazione corretta. Quindi la macchina detta "bomba" creata da Turing testava tutte le possibili impostazioni facendo ruotare diversi rotori che si fermavano nel caso di un assetto corretto.

La prima bomba di Turing venne sviluppata nel 1940, per un costo di 100.000 sterline. Era costituita da dodici gruppi di rotori collegati, che impiegavano 7 giorni per calcolare una chiave giornaliera. Cinque mesi dopo fu sviluppata Agnus Dei, la quale impiegava un'ora per calcolare la chiave giornaliera, a partire da un crib.

Vigenere e Enigma furono due grosse sfide combattute dai crittoanalisti, risolte sfruttando delle debolezze negli algoritmi:

- Natura ciclica delle sostituzioni, per quanto riguarda lo schema di Vigenere
- Uso improprio o leggero da parte degli utenti, per quanto riguarda Enigma

La debolezza del Vigenère sta nell'essere, di fatto, un insieme di n cifrari di Cesare, dove n è la lunghezza della chiave; se il crittoanalista riesce a determinare la lunghezza della chiave (nel nostro caso, n) la decrittazione diventa molto semplice. Per far ciò si possono utilizzare metodi statistici per trovare n, e successivamente applicare l'analisi delle frequenze a ciascun alfabeto cifrante.

Proviamo ad utilizzare una chiave lunga quanto il messaggio da cifrare con metodo Vigenere. Supponiamo che il testo cifrato sia: UCMMWNSMDOKONFTVBLQPA. Si suppone che alcune parole comuni siano presenti nel testo, ad esempio la parola "non":

Testo Cifrato: UCMMWNSMDOKONFTVBLQPA

Testo in chiaro: nonnon??????non??????

Chiave: HOZZIA??????ARG??????

Esaminiamo la parola "chi":

Testo Cifrato: UCMMWNSMDOKONFTVBLQPA

Testo in chiaro: chinon??????non??????

Chiave: SVEZIA??????ARG??????

A questo punto ipotizziamo che la chiave, già contenente la parola Svezia, contenga la parola argentina:

Testo Cifrato: UCMMWNSMDOKONFTVBLQPA

Testo in chiaro: chinon??????nonrosica

Chiave: SVEZIA??????ARGENTINA

Possiamo ipotizzare che il testo cifrato sia "chi non risica non rosica":

Testo Cifrato: UCMMWNSMDOKONFTVBLQPA

Testo in chiaro: chinonrisicanonrosica

Chiave: SVEZIABELGIOARGENTINA

Il cifrario di Vernam è un sistema crittografico basato sul cifrario di Vigénère, al quale aggiunge il requisito che la chiave sia lunga quanto il testo e non riutilizzabile (per questo viene spesso chiamato OTP, acronimo per l'inglese "One Time Pad", letteralmente "blocco monouso"). Il cifrario di Vernam è l'unico sistema crittografico la cui sicurezza è comprovata da una dimostrazione matematica. Si può facilmente capire quanto sia scomodo distribuire in modo sicuro chiavi di tale dimensione, ciò nonostante è stato utilizzato per le comunicazioni con le spie, che venivano equipaggiati di taccuini ("pad" in inglese) contenenti una lunga chiave per ogni pagina, da poter strappare una volta utilizzata ("one time", ovvero "un solo uso"). La sua forma più classica è quella dove la chiave ha la stessa forma del testo (a ogni lettera viene associato il numero corrispondente $A = 0, B = 1, C = 2$) e che sfrutta l'operazione di somma circolare (quella per cui dopo la lettera Z c'è di nuovo la lettera A, quindi $A+C = 0+2 = 2 = C$, $B + C = 1 + 2 = 3 = D$, $Z + C = 25 + 2 = 27 = 1 = B$, $Z + Z = 25 + 25 = 50 = 24 = Y$). É importante ribadire che questo tipo di chiave deve essere lunga quanto il messaggio che cifra e può essere utilizzata una sola volta, pena la perdita della validità delle ipotesi iniziali e la riduzione da sistema "inattaccabile" a sistema "facilmente attaccabile" dal metodo Kasiski, una specializzazione del metodo crittanalitico di analisi delle frequenze.

É tuttavia molto diffusa, specialmente nell'ambiente informatico, la forma che fa utilizzo dell'operazione logica XOR (disgiunzione esclusiva), che del resto non è altro che l'addizione circolare dei singoli bit. Un esempio è dato dalla Figura sottostante (Figura 19).

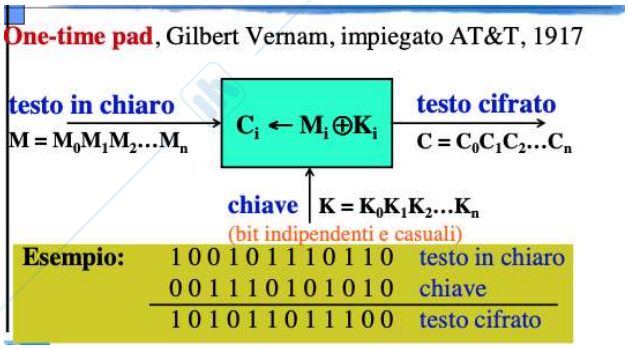


Figura 19: OTP - Un cifrario perfetto

Questo metodo è unconditionally secure, ossia indipendentemente dal tempo e dalle risorse a disposizione è impossibile decrittografare il testo cifrato. É impossibile, senza un repertorio di chiave, decifrare il messaggio.

Questo metodo è stato utilzzato da spie russe, per la famosa linea rossa, una linea diretta di telecomunicazioni fra gli Stati Uniti d'America e la Russia (fino al 1991 Unione Sovietica). La linea fu installata nel 1963, come conseguenza della crisi di Cuba fra il Cremlino, sede del governo sovietico, e il Pentagono, sede del Dipartimento della Difesa degli Stati Uniti.

É stata usata una tecnica One Time Pad per la comunicazione da Che Guevara a Fidel Castro. É stato scoperto nel 1969. Il 9 ottobre 1967, il rivoluzionario argentino Ernesto Che Guevara fu assassinato per ordine del dittatore boliviano Barrientos, su diretto suggerimento telefonico del presidente statunitense Lyndon Johnson. Era stato arrestato il giorno prima a Valleverde e in tasca gli era stato trovato un foglio con una lunga sequenza casuale di numeri, senza alcun ordine apparente. Come lo stesso Che racconta nel Diario di Bolivia, la sequenza gli serviva per codificare i messaggi scambiati con Castro secondo il classico metodo Vernam. Il testo da cifrare veniva anzitutto tradotto, secondo una tabella fissa, in una sequenza di numeri che veniva poi appaiata, cifra per cifra, alla sequenza casuale che costituiva la chiave. Il messaggio codificato consisteva della sequenza di numeri ottenuti sommando il messaggio originale e la chiave, cifra per cifra e senza riporti. Il metodo era, e rimane, perfettamente sicuro: se la chiave è effettivamente casuali, lo diventa anche il messaggio codificato, che può essere decodificato soltanto possendendo la chiave stessa. Il problema sta, appunto, nel "se": esistono sequenze di numeri veramente casuali? E, più in generale, esiste il caso?

2.3 Probabilità

Il calcolo delle probabilità nasce con lo studio dei giochi d'azzardo all'incirca nel XVII secolo. La definizione classica di probabilità è: *la probabilità, $P(A)$, di un evento A è il rapporto tra il numero N di casi "favorevoli" e il numero totale M di casi possibili e mutuamente escludentesi* $P(A) = N/M$. Gli eventi mutuamente escludentesi o incompatibili sono eventi aleatori che non possono verificarsi simultaneamente, ad esempio, l'apparizione di testa e di croce nel lancio di una moneta.

La valutazione di probabilità dipende dallo stato di informazione: invece di "probabilità assoluta", si parla di probabilità condizionata ad una certa informazione e si indica con $P(A|B)$, letta "probabilità di A dato B". Un esempio di questo tipo di probabilità è quando si dice che la probabilità della faccia di un dado sia $1/6$ si sta assumendo che dado e lancio siano perfettamente regolari. Se siamo interessati alla probabilità della faccia "6" del dado sapendo che è uscito un numero pari, possiamo calcolarla nel seguente modo: $P(6|\text{"pari"}) = 1/3$. Se vogliamo che la somma in due lanci di dadi sia 12 sapendo che in entrambi i lanci è uscito un numero pari, viene calcolata con $P(\text{"somma=12"}|\text{"entrambi pari"}) = 1/9$.

Con il termine probabilità composte si intende: "La probabilità del prodotto di due eventi è uguale al prodotto della probabilità di uno degli eventi per la probabilità condizionata dell'altro calcolata a condizione che il primo abbia luogo: $P(A \cap B) = P(A)P(B|A) = P(B)P(A|B)$. Se gli eventi sono indipendenti si ha che $P(B|A) = P(B)$ e $P(A|B) = P(A)$ quindi: la probabilità del prodotto di due eventi indipendenti è uguale al prodotto delle probabilità di questi eventi: $P(A|B) = P(B|A) = P(A)P(B)$. Il Teorema di Bayes indica che: $P(A|B) = \frac{P(B|A)*P(A)}{P(B)}$.

Un cifrario può essere definito da:

- Gen : algoritmo di generazione delle chiavi, dove K è lo spazio delle chiavi
- M spazio dei messaggi in chiaro
- C spazio dei messaggi cifrati, dove $Enc_k(m) = c$ è la funzione di cifratura; mentre $Dec_k(c) = m$ è la funzione di decrittazione.

Consideriamo i messaggi M e i testi cifrati C. $P(M)$ rappresenta la probabilità di osservare il messaggio M dal punto di vista dell'avversario, diversi messaggi hanno diverse probabilità. Invece, $P(C)$ rappresenta la probabilità di osservare il testo cifrato C. La distribuzione è determinata dalle distribuzioni di K e M. In un cifrario perfetto: M e C sono indipendenti.

Uno schema di cifratura dato da Gen, Enc e Dec su uno spazio dei messaggi M è perfettamente sicuro se: per ogni distribuzione di probabilità su M (per ogni messaggio m) e per ogni distribuzione di probabilità su C (per ogni testo cifrato c) si ha che $Prob(M|C) = Prob(M)$, ossia che l'avversario non ha alcuna informazione sul messaggio in chiaro M osservando il cifrato C.

$Prob(M|C) = \frac{Prob(M)Prob(C|M)}{Prob(C)} = \frac{Prob(M)Prob(C)}{Prob(C)} = Prob(M)$

É possibile dimostrare che se lo schema è perfetto allora $|K| \geq |M|$. I Risultati di Shannon indicano che considerando $|K| = |M| = |C|$, un cifrario ha sicurezza perfetta se e solo se:

1. Ogni chiave k è scelta con probabilità $1/|K|$ da Gen
2. Per ogni m e c, esiste una unica chiave k tale che $Enc_k(m) = c$.

Il cifrario One-time Pad può essere considerato un cifrario perfetto, dal momento che M e C sono indipendenti.

3 Crittoanalisi

In crittografia, il principio di Kerckhoffs (noto anche come assunzione, assioma o legge di Kerckhoffs), fu enunciato dal crittografo olandese Auguste Kerckhoffs alla fine del 1880. Esso afferma che la sicurezza di un crittosistema non deve dipendere dal tenere celato l'algoritmo crittografico ma solo dal tenere celata la chiave. In altre parole, il sistema deve rimanere sicuro anche nell'ipotesi che il nemico conosca l'algoritmo di crittazione.

In effetti, tutte le tecniche moderne di crittografia vengono rese pubbliche, così che si possa testarne la robustezza permettendo a chiunque di romperle.

Ci sono diverse situazioni di attacco, che corrispondono a diversi gradi di sicurezza. Più un crittosistema resiste a livelli di attacco più alti, più è robusto:

- Known Ciphertext Attack: l'avversario conosce solo il testo cifrato;
- Known Plaintext Attack: l'avversario conosce anche il testo in chiaro;
- Chosen Plaintext Attack: l'avversario può ottenere la cifratura di un testo in chiaro a sua scelta;
- Chosen Ciphertext Attack: l'avversario può ottenere la decifratura di un testo cifrato a sua scelta;
- Chosen Text Attack: l'avversario può ottenere la cifratura e la decifratura di coppie di testi chiaro/cifrato.

Se un cifrario viene rotto conoscendo solo il testo cifrato, allora è molto debole. Se, invece, un attaccante riceve tutte le coppie di testo in chiaro e cifrato che vuole, e non sa come uscirne, allora vuol dire che il cifrario è molto robusto.

3.1 Cifrari a shift

Questi cifrari sono quelli che prendono una lettera e la spostano di un numero di posizioni nell'alfabeto. La struttura sottostante del testo però rimane, ed in particolare la frequenza delle lettere cifrate corrisponde alla frequenza delle lettere in chiaro nella lingua in cui è stato scritto il

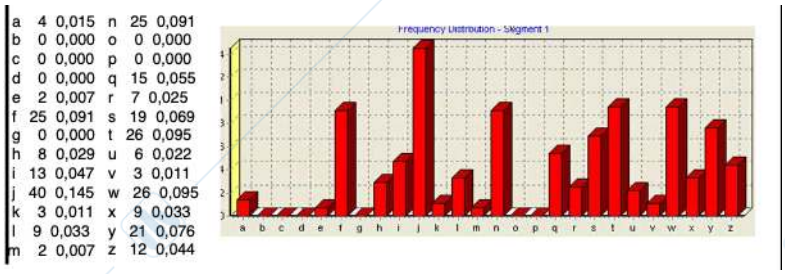


Figura 20: Crittoanalisi dei cifrari a shift

testo. Le frequenze di un testo si ricavano facilmente, poi si vede quali frequenze corrispondono a quelle del testo cifrato (Figura 20).

Nei crittosistemi a sostituzione monoalfabetica, bisogna mappare le lettere qua e là finché non viene trovata la chiave. Computazionalmente parlando è molto semplice.

3.1.1 Cifrario di Hill

Nella sezione 2.1.5 abbiamo visto il funzionamento del cifrario di Hill. Ricapitolando, il testo in chiaro $(y_1, y_2, ..., y_m)$ è uguale al testo cifrato $(x_1, x_2, ..., x_m)$, moltiplicato per la matrice $m \times m$ K. Quindi: $y = E_k(x) = xK$. Per decifrare riscriviamo il testo codificato come vettore, che poi moltiplicheremo semplicemente per la matrice inversa della matrice chiave K: $x = D_k(y) = yK^{-1}$. Quindi, K^{-1} è l'inversa della matrice chiave: $KK^{-1} = I$ (Matrice Identità). Una matrice A $n \times n$ è invertibile:

- se ha numeri reali, quando $det(A) \neq 0$
- se ha numeri modulo m, quando $gcd(det(A), m) = 1$

Quindi, una matrice A è invertibile modulo 26 se $gcd(det(A), 26) = 1$ o $det(A) \neq 0 \pmod{13}$ e $det(A) \neq 0 \pmod{2}$. Per quanto riguarda la sicurezza, sfortunatamente, il cifrario di Hill nella sua versione di base è vulnerabile ad un known-plaintext attack (KPA) in quanto è completamente lineare. Un avversario che intercetti n^2 coppie di caratteri in chiaro e cifrati può impostare un sistema lineare che può (di solito) essere risolto facilmente; può capitare che il sistema sia indeterminato, ma in questo caso basta aggiungere alcune altre coppie di caratteri in chiaro e cifrati per poterlo risolvere. Calcolare la soluzione con un algoritmo standard di algebra lineare richiede poco tempo.

3.1.2 Metodo Kasiski

Il cifrario di Vigenère resiste all'analisi delle frequenze, dal momento che una lettera cifrata corrisponde a più simboli in chiaro ed esiste un numero grande di chiavi. Babbage (1834) e Kasiski (1863) furono i primi a cimentarsi nella crittanalisi, effettuando lo studio delle ripetizioni per individuare la lunghezza della chiave e, successivamente applicando un'analisi delle frequenze in ognuno degli alfabeti cifranti corrispondenti alle lettere della chiave. Quindi, un possibile attacco al cifrario di Vigenère è il Known Ciphertext Attack, poiché conoscendo il testo cifrato è possibile utilizzare metodi statistici per risalire al messaggio in chiaro: utilizzando l'indice di coincidenza, in cui si risale alla lunghezza della chiave e utilizzando l'indice di mutua coincidenza per risalire alla chiave.

Il metodo Kasiski è un metodo crittoanalitico per l'attacco del cifrario di Vigenère e dei cifrari ad esso simili. Prende il nome dal maggiore prussiano Friedrich Kasiski, che nel 1863 pubblicò un metodo di decifratura della tavola di Vigénère. Il maggiore Kasiski notò che spesso in un crittogramma di Vigénère si possono notare sequenze di caratteri identiche, poste ad una

certa distanza fra di loro; questa distanza può, con una certa probabilità, corrispondere alla lunghezza della chiave, o a un suo multiplo. In genere la stessa lettera con il cifrario di Vigénère viene cifrata in modo diverso nelle sue varie occorrenze, come si confà ai cifrari polialfabetici, ma se due lettere del testo in chiaro sono poste ad una distanza pari alla lunghezza della chiave (o un suo multiplo), questo fa sì che vengano cifrate nello stesso modo. Individuando tutte le sequenze ripetute (cosa che avviene frequentemente in un testo lungo), si può dedurre quasi certamente che la lunghezza della chiave è il massimo comune divisore tra le distanze tra sequenze ripetute, o al più un suo multiplo. Conoscere la lunghezza n della chiave permette di ricondurre il messaggio cifrato ad n messaggi intercalati cifrati con un cifrario di Cesare facilmente decifrabile. Il test di Kasiski consente di determinare la lunghezza della chiave, ma non i caratteri che la compongono. Vedremo un altro metodo, basato su statistiche relativo al testo cifrato per determinare la lunghezza della chiave: indice di coincidenza. Invece, l'indice mutuo di coincidenza determina i caratteri della chiave.

3.1.3 Indice di coincidenza

L'indice di coincidenza è stato definito da Wolfe Friedman nel 1920. L'indice di coincidenza di una stringa $x_1x_2...x_n$ è rappresentato dalla notazione $IC(x_1x_2...x_n)$ e rappresenta la probabilità che due caratteri presi a caso nella stringa siano uguali.

Il test di Friedman si basa principalmente su alcuni semplici calcoli di probabilità e sul concetto di indice di coincidenza. Supponiamo di avere a disposizione n lettere, di cui n_i è il numero di lettere i-esime, e di volere sapere quale è il numero di coppie possibili formate da lettere uguali; allora questo sarà dato da: $\sum_{i=0}^{25} \frac{n_i(n_i-1)}{2}$. Quindi considerando una delle prime definizioni di probabilità, e cioè il numero di casi possibili fratto il numero di casi favorevoli, ne verrà che la probabilità di scegliere a caso una coppia di lettere uguali, tenendo presente che per una stringa lunga n, posso scegliere 2 caratteri in $\binom{n}{2}$ modi, cioè $\frac{n*(n-1)}{2}$, è:

$$IC(x_1x_2...x_n) = \frac{\sum_{i=0}^{25} \binom{n_i}{2}}{\binom{n}{2}} = \frac{\sum_{i=0}^{25} n_i(n_i-1)}{n(n-1)}$$

dove n_i rappresenta il numero di occorrenze del carattere i nella stringa $x_1x_2...x_n$.

Se $x_1x_2...x_n$ è un testo in inglese, allora $IC(x_1x_2...x_n) \approx \sum_{i=0}^{25} p_i^2 = 0.065$, dove p_i rappresenta la probabilità del carattere i-esimo in Inglese, dato dalla tabella, riportata nella Figura 21.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
7,3	1,3	3,5	4,3	12,8	3,0	2,0	3,5	7,8	0,3	0,5	3,7	2,8	7,8	7,5	2,8	0,5	8,5	6,0	9,3	3,0	1,5	1,5	0,5	2,3	0,3
p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}	p_{16}	p_{17}	p_{18}	p_{19}	p_{20}	p_{21}	p_{22}	p_{23}	p_{24}	p_{25}

Figura 21: Probabilità dei caratteri in lingua Inglese

Se $x_1x_2...x_n$ sono caratteri scelti a caso, allora $ICx_1x_2...x_n \approx \sum_{i=0}^{25} \frac{1}{26}^2 = 0.038$.

Se $x_1x_2...x_n$ è un testo in italiano, allora $IC(x_1x_2...x_n) \approx \sum_{i=0}^{25} p_i^2 = 0.075$, dove p_i rappresenta la probabilità del carattere i-esimo in Italiano, dato dalla tabella, riportata nella Figura 22.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
10,3	0,9	4,3	3,8	12,6	0,8	2,0	1,1	11,6	0,0	0,0	6,6	2,6	6,6	8,7	3,2	0,6	6,7	6,1	6,1	3,0	1,5	0,0	0,0	0,0	0,9
p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}	p_{16}	p_{17}	p_{18}	p_{19}	p_{20}	p_{21}	p_{22}	p_{23}	p_{24}	p_{25}

Figura 22: Probabilità dei caratteri in lingua Italiano

Se $x_1x_2...x_n$ sono caratteri scelti a caso, allora $ICx_1x_2...x_n \approx \sum_{i=0}^{25} \frac{1}{26}^2 = 0.038$.

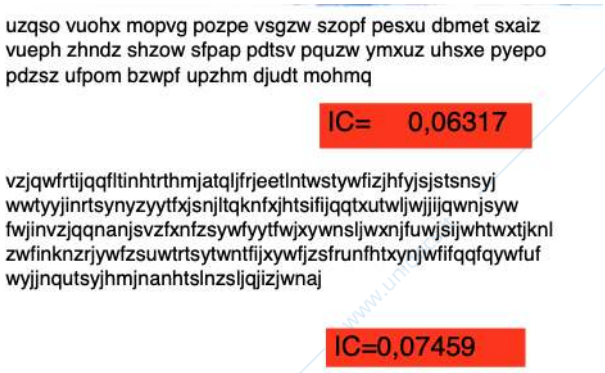


Figura 23: Sostituzione monoalfabetico

Calcolando l'indice di coincidenza per una lettera su un testo cifrato, possiamo stabilire con che lingua abbiamo a che fare, in base agli indici di incidenza dei vari alfabeti e, se si avvicina molto, possiamo anche stabilire che si tratta di una sostituzione monoalfabetica, dal momento che mantiene la frequenza linguistica.

Torniamo quindi al metodo per determinare la lunghezza t (t rappresenta il periodo di ripetizione) della chiave, andando per ipotesi:

- Se $t = 1$, significa che si sta utilizzando una sostituzione monoalfabetica. Calcolo l'indice di coincidenza sul testo cifrato, ipotizzando un periodo pari a 1, se ottengo valori lontani dagli indici di incidenza delle varie lingue, posso escludere la sostituzione monoalfabetica:

$$IC(C_0C_1...C_n) \approx \begin{cases} 0.075 & \text{se } t=1 \\ 0.038 & \text{se } t\neq 1 \end{cases}$$

Figura 24: Sostituzione monoalfabetica

- Se non ottengo risultati con $t = 1$, proseguo il mio ragionamento ipotizzando $t = 2$. Per poter calcolare l'IC con $t = 2$, spezzo il mio testo cifrato in due sottotesti, dati dai caratteri pari ($C_0C_2...$) e dai caratteri dispari ($C_1C_3...$).

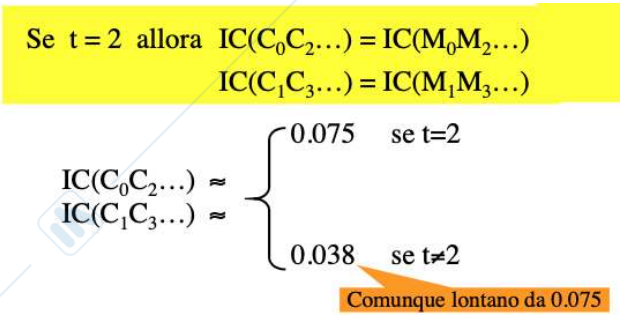


Figura 25: Sostituzione con periodo 2

- Se ottengo risultati distanti con $t = 2$, ipotizzo che il periodo sia $t = 3$.

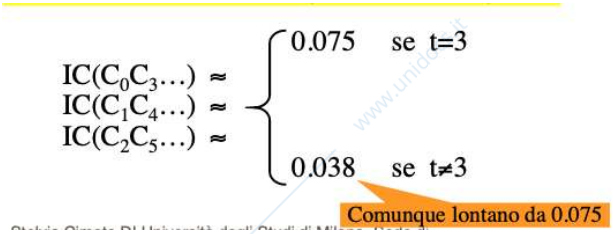


Figura 26: Sostituzione con periodo 3

- Proseguo fino a che non trovo tutti gli indici di coincidenza simili a un indice di coincidenza di una lingua conosciuta.

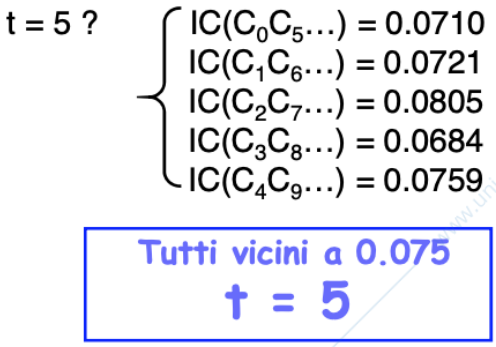


Figura 27: Sostituzione con periodo 5

Utilizzando l'indice di coincidenza per determinare la lunghezza della chiave t. Invece, per determinare il valore della chiave $K_0K_1K_2...K_{t-1}$ utilizzo l'indice mutuo di coincidenza.

Riassumendo, come applico questo metodo a Vigenere? Suddivido il mio testo cifrato in blocchi di una certa lunghezza, e li metto tutti in colonna, col primo blocco in alto e gli altri tutti sotto a seguire. Successivamente, calcolo l'indice di coincidenza delle colonne. Il motivo è che se il mio blocco è lungo come la chiave, allora vuol dire che i caratteri nella stessa colonna sono stati cifrati con la stessa lettera. Pertanto, se l'indice di coincidenza della colonna assomiglia a quello della lingua del testo in chiaro, allora posso presumere che la chiave è lunga come il mio blocco. Altrimenti, provo con un blocco via via più grande etc.

3.1.4 Indice mutuo di coincidenza

Per trovare i caratteri della chiave K0K1K2...Kt-1 si usa l'indice mutuo di coincidenza (IMC) che ci dice qual è la probabilità di estrarre da 2 stringhe 2 caratteri uguali. Quindi, date due stringhe $x_1x_2...x_n$ e $y_1y_2...y_n$, $IMC(x_1x_2...x_n; y_1y_2...y_n)$ corrisponde alla probabilità che un carattere in $x_1x_2...x_n$ ed uno in $y_1y_2...y_n$ presi a caso, siano uguali. L'equazione che ci permette di calcolare IMC è: $IMC(x_1x_2...x_n; y_1y_2...y_n) = \frac{\sum_{i=0}^{25} f_i \cdot f'_i}{n \cdot n'}$, dove f_i rappresenta il numero di occerrenze del carattere i-esimo in $x_1x_2...x_n$; mentre f'_i è il numero di occorrenze del carattere i-esimo in $y_1y_2...y_n$.

Ad esempio, se $K_0 = 1$ e $K_1 = 4$, avremo, sulla base della Figura 28:

- Prob. AA nel cifrato = Prob. ZW nel testo in chiaro
- Prob. BB nel cifrato = Prob. AX nel testo in chiaro
- Prob. CC nel cifrato = Prob. BY nel testo in chiaro

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Figura 28: Indice mutuo di coincidenza

Qual è il valore medio di $IMC(C_0C_tC2_t...; C_1C_{t+1}C_{2t+1}...)$?

- Prob. AA nel cifrato = $p_{-K_0}p_{-K_1}$
- Prob. BB nel cifrato = $p_{1-K_0}p_{1-K_1}$
- Prob. CC nel cifrato = $p_{2-K_0}p_{2-K_1}$

Quindi, abbiamo che $IMC(C_0C_tC2_t...; C_1C_{t+1}C_{2t+1}...) \approx \sum_{i=0} 25 p_{i-K_0}p_{i-K_i} = \sum_{h=0}^{25} p_h p_{h+(K_0-K_1)}$.

Siccome siamo in aritmetica modulare possiamo dire che $\sum_{h=0}^{25} p_h p_{h+(K_0-K_1)} = \sum_{h=0}^{25} p_h p_{h-(K_0-K_1)}$.

I valori possibili dell'indice mutuo di coincidenza, quindi, sono solamente 14. Quando troverò lo shift corretto, quindi, quei valori di K_0 e K_1, che permettono di rendere vera $K_0 - K_1 = 0$. In questo caso, l'indice mutuo di coincidenza congruente alla media IMC di una determinata lingua (Figura 29).

valore di K_0-K_1	media IMC
0	0.075
1, 25	0.033
2, 24	0.034
3, 23	0.034
4, 22	0.047
5, 21	0.027
6, 20	0.032
7, 19	0.026
8, 18	0.027
9, 17	0.023
10, 16	0.024
11, 15	0.027
12, 14	0.015
13	0.021

K_0-K_1= 0 ⇒ media IMC = 0.075

K_0-K_1≠ 0 ⇒ media IMC ≤ 0.047

Italiano

Figura 29: Indice mutuo di coincidenza per la lingua italiana

Supponiamo quindi di conoscere la dimesione t della chiave, la chiave è $K = (k_0, k_1, ..k_{t-1})$, con ogni k_i che indica il numero da addizionare per ottenere il testo cifrato. Si divide il ciphertext in t stringhe $y_0, y_1, ...y_{t-1}$ e si prendono due caratteri casuali di y_i e y_j:

- La probabilità che essi siano A è $P_{-k_i}p_{-k_j}$
- La probabilità che essi siano B è $P_{1-k_i}p_{1-k_j}$

Vediamo un esempio, sempre basandoci su supposizioni:

- Supponiamo che $K_0 - K_1 = 0$, cioè se K_0 e K_1 distano fra loro 0, cioè sono lo stesso carattere. Calcolando $IMC(C_0C_t...; C_1C_{t+1}...)$ bisogna ottenere un valore coerente con il valore di IMC medio della lingua in analisi. Se non è così si procede ipotizzando.
- Procediamo con $K_0 - K_1 = 1$, quindi calcoliamo $IMC(Y_0Y_t...; C_1C_{t+1}...)$, dove $Y_i \leftarrow (C_i - 1) mod 26$. Si procede così fino a che non si trova un valore coerente.

Vediamo un applicazione di MIC: possiamo fissare un sottotesto y_i e modificare y_j (sottraendo) da 1 a 25. Il valore che fa ottenere un MIC vicino a 0.075 (0.065 per la lingua inglese) indica il corretto valore di shift $k_i - k_j$.

Per applicare l'indice mutuo di coincidenza per trovare i caratteri della chiave dobbiamo considerare due blocchi di testo cifrato, uno cifrato con K_0 e l'altro con K_1. Voglio stabilire qual è lo shift tra K_0 e K_1. Il procedimento è, quindi, quello di calcolare lo shift relativo tra la prima colonna e la seconda colonna. Calcolato questo, posso calcolare quello tra la seconda e la terza e così via, poi tra la prima e la terza, la prima e la quarta etc.

K_1-K_0 = 16

.0325	.0415	.0422	.0436	.0385	.0444	.0388	.0390	.0347
.0350	.0404	.0315	.0419	.0398	.0370	.0380	.0703	.0314
.0346	.0356	.0436	.0269	.0327	.0298	.0381	.0371	

K_2-K_0 = 25

.0326	.0341	.0345	.0365	.0245	.0367	.0284	.0393	.0394
.0373	.0358	.0432	.0439	.0399	.0382	.0363	.0334	.0315
.0355	.0449	.0384	.0518	.0403	.0313	.0370	.0738	

Figura 30: Determinare la chiave, dal sistema lineare di equazioni

In questo modo ottengo un sistema lineare che si può risolvere facilmente, esprimendo tutti i K_i in funzione di K_0.

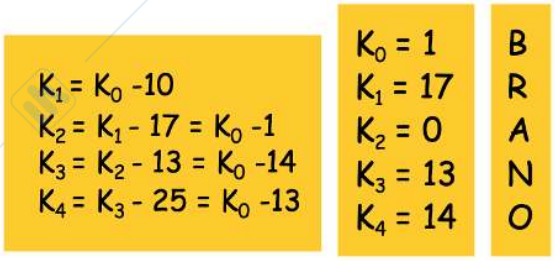


Figura 31: Determinare la chiave, dal sistema lineare di equazioni

A quel punto si provano tutti e 26 i possibili valori che può assumere K_0, scegliendo quello che rende più sensato il testo.

4 Cifratura Simmetrica - DES

In crittologia un algoritmo di cifratura a blocchi (dall'inglese: block cipher) è un algoritmo a chiave simmetrica operante su un gruppo di bit di lunghezza finita organizzati in un blocco. A differenza degli algoritmi a flusso che cifrano un singolo elemento alla volta, gli algoritmi a blocco cifrano un blocco di elementi contemporaneamente. I blocchi sono a lunghezza fissa. Alcuni esempi di cifrati a blocchi sono:

- Data Encryption Standard (DES)
- DES triplo
- Advanced Encryption Standard (AES)
- Blowfish
- RC5, RC6

A differenza dei cifrari a blocchi, la cifratura di flussi esegue la crittografia di un flusso digitale di dati (un bit o un byte per volta): esiste un flusso di chiavi. I cifrari a flusso costituiscono un approccio alla cifratura simmetrica differente da quello dei cifrari a blocchi: questi prevedono che si effettui su successivi estesi blocchi di simboli una determinata trasformazione che non cambia da blocco a blocco. Questa distinzione in realtà non è sempre netta: utilizzando alcune modalità di funzionamento si può utilizzare la cifratura a blocchi per cifrare flussi.

I cifrari a blocchi operano su blocchi di n bit di input per produrre blocchi di n bit di output. Infatti, con blocchi da n bit di testo in chiaro, si ottengono 2^n possibili input. Quando avviene la trasformazione del blocco di testo in chiaro in testo cifrato, occorre che la trasformazione da n bit in chiaro ad n bit cifrati sia univoca, ovvero reversibile, o non singolare. Ciò significa che ogni blocco di testo in chiaro deve produrre un blocco cifrato univoco.

La chiave è quella che effettua questa mappatura tra n bit in chiaro ed n bit cifrati. In generale, se ho un blocco di n bit, la chiave deve avere lunghezza di almeno $n * 2^n$ bit, perché deve coprire tutte le possibili mappature dei miei n bit. Infatti, con n bit di blocco, si avranno 2^n possibili valori del blocco. L'idea sarebbe che ogni singolo valore del blocco possa essere mappato in un altro singolo valore del blocco: una mappatura 1 a 1, in pratica. Occorrono quindi 2^n mappature. Ogni mappatura è composta da una sequenza di n bit, trattandosi appunto di un blocco di n bit.

Testo chiaro	cifrato	Testo chiaro	cifrato
00	11	00	11
01	10	01	10
10	00	10	01
11	01	11	01

Figura 32: Mappaggi

La Figura 32 mostra un esempio: se si sceglie di avere un blocco di 2 bit, ho i seguenti valori possibili: 00, 01, 10, 11 = $2^2 = 4$ valori. Quindi $n = 4$ bit di testo in chiaro implicano 16 possbili stati di input (2^4), mappati in 16 possibili output, ossia 4 bit di testo cifrato (Figura 33).

Crittografia		Decrittografia	
0000	1110	0000	1110
0001	0100	0001	0011
0010	1101	0010	0100
0011	0001	0011	1000
0100	0010	0100	0001
0101	1111	0101	1100
0110	1011	0110	1010
0111	1000	0111	1111
1000	0011	1000	0111
1001	1010	1001	1101
1010	0110	1010	1001
1011	1100	1011	0110
1100	0101	1100	1011
1101	1001	1101	0010
1110	0000	1110	0000
1111	0111	1111	0101

Figura 33: Mappaggi

I blocchi di piccola dimensione equivalgono a una cifratura a sostituzione, la quale è vulnerabile all'analisi statistica. Per quanto riguarda blocchi grandi, si mascherano le caratteristiche statistiche, ma sorgono problemi pratici per la dimensione della chiave. Infatti, come già detto in precedenza, per n bit, la dimensione della chiave è $n * 2^n$. Quindi, se scegliessimo $n = 64$, la chiave avrebbe dimensione: $64 * 2^{64} = 2^{70}$ circa 10^{21} bit. La chiave, guardando la Figura 33, è la colonna di destra. La procedura di crittazione funziona nel seguente modo: il blocco in chiaro ha valore 010, quindi si prende la 010-esima voce della tabella, e si ottiene la sua cifratura. Si vede quindi che la colonna di destra è composta da 16 voci da 4 bit l'una: per l'appunto $n * 2^n$. Considerare una chiave del genere è veramente troppo oneroso, se uso blocchi di grosse dimensioni. Bisognerà quindi approssimare questo sistema ideale in un sistema reale, con chiavi molto più piccole.

4.1 Cifratura 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 nel 1973, da cui ha preso il nome di rete di Feistel; moltissimi algoritmi di cifratura a blocchi la utilizzano, 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 per ottenere l'operazione inversa: quindi i circuiti di cifratura e decifratura spesso sono gli stessi. Il meccanismo di cifratura ricorda le operazioni in cascata di Enigma.

L'idea alla base del cifrario è l'uso di cifrature in sequenza per ottenere cifrature più complesse di ogni singola componente. Per fare ciò, alterna le seguenti operazioni:

- Permutazioni
- Sostituzioni

Queste operazioni, reiterate più volte (round), conferiscono alla rete di Feistel le proprietà di "confusione e diffusione" descritte da Claude Shannon. Confusione e diffusione sono due proprietà che un algoritmo di cifratura sicuro deve possedere per essere considerato più o meno robusto, ovvero scarsamente attaccabile da un attacco crittoanalitico, in particolare in grado di contrastare attacchi basati sull'analisi statistica:

- La diffusione è la capacità dell'algoritmo di distribuire le correlazioni statistiche del testo lungo tutto l'alfabeto utilizzato dall'algoritmo di cifratura rendendo quanto più difficile possibile un attacco statistico. Quindi, si tratta di un'espansione della struttura statistica del testo. Ogni cifra del testo cifrato è prodotta da più cifre del testo in chiaro: $Y_n = \sum m_{n+i}(mod 26)$ con $i = 1, .., k$. Per blocchi binari si utilizza la trasposizione.
- La confusione è il fatto che la relazione tra la chiave e il testo cifrato sia quanto più complessa e non correlata possibile, in modo tale che non si possa risalire ad essa a partire dal testo cifrato. Quindi, si tratta di una complicazione delle relazioni statistiche fra testo cifrato e valore della chiave. Per fare questo, si utilizza un meccanismo di sostituzione complesso (non lineare).

Le caratteristiche dei cifrari di Feistel sono le seguenti:

- Dimensioni del blocco: blocchi grandi migliorano la sicurezza, ma riducono la velocità.
- Dimensioni della chiave: chiavi grandi migliorano la sicurezza, ma riducono la velocità.
- Numero di fasi: tutte le fasi hanno la stessa struttura.
- Algoritmo di schedulazione della chiave: a partire dalla chiave iniziale vengono prodotte tante sottochiavi quanti sono i round
- Funzione round: più è complessa più resiste alla crittoanalisi.

Poniamo F essere la funzione dei passaggi e $K_0, K_1, \ldots, K_n$ le sotto-chiavi rispettivamente dei passaggi $0, 1, \ldots, n$. Le operazioni basilari sono dunque le seguenti:

Dividi i dati di ingresso in due parti uguali, (L_0, R_0)

Per ogni round $i = 1, 2, \ldots, n$ calcola $L_i = R_{i-1}$ e $R_i = L_{i-1} \oplus f(R_{i-1}, K_{i-1})$, dove f è la funzione del round e K_i è la chiave di sessione.

Si ottiene il testo cifrato (L_n, R_n).

Senza considerare la funzione f, la decifrazione si ottiene con $R_{i-1} = L_i$ e $L_{i-1} = R_i \oplus f(L_i, K_i)$. Quindi, indipendentemente da f, la funzione è invertibile.

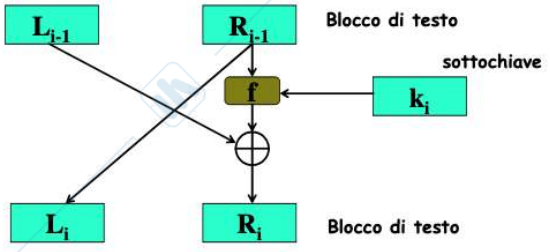


Figura 34: Struttura di Feistel

Un vantaggio di questo modello è che le funzioni f usate sono non invertibili e possono essere molto complesse.

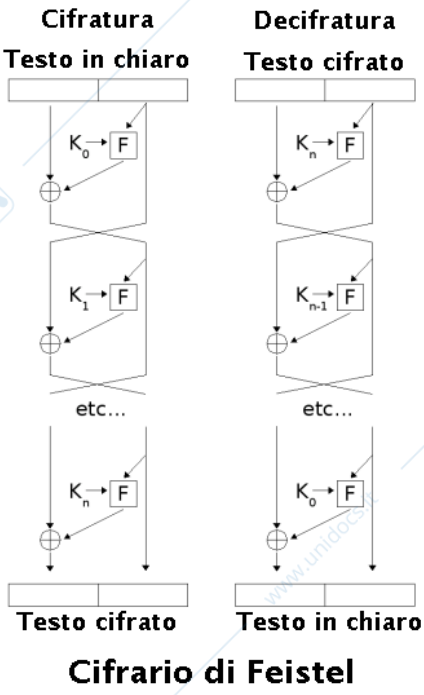


Figura 35: Struttura di Feistel

Il diagramma in Figura 35 mostra la cifratura e la decifratura del messaggio. Notare l'inversione della chiave di sessione per la decifratura, è l'unica differenza rispetto alla cifratura del messaggio.

DES e Blowfish sono due esempi di cifrari di Feistel.

4.2 DES

Le origini del DES risalgono all'inizio degli anni settanta. Nel 1973 l'NBS (National Bureau of Standards), ora chiamato NIST, dopo aver concluso una serie di studi sulla sicurezza informatica per il governo statunitense, stabilì che era necessario individuare un algoritmo di cifratura per documenti senza alcuna classificazione di sicurezza, pubblico e comune in modo da favorire la comunicazione protetta tra le varie organizzazioni governative statunitensi. Dopo essersi consultata con l'NSA il 15 maggio 1973 venne presentata la prima richiesta pubblica di uno standard di cifratura definito secondo criteri rigorosi. Nessuno degli algoritmi presentati superò i test dell'NSA e quindi il 27 agosto 1974 fu diramato un secondo bando. In quel periodo IBM sottopose come candidato il DEA, sviluppato tra il 1973 e il 1974 e basato su un precedente algoritmo denominato Lucifer (chiave da 128 a 56 bit). Il gruppo di lavoro che ad IBM era dedito allo studio degli algoritmi crittografici era capitanato da Horst Feistel: la struttura del Lucifer e del DEA era una novità del ricercatore e da lui prendeva il nome di rete di Feistel. Il 17 marzo 1975 il DES proposto fu pubblicato nel Federal Register. Furono richiesti commenti pubblici e l'anno seguente si tennero 2 congressi pubblici per discutere lo standard proposto. Nonostante le critiche, il DES fu approvato come standard federale nel novembre del 1976 e pubblicato il 15 gennaio 1977 come FIPS PUB 46, certificato per l'uso su tutti i dati non classificati. É stato in seguito riconfermato come standard nel 1983, 1987 (riesaminato come FIPS-46-1), 1992 (FIPS-46-2) e nuovamente nel 1999 (FIPS-46-3), nel quale si prescrive il "Triple DES". Il 26 maggio 2002, il DES è stato finalmente rimpiazzato dall'AES, l'Advanced Encryption Standard, in seguito ad un confronto pubblico. Ancora oggi, ad ogni modo, il DES è largamente utilizzato.

Il 19 maggio 2005, lo standard FIPS 46-3 è stato ufficialmente ritirato, ma il NIST ha approvato l'uso di Triple DES fino all'anno 2030 per informazioni governative sensibili.

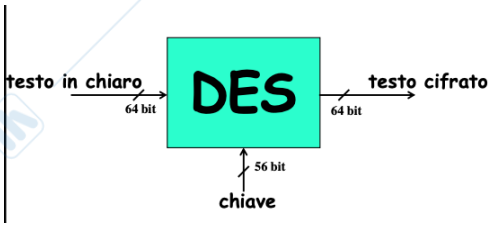


Figura 36: Data Encryption Standard

Il DES è l'archetipo della cifratura a blocchi, un algoritmo che prende in ingresso una stringa di lunghezza fissa di testo in chiaro e la trasforma con una serie di operazioni complesse in un'altra stringa di testo cifrato della stessa lunghezza. Nel caso del DES la dimensione del blocco è di 64 bit. Il DES usa inoltre una chiave per modificare la trasformazione in modo che l'operazione di decifratura possa essere effettuata solo conoscendo la chiave stessa. La chiave è lunga 64 bit ma solo 56 di questi sono effettivamente utilizzati dall'algoritmo. Otto bit sono utilizzati solo per il controllo di parità e poi scartati, per questo la lunghezza della chiave effettiva è riportata come di 56 bit. In particolare, l'ottavo bit di ogni blocco è il bit di parità e corrisponde allo XOR dei precedenti 7 bit.

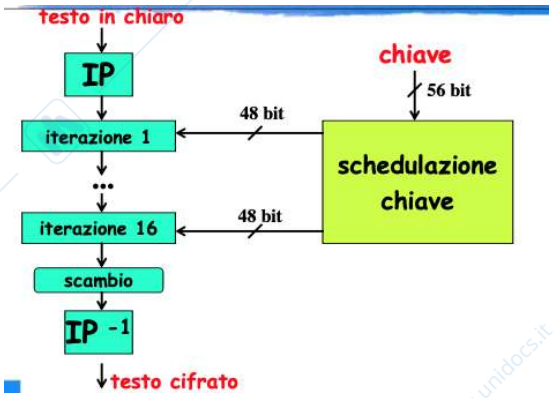


Figura 37: Struttura generale del Data Encryption Standard

La struttura generale dell'algoritmo è mostrata nella Figura 37: ci sono 16 fasi identiche di processo dette round, o cicli. Infatti, uno studio dell'NSA ha stabilito che 16 è il numero minimo di round, necessari per ottenere una sufficiente diffusione. Ci sono inoltre una permutazione iniziale ed una finale dette IP e IP^{-1}, che sono tra di loro inverse (IP "disfa" l'azione di IP^{-1} e viceversa). IP ed IP^{-1} non hanno alcuna importanza per la cifratura ma sono state probabilmente aggiunte per facilitare il caricamento dei blocchi sull'hardware tipico degli anni '70; come effetto collaterale queste fasi causano un rallentamento nelle implementazioni software del DES. La funzione IP cambia di posto i 64 bit del blocco, ed è una funzione invertibile. Infatti, la permutazione finale è l'inverso di quella iniziale.

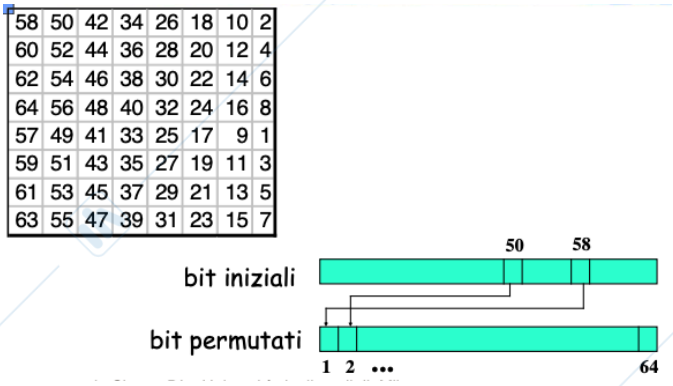


Figura 38: Permutazione iniziale IP

Prima del ciclo principale, il blocco è suddiviso in due metà di 32 bit e processato alternativamente; questo incrocio è detto rete di Feistel. La struttura della rete di Feistel assicura che la cifratura e la decifratura siano processi molto simili - la sola differenza è che le sottochiavi sono applicate nell'ordine inverso nella fase di decifratura. Il resto dell'algoritmo rimane identico. Questo semplifica enormemente l'implementazione, in particolare se effettuata direttamente con un circuito poiché non occorre avere algoritmi separati per cifrare e per decifrare. Il simbolo $\oplus$ denota l'operazione di OR esclusivo (XOR). La funzione Feistel (F-function nello schema) mescola metà del blocco con una parte della chiave. Il risultato della funzione Feistel è poi combinato con l'altra metà del blocco, e le due metà sono scambiate prima del ciclo successivo. Dopo il ciclo finale, le metà non sono scambiate per rendere le fasi di cifratura e decifratura più simili.

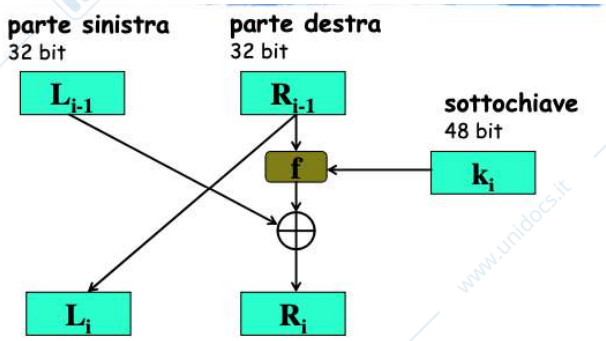


Figura 39: Singola iterazione

La funzione Feistel, rappresentata nella Figura 40, opera su mezzo blocco (32 bit) per volta e consiste di 4 passi:

- Espansione - il mezzo blocco di 32 bit è espanso fino a 48 bit utilizzando la permutazione di espansione contraddistinta con E nello schema, che duplica alcuni bit.
- Miscelazione con la chiave - il risultato è combinato con una sottochiave usando un'operazione di XOR. Sedici sottochiavi di 48 bit, una per ogni ciclo, sono derivate dalla chiave principale usando il gestore della chiave.
- Sostituzione - dopo la miscelazione con la sottochiave, il blocco viene diviso in 8 parti di 6 bit prima del processamento con le S-box o substitution box (scatole di sostituzione). Ognuna delle 8 S-box sostituisce 6 bit in input con 4 bit in output mediante una

trasformazione non lineare effettuata attraverso una tabella. Le S-box forniscono il cuore della sicurezza del DES — senza di esse, la cifratura sarebbe lineare e quindi facilmente violabile.

- Permutazione - infine, i 32 bit risultanti dalle S-box sono riordinati in base alle permutazioni fisse della P-box o permutation box.

L'alternanza di sostituzioni mediante le S-box, le permutazioni con la P-box e le espansioni forniscono la cosiddetta confusione e diffusione.

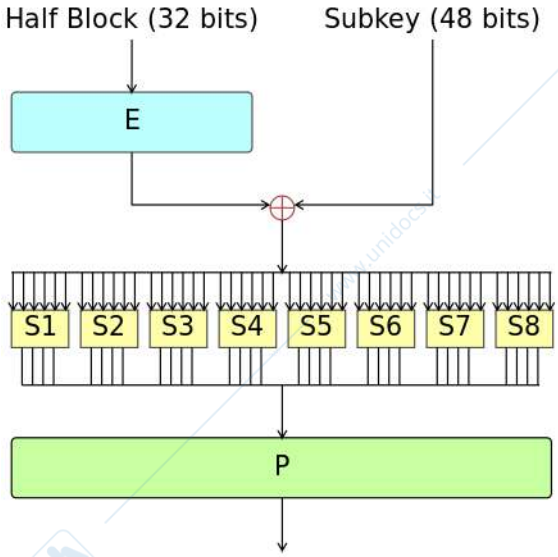


Figura 40: La funzione Feistel (funzione F) nel DES

In crittografia, le S-Box (o Substitution-box, letteralmente "scatole di sostituzione") sono dei componenti base degli algoritmi a chiave simmetrica. Nella crittografia a blocchi le S-Box vengono utilizzate per oscurare relazioni tra il testo in chiaro e il testo cifrato seguendo il principio della confusione enunciato da Shannon. Spesso le S-Box vengono appositamente progettate per resistere alla crittanalisi, come nel caso del DES. La Figura 41 è un esempio di matrice S-Box 4X16 del DES (S5). Questa S-Box ha 6 bit di ingresso e 4 in uscita. Il primo e l'ultimo bit vengono utilizzati per individuare la riga mentre i bit centrali vengono utilizzati per individuare la colonna. Per esempio il numero in ingresso "011011" ha agli estremi i bit "01" e centralmente i bit "1101"che producono come uscita il valore "1001".

S_5		4 bit centrali in ingresso															
		0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
bit esterni	**00**	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1100	0011	1001	1000	0110
	10	0100	0010	0001	1011	1100	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1100	0100	0101	0011

Figura 41: Funzionamento delle S-box

Le proprietà che le S-box soddisfano sono le seguenti:

- Ogni riga è una permutazione degli interi 0,..,15
- Nessuna S-box è una funzione affine o lineare dei suoi input

- Cambiando un solo bit di input ad una S-box variano almeno due bit nell'output
- Per ogni S-box S e per ogni input x a 6 bit: S(x) e S(x $\oplus$ 001100) differiscono in almeno due bit
- Per ogni S-box, per ogni input x e per ogni bit d,g, $S(x) \neq S(x \oplus 11dg00)$
- Per ogni S-box, il numero degli input per i quali il bit di output è 0 è circa uguale al numero degli input per i quali tale bit è 1.

La Figura 42 illustra il gestore della chiave per la cifratura, l'algoritmo che genera le sottochiavi. Inizialmente, vengono selezionati 56 bit della chiave dagli iniziali 64 bit mediante la funzione Permuted Choice 1 (PC-1): i rimanenti 8 bit sono scartati o utilizzati come bit di controllo della parità. Nella permutazione PC-1 i bit in posizione 8, 16, 24, 32, 40, 48, 56 e 64 sono considerati di parità e non vengono considerati. I 56 bit vengono poi suddivisi in 2 metà di 28 bit; ogni metà è poi trattata separatamente. Nei cicli successivi entrambe le metà vengono fatte slittare verso sinistra di 1 o 2 bit (per i round 1, 2, 9, 16 lo shift, cioè lo slittamento, è di 1 bit, per gli altri è di 2) e quindi vengono scelti 48 bit per la sottochiave mediante la funzione Permuted Choice 2 (PC-2): 24 bit dalla metà di sinistra e 24 bit da quella di destra. Il totale degli shift nelle 16 iterazioni è di 28 posizioni.

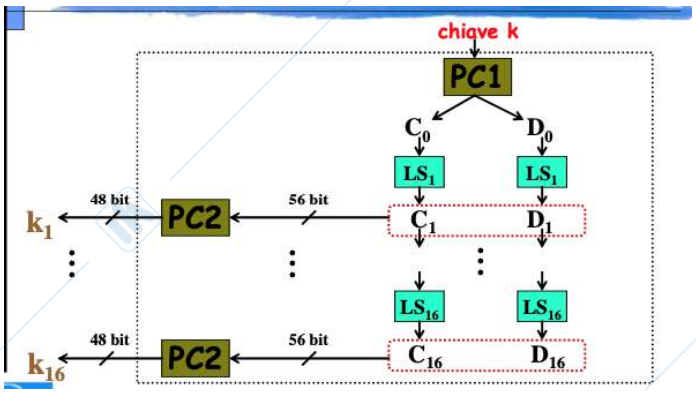


Figura 42: Schedulazione delle chiavi

La decifratura (Figura 43) si effettua mediante lo stesso algoritmo di cifratura, ma le sottochiavi vengono fornite in ordine inverso.

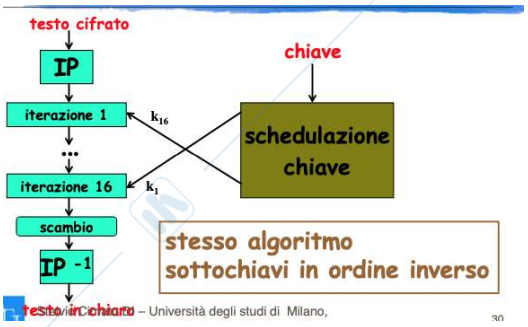


Figura 43: Decifratura

Vediamo un esempio per chiarire il concetto: consideriamo l'ultima operazione di cifratura (a meno della permutazione IP-1) - Figura 44, il primo round dell'operazione di decifratura è rappresentato nella Figura 45 e vediamo che viene sottoposta la chiave 16, cioè l'ultima chiave che è stata sottoposta nella Figura 44.

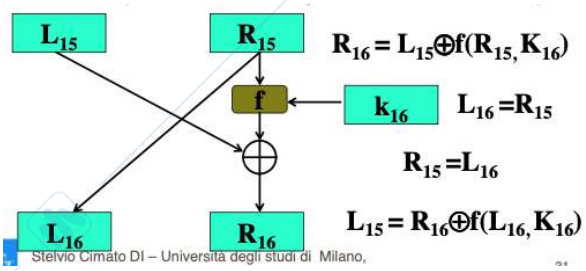


Figura 44: Ultima operazione di cifratura

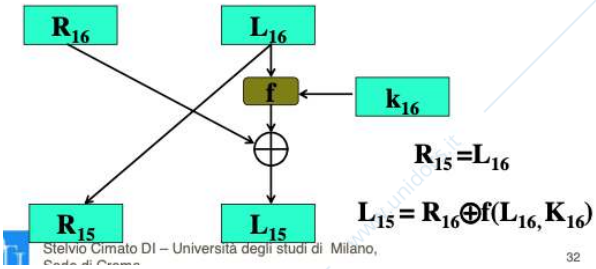


Figura 45: Primo round di decifratura

Il cifrario a blocchi DES ha alcune chiavi particolari denominate rispettivamente "chiavi deboli" e "chiavi semi-deboli". Queste chiavi, se utilizzate, fanno sì che la modalità di cifratura del DES operi nello stesso modo della modalità di decifratura (potenzialmente anche con una chiave differente).

Durante il funzionamento, il gestore della chiave del DES divide i 56 bit della chiave segreta in 16 sotto-chiavi: ogni sotto-chiave è poi usata per ognuno dei 16 passaggi del DES. Le chiavi deboli del DES sono quelle che producono 16 sotto-chiavi identiche. Ciò accade quando i bit della chiave sono (in esadecimale):

- una sequenza alternata di "0" ed "1": 0x0101010101010101
- una sequenza alternata di "F" ed "E": 0xFEFEFEFEFEFEFEFE
- 0xE0E0E0E0F1F1F1F1
- 0x1F1F1F1F0E0E0E0E

Usando le chiavi deboli la funzione PC1 (Permuted Choice 1) del gestore della chiave del DES genera sotto-chiavi composte tutte da "0", tutte da "1" oppure sequenze alternate di "0" ed "1". Dato che tutte le sotto-chiavi risultano identiche e dato che il DES è una rete di Feistel, la funzione di cifratura è auto-invertibile, vale a dire che eseguendo due cifrature si ottiene il testo in chiaro originale.

Il DES ha anche delle chiavi semi-deboli, che fanno generare 2 sole differenti sotto-chiavi, ognuna usata in 8 passaggi dell'algoritmo. Ciò significa che compaiono in coppie K1 e K2 ed hanno la seguente proprietà: $E_{K_1}(E_{K_2}(M)) = M$, dove $E_K(M)$ è l'algoritmo di cifratura che cifra il messaggio M con la chiave K. Ci sono 6 coppie di chiavi semi-deboli:

- 0x011F011F010E010E e 0x1F011F010E010E01
- 0x01E001E001F101F1 e 0xE001E001F101F101
- 0x01FE01FE01FE01FE e 0xFE01FE01FE01FE01

- 0x1FE01FE00EF10EF1 e 0xE01FE01FF10EF10E
- 0x1FFE1FFE0EFE0EFE e 0xFE1FFE1FFE0EFE0E
- 0xE0FEE0FEF1FEF1FE e 0xFEE0FEE0FEF1FEF1

Quindi, le sottochiavi schedulate sono solo due, ognuna usata 8 volte. Ci sono anche 48 possibili chiavi deboli che fanno generare solo 8 sotto-chiavi distinte (invece di 16).

Le chiavi deboli e semi-deboli non sono considerati "difetti fatali" del DES. Ci sono 2^{56} ($7,21 \times 10^{16}$, circa 72 milioni di miliardi) possibili combinazioni di chiavi per il DES, di cui 4 sono chiavi deboli e 12 sono semi-deboli. É nel complesso una piccolissima frazione di tutto lo spazio delle chiavi di cui si può non tenere conto. Se proprio si vuole, è sempre possibile eseguire un controllo per identificare le chiavi deboli e sotto-deboli durante la loro creazione: sono molto poche e facilmente riconoscibili. Resta il fatto che tutte le chiavi del DES possono essere recuperate tramite forza bruta con hardware il cui costo è alla portata di tutti. Infatti, considerando un computer che svolge:

- 1 crittografia DES al microsecondo ci vogliono circa 1142 anni per provare la metà dello spazio delle chiavi 255 ≈ 3,6 ·1016 chiavi
- Ma se considero 1 milione di crittografie DES al microsecondo ci vogliono solo circa 10,01 ore.

Con un computer a 500 Mhz che testa una chiave per ciclo di clock ci vogliono 144.115.188 secondi ≈ 834 giorni ≈ 2 anni e 3 mesi.

Inoltre, il DES possiede la proprietà complementare cioè che $E_K(P) = C \iff E_{\overline{K}}(\overline{P}) = \overline{C}$ dove $\overline{x}$ è il complemento bit a bit di x. E_K denota la cifratura con la chiave K. P e C indicano rispettivamente il testo in chiaro ed il testo cifrato. La proprietà complementare significa che la complessità di un attacco di forza bruta può essere ridotto di un fattore 2 (o un singolo bit) con il presupposto di un attacco con testo scelto (Chosen Plaintext). Quindi, si esaminano 2^{55} chiavi, invece che 2^{56}.

Le discussioni sulla possibile debolezza del DES dovuta alla dimensione troppo ridotta della chiave sono continuate praticamente senza sosta nella comunità crittologica, ma senza approdare a qualcosa di concreto. In effetti il DES, è passato già attraverso ben tre revisioni periodiche, una ogni cinque anni, che ne hanno riconfermato la validità come standard federale. Agli inizi del 1997 l'RSA, decise di verificare se i 56 bit del vecchio DES erano ancora sufficienti contro l'attacco di forza bruta. Il 28 gennaio 1997, l'RSA lanciò una pubblica sfida, estesa a chiunque, avente come oggetto la decifrazione di alcuni brevi testi cifrati col DES e con l'RC5 secondo chiavi di varie lunghezze. I messaggi da decifrare consistevano in una prima parte nota, in modo che per il decifratore era possibile accorgersi di aver trovato la giusta chiave di decifrazione, ed in una seconda parte segreta che costituiva la prova dell'avvenuta decifrazione. I messaggi da decifrare furono pubblicati il 28 gennaio 1997 sul sito web dell'RSA, e subito partì la gara. In palio, vi erano i seguenti premi in denaro:

- $1000 per la forzatura del cifrario RC5 con chiave di 40 bit;
- $5000 per la forzatura del cifrario RC5 con chiave a 48 bit;
- $10000 per la forzatura del cifrario RC5 con chiave maggiore di 48 bit;
- $10000 per la forzatura del DES con chiave fissa di 56 bit.

Lo spazio delle chiavi del DES è formato da 2^{56} elementi, che in notazione decimale equivale a circa 72 milioni di miliardi di combinazioni distinte, per cui un ipotetico computer a 500 MHz che potesse sondare una chiave ad ogni ciclo di clock impiegherebbe 144 milioni di secondi ad

esaminarle tutte; supponendo che sia sufficiente esaminare solo la metà delle chiavi occorrerebbero comunque ben 834 giorni, pari a 2 anni e 3 mesi di lavoro continuativo. Da ciò possiamo concludere che 56 bit sono sufficienti, per cui non basterebbe un supercomputer e nemmeno i calcolatori di un'intera università per effettuare una ricerca esaustiva. Così il 13 marzo 1997 Rocke Verser, un programmatore di Loveland nel Colorado, fece partire una gigantesca iniziativa, chiamata DESCHALL, con la quale mirava a demolire il DES utilizzando la potenza di calcolo, altrimenti inutilizzata, di migliaia di computer sparsi in tutto il mondo. Il progetto prevedeva la partecipazione cooperativa, volontaria e disinteressata, di migliaia di partecipanti, singoli utenti od organizzazioni. Costoro coordinarono i propri sforzi servendosi della rete Internet, suddividendo lo spazio delle chiavi ed esaminandone ciascuno una parte, a seconda delle proprie disponibilità di hardware e di tempo macchina. Verser scrisse un apposito client di ricerca che venne distribuito gratuitamente a tutti i partecipanti. Di giorno in giorno aumentarono coloro che si dedicavano a questa impresa. La sera del 17 giugno un programmatore, Michael K. Sanders, con un computer con le seguenti caratteristiche:

- Cpu Pentium 90 MHz;
- Ram 16 megabytes;
- Velocità (chiavi/secondo) 250.

riuscì a vincere la gara, traducendo in chiaro il seguente messaggio: "Strong cryptography makes the world a safer place".

Per decenni si pensava di costruire un computer specializzato in grado di forzare il DES mediante ricerca esaustiva delle chiavi. Nel 1977 Diffie ed Hellman, due importanti crittologi americani, cominciarono a chiedersi pubblicamente se 56 bit di chiave non fossero davvero pochi, e giunsero alla conclusione che il Governo di una potenza industriale avrebbe potuto tranquillamente investire una ventina di milioni di dollari necessari, a realizzare una macchina superparallela per forzare il DES. Essi suggerirono che si sarebbe potuto costruire un chip a tecnologia VLSI che avrebbe potuto testare 10^6 chiavi al secondo. Una macchina che utilizzava 10^6 chip di questo tipo avrebbe rotto il sistema in un giorno. Il costo di tale macchina era di $20.000.000. Altre macchine (con costi simili) sono state progettate e costruite, ma nonostante ciò il problema non è stato risolto in quanto bisognava "cablare" tale macchina sul canale di comunicazione, avendo, quindi, tutta una serie di problemi di natura diversa. Una di queste macchine fu progettata dall'EFF, che per mostrare che il DES non era più sicuro condusse una ricerca che mostrava che il DES poteva essere rotto con un costo basso (Deep Crack). Il sistema della EFF fu messo alla prova il 15 luglio 1998 quando la RSA diede il via ad una nuova sfida, ottenendo come risultato che in sole 56 ore il Deep Crack ricavò la chiave di cifratura, battendo ogni record. Il costo reale del DES Cracker era stato di circa $2050.000, di cui circa $130.000 di spese per i chip e circa $80.000 per il progetto e il suo sviluppo. L'ultima sfida "DES Challenge III", è stata pubblicata il 18 gennaio 1999. In essa vi era il seguente testo cifrato: 92 2C 68 C4 7A EA DF F2 ricavato da un messaggio sconosciuto di testo in chiaro, che aveva un messaggio fisso e conosciuto come header. La chiave segreta era generata in modo random e poi distribuita fra il software che generava la disputa così che nessuno, neanche gli amministratori della disputa, sapevano quale fosse. La distribuzione del premio per questa challenge è stata la seguente:

- $10.000 con l'impiego di massimo 24 ore;
- $5.000 con l'impiego di massimo 48 ore;
- $1.000 con un massimo di 56 ore;
- niente se si superano le 56 ore.

I vincoli di tempo hanno suggerito ai due vincitori delle dispute precedenti di collaborare, infatti per battere il record precedente di 56 ore, Distributed.Net, una coalizione mondiale di quasi 100.000 entusiasti di computer, ha lavorato con il "Deep Crack" dell' EFF, riuscendo a vincere la sfida dell'RSA, DES Challenge III, in un tempo record di 22 ore e 15 minuti. Partendo come base dalla RSA Data Security Conference & EXPO, una grande conferenza sulla sicurezza dei dati e sulla crittografia, che è stata tenuta a San Jose, California, il "Deep Crack" dell'EFF e i computer di Distributed.Net hanno testato 245 miliardi di chiavi per secondo finché la chiave non è stata trovata.

Un Deep Crack è formato da 24 unità: ogni unità lavora con una chiave e due blocchi. Con la chiave prova a decifrare il crittogramma corrente. Se il testo ottenuto è significativo, prova a decifrare anche l'altro crittogramma. Se il risultato è ancora significativo, la chiave viene segnalata al software di collaborazione, altrimenti passa alla chiave successiva.

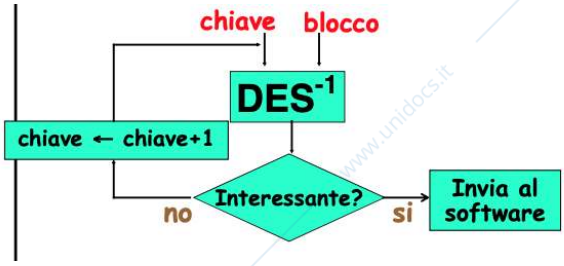


Figura 46: Deep Crack: Unità di ricerca

Supponendo di avere un clock di 40Mhz e di compiere una decifratura in 16 cicli di clock, questo significa che il numero di chiavi provate al secondo $\frac{40.000.000}{16} = 2.500.00$. Vediamo ora delle tecnologie di unità di ricerca a confronto:

- Chip: 24 unità di ricerca, significa provare 60.000.000 chiavi al secondo. Prova tutte le chiavi in 13.900 giorni (circa 38 anni)
- Board (formate da 64 chip): Prova $64 \cdot 60.000.000 = 3.840.000.000$ chiavi al secondo. Prova tutte le chiavi in circa 218 giorni.
- Chassis (12 schede (board)): Prova $12 \cdot 3.840.000.000 = 46.080.000.000$ chiavi al secondo. Prova tutte le chiavi in circa 18 giorni.
- EFF DES Cracker (1998): ricavata la chiave in sole 56 ore.

Riassumendo, alcune caratteristiche dell'algoritmo DES sono:

- Effetto valanga: Due testi che differiscono di un solo bit, cifrati con la stessa chiave differiscono per 34 bit. Due testi uguali cifrati con chiavi diverse per un solo bit differiscono per 35 bit.
- Alcuni criteri progettuali per la funzione F e le S-Box sono:
  - Strict Avalanche: per ogni bit input i invertito, il bit j di output cambia con $p = 1/2$
  - Bit Independence: due bit j e k di output cambiano indipendentemente se cambia il bit di input i
  - Guaranteed Avalanche: se cambia un bit di input, almeno g bit output cambiano, $1 < g < 6$.

DES è stato criticato, come dicevamo prima, per l'utilizzo di una chiave a 56 bit che non era considerata da molti abbastanza sicura. Inoltre sono state mosse critiche alle S-Box, per cui si

pensa possano esistere delle trapdoor. Con 8 iterazioni DES realizza una funzione random, ma, come dicevamo precedentemente, l'NSA ha stabilito che sono necessarie almeno 16 ripetizioni per soddisfare il requisito di confusione.

Infine, alcuni attacchi sofisticati che sono stati eseguiti al sistema crittografico DES sono:

- Crittoanalisi differenziale (Biham e Shamir, 1990): recupera la chiave a partire da 2^{47} coppie (plaintext, ciphertext) di testi scelti. Si tratta solo di un attacco teorico, come trovare 2^{47} testi scelti?
- Crittoanalisi lineare (Matsui, 1993): recupera la chiave a partire da 2^{43} coppie (plaintext, ciphertext) testi in chiaro noti.

I cifrari a blocchi hanno diversi utilizzi, cioè servono per:

- Cifrare
- Costruire stream cipher (cifrari a flusso)
- Costruire funzioni hash
- Costruire generatori pseudo-casuali
- Costruire Message Authentication Code (MAC), piccolo blocco di dati utilizzato per garantire l'autenticazione e integrità di un messaggio digitale, generato secondo un meccanismo di crittografia simmetrica.
- Costruire protocolli di key establishment

4.2.1 Modalità operative

In crittografia la modalità di funzionamento dei cifrari a blocchi è una serie di procedimenti standard di un algoritmo che operi tramite cifratura a blocchi per garantire la sicurezza di testi o messaggi di lunghezza a piacere. Le modalità furono definite inizialmente dal National Institute of Standards and Technology (NIST) nel 1980 con lo standard FIPS 81 intitolato Data Encryption Standard modes of operation affinché l'algoritmo di cifratura DES potesse essere applicato a una varietà di situazioni differenti. Lo standard ebbe successivamente delle estensioni, codificate nella 800-38A. Tali modalità possono essere utilizzate con qualsiasi algoritmo di cifratura simmetrica a blocchi, oltre al DES, tra cui Triple DES e AES.

Alcuni esempi di modalità operative del DES sono:

- Electronic codebook chaining (ECB)
- Cipher block chaining (CBC)
- Cipher feedback (CFB)
- Output feedback (OFB)
- Counter (CTR)

La modalità ECB è la più semplice. Il testo in chiaro viene gestito 64 bit per volta; ognuno dei blocchi di 64 bit viene cifrato con la stessa chiave. Per messaggi più lunghi di 64 bit, si procede suddividendo il messaggio in blocchi di 64 bit, utilizzando, se necessario, bit di riempimento nell'ultimo blocco. Per una data chiave, esiste un unico testo cifrato per ogni blocco di testo in chiaro di 64 bit; in altre parole, se nel messaggio compare più volte lo stesso blocco di 64 bit di testo in chiaro, verrà prodotto sempre lo stesso testo cifrato. Per questo motivo, il metodo ECB è ideale per limitati volumi di dati (ad esempio, la trasmissione di una chiave di cifratura), poiché

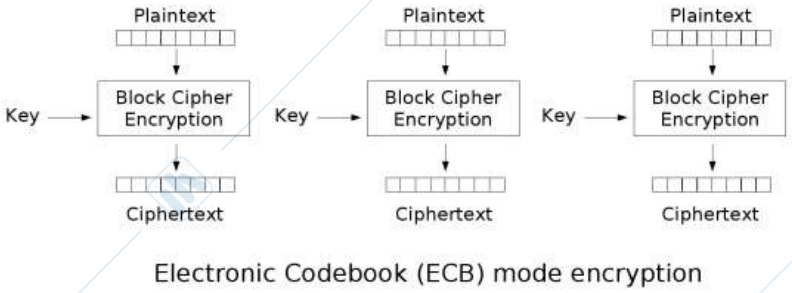


Figura 47: Funzionamento Electronic Codebook Chaining (ECB)

per messaggi più lunghi potrebbe essere insicuro: se si trattasse di dover cifrare un messaggio molto strutturato, l'analisi crittografica potrebbe sfruttarne le regolarità.

L'ECB è il metodo più semplice e veloce, dal momento che eventuali errori non si propagano e che la cifratura è deterministica, ossia allo stesso blocco in chiaro corrisponde lo stesso blocco cifrato. Come dicevamo precedente, non è sicuro per messaggi lunghi.

Per superare i limiti di sicurezza di ECB, è necessario l'utilizzo di una tecnica in cui lo stesso blocco di testo in chiaro, se ripetuto, produce blocchi di testo cifrato differenti. Questo è ciò che accade con la modalità CBC - Cipher Block Chaining, in cui l'input dell'algoritmo di crittografia è il risultato dello XOR tra il blocco di testo in chiaro corrente e il blocco di testo cifrato precedente; per ciascun blocco viene utilizzata la stessa chiave. In fase di decifratura, ciascun blocco di testo cifrato passa attraverso l'algoritmo di decrittografia; il risultato subisce uno XOR con il blocco di testo cifrato precedente, per produrre il blocco di testo in chiaro.

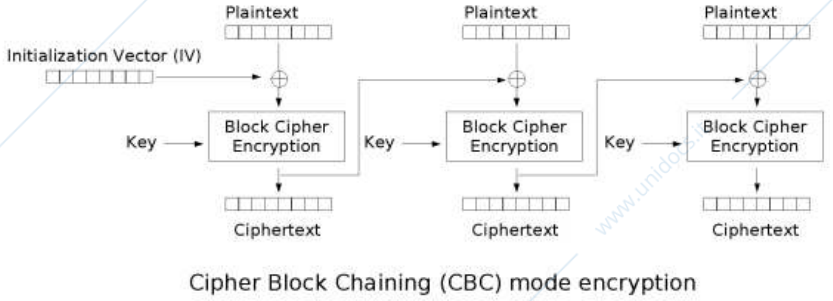


Figura 48: Funzionamento Cipher Block Chaining (CBC)

Per dimostrare la correttezza del meccanismo, si può scrivere:
$C_j = E_k[C_{j-1} \oplus P_j]$
Quindi:
$D_k[C_j] = D_k[E_k(C_{j-1} \oplus P_j)]$
$D_k[C_j] = (C_{j-1} \oplus P_j)$
$C_{j-1} \oplus D_k[C_j] = C_{j-1} \oplus C_{j-1} \oplus P_j = P_j$

Per produrre il primo blocco di testo cifrato, lo XOR viene effettuato tra un vettore di inizializzazione IV e il primo blocco di testo in chiaro. In decifratura, l'IV subisce uno XOR con l'output dell'algoritmo di decrittografia in modo da ottenere nuovamente il primo blocco di testo in chiaro. Il vettore di inizializzazione IV deve essere dunque noto non solo al mittente ma anche al destinatario, che tipicamente lo riceve assieme alla chiave; entrambi i valori vengono cifrati in modalità ECB. IV non deve essere protetto come la chiave: in genere è trasmesso in

chiaro come parte del messaggio $Y_1 = E_k[IV \oplus X_1] \rightarrow X_1 = IV \oplus D_k(Y_1)$. In conclusione, questa modalità è la più appropriata per cifrare messaggi più lunghi di 64 bit. Inoltre, la modalità CBC può essere utilizzata anche per l'autenticazione. Si tratta di un metodo meno veloce dell'ECB e soggetto alla propagazione errori. Inoltre c'è dipendenza tra i blocchi. Nonostante questo risulta più robusto di ECB ed è utilizzato come standard in molti sistemi: SSL, IPSec etc. Lo schema diventa uno schema di cifratura probabilistico: due cifrature dello stesso testo in chiaro appaiono differenti.

Per aumentare il livello di sicurezza ed evitare replay attack, si utilizza un IV casuale per ogni processo di cifratura. Questo permette di produrre un testo cifrato differente anche a parità di testo in chiaro e chiave di crittografia. In questi casi è desiderabile poter evitare di condividere con il destinatario ogni IV generato; questo è possibile, semplicemente premettendo al testo in chiaro un blocco di testo casuale: in questo modo verrà generato un primo blocco cifrato comunque utilizzabile nel processo di cifratura. Implementando questa modalità, in fase di decifratura sarà necessario generare un nuovo IV casuale, quindi scartare il primo blocco di testo in chiaro che verrà prodotto.

La modalità CFB è stata ideata per convertire idealmente una cifratura a blocchi in una cifratura a flusso. La cifratura a flussi non necessita di eseguire riempimenti e può inoltre operare in tempo reale. La lunghezza del testo cifrato corrisponde alla lunghezza del testo in chiato. Nell'operazione di cifratura, l'input della funzione di crittografia è un registro a scorrimento a 64 bit che inizialmente viene impostato con un vettore di inizializzazione IV (Initialization Vector). Gli s bit più significativi (ovvero quelli più a sinistra) dell'output subiscono uno XOR con il primo segmento di testo in chiaro P_1 per produrre la prima unità di testo cifrato C_1. Il contenuto del registro di scorrimento viene fatto scorrere a sinistra di s bit, e negli s bit meno significativi (quelli più a destra) del registro viene inserito C_1. Il processo viene reiterato fino all'esaurimento di tutte le unità di testo in chiaro. Il valore di s può essere scelto a piacimento. Come nella modalità CBC, il testo cifrato è una funzione del testo in chiaro. In questa modalità gli errori vengono propagati.

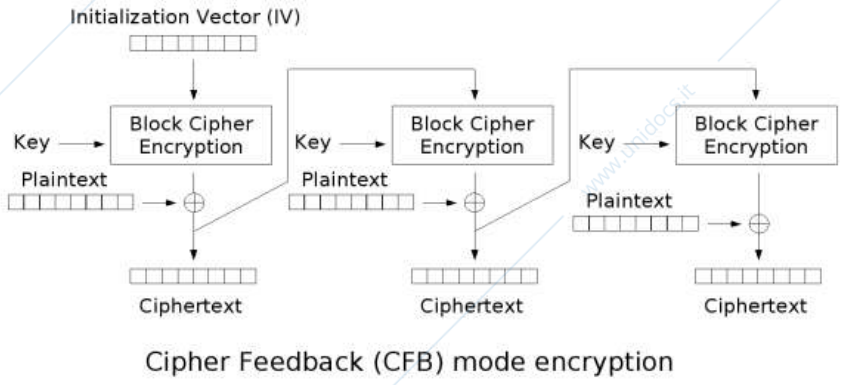


Figura 49: Funzionamento Cipher Feedback (CBC)

All'atto della decifratura, si utilizza il medesimo schema, tranne per il fatto che le unità di testo cifrato ricevute sono sottoposte ad uno XOR con l'output della funzione di crittografia. Non si utilizza dunque la funzione di decrittografia, perché, se $S_S(X)$ sono gli s bit più significativi di X, allora:

$C_1 = P_1 \oplus S_S(E_k(IV))$

Pertanto

$P_1 = C_1 \oplus S_S(E_k(IV))$

e lo stesso ragionamento vale per i passi successivi.

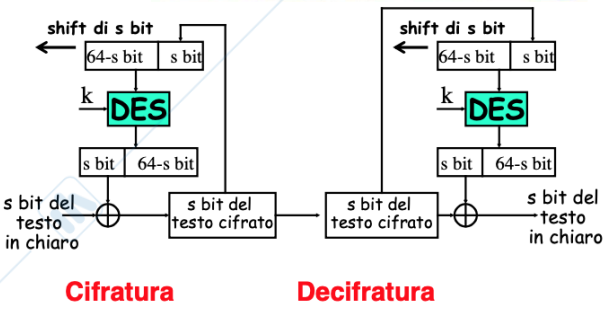


Figura 50: Decifratura Cipher Feedback (CBC)

La modalità OFB è molto simile alla CFB. Il vantaggio della OFB è che non propaga gli errori di trasmissione dei bit. Il suo svantaggio è che è più vulnerabile a un attacco a modifica del flusso dei messaggi (Cambiare un bit in y cambia un bit in x).

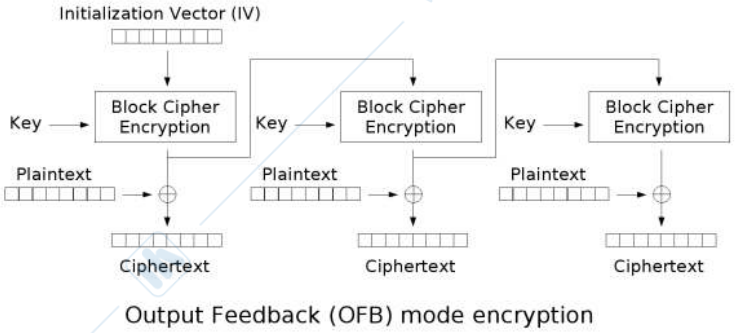


Figura 51: Funzionamento Output Feedback (OFB)

La struttura risulta molto simile al CFB, ma al registro viene mandato l'output del Block Cipher Encryption, invece che il testo cifrato. La Figura 52 raffigura la decifratura di questa modalità.

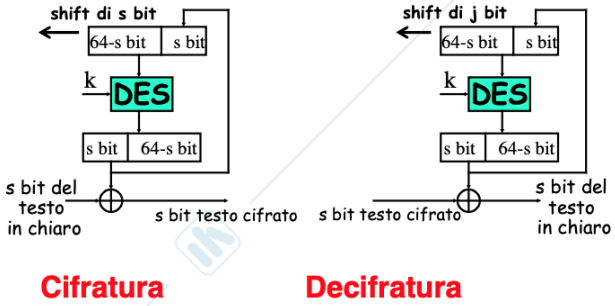


Figura 52: Decifratura Output Feedback (OFB)

La modalità Counter è stata introdotta successivamente alle altre quattro, per poter essere applicata a ATM (Asynchronous Transfer Mode) e a IPsec (IP security). In questa modalità viene utilizzato un contatore corrispondente alle dimensioni del blocco di testo in chiaro. Il requisito essenziale è che il suo valore sia differente per ciascun blocco da cifrare; in genere viene

inizializzato con un determinato valore e poi incrementato di un'unità per ogni blocco successivo (modulo 2b dove b corrisponde alle dimensioni del blocco). Per la cifratura, il contatore viene crittografato e poi si applica uno XOR col blocco di testo in chiaro per produrre il blocco di testo cifrato.

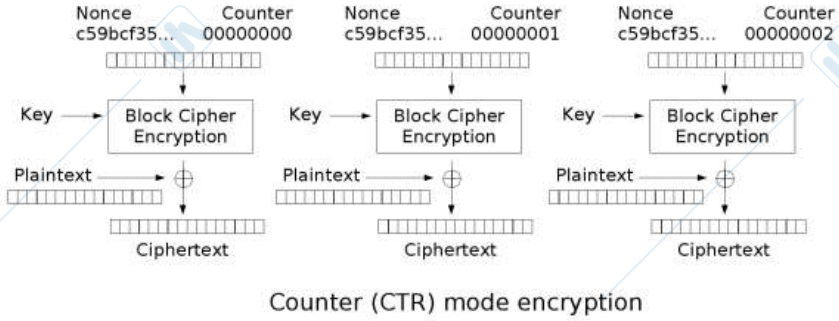


Figura 53: Funzionamento Counter (CTR)

Per la decifratura si utilizza la stessa sequenza di valori del contatore ai quali si applica lo XOR con i blocchi di testo cifrato.

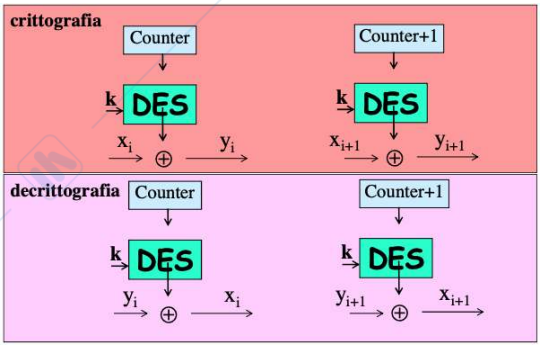


Figura 54: Decifratura Counter (CTR)

I vantaggi relativi all'utilizzo di quest'ultima modalità (CTR) sono:

- Efficienza hardware e software, infatti l'esecuzione può essere eseguita in parallelo su più blocchi.
- Preelaborazione: precalcolo dell'output DES
- Accesso Casuale
- Sicurezza dimostrabile
- Semplice, richiede solo l'algoritmo di crittografia

Le applicazioni tipiche delle varie modalità sono:

- ECB: Trasmissione sicura di singoli valori
- CBC:
  - Trasmissione di carattere generale orientata ai blocchi
  - Autenticazione

- CFB:
  - Trasmissione di carattere generale orientata al flusso di dati
  - Autenticazione
- OFB: Trasmissione orientata al flusso di dati su canali rumorosi
- CTR:
  - Trasmissione di carattere generale orientata ai blocchi
  - Utile per requisiti di alta velocità

Nei dispositivi di memorizzazione basati sui settori si legge/scrive l'intero settore. La dimensione del settore potrebbe non essere divisibile per la dimensione del blocco. Cifrando blocchi identici da memorizzare in settori differenti si deve ottenere un risultato differente. Si deve poter cifrare/decifrare qualsiasi settore o blocco in maniera diretta (random access), per questo motivo è stata introdotta una nuova modalità specifica per AES, ma applicabile a qualsiasi cifrario a blocchi: XTS-AES. É utilizzata per cifrare dati memorizzati in dispositivi basati su settori (ad esempio, dischi rigidi). Supporta settori la cui dimensione non è un multiplo della grandezza del blocco: ad esempio, settore di 520 byte e blocco di 16 byte. XTS-AES è un modo operativo che si basa su AES ed utilizza tre modi operativi:

- Xor-Encrypt Xor (XEX)
- Tweaked CodeBook (TCB)
- CipherText Stealing (CTS)

4.3 Sicurezza dei cifrari moderni

Con sicurezza perfetta intendiamo il fatto che l'avversario non possa ricavare alcuna informazione dall'osservazione del testo cifrato. Mentre, con sicurezza computazionale indentiamo il fatto che se dotato di abbastanza tempo e risorse, l'avversario riesce a rompere il cifrario. Più in dettaglio, la segretezza perfetta garantisce:

- assolutamente nessuna informazione per l'avversario che ascolta
- avversario con potere illimitato

Da un punto di vista pratico uno schema che rivela informazioni con probabilità 2^{-60} ad avversari che possono investire 200 anni di sforzo computazionale é ancora molto buono. Le definizioni di sicurezza che tengono di conto dei limiti computazionali dell'avversario e che ammettono una piccola probabilità di errore vengono dette computazionali (per distinguerle da quelle basate sulla teoria dell'informazione). In questo caso, la sicurezza é soltanto garantita rispetto ad avversari efficienti. Gli avversari possono potenzialmente avere successo, ma soltanto con probabilità molto piccola. La sicurezza computazionale prevede un rilassamento rispetto alla sicurezza perfetta. I cifrari moderni possono essere rotti con abbastanza tempo e computazione ma devono essere praticamente indecifrabili. Se vogliamo cifrare dei messaggi utilizzando una chiave corta, la sicurezza è ottenibile con:

- avversari che operano in un certo intervallo di tempo, quindi non riescono a fare brute force attack
- avversari che riescono a rompere il cifrario con una probabilità molto piccola. Se scelgo una chiave a caso fra le possibili, la probabilità che sia quella giusta deve essere piccola. Quindi, lo spazio delle chiavi deve essere grande.

Quindi, per ottenere cifrari computazionalmente sicuri:

- Sicurezza ottenibile con avversari che operano in un certo intervallo di tempo (tempo polinomiale).
- L'avversario riesce a rompere il cifrario con una probabilità molto piccola.

Ci sono due approcci per sviluppare una teoria significativa:

- Concreto
- Asintotico

4.3.1 Approccio concreto

Limita esplicitamente la probabilità di successo massima di qualsiasi avversario probabilistico che esegue per una specificata quantità di tempo: uno schema (t, ϵ) è sicuro se qualsiasi avversario che ha t tempo a disposizione riesce a rompere lo schema con probabilità ϵ. Ad esempio, un avversario che ci prova per 200 anni deve rompere lo schema con probabilità 10^{-30} oppure un avversario che gira per al più 2^{80} cicli di CPU ha probabilità di rompere lo schema 2^{-64}. I moderni schemi di cifratura simmetrici sono considerati quasi ottimali se, quando la chiave è lunga n, un avversario che ha a disposizione tempo t, rompe lo schema con probabilità $\frac{t}{2^n}$.

La condizione precedente corrisponde ad un attacco di forza bruta. Essenzialmente dice che non ci sono attacchi migliori della ricerca esaustiva. Assumiamo $c = 1$, lunghezza chiave 60 bit ed un PC che esegue 4×10^9 cicli per secondo. La ricerca esaustiva richiede: $\frac{2^{60}}{4\times10^9}$ secondi, che corrispondono a circa 9 anni. Rispetto ad un computer molto più potente che può effettuare 2×10^{16} cicli al secondo: $\frac{2^{60}}{2\times10^{16}}$ secondi, che corrispondono a circa 1 minuto. Con lo stesso computer, per una chiave lunga 80 bit, si ha che: $\frac{2^{80}}{2\times10^{16}}$, che corrispondono a circa 2 anni. La lunghezza raccomandata oggi è n = 128 bit. Per avere un'idea di quanto siano grandi questi numeri...

- 2^{128} è la cardinalità dello spazio delle chiavi
- 2^{58} è il numero di secondi di vita dell'Universo dal Big Bang ad oggi.

E per avere un'idea di quanto siano piccole le probabilità... Se un avversario ha successo in un anno di tempo con probabilità 2^{60}, è molto più probabile che Alice e Bob siano colpiti entrambi da un fulmine nello stesso periodo di tempo. Una scelta prudente è data dai seguenti valori:

- $n = 128$ bit
- $t = 2^{80}$ secondi
- $\epsilon = 2^{-48}$

4.3.2 Approccio asintotico

Nell'ambito della sicurezza computazionale, uno schema è considerato sicuro se un avversario che opera in tempo polinomiale rompe lo schema con probabilità trascurabile. Introduce un parametro di sicurezza n intero, utilizzato per esprimere:

- Il tempo di esecuzione a disposizione dell'avversario
- La probabilità di successo

. Inoltre, n parametrizza sia lo schema che le parti coinvolte: partecipanti, avversari e può essere pensato al momento come la lunghezza della chiave. n è noto agli avversari. Inoltre, i tempi di esecuzione degli avversari, dei partecipanti e la probabilità di successo sono funzioni del parametro di sicurezza n:

- Avversari efficienti: algoritmi probabilistici di tempo polinomiale in n, cioè $t = a \cdot n^c$.
- Partecipanti: algoritmi probabilistici di tempo polinomiale in n
- Probabilità di successo piccola: probabilità più piccola dell'inverso di ogni polinomio in n, cioè n^{-c}.

Usando l'acronimo PPT per probabilistico di tempo polinomiale diremo che uno schema è sicuro se qualsiasi avversario PPT ha successo nella rottura dello schema con al più probabilità trascurabile. Questa nozione è asintotica perché la sicurezza dipende dal comportamento dello schema per valori del parametro n sufficientemente grandi. Ad esempio, disponiamo di uno schema asintoticamente sicuro. Un avversario che esegue per n^3 minuti (tempo polinomiale) ha successo con probabilità $2^{40} \cdot 2^{-n}$ (probabilità trascurabile. i.e., $< \frac{1}{p(n)}$) riesce:

- per $n < 40$, esegue per 40^3 minuti (6 settimane), ed ha successo con probabilità $\epsilon = 1$.
- per $n = 50$, esegue per 50^3 minuti (3 mesi), ed ha successo con probabilità $\epsilon \approx \frac{1}{1000}$.
- per $n = 500$, esegue per 500^3 minuti (200 anni), ed ha successo con probabilità $\epsilon = 2^{-500}$.

Il parametro di sicurezza n è un meccanismo che permette di calibrare la sicurezza dello schema al livello desiderato. Relativamente agli schemi di cifratura, guardare al parametro di sicurezza n come alla lunghezza della chiave corrisponde essenzialmente al fatto che il tempo per una ricerca esaustiva cresce esponenzialmente nella lunghezza della chiave. Nota: incrementando n, incrementa anche il tempo di esecuzione per le parti oneste. Occorre trovare un giusto equilibrio tra:

- tempo richiesto alle parti oneste (più piccolo possibile)
- tempo richiesto all'avversario (più grande possibile)
- probabilità di successo (più piccole possibili)

Presentiamo un esempio: Computer più veloci giocano a favore delle parti oneste. Consideriamo un protocollo crittografico in cui:

- Le parti oneste eseguono per $10^6 \cdot n^2$ cicli di CPU
- L'avversario esegue per $10^8 \cdot n^4$ cicli di CPU, ed ha successo con probabilità $2^{20} \cdot 2^{-n}$

Per computer a 1GHz, ed n = 80:

- Le parti oneste eseguono per $10^6 \cdot 50^2$ cicli di CPU, 2.5 secondi
- L'avversario esegue per $10^8 \cdot 50^4$ cicli di CPU, circa 1 settimana, ed ha successo con probabilità 2^{-30}

Per computer a 16GHz, ed n = 100 (incrementando la lunghezza della chiave):

- Le parti oneste eseguono per $10^6 \cdot 100^2$ cicli di CPU a $16^6 \cdot 10^9$ cicli/secondo, per 0.625 secondi.
- L'avversario esegue per $10^8 \cdot 100^4$ cicli di CPU, 16 settimane, ed ha successo con probabilità 2^{-80}.

Pertanto, computer più veloci comportano un lavoro degli avversari più difficile.

Abbiamo definito la sicurezza di un cifrario data dalla impossibilità che l'avversario abbia alcuna informazione sul plaintext solo osservando il ciphertext. Equivalentemente possiamo definirla sull'*indistinguibilità* delle cifrature secondo il seguente esperimento: L'avversario ha due messaggi m_0 e m_1 della stessa lunghezza. L'esperimento consiste nel:

1. Scegliere una chiave k
2. Scegliere un bit casuale b
3. Generare la cifratura $c = E_k(m_b)$

L'avversario osservando solo $c = E_k(m_b)$ emette un bit b'. L'avversario ha successo se $b' = b$. Uno schema è sicuro se la probabilità di successo nell'esperimento precedente di un avversario polinomiale è negligibilmente più grande di $\frac{1}{2}$ (nota che A ha sempre prob $\frac{1}{2}$ basta tirare a caso).

Una stringa è pseudocasuale se la sua distribuzione è indistinguibile da una stringa scelta con distribuzione uniforme. Nell'informatica teorica e nella crittografia, un generatore pseudocasuale (PRG) per una classe di test statistici è un algoritmo deterministico che mappa un seme casuale a una stringa pseudocasuale più lunga in modo tale che nessun test statistico nella classe possa distinguere tra l'output del generatore e la distribuzione uniforme. Il seme casuale è in genere una breve stringa binaria disegnata dalla distribuzione uniforme. In poche parole, un pseudorandom generator è un algoritmo deterministico che avendo in input una stringa random, ne emette una di maggior lunghezza. Aumenta la pseudorandomness, con input una stringa lunga n, ne genera una lunga $p(n)$ (polinomio $p(n) > n$). Sia $mathcalA = \{A : \{0,1\}^n \to \{0,1\}^*\}$ una classe di funzioni. Queste funzioni sono i test statistici che il generatore pseudocasuale cercherà di ingannare e di solito sono algoritmi. A volte i test statistici sono anche chiamati avversari o distinguitori. Una funzione $G : \{0,1\}^\ell \to \{0,1\}^n$ con $\ell \leq n$ è un generatore pseudocasuale contro $\mathcal{A}$ con bias ϵ se, per ogni A in $\mathcal{A}$, la distanza statistica tra le distribuzioni $A(G(U_\ell))$ A e $A(U_n)$ è al massimo ϵ, dove U_k è la distribuzione uniforme su $\{0,1\}^k$. I generatori pseudocasuali hanno numerose applicazioni nella crittografia. Ad esempio, i generatori pseudocasuali forniscono un analogo efficiente dei pad one-time. É noto che per crittografare un messaggio m in modo che il testo cifrato non fornisca informazioni sul testo in chiaro, la chiave k utilizzata deve essere casuale su stringhe di lunghezza $|m|$. La crittografia perfettamente sicura è molto costosa in termini di lunghezza della chiave. La lunghezza della chiave può essere significativamente ridotta usando un generatore pseudocasuale, se la sicurezza perfetta è sostituita dalla sicurezza computazionale.

Lo schema, raffigurato nella Figura 55, usa uno pseudorandom generator: poiché la stringa prodotta è random (per un avversario PPT[1]) ottengo uno schema computazionalmente sicuro. La sicurezza posso esprimerla come indistinguibilità ottenuta nell'esperimento precedente, cioè l'avversario fallisce. Nell'esempio, in Figura 55, la funzione di cifratura è $C = G(k) \oplus m$, dove $G(k)$ rappresenta il generatore pseudocasuale, un algoritmo deterministico, ossia che non utilizzi al proprio interno nessuna forma di scelta casuale. G deve essere efficientemente calcolabile, altrimenti l'intero schema che stiamo costruendo diventerebbe problematico dal punto di vista computazionale. G deve permettere di espandere la stringa in input, facendola diventare sempre più lunga, come dicevamo precedentemente. G deve essere impredicibile: non è possibile che dati i primi bit di $G(k)$, un algoritmo restituisca i successivi bit. Inoltre, bisogna fare in modo che la stringa $G(k)$ abbia tutte le proprietà che ci aspettiamo da una chiave. Queste sono le proprietà (riassunte) di un generatore pseudocasuale. La decifratura è calcolata mediante: $m = G(k) \oplus c$.

[1] Probabilistic Polynomial Time

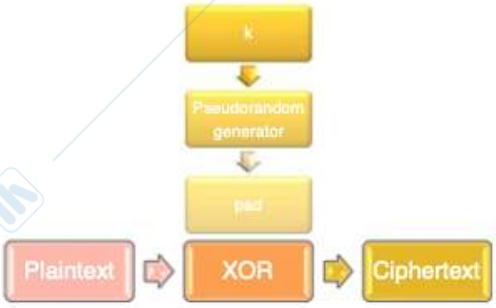


Figura 55: Secure Encryption Scheme

Un problema che sorge dall'utilizzo di OTP (One-Time Pad) è che non si può usare lo stesso flusso di chiavi. Ipotizziamo di avere $C_1 = G(k) \oplus m_1$ e $C_2 = G(k) \oplus m_2$, avremo che $C_1 \oplus C_2 = m_1 \oplus m_2$. Lo XOR di due testi cifrati è uguale allo XOR dei rispettivi testi in chiaro. Permette ad esempio di identificare immediatamente porzioni di testo uguali (in quanto il risultato dello XOR è una sequenza di 0). Quindi, la ridondanza nel testo fa in modo che conoscendo $m_1 \oplus m_2$ si può stabilire m_1 e m_2. Alcuni esempi reali sono:

- Progetto Venona (1940-1945): fu un programma di collaborazione tra le agenzie di spionaggio degli Stati Uniti d'America con il MI5 e il GCHQ del Regno Unito le cui attività si svolsero dal 1943 al 1980. Scopo era la decodifica dei i messaggi spediti da diverse agenzie di spionaggio sovietiche.
- MS-PPTP in Windows NT: PPTP è la sigla di Point to Point Tunneling Protocol, e si riferisce ad un protocollo di rete che attraverso la cifratura dei dati rende sicure la trasmissione in una rete privata, su una rete pubblica (VPN).
- WEP: è parte dello standard IEEE 802.11 (ratificato nel 1999) e in particolare è quella parte dello standard che specifica il protocollo di rete utilizzato per rendere sicure le trasmissioni radio delle reti Wi-Fi. WEP è stato progettato per fornire una sicurezza comparabile a quelle delle normali reti LAN cablate.

Esistono tuttavia altri attacchi possibili a OTP, ad esempio, sfruttando la sua malleabilità. Un cifrario è malleabile se è possibile alterare il testo cifrato in modo da ottenere un risultato specifico in seguito alla decifratura (attacco attivo). Quindi, l'integrità non è garantita, dal mento che avvengono modifiche non consentite al ciphertext con impatto sul plaintext. La cifratura è sempre data da $C = m \oplus G(k)$, ma un attaccante intercetta C e lo sostituisce con $C' = C \oplus N$. Il risultato della decifratura a quel punto sarà $M' = C' \oplus K = M \oplus K \oplus N \oplus K = M \oplus N$. Se l'attaccante conosce M, può scegliere un opportuno N tale che, in fase di decifratura, il testo in chiaro originario M sia sostituito da un testo arbitrario M', scelto dall'attaccante.

Complichiamo lo scenario: messaggi multipli. Consideriamo la costruzione precedente e il seguente esperimento: Input $M_0 = (0^n, 0^n), M_1 = (0^n, 1^n)$. Sia $C = (c^1, c^2)$ il cifrato corrispondente. L'esperimento $c^i = E_k(m_b^i)$ fallisce: se $c^1 = c^2 b = 0$ oppure se $c^1 \neq c^2 b = 1$. C'è bisogno di non determinismo, quindi introduco una quantità di randomness. Cifrature di stessi messaggi in input danno output diversi. La costruzione naive vista in precedenza non è sicura.

Un attacco a testo in chiaro scelto (CPA) è un modello di attacco per la crittoanalisi che presume che l'attaccante possa ottenere i cifrati per arbitraggi in chiaro. L'obiettivo dell'attacco è ottenere informazioni che riducano la sicurezza dello schema di crittografia. I cifrari moderni mirano a fornire sicurezza semantica, nota anche come indistinguibilità del testo cifrato in caso di attacco con testo in chiaro prescelto, e pertanto sono generalmente immuni da attacchi in

testo in chiaro scelti se implementati correttamente. Un attacco di Chosen Plaintext Attack (CPA) viene eseguito come segue:

1. L'attaccante può scegliere n caratteri in chiaro. (Questo parametro n è specificato come parte del modello di attacco, può o non può essere limitato.). Quindi scegli adattativamente il plaintext.

2. L'aggressore invia quindi questi n caratteri in chiaro all'oracolo della crittografia. $Enc(x)$ è come un oracolo al quale fare tutte le domande possibili.

3. L'oracolo della crittografia crittograferà quindi i testi in chiaro dell'attaccante e li rimanderà all'attaccante.

4. L'attaccante riceve n cifrato dall'oracolo, in modo tale che l'attaccante sappia quale testo cifrato corrisponde a ciascun testo in chiaro.

5. Basato sulle coppie di testo in chiaro e testo cifrato, l'attaccante può tentare di estrarre la chiave utilizzata dall'oracolo per codificare i testi in chiaro. Poiché l'attaccante in questo tipo di attacco è libero di creare il testo in chiaro per soddisfare le sue esigenze, la complessità dell'attacco può essere ridotta.

Considera la seguente estensione della situazione precedente:

- L'avversario genera due caratteri in chiaro m_0 e m_1.
- Un bit b viene scelto in modo uniforme a caso $b \leftarrow \{0,1\}$ e $c = E_k(m_b)$.
- A fa tante richieste all'oracolo $E_k(...)$, per determinare se $b = 0$ o $b = 1$.

Una cipher ha crittografie indistinguibili sotto un CPA se dopo aver eseguito l'esperimento sopra con $n = 1$ l'avversario non può indovinare correttamente (b = 0/1) con probabilità non trascurabilmente migliore di $\frac{1}{2}$. Un aneddoto dell'utilizzo di CPA nella II guerra mondiale è il seguente: i crittanalisti della Marina americana scoprirono che il Giappone stava pianificando di attaccare un luogo chiamato "AF". Credevano che "AF" potesse essere Midway Island, perché altre località nelle Isole Hawaiian avevano parole in codice che iniziavano con "A". Per provare la loro ipotesi che "AF" corrispondesse a "Midway Island", hanno chiesto alle forze statunitensi a Midway di inviare un messaggio in chiaro sulle scarse forniture. I giapponesi intercettarono il messaggio e riferirono immediatamente ai loro superiori che "AF" era a corto di acqua, confermando l'ipotesi della Marina e permettendo loro di posizionare la loro forza per vincere la battaglia.

Se lo schema di cifratura ha sicurezza CPA, allora ho sicurezza CPA anche nel caso di cifrature multiple. Per costruire uno schema CPA-sicuro dobbiamo utilizzare funzioni pseudorandom (PRF). Generalizzano la nozione di generatore pseudocasuale: i PRG producono stringhe che sembrano casuali, mentre le PRF sono funzioni che sembrano casuali. Non ha senso parlare di una funzione fissata, dobbiamo considerare una distribuzione di funzioni. Le funzioni parametrizzate da una chiave (keyed function) inducono naturalmente una distribuzione di funzioni. Una funzione parametrizzata da una chiave $F : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$ é una funzione con due input, in cui il primo é chiamato chiave e viene denotato con k. F è efficiente se esiste un algoritmo di tempo polinomiale per calcolare $F(k,x)$, dati k e x. In usi tipici, k viene scelto e fissato. Per cui $F_k(x) = F(k,x)$ consiste in $F_k : \{0,1\}^* \rightarrow \{0,1\}^*$ (funzione a una sola variabile). Quindi, per una fissata chiave k, F_k è indistinguibile da una qualsiasi f che opera da $\{0,1\}^* \rightarrow \{0,1\}^*$. Quindi, F é pseudocasuale se F_k , per k scelta uniformemente a caso, é indistinguibile da una funzione scelta uniformemente a caso dall'insieme di tutte le funzioni aventi lo stesso dominio e lo stesso codominio. Vediamolo in termini matematici: sia $F : K \times X \rightarrow Y$ una PRF, dove:

- $Funs[X, Y]$ rappresenta tutte le funzioni da X a Y.
- Si ha che: $S_F = \{F(k, *) s.t. k \in K\} \subseteq Funs[X, Y]$.

Quindi, una PRF è sicura se una F random in $Funs[X, Y]$ è indistinguibile da una random function in S_F.

Il concetto che sta alla base di Pseudorandom permutation (PRP) è che non succede mai che per x e y diversi, $f(x) = f(y)$, o meglio una funzione uniforme "appare identica" a una permutazione uniforme, a meno che il distinguisher D non trovi x ed y tali che $f(x) = f(y)$. La probabilitá di un tale evento é trascurabile utilizzando un numero polinomiale di query. Le Pseudo Random Function - PRF ($F : K \times X \to Y$) si possono considerare efficienti, se esiste un algoritmo efficiente per calcolare $F(k, x)$. Le Pseudo Random Permutation - PRP ($E : K \times K \to X$) sono efficienti se esiste un algoritmo efficiente per calcolare $E(k, x)$. La funzione E è invertibile e biettiva, conoscendo k, quindi esiste un algoritmo inverso chiamato $D(k, y)$.

Sia $F : K \times \{0,1\}^n \to \{0,1\}^n$ una PRF sicura, allora $G : K \to \{0,1\}^{nt}$ è un PRG (generatore pseudocasuale) sicuro: $G(k) = F(k, 0)||F(k, 1)||...||F(k, t-1)$. Per essere sicuro, è necessario che $F(k, *)$ indistinguibile dalla funzione random $f(*)$.

La costruzione di schemi di cifratura sicuri CPA (Figura 56) si basa sull'utilizzo di funzioni pseudocasuali. Si ottiene una cifratura probabilistica, ossia dato uno stesso input, otterremo output diversi, dipendenti dal messaggio iniziale. Operativamente, applichiamo F_k ad una stringa casuale r per produrre un "pad" pseudocasuale. Cifriamo calcolando l'xor tra il pad ed il messaggio m.

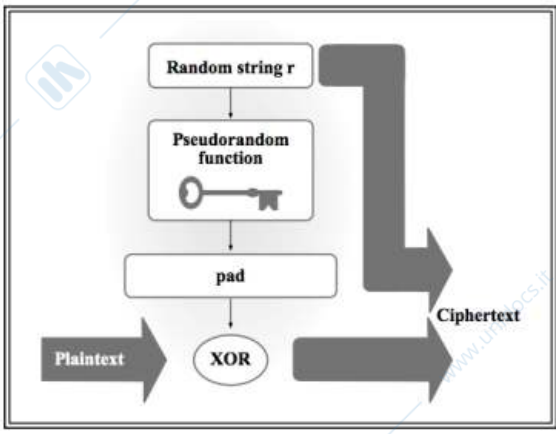


Figura 56: Schemi CPA

Dato in input un $m \in \{0,1\}^n$, si sceglie a caso $r \in \{0,1\}^n$. La cifratura è data da: $c =< r, F_k(r) \oplus m >$. Mentre, per quanto riguarda la decifratura, avendo in input un messaggio cifrato $c =< r, s >$, si ottiene il messaggio in chiaro, attraverso la funzione di decifratura: $m = F_k(r) \oplus s$. I cifrari a blocchi sono pseudorandom permutations. Servono quindi per creare schemi di cifratura sicuri. Possono essere messi al posto delle pseudorandom permutation nella costruzione precedente.

Abbiamo visto uno schema di cifratura CPA-sicuro basato su PRF (cifrario a blocchi), dove la lunghezza del cifrato è doppia rispetto alla lunghezza del messaggio Le modalità operative per i cifrari a blocchi sono metodi per cifrare messaggi di lunghezza arbitraria con cifrati "corti". Sia F un cifrario a blocchi con lunghezza di blocco n. Sia $m = m_1 m_2 ... m_n$, dove $m_i \in \{0,1\}^n$ per ogni $i = 1, ..., n$. Se necessario l'ultimo blocco viene "completato" (padded) in modo tale che $|m_n| = r$. La modalità operativa più semplice è l'Electronic Code Book (ECB) mode: $c :=< F_k(m_1), F_k(m_2), ..., F_k(m_n) >$. ECB é deterministico, quindi non può dare sicurezza

CPA. Non ha neanche cifrature indistinguibili rispetto ad un eavesdropper. Se un blocco si ripete nel messaggio in chiaro, allora il blocco si ripeterà nel cifrato. É facile allora distinguere tra due messaggi: Adv sceglie m_0 con un blocco ripetuto ed m_1 con blocchi tutti distinti e analizza i blocchi del cifrato. In conclusione: ECB non dovrebbe mai essere usato. CBC è una modalità probabilistica Se F é una permutazione pseudocasuale, allora la modalità CBC è CPA-sicura. Un inconveniente è che la cifratura deve essere effettuata sequenzialmente. Se il vettore di inizializzazione IV non viene scelto uniformemente in $\{0,1\}^n$, ma semplicemente in modo che i valori siano distinti, allora la variante non è più sicura. Le altre modalità operative (CBC senza variante, OFB e CTR) possono essere formalmente provate CPA sicure. Tutte le modalità presentate non sicure contro Chosen-ciphertext attack o CCA.

4.4 Crittoanalisi di DES

Di fatto, il modo più efficiente di violare il DES è un attacco a forza bruta: provare ogni possibile chiave, una dopo l'altra, dal momento che la chiave usa solo 56 bit. Ci sono due attacchi "a livello teorico" applicabili a ogni cifrario a blocchi (analisi crittografica):

- Crittoanalisi differenziale: "scoperta" da Biham e Shamir (circa nel 1990).
- Crittoanalisi lineare: scoperta da Matsui nel 1993.

4.4.1 Attacco a forza bruta

Per ogni algoritmo di cifratura, il metodo più semplice di attacco è quello con forza bruta, ovvero provare tutte le possibili chiavi a partire dall'inversione dell'algoritmo di cifratura che è quasi sempre noto. La lunghezza n della chiave determina il numero di chiavi possibili (2^n) e quindi la fattibilità dell'attacco a livello computazionale. Quindi, applicandolo al sistema DES, calcolo le 2^{56} chiavi possibili. Se si tratta di un attacco knwon plaintext, ossia un tipo di attacco crittanalitico dove l'attaccante è in possesso sia del testo in chiaro sia del testo cifrato. In media ne provo 2^{55}.

In informatica il compromesso time-memory è una situazione dove l'utilizzo della memoria può essere ridotto al prezzo di un rallentamento della velocità di esecuzione del programma o, viceversa, il tempo di calcolo può essere ridotto al prezzo di un incremento dell'utilizzo della memoria. Nel 1980 Martin Hellman primo ha proposto di utilizzare un compromesso tempo-memoria per crittoanalisi. La situazione più comune è un algoritmo che coinvolge una tabella di ricerca: un'implementazione può includere l'intera tabella, che riduce il tempo di elaborazione, ma aumenta la quantità di memoria necessaria, oppure può calcolare voci della tabella quando necessario, aumentando il tempo di calcolo, ma riducendo i requisiti di memoria. Se vengono calcolati tutti i dati:

- Per un testo in chiaro x calcolo $y_k = DES_k(x)$ per ogni k
- Mantengo una tabella (y_k, k)
- Quando ottengo y, cifrando x con una chiave sconosciuta, ricerco y nella tabella per scoprire k.

Il tempo di ricerca è logaritmico (o costante), mentre il tempo di precomputazione è 2^{56}. Lo spazio di memoria per la tabella è sempre 2^{56}.

L'idea è l'utilizzo di una funzione di riduzione del tipo: $F : \{0,1\}^{64} \to \{0,1\}^{56}s$. Sia x un input a 64 bit e $X(1,0)$ una stringa di 56 bit: calcolo $y = DES_{X(1,0)}(x)$. Inoltre, calcolo: $X(1,1) = F(y) = F(DES_{X(1,0)}(x))$. Iterando, ottengo una sequenza di stringhe di 56 bit: $X(1,0) \to X(1,1) \to X(1,2) \to ... \to X(1,t)$. Creo una tabella con m righe e t colonne per $X(i,j) = F(DES_{X(i,j-1)}(x))$ - Figura 57.

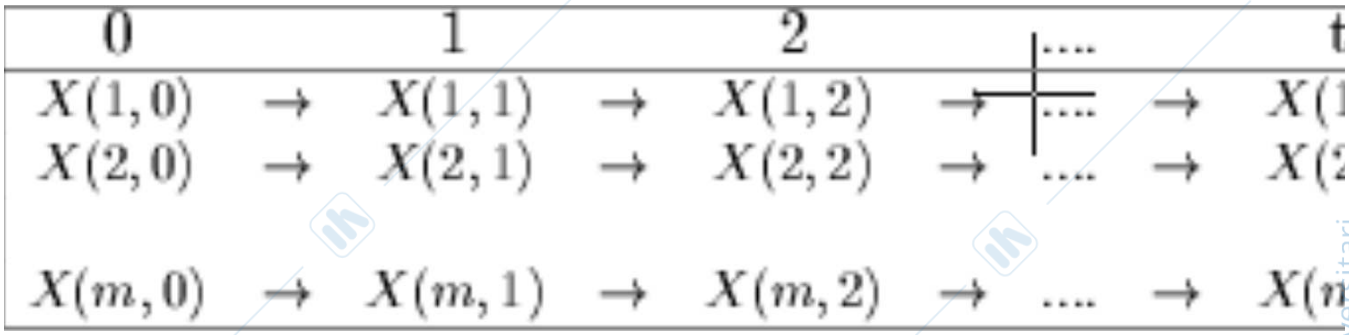


Figura 57: Time-space trade-off

Quando ho un y cifrato (64 bit) calcolo la funzione di riduzione $F(y)$, che mi permette di ottenere una stringa a 56 bit:

- Controllo l'ultima colonna della tabella. Se per un i, $X(i,t) = F(y)$ controllo che $DES_{X(i,t-1)}(x) = y$. Da notare che F non è iniettiva, ma ci sono $2^8 = 256$ preimmagini per ogni elemento di 56 bit.
- Altrimenti, controllo la $t-1$ colonna, successivamente la $t-2$, e cosi via. Devo sempre verificare se $DES_{X(i,t-j)}(x) = y$.

L'idea è quella di mantenere solo la colonna 0 e t:

- Se $F(y)$ è in t, e $F(y) = X(i,t)$: la chiave candidata è $X(i,t-1)$. Dal valore $X(i,0)$ posso calcolarla e provare a verificarla.
- Altrimenti cerco $F(y)$ in $t-1$, ma invece di calcolare la colonna $t-1$, cerco in t $F(DES_{F(y)}(x))$. Se $F(y)$ è in $t-1$, allora $F(DES_{F(y)}(x))$ è in t. Se ho successo per un j, $X(j,t-1)$, calcolo la chiave candidata $X(j,t-2)$ a partire da $X(j,0)$.

Itero il procedimento per tutte le t colonne, calcolando $y_{j+1} = F(DES_{y_j}(x))$. Posso mantenere T tabelle. Consideriamo i valori di m, t e T. Se $m * t^2 N = 2^{56}$, allora la chiave è nella tabella con $P = \frac{0,8*m*t}{/} N$. La complessità in termini di spazio è $T * 2 * N^{1/3} * 56 = 112 * N^{2/3}$. Mentre il tempo di precomputazione consiste in $N^{1/3} * N^{1/3} * N^{1/3} = N$. Il tempo di ricerca è uguale a $N^{2/3}$.

4.4.2 Crittoanalisi differenziale

Le tecniche di crittoanalisi lineare e differenziale sono tecniche generali di crittoanalisi che hanno successo con molti cifrari, anche se con DES non sono attacchi pratici. La crittanalisi differenziale è una forma generale di crittanalisi, applicabile principalmente ai cifrari a blocchi ma anche ai cifrari a flusso ed alle funzioni crittografiche di hash. Nel senso più ampio del termine, la crittanalisi differenziale di una funzione crittografica è lo studio di come le differenze nei dati forniti in ingresso alla funzione possono incidere sulle differenze risultanti in uscita dalla stessa. Nel caso di un cifrario a blocchi, si riferisce ad un insieme di tecniche per tracciare le differenze attraverso la rete delle trasformazioni che l'algoritmo opera durante la sua esecuzione, scoprendo dove il cifrario mostra comportamenti non casuali, permettendo di sfruttare tali proprietà per recuperare la chiave segreta. La scoperta della crittanalisi differenziale è generalmente attribuita ad Eli Biham e Adi Shamir alla fine degli anni ottanta, che pubblicarono un numero di attacchi contro diversi cifrari a blocchi e funzioni di hash, inclusa una debolezza teorica del DES. La crittanalisi differenziale è di solito un attacco con testo in chiaro scelto (Chosen Plaintext), vale a dire che l'attaccante deve essere in grado di ottenere i messaggi cifrati relativi a testi

in chiaro di sua scelta. Lo schema può crittanalizzare con successo il DES con l'uso di 2^{47} testi in chiaro scelti. Ci sono, peraltro, delle estensioni che permetterebbero l'uso di testo in chiaro noto oppure un attacco con solo testo cifrato. Come dicevamo prima, questa tipologia di crittoanalisi può essere solo teorica, dal momento che è difficile trovare 2^{47} testi scelti. In realtà, per altri tipi di algoritmi di cifratura, risulta efficace con un numero basso di iterazioni: con 8 iterazioni riesce in pochi minuti. Necessita di 2^{14} coppie di testi scelti. Non è pratico con 16 o più iterazione, perchè per recuperare la chiave necessita di 2^{47} coppie di testi scelti. 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 l'OR esclusivo (XOR). L'attaccante può calcolare le differenze dei corrispondenti testi cifrati, sperando di intercettare dei motivi statistici nella loro distribuzione. Quindi, si basa sul confronto tra lo xor bit a bit di due testi in chiaro e dei corrispondenti testi cifrati, per analizzare le differenze con lo scopo di trovare la chiave usata. Nel caso di una trasformazione lineare: se $C_1 = P_1 \oplus K$ e $C_2 = P_2 \oplus K$, allora $C_1 \oplus C_2 = P_1 \oplus K \oplus P_2 \oplus K$. Dall'ultima formula deriva che $C_1 \oplus C_2 = P_1 \oplus P_2$, dalla quale possiamo ricavare P_2: $P_2 = C_1 \oplus C_2 \oplus P_1$. Questo ragionamento non vale per il DES, dal momento che esso non applica una trasformazione lineare. Nell'attacco base ci si attende che una particolare differenza nei testi cifrati sia abbastanza frequente; in questo modo, il cifrario può essere distinto dal caso. Varianti dell'attacco più sofisticate permettono di recuperare la chiave più velocemente della ricerca esaustiva di tutte le possibili combinazioni. Si considerano due coppie di testi in chiaro/cifrati: (x_1, y_1) e (x_2, y_2) con $y_1 = DES_k(x_1)$ e $y_2 = DES_k(x_2)$, l'attaccante può calcolare le differenze dei corrispondenti testi cifrati, sperando di intercettare dei motivi statistici nella loro distribuzione, quindi si ragiona sulle differenze: $x' = x_1 \oplus x_2$ e $y' = y_1 \oplus y_2$. La coppia risultante delle differenze è detta una differenziale. Le loro proprietà statistiche (non-linearità) dipendono dalla natura delle S-box usate per la cifratura, così che l'attaccante analizza le differenziali (input e output) per ciascuna delle S-box S_i: $\delta = B \oplus B*$ e $\Delta = S(B) \oplus S(B*)$. Per una coppia di input $(B, B*)$ la S-box genera una coppia di output $(S(B), S(B*))$. La coppia $(B, S(B))(B*, S(B*))$ è caratterizzata da:

- differenza input $\delta = B \oplus B*$
- differenza output $\Delta = S(B) \oplus S(B*)$

Posso calcolare S^{δ}_{Δ} (2^6 righe $\times$ 2^4 colonne) - Figura 58, la quale rappresenta le uscite delle S-box.

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	0110	0111	1111
00	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
01	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
10	4	1	7	8	13	6	2	11	15	12	9	7	3	10	5	0
11	15	12	10	2	4	9	1	7	5	11	3	14	10	0	6	13

Figura 58: Tabella profilo XOR S_1

Per ogni valore di B definisco l'insieme $\Delta(B')$ come l'insieme delle coppie ordinate $(B, B*)$ tali che: $B' = B \oplus B*$. Si noti che $|\Delta(B')| = 2^6 = 64$. La non uniformità nella distribuzione servirà a portare l'attacco. Considero un certo valore $\delta = 110100$. Per ogni coppia $(B, B*)$ tale che $\delta = B \oplus B* = 110100$ (es. 000000, 110100) calcolo l'output XOR. Quindi: $S(000000) = 1110$ e $S(110100) = 1011$, posso ottenere questi risultati andando a consultare la tabella nella Figura 58. Infine, calcolo $\Delta = 1110 \oplus 1011 = 0101$. Ad esempio, per $\delta = 110100$, avremo $\Delta(B') = \Delta(110100) = \{(000000, 110100), (000001, 110101), ...\}$, ossia gli input che avranno come valore di differenza esattamente δ. Per ognuna delle $2^6 = 64$ coppie in $\Delta(B')$, calcolo l'output XOR di 4 bit e ottengo quindi una tabella di distribuzione dei valori possibili - Figura 59. La tabella raffigura, data la differenza in input δ, avremo quelle determinate differenze in output. I numeri rappresentano il numero di coppie che hanno differenza δ in input, che hanno quella determinata

differenza in output. Il crittoanalista analizza la distribuzione, per trovare le differenze. Se una tabella fosse distribuita uniformemente, non sarebbe utile in termini di crittoanalisi.

0000	0001	0010	0011	0100	0101	0110	0111
0	**8**	**16**	**6**	**2**	**0**	**0**	**12**
1000	1001	1010	1011	1100	1101	1110	1111
6	**0**	**0**	**0**	**0**	**8**	**0**	**6**

Figura 59: Tabella profilo XOR S_1

Posso ripetere il calcolo per tutti i 2^6 valori, ottengo una tabella con 64 righe e 16 colonne. La Figura 59 rappresenta la tabella per i primi valori di δ. Ogni cella indica il numero n di coppie input $(B, B+\delta)$ tale che l'output è proprio Δ e restituisce proprio una probabilità come $\frac{n}{64}$. Ad esempio, se ho $(\delta = 1, \Delta = 1) = 0$ significa che la differenza input 1 non causa differenze su output 1. Invece, se ho $(\delta = 2, \Delta = 3) = 8$ significa che la differenza input 2 causa differenze sull'output 3 8 volte su 64.

δ	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	**64**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
1	**0**	**0**	**0**	**6**	**0**	**2**	**4**	**4**	**0**	**10**	**12**	**4**	**10**	**6**	**2**	**4**
2	**0**	**0**	**0**	**8**	**0**	**4**	**4**	**4**	**0**	**6**	**8**	**6**	**12**	**6**	**4**	**2**
3	**14**	**4**	**2**	**2**	**10**	**6**	**4**	**2**	**6**	**4**	**4**	**0**	**2**	**2**	**2**	**0**
4	**0**	**0**	**0**	**6**	**0**	**10**	**10**	**6**	**0**	**4**	**6**	**4**	**2**	**8**	**6**	**2**

Figura 60: Tabella profilo XOR S_1

Prendiamo come esempio le S-Box del DES (Figura 61), la chiave non ha effetto sullo XOR: $B \oplus B* = (E \oplus J) \oplus (E * \oplus J) = E \oplus E*$. Quindi, δ (la differenza in input) non dipende dalla chiave. Quindi, possiamo dire che la tabella di profilo XOR non dipende dalla chiave. Se conosco $B, B*$, $\delta = B \oplus B*$ e Δ, cosa posso dire sulla chiave? Possiamo stabilire che l'input non dipende dalla chiave, al contrario dell'output dello XOR: $S(E \oplus J) \oplus (E * \oplus J)$.

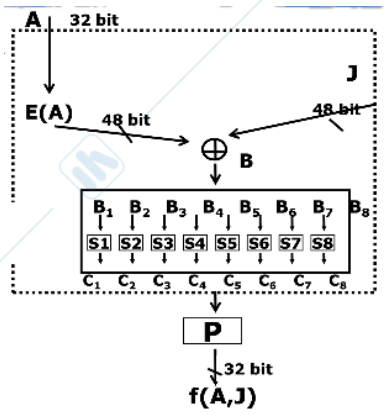


Figura 61: Tabella profilo XOR S_1

Il ragionamento con cui si applica la crittoanalisi differenziale è quello di andare a vedere la differenza in output e successivamente, calcolo tutti i possibili input che hanno quella differenza in output. Ad esempio, se conosco la coppia di input (B, $B*$) e conosco la differenza output Δ, posso calcolare la differenza di input δ e l'insieme IN, che rappresenta tutti possibili input che se messi in XOR, permettono di ottenere la stessa differenza in input e in output. A partire dall'insieme IN è possibile calcolare K_1, l'insieme delle possibili chiavi. Considerando una seconda coppia di input $(B', B*)$, si ottiene un altro insieme possibile K_2. La chiave si trova nell'intersezione di K_1 e K_2. Ripeto questo procedimento, finché non ottengo la chiave K input. Come si applica la procedura all'algoritmo DES? Si considerano sequenze input particolari $(A_1, 0)$ e $(A_2, 0)$ con $\delta = (A_1 \oplus A_2, 0)$ e $\Delta = (A_1 \oplus A_2, 0)$. Questo procedimento va effettuato per tutte le 8 S-Box, è per questo che si tratta di una crittoanalisi ipotetica. Se DES avesse avuto un numero inferiore di round, la crittoanalisi differenziale sarebbe molto più semplice e più rapida - Figura 62.

Numero round	Chosen plaintext	Known plaintext
8	2^{14}	2^{38}
9	2^{24}	2^{44}
10	2^{24}	2^{43}
11	2^{31}	2^{47}
12	2^{31}	2^{47}
13	2^{39}	2^{52}
14	2^{39}	2^{51}
15	2^{47}	2^{56}
16	2^{47}	2^{55}

Figura 62: Crittoanalisi differenziale: numero di round

4.4.3 Crittoanalisi Lineare

La crittanalisi lineare è una forma generale di crittanalisi basata sulla ricerca di approssimazioni affini al comportamento di un cifrario. Gli attacchi sono stati sviluppati per i cifrari a blocchi e quelli a flusso. La crittanalisi lineare è una delle due tecniche di attacco più utilizzate contro i cifrari a blocchi: l'altra è la crittanalisi differenziale. La scoperta della crittanalisi lineare è attribuita a Mitsuru Matsui nel 1993. L'attacco non è però generalmente praticabile, dato che richiede 2^43 testi in chiaro noti. Si tratta di un attacco di tipo Known Plaintext: l'idea è quella di trovare una relazione lineare $y = Ax + Bk$ al posto della sostituzione non lineare f: $y = f(x, k)$. La crittoanalisi lineare è un sistema di crittoanalisi basato sull'analisi di una coppia cifrato - decifrato intercettata. L'attacco ha lo scopo di scoprire la chiave utilizzata nei messaggi. Non è necessario essere in possesso del programma utilizzato per cifrare/decifrare. Il metodo è orientato all'analisi tramite un'approssimazione lineare volta a descrivere il comportamento del blocco di bytes a partire dal testo in chiaro. Più in particolare si lavora a livello di bit utilizzando un registro a scorrimento per lo shift dei bytes ed effettuando la somma di alcuni stadi con l'uscita di sinistra di tale registro. In base ai dati così raccolti ed attentamente esaminati si può scoprire la chiave. La probabilità di successo è alta.

Sono state suggerite una gran varietà di migliorie all'attacco, incluse l'uso di approssimazioni lineari multiple o l'incorporazione di espressioni non lineari. Dai nuovi cifrari in genere ci si attende che i loro sviluppatori pubblichino anche il livello di sicurezza contro la crittanalisi lineare.

Data una funzione booleana $f : \sum^n \to \sum$, la quale significa che a partire da una stringa a n bit si ottiene un bit (vero o falso). f è rappresentata dalla sua tavola di verità, ossia dal vettore $(f(a_0), ..., f(a_{2n-1})$, con $a_0 = 000...00, a_1 = 000...01, a_{2n-1} = 111..111$. Quindi se f è una funzione $\sum^3 \to \sum$ definita come $f(x) = x_1x_2x_3 \oplus x_1x_3 \oplus x_2 \oplus x_3 \oplus 1$, la sua tabella di verità è data dai seguenti valori:

- $f(000) = 1$
- $f(001) = 0$
- $f(010) = 0$
- $f(011) = 1$
- $f(100) = 1$
- $f(101) = 1$
- $f(110) = 0$
- $f(111) = 1$

Una funzione booleana è affine se può essere rappresentata come $f(x) = a_0 \oplus a_1x_1 \oplus ... \oplus a_nx_n)$ con $a_i \sum = 0, 1$. É lineare se a_0 è uguale a 0, ossia se manca il coefficiente. Quindi:

- $f(x_1, x_2, x_3) = x_3 \oplus x_1 \oplus 1$ è affine (a_0 è uguale a 1)
- $f(x_1, x_2, x_3) = x_3 \oplus x_1$ è lineare (a_0 è uguale a 0)

Se ho due funzioni booleane f e g la distanza è definita come la distanza fra le loro tabelle di verità $d(f, g) = H(f(x) \oplus g(x))$, dove H è il peso di Hamming (il numero di 1) delle tabella di verità. Questa funzione stabilisce permette di contare quanto diverse sono le due tabelle di verità. Una approssimazione lineare ad f è una funzione lineare $h*$ a distanza minima da f, ossia con il minimo numero di "differenze".

La non linearità $N(f)$ di una funzione è definita: $N(f) = mind(g, f)$ con g funzione affine. Se ho la distanza d considero anche $2^n - d$.

Le funzioni lineari su $\sum^2$ sono $0, x_1, x_2, x_1 \oplus x_2$. La Figura 63 rappresenta un esempio di applicazioni delle funzioni lineari su stringhe di bit con $n = 2$.

$x_2\,x_1$	F	0	x_1	x_2	$x_1 \oplus x_2$
00	0	0	0	0	0
01	0	0	1	0	1
10	0	0	0	1	1
11	1	0	1	1	0

Figura 63: Funzioni lineari su $\sum^2$

Ogni S-Box realizza una funzione $S : \sum^6 \to \sum^4$. Quindi le S-Box ricevendo in input sei bit, restituiscono 4 funzioni: $S(s_6, s_5, s_4, s_3, s_2, s_1) = (f_4, f_3, f_2, f_1)$. Posso, quindi, trattare una S-Box come insieme di 4 funzioni booleane $f_i : \sum^6 \to \sum$, con $i = 1, ..., 4$. Quindi ottengo $S(s) = (f_4(s), f_3(s), f_2(s), f_1(s))$, dove $s = (s_6, s_5, s_4, s_3, s_2, s_1)$, cioè i bit in input. Per ogni S-Boc si crea una tabella delle distanze, dove:

- Le righe sono le $2^6 = 64$ funzioni lineari possibili

- Le colonne sono le $2^4 = 16$ combinazioni lineari degli output della S-box rappresentate dalle funzioni f_i, $f = (a_4 f_4 \oplus a_3 f_3 \oplus a_2 f_2 \oplus a_1 f_1(s))$
- Ogni cella restituisce la distanza fra la funzione della S-box in colonna e la funzione lineare sulla riga. Per ogni input s e ogni colonna calcolo la distanza d tra: $S(s)$ output della S-box e $l(s)$ funzione lineare corrispondente.

Ad esempio, la riga $h_{31} = 110001$ corrisponde alla funzione lineare $l(s) = s_6 \oplus s_5 \oplus s_1$. La colonna $h_9 = 1001$ corrisponde a $f = f_4 \oplus f_1$ La cella (31,9) contiene il valore 34, cioè indica che ci sono 34 differenze fra la funzione l e la S. La migliore approssimazione lineare di una funzione f è la funzione affine l che è la più vicina (meno differenze con f). La tabella consente di scegliere la migliore approssimazione lineare di ogni f_i : basta guardare la min entry della S-Box. Guardando la min e max entry e stabilendo quanto è buona l'approssimazione: d restituisce il numero di input per cui differiscono. Per un input a caso c'è probabilità $(2^n - d)/2^n$ che l e f siano uguali. Usando le approssimazioni lineari per le S-Box e coppie di testi noti chiari/cifrati è possibile creare dei legami fra i bit della chiave, risalendo nella struttura a blocchi, creando e risolvendo equazioni lineari per i bit della chiave.

Le tecniche di crittoanalisi lineare e differenziale sono tecniche generali di crittoanalisi che hanno successo con molti cifrari, anche se con DES si tratta di attacchi non pratici. La crittoanalisi lineare è l'attacco più efficiente di tipo known plaintext. Ha bisogno di 2^{43} coppie plain/cipher text.

5 Cifratura AES

Il National Institute of Standard and Technology (NIST) propose il DES come standard nel 1977, il quale viene riaffermato nel 1993 fino a Dicembre 1998. Le critiche che vennero mosse al DES negli anni sono:

- chiave di soli 56 bit (112 per 3-DES)
- blocchi di 64 bit
- criteri costruttivi non chiari (ci sono trapdoor nelle S-box?)
- lento nell'implementazione software (3-DES ancora di più)

Quindi, l'obiettivo del NIST fu quello di sviluppare e introdurre un nuovo cifrario a blocchi per uso commerciale e governativo più sicuro ed efficiente di 3-DES.

Il 12 settembre 1997 il National Institute of Standards and Technology, o NIST, propose un bando per la realizzazione del nuovo algoritmo di cifratura a blocchi che il governo statunitense avrebbe utilizzato come standard per la cifratura dei dati sensibili e non classificati. L'Advanced Encryption Standard (AES) doveva rimpiazzare il Triple DES, questo era un algoritmo intermedio utilizzato dopo che il Data Encryption Standard (DES) si era rivelato forzabile grazie a un attacco a forza bruta. Questo era possibile dato che il DES utilizzava una chiave a 56 bit, troppo piccola per le moderne capacità di calcolo. Il triple DES era solo un rimedio temporaneo dato che questo non era altro che il DES applicato tre volte con due chiavi diverse e quindi la chiave totale era lunga 112 bit. Poi il DES era veloce se implementato in hardware ma lento se implementato in software e il Triple DES era a maggior ragione ancora più lento dovendo utilizzare tre volte l'algoritmo DES sullo stesso blocco di dati. Le specifiche dichiaravano che l'AES non doveva essere segreto, ci si aspettava che si sarebbe diffuso come algoritmo di cifratura standard anche al di fuori degli Stati Uniti d'America sostituendo il DES e il Triple DES. I requisiti del nuovo standard erano molto restrittivi. Doveva essere un algoritmo a blocchi, con blocchi di 128 bit, doveva gestire chiavi di 128, 192 e 256 bit. Doveva essere sicuro, veloce sia in hardware che in software e funzionare anche con apparecchiature dotate di pochissima RAM e ROM come i

sistemi dedicati e le smart card. Inoltre doveva essere royalty-free, ossia una tipologia di contratto fra due entità (agenzia ed utilizzatore o utente), che avviene quando la prima concede il diritto alla seconda di utilizzare una risorsa, ad esempio una fotografia o un brano musicale. Il termine royalty-free significa che una volta che la licenza è stata acquisita, l'utente può utilizzare la risorsa senza limiti di tempo e spazio senza dover sostenere ulteriori costi, naturalmente seguendo le linee guida della licenza stessa. Nel 1998 quindici diversi algoritmi vennero presentati da partecipanti provenienti da sette nazioni. In ordine alfabetico: CAST-256, CRYPTON, DEAL, DFC, E2, FROG, Hasty Pudding Cipher, LOKI97, MAGENTA, MARS, RC6, Rijndael, SAFER+, Serpent e Twofish. Alcuni candidati non erano sufficientemente sicuri anche se la maggior parte superò il test di sicurezza, infatti a marzo 1999 ci fu una presentazione in cui vennero presentati gli strumenti di analisi e testing. Il 9 agosto 1999 la commissione restrinse la rosa a cinque candidati: MARS, RC6, Rijndael, Serpent, e Twofish. Successivamente ad un pubblico scrutinio, ad aprile 2000 si tenne una conferenza per presentare ulteriori strumenti di analisi e testi. Il 2 ottobre, 2000, il NIST annunciò che Rijndael era stato selezionato per diventare la base dell'AES. Il 28 febbraio 2001 fu pubblicatio un Draft di Federal Information Processing Standard (FIPS). Si tenne un pubblico scrutinio della durata di 90 giorni e venne inoltrata la proposta al Secretary of Commerce per ricevere l'approvazione. Il 6 dicembre 2001 il NIST annunciò la definitiva approvazione dell'AES sotto il nome di FIPS PUB 197. L'utilizzo effettivo dell'AES come standard crittografico iniziò il 26 maggio 2002. Le piattaforme del NIST utilizzate per l'analisi dei candidati furono:

- PC IBM-compatibile, Pentium Pro 200MHz, 64MB RAM, WINDOWS 95
- Compilatori Borland C++ 5.0 ed il Java Development Kit (JDK) 1.1

La selezione del NIST fu basata sui seguenti criteri: sicurezza, efficienza implementazioni hardware e software, grandezza codice e memoria utilizzata. La documentazione che dovevano fornire i candidati era:

- Descrizione algoritmo
- Analisi algoritmo (vantaggi e limiti)
- Stima dell'efficienza computazionale
- Analisi dell'algoritmo rispetto agli attacchi di crittoanalisi più conosciuti (ad esempio known o chosen plaintext)
- Implementazione di riferimento in ANSI C
- Implementazione ottimizzata dell'algoritmo implementata in ANSI C e Java

Riassumendo, l'Advanced Encryption Standard (AES), conosciuto anche come Rijndael, di cui più propriamente è una specifica implementazione, 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) che ha perso poi efficacia per vulnerabilità intrinseche. AES è stato adottato dalla National Institute of Standards and Technology (NIST) e dalla US FIPS PUB nel novembre del 2001 dopo 5 anni di studi, standardizzazioni e selezione finale tra i vari algoritmi proposti. L'algoritmo scelto 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", derivato dai nomi degli inventori. Rijndael, in fiammingo, si pronuncia approssimativamente "rèin-daal".

A differenza del DES, Rijndael è una rete a sostituzione e permutazione, non una rete di Feistel, che implementa comunque il principio crittografico di Shannon di "confusione e diffusione".

Infatti, AES lavora in parallelo sull'intero blocco in input. AES è veloce sia se sviluppato in software sia se sviluppato in hardware, è relativamente semplice da implementare, richiede poca memoria ed offre un buon livello di protezione/sicurezza, motivi che complessivamente l'hanno preferito agli altri algoritmi proposti.

AES è un cifrario a blocchi iterato. Formalmente, AES non è equivalente al Rijndael (sebbene nella pratica siano intercambiabili) dato che il Rijndael gestisce differenti dimensioni di blocchi e di chiavi. Nell'AES il blocco è invece di dimensione fissa (128 bit) e la chiave può essere di 128, 192 o 256 bit mentre il Rijndael specifica solo che il blocco e la chiave devono essere un multiplo di 32 bit con 128 bit come minimo e 256 bit come massimo (Figura 64).

	Key Length (Nk words)	Block Size (Nb words)	Number of Rounds (Nr)
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Figura 64: Parametri AES

Con 128 bit, il numero di chiavi possibile è $2^{128} = 3.4 \times 10^{38}$. Una macchina che prova 2^{55} chiavi al secondo impiega 149.000 miliardi di anni per rompere AES. Ovviamente aumentando i bit del blocco, aumenta anche il numero di chiavi possibili:

- Con blocchi da 192 bit: $2^{192} = 6.2 \times 10^{57}$ chiavi possibili
- Con blocchi da 256 bit: $2^{256} = 1.1 \times 10^{77}$ chiavi possibili

Si pensa che AES resterà sicuro per i prossimi 20 anni.

La chiave di lunghezza variabile (128, 192, 256 bit) è rappresentata come una matrice di byte con 4 righe e Nk colonne, dove Nk è uguale alla lunghezza della chiave diviso 32 (Figura 65):

- Chiave 128 bit = 16 byte $\rightarrow Nk = 128/32 = 4$. Quindi la tabella avrà 4 righe e 4 colonne.
- Chiave 192 bit = 24 byte $\rightarrow Nk = 192/32 = 6$. Quindi la tabella avrà 4 righe e 6 colonne.
- Chiave 256 bit = 32 byte $\rightarrow Nk = 256/32 = 8$. Quindi la tabella avrà 4 righe e 8 colonne.

$K_{0,0}$	$K_{0,1}$	$K_{0,2}$	$K_{0,3}$
$K_{1,0}$	$K_{1,1}$	$K_{1,2}$	$K_{1,3}$
$K_{2,0}$	$K_{2,1}$	$K_{2,2}$	$K_{2,3}$
$K_{3,0}$	$K_{3,1}$	$K_{3,2}$	$K_{3,3}$

Figura 65: Matrice Chiave

Il blocco è rappresentato come una matrice di byte, funzionante esattamente come la chiave. Quindi, in caso di blocchi da 128 bit, si avrà una matrice 4×4 (Figura 66).

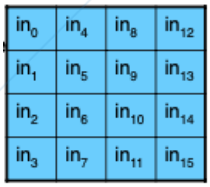


Figura 66: Matrice Blocco AES

AES opera utilizzando matrici di 4×4 byte chiamate stati (states). Quando l'algoritmo ha blocchi di 128 bit in input, la matrice State ha 4 righe e 4 colonne; se il numero di blocchi in input diventa di 32 bit più lungo, viene aggiunta una colonna allo State, e così via fino a 256 bit. In pratica, si divide il numero di bit del blocco in input per 32 e il quoziente specifica il numero di colonne. La matrice di byte in input viene copiata nella matrice state; al termine, la matrice state verrà copiata nella matrice di output.

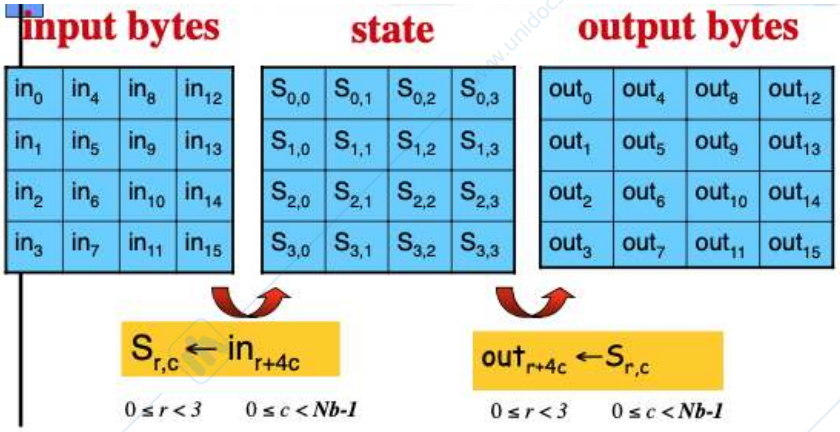


Figura 67: Matrice Stato

Il byte è l'unità di base nella computazione dell'AES. I valori del byte sono rappresentati in notazione esadecimale: due cifre esadecimali per ciascun byte $\{11010100\} \rightarrow d4$. Ciascun byte è interpretato come un elemento del campo finito $GF(2^8)$: $\{b_7b_6b_5b_4b_3b_2b_1b_0\} \rightarrow b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$. Quindi: $\{11010100\} \rightarrow (x^7 + x^6 + x^4 + x^2) = 212$. L'addizione in $GF(2^8)$ corrisponde al polinomio i cui coefficienti sono la somma modulo 2 dei coefficienti dei due polinomi: $(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2 \rightarrow \{01010111\} \oplus \{10000011\} = \{11010100\} \rightarrow \{57\} \oplus \{83\} = \{d4\}$. La moltiplicazione in $GF(2^8)$ (denotata da $\cdot$) corrisponde alla moltiplicazione di polinomi modulo un polinomio irriducibile[2] di grado 8. Il risultato è un polinomio di grado ≤ 7. Il poliniomo irriducibile usato per AES è $m(x) = x^8 + x^4 + x^3 + x + 1$. É solo uno dei 30 polinomi irriducibili di grado 8. Veediamo un esempio di moltiplicazione: $\{01010111\} \cdot \{10000011\} = \{11000001\} \rightarrow \{57\} \cdot \{83\} = \{c1\}$. Vediamo un esempio dettagliato del procedimento: $(x^6+x^4+x^2+x+1)(x^7+x+1) = x^{13}+x^{11}+x^9+x^8+x^7+x^7+x^5+x^3+x^2+x+x^6+x^4+x^2+x+1 = x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1$. Ora prendiamo questo output e applichiamo il modulo di $x^8 + x^4 + x^3 + x + 1$, ossia dividiamo per $m(x)$ e teniamo il resto, ottenendo $x^7 + x^6 + 1$. La moltiplicazione ha le seguenti proprietà:

- Associativa
- Identità $\{01\}$
- Esiste inverso $a^{-1}(x)$ per ogni $a(x)$, calcolato con Euclide esteso

[2]Polinomi i cui unici divisori sono 1 e se stesso

La struttura del campo finito $GF(2^8)$ consiste in 256 valori memorizzati in un byte con le operazioni di addizione (XOR) e moliplicazione (polinomi modulo polinomio irriducibile).

I polinomi con quattro termini possono essere rappresentati con coefficienti che sono elementi di un campo finito. Vediamo ad esempio: $a(x) = a_3 \cdot x^3 + a_2 \cdot x_2 + a_1 \cdot x + a_0$, che viene denotato in termini di word nella forma $[a_0, a_1, a_2, a_3]$. Bisogna notare che i polinomi definiti in questo modo si comportano in maniera piuttosto differente dai polinomi usati per definire gli elementi di un campo finito, anche se utilizzano entrambi lo stesso valore indeterminato x. Infatti, i coefficienti dei polinomi che stiamo adesso considerando sono essi stessi degli elementi di un campo finito, sono in questo caso byte, mentre nella definizione precedente erano bit. Così, un vettore di 4 byte corrisponde ad un polinomio di grado minore di 4 con coefficienti nel campo finito GF(28). Questi polinomi possono essere addizionati sommando semplicemente i coefficienti aventi potenze di x corrispondenti. La somma equivale ad una operazione di XOR tra byte corrispondenti in ognuna delle word: $a(x) + b(x) = (a_3 \oplus b_3) \cdot x^3 + (a_2 \oplus b_2) \cdot x^2 + (a_1 \oplus b_1) \cdot x + (a_0 \oplus b_0)$. La moltiplicazione è più complicata. Supponiamo di avere due polinomi con coefficienti nel campo finito $GF(2^8)$: $a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ e $b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$. Il loro prodotto $c(x) = a(x) \cdot b(x)$ è dato da: $c(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0$ con:

- $c_0 = a_0 \cdot b_0$
- $c_1 = a_1 \cdot b_0 \oplus a_0 \cdot b_1$
- $c_2 = a_2 \cdot b_0 \oplus a_1 \cdot b_1 \oplus a_0 \cdot b_2$
- $c_3 = a_3 \cdot b_0 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \oplus a_0 \cdot b_3$
- $c_4 = a_3 \cdot b_1 \oplus a_2 \cdot b_2 \oplus a_1 \cdot b_3$
- $c_5 = a_3 \cdot b_2 \oplus a_2 \cdot b_3$
- $c_6 = a_3 \cdot b_3$

Chiaramente, $c(x)$ può essere rappresentato da un vettore di 4 byte. Riducendo $c(x)$ modulo un polinomio di grado 4, il risultato può essere ridotto ad un polinomio di grado inferiore a 4, in Rijndael viene utillizzato il polinomio $m(x) = x^4 + 1$. Dal momento che $x^i mod(x^4 + 1) = x^{(imod4)}$, il prodotto modulare di $a(x)$ e $b(x)$, denotato con $d(x) = a(x) \oplus b(x)$, è dato da: $d(x) = d_3x^3 + d_2x^2 + d_1x + d_0$ con:

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

L'operazione che consiste nella moltiplicazione di un polinomio fisso $a(x)$, può essere scritta come moltiplicazione matriciale, dove la matrice è una matrice circolare - Figura 68.

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

Figura 68: Polinomi con coefficienti in $GF(2^8)$

Dal momento che $x^4 + 1$ non è un polinomio irriducibile sul campo finito $GF(2^8)$, la moltiplicazione con un fissato polinomio di 4 termini non è necessariamente invertibile. Pertanto Rijndael specifica un fissato polinomio con 4 termini che ha un inverso moltiplicativo: $a(x) = 03x^3 + 01x^2 + 01x + 02$ e $a^{-1}(x) = 0bx^3 + 0dx^2 + 09x + 0e$ che sarà utilizzato nella cifratura e nella decifratura.

C'è un passaggio iniziale: AddRoundKey – Ogni byte della tabella viene combinato con la chiave di sessione, la chiave di sessione viene calcolata dal gestore delle chiavi. Si tratta della schedulazione della chiave: 44, 52 o 60 sottochiavi a 32 bit. Successivamente per cifrare sono previsti diversi round o cicli di processamento (solitamente 10, 12 o 14): ogni round (fase) dell'AES (eccetto l'ultimo) consiste dei seguenti quattro passaggi:

- SubBytes: nel passaggio SubBytes ogni byte della matrice viene modificato tramite la S-box a 8 bit. Questa operazione provvede a fornire la non linearità all'algoritmo. La S-box utilizzata è derivata da una funzione inversa nel campo finito $GF(2^8)$, conosciuta per avere delle ottime proprietà di non linearità. Per evitare un potenziale attacco basato sulle proprietà algebriche la S-box è costruita combinando la funzione inversa con una trasformazione affine invertibile.

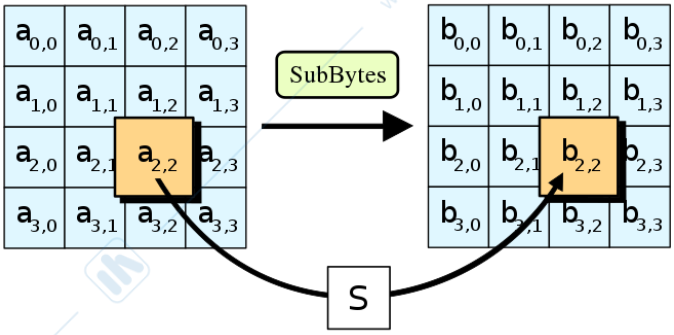


Figura 69: Nel passaggio SubBytes, ogni byte della matrice è sostituito con i dati contenuti nella trasformazione S; $b_{ij} = S(a_{ij})$

La tabella 16×16 contiene una permutazione dei 256 valori possibili di un byte. I primi 4 bit determinano l'indice di riga, gli altri 4 l'indice di colonna.

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	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figura 70: S-Box

La costruzione della S-Box si esegue secondo le seguenti azioni:

– Inizializzare la S-box con i valori dei byte in ordine ascendente riga per riga
  * Prima riga: 00, 01, ..., 0F,
  * Seconda riga: 10, 11, ..., 1F,
  * ...
– Sostituire ciascun byte con il suo inverso moltiplicativo in $GF(2^8)$: {00} resta {00}
– Applicare una trasformazione affine in $GF(2^8)$: $b_i' \leftarrow b_i \oplus b_{(i+4)mod8} \oplus b_{(i+5)mod8} \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8} \oplus \{01100011\}_i$.

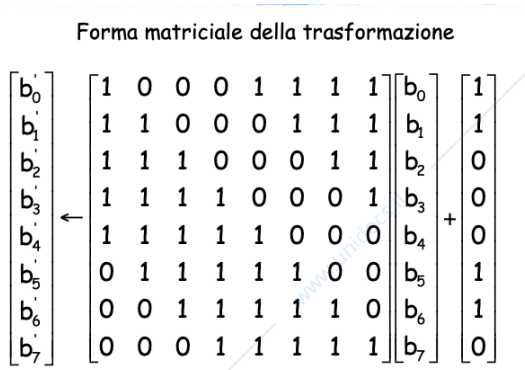


Figura 71: Forma matriciale della trasformazione nella S-Box

L'output non è una funzione lineare dell'input. Non ha punti fissi diretti né opposti: S-box(a) $\neq a$ e S-box($\bar{a}$) $\neq \bar{a}$. É invertibile: Inverse_S-box(S-box(a)) $= a$. Non è self-invertibile S-box(a) $\neq$ Inverse_S-box(a), ad esempio, S-box(95)=2A, ma IS-box(95)=AD. É progettata per resistere ad attacchi crittoanalitici noti.

- ShiftRows – Spostamento dei byte di un certo numero di posizioni dipendente dalla riga di appartenenza (Permutazione). Il passaggio ShiftRows provvede a scostare le righe della matrice di un parametro dipendente dal numero di riga. Nell'AES la prima riga resta invariata, la seconda viene spostata di un posto verso sinistra, la terza di due posti e la quarta di tre. In questo modo l'ultima colonna dei dati in ingresso andrà a formare la diagonale della matrice in uscita. (Rijndael utilizza un disegno leggermente diverso per via delle matrici di lunghezza non fissa.). Tutte le operazioni sono effettuate utilizzando l'indice della colonna "modulo" il numero di colonne. I 4 byte di una colonna sono spostati su colonne differenti.

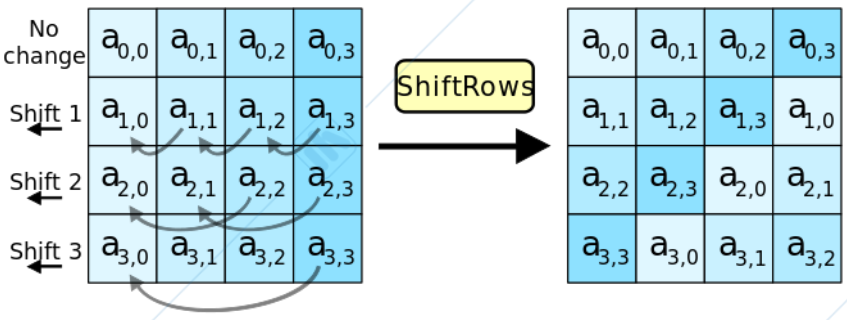


Figura 72: Nel passaggio ShiftRows, i byte di ogni riga vengono spostati verso sinistra dell'ordine della riga

- MixColumns – Combinazione dei byte con un'operazione lineare, i byte vengono trattati una colonna per volta. Si tratta di una sostituzioe che usa aritmetica sul campo finito $GF(2^8)$. Il passaggio MixColumns prende i quattro byte di ogni colonna e li combina utilizzando una trasformazione lineare invertibile. Utilizzati in congiunzione, ShiftRows e MixColumns provvedono a far rispettare il criterio di confusione e diffusione nell'algoritmo (teoria di Shannon). Ogni colonna è trattata come un polinomio in $GF(2^8)$ e viene moltiplicata modulo $x^4 + 1$ per un polinomio fisso $c(x) = 3x^3 + x^2 + x + 2$.

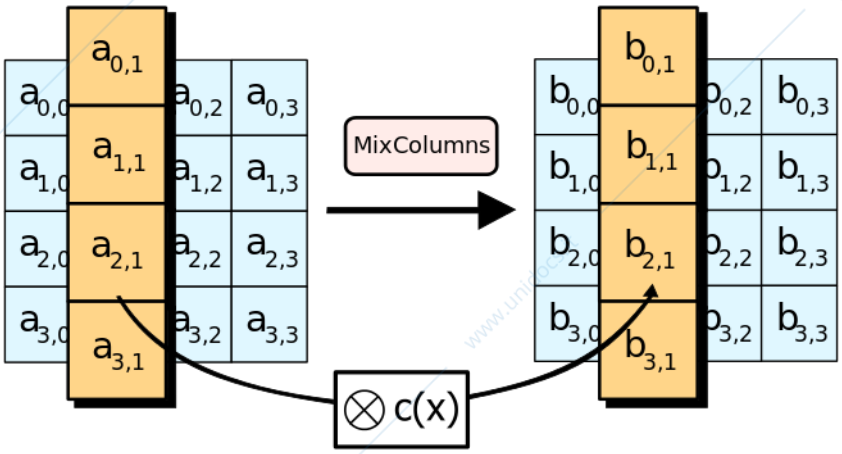


Figura 73: Nel passaggio MixColumns ogni colonna di byte viene moltiplicata per un polinomio fisso $c(x)$

I byte nelle colonne sono combinati linearmente.

- AddRoundKey – Ogni byte della tabella viene combinato con la chiave di sessione, la chiave di sessione viene calcolata dal gestore delle chiavi. Quindi, si tratta di uno XOR bit a bit con la chiave espansa. Il passaggio AddRoundKey combina con uno XOR la chiave di sessione con la matrice ottenuta dai passaggi precedenti (State). Una chiave di sessione viene ricavata dalla chiave primaria ad ogni round (con dei passaggi più o meno semplici, ad esempio uno shift di posizione dei bit) grazie al Key Scheduler.

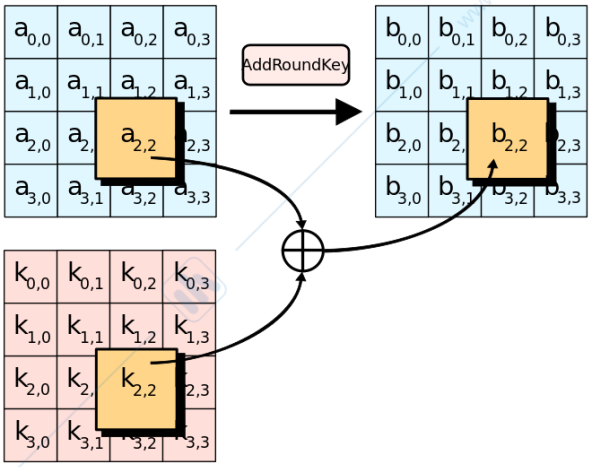


Figura 74: Nel passaggio AddRoundKeys ogni byte della matrice viene combinato con la sua sottochiave tramite un'operazione di XOR

Il numero di round o cicli di processamento/elaborazione crittografica dei quattro passaggi precedenti è 10 con l'ultimo round che salta il passaggio MixColumns. A seguito la descrizione di ogni singolo passaggio.

Rijndael utilizza una chiave di cifratura K e implementa una routine di espansione della chiave per generare una chiave schedulata. A partire dalla chiave iniziale di $4 * Nk$ byte, genera un array di $Nb * (Nr + 1)$ word (chiavi schedulate): l'algoritmo di cifratura richiede un insieme iniziale di Nb word, ed ognuno degli Nr round richiedono Nb word della chiave. La chiave schedulata risultante consiste di un array lineare di word di 4 byte, denotato con $[w_i]$, con i che varia tra $0 < i < Nb(Nr+1)$. Utilizza due routine e un array di word costanti. Le routine sono:

- SubWord(): funzione che prende in input una word di 4 byte e applica la S-Box ad ognuno dei 4 byte per produrre una word di output.
- RotWord(): prende una word [a0,a1,a2,a3] come input, esegue una permutazione ciclica e restituisce una word [a0,a1,a2,a3].

L'array di word costanti è $Rcon[i]$, che contiene i valori dati da $[x^{i-1}, \{00\}, \{00\}, \{00\}]$, con x^{i-1} che rappresenta l'$(i-1)$-esima potenza di x nel campo $GF(2^8)$ (x è uguale a 0 2). Bisogna notare inoltre che la variabile i inizia con il valore di 1 anziché 0.

i	1	2	3	4	5	6	7	8	9	10
RC[i]	{01}	{02}	{04}	{08}	{10}	{20}	{40}	{80}	{1B}	{36}

Figura 75: Espansione chiave

Si può notare dalla Figura 76 che le prime Nk word della chiave espansa sono ottenute direttamente dalla chiave di cifratura, mentre ogni word successiva, $w[i]$, è uguale allo XOR della word precedente, $w[i-1]$, con la word di Nk posizioni più indietro. Per le word con posizioni che sono multiple di Nk, viene applicata a $w[i-1]$ una trasformazione prima dello XOR, seguita da uno XOR con una costante di round, $Rcon[i]$. Questa trasformazione consiste di uno shift ciclico dei byte in una word (RotWord()), seguito da una funzione SubWord() che applica una S-Box a tutti e quattro i byte della word. É importante notare che l'algoritmo di espansione della chiave per una chiave di 256 bit ($Nk = 8$) si comporta diversamente rispetto ad una chiave di 128 o 192 bit. Se $Nk = 8$ ed $i - 4$ è multiplo di Nk, la trasformazione SubWord() è applicata a $w[i-1]$ prima dello XOR.

```
KeyExpansion(byte key[4 * Nk], word w[Nb * (Nr + 1)], Nk)
begin
    i=0
    while (i < Nk)
        w[i] = word[key[4*i],key[4*i+1],key[4*i+2],key[4*i+3]]
        i = i + 1
    end while

    i = Nk
    while (i < Nb * (Nr + 1))
        word temp = w[i - 1]
        if (i mod Nk = 0)
            temp = SubWord(RotWord(temp)) xor Rcon[i / Nk]
        else if (Nk = 8 and i mod Nk = 4)
            temp = SubWord(temp)
        end if
    w[i] = w[i - Nk] xor temp
    i = i + 1
    end while
end
```

Figura 76: Espansione chiave

La Figura 77 rappresenta la struttura di AES, dove:

- Key Addition layer: la round key è addizionata (XOR) allo stato
- Byte Substitution layer (S-Box): ogni elemento dello stato è trasformato non linearmente (introduce confusion nei bit dello stato)
- Diffusion layer: introduce diffusion su tutti i bit dello stato ed è costituito da due sub-layer ShiftRows e MixColumn.

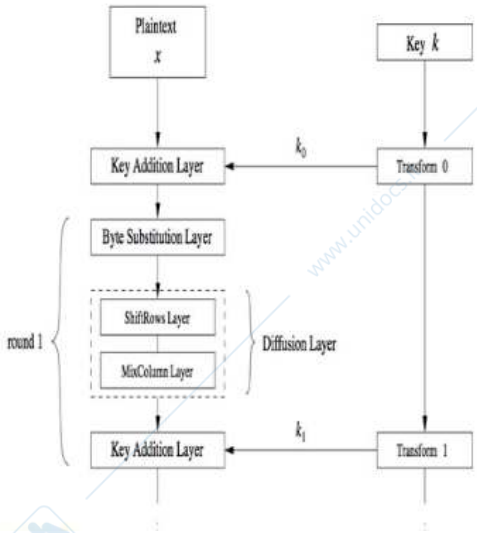


Figura 77: Struttura di AES

La Figura 78 rappresenta il valore di Security Strength, cioè un numero associato alla quantità di lavoro (ovvero il numero di operazioni) necessario per interrompere un algoritmo o un sistema crittografico. In questa raccomandazione, la sicurezza è specificata in bit ed è un valore specifico dell'insieme 80, 112, 128, 192, 256. Si noti che un livello di sicurezza di 80 bit non è più considerato sufficientemente sicuro. Da notare il numero di security strength non indica il numero di bit di cui è composta la chiave, ma si riferisce alla quantità di lavoro.

Security Strength	Symmetric key algorithms
≤ 80	2TDEA[21]
112	3TDEA
128	AES-128
192	AES-192
256	AES-256

Figura 78: Sicurezza di AES

5.1 Decifratura di AES

La fase di decifratura non è identica a quella di cifratura dal momento che gli step sono eseguiti in ordine inverso. 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 (Figura 79 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
}
```

Figura 79: Decifratura di AES

5.1.1 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.

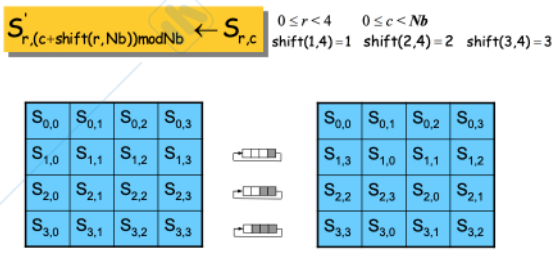


Figura 80: Trasformazione Shift Rows inversa

5.1.2 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

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 01 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} = \begin{bmatrix} 01 & 00 & 00 & 00 \\ 00 & 01 & 00 & 00 \\ 00 & 00 & 01 & 00 \\ 00 & 00 & 00 & 01 \end{bmatrix}$$

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 $b(x) = c(x)^{-1} \pmod{(x^4 + 1)}$, con $c(x)$ il polinomio usato in MixColumns.

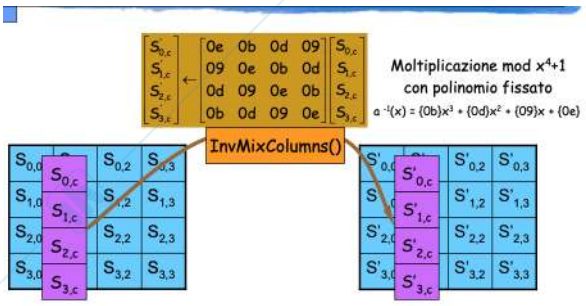


Figura 81: Trasformazione Mix Columns inversa

5.1.3 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. 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: $b'_i = bi \oplus b_{(i+2) \pmod 8} \oplus b_{(i+5) \pmod 8} \oplus b_{(i+7) \pmod 8} \oplus d_i$.

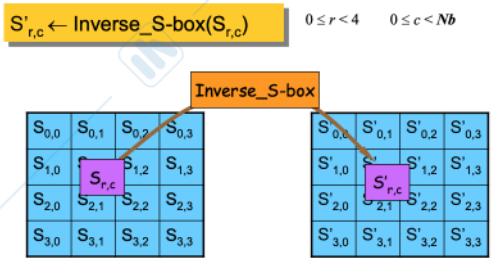


Figura 82: Trasformazione Substitute Bytes inversa

5.1.4 AddRoundKey Transformation

L'inversa di AddRoundKey è l'inversa di se stessa, come mostrato nella Figura 83.

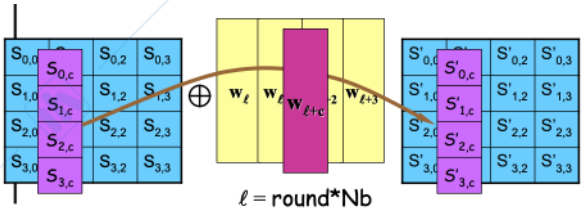


Figura 83: AddRoundKey Transformation

5.1.5 Decifratura AES

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, perchè necessità di una doppia implementazione. 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.

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

Considerando la chiave come una sequenza di Word allora sia la funzione AddRoundKey che la funzione InvMixColumns 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. InvMixColumns non viene applicato alla prima e ultima chiave di fase.

Nei microprocessori a 8 bit:

- ShiftRow è un'operazione di scorrimento
- AddRoundKey è uno XOR
- SubBytes richiede una tabella di 256 bytes
- MixColumns è una moltiplicazione matriciale su GF(2^8)
- Implementabile con shift e XOR condizionati

Nei microprocessori a 32 bit:

- Operazioni su word di 32 bit
- É possibile esprimere l'output di ogni fase come un'unica trasformazione complessa che è combinazione lineari di vettori

- Si usano 4 tabelle che contengono 256 word a 32 bit (4x1Kb)
- Tabelle differenti per l'ultima fase (basta 1 tabella con tre rotazioni)
- In totale 4 XOR e 4 ricerche per ogni colonna di output
- Vale anche per la decifratura con tabelle diverse (utilizzano S^{-1} Box).

6 DES: Cifratura multipla

Come è possibile costruire un cifrario più sicuro a partire dal DES (senza modificarne la struttura)? Attraverso la cifratura multipla: cifrare il messaggio varie volte con chiavi differenti, sperando che questo aumenti la sicurezza.

6.1 DES Doppio

Una delle debolezze del DES relative alla sicurezza è la lunghezza della chiave che, essendo pari a 56 bit (64 bit se consideriamo gli 8 bit di parità) fornisce uno spazio delle chiavi piuttosto piccolo: con un attacco che esamina lo spazio delle chiavi in maniera esaustiva, sono necessari in media 255 tentativi per indovinare la chiave. Da questa osservazione è nata l'idea di progettare un cifrario a blocchi alternativo al DES: si è pensato, infatti, di rendere il DES stesso più resistente agli attacchi di forza bruta, cifrando due volte il messaggio in chiaro con due chiavi diverse (una per ogni passo di cifratura). In tal modo la chiave diventa lunga $56 \times 2 = 112$ bit con un notevole incremento dello spazio delle chiavi. Il cifrario a blocchi che si ottiene apportando tali modifiche al DES viene detto Doppio DES. Dato un blocco x di 64 bit del testo in chiaro e 2 chiavi k_1 e k_2 a 56 bit, cifriamo x con la chiave k_1 ed otteniamo un blocco A di 64 bit; cifriamo quindi A con la chiave k_2 ed otteniamo il corrispondente testo cifrato Y: $Y = DES_{k_2}(DES_{k_1}(X))$ - Figura 84.

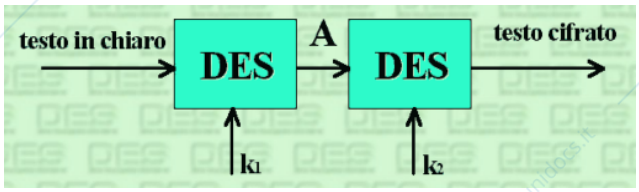


Figura 84: Cifratura del DES doppio

La decifratura è analoga alla cifratura, ma richiede che le chiavi siano applicate nell'ordine inverso. In particolare, dato un blocco Y di 64 bit del testo cifrato e 2 chiavi k_1 e k_2 a 56 bit, decifriamo Y con la chiave k_2 ed otteniamo un blocco A di 64 bit; decifriamo, quindi, A con la chiave k_1 ed otteniamo il corrispondente testo in chiaro X: $X = DES^{-1}_{k_1}(DES^{-1}_{k_2}(Y))$ - Figura 85.

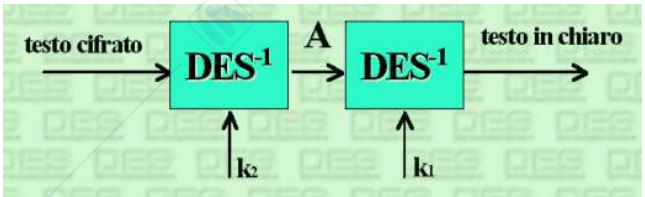


Figura 85: Decifratura del DES doppio

Soltanto nel 1992 si è avuta la certezza che "l'allungamento" della chiave da 56 bit a 112 bit rende effettivamente il Doppio DES più resistente del DES agli attacchi di forza bruta. É stato

dimostrato, infatti, che DES non è un gruppo e che quindi non è chiuso rispetto all'operazione di cifratura/decifratura. Se fosse stato vero il contrario, se cioè l'utilizzo di due chiavi fosse stato equivalente all'utilizzo di una sola chiave, allora la cifratura doppia, ed in generale la cifratura con un numero qualsiasi di passi (e di chiavi), sarebbe stata inutile; in tal caso, infatti, la ricerca esaustiva della chiave di cifratura del Doppio DES avrebbe richiesto uno sforzo computazionale pari a quello necessario per rompere il DES singolo. Vale, dunque, la seguente proposizione: Per ogni coppia di chiavi (k_1, k_2) non esiste alcuna chiave k_3 tale che $DES_{k_2}(DES_{k_1}(X)) = DES_{k_3}(X)$ per ogni testo in chiaro X. Diamo ora un'idea che ci permette di considerare valida tale proposizione. Consideriamo la cifratura col DES come una funzione che associa blocchi di 64 bit a blocchi di 64 bit: $DES : \{0,1\}^{64}$ ®$\{0,1\}^{64}$. Più precisamente a ciascuno dei 2^{64} possibili blocchi in input, la cifratura col DES associa uno dei 2^{64} possibili blocchi in modo univoco. Questo è necessario affinché la decifratura sia possibile; se, infatti, a due distinti blocchi input fosse associato, utilizzando una specifica chiave, lo stesso blocco output, allora in fase di decifratura sarebbe impossibile risalire al testo in chiaro originario. Dunque, partendo dai 2^{64} possibili blocchi del testo in chiaro, la cifratura col DES produce 2^{64} blocchi di testo cifrato, i quali non sono altro che una permutazione dei blocchi di partenza. Essendo i blocchi considerati pari a 2^{64}, vi sono $(2^{64})!$ funzioni distinte che generano una permutazione di tali blocchi. Per avere un'idea dell'ordine di grandezza di questo numero, osserviamo che $(2^{64})! > 10347380000000000000000 > (10^{10})^{20}$. D'altra parte il DES definisce una corrispondenza tra blocchi, diversa per ogni chiave; perciò, se utilizziamo una singola chiave di 56 bit, il numero totale di associazioni distinte è pari a 2^{56}. Osserviamo che $2^{56} = 10^{[(log2)\times 56]} >> 10^{56/3.32} >> 10^{16.86} << (10^{10})^{20}$. Pertanto è ragionevole pensare che, se applichiamo DES utilizzando due chiavi differenti, l'associazione univoca dei blocchi di 64 bit prodotta sarà una delle $(2^{64})!$ associazioni possibili e non una delle 2^{56} che si possono avere con una singola applicazione del DES.

6.1.1 Attacco Meet in the Middle

Sebbene resista molto bene agli attacchi di forza bruta, il Doppio DES non viene utilizzato in pratica, in quanto non incrementa la sicurezza rispetto alla singola cifratura. Esiste, infatti, un algoritmo che permette la rottura di tale schema con uno sforzo computazionale pari a quello necessario per rompere il DES singolo. Tale algoritmo, detto Meet in the Middle, non dipende dalla particolare struttura del DES, e pertanto può essere usato per attaccare un qualunque cifrario a blocchi. Esso si basa sul fatto che, se $y = DES_{k_1}(DES_{k_2}(x))$ allora $A = DES_{k_1}(x) = DES^{-1}_{k_2}(y)$. Supponiamo di conoscere una particolare coppia (x, y), dove x è il testo in chiaro e y il testo cifrato, e di voler determinare la chiave (k_1, k_2) utilizzata nella cifratura di x. L'attacco Meet in the Middle viene effettuato nel modo seguente. Per prima cosa cifriamo x utilizzando tutte le 2^{56} possibili chiavi k_1, memorizziamo i testi cifrati in una tabella ed eventualmente ordiniamo la tabella in base a questi risultati (Figura 86).

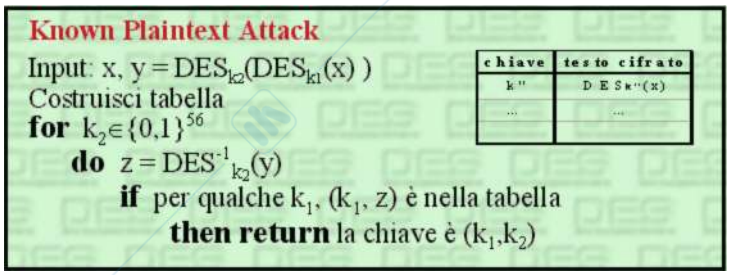


Figura 86: Costruzione Tabella

Successivamente decifriamo y usando tutte le 2^{56} possibili chiavi k_2. A questo punto per conoscere la coppia di chiavi (k_1, k_2) che fa passare dal testo in chiaro x al testo cifrato y, basta confrontare i valori $DES_{k_1,i}(x) e DES^{-1}_{k_2,j}(y)$. Se $DES_{k_1,i}(x) = DES^{-1}_{k_2,j}(y)$ per qualche i e j

allora le due chiavi corrispondenti potrebbero formare la coppia cercata. Non è detto, però, che questa sia effettivamente la chiave corretta perché, in realtà, possono esistere diverse chiavi che dallo stesso testo in chiaro x generano lo stesso testo cifrato y. In particolare, per ogni testo in chiaro x, il Doppio DES può produrre 2^{64} possibili valori di testo cifrato. D'altra parte il Doppio DES usa una chiave a 112 bit, per cui le chiavi possibili sono 2^{112}. Segue che, se scegliamo a caso una coppia (x, y), il numero di coppie di chiavi che trasformano x in y è in media $\frac{2^{112}}{2^{64}} = 2^{48}$. Pertanto l'attacco considerato produce circa 2^48 collisioni sulla coppia (x, y), e questo non ci consente di risalire alla coppia di chiavi realmente utilizzata, cioè non rende ancora possibile la rottura del cifrario. Allora scegliamo a caso un'altra coppia (x', y') di testo in chiaro-testo cifrato (indipendente dalla prima) tale che y' è prodotto da x' utilizzando la stessa coppia di chiavi (k_1, k_2). Costruiamo, poi, la relativa tabella cifrando x' per tutti i possibili valori di k_1 (Figura 87).

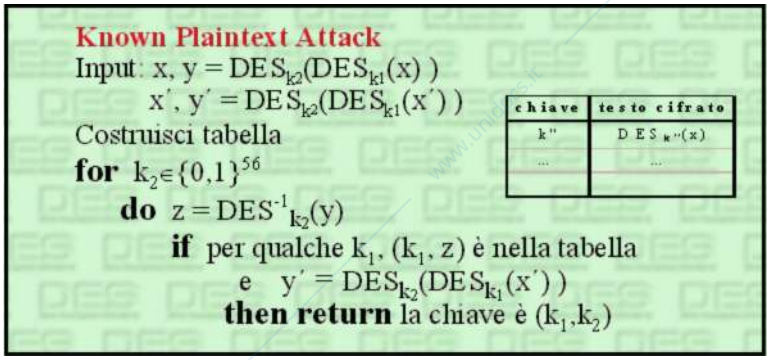


Figura 87: Costruzione Tabella

Se in corrispondenza di una stessa coppia di chiavi $(k_{1,i}, k_{2,j})$ si ha $DES_{k_1,i}(x) = DES^{-1}_{k_2,j}(y)$ e $DES_{k_1,i}(x') = DES^{-1}_{k_2,j}(y')$, allora con molta probabilità la coppia di chiavi $(k_{1,i}, k_{2,j})$ è quella effettivamente utilizzata nella cifratura. Osserviamo, infatti, che il numero di collisioni è, in questo caso, estremamente basso, ovvero $\frac{2^{112}}{2^{128}} = 2^{-16}$: pertanto la probabilità di determinare la chiave corretta è $1 - 2^{-16} = 0.99998474$. Questa probabilità può essere aumentata ulteriormente se, invece di una coppia di valori, ne consideriamo una tripla, una quadrupla, e così via. Calcoliamo il tempo richiesto da questa procedura:

- la costruzione della tabella richiede uno sforzo computazionale pari a 2^{56};
- la ricerca richiede un tempo che dipende dalla struttura utilizzata per memorizzare i dati. Se, ad esempio, ordiniamo tali dati rispetto alla cifratura DES, allora la ricerca richiede $log_2 2^{56} = 56$ tentativi. Se, invece, utilizziamo una tabella hash (ammesso che riusciamo a costruire una tabella hash così grande), allora il tempo è, in media, una costante.

Osserviamo che la precomputazione delle 2 tabelle, eventualmente effettuata nei tempi morti della macchina, richiede un tempo elevato, ma rende la ricerca estremamente veloce. In generale, il numero di test effettuati nella ricerca è abbastanza piccolo, pertanto il costo dell'intera procedura è dell'ordine di 2^{56}. In definitiva il Doppio DES è un cifrario a blocchi che utilizza una chiave di 112 bit e che può essere rotto con uno sforzo dell'ordine di 2^{56}, quindi non è migliore del DES singolo. Per questo motivo il Doppio DES non viene utilizzato in pratica.

6.2 DES Triplo

Il discorso fatto precedentemente per il DES doppio può essere ampliato per un cifrario ad n stadi. In particolare il DES triplicato, che effettua la cifratura di un testo in chiaro mediante l'uso di 3 stadi con tre chiavi differenti (k_1, k_2 e k_3, vedi figura 88), può essere sottoposto ad un attacco del tipo "meet in the middle".

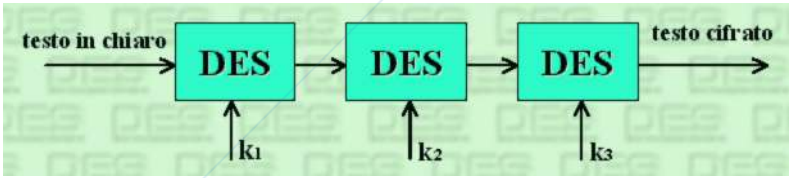


Figura 88: DES triplicato

Tale attacco può essere portato in due modi; vediamo il primo (Figura 89):

- Si costruisce una tabella di dimensione 2^{112} la quale contiene coppie del tipo $((k_1, k_2), DES_{k_1} \cdot (DES_{k_2}(x)))$ che rappresentano l'insieme di tutte le possibili cifrature di x mediante due chiavi.
- Successivamente si effettuano tutte le 2^{56} possibili decifrature di y con la chiave k_3, e, per ognuna di esse, si ricerca il valore ottenuto nella tabella.
- Quando la ricerca ha esito positivo si ottiene la chiave cercata che è (k_1, k_2, k_3).

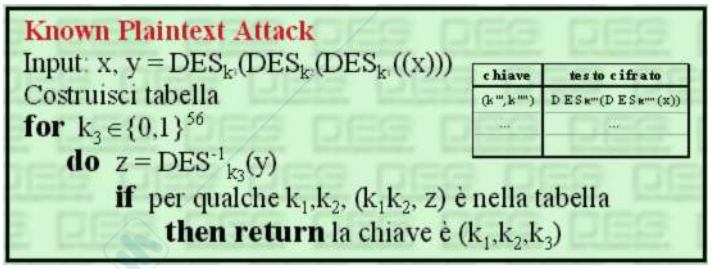


Figura 89: Costruzione tabella

La complessità in termini di spazio del suddetto algoritmo è 2^{112} (righe della tabella); mentre la complessità di tempo è di 2^{112} (per la costruzione della tabella) + 2^{56} (le cifrature di y) + 2^{56} (numero delle ricerche).

Il secondo metodo si differenzia dal primo per il punto in cui viene portato l'attacco (Figura 90):

- Si costruisce una tabella di dimensione 2^{56} la quale contiene coppie del tipo $(k_1, DES_{k_1}(x))$ che rappresentano l'insieme di tutte le possibili cifrature di x mediante k_1.
- Successivamente si effettuano tutte le 2^{112} possibili decifrature di y con le chiavi k_2 e k_3, e, per ognuna di esse, si ricerca il valore ottenuto nella tabella.
- Quando la ricerca ha esito positivo si ottiene la chiave cercata che è (k_1, k_2, k_3).

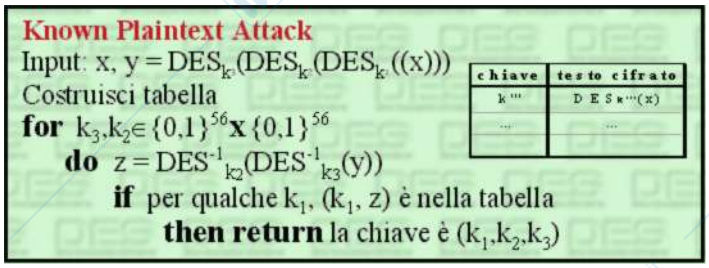


Figura 90: Costruzione tabella

La complessità in termini di spazio del suddetto algoritmo è 2^{56} (righe della tabella); mentre la complessità di tempo è di 2^{56} (per la costruzione della tabella) + 2^{112} (le cifrature di y) + 2^{112} (numero delle ricerche). Quindi, per quanto riguarda il DES triplo, la complessità relativa al Known Plaintext Attack è 2^{112}, mentre quella relativa alla ricerca esaustiva su tutte le chiavi è 2^{168}. É "equivalente" ad un cifrario con una chiave di 112 bit, e non di 168 bit.

Il DES triplicato può essere ulteriormente modificato sostituendo il secondo passo di cifratura con un passo di decifratura, sostituendo inoltre k_3 con k_1. Pertanto, dato il testo in chiaro x, si ha: $y = DES_{k_1}(DES^{-1}_{k_2}(DES_{k_1}(x)))$.

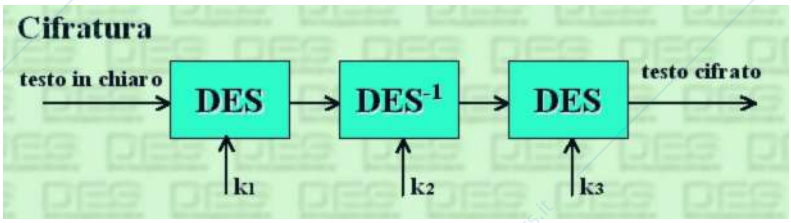


Figura 91: Triplo DES

Questo origina l'algoritmo detto Triplo DES utilizzato anche per cifrare e decifrare dati col DES singolo; infatti, dato x, vale la seguente uguaglianza: $y = DES_k(DES^{-1}_k(DES_k(x))) = DES_k(x)$ (al secondo membro la cifratura e la decifratura più esterne si annullano). Analizziamo, infine, la decifratura del Triplo DES. Dato il testo cifrato y e la coppia di chiavi (k_1, k_2), per ottenere il testo in chiaro x dobbiamo effettuare le operazioni inverse alla cifratura, cioè dobbiamo decifrare y con la chiave k_1, quindi cifrare il valore ottenuto con la chiave k_2 ed infine decifrare il risultato utilizzando di nuovo la chiave k_1: $x = DES^{-1}_{k_1}(DES_{k_2}(DES{k_1}^{-1}y)))$.

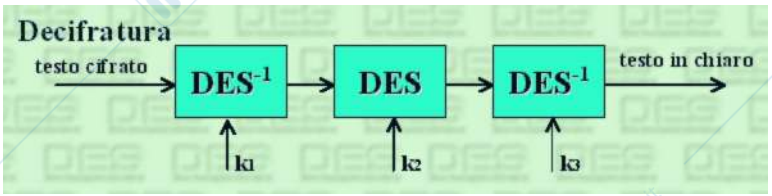


Figura 92: Triplo DES

Questa ultima versione del Triplo DES è spesso utilizzata in pratica in alternativa al DES ed il suo grado di sicurezza è considerato piuttosto alto; lo svantaggio è che risulta meno efficiente del DES singolo di un fattore 3. Attualmente non sono noti attacchi crittoanalitici pratici al Triplo DES. Coppersmith ha osservato che un attacco di forza bruta per la ricerca della chiave è dell'ordine di $2^{112} = 5 \times 10^{35}$ (complessità pari a quella dell'attacco meet in the middle) e ha stimato che il costo della crittoanalisi differenziale cresce in modo esponenziale rispetto al DES singolo, ed è maggiore di 10^{52}. Il Triplo DES è stato adottato negli standard per il trattamento delle chiavi ANSI X9.17 ed ISO 8732, e nel PEM (Privacy Enhanced Mail).

Il NIST ha approvato l'uso di Triple DES fino all'anno 2030 per informazioni governative sensibili. L'algoritmo è anche descritto nell'ANSI X3.92, nel NIST SP 800-67[1e nell'ISO/IEC 18033-3 (come un componente di TDEA). Nei documenti del NIST invece di DES si trova il termine DEA. Viene distinto l'Algoritmo dallo Standard. TDEA è $DES_{k_3}(DES^{-1}_{k_2}(DES_{k_1}(X)))$. Nel caso in cui $k_1 = k_3$ si è in presenza del Two-key Triple Data Encryption Algorithm (2TDEA); mentre nel caso in cui $k_1 \neq k_2 \neq k_3$, si parla di Three-key Triple Data Encryption Algorithm (3TDEA). Nella pubblicazione NIST Special Publication 800-57 Recommendation for Key Management Part 1 Revision 4, January 2016 è indicato come utilizzare TDEA. Dopo il 31 dicembre 2015 non si può più utilizzare 2TDEA per cifrare, si deve usare 3TDES. In SP800-57 la sicurezza di xTDES è stimata in:

- 2TDEA 80 bit (chiave di $112 = 56 \times 2$ bit)
- 3TDEA 112 bit (chiave di $168 = 56 \times 3$ bit)

6.2.1 DES-X

In crittografia il DES-X (o anche DESX) è una variante del cifrario a blocchi DES (Data Encryption Standard) proposta da Ronald Rivest nel 1984 e sviluppata per incrementarne la resistenza agli attacchi a forza bruta attraverso una tecnica chiamata key whitening. Ronald Rivest presentò perciò il DES-X allo scopo di incrementare la dimensione della chiave del DES senza alterarne la struttura: il suo algoritmo fu inserito nella libreria crittografica BSAFE da RSA Security alla fine degli anni ottanta. Questa tecnica, a differenza della cifratura multipla, non rafforza il cifrario contro attacchi analitici (ad esempio, crittoanalisi differenziale e lineare). Il DES-X differisce dal DES originale per due operazioni di XOR applicate al testo in chiaro prima e dopo l'esecuzione del DES: questa tecnica, denominata key whitening ed utilizzata anche in altri cifrari (ad esempio nel Twofish), serve ad incrementare la sicurezza dell'algoritmo senza appesantirne eccessivamente i calcoli. Nel DES-X, per ogni operazione di XOR vengono utilizzati 64 bit della chiave: $\text{DES-X}(M) = K_2 \oplus \text{DES}_K(M \oplus K_1)$, dove:

- K_1: 64 bit della chiave applicati al testo in chiaro prima di eseguire il DES;
- K_2: 64 bit della chiave applicati al testo in chiaro dopo aver eseguito il DES;
- M: messaggio in chiaro;
- DES_k: algoritmo DES con chiave k.

Un attacco banale di forza bruta richiede tempo proporzionale a 2^{t+2n}. Un attacco meet-in-the-middle richiede tempo proporzionale a 2^{t+n} e spazio proporzionale a 2^n. Se un avversario può collezionare 2^m coppie (plaintext, ciphertext), esiste un attacco evoluto che richiede tempo proporzionale a 2^{t+n-m}. Nel caso del DES ($t+2n = 56+2{\cdot}64 = 184$), se l'avversario colleziona 2^{32} coppie, l'attacco richiede tempo proporzionale a $2^{56+64-32} = 2^{88}$. L'attaccante deve collezionare circa 32GB di dati.

7 Altri cifrari simmetrici

7.1 Blowfish

Blowfish è un cifrario a blocchi a chiave simmetrica sviluppato da Bruce Schneier nel 1993. Questo algoritmo utilizza varie tecniche tra le quali la rete Feistel, le S-box dipendenti da chiavi e funzioni F non invertibili che lo rendono, forse, l'algoritmo più sicuro attualmente disponibile. Le chiavi utilizzate sono di dimensioni variabili fino ad un max. di 448 bit (14 word da 32 bit) e i blocchi utilizzati per la cifratura sono di 64 bit. Non si conoscono al momento tecniche di attacco valide nei suoi confronti. É considerato uno degli algoritmi di cifratura a blocchi più veloce (risulta più veloce di DES e IDEA). Blowfish non è brevettato ed è di dominio pubblico. La procedura consiste in due fasi: una fase di espansione della chiave e una fase di cifratura dei dati. L'espansione chiave converte una chiave di al più 448 bit in un'array di sottochiavi per un totale di 4168 byte. L'algoritmo si compone di 16 round, in ciascuno dei quali si utilizza una sottochiave, più un round finale, in cui è previsto l'utilizzo delle sottochiavi K_{17} e K_{18}. Le sole operazioni usate sono XOR, somme su parole a 32-bit e quattro letture per round in array di dati indicizzati. Si tratta di un algoritmo veloce (su microprocessori a 32 bit opera a 18 cicli di clock per byte). La maggiore velocità è ottenuta attraverso la rimozione di ogni permutazione (operazione che non introduce alcun ritardo se realizzata via hardware, ma abbastanza dispendiosa se realizzata via software) e l'introduzione di alcune modifiche alle

funzionalità offerte da una rete di Feistel. É compatto (bastano meno di 5Kb di memoria) e semplice da implementare e da analizzare. Viene implementato in numerosi prodotti software.

7.1.1 Espansione della chiave

Converte una chiave di al più 14 word a 32 bit (K-array) in un array di 18 sottochiavi a 32 bit (P-array). Genera 4 S-box, ognuna costituita da 256 sottochiavi a 32 bit. Le sottochiavi sono calcolate usando la seguente procedura:

1. Inizializzare nell'ordine il P-array e, a seguire, le quattro S-boxes con una stringa fissa (con i valori della parte frazionaria di π).
2. Fare lo XOR tra P1 e i primi 32 bit della chiave, poi lo XOR tra P2 e i successivi 32-bit della chiave, e così via per tutti i bit della chiave (eventualmente fino a P14 in quanto la chiave simmetrica è lunga al più 448 = 14*32 bit). Ripetere il ciclo fino a che non è stato eseguito lo XOR dell'intero P-array con i bit della chiave.
3. Sia $E_{P,S}[Y]$ la cifratura di Y con il P-array e le S-box. Calcola:
   - $P_1, P_2 = E_{P,S}[0]$
   - $P_3, P_4 = E_{P,S}[P_1||P_2]$
   - ...
   - $P_{17}, P_{18} = E_{P,S}[P_{15}||P_{16}]$
   - $S_{1,0}, S_{1,1} = E_{P,S}[P_{17}||P_{18}]$
   - ...
   - $S_{4,254}, S_{4,255} = E_{P,S}[S_{4,252}||S_{4,253}]$

In parole più semplici, la lista delle chiavi di blowfish viene generata caricando i valori iniziali del P-array e delle S-Box con rappresentazioni esadecimali delle cifre di pi greco, che apparentemente non hanno periodicità o schemi ripetitivi. La chiave segreta viene quindi messa in XOR con ciascun elemento del P-array (ripetendo la chiave, quando necessario). Un blocco da 64bit di valore zero viene criptato con l'algoritmo stesso, e il risultato cifrato sostituisce gli elementi P-1 e P-2. Il risultato cifrato viene quindi codificato nuovamente con le nuove sottochiavi, e va a sostituire gli elementi P-3 e P-4. Questo procedimento continua fino a rimpiazzare tutti gli elementi del P-array e delle S-Box. In tutto, l'algoritmo di cifratura di blowfish viene eseguito 521 volte in modo da generare tutte le sottochiavi. Poiché ogni iterazione genera due sottochiavi, sono necessarie 521 iterazioni per generare tutte le sottochiavi (18/2 iterazioni per generare il P-array e 256/2 iterazioni per ognuna delle quattro S-box). Blowfish non è adatto per applicazioni in cui la chiave cambia frequentemente. P e S possono essere precalcolati e memorizzati (almeno 4Kb).

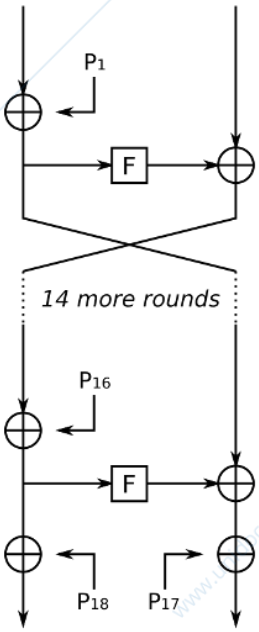


Figura 93: Funzionamento Blowfish

La Figura 93 mostra come opera blowfish. Ogni linea rappresenta 32 bit. L'algoritmo gestisce due tipi di liste di sottochiavi: una lista di 18 elementi, definita P-array, e quattro liste da 256 elementi ciascuna di S-Box (S0, S1, S2 e S3), per un totale di 5 array di sottochiavi. Ciascuna S-Box ha in ingresso 8 bit di informazioni, e produce in uscita 32 bit. A ogni ripetizione del ciclo si usa un elemento diverso del P-array, e dopo l'ultimo ciclo a ogni metà del blocco di dati viene moltiplicata in XOR con uno dei due elementi inutilizzati del P-array. Il diagramma della didascalia rappresenta la funzione F di blowfish. La funzione divide 32 bit in ingresso in quattro 8 bit, utilizzati a loro volta come ingressi delle S-Box. I risultati sono sommati in modulo 2^{32} e messi in XOR, per ottenere il risultato finale di 32 bit cifrati. Dato che blowfish è una rete di Feistel, può essere invertito semplicemente mettendo in XOR gli elementi 17 e 18 del P-array, e quindi utilizzando gli elementi del P-array in ordine inverso. La Figura 94 rappresenta il codice di funzionamento dell'algoritmo blowfish, dove L_0R_0 rappresenta il testo in chiaro; mentre $L_{17}R_{17}$ rappresenta il testo cifrato.

For i=1 to 16 do
$R_i = L_{i-1} \oplus P_i$;
$L_i = F[R_i] \oplus R_{i-1}$;
$L_{17} = R_{16} \oplus P_{18}$;
$R_{17} = L_{16} \oplus P_{17}$;

Figura 94: Funzionamento algoritmo Blowfish

La Figura 95 riporta uno schema delle quattro azioni di Blowwfish.

Azione 1	XOR la metà sinistra (L) dei dati con la *r* th entrata P-array
Azione 2	Utilizzare i dati XORed come input per F-funzione di Blowfish
Azione 3	XOR uscita del F-funzione con la metà destra (R) dei dati
Azione 4	Swap L e R

Figura 95: Funzionamento algoritmo Blowfish

L'input a 32 bit di F viene diviso in 4 byte a, b, c, d: $F[a,b,c,d] = ((S_{1,a} + S_{2,b}) \oplus S_{3,c}) + S_{4,d}$. Ciascuna fase include somme modulo 2^{32}, XOR e sostituzioni con la S-Box.

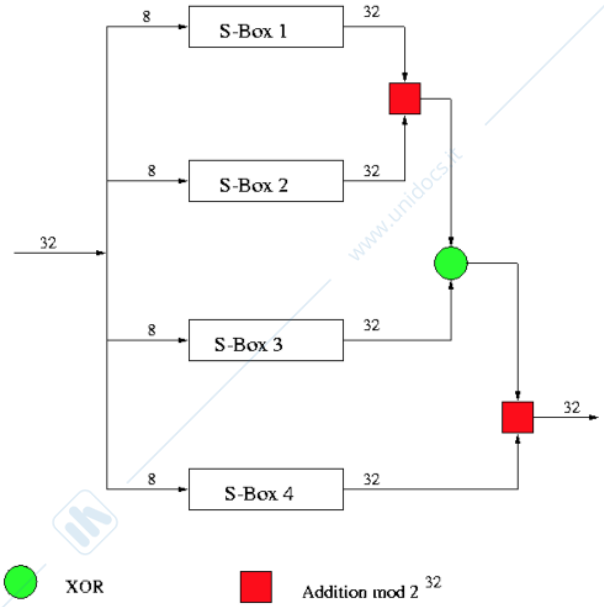


Figura 96: Funzionamento della funzione F

La decodifica funziona allo stesso modo della codifica ad eccezione dell'ordine in cui vengono usate le sottochiavi P1, P2,..., P18 che è invertito. La Figura 97 raffigura l'algoritmo di decodifica, dove: L_0R_0 rappresenta il testo cifrato; mentre $L_{17}R_{17}$ rappresenta il testo in chiaro.

For i=1 to 16 do
$R_i = L_{i-1} \oplus P_{19-i}$;
$L_i = F[R_i] \oplus R_{i-1}$;
$L_{17} = R_{16} \oplus P_1$;
$R_{17} = L_{16} \oplus P_2$;

Figura 97: Algoritmo di decifratura

Le caratteristiche dell'algoritmo sono:

- Sia le sottochiavi che le S-box dipendono dalla chiave, mentre in DES le S-box sono fissate
- In ogni round le operazioni coinvolgono tutto il blocco, mentre in DES, solo la parte destra.
- Risulta invulnerabile ad attacchi di forza bruta (se la dimensione della chiave è 14 word): per testare una sola chiave sono necessarie 522 esecuzioni dell'algoritmo. Il Blowfish è stato presentato per la prima volta da B. Schneier sulle pagine del noto Dr. Dobb's Journal nel

1994. La stessa rivista ha poi sponsorizzato una gara di crittoanalisi (1000 dollari di premio per il vincitore) terminata nell'aprile del 1995. In questo contesto Jon Kelsey ha sviluppato un attacco capace di rompere una versione del Blowfish che lavora con solo 3 round, ma non è riuscito ad estendere tale attacco al Blowfish completo. Vikramijit Singh Chhabra ha creato una macchina in grado di rendere efficiente la ricerca della chiave tramite un attacco di tipo "forza bruta", ma senza evidenziare rischi reali per l'algoritmo. I risultati più interessanti sono stati ottenuti da Serge Vaundenay che, oltre a definire attacchi a versioni modificate del Blowfish, ha evidenziato la presenza di chiavi deboli nell'algoritmo.

Sebbene ci sia una fase di inizializzazione molto complessa (la costruzione del P-array e delle S-box che comunque può essere precomputata), le fasi di codifica e decodifica dei dati sono molto veloci come viene evidenziato dalla Figura 98 che confronta il Blowfish con alcuni dei cifrari a chiave simmetrica più conosciuti ed usati.

Algorithm	Clock Cycles per round	# of rounds	# of clock cycles per byte encrypted
Blowfish	**9**	**16**	**18**
RC5	12	16	23
DES	18	16	45
IDEA	50	8	50
Triple-DES	18	48	108

Figura 98: Confronto con cifrari a chiave simmetrica

7.2 RC5

In crittografia l'RC5 è un algoritmo di cifratura a blocchi progettato da Ronald Rivest nel 1995. É degno di nota per la sua semplicità e perché una sua evoluzione (l'RC6) è stato fra i candidati per l'Advanced Encryption Standard. Al contrario della maggior parte degli algoritmi di cifratura a blocchi, nei quali è parametrizzabile al più una delle costanti dell'elaborazione, solitamente la dimensione della chiave, nell'RC5 tutti i parametri di base sono variabili: è possibile infatti scegliere fra diverse dimensioni del blocco (32, 64 o 128 bit), della chiave (da 0 a 255 byte) e del numero di passaggi o rounds (da 0 a 255). Non tutti i valori ammessi portano ad un risultato utilizzabile; ad esempio, se la dimensione della chiave o il numero di passaggi è 0, praticamente non avviene nessuna elaborazione. La combinazione suggerita dall'autore è di 12 passaggi con una chiave da 128 bit e con blocchi da 64 bit. Una delle operazioni centrali dell'RC5 consiste in una rotazione dei bit dei dati di un valore dipendente dal dato di ingresso; si può anche affermare che il cifrario stesso nacque proprio per studiare le proprietà crittografiche di questa operazione. In ogni ciclo si effettuano anche operazioni di OR esclusivo e addizioni modulari, organizzate in una rete di Feistel esprimibili con poche righe di codice. Il gestore della chiave è, invece, più complesso: esso espande la chiave utilizzando una funzione monodirezionale che utilizza le espansioni binarie di e e della sezione aurea come sorgenti di "numeri dalle proprietà non nascoste" (o nothing up my sleeve numbers" in inglese).

La caratteristica principale dell'RC5 è quella di essere un algoritmo orientato all'architettura del computer sul quale è in esecuzione. In particolare, tutte le elaborazioni sono effettuate su

stringhe di lunghezza pari a quella della word della macchina, ossia dipendono dall'architettura del computer su cui sta girando il cifrario, con ovvi vantaggi in termini di prestazioni: le operazioni su stringhe di lunghezza pari a quella della word del computer richiedono meno cicli macchina). Un'altra caratteristica è quella di usare istruzioni "base", comuni alla maggior parte dei processori, in quanto opera su dati la cui lunghezza è identica a quella word, anche se la stessa può essere di lunghezza diversa (come già detto, la lunghezza di una word può variare in funzione dell'architettura del computer), ed anche rotazioni dipendenti dai dati.

L'RC5 è un algoritmo che usa poca memoria, quindi risulta adatto per dispositivi quali le smart card, ed è semplice da analizzare ed implementare; infatti l'RC5 è stato utilizzato in diversi prodotti, ad esempio le librerie crittografiche BSAFE e JSAFE di RSA Security.

Si può affermare che l'RC5 è una famiglia di cifrari dove ogni membro è indicato come RC5-w/r/b, con le lettere w/r/b che indicano i seguenti parametri:

- w, word, denota la lunghezza della word del computer (32, 64 o 128 bit). La dimensione del blocco è pari al doppio della lunghezza della word, o 2w, mentre tutte le funzioni interne dell'RC5 ricevono in ingresso e forniscono in uscita dati lunghi esattamente w bit.
- r, round, indica il numero di passaggi che varia da 0 a 255 (anche se, come è stato specificato, utilizzare 0 passaggi equivale a non eseguire il processo di cifratura per cui il numero minimo di passaggi è 1). L'RC5 utilizza un passaggio iniziale differente da tutti gli altri utilizzati durante il processo di cifratura; ogni passaggio utilizza 2 sottochiavi. Dalla chiave iniziale sono generate 2 sottochiavi per ogni passaggio comune, quindi 2r, e 2 sottochiavi per la fase iniziale, per un totale di 2r+2.
- b, byte, è la lunghezza della chiave espressa in byte, che varia da 0 a 255 (anche qui, utilizzare una chiave di 0 byte equivale a non effettuare nessuna operazione di cifratura). Il gestore della chiave genera 2r+2 sottochiavi ognuna lunga w bit.

Perciò quando si scrive RC5 w/r/b significa che si sta indicando il RC5 che lavora su 2 word di w bit con un numero di passaggi pari ad r e con una chiave lunga b byte.

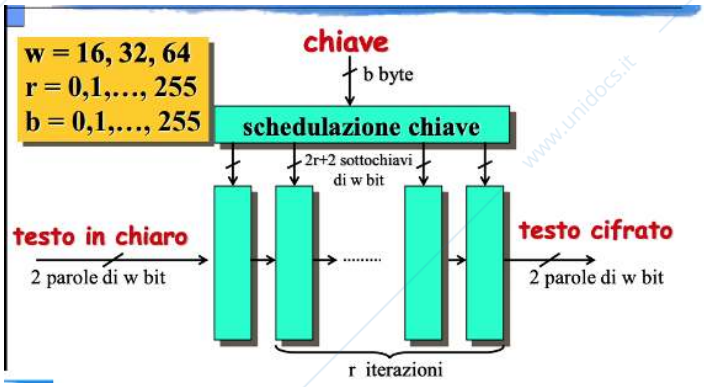


Figura 99: Algoritmo RC5

Come detto, le operazioni eseguite dall'RC5 sono tutte operazioni di w bit, e sono le seguenti:

- la somma $a + b$ modulo 2w;
- la sottrazione $a - b$ modulo 2w;
- $a \oplus b$: XOR bit a bit
- $a << b$, ossia shift a sinistra di b bit applicato alla word a

- $a >> b$, ossia rotazione a destra di b bit applicato alla word a

La prima operazione da vedere è la schedulazione della chiave. Partendo da una chiave di b byte, dove b è un parametro di RC5, immaginiamo che questi byte siano memorizzati in un array K: $K[0, ..., b-1]$. A partire da questa chiave l'algoritmo produce un altro array denominato S, l'array delle sottochiavi, perché in totale occorrono $2r+2$ sottochiavi: $S[0, \ldots, 2r+1]$. In effetti, per produrre S, che verrà poi usato nella fase di cifratura, occorre effettuare una conversione dai byte per ottenere delle word, dopo di che su queste ultime saranno eseguite delle operazioni per andare ad aggiornare il contenuto di S(funzione di mescolamento, o Mixing Function). Questo array intermedio è denominato L e si ottiene dall'array K. Su un computer ad architettura little-endian viene semplicemente copiato il contenuto dell'array K nell'array L; in totale, L avrà c locazioni, dove $c = \lceil (8 * b)/w \rceil$. Nel caso in cui $8 * b$ non dovesse essere un multiplo di w, saranno aggiunti degli "zero" per completare il valore (operazione di padding). La Figura 100 raffigura l'algoritmo di schedulazione della chiave.

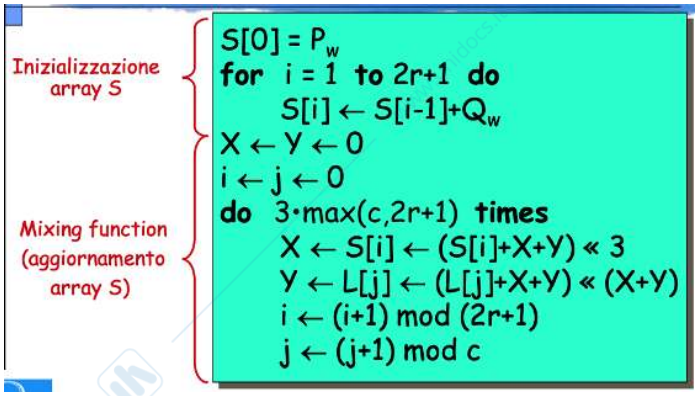


Figura 100: Codice algoritmo RC5

L'algoritmo di schedulazione della chiave è diviso in 2 fasi:

1. La prima fase inizializza l'array S (quello finale) con dei valori che dipendono da alcune costanti definite Pw e Qw, di w bit. Pw è l'espansione binaria a w bit del numero di Nepero (e=2,71828182459045... in decimale): in particolare $Pw = Odd[(e-2)2w]$ dove $Odd(x)$ indica l'intero dispari più vicino ad x. Qw è l'espansione binaria a w bit del rapporto aureo ($\phi = 1,6180339887...$ in decimale): in particolare, $Qw = Odd[(\phi - 1)2w]$. Questi valori sono già stati calcolati e memorizzati in una tabella, ed il loro valore dipende da w. In seguito l'array S viene modificato usando l'array L, che contiene la chiave. In particolare, nella prima locazione di S viene posto Pw, e poi, dalla locazione 1 fino a $2r+1$, viene aggiornata la locazione $S[i]$ andando a sommare la costante Qw alla locazione immediatamente precedente. La chiave non è stata ancora usata.

2. La seconda fase, chiamata Mixing Function, sfrutta l'array L, nel quale è stata copiata in precedenza la chiave: viene eseguito un ciclo che è eseguito 3 volte il numero massimo di elementi contenuto tra c e $2r+2$. Ad ogni ciclo S e L vengono aggiornati. Vengono usati due puntatori i e j: S è aggiornato in base al valore attuale di $S[i]$ e a 2 word a w bit denominate X e Y, inizialmente poste a 0, e poi con una rotazione a sinistra di 3 locazioni; L viene aggiornato in base al valore attuale di $L[j]$ e a 2 word a w bit denominate X e Y e poi una rotazione che dipende dai dati, cioè la rotazione dipende dal valore di $X+Y$. Dopo di che, ogni volta, gli indici i e j vengono incrementati.

In pratica l'array S che era stato inizializzato con le costanti Pw e Qw subisce degli aggiornamenti che dipendono dall'array L, array di c word ottenuto dalla chiave iniziale. Il risultato finale

equivale ad un array S aggiornato, array di $2r + 2$ locazioni, ognuna di w bit. Quindi, la fase di schedulazione della chiave prende in input una chiave di b byte e produce $2r + 2$ sottochiavi, ognuna di w bit.

w	16 bit	32 bit	64 bit
P_w	b7 e1	b7 e1 51 63	b7 e1 51 62 8a ed 2a 6b
Q_w	9E 37	9E 37 79 b9	9E 37 79 b9 7f 4a 7c 15

Figura 101: Pw e Qw

La fase di cifratura parte da un blocco di 2 word di w bit e produce un altro blocco di 2 word di w bit, tramite r passaggi. Vi è un passaggio iniziale che usa 2 sottochiavi e poi ogni passaggio successivo utilizza altre 2 chiavi sottochiavi, per un totale di 2r+2. Nell'algoritmo, in pratica, sono svolte operazioni di addizione su word di w bit, operazioni di XOR e rotazioni dei bit. Il testo in chiaro è memorizzato in 2 word denominate A e B, ognuna di w bit, in cui viene restituito anche l'output dell'algoritmo. Si aggiorna A con la prima sottochiave e B con la seconda sottochiave (le prime 2 sottochiavi utilizzate), dopo di che vengono eseguiti r passaggi in cui vengono usate 2 diverse sottochiavi S[2i] e S[2i+1] (altre 2r sottochiavi). Durante un singolo passaggio viene eseguita prima l'operazione di XOR bit a bit tra A e B, poi una rotazione di B, ed infine una addizione con la sottochiave $S[2i]$ col il risultato finale che aggiorna A. Inoltre, nel passaggio viene eseguito anche uno XOR bit a bit tra B ed A, poi viene eseguita una rotazione a sinistra che dipende dal valore di A, ed infine una somma con la sottochiave S[2i+1], aggiornando il valore di B. A differenza del DES e dei cifrari di Feistel, nell'RC5 entrambe le metà del blocco dati sono aggiornate nel passaggio, sia la parte A sia la parte B. Nei cifrari di Feistel invece una parte è ricopiata e si aggiorna l'altra metà del blocco. La decifratura è l'operazione inversa della cifratura: infatti sono svolti prima gli r passaggi a partire dall'ultimo al primo e poi gli ultimi 2 aggiornamenti. L'unica differenza tra la fase di cifratura e quella di decifratura è che in quest'ultima le operazioni sono di sottrazione invece che di somma; restano invariate le operazioni di XOR e di rotazione.

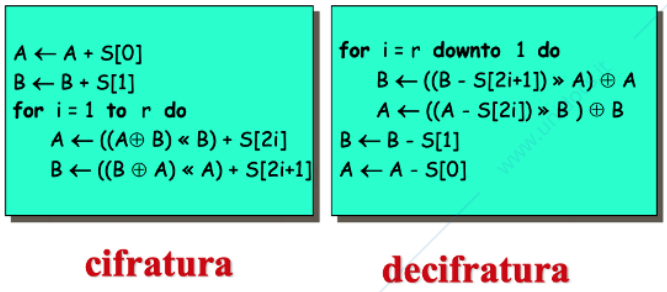


Figura 102: Cifratura e decifratura RC5

Dobbiamo anche ricordare che RC5 si basa su una macchina con un'architettura little-endian (il bit meno significativo è posto a destra; tipicamente processori Intel); in realtà può funzionare anche su una macchina con architettura big endian (ovvero, tipicamente processori Motorola), ma il procedimento d'espansione della chiave ha bisogno di alcune modifiche:

- Little-endian (INTEL): il valore di $a_1a_2a_3a_4$ è $a_1 + a_2 2^8 + a_3 2^{16} + a_4 2^{24}$
- Big-endian (SPARC): il valore di $a_1a_2a_3a_4$ è $a_4 + a_3 2^8 + a_2 2^{16} + a_1 2^{24}$.

Le caratteristiche salienti di RC5 sono:

- le rotazioni sono dipendenti dai dati e non sono fisse; questo fatto complica gli attacchi di crittanalisi differenziale e lineare, proprio perché l'ammontare dello shift dipende dal dato;

- in ogni passaggio le operazioni coinvolgono tutto il blocco, a differenza di quello che accadeva nel DES;
- l'RC5 è una famiglia che dipende dai parametri w/r/b, quindi per specificare un particolare cifrario dobbiamo indicare tali valori.

Alcune modalità operative di RC5 sono:

- RC5 Block Cipher (ECB): Blocco input fisso di 2w bit
- RC5-CBC: Blocchi concatenati multipli di 2w
- RC5-CBC PAD: Padding del blocco input
- RC5-CTS

7.3 Stream Cipher - Cifrari a flusso

In crittografia un cifrario a flusso (detto anche cifrario a caratteri) è un cifrario simmetrico nel quale i simboli (i bit) che codificano il testo in chiaro sono cifrati indipendentemente l'uno dall'altro e nel quale la trasformazione dei simboli successivi varia con il procedere della cifratura. Un altro termine usato per tale cifrario è cifrario a stati, termine che ricorda che la cifratura di ogni simbolo dipende da uno stato corrente. Tipicamente nella pratica, i simboli sono singoli bit o byte. Nei cifrari a flusso cifra il messaggio un byte (o bit) alla volta e utilizza una sequenza (keystream) pseudo-casuale generata a partire dalla chiave (piccola). Combina la keystream con il messaggio (XOR) bit a bit.

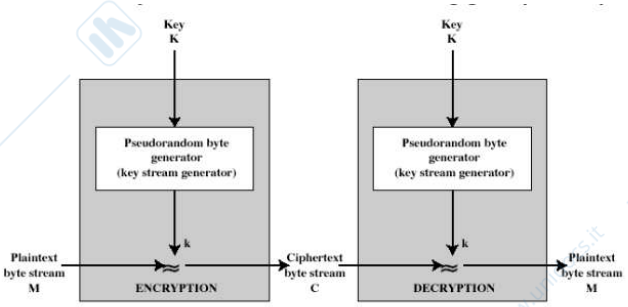


Figura 103: Stream Cipher

Si tratta di un algoritmo molto più veloci dei cifrari a blocchi, perchè composto da poche linee di codice e operazioni semplici. Per complicare la crittoanalisi si utilizzano keystream con lungo periodo (più tardi inizia a ripetersi, meglio è). Le keystream hanno le stesse caratteristiche di una sequenza casuale. In crittografia l'RC4 è uno tra i più famosi e diffusi algoritmi di cifratura a flusso a chiave simmetrica, utilizzato ampiamente in protocolli quali l'SSL ed il WEP. L'RC4 genera un flusso di bit pseudo casuali (keystream); tale flusso è combinato mediante un'operazione di XOR con il testo in chiaro (plaintext) per ottenere il testo cifrato. L'operazione di decifratura avviene nella stessa maniera, passando in input il testo cifrato ed ottenendo in output il testo in chiaro (questo perché lo XOR è un'operazione simmetrica). La Figura 104 mostra un confronto tra i principali schemi di crittografia simmetrici.

Algoritmo	Chiave (bit)	Velocità (Mbit/s)
RC4	8-2048	45
RC2	8-1024	0.9
DES	56	9
Triple-DES	168	3

Figura 104: Confronto con altri sistemi

7.3.1 Generazione di un keystream

In crittografia con il termine keystream si indica un flusso di caratteri pseudo-casuali che sono combinati con il messaggio in chiaro per produrre il messaggio cifrato. I "caratteri" nel keystream possono essere bit, byte, numeri o caratteri alfanumerici, a seconda dei casi. I keystream sono alla base dei cifrari a flusso (stream cipher) ma sono di solito utilizzati anche nei cifrari one-time pad. I cifrari a blocchi possono essere impiegati per produrre dei keystream utilizzandoli in modalità CTR: in questa modalità il cifrario a blocchi si trasforma in un vero e proprio cifrario a flusso. Il keystream, o flusso chiave, è generato da un apposito algoritmo del cifrario, che serve a creare il flusso di caratteri pseudo-casuali che saranno utilizzati per la cifratura, normalmente inizializzato utilizzando la chiave di cifratura fornita dall'utente; in genere viene anche utilizzato un nonce, un valore numerico introdotto nel keystream in modo da variare il flusso di dati. Infatti, un noto problema di questo modo di operare è che un generatore di keystream produrrà flussi di dati identici con chiavi identiche: ecco perché è consigliabile sempre, con i cifrari a flusso, cambiare la chiave segreta ad ogni operazione di cifratura oppure introdurre un nonce per variare il keystream. Per la sicurezza stessa del cifrario, è necessario che il keystream risponda a tre requisiti fondamentali:

- deve apparire il più casuale possibile;
- deve avere un periodo lungo almeno quanto il testo da cifrare;
- non deve avere sequenze di caratteri che si ripetono periodicamente.

É ovvio che non è possibile utilizzare un keystream completamente casuale dato che il keystream deve essere identico sia per l'operazione di cifratura che per quella di decifratura: se si avesse un keystream veramente casuale, non si potrebbe riottenere il testo in chiaro originale dato che anche usando la stessa chiave segreta il generatore di keystream produrrebbe un flusso totalmente differente da quello creato per la cifratura. Ecco perché il flusso deve apparire il più casuale possibile agli occhi di un crittoanalista ma essere ripetibile: si parla in questo caso di pseudo-casualità. Relativamente alla lunghezza del periodo, è importante che il generatore non ripeta la stessa sequenza già generata prima che sia stato cifrato tutto il testo in chiaro, altrimenti diverse porzioni di quest'ultimo sarebbero cifrate con lo stesso flusso di dati offrendo ad un crittoanalista utili informazioni per la decrittazione del testo cifrato. Ma se anche la lunghezza del periodo è di importanza vitale, non è però l'unico elemento che ne determina la sicurezza: è anche importante che non presenti sequenze periodiche o troppi bit di valore zero o uno. Se infatti un generatore produce bit zero per la quasi totalità del keystream il testo in chiaro sarà in questo caso quasi identico al testo cifrato. Ma anche la ripetitività è un fattore che diminuisce la sicurezza del keystream: un crittoanalista può infatti selezionare una sequenza di bit e predire i dati che saranno generati in seguito.

Supponiamo di avere una stringa binaria di m bit $k_1, ..., k_m$ e sia $z_i = k_i$ per $i = 1, ..., m$. Posso definire il keystream usando una ricorrenza lineare di grado m: $z_{i+m} = \sum_{i=0}^{m-1} c_j z_{i+j}($ (mod 2)), dove $c_0, ..., c_{m-1}$ sono costanti

- di grado m perché ogni termine dipende dagli m valori precedenti
- lineari perché ogni termine è combinazione lineare di altri termini
- c_0 è sempre 1 altrimenti l'equazione sarebbe di grado al più $m-1$

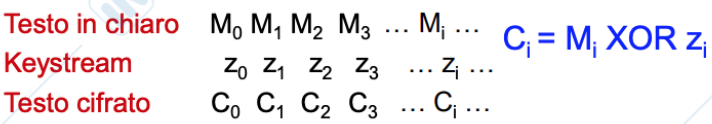


Figura 105: Generazione di keystream

Un altro metodo di generazione di flusso di chiavi è linear feedback shift register (LFSR). Il registro a scorrimento a retroazione lineare (linear feedback shift register, LFSR) è una tipologia di registri di traslazione i cui dati in ingresso sono prodotti da una funzione lineare dello stato interno. Le uniche funzioni lineari di singoli bit sono lo XOR e lo XNOR (xor inverso); perciò è un registro di traslazione i cui bit in ingresso sono prodotti dall'or esclusivo (xor) di alcuni bit memorizzati all'interno dei registri. Il valore iniziale di un LFSR è chiamato seme, e poiché l'operazione del registro è deterministica, la sequenza di valori prodotta dal registro è completamente determinata dal suo stato corrente o precedente. Allo stesso modo, poiché il registro ha un numero finito di stati possibili, prima o poi i valori in uscita si ripetono; ciò nonostante, un LFSR con una funzione di retroazione ben scelta può produrre una sequenza di bit che appare casuale ed ha un periodo molto lungo. Consideriamo una chiave di 2m valori: $k_1, ..., k_m, c_0, ..., c_{m-1}$. Una chiave piccola m genera un ciclo di $2^m - 1$ valori. Supponiamo di avere una chiave grande $m = 4$, la ricorrenza sarà dato da: $z_{i+4} = (z_i + z_{i+1}) mod 2$. In questo caso, il vettore inizializzazione, o seme, è 1000. La Figura 106 mostra un esempio dei primi 2 passaggi per il calcolo.

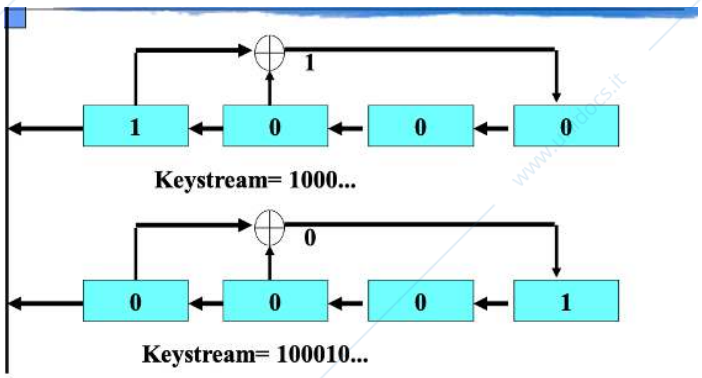


Figura 106: Funzionamento LFSR

LFSR non resiste ad attacchi known plaintext. Se l'avversario conosce un testo in chiaro di n caratteri e $n > 2m$, può computare la sequenza di inizializzazione (sistema di m equazioni in m incognite) e quindi l'intero keystream. Per ovviare questo problema si utilizza una combinazione non lineare di più bit dallo stato LFSR (Figura 107).

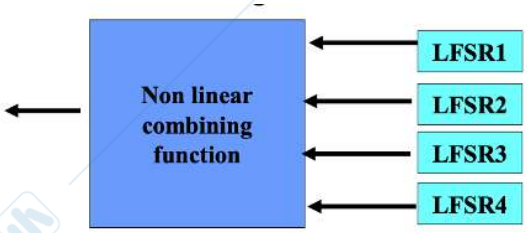


Figura 107: Funzionamento LFSR multiplo

7.3.2 RC4

RC4 è un cifrario a flusso progettato da Ron Rivest (la "R" di RSA) nel 1987. Era un segreto commerciale della RSA Security, ma nel 1994 è stato inviato in maniera anonima a una mailing list su Internet, e da allora è stato ampiamente analizzato. É utilizzato, tra l'altro, nel protocollo TLS/SSL (Transport Layer Security / Secure Sockets Layer) per la trasmissione sicura di pagine web e nel protocollo WEP (Wired Equivalent Privacy) all'interno dello standard IEEE 802.11b. É semplice e veloce. RC4 è una funzione che, a partire da una chiave (lunga da 1 a 256 byte), genera una sequenza pseudocasuale (keystream) utilizzata per cifrare e decifrare (mediante XOR) un flusso dati. Il keystream generato ha un periodo maggiore di 10^{100}. RC4 usa operazioni orientate ai byte.

RC4 mantiene come informazione di stato:

- un vettore di byte S (stato) contenente una permutazione degli interi da 0 a 255: $S[0]...S[255]$, che prevede un'inizializzazione $S[i] = i, i = 0, ..., 255$
- due contatori: i e j.

La chiave K (lunga da 1 a 256 byte) viene utilizzata solo in fase di inizializzazione dello stato. Il vettore S contiene in ogni istante una permutazione dei valori da 0 a 255. Quindi, genera la keystream da S, un byte alla volta: k rappresenta un byte della keystream. Effettua lo XOR di ciascun byte della keystream con un byte del testo in chiaro. Dopo aver generato ogni byte della keystream, permuta il vettore S. L'inizializzazione dell'algoritmo RC4 si compone di più fasi:

- Inizializzazione dello stato:
  - In S si inseriscono i valori da 0 a 255: $S[n] = n$;
  - In un altro vettore temporaneo T (di 256 byte) si inserisce la chiave K (ripetendola se più corta);

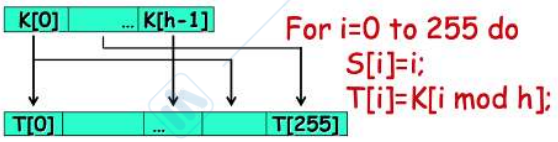


Figura 108: La chiave

  - I contatori i e j si inizializzano a zero;
  - Si percorre S scambiando l'elemento corrente (iesimo) con un altro determinato usando la chiave:

    for i = 0 to 255

    $j = (j + S[i] + T[i]) \pmod{256}$

$scambia(S[i], S[j])$

T è usato per aggiornare il vettore S. Produce una permutazione e dopo l'aggiornamento, la chiave non viene più usata.

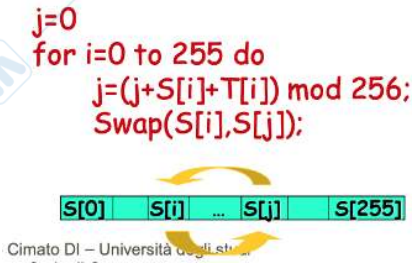


Figura 109: Aggiornamento

– Si reinizializzano i e j a zero e si scarta T (la chiave non viene più utilizzata).

- Generazione del keystream: Si percorre il vettore S, scambiando l'elemento corrente (i-esimo) con un altro determinato dallo stato corrente di S e j. Per generare un byte k del keystream, dallo stato corrente (S, i, j):

  $i = (i + 1) \pmod{256}$
  $j = (j + S[i]) \pmod{256}$
  $scambia(S[i], S[j])$
  $t = (S[i] + S[j]) \pmod{256}$
  $k = S[t]$

  Quando si raggiunge $S[255]$ si ricomincia da $S[0]$. Considerando S, i, j, RC4 può trovarsi in ben $256! \times 256^2$ (circa 2^{1700}, o $5,62^{511}$) stati.

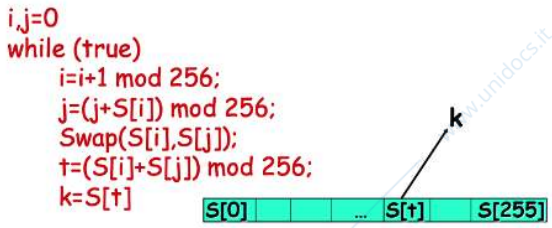


Figura 110: Generazione keystream

Quando si parla di crittografia l'operazione di XOR avviene tra k col successivo byte plaintext; mentre quando si parla di decrittografia, lo XOR avviene tra k col successivo byte cifrato.

L'RC4 paga la sua semplicità in termini di sicurezza: questa è molto debole e l'algoritmo è violabile con relativa facilità e velocità, tanto che il suo uso non è più consigliabile. L'RC4 mostra infatti un comportamento non da vero PRNG[3]: l'analisi del flusso di dati casuali denota una certa periodicità nei primi 256 byte.

[3]Numeri pseudo-casuali

7.3.3 Attacchi a WEP

RC4 è un algoritmo ben noto e molto utilizzato, e se usato correttamente è considerato sicuro. RC4 è utilizzato tra l'altro all'interno dello standard IEEE 802.11b per la cifratura delle trame sulla tratta radio (WEP - Wired Equivalent Privacy), l'utilizzo di RC4 fatto da WEP presenta però diversi problemi. Le stazioni condividono una chiave segreta condividono una chiave segreta K (40 bit, oppure 104 bit nel cosiddetto "WEP a 128 bit") con l'access point (può essere la stessa chiave per tutte le stazioni). Viene calcolata una checksum (CRC) sul payload. Il payload e il CRC vengono cifrati con RC4, utilizzando come chiave la concatenazione di K e di un vettore di inizializzazione (IV) di 24 bit. IV è scelto dal trasmettitore e inserito in chiaro nel pacchetto. Il ricevitore estrae IV, decifra il payload e la checksum, controlla la checksum.

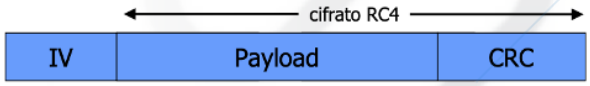


Figura 111: Chiavi WEP

La presenza di IV serve a cambiare il keystream. Ecco che cosa accade se due messaggi sono cifrati con lo stesso keystream: $C_1 = P_1 \oplus RC4(IV||K)$ e $C_2 = P_2 \oplus RC4(IV||K)$. $C_1 \oplus C_2 = P_1 \oplus RC4(IV||K) \oplus P_2 \oplus RC4(IV||K) = P_1 \oplus P_2$. Facendo lo XOR di due messaggi cifrati si ottiene lo XOR dei due messaggi in chiaro. Se si conosce uno dei due messaggi si ottiene l'altro. Se si hanno molti messaggi cifrati con lo stesso keystream le cose si fanno più semplici. In un cifrario a flusso non si deve riutilizzare il keystream. La presenza di IV serve a evitare i problemi visti, ma gli IV possibili sono $2^{24} = 16.777.216$ (pochi). Il numero degli IV è indipendente dalla lunghezza della chiave K (IV è comunque di 24 bit). Inoltre, IV è trasmesso in chiaro, quindi è facile vedere se due messaggi hanno lo stesso IV. In più: K cambia molto raramente (richiede riconfigurazione manuale) e spesso è comune a tutte le stazioni. Gli attacchi visti finora sono indipendenti da eventuali debolezze di RC4. In un articolo dell'agosto 2001 sono state però pubblicate delle debolezze di RC4 dalle quali WEP, per il modo in cui genera e usa le chiavi, è affetto (mentre TLS/SSL ad esempio non lo è). Si sfrutta il fatto che alcuni IV danno luogo a chiavi deboli (e gli IV viaggiano in chiaro...). Si sfrutta il fatto che si conosce l'inizio (primo byte) dei pacchetti in chiaro. Avendo a disposizione circa 5 milioni di pacchetti, si è solitamente in grado di ottenere la chiave segreta e l'attacco è totalmente passivo. Infatti, l'attacco si basa unicamente sulla conoscenza di:

- La prima word output di RC4 con il corrispondente IV
- In WEP IV viene trasmesso in chiaro
- La prima word per molti pacchetti è costante

WEP, pur utilizzando un algoritmo crittografico noto e studiato (RC4) lo fa in modo non corretto:

- chiavi corte e "simili" (la parte maggiore - K - è fissa);
- limitazione degli IV (i valori possibili sono troppo pochi);
- riutilizzo keystream.

La debolezza di RC4 risiede nella schedulazione della chiave: i primi bytes di output rivelano informazioni sulla chiave. Nel 2005, Andreas Klein ha presentato una analisi del RC4 stream mostrando ancora più correlazioni tra RC4 keystream e la chiave. Nel 2013, un gruppo di ricercatori presso Information Security Group at Royal Holloway, University of London ha riportato un attacco che può diventare effettivo usando solo 224 connections. L'uso di RC4 in TLS è

proibito secondo la RFC 7465 February 2015. Nel marzo 2015 I ricercatori di Royal Holloway hanno migliorato l'attacco richiedendo specifico per l'uso di RC4 in TLS.

L'eSTREAM è un progetto europeo sponsorizzato dal consorzio ECRYPT (European Network of Excellence for Cryptology) per identificare nuovi cifrari a flusso da utilizzare diffusamente. L'eSTREAM è stato avviato ufficialmente nel novembre del 2004 ed è durato 4 anni: ha portato al rilascio, nell'aprile del 2008 di un portafoglio di cifrari a flusso di nuova concezione (rivisitato nel settembre dello stesso anno), suddivisi in due grandi famiglie, una ottimizzata per l'implementazione in software ed una per quella in hardware. In Settembre 2011 eSTREAM portfolio contiene 7 algoritmi, tra cui SALSA20, un cifrario a flusso sviluppato da Daniel Bernstein nel 2005 e selezionato per il portafoglio del progetto eSTREAM per il Profilo 1 (implementazione software). È costruito su una funzione pseudo-casuale basata sull'addizione a 32 bit, sull'OR esclusivo (XOR) e sulle operazioni di rotazione dei bit, che utilizza chiavi a 256 bit (ma possono essere utilizzate anche chiavi a 128 bit) e nonce a 64 bit. La funzione lavora effettuando 20 passaggi, da cui il nome dell'algoritmo.

8 Crittografia asimmetrica

Il principale problema della crittografia simmetrica sta nella necessità di disporre di un canale sicuro per la trasmissione della chiave. Prevede, infatti, l'utilizzo di una canale privato o di una terza parte fidata, che stabilisca la chiave di sessione e la invia ad entrambi in modo sicuro. Inoltre, se n persone devono comunicare tra loro, ogni utente deve memorizzare $n-1$. Il numero totale di chiavi necessarie $n(n-1)/2$ chiavi, quindi il numero totale delle chiavi segrete è dell'ordine di $n^2/2$. L'aggiunta di un nuovo utente alla rete implica la distribuzione della chiave a tutti i precedenti utenti. Si giunse così all'invenzione della crittografia asimmetrica (detta anche "a chiave pubblica"). L'approccio infatti è completamente diverso da quello basato su sostituzioni e permutazioni che aveva imperversato dai tempi di Cesare fino ai giorni nostri (DES, AES, etc.). L'idea alla base della crittografia asimmetrica è quello di avere due chiavi diverse, una pubblica per la criptazione e una privata per la decriptazione, che deve essere mantenuta segreta. Fra due interlocutori, non vi è dunque la necessità di scambiarsi le chiavi. L'idea di base dei cifrari asimmetrici è l'utilizzo di una cassaforte con due lucchetti:

- Con una chiave (pubblica) chiudiamo la cassaforte
- Con l'altra chiave (privata) apriamo la cassaforte

Un utente cifra o verifica la firma; l'altro decifra o crea la firma. L'asimmetria deriva dal fatto che la chiave pubblica sia diversa da quella privata. Quindi, la comunicazione tra Alice e Bob avviene nel seguente modo:

- Alice invia a Bob un messaggio cifrandolo con la chiave pubblica di Bob
- Solo Bob può decifrarlo usando la corrispondente chiave privata

La Figura 112 raffigura uno schema di comunicazione tramite cifratura asimmetrica.

Le chiavi pubbliche risultano inutili senza la corrispettiva chiave privata, quindi si possono trasmettere su canali non sicuri. Chiunque può cifrare un messaggio per Bob, ma solo Bob può decifrare un messaggio cifrato per lui. Non ci sono chiavi condivise tra gli utenti: ogni utente genera da solo la propria coppia di chiavi (public key, private key) e rende pubblica la chiave pubblica. Ogni utente memorizza una sola chiave (privata). Il meccanismo si basa sul fatto che, se con una delle due chiavi si cifra (o codifica) un messaggio, allora quest'ultimo sarà decifrato solo con l'altra.

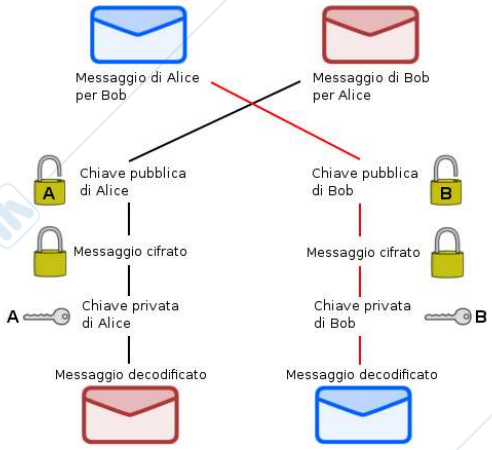


Figura 112: Crittografia asimmetrica

Risulta computazionalmente inammissibile trovare la chiave di decifratura dall'algoritmo e dalla chiave di cifratura; mentre è computazionalmente facile de/cifrare un messaggio conoscendo le rispettive chiavi.

Gli usi della crittografia asimmetrica si possono riassumere in tre categorie:

- encryption/decryption (segretezza)
- digital signatures (autenticazione): oltre alla cifratura dei dati di una comunicazione, la crittografia asimmetrica presenta altri possibili impieghi: firma digitale per verificare l'autenticazione del mittente e l'integrità informativa del messaggio, fornire una condizione di ending e per i programmi che tentano la forzatura delle chiavi. Un utente può firmare un messaggio utilizzando la propria chiave privata; per far ciò, viene creata un'impronta (digest) del messaggio da firmare e questa, criptata con la chiave privata, rappresenta la firma ed è inviata assieme al messaggio (l'impronta, generata per mezzo di un algoritmo di Hash, è tale che varia sensibilmente al minimo variare del messaggio).
- key exchange (per session keys)

Alcuni algoritmi sono adatti a tutti gli usi, altri solo ad alcuni.

La Figura 113 mostra le applicazioni possibili di diversi algoritmi basati sulla cifratura asimmetrica.

Algoritmo	De/Cifra	Firma	Scambio chiavi
RSA	Si	Si	Si
Curva Ellittica	Si	Si	Si
Diffie-Helmann	No	No	Si
DSS	No	Si	No

Figura 113: Algortimi e applicazioni

I cifrari simmetrici e quelli asimmetrici presentano ognuno i propri vantaggi:

- Vantaggi della crittografia a chiave privata (Cifratura simmetrica):
  - Molto più veloce (ad es., DES è $\geq$ 100 volte più veloce di RSA, in hardware tra 1.000 e 10.000 volte)

  - Sufficiente in diverse situazioni (ad esempio, applicazioni per singolo utente)

- Vantaggi della crittografia a chiave pubblica (Cifratura asimmetrica):
  - Key Distribution: non ho bisogno di un KDC, Chiavi private mai trasmesse
  - Possibile la firma digitale

Quindi, riassumendo, nei cifrari simmetrici vengono usati lo stesso algoritmo con la stessa chiave condivisa. La chiave è segreta ed è impossibile decifrare un messaggio senza chiave. La conoscenza del messaggio e del testo cifrato non dà la chiave. Mentre, nei cifrari asimmetrici si utilizza un unico algoritmo con una coppia di chiave: mittente e destinatario utilizzano una coppia di chiavi correlate, ma distinte. Una delle chiavi è segreta ed è impossibile decifrare un messaggio senza chiave. La conoscenza dell'algoritmo e di una delle chiavi non dà l'altra chiave.

8.1 Cifrari ibridi

Gli algoritmi a chiave pubblica non sono una panacea. Molti algoritmi simmetrici sono più forti dal punto di vista della sicurezza; le operazioni di criptazione e decriptazione a chiave pubblica sono più costose delle corrispondenti operazioni dei sistemi simmetrici. Ciò nonostante, gli algoritmi a chiave pubblica rappresentano uno strumento efficace per distribuire le chiavi degli algoritmi simmetrici e per questo vengono usati in sistemi di crittografia ibridi. Un algoritmo ibrido utilizza sia un sistema simmetrico che uno a chiave pubblica. In particolare esso funziona utilizzando un algoritmo a chiave pubblica per condividere una chiave per il sistema simmetrico. Il messaggio effettivo è quindi criptato usando tale chiave e successivamente spedito al destinatario. Poiché il metodo di condivisione della chiave è sicuro, la chiave simmetrica utilizzata è differente per ogni messaggio spedito. Per questo viene detta a volte chiave di sessione. Ogni utente ha una coppia di chiavi privata e pubblica. Ogni volta che il mittente apre una nuova sessione viene generata una chiave di sessione, la quale viene cifrata con la chiave pubblica del destinatario e inviata insieme al messaggio. Il destinatario, per prima cosa decifra la chiave di sessione usando la propria chiave privata asimmetrica, poi usa la chiave di sessione, simmetrica, per decodificare il messaggio. Il vantaggio di questo sistema è che unisce la comodità nella gestione delle chiavi del sistema asimmetrico alla velocità propria di quello simmetrico e consente uno scambio di chiavi sicuro anche su un canale non sicuro (ossia normalmente non crittografato e potenzialmente accessibile a tutti).

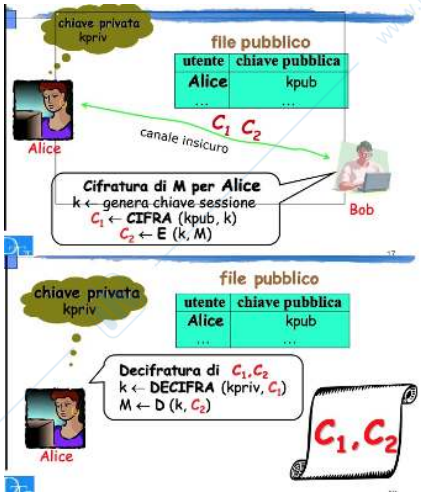


Figura 114: Funzionamento cifrari ibridi

Un problema computazionale è inteso come un obiettivo che sia, in via di principio, facilmente ottenibile da un calcolatore con sufficienti risorse a sua disposizione. In altre parole,

quanto enunciato è equivalente a dire che un problema computazionale può essere risolto mediante l'uso meccanico di passaggi matematici predefiniti (un algoritmo). Un problema è detto essere intrinsecamente difficile se la sua soluzione richiede una quantità significativa di risorse a prescindere dall'algoritmo usato. La teoria della complessità computazionale formalizza questa intuizione introducendo modelli matematici per studiare i problemi computazionali e quantificare le risorse richieste per risolverli. Le misure di complessità solitamente adottate sono Tempo e Spazio. Quanto richiesto da un crittosistema asimmetrico è un problema relativamente facile da creare, ma computazionalmente complicato da risolvere senza le giuste informazioni. Quindi si tratta di algoritmi "facili" da calcolare e "difficili" da invertire, a meno che non si conosca una "trapdoor". Una funzione f è pseudo-unidirezionale se appare come unidirezionale per chiunque non sia in possesso di una particolare informazione sulla sua costruzione (trapdoor). Nel contesto della crittografia asimmetrica, con "problema facile da creare" si intende che la funzione di cifratura/verifica di un crittosistema deve avere complessità polinomiale in tempo nella dimensione dell'input. Invece, con "problema computazionalmente complicato da risolvere senza le giuste informazioni" si intende che, usando una macchina deterministica, la funzione di decifratura/firma deve avere complessità polinomiale in tempo nella dimensione dell'input se e solo se si conosce la chiave di decifratura. In caso non si conosca la chiave privata, per un calcolatore deterministico non-quantistico, l'algoritmo di decifratura/firma deve avere complessità almeno esponenziale o sub-esponenziale. I Cifrari asimmetrici più comuni sono basati sulla Teoria dei Numeri. La teoria dei numeri computazionale studia algoritmi importanti nella teoria dei numeri. Algoritmi efficienti per la verifica della primalità e la fattorizzazione di interi hanno importanti applicazioni nella crittografia.

8.2 RSA

In crittografia la sigla RSA indica un algoritmo di crittografia asimmetrica, inventato nel 1977 da Ronald Rivest, Adi Shamir e Leonard Adleman utilizzabile per cifrare o firmare informazioni. É nel 1978 che questo sistema trova la sua applicazione reale, quando i tre ricercatori del MIT Ronald Rivest, Adi Shamir e Leonard Adleman seppero implementare tale logica utilizzando particolari proprietà formali dei numeri primi con alcune centinaia di cifre. L'algoritmo da loro inventato, denominato RSA dalle iniziali dei loro cognomi, non è sicuro da un punto di vista matematico teorico, in quanto esiste la possibilità che tramite la conoscenza della chiave pubblica si possa decifrare un messaggio; ma l'enorme mole di calcoli e l'enorme dispendio in termini di tempo necessario per trovare la soluzione fanno di questo algoritmo un sistema di affidabilità pressoché assoluta. RSA è il più noto e diffuso schema a chiave pubblica.

RSA è basato sull'elevata complessità computazionale della fattorizzazione in numeri primi. La sicurezza si basa sulla difficolta di fattorizzare grandi interi. La fattorizzazione prende $O(e^{lognloglogn})$ operazioni.

8.2.1 Generazione delle chiavi

Ogni utente genera una coppia di chiavi pubblica/privata:

1. scelgono a caso due numeri primi, p e q abbastanza grandi da garantire la sicurezza dell'algoritmo (ad esempio, il più grande numero RSA, RSA-2048, utilizza due numeri primi lunghi più di 300 cifre).

2. si calcola il loro prodotto $n = pq$, chiamato modulo (dato che tutta l'aritmetica seguente è modulo n), e il prodotto $\varphi(n) = (p-1)(q-1)$. Si considera che la fattorizzazione di n è segreta e solo chi sceglie i due numeri primi, p e q, la conosce.

3. Si sceglie poi un numero e (chiamato esponente pubblico), coprimo[4] con $\varphi(n)$ e più piccolo di $\varphi(n)$. Quindi significa che e deve essere $1 < e < \varphi(n)$ e $gcd(e\varphi(n)) = 1$.

4. si calcola il numero d (chiamato esponente privato) tale che il suo prodotto con e sia congruo a 1 (mod $\varphi(n)$) ovvero che $ed \equiv 1 \pmod{\varphi(n)}$. Inoltre, d deve essere compreso tra: $0 \leq d \leq N$. La proprietà che e e $\varphi(n)$ sono primi garantisce sempre l'esistenza dell'inverso moltiplicativo d.

La chiave pubblica è $K_U = \{e, n\}$, mentre la chiave privata è $K_R = \{d, n\}$. La forza dell'algoritmo sta nel fatto che per calcolare d da e (o viceversa) non basta la conoscenza di n, ma serve il numero $\varphi(n) = (p-1)(q-1)$, e che il suo calcolo richiede tempi molto elevati; infatti fattorizzare in numeri primi (cioè scomporre un numero nei suoi divisori primi) è un'operazione computazionalmente costosa.

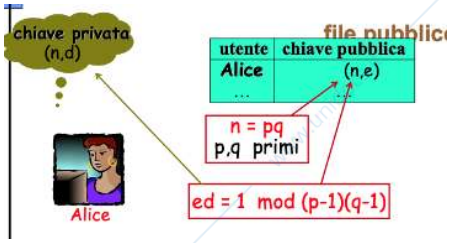


Figura 115: Generazione delle chiavi

8.2.2 Cifratura e decifratura

Un messaggio m viene cifrato, dopo aver ottenuto la chiave pubblica del destinatario $K_U = \{e, n\}$, attraverso l'operazione $c = m^e \pmod{n}$ trasformandolo nel messaggio cifrato c, dove $0 \leq m < n$.

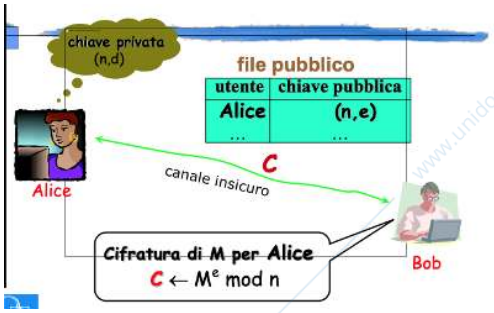


Figura 116: Cifratura RSA

Una volta trasmesso, c viene decifrato, mediante l'utilizzo della chiave privata $K_R = \{d, n\}$ con $m = c^d \pmod{n}$. Il procedimento funziona solo se $m < n$ e la chiave e utilizzata per cifrare e la chiave d utilizzata per decifrare sono legate tra loro dalla relazione $ed \equiv 1 \pmod{\varphi(n)}$, quindi quando un messaggio viene cifrato con una delle due chiavi può essere decifrato solo utilizzando l'altra. Inoltre, il messaggio m deve essere più piccolo del modulo n.

[4]Gli interi a e b si dicono coprimi (o primi tra loro o relativamente primi) se e solo se essi non hanno nessun divisore comune eccetto 1 e -1 o, in modo equivalente, se il loro massimo comune divisore è 1.

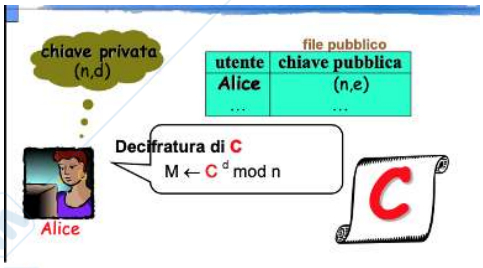


Figura 117: Decifratura RSA

8.2.3 "Piccolo Esempio"

Vediamo un esempio del procedimento con numeri piccoli:

- A sceglie due numeri primi p e q: $p = 47, q = 71$
- A calcola $n = p \times q$ e $f(n) = (p-1) \times (q-1)$. $n = 47 \times 71 = 3337$ e $\varphi(n) = 46 \times 70 = 3220$.
- A sceglie e tale che $gcd(e, f(n)) = 1$: $e = 79$
- A calcola $d = e^{-1} \pmod{\varphi(n)}$: $d = 79^{-1} \pmod{3220} = 1019$
- La chiave pubblica è la coppia $(e, n) = (79, 3337)$
- La chiave privata è la coppia $(d, n) = (1019, 3337)$

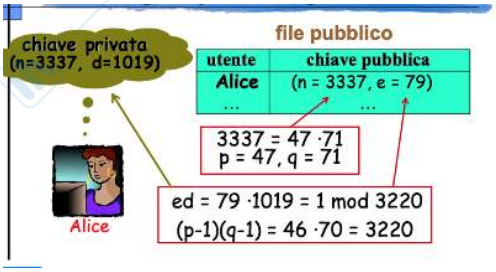


Figura 118: Generazioni chiavi RSA

- Un utente B spedisce ad A il messaggio $M = 688$, conoscendo la sua chiave pubblica $P_A = (79, 3337)$
- B calcola $C = M^e \pmod{n} = 688^{79} \pmod{3337} = 1570$

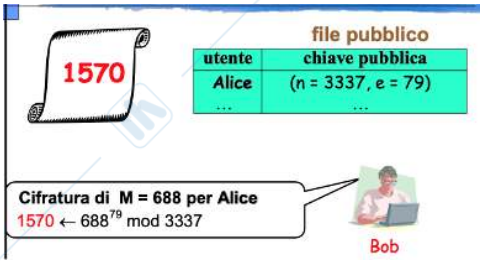


Figura 119: Cifratura RSA

- A riceve il messaggio cifrato $C = 1570$
- A si ricava M con la formula $M = C^d \pmod{n} = 1570^{1019} \pmod{3337} = 688$

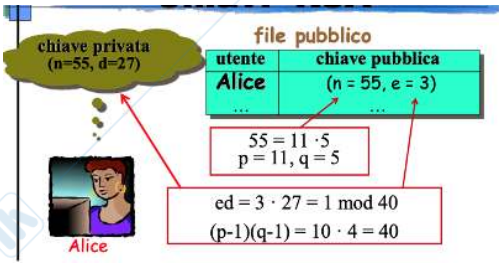


Figura 120: Decifratura RSA

Bisogna sempre ricordare che il messaggio da trasmettere deve essere più piccolo del valore di n, se così non fosse il messaggio decifrato dal destinatario non coincide con il messaggio inviato dal mittente. Nel caso di un numero grande è possibile pensarlo in binario e suddividerlo in parti da convertire singolarmente in decimale.

In RSA l'avversario può accedere all'algoritmo di cifratura: conosce la chiave pubblica. Vediamo un esperimento di indistinguibilità:

- A conosce la public key e ha due messaggi m_0 e m_1
- Si sceglie un bit b random e un cifrato $c = E_{pk}(m_b)$
- A sceglie b'
- L'esperimento riesce se $b' = b$

La sicurezza dell'RSA si basa sul fatto che la funzione di cifratura $x^e mod n$ è una funzione "one-way" che è computazionalmente difficile da invertire per un nemico che volesse decifrare un messaggio. Solo conoscendo la fattorizzazione di n è possibile trovare il valore delle chiavi. Infatti conoscendo la fattorizzazione di n è possibile calcolare $\varphi(n) = (p-1)(q-1)$ e calcolare $d = e^{-1} \pmod{\varphi(n)}$ usando l'algoritmo esteso di Euclide. Da ciò si deduce che la sicurezza dell'RSA dipende dal problema di fattorizzare grandi numeri. Di conseguenza il modo più ovvio di attaccare il sistema, è quello di tentare di risolvere quest'ultimo problema. Gli algoritmi più efficienti, conosciuti fino ad ora, hanno una complessità $O(e^{\sqrt{log(n)(log(log(n))}})$. Fattorizzare un numero di 664 bit richiede almeno 10^{23} passi usando gli algoritmi più efficienti; per cui ipotizzando di avere una rete costituita da un milione di computer con ciascuno di loro che esegue un milione di passi al secondo, il tempo impiegato per fattorizzare n sarebbe dell'ordine dei 4000 anni. Se poi n fosse un numero a 1024 bit la stessa rete impiegherebbe 10^{10} anni per fattorizzarlo. Ciò sottolinea l'osservazione fatta nell'introduzione e cioè che non è impossibile decifrare un testo cifrato con il crittosistema RSA, ma piuttosto è computazionalmente difficile.

Dati due messaggi m_0 e m_1 e un cifrato c. Se conosce la chiave pubblica può sempre determinare se $c = m_1$ o $c = m_2$. In generale, nessun schema a chiave pubblico deterministico può avere indistinguibilità delle cifrature in presenza di un avversario che intercetta.

8.2.4 Trapdoor Functions

Una funzione botola (dall'inglese trapdoor function) è una funzione facile da computare in una direzione, ma difficile da calcolare nella direzione opposta (ossia trovarne l'inversa) se non si conoscono determinate informazioni, chiamate appunto botole. Queste funzioni sono largamente utilizzate nell'ambito della crittografia. In termini matematici, sia F una funzione botola, allora esiste una qualche informazione segreta y tale che date $F(x)$ e y risulti facile calcolare x. Si consideri ad esempio un lucchetto e la relativa chiave. É banale cambiare il lucchetto da aperto a chiuso senza utilizzare la chiave, è sufficiente far scattare la serratura. Tutt'altra cosa invece è aprire il lucchetto, in questo caso l'ausilio della chiave risulta indispensabile. La chiave è la

botola. Le funzioni botola hanno fatto la loro comparsa nell'ambito della crittografia intorno alla metà del 1970 con la pubblicazione di Tecniche di crittografia asimmetrica (o a chiave pubblica) da parte di Diffie, Hellman, and Merkle. Di fatto il termine botola venne proprio coniato da Diffie e Hellman (Diffie e Hellman, 1976). Inizialmente vennero proposte diverse classi di funzioni, ma fu subito chiaro che delle funzioni botola vere e proprie sono molto più difficili da trovare di quanto si pensasse. Fino al 2004 i candidati migliori per svolgere il ruolo di funzioni botola erano l'RSA e le funzioni della famiglia Rabin. Entrambi sono scritti come elevamento a potenza modulo di un numero complesso ed entrambi sono legati al problema della fattorizzazione dei numeri primi.

Una funzione f si dice one-way se $\forall x$ il calcolo computazionale di $y = f(x)$ è semplice ($\in P$) mentre il calcolo di $x = f^{-1}(y)$ è computazionalmente difficile ($\in NP$). Una funzione one-way è detta trapdoor se il calcolo $x = f^{-1}(y)$ può essere reso facile qualora si conoscano informazioni aggiuntive (private). Una Trapdoor function $X \to Y$ è definita dalla tripla (G, F, F^{-1}), dove $\forall(p_k, s_k)$ output da G:

- $G()$: emette una coppia di chiavi (p_k, s_k)
- $F(p_k, \cdot)$: definisce una funzione $X \to Y$
- $F^{-1}(s_k, \cdot)$: definisce funzione $Y \to X$, che inverte $F(pk, \cdot)$

$\forall x \in X : F^{-1}(s_k, F(p_k, x)) = x$. Una trapdoor function (G, F, F^{-1}) è sicura se $F(p_k, \cdot)$ è una "one-way" function se può essere calcolata, ma non può essere invertita, a meno che non si conosca s_k. Avendo una trapdoor function sicura è possibile definire un sistema di cifratura a chiave pubblica:

- Trapdoor sicura $(G, F, F^{-1}) : X \to Y$, dove $G()$ genera chiavi RSA $pk = (N, e)$ e $sk = (N, d)$ con $N = pq$ ed $ed = 1 mod \varphi(n)$
- Sistema di cifratura simmetrica autenticata (E, D) operante su (K, M, C)
- Funzione hash $H : X \to K$

8.2.5 RSA e Teoria dei numeri

La segretezza che l'algoritmo RSA promette (e mantiene) è garantita proprio da una sconfitta della matematica: il non saper fattorizzare in tempi ragionevoli il prodotto di due numeri primi grandi.

La teoria dei numeri è quella parte della matematica che si occupa dei numeri interi; in pratica teoria dei numeri è sinonimo di aritmetica. Chi ha studiato aritmetica solo alle scuole elementari e medie potrebbe essere indotto a pensare che l'aritmetica sia la parte più semplice e banale della matematica. È vero il contrario: alcuni dei più formidabili problemi matematici sono appunto problemi di teoria dei numeri. In particolare sappiamo ancora molto poco sui numeri primi. Questa che per il matematico teorico è una grossa spina, è diventata viceversa una fortuna per la crittologia: molti cifrati moderni a cominciare da RSA si basano proprio sulla estrema complessità di alcuni problemi aritmetici: la fattorizzazione di un numero (RSA), il calcolo del logaritmo discreto e quello della radice discreta.

Un intero $p > 1$ è un numero primo se e solo se i suoi unici divisori sono 1 e se stesso. Alcuni esempi sono: 2, 3, 5, 7, 11, 13, 17, 19, etc. Il teorema fondamentale dell'aritmetica afferma che: Ogni numero naturale maggiore di 1 o è un numero primo o si può esprimere come prodotto di numeri primi. Tale rappresentazione è unica, se si prescinde dall'ordine in cui compaiono i fattori. Ossia, il numero n è rappresentabile come: $n = p_1^{e_1} p_2^{e_2} ... p_k^{e_k}$, dove p_i rappresenta un numero primo, mentre e_i è un intero positivo.

L'insieme dei numeri il cui prodotto è uguale a n è chiamato fattorizzazione di n. Per ogni numero n ne esistono svariati insiemi, ma la scomposizione in numeri primi è unica. Due numeri

a e b sono coprimi quando non hanno fattori in comune, ovvero il loro massimo comune divisore è 1. Ad esempio: 14 e 9 non sono numeri primi, ma sono coprimi tra loro. Naturalmente due numeri primi saranno sempre coprimi tra loro.

Il teorema della divisione enuncia che: dati $a \in Z$, $n \in N$ esiste un'unica coppia (q, r) con $0 \leq r \leq n-1$ tale che $a = q \cdot n + r$, dove q rappresenta il quoziente e r rappresenta il resto. r è indicato con $a \pmod n$. Poniamo che $a = 11, n = 7$. Si ha che $11 = 1 \cdot 7 + 4$, quindi si ha che $r = 11 \pmod 7 = 4$. Con l'operatore $\pmod{}$ si indica il resto intero della divisione di un numero. In particolare, scrivendo $b = a \pmod n$ si indica che b è uguale al resto della divisione di a per n, ovvero $a = k \cdot n + b$. Ad esempio: $27 \pmod{11} = 5$ perché $7 = 2 * 11 + 5$. Inoltre, data l'espressione $b = a \pmod n$, il valore di b può essere un qualunque numero compreso tra 0 e $n-1$. L'aritmetica modulare si basa sul concetto di congruenza modulo n. Dati tre numeri interi a, b, n, con $n \neq 0$, diciamo che a e b sono congruenti modulo n, oppure che a è congruo a b modulo n, se la differenza $(a-b)$ è un multiplo di n. In questo caso scriviamo $a \equiv b \pmod n$. Per esempio, possiamo scrivere $38 \equiv 14 \pmod{12}$, poiché 38 - 14 = 24, che è un multiplo di 12. Nel caso entrambi i numeri siano positivi, si può anche dire che a e b sono congruenti modulo n se hanno lo stesso resto nella divisione per n. Quindi possiamo anche dire che 38 è congruo a 14 modulo 12 poiché sia 38 sia 14 hanno resto 2 nella divisione per 12.

Per definizione l'insieme Z_n è il quoziente di Z per la relazione di congruenza modulo n. Dunque per definizione l'insieme Z_n è formato dalle classi di equivalenza rispetto a questa relazione. Tali classi di equivalenza sono anche chiamate classi di resto modulo n. Se $x \in Z$, indichiamo con $[x]_n$ la sua classe di equivalenza rispetto alla congruenza modulo n. Quindi $[a]_n = \{a + kn, k \in Z\}$ rappresenta l'insieme dei numeri che divisi per n danno lo sesso resto dato da $a \pmod n$. Viene rappresentata dal più piccolo intero positivo che è in essa: $b \in [a]_n \leftrightarrow b = a + kn \leftrightarrow b - a = kn \leftrightarrow n|(b-a) \leftrightarrow b \equiv a \pmod n$. Quindi $Z_n = \{[a]_n, 0 \leq a \leq n-1\}$, o per semplicità $Z_n = \{0, ..., n-1\}$. Ad esempio: $Z_4 = \{[0]_4, [1]_4, [2]_4, [3]_4\}$, significa che:

- $[0]_4 = \{..., -12, -8, -4, 0, 4, 8, 12, ...\}$, ossia tutti quei numeri che divisi per 4 danno resto 0;
- $[1]_4 = \{..., -11, -7, -3, 1, 5, 9, 13, ...\}$, ossia tutti quei numeri che divisi per 4 danno resto 1;
- $[2]_4 = \{..., -10, -6, -2, 2, 6, 10, 14, ...\}$, ossia tutti quei numeri che divisi per 4 danno resto 2;
- $[3]_4 = \{..., -9, -5, -1, 3, 7, 11, 15, ...\}$, ossia tutti quei numeri che divisi per 4 danno resto 3.

d è il massimo comune divisore di a e n se d è un divisore di a e n e se ogni divisore di a e n è un divisore di d: $d = a \cdot x + n \cdot y$. Se $n = p_{i1}^{e1} p_{i2}^{e2} ... p_{ik}^{ek}$ e $a = p_{j1}^{f1} p_{j2}^{f2} ... p_{jk}^{fk}$, (dove p_l è un numero primo e e è un intero positivo), $gcd(a, n) = p_{ij}^{min(ei, fj)}$. Le proprietà di cui gode il massimo comune divisore (o gcd) sono:

- $gcd(a, n) = gcd(a, -n) = gcd(-a, -n) = gcd(|a|, |n|)$
- $gcd(a, 0) = |a|$
- Se $gcd(a, n) = 1$, a e n sono relativamente primi. L'unico fattore intero positivo in comune è 1.

L'algoritmo di Euclide è un algoritmo per trovare il massimo comune divisore (indicato di seguito con MCD) tra due numeri interi. É uno degli algoritmi più antichi conosciuti, essendo presente negli Elementi di Euclide intorno al 300 a.C.. Dati due numeri naturali a e b, si controlla se b è zero. Se lo è, a è il MCD. Se non lo è, si divide a/b e si assegna ad r il resto della divisione

(operazione indicata con "$a \pmod b$"). Se $r = 0$ allora si può terminare affermando che b è il MCD cercato, altrimenti occorre assegnare $a' = b$ e $b' = a \pmod b$ e si ripete nuovamente l'algoritmo. La Figura 121 riporta l'algoritmo.

```
Euclide (a,b)
if  b = 0  then return a=gcd(a,b)
           else  return  Euclide (b, a mod b)
```

Figura 121: Algoritmo Euclide

Per tutti gli interi $a \geq 0$ e $b \geq 0$, $gcd(a, b) = gcd(b, a \pmod b)$. Quindi, l'insieme dei divisori comuni di a e b è uguale all'insieme dei divisori comuni di b e di $a \pmod b$.

Assumendo a > b si ha che l'algoritmo di Euclide esegue al più $logb$ chiamate, in ogni chiamata si hanno $O((loga)^2)$ operazioni sui bit da cui in totale si hanno $O((loga)^3)$ operazioni sui bit. Dall'analisi in realtà si ottiene che Euclide richiede al più $O((loga)^2)$ operazioni sui bit.

Dato un insieme finito G con un'operazione che indichiamo con il simbolo $*$, possiamo considerare la tabella dell'operazione di G. Per far ciò indichiamo con $a_1, a_2, ..., a_n$ gli elementi di G. La tabella dell'operazione di G è la matrice quadrata di ordine n tale che al posto i, j vi è l'elemento $a_i * a_j$. Per rendere più esplicito il tutto si scrivono a lato gli elementi. L'insieme Z_n^* è uguale a $\{[a]_n, 0 \leq a \leq n-1$, in cui $gcd(a, n) = 1$. In modo analogo si può scrivere (esercizio) la tabella dell'addizione. Si ottengono quindi due tabelle come quelle in Figura 122.

	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4	4	5	6	7	0	1	2	3
5	5	6	7	0	1	2	3	4
6	6	7	0	1	2	3	4	5
7	7	0	1	2	3	4	5	6

Addizione

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	0	2	4	6
3	0	3	6	1	4	7	2	5
4	0	4	0	4	0	4	0	4
5	0	5	2	7	4	1	6	3
6	0	6	4	2	0	6	4	2
7	0	7	6	5	4	3	2	1

Moltiplicazione

Figura 122: Tabella per addizione moltiplicazione per aritmetica modulo 8

Non tutti gli interi in Z_8 hanno un'inversa moltiplicativa (cella con cifra 1), al contrario dell'addizione. Gli interi in $Z_8^* = \{[1]_8, [3]_8, [5]_8, [7]_8\}$ hanno l'inversa moltiplicativa.

In matematica, la funzione φ di Eulero o semplicemente funzione di Eulero o toziente, è una funzione definita, per ogni intero positivo n, come il numero degli interi compresi tra 1 e n che sono coprimi con n. La funzione di Eulero associa a un numero intero N il numero dei numeri interi primi con N e minori di N (compreso l'uno); detto altrimenti i p tali che $MCD(N, p) = 1$; si tratta di una funzione basilare della teoria dei numeri, che interviene in molti teoremi come quello di Fermat-Eulero, oltre ad essere uno degli ingredienti fondamentali del cifrario RSA. La funzione toziente di Eulero $\varphi(N) = (p-1)(q-1)$ indica il numero di interi primi relativi (cioè non hanno fattori primi in comune) minori di N. Ad esempio: $\varphi(37) = 36$, $\varphi(35) = (24)$. Per un numero primo p si ha che $\varphi(p) = p - 1$. Per due numeri primi p e q si ha che $\varphi(n) = \varphi(p * q) = \varphi(p) * \varphi(q) = (p-1) * (q-1)$, perchè considerato Z_n, gli elementi non primi relativi di n sono i $(q-1)$ multipli di p e i $(p-1)$ multipli di q e 0 cioè: $\varphi(n) = (p * q) = pq - [(q-1) + (p-1) + 1] = pq - q + p - 1 = (p-1)(q-1)$. Ad esempio: $\varphi(21) = \varphi(3*7) = \varphi(3) * \varphi(7) = (3-1) * (7-1) = 12$. In un'aritmetica modulare di ordine N il valore di $\varphi(N)$ dà anche il numero di elementi che ammettono elemento inverso. Quindi, $\varphi(N)$ rappresenta la cardinalità di Z_n^*:

- $\varphi(p) = p - 1$ se p è primo

- $\varphi(pq) = (p-1)(q-1)$ se p e q sono primi
- $\varphi(n) = n \cdot (1 - \frac{1}{p_1}) \cdot (1 - \frac{1}{p_2})...(1 - \frac{1}{p_k})$ altrimenti

Il piccolo teorema di Fermat dice che se p è un numero primo, allora per ogni intero a: $a^p \equiv a \pmod{p}$. Questo significa che se si prende un qualunque numero a, lo si moltiplica per se stesso p volte e si sottrae a, il risultato è divisibile per p. É spesso espresso nella forma equivalente: se p è primo e a è un intero coprimo con p, allora: $a^{p-1} \equiv 1 \pmod{p}$. Ad esempio, con $a = 7$ e $p = 19 \rightarrow 7^{18} = 1 \pmod{19}$. In matematica, e in particolare in teoria dei numeri, il teorema di Eulero (detto anche teorema di Fermat-Eulero) afferma che se n è un intero positivo ed a è coprimo rispetto ad n, allora: $a^{\phi(n)} \equiv 1 \mathrm{mod} n$, dove $\phi(n)$ indica la funzione phi di Eulero e $\equiv$ la relazione di congruenza modulo n. Questo teorema è una generalizzazione del piccolo teorema di Fermat e vale se $gcd(a, n) = 1$ (primi relativi). Ad esempio, con $a = 3, n = 10$ e $\varphi(10) = 4 \rightarrow 3^4 = 81 = 1 \pmod{10}$.

Per il teorema di Eulero, per M e n primi relativi si ha che $M^{\phi(n)} \pmod{N} = 1$, dove i primi relativi sono identificati da $gcd(a, N) = 1$. In RSA abbiamo:

- $N = p \cdot q$
- $\varphi(N) = (p-1)(q-1)$
- Scelti e e d come inversi di $\pmod{\varphi(N)}$
- Quindi $ed = 1 + k \pmod{\varphi(N)}$ per qualche k

La funzione di decifratura è data da $C^d \pmod{n} = (M^e)^d \pmod{n} = M^{ed} \pmod{n}$, ma $ed = 1 \pmod{(p-1)(q-1)}$. Quindi si ottiene che $C^d \pmod{n} = M^{1+k(p-1)(q-1)} = M \cdot (M^{(p-1)(q-1)})^k$. Per il teorema di Eulero si ha che $M^{(p-1)(q-1)} = 1 \pmod{n}$, quindi si ha che $C^d \pmod{n} = M \pmod{n} = M$, poichè $0 \leq M < n$.

8.2.6 Elevazione a potenza modulare

Le procedure del crittosistema RSA si basano sull'elevazione a potenza modulare e sulla risoluzione di equazioni modulari. L'operazione dell'elevazione a potenza modulare consiste nel calcolare $x^y \pmod{z}$ dove x, y e z sono interi. Presentiamo varie procedure per effettuare il calcolo:

- Metodo naive: Intuitivamente si può semplicemente moltiplicare x per se stesso y volte facendo ad ogni passo modulo z per evitare cifre troppo grandi. Tale metodo è alla base del seguente algoritmo (Figura 123). Tale algoritmo in realtà non è utilizzato in quanto

```
Potenza_Modulare_naive (x, y, z)
 a ← 1
for i = 1 to y do
        a ← (a · x) mod z
return a
```

Figura 123: Metodo naive

  poco efficiente. Il numero cicli, infatti, è asintoticamente uguale a y. Quindi se y è di 512 bit si avranno 2^{512} cicli. Questo metodo è esponenziale nella lunghezza dell'esponente

- Metodo Left-to-Right: Ricordando che vogliamo calcolare $x^y \pmod{z}$, supponiamo che $y_0 y_1 y_2 y_3 ... y_t$ siano i bit da cui è formato y espresso in base due, il valore di y è quindi: $y = y_0 + 2(y_1 + 2(y_2 + 2(y_3 ... 2(y_t - 1 + 2y_t)\ldots)))$, da cui $x^y = x^{y_0}(...(x^{y_{t-1}}(x^{y_t})^2)^2...)^2$. La Figura 124 mostra un esempio del metodo left-to-right.

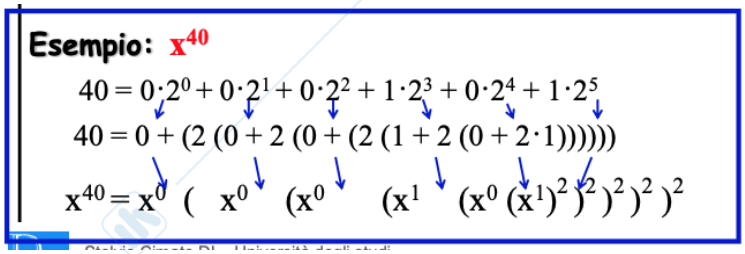


Figura 124: Metodo Left-to-right

Da tali considerazioni si ottiene il seguente algoritmo (Figura 125).

```
Potenza_Modulare (x, y, z)
a ← 1
for i = t downto 0 do
        a ← (a · a) mod z
    if y_i = 1 then a ← (a · x) mod z
return a
```

Figura 125: Algoritmo left-to-right

Chiaramente tale algoritmo è migliore rispetto a quello precedente infatti se y è composto da 512 bit ora il numero di cicli è 512.

- Metodo right-to-left: Anche tale metodo segue da considerazioni matematiche infatti si ha che: $y = y_0 2^0 + y_1 2^1 + ... + y_{t-1} 2^{t-1} + y_t 2^t$ da cui $(x^{2^0} \cdot (x^{2^1})^{y_1} \cdot ... \cdot (x^{2^t})^{y_t}$. Si chiama anche binary exponentiation o square & multiply o Indian exponentiation.

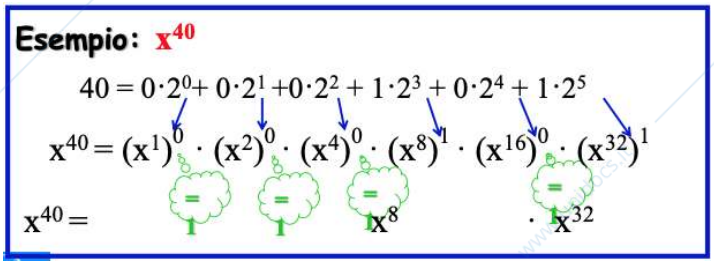


Figura 126: Metodo right-to-left

Da ciò segue l'algoritmo per il calcolo di $x^y \pmod z$ è quello raffigurato in Figura 127.

Anche questo metodo impiega 512 cicli se y è di 512 bit. Quindi, si tratta di un metodo polinomiale nella lunghezza dell'esponente. Le operazioni di elevazioni al quadrato (squaring) sono molto più veloci delle moltiplicazioni modulari. Esistono metodi di esponenziazione binaria che consentono di precomputare alcuni valori.

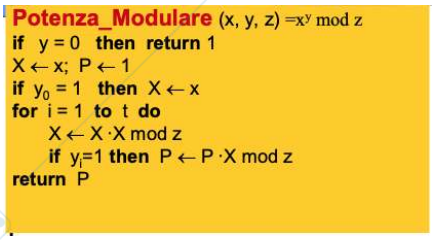


Figura 127: Algoritmo right-to-left

8.2.7 Attacco di Wiener

Ci troviamo nello scenario in cui A cerca di mandare un messaggio segreto m a B, utilizzando il sistema crittografico a chiave pubblica RSA, ed in cui E cerca di intercettare e decifrare il messaggio. Quindi, avremo che B sceglie due primi p e q grandi e distinti e li moltiplica tra di loro ottenendo $n = pq$ detto Modulo. Viene poi scelto l'esponente di cifratura e scelto in modo che: $MCD(e, \varphi(n)) = 1$, dove $\varphi(n)$ è la funzione di Eulero che ci da l'ordine del gruppo moltiplicativo degli elementi invertibili dell'anello Z_n e se si conosce la fattorizzazione di n, ovvero in questo caso p e q, si calcola facilmente con la formula $\varphi(n) = (p-1)(q-1)$. Viene poi calcolato d, l'esponente di decifrazione, in modo che $de \equiv 1 \pmod{\varphi(n)} \rightarrow d \equiv e^{-1} \pmod{\varphi(n)}$ (si può calcolare con l'algoritmo esteso di Euclide); quindi per B si avranno le seguenti chiavi:

- CHIAVE PUBBLICA: (n, e)
- CHIAVE PRIVATA: (n, d)

Ora, se A vuole mandare un messaggio m a B, calcola: $C = m^e \pmod{n}$ e manda C a B, che potrà decifrare il messaggio grazie alla sua chiave segreta d calcolando: $m \equiv C^d \pmod{n} \equiv m^{de} \pmod{n}$. Lo scopo dell'avversario E sarà quello di ottenere la chiave segreta d di B a partire da ciò che conosce, cioè la chiave pubblica (n, e).

L'attacco di Wiener con esponente di cifratura basso, come ogni altro attacco crittografico, si basa su una debolezza del sistema, in questo caso sarà la lunghezza dell'esponente di decifrazione o cifratura. Quindi tale attacco sarà realizzabile se l'esponente sarà scelto abbastanza piccolo. Infatti in alcuni casi è preferibile usare un piccolo esponente di decifrazione in quanto viene così ridotto il tempo di decriptazione. Questo si ha perché per un modulo di lunghezza fissata, il tempo di decifrazione o cifratura è proporzionale al numero di bit dell'esponente scelto. Uno di questi casi, in cui scegliere un esponente corto porta un vantaggio, si verifica quando c'è una grossa differenza di potenza di calcolo tra due apparecchi, ad esempio quando RSA viene usato nella comunicazione tra una carta di credito e un computer. In questo caso, sarebbe preferibile per il calcolatore della carta avere un esponente in grado di ridurre i processi computazionali richiesti. Un altro caso si ha quando c'è una esigenza di maggiore velocità di decriptazione da parte dell'utente B, che lo porterebbe a scegliere un esponente più piccolo.

8.2.8 Generazione di numeri primi grandi

In matematica, un numero primo (in breve anche primo) è un numero intero positivo che abbia esattamente due divisori distinti. In modo equivalente si può definire come un numero naturale maggiore di 1 che sia divisibile solamente per 1 e per sé stesso; al contrario, un numero maggiore di 1 che abbia più di due divisori è detto composto. Quello di numero primo è uno dei concetti basilari della teoria dei numeri, la parte della matematica che studia i numeri interi: l'importanza sta nella possibilità di costruire con essi, attraverso la moltiplicazione, tutti gli altri numeri interi, nonché l'unicità di tale fattorizzazione. I primi sono inoltre infiniti. Procediamo per assurdo e assumiamo che p_{max} è il massimo numero primo. Calcola $N = p_1 * p_2 * \ldots * p_{max} + 1$. N deve

essere composto perché è più grande di p_{max}, ma non è vero perché tutti i primi noti dividono N con resto 1. Quindi N è primo: contraddicendo l'ipotesi. La generazione di numeri primi avviene secondo il seguente metodo:

1. Genera a caso un dispari p di grandezza appropriata
2. Testa se p è primo
3. Se p è composto, go to 1

Una variante con ricerca del seguente metodo è la seguente:

1. Genera a caso un dispari p di grandezza appropriata
2. Testa se p è primo
3. Se p è composto, $p \to p+2$, go to 2.

In teoria dei numeri, il teorema dei numeri primi descrive la distribuzione asintotica dei numeri primi, dando una descrizione approssimativa di come i numeri primi sono distribuiti. Per ogni numero reale positivo x, si definisca la funzione: $\pi(x) :=$ numero di primi minori o uguali a x. Il teorema dei numeri primi afferma che: $\pi(x) \sim \frac{x}{\ln(x)}$, dove $ln(x)$ è il logaritmo naturale di x. Questa notazione vuole significare solo che il limite del quoziente delle due funzioni $\pi(x)$ e $\frac{x}{ln(x)}$ per x che tende ad infinito è 1; ciò non significa che il limite della differenza delle due funzioni, per x che tende ad infinito, è 0. Ad esempio, supponiamo di avere $\pi(10^{10}) = 455.052.511$, dal momento che si ha $\frac{10^{10}}{ln(10^{10})} \approx 434.294.481,9$(4% in meno). Già da molti secoli la ricerca di numeri primi "grandi" ha destato l'interesse dei matematici; tuttavia questa ricerca ha assunto una particolare importanza negli ultimi decenni, a causa del bisogno di tali numeri che caratterizza algoritmi quali l'RSA. Si supponga di dover scegliere un numero primo a 512 bit: scelto un intero in $[2, 2^{512}]$ la probabilità che sia primo è circa 1 su $ln(2^{512}(ln(2^{512}) \approx 354,89)$. Il numero medio di tentativi è $\approx 354,89$; se si scelgono solo dispari dimezza $\approx 177,44$z. Se, invece, si scelgono solo dispari in $[2^{511}, 2^{512}]$ la probabilità è $\approx (\frac{2^5 12}{ln(2^{512})} - \frac{2^5 11}{ln(2^{511})})\frac{1}{2^{511}/2} \approx \frac{1}{177,79}$.

Il test di primalità è una procedura algoritmica che, dato un numero naturale n in input, restituisce PRIME se n è un numero primo, COMPOSITE se n è un numero composto. Questo algoritmo, noto già da secoli, è oggi utilizzato in svariate procedure di grande rilevanza, tipicamente quando è necessario manipolare numeri primi giganteschi. Algoritmi noti che si appoggiano a questa procedura, nel campo della crittografia, sono ad esempio RSA, El Gamal e altri. Fino al 2002, il miglior algoritmo deterministico era una variante di Cohen e H. Lenstra del test di Adleman, Pomerance e Rumely (1983), con complessità $(lnp)^{O(lglglgp)}$. Nel 2002 Agrawal et al. scoprirono un algoritmo deterministico polinomiale in grado di testare se un numero è primo (AKS test), con complessità $O(lnp)^{6+\epsilon}$. Chiameremo "test deterministici di primalità" quelle procedure che consentono di determinare se un numero è primo applicando uno specifico algoritmo. Il più semplice test deterministico di primalità per un numero n è quello che si basa sulle successive divisioni di n per tutti i primi inferiori alla radice quadrata di n. Naturalmente questo test non è applicabile per interi sufficientemente grandi. I test probabilistici di primalità consentono di verificare l'ipotesi nulla o di base: H_0: n è un numero primo. Se l'ipotesi viene rifiutata il numero è sicuramente composto. Se il test fa rifiutare l'ipotesi di base vi è probabilità pari a uno che il numero sia composto. Si tratta di un test statistico in cui la probabilità dell'errore di seconda specie, ossia di accettare un'ipotesi falsa, è un numero diverso da zero. La probabilità di errore (n è composto ma viene dichiarato primo) di tale test è pari a $1/2$. Se il test viene ripetuto indipendentemente t volte, allora la probabilità di errore è $(1/2)^t$.

Il test di primalità di Miller-Rabin è un test di primalità, ossia un algoritmo per determinare se un numero intero è primo. La sua versione originale, dovuta a Gary Miller, è deterministica, ma dipende dall'ipotesi di Riemann generalizzata, un'importante congettura matematica tuttora

aperta. L'algoritmo è stato successivamente modificato da Michael Rabin che ne ha ottenuto una versione probabilistica simile ai test di Fermat e di test di Solovay-Strassen. Definiamo testimone un numero a per cui vale il cosiddetto piccolo teorema di Fermat[5], pur essendo il numero composto. Il test in questione si basa sul seguente Teorema: Se n è un numero dispari composto, allora il numero dei testimoni da cui n è composto è almeno $(n-1)/2$. L'insieme dei testimone $W(p)$ è dato da:

- Dato $a \in [0, p-1]$ è facile verificare se $a \in W(p)$
- se p è primo allora $W(p)$ è vuoto
- se p è composto allora $|W(p)| \geq p/2$

Quindi, il test di Miller-Rabin è basato sul Teorema di Fermat: se p è primo, per ogni $a \in Z_p^*$: $a^{p-1} = 1 \pmod p$. La prima proprietà prevede che se p è primo e $a < p$ allora $a^2 \pmod p = 1$ se e solo se $a \pmod p = 1$ o $a \pmod p = -1 = p-1$. Ricorda che $a^2 \pmod p = (a \pmod p)(a \pmod p)$. La seconda proprietà prevede che per p dispari e $p-1$ pari: $p-1 = 2^k q$, con q dispari e $k > 0$. Scegli $1 < a < p-1$ e calcola $a^q, a^{2q}, \ldots, a^{2^k q}$. Se p è primo, $a^{2^k q} = 1 \pmod p$ (teorema di Fermat). Vale:

- $a^q = 1 \pmod p$
- Se esamino la sequenza $a^{2^j q}$,ossia $a^q, a^{2q}, a^{4q}, ..., a^{2^{k-1}q}, a^{2^k q}$
- L'ultimo numero vale 1 e ogni numero della sequenza è il quadrato di quello che lo precede. La lista contiene un elemento uguale a $p-1$.

L'insieme di witness di Miller-Rabin è dato da $WMR(p) = \{a : a^q \neq 1 \pmod p and a^{2^j q} \neq -1 \pmod p$ per ogni $j \in [0, k-1]\}$. Se p è un primo dispari $WMR(p)$ è vuoto. Se p è un numero composto dispari $|WMR(p)| \geq (3/4) \cdot \phi(p)$. Ricordiamo che $2^j q = p-1$, dove q è un numero dispari. Ad esempio, ipotizziamo che $p = 29$, $29 - 1 = 2^2 \cdot 7$. Abbiamo che $WMR(29) = \{a : a^7 \neq 1 \pmod{29}$ and $a^{2^j \cdot 7} \neq -1 \pmod{29}$ per ogni $j \in [0, 1]\}$. Per $a = 10$:

- $10^7 \pmod{29} = 17$
- $(10^7)^2 \pmod{29} = 28 = -1 \pmod{29}$ (stop: "primo")

Per a=2:

- $2^7 \pmod{29} = 12$
- $(2^7)^2 \pmod{29} = 28 = -1 \pmod{29}$ (stop: "primo")

Prova per tutti gli $a \in [1, 28]$: sempre output "primo".

Il funzionamento del metodo di Miller-Rabin si può riassumere nel seguente modo: mentre si calcola l'evamento a potenza modulare a^{n-1}, si controlla se esistono radici quadrate non banali di 1 $\pmod n$. Se ciò accade, si interrompe l'algoritmo e si segnala che n è composto. Si tratta di un test di primalità probabilistico, dal momento che un numero potrebbe passare il test, ma essere lo stesso composto.

[5]Se si prende un qualunque numero a, lo si moltiplica per se stesso p volte e si sottrae a, il risultato è divisibile per p.

8.2.9 Generazione di e e d

Una volta trovati i primi p e q, bisogna selezionare e, tale che $gcd(\phi(n), e) = 1$ e calcolare $d = e^{-1} \pmod{\phi(n)}$. L'algoritmo esteso di Euclide consente di calcolare il massimo comune divisore di due interi e, nel caso in cui $gcd = 1$, di determinare l'inverso di uno degli interi modulo l'altro. Quindi, l'algoritmo esteso di Euclide è un'estensione dell'algoritmo di Euclide che calcola non solo il massimo comun divisore (indicato con MCD nel seguito) tra due interi a e b , ma anche i coefficienti dell'identità di Bézout x e y tali che: $ax + by = \text{MCD}(a, b)$. La Figura 128 riporta l'algoritmo di Euclide esteso.

```
Euclide-esteso (a,n)
if  n = 0  then return (a, 1, 0)
(d´, x´, y´) ← Euclide-esteso (n, a mod n)
(d, x, y) ←  (d´, y´, x´ - ⌊a/n⌋ y´)
return  (d, x, y)
```

Figura 128: Algoritmo di Euclide esteso

La Figura 129 riporta la tabella relativa alle operazioni in modulo 8. Vediamo alcuni esempi di calcolo del tipo $ax \equiv b \pmod{n}$:

- $3x \equiv 7 \pmod 8$ soluzione 5. Ricerco sulla riga corrispondente al 3 il numero 7.
- $2x \equiv 4 \pmod 8$ soluzione 2,6
- $4x \equiv 2 \pmod 8$ nessuna soluzione

La congruenza $ax \equiv b \pmod{n}$ ha soluzioni se e solo se $g|b$, dove $g = gcd(a, n)$. Se $g|b$ ci sono esattamente g distinte soluzioni $\pmod n$: $x' \frac{b}{g} + i\frac{n}{g}$ per $i = 0, 1, ..., g - 1$. Per il teorema di Euclide esteso $g = gcd(a, n)$ è uguale a $g = a \cdot x' + n \cdot y$.

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	0	2	4	6
3	0	3	6	1	4	7	2	5
4	0	4	0	4	0	4	0	4
5	0	5	2	7	4	1	6	3
6	0	6	4	2	0	6	4	2
7	0	7	6	5	4	3	2	1

Figura 129: Tabella risolutiva $\pmod 8$

$ax \equiv 1 \pmod{n}$ ha soluzioni se e solo se $gcd(a, n) = 1$. Si tratta di un caso specifico di $ax \equiv b \pmod{n}$ con n primo. Se $gcd(a, n) = 1$ l'unica soluzione $\pmod n$ è: x' dove $1 = a \cdot x' + n \cdot y$ (da Euclide-esteso (a, n)). Tale soluzione viene denotata con $a^{-1} \pmod{n}$ (inversa moltiplicativa di a, $\pmod n$). Riprendendo la Figura 129, vediamo alcuni esempi di operazione del tipo $ax \equiv 1 \pmod 8$, per cui esiste una soluzione se e solo se $gcd(a, 8) = 1$:

- $1 \equiv 1^{-1} \pmod 8$. Ricerco sulla riga corrispondente all'1 il numero 1.
- $1 \equiv 3^{-1} \pmod 8$

- $1 \equiv 5^{-1} \pmod 8$
- $1 \equiv 7^{-1} \pmod 8$

La Figura 130 riporta un esempio di calcolo ($5^{-1} \pmod 7$) con l'algoritmo di Euclide esteso. Quello che cerco è la y dell'equazione dell'algoritmo di Euclide esteso, per poter calcolare $d = x \cdot a + y \cdot n$.

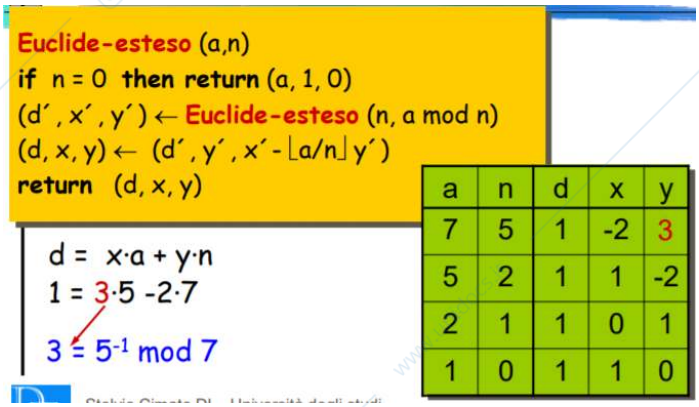


Figura 130: Calcolo di $5^{-1} \pmod 7$

Se n è compost0, ad esempo $n = pq$, allora la soluzione di $ax \equiv b \pmod n$ può essere vista come soluzione del sistema di congruenze: Soluzione di $ax \equiv b \pmod p$ e Soluzione di $ax \equiv b \pmod q$.

In matematica, il termine teorema cinese del resto comprende diversi risultati in algebra astratta e teoria dei numeri. Gli interi in un intervallo posso essere rappresentati dai resti modulo una coppia di numeri primi relativi. Per $Z_{10} = 0, 1, 2, \ldots, 9$, se considero una coppia di primi relativi, ad esempio 2 e 5, e la coppia di resti 0 e 3, allora esiste un unico intero rappresentato da quei resti, cioè esiste un unico x tale che $x \equiv 0 \pmod 2$ e $x \equiv 3 \pmod 5$. In questo caso si tratta di $x = 8$. Si supponga che $n_1, ..., n_k$ siano interi a due a due coprimi (il che significa che $MCD(n_i, n_j) = 1$ quando $i \neq j$). Allora, comunque si scelgano degli interi $a_1, ..., a_k$, esiste un intero x soluzione del sistema di congruenze $x \equiv a_i \pmod{n_i}$ per $i = 1, \ldots, k$. Inoltre, tutte le soluzioni x di questo sistema sono congruenti modulo il prodotto $n = n_1 ... n_k$. Si può trovare una soluzione x come segue. Una soluzione del sistema di congruenze è quindi $x = \sum_{i=1}^{k} a_i N_i y_i \pmod n$, dove $N_i = N/n_i$ e $y_i = N_i^{-1} \pmod n$. Ad esempio, abbiamo un sistema di congruenze come il seguente $x \equiv 2 \pmod 5$ e $x \equiv 3 \pmod{13}$. Si ha che $a_1 = 2, n_1 = 5, N_1 = 13, y_1 = 13^{-1} \pmod 5 = 2$ e $a_2 = 3, n_2 = 13, N_2 = 5, y_2 = 5^{-1} \pmod{13} = 8$. La soluzione è $N = 65$. Quindi, la soluzione del sistema è $X = \sum_{i=2}^{2} a_i \cdot N_i \cdot y_i \pmod{65} = 2 \cdot 26 + 3 \cdot 40 \pmod{65} = 42 \pmod{65} = 42$. Per trovare la soluzione di $x \pmod N$ dal sistema $x_i \pmod{p_i}$ con $N = p_1 p_2 \ldots p_r$. Per tutti gli $i = 1, \ldots, r$ calcola: le inverse y_i di $N/p_i \pmod{p_i}$ e $X = \sum_{i=1,\ldots r} N/p_i * x_i y_i \pmod N$. Il teorema cinese del resto è importante anche in crittografia; infatti il cifrario RSA deve fare calcoli modulo pq, dove p e q sono due numeri primi molto grandi (1024 o 2048 bit) e le operazioni modulari con modulo così grande richiedono tempi molto lunghi. Il teorema del resto permette di fare i calcoli in modulo p e q con una sensibile riduzione dei tempi. La Figura 131 mostra il funzionamento dell'algoritmo di generazione delle chiavi con l'inserimento del teorema cinese del resto (CTR).

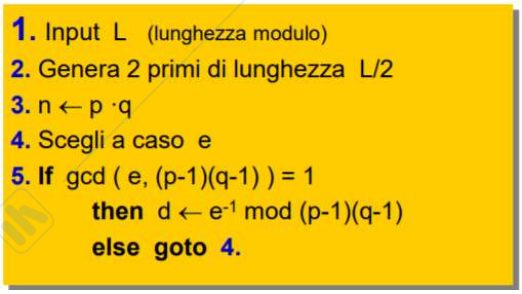


Figura 131: Generazione chiavi

8.2.10 Sicurezza di RSA

Conoscendo la chiave pubblica (n, e), un attaccante vuole calcolare la chiave privata $d = e^{-1} \pmod{(p-1)(q-1)}$.

Una prima tipologia di attacco prevede che se un attaccante potesse fattorizzare n, saprebbe computare d:

1. Fattorizza n
2. Computa $\phi(n) = (p-1)(q-1)$
3. Computa $d \leftarrow e^{-1} \pmod{(p-1)(q-1)}$

Una seconda tipologia di attacco prevede che se un attaccante potesse computare $\phi(n) = (p-1)(q-1)$, saprebbe fattorizzare n:

1. Risolvere il seguente sistema $\begin{cases} n = pq \\ \phi(n) = (p-1)(q-1) \end{cases}$
2. Sostituendo $p = n/q$, si ottiene $p^2 - (n - \phi(n) + 1)p + n = 0$. Si ottengono due soluzioni p e q.

Una terza tipologia di attacco prevede che se un attaccante potesse computare d saprebbe fattorizzare n. Un algoritmo che computa d (con input n, e) può essere usato come oracolo in un algoritmo Las Vegas*[6] che fattorizza n con probabilità $\geq 1/2$.

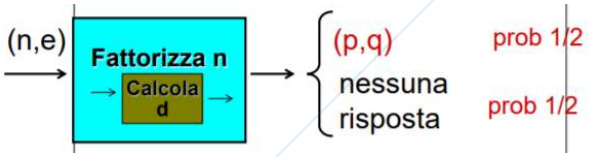


Figura 132: Computare d

Computare d è equivalente a fattorizzare n.

Ad oggi, i numeri più difficili da fattorizzare sono del tipo $n = p \cdot q$ con p, q primi della stessa lunghezza e di almeno:

- 768 bit per uso personale
- 1024 bit per le aziende

[6]Nell'informatica, un algoritmo di Las Vegas è un algoritmo randomizzato che fornisce sempre risultati corretti; cioè produce sempre il risultato corretto o informa del fallimento.

- 2048 per chiavi "importanti" - ad esempio, Autorità di Certificazione

Conoscendo la chiave pubblica (n, e) e il messaggio cifrato $C \leftarrow M^e \pmod{n}$, un attaccante vuole calcolare il messaggio M. Se un attaccante potesse fattorizzare n saprebbe computare M:

- Fattorizza n
- Computa $\phi(n) = (p-1)(q-1)$
- Computa $d \leftarrow e^{-1} \pmod{(p-1)(q-1)}$
- Ricava M decifrando C

La Figura 133 mostra gli sforzi necessari per fattorizzare n.

Dimensione della chiave in bits	Anni – MIPS necessari per la fattorizzazione
512	30.000
768	200.000.000
1024	300.000.000.000
2048	300.000.000.000.000.000.000

Figura 133: Stima dello sforzo richiesto

Gli attacchi non basati sul problema della fattorizzazione sono:

- Chosen ciphertext attack: Daniele intercetta i messaggi cifrati di Elena (C). Vuole calcolare calcolare $M = C_d$. Inoltre, conosce la chiave pubblica pubblica di Elena (m ed e). Daniele sceglie un numero casuale r tale che $r < n$, quindi può calcolare:
  - $x = r^e \pmod{n}$
  - $y = xC \pmod{n}$
  - $t = r^{-1} \pmod{n}$

  Se $x = r^e \pmod{n}$ allora $r = x^d \pmod{n}$, quindi $t = x^{-d} \pmod{n}$. Daniele manda y a Elena chiedendogli di firmarlo. Elena spedisce spedisce a Daniele Daniele $u = y^d \pmod{n}$. Daniele può facilmente facilmente risalire risalire ad M: $t \cdot u \pmod{n} = x^{-d} y^d C^d \pmod{n} = C^d \pmod{n} = M$
- Common modulus attack: esiste una vulnerabilità di RSA se viene utilizzato utilizzato lo stesso modulo da due utenti. La chiave di Alice è (n, e_1), mentre la chiave di Bob è (n, e_2). Il $gcd(e_1, e_2)$ è 1. Se lo stesso messaggio M inviato ai vari utenti:
  - Cifratura per Alice: $C_1 = M^{e_1} \pmod{n}$,
  - Cifratura per Bob: $C_2 = M^{e_2} \pmod{n}$

  É semplice risalire ad M, usando Euclide–esteso per calcolare x, y tali che $1 = e_1 \cdot x + e_2 \cdot y$: $C_1^x \cdot C_2^y \pmod{n} = (M^{e_1})^x \cdot (M^{e_2}) \cdot y = M^{e_1^x + e_2^y} = M$.
- Low exponent attack: il valore dell'esponente e può essere arbitrario, arbitrario, anche se è consigliabile scegliere un valore uguale per tutti gli utenti del sistema. Il vantaggio di un e piccolo è la maggiore velocità nella fase di cifratura e di firma. Supponiamo di scegliere $e = 3$: tre utenti scelgono e uguale, ma moduli diversi (supponiamo siano a due a due coprimi): $e_A = e_B = e_C = 3$. Quindi le chiavi di Alice, Bob e Carla sono:

- Chiave Alice: $(n_1, 3)$,
- chiave Bob: $(n_2, 3)$,
- chiave Carla: $(n_3, 3)$

Un quarto utente vuole mandare mandare lo stesso messaggio m ad Alice,Bob e Carla:

- Cifratura per Alice: $C_1 = M^3 \pmod{n_1}$
- Cifratura per Bob: $C_2 = M^3 \pmod{n_2}$
- Cifratura per Carla: $C_3 = M^3 \pmod{n_3}$

Chiunque intercetti i tre messaggi messaggi cifrati può risalire ad m. É possibile possibile calcolare x con il Teorema Teorema Cinese del resto (Figura 134) A questo punto, basta

$$\begin{cases} x \equiv c_A \bmod n_A \\ x \equiv c_B \bmod n_B \\ x \equiv c_C \bmod n_C \end{cases}$$

Figura 134: Calcolo messaggio in chiaro

calcolare la radice cubica di della soluzione del sistema.

- Timing Attack [Kocher, 97]: ricava i bit di d uno alla volta, analizzando il tempo richiesto per l'esponenziazione modulare (decifratura)
- Power Attack [Kocher, 99]: ricava d analizzando la potenza consumata da una smartcard durante la decifratura

9 Altri cifrari asimmetrici

Il problema di RSA è che si basa sul seguente problema: Dati il modulo pubblico $N = p * q$ e l'esponente pubblico e, trovare l'esponente privato d. Abbiamo dimostrato che il problema RSA è equivalente al problema della fattorizzazione: se sappiamo fattorizzare N sappiamo risolvere il problema RSA.

Il logaritmo discreto è in algebra astratta il corrispettivo del logaritmo $logx(e^{logx} = x)$. In maniera più rigorosa: sia G un gruppo finito con operazione $\otimes$, e siano $a \in G$ e $b \in H$ con H sottogruppo generato da a $(a^i : i \geq 0)$. Trova l'unico intero x, $0 \leq x \leq |H| - 1$, tale che $a^x = a \otimes a \otimes ... \otimes a = b$. L'intero x viene denotato con $log_a(b)$ e viene chiamato logaritmo discreto di b in base a. Quindi il problema è, dati a, n, b calcolare x tale che $a^x = b \pmod{n}$. Se n è primo, i migliori algoritmi hanno complessità $L_n[a, c] = O(e^{c+o(1)(lnn)^a(lnlnn)^{1-a}}$, con $c > 0$ e $0 < a < 1$. Il miglior algoritmo utilizzabile è il Number field sieve che ha un tempo medio euristico $L_n[1/31.923]$. La sicurezza di molte tecniche crittografiche si basa sulla intrattabilità del logaritmo discreto:

- Crittosistema di El-Gamal
- Firme digitali di El-Gamal e DSS
- Accordo su chiavi di Diffie-Hellman

In un'aritmetica finita di ordine N si possono dare particolari numeri g detti generatori; un generatore è una base che successivamente elevata a potenza, genera tutti i numeri che sono primi con N. g è un generatore di Z_p^* se $\{g^i | 1 \leq i \leq p - 1\} = Z_p^*$. Quindi, $a \in Z_n^*$ è un

generatore di Z_n^* se la funzione $a^k \pmod n$, dove $1 \leq k \leq \phi(n)$, genera tutti e soli gli elementi di Z_n^*. Produce come risultati tutti gli elementi di Z_n^*, ma in un ordine difficile da prevedere. La Figura 135 mostra un esempio di generatore per il numero 13.

ESEMPIO: g = 2 è un generatore di Z^*_{13} = {1,2,..., 12}

x	1	2	3	4	5	6	7	8	9	10	11	12
2^x	2	4	8	3	6	12	11	9	5	10	7	1

Figura 135: Calcolo di generatori

Non è detto che un generatore esista; per esempio in un'aritmetica modulo 12 (Z_{12}^*), tipo aritmetica dell'orologio, non c'è alcun generatore. Solo 5,7,11 sono primi con 12 e tutti i numeri minori di 12 innestano cicli brevi che generano troppi pochi numeri. Ne consegue che se N è primo allora g genererà successivamente tutti i numeri minori di N. Se n è un numero primo, Z_n^* ha almeno un generatore. Per n primo, non tutti gli elementi di Z_n^* sono suoi generatori (1 non è mai un generatore, e altri elementi possono non esserlo). Per n primo, i generatori di Z_n^* sono in totale $\phi(n-1)$.

9.1 Crittostistema di El-Gamal

ElGamal è un sistema di cifratura a chiave pubblica, proposto dal ricercatore egiziano-americano Taher Elgamal nel 1985. Lo schema è basato sulla difficoltà del calcolo del logaritmo discreto.

La generazione delle chiavi in questo crittosistema avviene nel seguente modo:

- Ogni utente u genera un numero primo a caso, p, trova un generatore g di Z_p^*, ed un numero α casuale, compreso tra 1 e $p-2$.
- La chiave pubblica dell'utente consiste nella tripla (p, g, β), dove $\beta = g^\alpha \pmod p$.
- La sua chiave privata consiste nel numero α

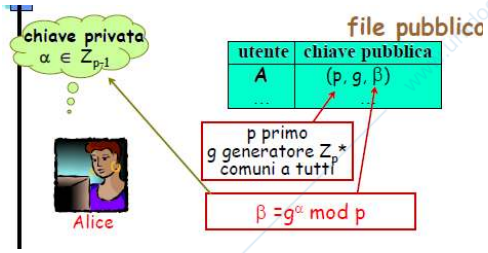


Figura 136: Calcolo delle chiavi

Per inviare un messaggio cifrato all'utente u, è necessario:

- Ottenere la chiave pubblica di u, ovvero la tripla (p, g, β)
- Codificare il messaggio come un intero M in Z_p.
- Scegliere un intero k a caso, tale che $1 \leq k \leq p-2$
- Calcolare $y_1 = g^k \pmod p$
- Calcolare $y_2 = M\beta^k \pmod p$

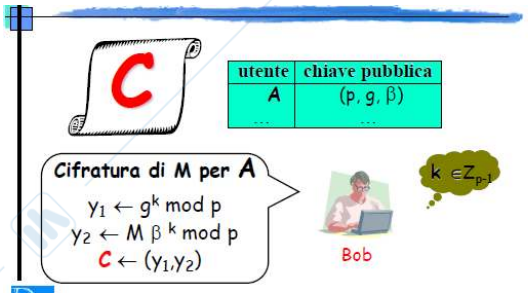


Figura 137: Cifratura

- Il messaggio cifrato c è dato dalla coppia (y_1, y_2)

L'utente u per decifrare il messaggio criptato (x, y), deve:

- Calcolare, usando la sua chiave privata α, il numero $x - a$.
- Calcolare il messaggio in chiaro, calcolando dapprima $z = y_1^{-\alpha} \pmod p$. Infatti, $M = z^{-1} \cdot y_2 \pmod p$

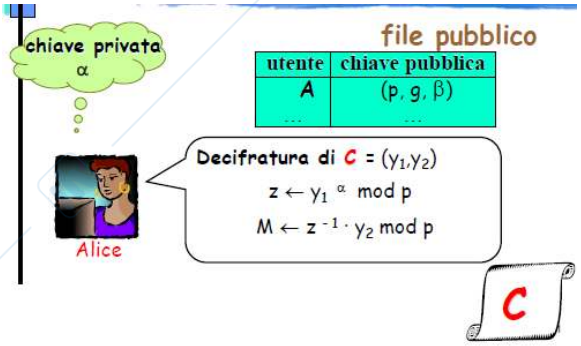


Figura 138: Decifratura

Conoscendo (p, g, β) e sapendo che $\beta = g^{\alpha} \pmod p$, un attaccante vuole calcolare α e sarebbe possibile se sapesse calcolare il logaritmo discreto di $\beta = \alpha \pmod p$. Il problema del logaritmo discreto è un problema che è ampiamente riconosciuto come difficile. Si conoscono algoritmi generici esponenziali, ed algoritmi specifici polinomiali, che funzionano solo se i dati del problema soddisfano determinati criteri aggiuntivi. Non è stato dimostrato nessun lower bound esponenziale per il problema generico. Per di più, non è stato dimostrato il fatto che El Gamal sia legato esclusivamente al logaritmo discreto. Potrebbe cioè esistere un sistema per ottenere M data la chiave pubblica ed il testo cifrato, senza dover ricorrere al calcolo del logaritmo discreto. Quindi, conoscendo la chiave pubblica e il testo cifrato il calcolo del messaggio in chiaro è equivalente al problema di Diffie-Hellman

9.2 Crittosistemi su Curve Ellittiche

Dall'invenzione dei crittosistemi a chiave pubblica (Diffie-Hellman 1976) ad oggi, sono stati proposti numerosi sistemi crittografici. La sicurezza di questi crittosistemi dipende dalla difficoltà del problema matematico su cui essi si basano. Nel corso degli anni molti di essi sono stati "rotti", oggi solo tre tipi di problemi su cui si basano tali sistemi possono essere considerati sicuri ed efficienti. Essi sono:

- Integer Factorization Problem (IFP)
- Discrete Logaritm Problem (DLP)
- Ellipt Curve Discrete Logaritm Problem (ECDLP)

La crittografia su curve ellittiche, è una tecnica molto promettente. I sistemi crittografici basati su curve ellittiche, sono equivalenti all'RSA, con il vantaggio di utilizzare chiavi di lunghezza inferiore. La prima proposta di applicazione di curve ellittiche nei sistemi crittografici, avvenne nel 1985, in maniera indipendente, da parte di Victor Miller (IBM) e Neil Koblitz (University of Washington). In tali crittosistemi le operazioni basate su aritmetica modulare sono sostituite da operazioni su curve ellittiche. Le curve ellittiche sono descritte da equazioni cubiche: $y^2 + axy + by = x^3 + cx^2 + dx + e$, le quali includono un punto all'infinito o punto zero O. La Figura 139 mostra alcuni esempi di curve ellittiche.

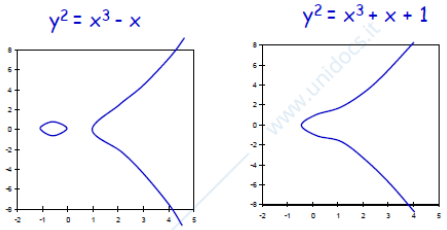


Figura 139: Esempi di curve ellittiche

La regola per addizionare due punti su una curva ellittica $E(F_p)$ è chiamata "regola della corda e della tangente". Il risultato di tale addizione è un terzo punto sulla curva (R). L'insieme dei punti $E(F_p)$ con l'operazione di addizione forma un gruppo, tale gruppo è usato per la costruzione di crittosistemi su curve ellittiche. Vediamo il significato geometrico dell'addizione di due punti $P_1 = (x_1, y_1)$ e $P_2 = (x_2, y_2)$ distinti su una curva ellittica E. La somma di P_1 e P_2, denotata con $R = (x_3, y_3)$, è definita come segue:

- Si disegna una linea tra P_1 e P_2 per trovare il terzo punto di intersezione Q.
- R è il punto di intersezione tra la curva e la linea verticale che attraversa Q.

Se $P = (x_1, y_1)$, $P + P$, denotato con $R = (x_3, y_3)$, è ottenuto nel seguente modo. Si traccia la tangente passante per il punto P la quale interseca la curva in un secondo punto Q, R è il punto di intersezione tra la curva e la linea ortogonale all'asse delle x passante per Q (Figura 140).

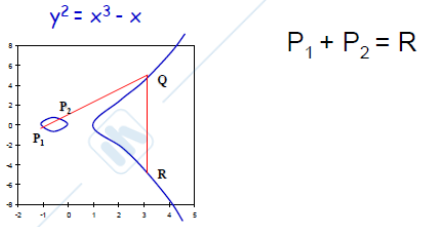


Figura 140: Addizione

Alcune proprietà:

- $P_1 + P_2 = O \rightarrow P_1 = -P_2$
- $P_1 + O = O + P_1 = P_1$

Il concetto di moltiplicazione non è altro che è un'estensione dell'operazione di addizione: kP_1 è uguale al valore P_1 sommato k volte.

Sia $p > 3$ un primo dispari, una curva ellittica E su F_p è definita da una equazione del tipo: $y^2 = x^3 + ax + b \pmod p$, dove a, b sono selezionati in Z_p tali che $4a^3 + 27b^2 \neq 0 \pmod p$. Quindi, l'insieme $E(F_p)$ consiste di tutti i punti (a, b) con $a, b \in F_p$ che soddisfano l'equazione $y^2 = x^3 + ax + b \pmod p$, insieme ad un punto speciale O, chiamato punto all'infinito. Le formule per l'addizione sono le seguenti:

- $x_3 = \lambda^2 - x_1 - x_2$
- $y_3 = \lambda(x_1 - x_3) - y_1$

Dove λ è:

- $\frac{y_2 - y_1}{x_2 - x_1}$ se $P_1 \neq P_2$
- $\frac{3x_1 + a}{2y_1}$ se $P_1 = P_2$

Ad esempio, sia $p = 23$, consideriamo una curva ellittica $E : y^2 = x^3 + x + 4$ definita su F_{23} (in questo caso a = 1 e b = 1). Risulta: $4a^3 + 27b^2 = 4 + 27 = 8 \neq 0 \pmod{23}$. Quindi E è effettivamente una curva ellittica. I punti in $E(F_{23})$ sono O e quelli rappresentati in Figura 141.

(0,1)	(6,4)	(12,19)
(0,22)	(6,19)	(13,7)
(1,7)	(7,11)	(13,16)
(1,16)	(7,12)	(17,3)
(3,10)	(9,7)	(17,20)
(3,13)	(9,16)	(18,3)
(4,0)	(11,3)	(18,20)
(5,4)	(11,20)	(19,5)
(5,19)	(12,4)	(19,18)

Figura 141: Punti per $E_{23}(1, 1)$

La Figura 142 riporta il codice dell'algoritmo per la costruzione dei punti.

```
for x = 0,1,2,... , 22
  if x^3 + x + 1 = 0 (mod 23)
    then output (x,0)
    else if x^3 + x + 1 è un quadrato in Z_23
           then y_1,y_2 ← le 2 radici di x^3 + x + 1 in Z_23
                output (x, y_1), (x, y_2)
```

Figura 142: Algoritmo per costruzione punti per $E_{23}(1, 1)$

La Figura 143 riporta un esempio di moltiplicazione, caso particolare dell'operazione di addizione.

$2 \cdot (17,20) = (17,20) + (17,20)$

$x_3 = \lambda^2 - x_1 - x_2$

$y_3 = \lambda(x_1 - x_3) - y_1$

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & \text{se } P_1 \neq P_2 \\ \frac{3x_1^2 + a}{2y_1} & \text{se } P_1 = P_2 \end{cases}$$

$\lambda = (3(3^2)+1)/(2 \cdot 10) = 5/20 = 6 \bmod 23$

$x_3 = 6^2 - 3 - 3 = 30 = 7 \bmod 23$

$y_3 = 6(3-7) - 10 = -34 = 12 \bmod 23$ $= (7,12)$

Figura 143: Moltiplicazione

Il numero di punti in $E_p(a,b)$ è dato dall'equazione $p+1-2\sqrt{p} \leq |E_p(a,b)| \leq p+1+2\sqrt{p}$. Si utilizza l'algoritmo di Schoof per determinare la cardinalità. La complessità dell'algoritmo è $O((logp)^8)$ operazioni su bit.

Vediamo ora come cambia la crittografia di El Gamal con l'introduzione delle curve ellittiche. La chiave pubblica di Alice è data da una curva ellittica $E_p(a,b)$, un generatore G e P_A. $E_p(a,b)$ e G potrebbero essere comuni a tutti, ciò che caratterizza la chiave di Alice è P_A, che è data da $P_A = x \cdot G$, dove x è un numero che conosce solo Alice. Bob vuole cifrare il messaggio P_M in $E_p(a,b)$ ed inviarlo ad Alice. Bob genera un intero casuale k e calcola:

- $C_1 \leftarrow k \cdot G$
- $C_2 \leftarrow P_M + k \cdot P_A$

Per decifrare il messaggio, Alice non deve far altro che $P_M \leftarrow C_2 - x \cdot C_1$.

Sapendo che:

- $C_2 = P_M + k \cdot P_A$
- $C_1 = k \cdot G$
- $P_A = x \cdot G$

vogliamo controllare che la decifratura sia corretta: $C_2 - x \cdot C_1 = P_M + k \cdot P_A - x \cdot k \cdot G = P_M + k \cdot x \cdot G - x \cdot k \cdot G = P_M$. Quindi la decifratura è corretta.

La Figura 144 riporta le lunghezze consigliate per i vari utilizzi in bit.

	Cifrari a blocchi	RSA	Curve Ellittiche	DSA
Personale	56/64	768	136	768/136
Commerciale	128	1024	160	1024/160
Militare	160	2048	200	2048/200

Figura 144: Lunghezza chiavi in bit

Mentre, la Figura 145 riporta anche il tempo di rottura.

Lunghezza chiave RSA	Tempo per rottura (MIPS anni)	Lunghezza chiave ECC	RSA:ECC rapporto lung. chiavi
512	10^4	106	5:1
768	10^8	132	6:1
1024	10^{11}	160	7:1
2048	10^{20}	210	10:1
21000	10^{78}	600	35:1

Figura 145: Lunghezza chiavi in bit

10 Funzioni hash

Una funzione di hash h, o message digest function, è una funzione che mappa un messaggio arbitrariamente lungo una stringa di lunghezza prefissata, cercando di far in modo che da questa stringa non si possa risalire al messaggio che l'ha generata. La lunghezza della stringa finale è direttamente correlata con la sicurezza della funzione di hash, perché più una stringa è lunga, minore sarà la probabilità di trovare due messaggi con lo stesso digest, con la stessa stringa finale.

La stringa $d = h(m)$ può essere vista come l'impronta digitale del messaggio, teoricamente irripetibile.

Ciò che si chiede da una funzione di hash è:

- l'efficienza computazionale, perché il messaggio m potrebbe essere molto lungo;
- la probabilità che accada $d = h(m)$ per un qualsiasi messaggio casuale m deve essere 2^{-n}, dove il digest è lungo n bit.
- one way property o preimage resistance: dato il digest d, dev'essere computazionalmente impossibile calcolare il messaggio che l'ha generato;
- dato il messaggio m tale per cui $d = h(m)$, deve essere computazionalmente impossibile trovare un altro messaggio m' tale per cui $h(m) = h(m') = d$;
- deve essere computazionalmente impossibile trovare due messaggi m' ed m'' che collidono, qualunque sia il loro digest.

Un'importante applicazione delle funzioni crittografiche di hash è nella verifica dell'integrità di un messaggio. Per mezzo di tali funzioni è possibile determinare, ad esempio, se sono state compiute modifiche ad un messaggio (o ad un file) confrontando il suo hash prima e dopo la trasmissione. Un particolare uso ne viene fatto nella maggior parte degli algoritmi di firma digitale, i quali utilizzano le funzioni di hash al fine di produrre una firma tale da garantire l'autenticità di un messaggio, evitando che un possibile destinatario modifichi un documento firmato da qualcun altro. Per questo motivo il valore di hash viene anche detto impronta digitale del messaggio. La verifica quindi dell'autenticità del valore di hash del messaggio viene considerata come prova di autenticità del messaggio stesso. L'hash è anche utile per dare una certificazione del tempo, mediante un timestamp.

La firma digitale è un metodo matematico teso a dimostrare l'autenticità di un messaggio o di un documento digitale inviato tra mittente e destinatario attraverso un canale di comunicazione non sicuro, garantendo al destinatario che:

- il mittente del messaggio sia chi dice di essere (autenticazione),
- il mittente non possa negare di averlo inviato (non ripudio)
- il messaggio non sia stato alterato lungo il percorso dal mittente al destinatario (integrità).

Le firme digitali si basano su schemi o protocolli crittografici comunemente usati nella distribuzioni di software, nelle transazioni finanziarie e in altri casi in cui si debba rilevare la falsificazione o l'alterazione del messaggio. Un problema sorge nell'apporre una firma digitale di messaggi lunghi. Una soluzione naive potrebbe essere la divisione in blocchi e la firma per ogni blocco, ma questo provoca un problema per la sicurezza, infatti una permutazione/composizione delle firme è una nuova firma. La firma digitale viene realizzata tramite tecniche crittografiche a chiave pubblica insieme all'utilizzo di particolari funzioni matematiche, chiamate funzioni hash unidirezionali. Il processo di firma digitale passa attraverso tre fasi:

- Generazione dell'impronta digitale.
- Generazione della firma.
- Apposizione della firma.

Quindi, la soluzione d'uso corrente consiste nel firmare il valore hash del messaggio: $F_k(h(M))$.Poiché la dimensione del digest message è fissa, e molto più piccola di quella del messaggio originale; la

generazione della firma risulta estremamente rapida. L'utilità dell'uso delle funzioni hash consente di evitare che per la generazione della firma sia necessario applicare l'algoritmo di cifratura, che è intrinsecamente inefficiente, all'intero testo che può essere molto lungo.

Poiché la cifratura di un intero documento è una procedura lunga, ad abbreviare i tempi e conseguire il risultato di assicurare al destinatario che il documento proviene da un determinato soggetto e non è stato alterato, soccorre la procedura dell'impronta che consiste in questo. Al documento viene applicata una determinata funzione, denominata "hash function", che produce un riassunto chiamato "impronta" di lunghezza fissa (20 caratteri), indipendentemente dalle dimensioni dell'originale. L'impronta è unica, nel senso che modificando anche un solo carattere del testo si otterrà un'impronta diversa. L'impronta, e non l'intero documento, viene quindi criptata con la chiave privata del mittente in questo modo si ottiene la generazione della "firma digitale" che verrà apposta al documento originale. Al destinatario vengono spediti: il documento con la "firma digitale" in calce e il certificato rilasciato dalla competente autorità di certificazione a garanzia della titolarità della chiave pubblica necessaria per decriptare la firma digitale. Chi riceve il messaggio procederà all'apertura e/o verifica dello stesso mediante il proprio software per l'attività di firma. Il programma acquisirà dal certificato annesso al documento firmato, la chiave pubblica del mittente. Con tale chiave viene decifrata la stringa della "firma digitale" che darà come risultato l'impronta del documento. Il destinatario prenderà il documento originario, lo farà passare attraverso la funzione di "hash" e genererà l'impronta: se coincide con quella decrittata del mittente allora sarà sicuro dell'integrità e provenienza del documento.

La certificazione del tempo la ottengo generando l'hash del messaggio unito alla data di creazione. Se qualcuno modifica la data, anche l'hash cambierà, e il destinatario se ne accorge.

Tutte le cose viste qui sopra sono dovute a tre proprietà:

- One-way (pre-image resistant) - Resistenza alla preimmagine. La proprietà di one-wayness stabilisce che è impossibile risalire da un valore hash al messaggio che l'ha generato. Ciò vuol dire che se ho il messaggio M, ed il suo hash è $h(M)$, avendo il solo $h(M)$, non è possibile stabilire niente del messaggio M. Sopra abbiamo detto che le funzioni hash prendono in input un messaggio di lunghezza arbitraria, e generano un output di lunghezza fissa, che supponiamo essere di n bit. Ciò significa che tutti gli infiniti messaggi vengono mappati dalla funzione hash in "soli" 2^n valori. Questo comporta che è perfettamente plausibile che due messaggi diversi possano avere lo stesso hash. Ciononostante, la one-wayness garantisce che, dato un hash, non è possibile dire niente a proposito del messaggio che lo origina. Infatti, ci saranno infiniti messaggio che vengono mappati nello stesso valore hash, proprio per il fatto che l'hash ha una dimensione finita.

- Weak collision resistance (2nd pre-image resistance) - Resistenza alla seconda preimmagine. La weak collision resistance sostiene che, se si possiede un messaggio M ed il corrispondente valore hash $h(M)$, non è possibile calcolare in qualche modo un messaggio Z tale che $h(Z) = h(M)$. In pratica, non è possibile trovare un messaggio che dia come valore hash lo stesso hash di un messaggio che si possiede. La collisione si ha quando due messaggi diversi hanno lo stesso hash.

- strong collision resistance (collision resistance) - Resistenza alla collisione. La strong collision resistance è una proprietà per cui non si è in grado di calcolare ex-novo due messaggi, M e Z, che diano lo stesso hash. Da notare la differenza con la weak: la weak parte dal presupposto che l'utente possegga un M e il suo hash. In questo caso non si ha in mano nulla e si vuole fare il "miracolo" di generare con un algoritmo due valori x e y che diano lo stesso hash.

Una One-Way Hash Function (OWHF) verifica le proprietà pre-image e 2nd preimage resistant; viene detta weak one-way hash function. Una Colllision Resistant Hash Function (CRHF) verifica la proprietà di collision resistance e viene anche detta strong one-way hash function.

Una funzione f è One-Way (OWF) se per ogni x nel dominio di f è facile calcolare $y = f(x)$, ma, dato y è computazionalmente inammissibile trovare x tale che $y = f(x)$. Le differenze con OWHF sono:

- Non ci sono limitazioni sul codominio (non necessariamente comprime)
- Non è richiesta la sicurezza debole (2nd preimage)

La sicurezza forte implica la sicurezza debole (collision → 2nd preimage): sia h collision resistant e fissa un input x_1. Se h non è anche 2nd pre-image, allora è possibile trovare x_2 tale che $h(x_1) = h(x_2)$. Quindi, (x_1, x_2) è una coppia di collisioni.

Le proprietà si implicano a vicenda: strong → weak → one-wayness.

É possibile dimostrare che un hash collision resistant deve essere one-way. Si prova per contraddizione: se esiste un algoritmo di inversione, allora esiste un algoritmo Las Vegas (dà una risposta esatta o non risponde) che trova collisioni con probabilità $\geq \frac{1}{2}$.

La Figura 146 mostra la dimostrazione di quanto detto in cui $h : X \to Z$ funzione hash, $|X| \geq 2 \cdot |Z|$. Supponiamo che ALG sia un algoritmo di inversione per h, allora esiste un algoritmo Las Vegas che trova collisioni con probabilità $\geq \frac{1}{2}$.

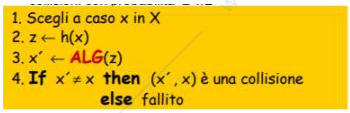


Figura 146: Sicurezza forte → one-way

Quanto grande l'hash per la sicurezza forte? Se $|Z| = 23$ bit, quanti valori scegliere per essere certi di trovare almeno una collisione? Con $|Z| + 1$ diversi valori di M, si ha la certezza di trovare almeno una collisione.

Il paradosso afferma che la probabilità che almeno due persone in un gruppo compiano gli anni lo stesso giorno è largamente superiore a quanto potrebbe dire l'intuito: infatti già in un gruppo di 23 persone la probabilità è circa 0,51 (51%); con 30 persone essa supera 0,70 (70%), con 50 persone tocca addirittura 0,97 (97%), anche se per arrivare all'evento certo occorre considerare un gruppo di almeno 366 persone (367 se si considera l'anno bisestile). Scegliamo a caso elementi in un insieme di cardinalità n. Quanti elementi scegliere se si vuole che la probabilità che ci siano almeno due elementi uguali sia ε? $t \approx \sqrt{n \cdot 2ln(\frac{1}{1-\varepsilon}}$

Torniamo alle proprietà delle nostre funzioni hash: quanti messaggi casuali devo generare per trovarne due che hanno lo stesso hash, cioè avere una collisione? La risposta è che devo generare in media $2^{\frac{n}{2}}$ messaggi, dove n è la lunghezza in bit dell'output della funzione hash. Se una funzione hash ha un output di 64 bit, vuol dire che devo generare 232 messaggi in media per avere una collisione: con la tecnologia odierna, tutti i calcoli che sono inferiori asintoticamente a 2^{80} sono potenzialmente insicuri. Quindi, una funzione hash con un output di 64 bit è altamente insicura. Ecco perché le funzioni hash moderne hanno output di almeno 160 bit.

Il paradosso del compleanno può essere utilizzato per attaccare le funzioni hash. Se il digest prodotto è di n bit genera r_1 e r_2, varianti del messaggio originale. La probabilità di trovare una coppia di varianti (una di r_1 e una di r_2) che producono lo stesso digest è $> \frac{1}{2}$ quando la funzione hash è applicata a k. Quindi, in che modo Cattivo può sfruttare questa proprietà a suo vantaggio? Oscar prepara un contratto m corretto ed uno m' fraudolento, poi trova un certo numero di punti del contratto per cui m può essere modificato senza cambiarne il significato, ad esempio inserendo delle virgole, delle linee vuote, uno o due spazi dopo una frase, sostituendo dei sinonimi ecc. Combinando queste modifiche, Oscar può creare un notevole numero di varianti di m (dell'ordine di 2^k, dove k è il numero delle modifiche) che sono in pratica tutti contratti corretti. In maniera simile crea anche un gran numero di varianti del contratto fraudolento m'. Alla fine applica la funzione hash a tutte queste varianti finchè non ne trova una del contratto

corretto e una di quello fraudolento che hanno lo stesso digest, cioè tali che $f(m) = f(m')$. A quest punto, Oscar presenta a Luca la versione corretta per la firma. Dopo che Luca lo ha firmato, Oscar prende la firma e la attacca al contratto fraudolento. Adesso la firma prova che Luca ha firmato il contratto fraudolento. La Figura 183 mostra come si possa creare un gran numero di documenti con ugual significato, ma con digest diverso.

Figura 147: Varianti di uno stesso documento con digest diverso

Per evitare questo attacco la lunghezza dell'output di una funzione hash utilizzata in uno schema di firma digitale, deve essere grande abbastanza in modo che l'attacco del compleanno divenga computazionalmente impossibile. In genere si è nell'ordine del doppio dei bit necessari per prevenire un classico attacco a forza bruta.

Gli oracoli casuali sono in genere utilizzati come sostituti ideali delle funzioni di hash crittografiche in schemi in cui sono necessari forti presupposti di casualità dell'output della funzione hash. Quindi, il modello random oracle è un modello matematico per una funzione hash ideale: $h : X \to Y$. h è una funzione scelta a caso fra tutte le funzioni che mappano X in Y. L'unico modo per calcolare il valore $h(x)$ è consultare un oracolo, ossi consultare una tabella che riporta il valore $h(x)$ scelto in maniera casuale.

Una funzione di hash deve essere in grado di elaborare un messaggio di lunghezza arbitraria in un output di lunghezza fissa: ciò può essere ottenuto spezzando l'input in una serie di blocchi di uguale dimensione e operando su di essi in sequenza o in parallelo, usando una funzione di compressione a senso unico. La funzione di compressione può essere progettata appositamente per l'hashing o per essere costruita da una cifratura a blocchi. In crittografia, la costruzione di Merkle-Damgård è un metodo per costruire delle funzioni crittografiche di hash resistenti alle collisioni utilizzando delle funzioni a compressione a senso unico. Questa particolare costruzione è stata utilizzata nell'implementazione di molti algoritmi di hash, come ad esempio MD5, SHA1 e SHA2. Ralph Merkle e Ivan Damgård dimostrarono, separatamente, che questa struttura effettivamente funziona, e cioè che, con un appropriato schema e con una funzione di compressione resistente alle collisioni, la funzione hash che si ottiene è anch'essa resistente alle collisioni. La trasformazione consiste nel costruire una funzione hash a partire da una funzione di compressione unidirezionale resistente alle collisioni. Questo metodo rende possibile una conversione da una qualsiasi funzione hash di lunghezza fissata (ossia da una funzione di compressione), a funzioni hash vere e proprie, mantenendo la proprietà di essere resistenti alle collisioni.

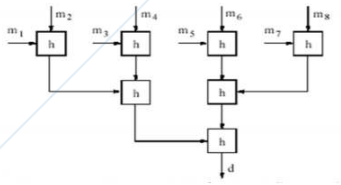


Figura 148: Modello generale Hash parallelo

La maggior parte delle funzioni hash sono progettate come programmi iterativi per i quali si ha che input di lunghezza arbitraria sono processati trasformandoli in blocchi successivi di dimensione fissata. Un input x di lunghezza finita ed arbitraria è diviso in blocchi x_i di lunghezza fissata, r-bit. Questa preelaborazione tipicamente richiede di appendere dei bit extra (padding) poiché è necessario ottenere una lunghezza in bit complessiva che è un multiplo della lunghezza del blocco di r bit, e spesso include, per ragioni di sicurezza un blocco o blocco parziale indicante la lunghezza dell'input prima del padding. Ogni blocco x_i, quindi, serve come input ad una funzione hash f di dimensione fissata, la funzione di compressione di h, che calcola un nuovo risultato intermedio di lunghezza n bit per qualche n fissato, come funzione del risultato intermedio di n bit e il prossimo blocco x_i . Pertanto se H_i è il risultato parziale dopo la fase i, il processo generale per una funzione hash iterata con input $x = x_1x_2...x_n$ può essere modellato come segue:

$$H_0 = IV$$

$$H_i = f(x_i, H_{i-1})$$

$$H_n = f(x_n, H_{n-1})$$

H_{i-1} serve come variabile di n bit di concatenamento tra la fase $i-1$ e la fase i, e H_0 è un valore pre-definito di partenza o valore di inizializzazione (IV).

Figura 149: Modello generale hash iterate

Una collisione per h implica una collisione per f.

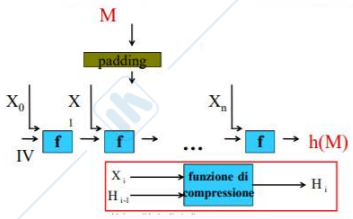


Figura 150: Modello generale hash iterate

Nel modello a cascata, le funzioni hash sono composte in "parallelo". Trovare una collisione per $H(M) = H1(M)$. $H2(M)$ significa trovare una collisione sia per $H1$ che per $H2$.

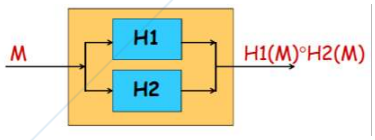


Figura 151: Funzioni hash a cascata

Da un punto di vista strutturale le funzioni hash possono essere classificate in base alla natura delle operazioni che compongono le loro funzioni interne di compressione. Le tre categorie di funzioni hash iterate sono le funzioni hash basate sui cifrari a blocchi, le funzioni hash su misura, e le funzioni hash basate sulla aritmetica modulare. Le funzioni hash su misura sono quelle progettate specificamente per l'hashing, con l'obiettivo della velocità ed indipendenti da altre componenti di sistema (per esempio cifrari a blocchi). Una motivazione pratica per costruire funzioni hash da cifrari a blocchi è che se un'efficiente implementazione di un cifrario a blocchi è già disponibile in hardware o in software dentro un sistema, allora usarlo come componente centrale per una funzione hash può fornire una seconda funzionalità ad un piccolo costo addizionale. Lo scopo è che un buon cifrario a blocchi può servire come un blocco di costruzione per la creazione di una funzione hash con proprietà adeguate per varie applicazioni. Inoltre, le costruzioni di funzioni hash sono date con sicurezza provabile assumendo certe proprietà ideali del cifrario a blocchi usato. Naturalmente, i cifrari a blocchi non posseggono le proprietà delle funzioni random (per esempio, essi sono invertibili). Per giunta i cifrari a blocchi esibiscono delle proprietà addizionali di regolarità o punti deboli. Per esempio per un cifrario a blocchi E, la doppia cifratura usando una cascata con cifratura-decifratura con chiavi K_1, K_2 risulta una funzione identità se $K_1 = K_2$. In sintesi, mentre varie condizioni necessarie sono conosciute, non è chiaro esattamente quali siano i requisiti di un cifrario a blocchi che sono sufficienti per costruire una funzione hash sicura, e proprietà adeguate per un cifrario a blocchi (esempio resistenza ad un attacco chosen-text) possono non garantire una buona funzione hash. I tre schemi che presentiamo sono funzioni hash a singola lunghezza basati su cifrari a blocchi. Questi possono usare le seguenti tre componenti predefinite:

- un cifrario generico E_k a blocchi di n-bit parametrizzato da una chiave K;
- una funzione g che associa ad input di n-bit chiavi K adatte per E (se le chiavi per E sono anche di lunghezza n, g può essere la funzione identità);
- un valore iniziale fissato IV (solitamente di n-bit), adatto per uso con E.

$M = M_1...M_t$ rappresenta il messaggio, eventualmente con l'aggiunta di padding. Mentre, H_0 è una costante predefinita e H_t è il valore di hash. Alcuni metodi per trasformare un normale cifrario a blocchi in una funzione di compressione a senso unico sono:

- Davies-Meyer,
- Matyas-Meyer-Oseas,
- Miyaguchi-Preneel (funzioni di compressione con lunghezza singola del blocco)

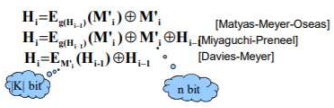


Figura 152: Funzioni Hash basate su cifrari a blocchi

L'algoritmo Matyas-Meyer-Oseas riceve in input una stringa di bit x e fornisce come output un valore di hash di t-bit di x. Si tratta di un cifrario a blocchi con block size uguale a t e key size uguale ad m. La funzione g trasforma un blocco di t bit in una chiave di m bit. In caso di AES, abbiamo $t = m$. L'input x è diviso in blocchi di n-bit ed effettuato il padding, se necessario, per completare l'ultimo blocco. Si denoti il nuovo messaggio consistente di n blocchi di t-bit: $x_1x_2...x_n$. Una costante iniziale IV di t-bit deve essere pre-specificata. L'output è H_n definito da: $H_0 = IV$ e $H_i = E_{g(H_{i-1})}(x_i) \oplus x_i$.

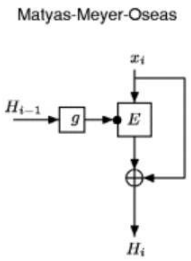


Figura 153: Matyas-Meyer-Oseas

Per gli hash che hanno una struttura a catena, può essere utilizzata una variante del birthday attack (attacco del compleanno), con lo scopo di generare un falso messaggio $R = (r_1, r_2)$ diverso da $M = (m_1, m_2)$ con $h(R) = h(M)$. L'attacco si svolge in questo modo: genera r_1 varianti della prima parte e vai alla fase intermedia; genera r_2 varianti della seconda parte e vai indietro alla fase intermedia; Trova una collisione (la probabilità di successo è la stessa del birthday attack). Considera lo schema di hash per $M = (m_1, m_2)$: $h_1 = E(m_1, IV)$, $d = h(m) = E(m_2, h_1)$, con E cifratura e IV valore iniziale. L'attacco consiste nel trovare $M^* = (m_1^*, m_2^*)$ collida con M:

- Trova $i = 1, \ldots, r_1$ varianti $m_{1,j}^*$, e calcola $h_1^*,i = E(m_1^*, IV)$;
- Trova $i = 1, \ldots, r_2$ varianti $m_{2,j}^*$, e calcola $h_2^*,i = E(m_2^*, IV)$;

La probabilità di trovare un match tra $h_{1,i}^*$ e $h_{1,i}^*$ è la stessa probabilità di successo in un birthday attack. Vale lo stesso risultato se si utilizzano più round.

10.1 MD4

L'MD4 è una funzione crittografica di hashing. L'MD4 è utilizzato per la generazione di un message digest (o "impronta del messaggio", una stringa di lunghezza fissa) di 128 bit da un messaggio di lunghezza variabile. L'algoritmo ha influenzato successivi codici quali l'MD5, l'SHA ed il RIPEMD. L'algoritmo non è sicuro ed il suo uso è pertanto sconsigliato in applicazioni in cui si richiede un elevato grado di sicurezza.MD4 fu introdotto da Ron Rivest nel 1990, le prime 2 lettere del nome sono: MD la sigla di message digest, il termine che in letteratura viene usato per indicare l'output di un algoritmo di hashing. Le caratteristiche salienti di MD4 sono: velocità, compattezza e l'uso di tabelle di memorizzazione di ridotte dimensioni, ma purtroppo non è sicuro. L'algoritmo richiede che sia soddisfatto un piccolo insieme di condizioni preliminari: è scritto per architetture di tipo little-endian, con parole di 32 bit, la lunghezza massima dell'input è 264 bit e il valore hash restituito ha dimensione 128 bit. Di seguito viene riportato un breve glossario dei termini usati:

- Parola macchina o parola: unità di misura della unità della macchina sottostante corrispondente a 32 bit.
- Little-endian: Supponiamo di avere una parola macchina a 32 bit, questa sarà composta da 4 byte, indichiamoli con $b1, b2, b3, b4$, il valore numerico della parola sarà dato da: $b4 \cdot 2^{24} + b3 \cdot 2^{16} + b2 \cdot 2^8 + b1$.

- Big-endian: La situazione è identica a quella suddetta, ma il valore numerico sarà dato da: $b1 \cdot 2^{24} + b2 \cdot 2^{16} + b3 \cdot 2^8 + b4$.

Gli obiettivi di progettazione sono:

- Sicurezza forte: computazionalmente difficile trovare 2 messaggi con lo stesso valore hash
- Sicurezza diretta: sicurezza non basata su problemi teorici difficili computazionalmente
- Velocità: algoritmo adatto per implementazioni software molto veloci
- Semplicità e Compattezza: semplice da descrivere e da implementare

MD4 funziona con parole a 32 bit: sia M il messaggio da hashare. Il messaggio M è sistemato in modo che la sua lunghezza (in bit) sia uguale a 448 modulo 512, ovvero il messaggio, a seguito del processo di padding, è 64 bit in meno di un multiplo di 512. Il riempimento è costituito da un singolo bit 1, seguito da abbastanza zeri per riempire il messaggio per la lunghezza richiesta. Il padding viene sempre utilizzato, anche se la lunghezza di M è uguale a 448 mod 512. Di conseguenza, c'è almeno un bit di padding e al massimo 512 bit di padding. Quindi la lunghezza (in bit) del messaggio (prima del riempimento) viene aggiunta come blocco a 64 bit.

Il messaggio modificato è un multiplo di 512 bit e, pertanto, è anche un multiplo di 32 bit. Sia M il messaggio e N il numero di parole a 32 bit nel messaggio (imbottito). A causa del padding, N è un multiplo di 16.

Un buffer di quattro parole (A, B, C, D) viene utilizzato per calcolare il digest del messaggio. A, B, C, D rappresentano un registro a 32 bit. Questi registri sono inizializzati ai seguenti valori in esadecimali:

- word A: 01 23 45 67
- word B: 89 ab cd ef
- word C: fe dc ba 98
- word D: 76 54 32 10

Definiamo tre funzioni ausiliarie che prendono ciascuna come input tre parole a 32 bit e producono come output una parola a 32 bit (Figura 154). Il cuore dell'algoritmo, cioè l'elaborazione del messaggio, è diviso in 3 round, ognuno dei quali svolge 16 operazioni, ognuno dei round usa 1 sola delle 3 funzioni logiche definite nella Figura.

$$F(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$
$$G(X, Y, Z) = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$$
$$H(X, Y, Z) = X \oplus Y \oplus Z$$

Figura 154: Operazioni nei tre round di MD4

MD4 è stato oggetto di molti attacchi:

- crittoanalisi dei primi 2 round: Merkle ha provato che è facile trovare collisioni con round 3 omesso
- crittoanalisi degli ultimi 2 round: den Boer e Bosselaers [Crypto '91] hanno trovato collisioni con round 1 omesso
- Settembre 1995: Dobbertin [FSE 1996] ha trovato collisioni per MD4 con un PC in pochi secondi

10.2 MD5

Nella crittografia, MD5 (algoritmo Message-Digest 5) è una funzione di hash crittografica ampiamente utilizzata con un valore di hash a 128 bit. Come standard Internet (RFC 1321), MD5 è stato utilizzato in un'ampia varietà di applicazioni di sicurezza e viene anche comunemente utilizzato per verificare l'integrità dei file. Un hash MD5 è in genere espresso come un numero esadecimale di 32 cifre.

MD5 è una versione rafforzata di MD4. Come MD4, l'hash MD5 è stato inventato dal professor Ronald Rivest del MIT. MD5 è basato su MD4, rispetto a quest ultimo è più lento, ma più sicuro, le principali differenze possono essere riassunte in:

- Un IV round di 16 operazioni.
- L'ordine d'accesso alle parole nei round 2 e 3.
- Il numero delle operazioni di shift.
- La funzione del round 2 diventa $G(X, Y, Z) = (X \wedge Y) \vee (Y \wedge \neg Z)$
- Ad ogni passo è usata una costante additiva differente. La Figura 155 mostra le costanti additive per l'MD5.

1	d76aa478	17	f61e2562	33	fffa3942	49	f4292244
2	e8c7b756	18	c040b340	34	8771f681	50	432aff97
3	242070db	19	265e5a51	35	6d9d6122	51	ab9423a7
4	c1bdceee	20	e9b6c7aa	36	fde5380c	52	fc93a039
5	f57c0faf	21	d62f105d	37	a4beea44	53	655b59c3
6	4787c62a	22	02441453	38	4bdecfa9	54	8f0ccc92
7	a8304613	23	d8a1e681	39	f6bb4b60	55	ffeff47d
8	fd469501	24	e7d3fbc8	40	bebfbc70	56	85845dd1
9	698098d8	25	21e1cde6	41	289b7ec6	57	6fa87e4f
10	8b44f7af	26	c33707d6	42	eaa127fa	58	fe2ce6e0
11	ffff5bb1	27	f4d50d87	43	d4ef3085	59	a3014314
12	895cd7be	28	455a14ed	44	04881d05	60	4e0811a1
13	6b901122	29	a9e3e905	45	d9d4d039	61	f7537e82
14	fd987193	30	fcefa3f8	46	e6db99e5	62	bd3af235
15	a679438e	31	676f02d9	47	1fa27cf8	63	2ad7d2bb
16	49b40821	32	8d2a4c8a	48	c4ac5665	64	eb86d391

Figura 155: Costanti additive MD5

- L'output del passo precedente è sommato al passo corrente.

Le operazioni preliminari dell'algoritmo sono le stesse del MD4, anche le costanti che vengono inserite nei registri A, B, C, D sono identiche. Il IV round usa una nuova funzione: $K(X, Y, Z) = Y \oplus (X \vee \neg Z)$. La funzione g, che calcolava la maggioranza dei 3 parametri in input, è stata sostituita a causa della sua simmetria. La notazione, $[ABCD\ K\ S\ I]$ rappresenta l'operazione: $A = B + ((A + F(B, C, D) + X[K] + T[I]) << S)$. L'operazione di somma rappresenta la somma intera modulo 2^{32}, mentre $<<$ rappresenta lo shift a sinistra di 2 posti. La Figura 156 mostra il funzionamento dell'algoritmo di MD5.

Quindi, ogni round consiste di 16 operazioni $[ABCD\ K\ S\ I]$. Ogni operazione agisce sul buffer di 4 word ABCD: $A = B + ((A + F(B, C, D) + X[K] + T[I]) << S)$, dove:

- K è l'indice della parola,
- s indica lo shift ciclico,
- i è l'indice dell'iterazione,
- W è la funzione del round (F, G, H, I)
- $X[k] \leftarrow M'[16i + k]$ è la k-esima word di 32 bit nell'iesimo blocco
- $T[i]$ è l'iesimo elemento della tabella di 64 valori

```
A ← 67452301; B ← efcdab89;  C ← 98badcfe;
D ← 10325476;
for i=0  to  (N/16)-1  do
  for j=0  to  15 do  X[j] ← M'[16i+j]
  AA ← A;  BB ← B;  CC ← C;  DD ← D;
   round 1;
   round 2;
   round 3;
   round 4;
   A ← A+AA;  B ← B+BB;  C ← C+CC;  D ← D+DD;
output: (A, B, C, D)
```

Figura 156: Algoritmo MD5

La Figura 157 riassume le differenze tra l'algoritmo MD4 e MD5.

MD5	MD4
4 round con 4 ·16 operazioni	3 round con 3 ·16 operazioni
4 funzioni logiche	3 funzioni logiche
64 costanti additive	2 costanti additive
ogni passo aggiunge il risultato del passo precedente	non accade

Figura 157: Differenze MD4 e MD5

10.3 SHA

Lo standard americano per l'hashing: SHS (Secure hashing standard) è una variante di MD4 fu introdotto dal Nist nel 1993 e modificato nel 1994. L'algoritmo che implementa lo standard è SHA (Secure hashing algorithm), la sua versione modificata è: SHA-1, la cui unica differenza consiste nell'aggiunta di uno shift nell'espansione dei blocchi. L'input è un messaggio M di lunghezza inferiore a 264 bit, e produce in output un messaggio X di 160 bit che continueremo ad indicare come message digest. Si tratta di operazioni efficienti su architetture 32 bit big-endian. Ha gli stessi principi di MD4 ed MD5, ma è più sicuro.

La prima operazione dell'algoritmo è il padding dell'input identico a quello eseguito in MD4, alla stringa iniziale è concatenato un singolo bit 1 e un numero opportuno di bit 0 in modo da rendere la lunghezza multipla di 512, a questa nuova stringa è aggiunta la rappresentazione binaria a 64 bit della lunghezza del messaggio originale, per ottenere un digest di 160 bit sono usati 5 registri: A, B, C, D, E ognuno di 32 bit, l'algoritmo è definito per architetture di tipo big-endian. Il messaggio M esteso può essere considerato come un vettore $M(1), ..., M(n)$ dove il singolo elemento è composto da gruppi di parole per 512 bit. Ci sono delle funzioni logiche $f_0, .., f_{79}$ che lavorano su gruppi di 3 word, ne sono definite 4 diverse nel seguente modo (Figura 158).

Intervallo	Valore
0..19	$(X \wedge Y) \vee (\neg X \wedge Z)$
20..39	$X \oplus Y \oplus Z$
40..59	$(X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$
60..79	$Y \oplus (X \vee \neg Z)$

Figura 158: Funzioni logiche in SHS

Allo stesso modo delle funzioni viene definita una sequenza di costanti additive a 32 bit, indicate come $C_0, C_1, .., C_{79}$ (Figura 159)

Intervallo	Valore
0..19	*5A827999*
20..39	*6ED9EBA1*
40..59	*8F1BBCDC*
60..79	*CA62C1D1*

Figura 159: Costanti additive in SHS

La sequenza di bit "messaggio+imbottitura+lunghezzaMessaggio" viene divisa in blocchi da 512bit, che identificheremo con B_n con n che va da 0 a L. Il fulcro dell'algoritmo SHA-1 è chiamato compression function ed è formato da 4 cicli di 20 passi cadauno. I cicli hanno una struttura molto simile tra di loro se non per il fatto che utilizzano una differente funzione logica primitiva. Ogni blocco viene preso come parametro di input da tutti e 4 i cicli insieme ad una costante K e i valori dei 5 registri. Alla fine della computazione otterremo dei nuovi valori per A, B, C, D, E che useremo per la computazione del blocco successivo sino ad arrivare al blocco finale F. Il passo di espansione nella sua versione revisionata introduce uno shift circolare di 1 posizione. L'operazione risultante è: $X[I] = (X[I-3]X[I-8]X[I-14]X[I-16]) << 1$. Dopo l'espansione il messaggio può essere visto come un insieme di n blocchi M(i) con $1 < i < n$, ognuno della dimensione di 80 parole. Il risultato della funzione è ottenuto mediante n applicazioni dell'algoritmo riportato in Figura 160, dove il risultato dell'i-esimo passo fa da vettore d'inizializzazione per l'$(i+1)$-esimo passo.

Figura 160: Algoritmo SHA-1

Attraverso le quattro proprietà presentate con la descrizione di MD4, vediamo un confronto tra MD5 e SHA-1:

- Sicurezza forte: maggiore in SHA-1, output 32 bit più lungo di MD4/5 (160 contro 128). Attacco a compleanno 80 bit contro 64 bit.
- Sicurezza contro l'analisi: MD5 è soggetta ad alcuni attacchi
- Velocità: entrambi algoritmi molto veloci; SHA-1 ha più passi (80 contro 64) e il buffer ha 160 rispetto ai 128 bit di MD5.
- Semplicità e Compattezza: semplice da descrivere e da implementare, nessun uso di tabelle e di complesse strutture dati.

10.3.1 SHA-256, SHA-512, SHA-384

Gli algoritmi che abbiamo visto producono un valore hash di lunghezza 128 bit (MD4-MD5) e 160 bit (SHA-1) e permettono una sicurezza di 64 e 80 bit rispettivamente, rispetto all'attacco del compleanno. Lo scopo del nuovo Advanced Encryption Standard è stato quello di offrire alle variabili crittografiche una sicurezza di 128, 192 e 256 bit. Questo ha costretto ad apportare modifiche agli algoritmi hash in modo da ottenere tali livelli di sicurezza. Così abbiamo nuovi algoritmi (SHA-256, SHA-512, SHA-384) descritti di seguito, che sono stati sviluppati e proposti nel Draft Federal Information Processing Standard (FIPS)180-2 nel 2001.

SHA-256 restituisce un valore hash di 256 bit e garantisce una sicurezza di 128 bit rispetto all'attacco del compleanno; SHA-512 restituisce un valore hash di 512 bit e garantisce una sicurezza di 256 bit rispetto all'attacco del compleanno; ed infine SHA-384 restituisce un valore hash di 384 bit e garantisce una sicurezza di 192 bit.

SHA-256 opera nello stesso modo degli algoritmi MD4, MD5 e SHA-1. Processa il messaggio in blocchi di 512 bit. Ogni blocco consta di 16 parole da 32 bit.

SHA-512 è una variante di SHA-256. Processa il messaggio in blocchi di 1024 bit. Ogni blocco consta di 16 parole da 64 bit.

SHA-384 è definito nello stesso modo di SHA-512, ma il risultato finale di 384 bit è ottenuto troncando l'output di 512 bit. I bit che fanno parte del nuovo output vengono presi da sinistra.

Nel 2005 il NIST ha annunciato di voler abolire SHA-1, per passare allo SHA-512, che presenta le seguenti migliorie:

- attacco del compleanno: collisioni con 2^{69} operazioni, meno di 2^{80}
- Output a 512 bit
- Messaggio diviso in blocchi di 1024 bit = 128 word
- Parole da 64 bit
- Buffer: 8 registri da 64 bit

10.4 Whirpool

Whirlpool è una funzione di hash crittografico. É stato progettata da Vincent Rijmen (co-creatore della Advanced Encryption Standard (AES)) e Paulo Barreto SLM, che per primo la descrisse nel 2000. É stata raccomandata nel progetto NESSIE. É stata anche adottata dalla International Organization for Standardization (ISO) e l'International Electrotechnical Commission (IEC), come parte del giunto ISO / IEC 10.118-3 standard. Whirlpool prende un messaggio di qualsiasi lunghezza inferiore a 2^{256} bit e restituisce un 512 bit message digest. É basata su una cifratura a blocchi W a sua volta derivata da AES, in cui:

- H_0
- $H_i = W_{Hi-1}(m_i) \oplus H_{i-1} \oplus m_i$

10.5 Conclusioni

Altre funzioni di hash conosciute sono:

- Snefru, Ralph Merkle [1990], 128 oppure 256 bit
- N-hash, Nippon Telephone and Telegraph [1990], 128 bit
- HAVAL, Zheng-Pieprzyk-Seberry [1992] 128-160-192-224-256 bit
- FFT-hash I, C. Schnorr [1991], rotto dopo pochi mesi

- FFT-hash II, C. Schnorr [1992], rotto dopo poche settimane
- RIPEMD-160 Bosselaers, Dobbertin, Praeneel [1996]

In Luglio 2004 sono state annunciate collisioni in MD4, MD5, HAVAL-128 e RIPEMD (Wang et al Crypto 2004). In Febbraio 2005, Wang et al hanno presentato un attacco contro SHA-1 che impiega 269 calcoli di hash (differential path attack). Antoine Joux ha presentato collisioni in SHA-0. Nel 2012, Mark Stevens ha usato una serie di cloud servers per portare un attacco differential path su SHA-1, producendo una near-collision dopo 258.5 cicli. Una stima prevede una fullcollision dopo 261 cicli. Per SHA-2, gli attacchi si limitano a versioni limitate sui round SHA-2. L'attacco più effettivo è contro un 46-round SHA-2 (512-bit variant) e contro un 41-round SHA-2 (256-bit variant). Necessita di 2253.6 cicli per rompere la variante a 256-bit e 2511.5 cicli per la variante a 512-bit. Nel 2008, NIST ha lanciato la competizione per SHA-3, con i seguenti requisiti:

- La funzione hash candidata doveva funzionare bene indipendentemente dall'implementazione.
- La funzione candidata doveva essere prudente riguardo alla sicurezza. Dovrebbe resistere ad attacchi noti. Dovrebbe emettere le stesse quattro dimensioni di hash di SHA-2 (larghezza 224-, 256-, 384- o 512-bit)
- La funzione candidata deve essere sottoposta a crittoanalisi.
- La funzione candidata doveva esercitare la diversità del codice. Non è stato possibile utilizzare il motore Merkle-Damgard per produrre l'hash del messaggio.

La competizione per SHA-3 ha visto 51 candidati nel primo round, 14 sono passate al secondo round. Il terzo round ha ristretto la rosa a 5 candidati e alla fine Keccak («ket-chak») è il vincitore (Ottobre 2012). Nell'ottobre 2012, l'Istituto Nazionale di Standard e tecnologia (NIST) scelse l'algoritmo Keccak come il nuovo SHA-3. standard.

10.6 Marca temporale (timestamp)

La marca temporale di un documento è qualcosa aggiunto ad esso che prova che il documento è stato "prodotto" prima, dopo oppure ad un fissato momento. Una marca temporale (timestamp) è una sequenza di caratteri che rappresentano una data e/o un orario per accertare l'effettivo avvenimento di un certo evento. La data è di solito presentata in un formato compatibile, in modo che sia facile da comparare con un'altra per stabilirne l'ordine temporale. La pratica dell'applicazione di tale marca temporale è detto timestamping. La marcatura temporale è il processo di generazione e apposizione di una marca temporale su un documento informatico, digitale o elettronico. Il processo di marcatura temporale consiste nella generazione, da parte di una terza parte fidata (il Certificatore accreditato), di una "firma digitale del documento" cui è associata l'informazione relativa ad una data e ad un'ora certa. L'apposizione della marca temporale consente di stabilire l'esistenza di un documento informatico a partire da un certo istante e di garantire la validità nel tempo. Alcune situazioni in cui è utile il timestamp sono:

- Depositare il documento presso un notaio
- Inviare il documento a se stesso, tramite il servizio postale
- Brevetto (se brevettabile...)
- Pubblicare il documento su di un giornale
- Uso di un registro di protocollo

- Foto con un quotidiano (se è un sequestro...)

É in genere facile provare che un documento è stato prodotto dopo una data fissata, mentre è in genere difficile provare che un documento è stato prodotto prima di una data fissata.

La Figura 161 mostra una soluzione naive all'apposizione della marca temporale.

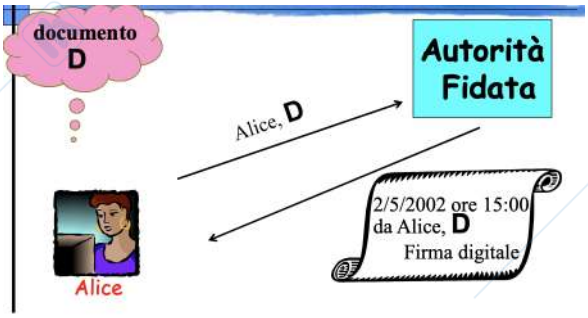


Figura 161: Soluzione naive per la marca temporale

I problemi che sorgono dall'utilizzo di questa soluzione sono:

- Dimensioni del documento D (per la comunicazione e per la memorizzazione dell'Autorità Fidata)
- Privatezza del contenuto di D
- Quanto è fidata l'Autorità Fidata?

Introducendo le funzioni di hash riduciamo i problemi al solo problema 3. Infatti, invece che passare il contenuto di D all'autorità fidata, passo l'hash di D, che ha dimensione fissata e che preserva la privatezza del dato.

Esistono due tipologie differenti di protocolli:

- Protocolli distribuiti (senza Autorità Fidata): avere più "testimonianze" del tempo
- Protocolli con "link" (con Autorità Fidata): collegare tra loro le marche dei documenti

La Figura 162 mostra un esempio di protocollo distribuito.

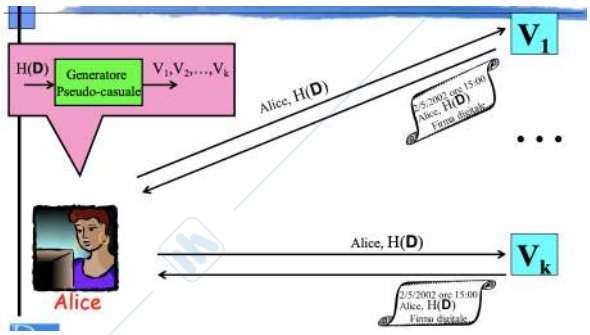


Figura 162: Protocollo distribuito per marche temporali

Per k grande è difficile per Alice corrompere k persone. La scelta delle persone da contattare è casuale, ma dipendente dal documento. Il problema che nasce dall'utilizzo di questo protocollo è il fatto che ci vogliono molte persone in grado di rispondere immediatamente ad Alice. Un altro problema è relativo alla durata (vita) delle firme digitali, quindi la firma potrebbe non essere

più valida al tempo della verifica della marca temporale: la chiave privata è stata compromessa, oppure lo schema di firme è stato rotto.

Nel protocollo con link l'autorità fidata (ma non troppo) è il Time Stamping Service, il quale riceve tutte le richieste in intervalli prefissati.

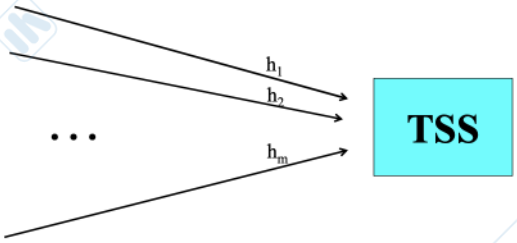


Figura 163: Protocollo con link per marche temporali

Collega tra loro le richieste, attraverso un albero di hash ed invia ad ognuno una marca temporale. Vincola se stesso a "non poter predatare".

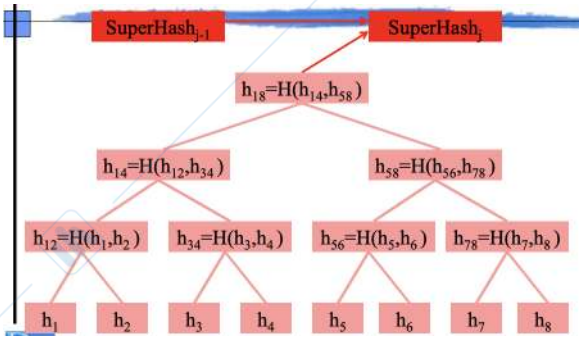


Figura 164: Protocollo con link per marche temporali

La marca temporale risultante da ogni richiesta è come lo schema mostrato in Figura 164.

ID utente della richiesta
h_i
data ed ora
h_{1m} (valore hash della radice dell'albero)
info necessarie per verificare che h_i è stato utilizzato per costruire l'albero con radice h_{1m}
$SuperHash_{j-1}$ e $SuperHash_j$
Firma del TSS

Figura 165: Strutture delle marche temporali

La Figura 166 mostra le informazione necessarie per ricostruire la radice h_{18} dell'abero per verificare h_3.

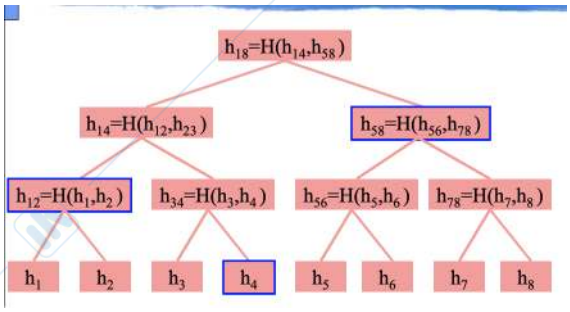


Figura 166: Albero di ricerca

Fissato il valore hash della radice, non è possibile né inserire un nuovo valore nell'albero di hash, né cambiare anche un solo valore nell'albero di hash, altrimenti si determinerebbe una collisione per la funzione hash.

Si potrebbe rompere lo schema colludendo solo con il TSS e creando una insieme di alberi collegati lunghi "a sufficienza". Una possibile soluzione è pubblicizzare SuperHash ad intervalli regolari (ogni giorno su Internet, su quotidiani, etc. e distribuzione mediante e-mail, CD, etc.).

10.6.1 Digital Notary

Un digital notary è un notaio in grado di compiere atti notarili elettronicamente, attraverso Timestamp affidabili, un metodo elettronico per creare prove permanenti dell'esistenza di un documento o di altre informazioni elettroniche in una determinata forma in un determinato momento. Il cliente usa un software venduto dalla Surety, che utilizza una funzione hash con un digest di 288 bit (MD5+SHA). Il sistema usa una struttura ad albero, in cui l'unità di tempo corrisponde ad un secondo, Un numero seriale viene inserito nel documento. Il SuperHash è pubblicato in posti accessibili via rete, su un CD-ROM, ed ogni settimana sul Sunday New-York Times.

10.6.2 PGP Digital Timestamping Service

Il TSS firma ogni documento che riceve. Ogni firma ha un numero seriale. Il TSS memorizza tutte le firme che genera. Tutte le marche (Serial Number, Date, Time) emesse possono essere esaminate. Ogni giorno pubblica il numero seriale dell'ultima firma effettuata e tutte le marche emesse nella giornata.

La Figura 167 mostra un esempio di riepilogo di PGP.

Figura 167: PGP Digital Timestamping Service

11 Message Authentication Code (MAC)

In crittografia un message authentication code (MAC) è un piccolo blocco di dati utilizzato per garantire l'autenticazione e integrità di un messaggio digitale, generato secondo un meccanismo di crittografia simmetrica: un algoritmo MAC accetta in ingresso una chiave segreta e un messaggio da autenticare di lunghezza arbitraria, e restituisce un MAC (alle volte chiamato anche tag). In ricezione il destinatario opererà in maniera identica sul messaggio pervenuto in chiaro ricalcolando il MAC con lo stesso algoritmo e la stessa chiave: se i due MAC coincidono si ha autenticazione e integrità del messaggio inviato.

Il valore MAC protegge dunque sia l'integrità dei dati del messaggio sia la sua autenticità permettendo al destinatario dello stesso (che deve anch'egli possedere la chiave segreta) di rilevare qualsiasi modifica al messaggio: ecco perché dovrebbe essere chiamato Message Authentication and Integrity Code, MAIC. Come si desume quindi dalla descrizione, l'algoritmo MAC protegge solo da integrità dei dati, autenticità del mittente del messaggio, ma non dalla confidenzialità delle informazioni contenute nello stesso.

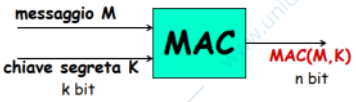


Figura 168: Message Authentication Code (MAC)

Ad esempio, consideriamo la comunicazione tra 2 utenti Annarella e Biagio, e supponiamo che Annarella debba inviare un messaggio a Biagio, utilizzando il MAC con chiave privata K, nota ad entrambi. Annarella invia a Biagio la coppia di valori (messaggio, impronta del messaggio). Biagio una volta ricevuta la coppia (messaggio, impronta del messaggio) verifica l'autenticità del messaggio in tale modo:

- calcola l'impronta del messaggio ricevuto
- confronta l'impronta calcolata con quella ricevuta:
  - se sono uguali: OK (il messaggio è autentico ed integro)
  - altrimenti: SCARTA (il messaggio non è autentico)

La Figura 169 mostra lo schema generale del funzionamento di Message Authentication Code:

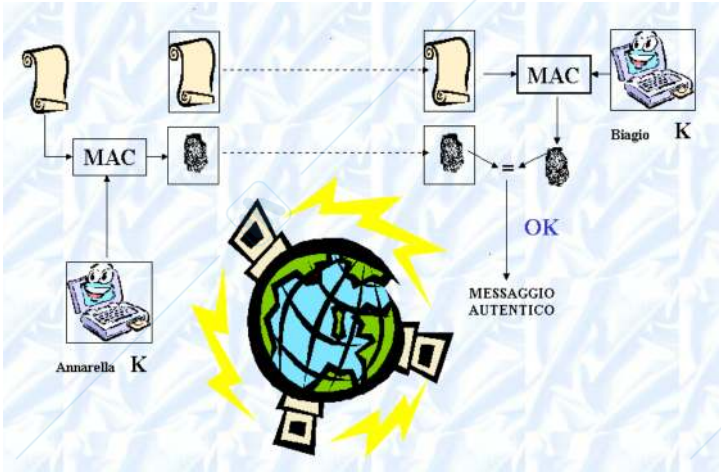


Figura 169: Message Authentication Code (MAC)

Gli algoritmi MAC sono una famiglia di funzioni g_K parametrizzate con una chiave segreta K aventi le seguenti proprietà:

- easy of computation: per una funzione conosciuta g_K, dato un valore K e un input $M, g_K(M)$ è facile da computare. Questo risultato è detto il valore-MAC oppure semplicemente MAC di M.
- compression: g_K è progettata per un input M di lunghezza finita ed arbitraria ed un output di lunghezza fissata.

Inoltre, data una descrizione della famiglia delle funzioni g, per ogni fissato valore di K (sconosciuto ad un avversario) deve possedere la proprietà di computation-resistance, per cui dato una o più coppie testo-MAC $(M_i, g_K(M_i))$ non è possibile calcolare qualsiasi altra coppia $(M, g_K(M))$ per ogni altro nuovo input $M \neq M_i$. Se la proprietà di computation-resistance non è rispettata, un algoritmo di MAC può essere soggetto ad attacchi. Mentre la proprietà di computation-resistance implica la proprietà di key-non-recovery (cioè è impossibile ricavare K, data una o più coppie testo-MAC $(M_i, g_K(M_i))$) la proprietà di key-non-recovery non implica la proprietà di computation-resistance (cioè non è sempre necessario conoscere la chiave K per ottenere una nuova coppia testo-MAC non conosciuta).

Come dicevamo, l'algoritmo MAC protegge solo da integrità dei dati, autenticità del mittente del messaggio, ma non dalla confidenzialità delle informazioni contenute nello stesso. Può però essere utilizzato insieme a un algoritmo di crittografia simmetrica che crittografa tutto il messaggio M con una chiave K', verrà effettuato quindi il MAC del messaggio M crittografato con K' usando la chiave K. Verrà poi crittografato con un algoritmo di crittografia simmetrica (usando la chiave K') il messaggio M e il suo MAC preparato con chiave K. Verranno quindi inviati al destinatario: $E_{K'}(M)$, $MAC(E_{K'}(M), K)$ e $E_{K'}(M, MAC(M, K))$. Le due chiavi K' (usata nella crittografia simmetrica) e K (usata nella creazione del MAC) non dovranno però essere uguali.

Gli attacchi ai sistemi di autenticazione e integrità MAC si differenziano, sulla base dello scopo dell'attacco:

- Total break, che ha come scopo quello di determinare la chiave K
- Selective forgery: attacco in cui un avversario è capace di produrre una nuova coppia testo-MAC per un testo scelto liberamente dall'avversario stesso. Dato un messaggio M, determinare y tale che $y = MAC(M, K)$.
- Existential forgery: attacco in cui un avversario è capace di produrre una nuova coppia testo-MAC solo che l'avversario non ha nessun controllo sul valore del testo scelto. Determinare una coppia (M, y) tale che $y = MAC(M, K)$.

Uno dei seguenti scenari di attacchi si può presentare per il MAC:

- known-text attack: una o più coppie testo-MAC $(M_i, MACK(M_i))$ sono disponibili. "Oscar conosce una lista di messaggi ed i relativi MAC"
- chosen-text attack: una o più coppie testo-MAC $(M_i, MACK(M_i))$ sono disponibili per M_i scelti da un avversario. "Oscar sceglie dei messaggi e chiede ad Annarella (o Biagio) di computarne i MAC."
- adaptive chosen-text attack: gli M_i possono essere scelti come nel chosen-text attack solo che da ora in poi le scelte sono basate sui risultati precedentemente consultati.

La chiave di un algoritmo MAC può essere determinata utilizzando una ricerca esaustiva. Con la conoscenza di una sola coppia testo-MAC, un avversario può calcolare il valore MAC del

testo con tutte le possibili chiavi, e controllare quale delle computazioni MAC-valore corrisponde con la coppia conosciuta. Per uno spazio di una chiave di t-bit sono richiesti al più 2^t calcoli di valori MAC. Per una chiave di t-bit e per un fissato input una scelta casuale di una chiave può produrre un corretto m-bit valore-MAC con probabilità di circa 2^{-t} per $t < m$. Operativamente, date le coppie (M_i, y_i), con $y_i = MAC(M_i, k)$, determiniamo K. Supponiamo che sia $k > n$. Prova tutte le 2^k chiavi sulla coppia (M_1, y_1): circa 2^{k-n} match. Prova le 2^{k-n} chiavi precedenti sulla coppia (M_2, y_2): circa 2^{k-2n} match, etc. In generale, se $k = \beta \cdot n$, necessari circa β round.

Figura 170: Message Authentication Code (MAC)

Dato M determiniamo $y = MAC(M, K)$, senza conoscere K. Scegliendo a caso y, abbiamo una probabilità di successo: $\frac{1}{2}^n$. Ma come controlliamo che sia il valore giusto, senza avere K?

Quindi, si può eseguire una ricerca esaustiva, provando tutti i k e verificando. Eseguire questo deve essere un compito computazionalmente difficile. Lo sforzo computazionale richiesto è $min(2^k, 2^n)$. La raccomandazione è quella di avere $min(k, n) \geq 128$.

Una pseudo random function PRF $F : K \times X \to Y$ può essere usata per definire un MAC: $MAC(k, m) := F(k, m)$. La Figura 171 mostra un esempio in tal senso.

Figura 171: PRF sicura → MAC sicuro

Esistono diverse costruzioni di MAC basate su:

- MAC basati su cifrari a blocchi (CBC-MAC)
- MAC basati su funzioni hash (Metodo del prefisso segreto, Metodo del suffisso segreto, HMAC)
- MAC from scratch (MAA, MD5-MAC)

11.1 CBC-MAC

Il più comune algoritmo di MAC su cifrario a blocchi fa uso di Cipher-Block-Chaining (CBC) ed è detto CBC-MAC. Il CBC-MAC è lo schema di autenticazione dei messaggi nel quale l'etichetta di un messaggio è il risultato dell'ultimo blocco del testo cifrato ottenuto cifrando il messaggio M in modalità CBC con vettore di inizializzazione IV=0.

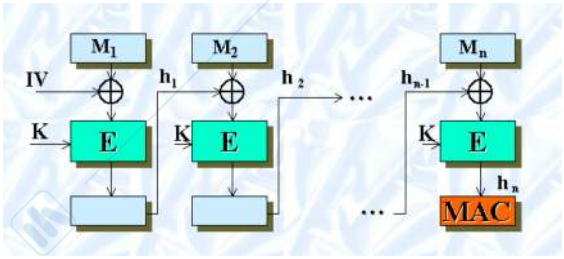


Figura 172: Funzionamento CBC-MAC

L'algoritmo CBC-MAC è il seguente:
INPUT: messaggio M; specificazione del cipher block E; chiave segreta K per E.
OUTPUT: valore MAC di M composto da L bit (dove L è la lunghezza dell'output prodotto dalla funzione hash utilizzata come cipher block).

1. Padding and blocking: Il messaggio M viene diviso in blocchi di L bit $M_1, M_2, ..., M_n$, se la lunghezza di M non è multiplo di L si fa padding con 10...0.

2. CBC processing: Sia $E_K(X)$ la codifica del valore X prodotta dal blocco E con chiave K, ogni valore h_i è calcolato come segue: $h_1 \leftarrow E : K(IV \oplus M_1), h_i \leftarrow E_K(h_{i-1} \oplus M_i), 2 \leq i \leq n$.

3. Optional process to increase strength of MAC: Usando una seconda chiave segreta $K' \neq K$ calcoliamo : $h'_n \leftarrow E^{-1}_{K'}(h_n)$; $h_n \leftarrow E_K(h'_n)$.(Questo equivale ad usare una tripla codifica con due chiavi).

4. Completion: Il valore MAC di M sono gli L bit di h_n.

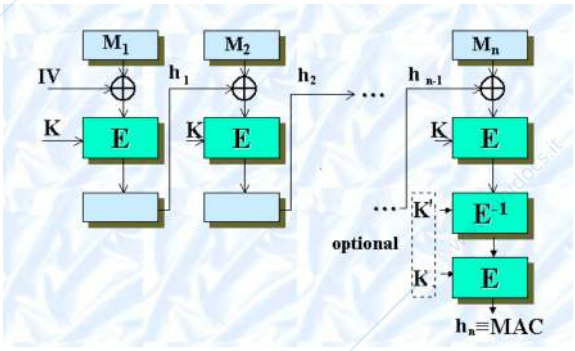


Figura 173: Funzionamento CBC-MAC

Il processo opzionale (3) riduce il pericolo di un attacco di ricerca esaustiva senza impatti sull'efficienza media dell'algoritmo. In genere come cipher block E si usa il DES, in tal caso il valore MAC è composto da 64 bit e la chiave segreta è di 56 bit. Uno degli svantaggi del DES sono le restrizioni riguardanti la sua esportazione dagli U.S.A., in particolare queste restrizioni riguardano la lunghezza della chiave che non puo' essere superiore a 56 bit. Il motivo per cui sono state imposte tali restrizioni non è stato reso noto. Ma è evidente che chi dispone di forti capitali come ad esempio una nazione può crittoanalizzare qualsiasi messaggio cifrato con il DES avente chiave di 56 bit; però bisogna tener presente che ultimamente il governo degli U.S.A. è intenzionato a togliere tali restrizioni e si pensa che lo faccia a breve.

La scelta dello spazio dei messaggi è importante per la sicurezza del CBC-MAC. Se sono permessi messaggi di varia lunghezza lo schema risulta insicuro, ma se la lunghezza dei messaggi è ristretta a qualche singolo valore fissato lo schema diventa più sicuro.

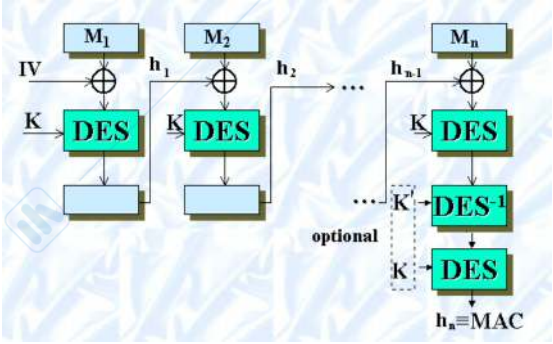


Figura 174: Funzionamento CBC-MAC con utilizzo algoritmo DES

11.2 MAC basati su funzioni Hash

I vantaggi che derivano dall'uso di funzioni hash per la costruzione di MAC sono:

- Sono più veloci dei cifrari a blocchi, in genere
- Sono incluse nelle funzioni di libreria, in genere
- Non ci sono restrizioni sull'esportazione dagli USA

Uno svantaggio delle funzioni hash è che non sono state originariamente progettate per il MAC e ciò comporta non poche difficoltà nel realizzare il MAC con esse. Comunque esistono algoritmi basati su funzioni hash progettati appositamente per il MAC come MAA ed MD5-MAC che vedremo di seguito.

Le proposte di costruzione di MAC basate sulle funzioni hash che considereremo sono:

- Metodo del prefisso segreto
- Metodo del suffisso segreto
- Metodo dell'inviluppo con padding
- HMAC

Successivamente indicheremo l'operazione di concatenazione con il simbolo $||$.

Sia H una funzione hash iterata con funzione di compressione f, dove l'input è M, dopo il padding n blocchi $X_1X_2 \ldots X_n$. La funzione di compressione è data da H_0 una costante iniziale, e dalla computazione di $H_i = f(X_i, H_{i-1})$. Il valore hash $H_n = f(X_n, H_{n-1})$. La Figura 175 mostra uno schema di questo procedimento.

Figura 175: MAC basati su funzioni hash

11.2.1 Metodo del segreto prefisso

Consideriamo un messaggio M suddiviso in blocchi di taglia uguale $M_1, M_2, ..., M_n$, ed una funzione hash iterata secondo il modello generale di funzione hash, con funzione di compressione f, con definizione: $h_0 = IV$, $h_i = f(h_{i-1}, M_i)$ per $1 \leq i \leq n, H(M) = h_n$. Supponiamo di voler usare H come un algoritmo MAC pre-aggiungendo una chiave segreta K al messaggio

M, cosicché il MAC proposto di M sarà $MACK(M) = H(K||M)$. Un nemico estendendo il messaggio M di un blocco x, potrebbe ricavare $MACK(M||x)$ come $f(H(K||M), x)$ (infatti $f(H(K||M), x) = H(K||Mx)$) senza conoscere la chiave segreta K (il MAC originale è utilizzato come variabile di concatenamento)(padding attack).

Questo è vero anche per le funzioni come MD4 in cui il pre-processo di padding aggiunge un ultimo blocco z di 512-bit contenente 448 bit di zero seguiti da 64 bit contenenti la lunghezza del messaggio originario M. Questa misura aggiuntiva non allevia il padding attack in quanto l'attaccante può includere questo blocco aggiuntivo del messaggio originario come parte del messaggio da falsificare (cioè del messaggio di cui si vuole dedurre il MAC senza conoscere la chiave K). Questo attacco è possibile in quanto l'attaccante conosce il funzionamento di MD4, il valore del messaggio M e il MAC di M. Una ulteriore contromisura per questo tipo di attacco potrebbe essere di includere la lunghezza del messaggio M come parte del primo blocco.

Per simili ragioni, è insicuro utilizzare una funzione hash per costruire un algoritmo MAC usando la chiave K come IV. Se K comprende l'intero primo blocco, allora $f(IV, K)$ può essere pre-calcolata, dimostrando che un avversario ha solo bisogno di trovare un K' (non necessariamente K) tale che $f(IV, K) = f(IV, K')$ e questo equivale ad utilizzare un segreto IV.

11.2.2 Metodo del segreto suffisso

Una proposta alternativa al metodo del prefisso segreto è il metodo del suffisso segreto. Il metodo del suffisso segreto utilizza come il metodo del prefisso segreto una funzione hash iterata H con funzione di compressione definita come sopra, con la differenza che la chiave segreta K viene utilizzata come suffisso al messaggio M. Quindi l'input della funzione hash H sarà il valore $M||K$ ottenuto post-aggiungendo al messaggio M la chiave segreta K, ed il MAC proposto di M sarà $H(M||K)$ (costituito da j bit).

Se H utilizza la chiave segreta K solo nell'ultimo passo può essere applicato a tale metodo l'attacco del compleanno in quanto se $H(M) = H(M')$ abbiamo che $H(M||K) = H(M'||K)$ per $M \neq M'$.

Infatti, un avversario scegliendo liberamente il messaggio M, in $O(2^{|hash(\cdot)|/2})$ operazioni riesce a trovare una coppia M, M' (dove $M \neq M'$) tale che $H(M) = H(M')$ ($|hash(\cdot)|$ è la lunghezza dell'output della funzione hash utilizzata). Questo può essere fatto off-line, e non è richiesta la conoscenza della chiave K. Quindi, un avversario ottenendo un giusto MAC su M può produrre una corretta coppia testo-MAC per un nuovo messaggio $M' \neq M$.

In questa debole forma di MAC, il valore-MAC dipende solo dall'ultima iterazione della funzione hash.

11.2.3 HMAC

HMAC è usata in combinazione con una funzione hash ed usa una chiave segreta K per il calcolo e la verifica del MAC. La funzione hash è utilizzata in modo black-box così che possa essere implementata con il codice disponibile ed un eventuale suo rimpiazzamento a causa della sicurezza o delle prestazioni può essere effettuato agevolmente. Gli obiettivi che sono dietro la costruzione di HAMC sono:

- usare una funzione hash disponibile senza modificarla, in particolare una funzione hash che operi bene nel software e il cui codice è distribuito gratuitamente;
- conservare le originali prestazioni della funzione hash;
- usare la chiave in modo semplice;
- avere una buona e comprensibile analisi crittografica della forza del meccanismo di autenticazione basata su ragionevoli assunzioni che sono alla base della funzione hash utilizzata;

- permettere un facile rimpiazzamento della funzione hash utilizzata nell'eventualità che una funzione hash piu' veloce e piu' sicura sia disponibile.

Sia M il messaggio al quale la funzione MAC deve essere applicata e H la funzione hash utilizzata da HMAC. Assumiamo che H è una funzione hash iterata che usa una funzione di compressione che opera su blocchi di B byte (B=64 per la maggior parte delle funzioni hash). Sia K la chiave segreta e condivisa dell'autenticazione del messaggio delle due parti, se la chiave ha lunghezza maggiore di B byte si calcola il valore hash di K cioè $h(K)$ per ottenere una nuova chiave $K = H(K)$ di L byte (dove L è la lunghezza dell'output della funzione hash utilizzata). Se, invece, K ha lunghezza minore di B byte sono aggiunti degli zeri per rendere la lunghezza di esattamente B byte (padding). Fissiamo inoltre due differenti stringhe di B byte "ipad" e "opad" come segue:

- ipad= il byte 0x36 ripetuto $B/8$ volte
- opad= il byte 0x5c ripetuto $B/8$ volte

La funzione HMAC prende come input la chiave K ed il messaggio M, e produce come output il valore MAC di M con chiave K calcolato come segue: $H(K \oplus opad||H((K \oplus ipad||M))$, dove $x||y$ equivale a: y è accodato ad x. I passi svolti dalla funzione HMAC supponendo che $B = 64$ byte sono i seguenti:

1. se $|K| > 64$ byte allora calcoliamo il valore hash di K per ottenere una nuova chiave $K = H(K)$ di L byte.
2. Se $|K| < 64$ byte padding con 0..00 alla fine di K per creare una stringa di 64 byte.
3. XOR della stringa di 64 byte calcolata nel passo (2) con ipad.

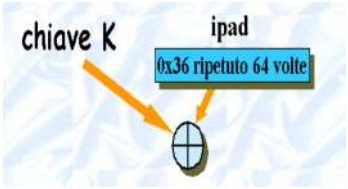


Figura 176: XOR della stringa da 64 byte e ipad

4. Il messaggio M è accodato alla stringa generata al passo (3)
5. H è applicata alla stringa generata al passo (4)

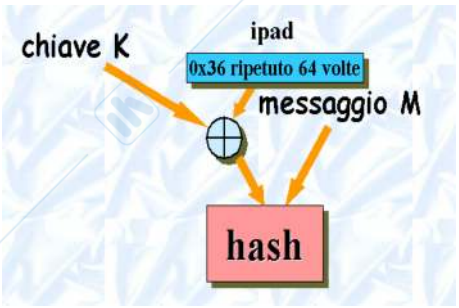


Figura 177: Funzione hash

6. XOR della stringa di 64 byte calcolata al passo (2) con opad

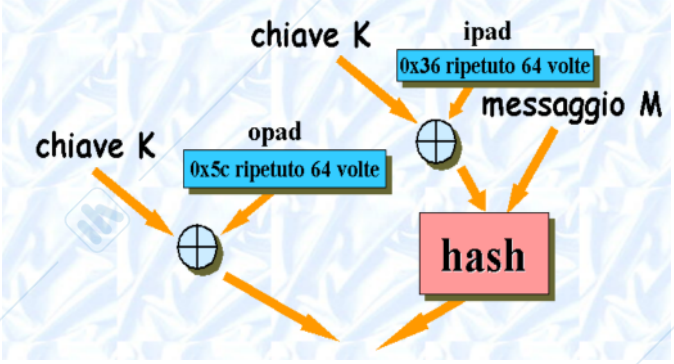


Figura 178: XOR con opad

7. Il risultato hash di H del passo (5) è accodato alla stringa di 64 byte risultante dal passo (6)

8. H è applicata alla stringa generata al passo (7), il risultato è restituito in output come MAC di M

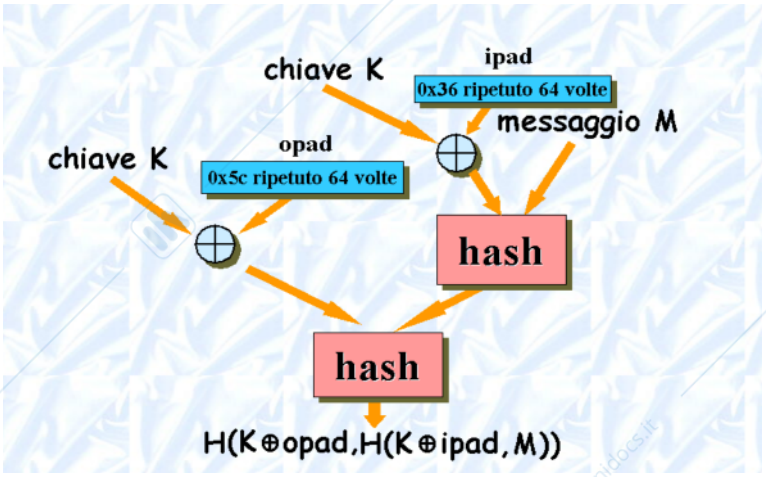


Figura 179: MAC di M

Il costo totale dell'autenticazione del messaggio M è prossimo a quello del calcolo della funzione hash del messaggio specialmente se M è grande. HMAC è stato recentemente scelto come obbligatorio per implementare l'autenticazione per i protocolli di sicurezza internet che sono stati progettati per IPSEC dal gruppo di lavoro dell'IETF. Per questo scopo HMAC è descritto in una RFC. Altri protocolli internet stanno adottando HMAC come per esempio s-http, SSL.

La chiave utilizzata per HMAC può avere qualsiasi lunghezza, anche se è sconsigliato utilizzare una chiave con meno di L byte (dove L è la lunghezza dell'output della funzione hash utilizzata) in quanto diminuirebbe la sicurezza della funzione hash. Mentre una chiave di lunghezza maggiore di L byte non incrementa in modo significativo la forza della funzione. La chiave deve essere scelta casualmente, cambiata periodicamente e deve essere tenuta segreta da entrambi gli utenti che la condividono.

11.3 Output troncato

Diversi algoritmi MAC troncano il loro output, ossia utilizzano come MAC solo i primi t byte del loro output. Quando è usato, un HMAC troncato solo i primi t byte del risultato prodotto da

HMAC sono utilizzati come MAC. La lunghezza di t non dovrebbe essere minore di $L/2$, in ogni caso t non deve essere meno di 80 bit. Una notazione proposta è la seguente: HMAC-H-t che indica la realizzazione di HMAC utilizzando la funzione hash H e l'output è di t-bit; ad esempio HMAC-SHA1-96 indica un HMAC realizzato utilizzando SHA-1 con l'output troncato a 96 bit. Il vantaggio dovuto al troncamento è dato da una minore informazione che si da' all'attaccante, lo svantaggio invece consiste in un minor numero di byte da predire all'attaccante.

11.3.1 Sicurezza HMAC

Come già detto le funzioni hash non sono originariamente progettate per l'autenticazione dei messaggi. In particolare, esse non sono delle primitive con chiave, e non è chiaro quale sia la chiave migliore per loro. In questo modo è necessario essere particolarmente attenti nell'adottare una funzione hash per il MAC. La massima che guida la costruzione di HMAC è stata: l'assenza di attacchi oggi non implica la sicurezza per il futuro. Le assunzioni sulla sicurezza delle funzioni hash non dovrebbero essere troppo forti, poiché non è stata ancora raggiunta un sufficiente collaudo con le candidate (come MD5 e SHA). Ciò che si è mostrato è che se la funzione hash ha proprietà di sicurezza allora HMAC è sicuro. Quindi l'unico modo in cui HMAC potrebbe fallire è se la funzione hash utilizzata da esso fallisce. Le caratteristiche di sicurezza possono essere così riassunte: se HMAC fallisce ad essere un MAC sicuro, significa che ci sono sufficienti punti deboli nella funzione hash utilizzata che ha bisogno di essere esclusa non solo da questo particolare uso ma da un ampio raggio di altri usi popolari ai quali essa è attualmente soggetta. Quindi HMAC ha prestazioni che sono essenzialmente quelle della funzione hash utilizzata.

Quindi, la sicurezza dipende dalle proprietà della funzione hash usata da HMAC. Se l'attaccante ha successo in un attacco ad HMAC allora:

- Può computare l'output della funzione di compressione anche quando IV è casuale e sconosciuto all'attaccante
- Può computare collisioni nella funzione hash anche quando IV è casuale e sconosciuto all'attaccante

Un attaccante vuole trovare due messaggi m e m' tali per cui $HMAC_k(m,h) = HMAC_k(m',h)$; conosce IV, h ma non conosce k Si applica il paradosso del compleanno (blocco lungo n implica collisione con $2^{n/2+1}$ tentativi)? No. Infatti, per attaccare HMAC, sono necessarie molte coppie M, HMAC, ma l'avversario non può calcolarle perché non conosce K. Deve osservare un flusso di messaggi generati con la stessa chiave. Possiamo dire che HMAC con hash sicuro è ragionevolmente sicura.

Le funzioni HASH e MAC insieme all'uso di cifrari a/simmetrici garantiscono diverse proprietà durante lo scambio di messaggi: Segretezza e Autenticazione (del testo in chiaro e del testo cifrato). La cifratura autenticata è un tipo di cifratura atto a fornire simultaneamente proprietà di confidenzialità, autenticità e integrità ad un certo dato trasmesso. Un cifrario (E, D) fornisce authenticated encryption (AE) se:

- É semantically secure nello scenario CPA[7]
- Ha ciphertext integrity, cioè l'attaccante non deve poter creare dei nuovi ciphertexts che si decifrano correttamente

Esistono diversi approcci alla crittografia autenticata fra i vari protocolli di sicurezza diffusi attualmente. Particolare rilevanza è rivestita dalle tre seguenti combinazioni di algoritmo di criptazione e MAC:

[7]Sicurezza rispetto a Chosen Plaintext Attack (CPA)

- Encrypt-then-MAC (EtM): Secondo l'approccio EtM, viene dapprima eseguita la criptazione del messaggio e poi generato un codice MAC a partire dal testo cifrato. $m' = E_k(m) \| MAC(E_k(m))$. Il testo cifrato e il suo MAC vengono inviati assieme. Viene usato, ad esempio, nel protocollo IPsec. Questo metodo fa parte dello standard ISO/IEC 19772:2009. É inoltre l'unico metodo che può raggiungere la più alta definizione di sicurezza in quanto a cifratura autenticata, ma ciò è possibile solo se il MAC usato è fortemente non falsificabile.

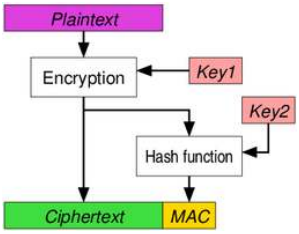


Figura 180: Approccio Encrypt-then-MAC (EtM)

- Encrypt-and-MAC (E&M): Secondo l'approccio E&M, il messaggio in chiaro viene cifrato e autenticato separatamente. $m' = E_k(m) \| MAC(m)$. Il testo cifrato e il codice MAC vengono successivamente inviati assieme. Viene usato, ad esempio, nel protocollo SSH. Non è mai stata dimostrata una forte non-falsificabilità di questo metodo in sé.

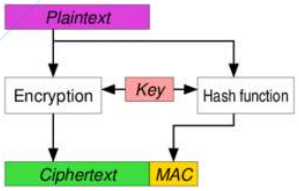


Figura 181: Approccio Encrypt-and-MAC (E&M)

- MAC-then-Encrypt (MtE): Secondo l'approccio MtE, viene dapprima generato il codice MAC del messaggio in chiaro, e successivamente viene eseguita la cifratura di tutto quanto. $m' = E_k(m \| MAC(m))$. Viene spedito un messaggio cifrato contenente il messaggio originale e il suo codice MAC. Viene usato, ad esempio, nel protocollo SSL/TLS. Nonostante non sia stata dimostrata una forte non-falsificabilità di questo metodo in sé, l'implementazione SSL/TLS è ritenuta sicura. L'approccio di quest'ultima può infatti essere descritto come MAC-then-pad-then-encrypt, ovvero è previsto un padding del testo prima della cifratura.

L'Authenticated encryption fornisce sicurezza CPA e integrità del testo cifrato. Inoltre, fornisce confidentialità in presenza di avversari attivi e previene chosen-ciphertext attacks.

12 Firme digitali

La firma digitale è un metodo matematico teso a dimostrare l'autenticità di un messaggio o di un documento digitale inviato tra mittente e destinatario attraverso un canale di comunicazione non sicuro, garantendo al destinatario le proprietà di autenticazione, non ripudio e integrità. Le firme digitali si basano su schemi o protocolli crittografici comunemente usati nella distribuzioni di software, nelle transazioni finanziarie e in altri casi in cui si debba rilevare la falsificazione o l'alterazione del messaggio.

In particolare le proprietà che deve avere una firma digitale per essere ritenuta valida sono:

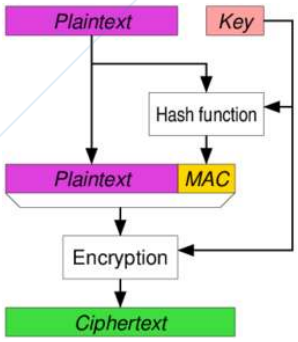


Figura 182: Approccio MAC-then-Encrypt (MtE)

- Autenticità della firma: la firma deve assicurare il destinatario che il mittente ha deliberatamente sottoscritto il contenuto del documento.
- Non falsificabilità: la firma è la prova che solo il firmatario e nessun altro ha apposto la firma sul documento.
- Non riusabilità: la firma fa parte integrante del documento e non deve essere utilizzabile su un altro documento.
- Non alterabilità: una volta firmato, il documento non deve poter essere alterato.
- Non contestabilità: il firmatario non può rinnegare la paternità dei documenti firmati; la firma attesta la volontà del firmatario di sottoscrivere quanto contenuto nel documento.

I requisiti per la Firma Digitale sono:

- La firma digitale deve poter essere facilmente prodotta dal legittimo firmatario
- Nessun utente deve poter riprodurre la firma di altri
- Chiunque può facilmente verificare una firma

I metodi crittografici a chiave pubblica possono essere utilizzati per la costruzione di strumenti per la firma digitale, variamente concepiti. Mentre nella crittografia la chiave pubblica viene usata per la cifratura, ed il destinatario usa quella privata per leggere in chiaro il messaggio, nel sistema della firma digitale il mittente utilizza la funzione di cifratura e la sua chiave privata per generare un'informazione che (associata al messaggio) ne verifica la provenienza, grazie alla segretezza della chiave privata. Chiunque può accertare la provenienza del messaggio utilizzando la chiave pubblica.

Un tipico schema di firma elettronica basata sulla crittografia a chiave pubblica si compone dei seguenti algoritmi:

1. un algoritmo per la generazione della chiave, che seleziona in modo casuale una chiave privata da un insieme di possibili valori, e restituisce una coppia di chiavi, la chiave privata con cui si firma il documento e la corrispondente chiave pubblica di verifica della firma;
2. un algoritmo di firma che, presi in input un messaggio e la chiave privata, calcola il codice hash del messaggio e lo crittografa con la chiave privata, producendo una firma;
3. un algoritmo di verifica che, presi in input un messaggio, la chiave pubblica e la firma, accetta o rifiuta la firma che compare nel messaggio.

Nella prima fase viene applicata al documento in chiaro una funzione di hash appositamente studiata che produce una stringa binaria di lunghezza costante e piccola, normalmente 128 o 160 bit, chiamata digest message, ossia impronta digitale. Queste funzioni devono avere due proprietà:

- unidirezionalità, ossia dato x è facile calcolare $f(x)$, ma data $f(x)$ è computazionalmente difficile risalire a x.
- prive di collisioni (collision-free), ossia a due testi diversi deve essere computazionalmente impossibile che corrisponda la medesima impronta.

Poiché la dimensione del digest message è fissa, e molto più piccola di quella del messaggio originale; la generazione della firma risulta estremamente rapida.

L'utilità dell'uso delle funzioni hash consente di evitare che per la generazione della firma sia necessario applicare l'algoritmo di cifratura, che è intrinsecamente inefficiente, all'intero testo che può essere molto lungo. Mediante un software adatto, nel nostro caso quello di Entrsut, al sistema crittografico adottato, si genera una coppia di chiavi da utilizzare: una, che verrà mantenuta segreta, per l'apposizione della firma; l'altra, destinata alla verifica, che verrà resa pubblica.

Quindi la seconda fase, la generazione della firma, consiste semplicemente nella cifratura con la propria chiave privata dell'impronta digitale generata in precedenza. In questo modo la firma risulta legata, da un lato (attraverso la chiave privata usata per la generazione) al soggetto sottoscrittore, e dall'altro (per il tramite dell'impronta) al testo sottoscritto. In realtà l'operazione di cifratura viene effettuata, anziché sulla sola impronta, su una struttura di dati che la contiene insieme con altre informazioni utili, quali ad esempio l'indicazione della funzione hash usata per la sua generazione. Sebbene tali informazioni possano essere fornite separatamente rispetto alla firma, la loro inclusione nell'operazione di codifica ne garantisce l'autenticità.

Nell'ultima fase, la firma digitale generata precedentemente viene aggiunta in una posizione predefinita, normalmente alla fine del testo del documento.

Chi riceve il messaggio procederà all'apertura e/o verifica dello stesso mediante il proprio software per l'attività di firma. Il programma acquisirà dal certificato annesso al documento firmato, la chiave pubblica del mittente. Con tale chiave viene decifrata la stringa della "firma digitale" che darà come risultato l'impronta del documento. Il destinatario prenderà il documento originario, lo farà passare attraverso la funzione di "hash" e genererà l'impronta: se coincide con quella decrittata del mittente allora sarà sicuro dell' integrità e provenienza del documento.

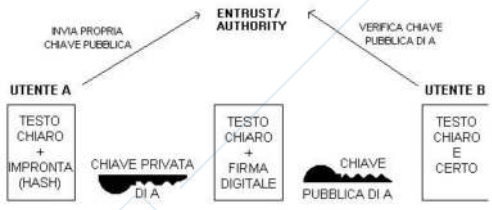


Figura 183: Firme digitali

Sono richieste le seguenti due proprietà: primo, l'autenticità di una firma generata da un messaggio fisso e da una chiave privata deve essere verificata facendo uso della corrispondente chiave pubblica. In secondo luogo, dovrebbe essere computazionalmente impossibile generare una firma valida per un messaggio senza avere a disposizione la chiave privata.

Nel loro fondamentale lavoro, Goldwasser, Micali, e Rivest illustrarono una gerarchia di modelli di attacco alla firma digitale:

- in un attacco key-only, l'attaccante possiede solo la chiave pubblica di verifica. "Oscar conosce solo kpub di Alice"
- in un attacco known message, l'attaccante possiede diverse firme valide per un ampio numero di messaggi che egli conosce, ma non scelti da lui. "Oscar conosce una lista di messaggi e le relative firme di Alice"
- Chosen Message Attack "Oscar sceglie dei messaggi e chiede ad Alice di firmarli"
- in un attacco adaptive chosen message, l'attaccante per prima cosa studia le firme presenti su messaggi casuali a sua scelta. "Oscar sceglie interattivamente i messaggi da firmare"

Gli scienziati formularono anche una gerarchia basata sui risultati dei possibili attacchi:

- un attacco total break consente di impossessarsi della chiave di firma. Determinare kpriv di Alice per poter firmare qualsiasi messaggio
- un attacco universal forgery conduce alla falsificazione della firma per ogni messaggio.
- un attacco selective forgery ha la conseguenza di poter apporre la firma su un messaggio a scelta dell'avversario. Dato un messaggio M, determinare la firma F tale che VERIFICA $(F, M,\text{kpub}) = \text{SI}$.
- un attacco existential forgery porta semplicemente a comporre una valida accoppiata messaggio/firma non ancora nota all'avversario. Determinare una coppia (M, F) tale che VERIFICA $(F, M,\text{kpub}) = \text{SI}$.

12.1 Firma RSA

Alice, per firmare un documento m con la firma RSA, deve compiere i seguenti passi:

1. Alice genera p e q, primi grandi e calcola $n = pq$.
2. Calcola $ed \equiv 1 \pmod{(p-1)(q-1)}$.
3. Alice pubblica (e, n) e tiene d, p, q privati.
4. La firma di Alice è : $F \equiv M^d \pmod{n}$
5. La coppia (m, F) è resa pubblica.

Bob, ricevendo il messaggio m, verifica l'autenticità della firma in questo modo:

1. Ottiene i parametri (e, n) di Alice.
2. Calcola $z \equiv F^e \pmod{n}$
3. Se $z = m$, la firma è valida, altrimenti la firma non è autentica.

Poichè la firma F è propria del documento m, un antagonista non può incollare F su un altro messaggio m_1 e usare la coppia (m_1, F), in quanto F $F^e \equiv m_1 \pmod{n}$. Eva necessita di un F_1 tale che $F_1^e \equiv m_1 \pmod{n}$, ma questo è equivalente alla decifrazione del messaggio m_1, cifrato con l'algoritmo RSA, in modo da ottenere il testo in chiaro F_1, problema che si ritiene difficile. Eva, allora, potrebbe scegliere prima F_1 e, in seguito, il messaggio $m_1 \equiv F_1^e \pmod{n}$; in questo modo la firma risulterebbe originale, tuttavia, poiché Eva non può controllarne il contenuto, è assai improbabile che m_1 risulti un messaggio di senso compiuto. In ogni modo, per escludere quest'ultimo caso, e per una maggior efficienza nell'uso della firma digitale, si introducono le funzioni hash crittografiche.

Le funzioni hash sono particolarmente utili nelle firme digitali sia per evitare la situazione descritta precedentemente, sia perché spesso capita che il documento sia molto lungo. Poichè una firma digitale F è spesso lunga quanto il documento che deve essere firmato m, risulta più conveniente firmare $h(m)$, con h funzione hash crittografica. Infatti, è più efficiente firmare una stringa di lunghezza fissata pari a n, piuttosto che un lungo documento. Il valore hash $h(M)$ è una rappresentazione non ambigua e non falsificabile del messaggio M. Le proprietà di cui gode sono:

- comprime
- facile da computare
- Sicurezza forte: computazionalmente difficile trovare 2 diversi messaggi con lo stesso valore hash
- One-way: dato y è computazionalmente difficile trovare M tale che $y = h(M)$

La Figura 184 mostra il funzionamento delle firme digitali con l'utilizzo delle funzioni di hash.

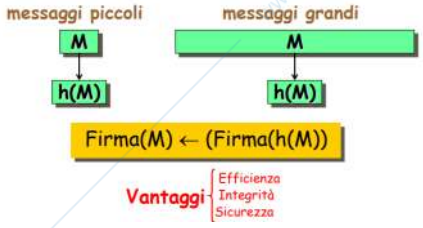


Figura 184: Firme digitali con hash

Inoltre, firmando $h(m)$, si evita la possibilità che Eva trovi un testo m_1 corrispondente alla firma F_1; infatti, in questo caso, m_1 dovrebbe soddisfare: $F_1 = (h(m_1))^d \pmod n$. Eva dovrebbe, dato F_1, trovare un messaggio m_1 tale che $h(m_1) = F_1^e \pmod n$, ovvero trovare la controimmagine di $h(m_1)$, ma questo è impossibile poichè h è unidirezionale.

Chiunque può verificare l'autenticità di un documento: per farlo, decifra la firma del documento con la chiave pubblica del mittente, ottenendo l'impronta digitale del documento, e quindi confronta quest'ultima con quella che si ottiene applicando la funzione hash al documento ricevuto; se le due impronte sono uguali, l'autenticità e l'integrità del documento sono garantite.

La firma che Alice apporrà sul messaggio è $F \leftarrow [h(M)]^d \pmod n$. La verifica che Bob attua è la seguente: verifica la firma di M, risulta vera se $h(M) = F^e \pmod n$, falsa altrimenti.

Supponiamo che un attaccante voglia generare messaggi e firme da parte di A:

- Sceglie F a caso
- $z \leftarrow F^e \pmod n$
- $M \leftarrow h^{-1}(z)$

Come faccio ad invertire h? Scelgo M_1 e M_2 tali che $h(M_1) = h(M_2)$. Ottengo F_1 come firma di M_1. (M_2, F_1) è una firma valida.

12.1.1 RSA-PSS

Probabilistic Signature Scheme (PSS) è uno schema di firma crittografica progettato da Mihir Bellare e Phillip Rogaway. RSA-PSS è un adattamento del loro lavoro ed è standardizzato come parte di PKCS # 1 v2.1.

Si tratta di un modello RSA, con modulo N, e, d. Se N ha k bit (es. 1024,) si selezionano k_0 e k_1, tale che $k_0 + k_1 < k - 1$ (ad esempio, $k_i = 128$). Si selezionano due funzioni hash:

- $G : \{0,1\}^{k_1} \to \{0,1\}^{k-1-k_1}$
- $H : \{0,1\}^* \to \{0,1\}^{k_1}$

$Tk(w)$: resituisce gli ultimi k bit di w.
La firma di un messaggio m consiste nei seguenti passaggi:

- Si sceglie un valore casuale r di k_0 bit
- Sia $w = H(m||r)$
- Sia $y = 0||w||(T_{k_0}(w) \oplus r)||T_{k-1-k_0-k_1}(w)$

La firma è $s = y^d \pmod N$.
La verifica della firma s su un messaggio m consiste nei seguenti passaggi:

- Si calcoli $y = s^e \pmod N$
- Sia $y = a||b||c||d$, dove:
  - a lunga 1 bit
  - b k_1 bit
  - c k_0 bit
  - d $k - 1 - k_1 - k_0$ bit
- Sia $r = c \oplus T_{k_0}(w)$

La firma è verificata se $a = 0$, $T_{k-1-k_0-k_1}(w) = d$ e $H(m||r) = w$.

12.2 Altri schemi e DSS

Altri esempi di schemi di firma digitale sono:

- El-Gamal (una versione modificata è DSS)
- Schnorr
- Fiat-Shamir
- GQ (Guillou-Quisquater)
- ECDSA (DSA su curve ellittiche)
- One-time signatures: schemi validi per un solo messaggio (altrimenti la firma è compromessa)
  - Rabin
  - Lamport-Merkle
  - Matyas_Meyer (basato su DES)
  - GMR (Goldwasser,MIcali,Rivest)
- Blind signatures: Il firmatario non vede il messaggio che firma
- Arbitrated signatures: Richiedono una terza parte fidata (TTP) per la generazione e la verifica della firma

- Undeniable signatures: La verifica viene fatta in collaborazione con il firmatario
- Fail stop signatures: Una firma falsa viene sempre scoperta (il firmatario può sempre dimostrarlo)
- Chameleon signatures: Solo il destinatario della firma è in grado di provare la validità

In crittografia, le Blind Signatures (Firme Cieche) sono state introdotte da David Chaum come una forma di firma digitale in cui il contenuto di un messaggio viene nascosto prima di essere firmato. Il messaggio viene quindi firmato in modo cieco, infatti il firmatario non conosce il contenuto del messaggio. Le blind signatures sono utilizzate nei protocolli per la privacy dove il firmatario e l'autore del messaggio sono differenti, ad esempio sono molto utilizzate nel campo dell'e-voting (voto elettronico) e dell'e-cash (denaro elettronico). Gli schemi di blind signatures possono essere implementati usando i comuni schemi di firma a chiave pubblica come RSA o DSA. La Figura 185 rappresenta l'implementazione delle blind signatures con lo schema di crittografia RSA.

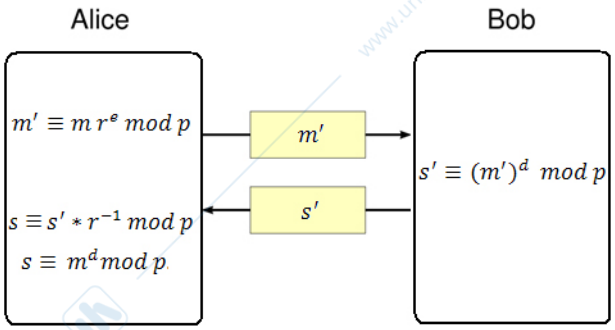


Figura 185: Blind signature con RSA

Si considera inizialmente la chiave pubblica di Bob (p, e), dove p è il prodotto di due numeri primi ed e è l'esponente pubblico della chiave. Alice sceglie un numero casuale r (detto anche blinding factor) coprimo con p e calcola $m^{'} \equiv mr^e \mathrm{mod} p$ inviandolo attraverso il canale pubblico a Bob. Bob calcola $s^{'} \equiv (m^{'})^d \mathrm{mod} p$ usando la sua chiave privata (p, d) e invia $s^{'}$ ad Alice. A questo punto Alice può togliere la propria firma e ottenere il messaggio originale m firmato da Bob nel seguente modo $s \equiv s^{'} * r^{-1} \mathrm{mod} p \equiv m^d \mathrm{mod} p$.

12.2.1 DSS

Digital Signature Algorithm (DSA) è uno standard FIPS per la firma digitale proposto dal National Institute of Standards and Technology (NIST) nell'agosto del 1991 per essere impiegato nel Digital Signature Standard (DSS), le sue specifiche sono contenute nel documento FIPS 186, viene definitivamente adottato nel 1993. In seguito è stato riveduto ulteriormente nel 1996 con FIPS 186-1, nel 2000 con FIPS 186-2, nel 2009 con FIPS 186-3, e nel 2013 con FIPS 186-4. Negli Stati Uniti l'algoritmo DSA è coperto da brevetto attribuito a David W. Kravitz, un ex-ricercatore della NSA; il NIST ha reso questo brevetto disponibile liberamente per qualsiasi uso.

DSS fa uso di un sistema crittografico a chiave pubblica simile ad ElGamal. Lo schema di firma ElGamal è un crittosistema di firma digitale basato sulla presunta difficoltà computazionale del calcolo di logaritmi discreti. É stato descritto da Taher Elgamal nel 1984. L'originale algoritmo di firma di Elgamal è raramente usato nella pratica, in favore di una variante sviluppata dalla NSA nota come Digital Signature Algorithm (DSA). Esistono molte altre varianti. Lo

schema di firma ElGamal non dev'essere confuso con l'omonimo sistema di cifratura a chiave pubblica, anch'esso proposto da Taher Elgamal.

Utilizza la funzione hash SHA (message digest di 160 bit) e le firme DSS sono sempre di 320 bit (buone per smart card).

Come molti sistemi di crittografia a chiave pubblica, la sicurezza di DSA si basa sull'intrattabilità di un problema matematico, in questo caso sull'inesistenza di un algoritmo efficiente per il calcolo del logaritmo discreto.

Di seguito vengono descritti i passi fondamentali per l'impiego di DSA:

- Si scelga un numero primo di d-bit q (160 bit).
- Si scelga un numero primo p lungo L bit, tale che $p = qz + 1$ per un qualche numero intero p, con $512 \leq L \leq 1024$ e divisibile per 64 (nell'ultima revisione dello standard si specifica che L deve corrispondere a 1024).
- Si scelga h tale che $1 < h < p - 1$ e $\alpha = h^z \mathrm{mod} p > 1$
- Si generi un numero casuale s tale che $0 < s < q$
- Calcolare $\beta = \alpha^s \mathrm{mod} p$

La chiave pubblica è β, la chiave privata è s. I parametri (p, q, α) sono pubblici e possono essere condivisi da diversi utenti.

Esistono algoritmi efficienti per il calcolo delle esponenziazioni modulari $h^z \mathrm{mod} p$ e $\alpha^s \mathrm{mod} p$. Per la generazione di α dobbiamo introdurre il concetto di ordine di un elemento. L'ordine di $\alpha \in Z_n^* =$ il più piccolo intero positivo r tale che $\alpha^r = 1 \pmod n$. Secondo il Teorema di Lagrange: per ogni $\alpha \in Z_n^*$, $ord(\alpha) divide \phi(n)$. Se n è primo, $ord(\alpha)$ divide $n - 1$. Ad esempio, $n = 15, \phi(n) = (3 - 1)(5 - 1) = 8$. $Z_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$:

- $ord(1) = 1, ord(2) = 4$,
- $ord(4) = 2, ord(7) = 4$,
- $ord(8) = 4, ord(11) = 2$,
- $ord(13) = 4, ord(14) = 2$.

Quindi, dato un gruppo G e un suo elemento e si definisce ordine di e, e si scrive $ord(e)$, il minimo numero intero i per il quale è $e^i = I$ (dove I è l'elemento neutro di G). Facciamo un altro esempio, Consideriamo il gruppo moltiplicativo Z_5, la cui tavola di moltiplicazione è riportata in Figura 186.

Tavola del gruppo Z* Φ(5) = 4				
	1	2	3	4
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

Figura 186: Tavola di moltiplicazione

L'elemento 1 ha ovviamente ordine 1. L'elemento 2 ha ordine 4, infatti $2^4 = 16 \pmod 5 = 1$. L'elemento 3 ha ordine 4, infatti $3^4 = 81 \pmod 5 = 1$. L'elemento 4 ha ordine 2, infatti $4^2 = 16 \pmod 5 = 1$.

Se G è un gruppo di ordine finito N, allora l'ordine di un qualsiasi suo elemento e è un divisore di N. Sia $\alpha \in Z_n^*$ e sia $q = ord(\alpha)$. Se $\alpha^t = 1 \pmod p$, allora q divide t. Se $n = s \pmod q$, cioè $s = kq + n$ per un intero k, $\alpha^n = \alpha^{s \pmod q} \pmod p = \alpha^s \pmod p$. Infatti, $\alpha^s \pmod p = \alpha^{kq+n} \pmod p = (\alpha^{kq}\alpha^n) \pmod p = ((\alpha^q \pmod p))^k(\alpha^n \pmod p)) \pmod p = \alpha^n \pmod p$.

La Figura 187 mostra un esempio di generazione delle chiavi in DSA.

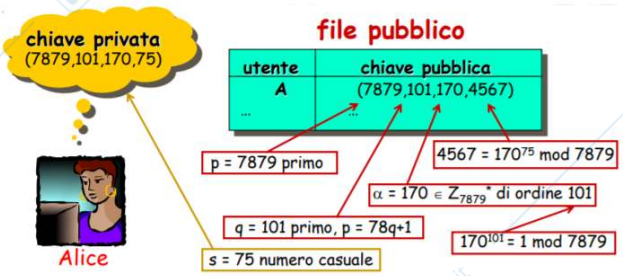


Figura 187: Generazione delle chiavi

Il procedimento di calcolo e apposizione della firma consta dei seguenti passaggi:

- Si generi un numero casuale r tale che $0 < r < q$
- Calcolare $\gamma = (a^r \mathrm{mod} p) \mathrm{mod} q$.
- Calcolare $\delta = (r^{-1}(H(m)+s*\gamma)) \mathrm{mod} q$, dove $H(m)$ è una funzione di hash SHA-d applicata al messaggio m.
- Nel caso in cui $\gamma = 0$ o $\delta = 0$ bisogna ricalcolare la firma.
- La firma è (γ, δ).

L'algoritmo esteso di Euclide può essere usato per calcolare l'inverso modulare $r^{-1} \mathrm{mod} q$. Infatti, $r^{-1} \pmod q$ esiste perché $r < q$ e q primo, quindi $gcd(q, r) = 1$.

La Figura 188 mostra un esempio di apposizione della firma in DSA.

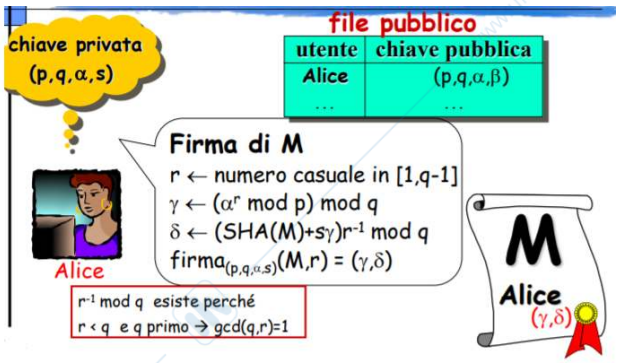


Figura 188: Generazione delle firme

La verifica della firma da parte del destinatario è possibile grazie ai seguenti passaggi:

- Rifiutare firme se non sono soddisfatte le condizioni $0 < \gamma < q$ e $0 < \delta < q$.
- Calcolare $e'' = \gamma\delta^{-1} \mathrm{mod} q$.

- Calcolare $e' = (H(m) * e'') \mathrm{mod} q$.
- Calcolare $v = ((\alpha^{e'} * \beta^{e''}) \mathrm{mod} p) \mathrm{mod} q$
- La firma è verificata se $v = \gamma$.

La Figura 189 mostra l'algoritmo che permette di verificare la firma.

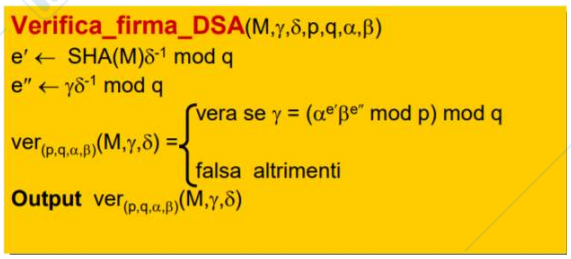


Figura 189: Verifica delle firme

La Figura 190 mostra la dimostrazione della correttezza della verifica firma DSA.

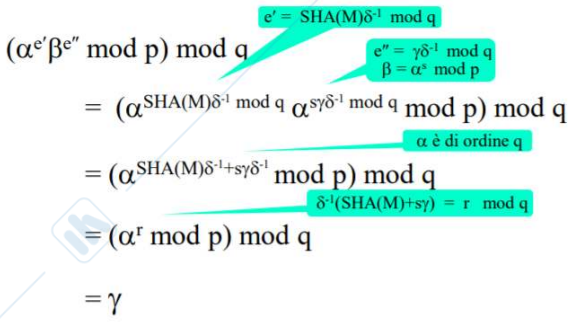


Figura 190: Correttezza della verifica

Come molti sistemi di crittografia a chiave pubblica, la sicurezza di DSA si basa sull'intrattabilità di un problema matematico, in questo caso sull'inesistenza di un algoritmo efficiente per il calcolo del logaritmo discreto.

Particolare attenzione va posta al calcolo della quantità r: deve essere generata casualmente in modo che non sia possibile risalirvi, nel qual caso sarebbe possibile risalire facilmente ad s (la chiave privata) a partire dalla firma.

La Figura 191 mostra l'algoritmo con cui si genera la firma. La lunghezza della firma è uguale a 320 bit. Le computazioni off-line sono: $r, s\gamma, r^{-1} \pmod q$. Le computazioni on-line sono: $SHA(M), +, \cdot$.

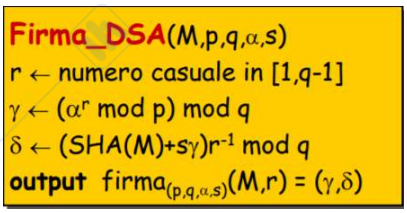


Figura 191: Generazione della firma

DSS richiede la scelta di due primi p e q con $2^{159} < q < 2^{160}$ e $2^{L-1} < p < 2^L$ con $L = 512 + 64j$. Q divide $p - 1$. Come scegliere p, q, α? Una prima ipotesi è quella di scegliere p

e, di conseguenza, scegliere q di 160 bit tale che $q|(p-1)$, ma ciò è impossibile perché si suppone che si conosca la fattorizzazione di p. Quindi si prosegue con il seguente procedimento:

- Scegli un primo q di 160 bit. La Figura 192 mostra la procedura di scelta di q.

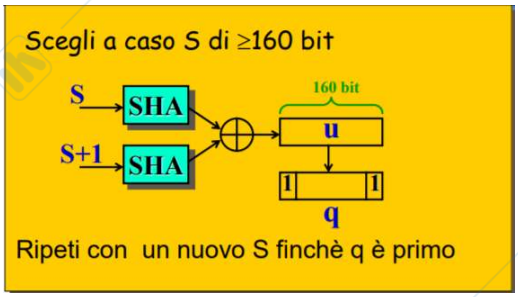


Figura 192: Scelta di q

- Calcolo di $p = qz + 1$, con z intero qualsiasi;
- p deve essere un numero primo multiplo di 64 lungo L bit, con $L \in [512, 1024]$. Dal 2000 il NIST impone la scelta a 1024
- Ne deriva che $z = (p-1)/q$;

Nello standard: Scegli un primo p di 512/1024 bit tale che $q|(p-1)$.

- Scegli X di 512 bit (oppure ... 1024 bit)
- $p \leftarrow X - ((X \pmod{2q}) - 1)$
  - se p è primo e $p \geq 2^{511}$ esci
  - altrimenti riprova

La Figura 193 mostra l'algoritmo con cui vengono scelti p e q.

```
Seleziona_pq(L)
(1)  Computa interi n e b tali che L-1=160n+b
(2)  repeat
(3)      repeat
(4)          S ← sequenza casuale di almeno 160 bit
(5)          g ← |S|
(6)          U ← SHA(S)⊕ SHA((S+1) mod 2^g)
(7)          Forma q da U ponendo il MSB ed il LSB ad 1
(8)      until q primo
(9)      C ← 0
(10)     N ← 2
(11)     repeat
(12)         for k=0 to n do V_k ← SHA(S+N+k) mod 2^g
(13)         W ← V_0+V_1·2^160+...+V_{n-1}·2^{160(n-1)}+(V_n mod 2^b)·2^{160n}
(14)         X ← W+2^{L-1}
(15)         p ← X-((X mod 2q)-1)
(16)     until (p primo) or (p<2^{L-1})
(17)     if p<2^{L-1}
(18)         then C ← C+1
(19)              N ← N+n+1
(20)              if C<4096 then goto step (12)
(21)                        else Help ← falso
(22)         else Help ← vero
(23) until Help
(24) return p,q,S,C
```

Figura 193: Algoritmo di scelta per p e q

La Figura 194 mostra nel dettaglio la scelta di p.

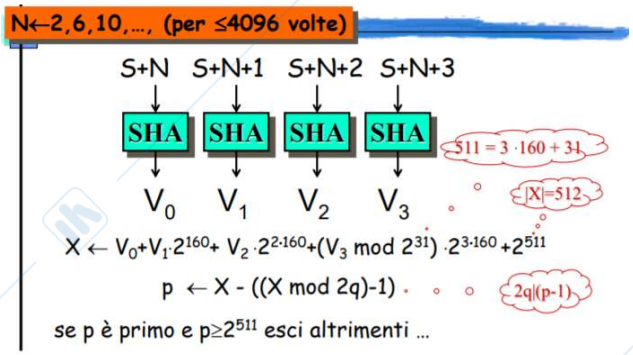


Figura 194: Scelta di p

La Figura 195 mostra l'algoritmo con cui calcolare $q|(p-1)$.

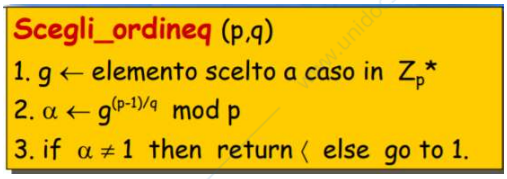


Figura 195: Calcolo di $q|(p-1)$

Quindi, $\alpha^q = (g^{(p-1)/q})^q = g^{p-1} = 1 \pmod p$. Quindi, $ord(\alpha)$ divide q. q primo $\rightarrow ord(\alpha) = 1$ oppure $ord(\alpha) = q$:

- $ord(\alpha) = 1$ se e solo se $\alpha = 1$
- $ord(\alpha) = q$ se $\alpha \neq 1$

Se g è un generatore allora $g^{\frac{(p-1)}{q}} \neq 1 \pmod p$. Il numero di generatori per un modulo primo p è $\phi(\phi(p)) = \phi(p-1)$. Risulta $\phi(p) > \frac{p}{(6lnln(p))}$ per $p > 4$. Quindi $\phi(\phi(p)) = \phi(p-1) > \frac{(p-1)}{(6lnln(p-1))}$. La probabilità successo è maggiore o uguale alla probabilità che g (scelto a caso) è generatore ($> \frac{1}{(6lnln(p-1))}$. Il numero medio di iterazioni $< 6lnln(p-1)$. La Figura 196 mostra il numero di iterazioni, a seconda dell'aumento dei bit.

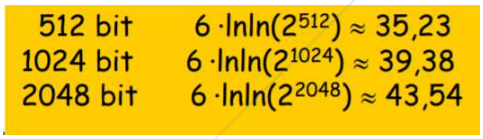


Figura 196: Numero iterazioni

Un attacco che si vuole provare è la volontà di falsificare la firma di M da parte di A. Devo calcolare $s = log_\alpha \beta \pmod p$. Quindi, devo calcolare Devo calcolare $\delta = log_\gamma(\alpha^{SHA(M)} \cdot \beta^\gamma$. Per fare questo:

- Scelgo γ a caso
- Determino δ tale che $\delta \leftarrow (SHA(M) + s\gamma)r^{-1} \pmod q$

Un altro attacco è dato dalla volontà di generare messaggi e firme da parte di A. Devo calcolare $z = log_\alpha(\gamma^\delta \beta^{-\gamma})$ e $M \leftarrow SHA^{-1}(z)$. Per fare questo:

- Scelgo γ, δ a caso
- Calcolo z tale che $\alpha^z = \gamma^\delta \beta^{-\gamma}$

La sicurezza di DSA è basata sul valore privato s. I valori p, q, α possono essere gli stessi per un gruppo di utenti. Un'autorità sceglie p, q, α; mentre il singolo utente sceglie solo s e calcola β.

La Figura 197 mostra un confronto dei tempi tra firme RSA e DSA.

	DSA	RSA	DSA con p,q,α comuni
precomputazioni	14 sec		4 sec
firma	0.3 sec	15 sec	0.3 sec
verifica	16 sec	1.5 sec	10 sec
	1-5 sec Off Cards		1-3 sec Off Cards

Figura 197: Tempi per firme RSA e DSA

Invece, la Figura 198 mostra un confronto delle prestazioni degli algortimi con processore Celeron 850MHz e sistema operativo Windows 2000. I valori in tabella hanno come unità di misura millisecondi/operazione.

	bit chiave	firma	Firma con precomputazione	verifica
RSA	512	1,92		0,13
DSA	512	1,77	1,19	2,02
RSA	1024	10,29		0,30
DSA	1024	5,50	2,27	6,38

Figura 198: Algoritmi per firme digitali

La Figura 199 mostra un confronto degli algoritmi con processore Pentium II 400 in OpenSSL

	bit chiave	firme/s	verifiche/s
RSA	512	342	3287
DSA	512	331	273
RSA	1024	62	1078
DSA	1024	112	94
RSA	2048	10	320
DSA	2048	34	27

Figura 199: Algoritmi per firme digitali

13 PKI - Public Key Infrastructure

Come vengono distribuite le chiavi pubbliche? Chi ci assicura che una chiave pubblica è quella di un prefissato utente? Esistono diverse tecniche:

- Invio point-to-point su canale fidato (ad esempio: scambio diretto, uso di un corriere fidato, etc.; oppure invio su canale pubblico o autenticazione (per esempio: hash su canale fidato)). Questa modalità è adatta se si tratta di un uso non frequente o di piccoli sistemi.

- Annuncio pubblico: invio ad altri utenti/broadcast chiave. Ad esempio: aggiunta della chiave pubblica o PGP ai messaggi inviati a forum pubblici. Il problema che sorge dall'utilizzo di questa tecnica è: ci dobbiamo fidare dell'annuncio?

- Directory disponibile pubblicamente. Un'entità fidata gestisce la directory di chiavi pubbliche, mentre ogni partecipante: registra la propria chiave pubblica di persona o in modo autenticato; può aggiornare la propria chiave se usata da troppo tempo o se la chiave privata è stata compromessa; e può accedere alla directory. Un prerequisito necessario per questa tecnica è la presenza di una comunicazione sicura ed autenticata.

- Autorità per le chiavi pubbliche. L'autorità gestisce la directory delle chiavi pubbliche. Ha una chiave pubblica nota a tutti gli utenti. Ogni utente chiede la chiave pubblica desiderata, l'autorità la invia. Gli svantaggi di questa tecnica è che preveda un server online e che si può creare un collo di bottiglia. La Figura 200 mostra un esempio di autorità per le chiavi pubbliche. Nella Figura, si suppone che Alice richieda all'autorità la chiave di Bob.

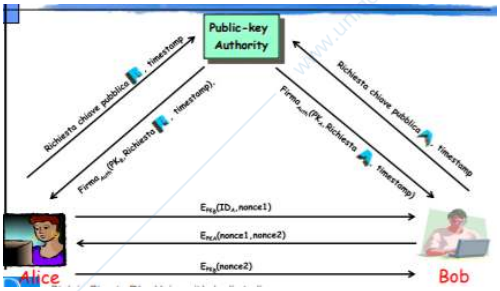


Figura 200: Autorità per le chiavi pubbliche

  Ottenuta una chiave si memorizza, attraverso il caching, ma è necessario aggiornarla periodicamente, richiedendo nuovamente la chiave all'autorità.

- Certificati per le chiavi pubbliche. Per quanto riguarda i certificati, nel mondo fisico (ad esempio, Carta di identità), un'autorità riconosciuta lega un nome ad una foto. Nel mondo digitale, un'autorità riconosciuta lega un nome ad una chiave pubblica.

Nella crittografia asimmetrica un certificato digitale è un documento elettronico che attesta l'associazione univoca tra una chiave pubblica e l'identità di un soggetto (una persona, una società, un computer, etc) che dichiara di utilizzarla nell'ambito delle procedure di cifratura asimmetrica e/o autenticazione tramite firma digitale.

Il certificato digitale contiene informazioni sulla chiave, informazioni sull'identità del proprietario (denominato soggetto) e la firma digitale di un'entità che ha verificato i contenuti del certificato (denominato emittente). Se la firma è valida e il software che esamina il certificato si affida all'emittente, allora può utilizzare tale chiave per comunicare in modo sicuro con il soggetto del certificato. Nella crittografia email, nella firma di codice e nei sistemi di firma elettronica, un soggetto del certificato è tipicamente una persona o un'organizzazione.

I certificati sono utili per la crittografia a chiave pubblica quando usata su larga scala. Infatti, scambiare la chiave pubblica in modo sicuro tra gli utenti diventa impraticabile, se non impossibile, quando il numero di utenti comincia a crescere. I certificati digitali sono una soluzione per superare questa difficoltà.

Lo scopo del certificato digitale è quello di garantire che una chiave pubblica sia associata alla vera identità del soggetto che la rivendica come propria. Le proprietà che un certificato digitale deve soddisfare sono:

- Ognuno può leggerli e determinare nome e chiave pubblica

- Ognuno può verificarli ed assicurarsi dell'autenticità
- Solo l'Autorità può crearli ed aggiornarli

I certificati possono contenere altri dati oltre alla chiave pubblica:

- Periodo di validità chiave pubblica
- Numero seriale o identificatore chiave
- Informazioni addizionali sulla chiave (ad esempio, algoritmi ed utilizzo)
- Informazioni addizionali sull'utente
- Stato della chiave pubblica

Il formato più comune per i certificati di chiave pubblica è definito da X.509. Poiché X.509 è molto generale, il formato è ulteriormente vincolato dai profili definiti per alcuni casi di utilizzo, come ad esempio l'infrastruttura di chiave pubblica (X.509) come definito in RFC 5280.

In principio quindi se Mario Rossi voleva inviare/ricevere un messaggio cifrato oppure voleva firmare digitalmente un documento doveva divulgare la sua chiave pubblica agli altri attori della comunicazione. Ogni persona che la possedeva poteva inviargli o ricevere da lui messaggi sicuri tramite cifratura asimmetrica con quella chiave pubblica oppure ricevere da lui documenti firmati con la chiave privata per poi verificare la firma con la suddetta chiave pubblica; tuttavia qualsiasi individuo poteva divulgare una differente chiave pubblica (di cui conosceva la relativa chiave privata) e dichiarare che era la chiave pubblica di Mario Rossi. Per evitare questo problema, Mario Rossi inserisce allora la sua chiave pubblica in un certificato firmato da una terza parte fidata ("trusted third party"): tutti quelli che riconoscono questa terza parte devono semplicemente controllarne la firma per decidere se la chiave pubblica appartiene veramente a Mario Rossi. In una PKI, la terza parte fidata sarà un'autorità di certificazione che apporrà anch'essa una firma sul certificato per validarlo. Nel web of trust, invece, la terza parte può essere un utente qualsiasi (ovvero la firma è quella o dello stesso utente (un'auto-certificazione) oppure di altri utenti ("endorsements")) e sarà compito di chi vuole comunicare con Mario Rossi decidere se questa terza parte è abbastanza fidata. In entrambi i casi, la firma certifica che la chiave pubblica dichiarata nel certificato appartiene al soggetto descritto dalle informazioni presenti sul certificato stesso (nome, cognome, indirizzo abitazione, indirizzo IP, etc.).

La Figura 201 mostra il funzionamento dell'operazione di rilascio di un certificato da parte dell'autorità.

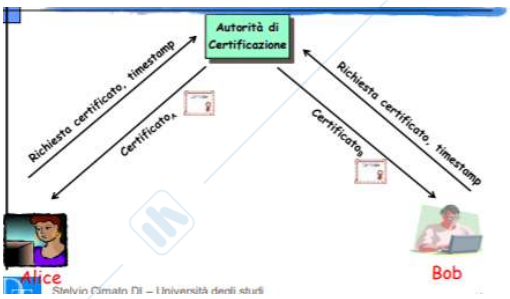


Figura 201: Richiesta di un certificato

La Figura 203 mostra il funzionamento dell'operazione di scambio di chiavi, mediante l'uso dei certificati.

Un certificato solitamente ha un intervallo temporale di validità, al di fuori del quale deve essere considerato non valido. Un certificato può essere revocato se si scopre che la relativa chiave privata è stata compromessa, oppure se la relazione specificata nello stesso (cioè la relazione tra

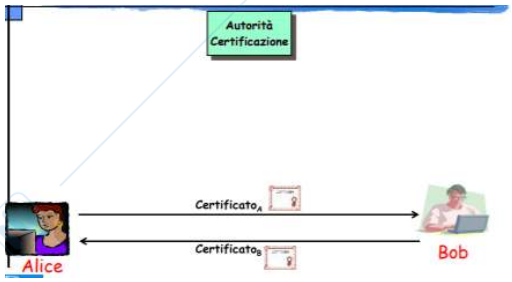


Figura 202: Scambio di un certificato

un soggetto ed una chiave pubblica) è incorretta o è cambiata; questo potrebbe succedere se, per esempio, una persona cambia lavoro oppure indirizzo. Ciò significa che un utente oltre a controllare che il certificato sia fidato (verificare che sia firmato da una CA riconosciuta) e non sia scaduto dovrebbe controllare anche che non sia stato revocato. Questo può essere fatto attraverso la lista dei certificati revocati (CRL). Una funzione chiave della PKI è proprio quella di tenere aggiornata la CRL. Un altro modo per verificare la validità di un certificato è quello di interrogare la CA attraverso un protocollo specifico come, per esempio, Online Certificate Status Protocol (OCSP). Riassunemndo, l'utente può richiedere la revoca di un certificato per diverse motivazioni:

- Compromissione chiave privata
- Informazioni non più valide (ad esempio, cambio affiliazione)
- Non più utile per lo scopo prefissato
- Compromissione algoritmo
- Perdita o malfunzionamento di security token, perdita di password o PIN
- Cambio politiche di sicurezza (ad esempio, la CA non supporta più servizi per certificati)

Può eseguire la revoca mediante diverse modalità:

- Data scadenza dentro un certificato: certificati "a breve scadenza"
- Notifica manuale: informazione tramite canali speciali. Metodo utilizzabile solo per sistemi piccoli o chiusi
- File pubblico di chiavi revocate: Certificate Revocation List (CRL). Un CRL (Certificate Revocation List) è un elenco di certificati digitali che sono stati revocati dalla Certification Authority (CA), prima della data di scadenza pianificata e non dovrebbero più essere considerati attendibili. Si tratta di una lista firmata dalla CA contenente:
  - Numeri seriali dei certificati emessi revocati (ma non ancora scaduti)
  - Quando è avvenuta la revoca
  - Altro (per esempio, motivi)

  La data della CRL indica quanto sia aggiornata. Le CRL vengono consultate per assicurarsi la validità e affidabilità dei certificati che si stanno per utilizzare. Operazione svolta dal browser ogni volta che ci si collega a un sito tramite SSL/TLS.
- Certificato di revoca: sostituisce certificato revocato nella directory

X.509 è uno standard proposto dall'Unione internazionale delle telecomunicazioni (ITU-T), usato per definire il formato dei certificati a chiave pubblica (PKC) e delle autorità di certificazione (CA). I certificati vengono utilizzati per la validazione dell'identità e la trasmissione di dati criptati che solo il possessore (persona, organizzazione o applicazione) di uno specifico certificato è in grado di decifrare e leggere. Questi vengono rilasciati dalle Certificate authority (CA), un soggetto terzo fidato che assicura la corrispondenza di una chiave pubblica a una determinata identità. Uno degli usi più diffusi di X.509 è nell'ambito internet, il certificato SSL/TLS viene usato nell'omonimo protocollo per criptare le comunicazioni tra un sito web e il nostro browser. Lo standard inoltre definisce sotto diversi aspetti l'utilizzo dei certificati nelle infrastrutture a chiave pubblica (PKI) e nei Privilege Management Infrastructure (PMI). Oltre al certificato a chiave pubblica e ai certificati di attributi vengono descritti anche i seguenti tipi di dato: certificate revocation list (CRL) e attribute certificate revocation list (ACRL). X.509 è una parte della serie X.500, serie di elenchi standard su computer per i servizi di directory.

La versione tre (v3) dei certificati digitali X.509 ha tre voci principali: il certificato, l'identificativo dell'algoritmo di firma del certificato e la firma del certificato.

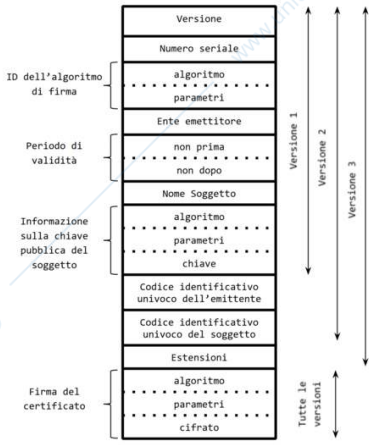


Figura 203: Struttura dei certificati digitali X.509

Il certificato viene descritto attraverso una serie di attribut:

- la versione
  - 1 default
  - 2 se presente "Issuer unique identifier" oppure "Subject unique identifier"
  - 3 se ci sono estensioni
- l'ID dell'algoritmo: algoritmo usato per firmare il Certificato, con i parametri associati. Si tratta di un'informazione ripetuta, e di conseguenza di un campo poco importante.
- il numero seriale: Si tratta di un valore intero, unico per ogni CA, che identifica senza ambiguità il certificato.
- l'emittente: Nome X.500 della CA che ha creato e firmato il Certificato
- il richiedente: Nome utente del Certificato, cioè chi conosce la chiave privata corrispondente
- la validità: viene indicata tramite la data d'inizio e quella di fine, che eventualmente determina il periodo di vita del certificato.

- informazioni sulla chiave pubblica del richiedente. Le informazioni sulla chiave pubblica del richiedente sono poi ulteriormente dettagliate con l'algoritmo utilizzato e la chiave pubblica stessa.
- Issuer unique identifier: campo opzionale. Si tratta di una stringa di bit utile per identificare la CA che ha emesso il Certificato nel caso che il nome X.500 sia stato riutilizzato.
- Subject unique identifier: campo opzionale. Si tratta di una stringa di bit utile per identificare il soggetto nel caso che il nome X.500 sia stato riutilizzato.
- estensioni
- altre voci facoltative

I codici identificativi univoci dell'emettitore e del richiedente sono stati introdotti nella versione due(v2), per poter permettere il loro riutilizzo in altri certificati. Per esempio se una Certification Authority (CA) fallisce e non è più autorizzata ad esercitare. In seguito un'altra CA con lo stesso nome, potrebbe registrarsi nella lista pubblica delle CA, anche se non ha nessuna relazione con la prima CA. Tuttavia la IETF raccomanda di non riutilizzare lo stesso nome. Perciò la versione due(v2) non ha avuto un ampio utilizzo in Internet. Le estensioni sono state introdotte nella versione tre (v3). Una CA può usare le estensioni solo per certificati che hanno uno specifico uso, come la firma di oggetti digitali. Ogni estensione del certificato ha un proprio identificativo, espresso come object identifier (OID), un insieme di valori che possono essere critici o non critici. Se analizzando un certificato un sistema incontra un'estensione critica che non riconosce, il certificato viene rigettato. Un'estensione non critica può essere ignorata se non riconosciuta, ma altrimenti deve essere processata. In tutte le versioni, il numero seriale deve essere univoco per ogni certificato emesso da una specifica Certification Authority(CA).

Con la notazione $Y << X >>$ si intende un certificato utente X creato dalla Autorità Y; mentre, con la notazione $Y\{I\}$ si intende la firma dell'hash di I da parte di Y.

Un servizio che utilizza i certificati X.509, richiede la conoscenza di una chiave pubblica, prima di usare il certificato che la contiene questo deve essere validato. Se il servizio non ha già una copia valida della chiave pubblica del CA che ha firmato il certificato, il nome della CA, e le relative informazioni (come il periodo di validità), si ha bisogno di ulteriori certificati per ottenere quella chiave pubblica. In generale, si ha bisogno di una lista di certificati, incluso il certificato del proprietario della chiave pubblica (end-entity certificate) firmato da una CA, seguito da uno o più certificati di CA firmati da altre CA. Queste catene di certificati (chiamate anche certification paths) sono necessarie poiché i software di solito sono inizializzati con un limitato numero di chiavi pubbliche di CA verificate.

Supponiamo che Alice voglia conoscere la chiave pubblica di Bob. Bob gli fornisce un certificato fornito da CA_2, ma Alice non conoscendo la chiave pubblica di CA_2 non può verificare l'autenticità della chiave pubblica di Bob. Attraverso la catena $CA_1 << CA_2 >> CA_2 << B >>$, Alice può verificare il certificato di Bob.

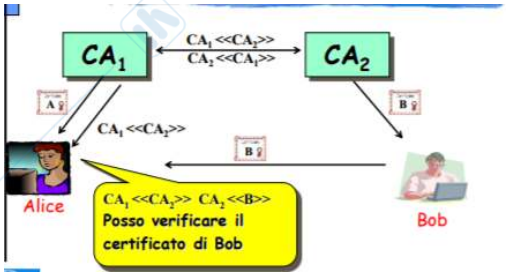


Figura 204: Catene di certificati digitali X.509

La Figura 205 mostra un esempio di gerarchia X.509, che mostra il certication path per cui A verifica il certificato di B.

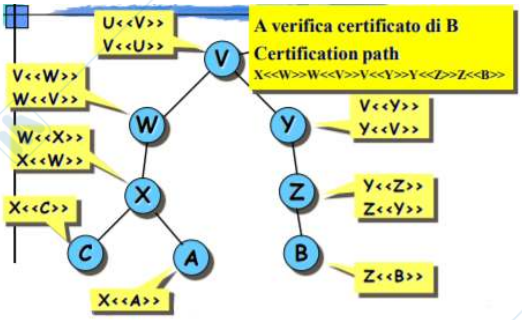


Figura 205: Catene di certificati digitali X.509

Lo standard X.509 definisce il formato e la semantica delle certificate revocation list (CRL), liste di revoca di certificati, per le PKI. Ogni oggetto della Certificate Revocation List include il numero seriale del certificato revocato e la data di revoca. Anche i file delle CRL sono firmate dalla Certificate Authority per prevenire manomissioni. Possono essere aggiunte informazioni opzionali come un limite temporale se la revoca si applica solo per un periodo di tempo o la ragione della revoca. Il problema con le CRL, come con tutte le blacklist, è la difficoltà di manutenerle e sono un modo inefficiente di distribuire informazioni critiche in sistema real-time. Tutto dipende dalla frequenza di aggiornamento delle CRL, anche se ad esempio sono aggiornate ogni ora, un certificato revocato potrebbe ancora essere accettato. Inoltre se una CRL non è disponibile, qualsiasi servizio si basi su questa non è utilizzabile. La Figura 206 mostra un esempio di struttura di una CRL.

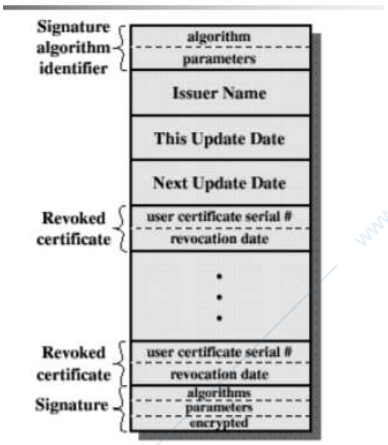


Figura 206: CRL in X.509

X.509 fornisce anche tre procedure di autenticazione:

- Autenticazione One-way
- Autenticazione Two-way
- Autenticazione Three-way

Assumiamo che entrambi conoscano le chiavi pubbliche, attraverso uno scambio dei certificati come primo messaggio, oppure certificati ottenuti dalla directory. Nell'autenticazione one-way

si ha un singolo trasferimento di informazioni da un utente (A) a un altro (B) e stabilisce quanto segue:

- Identità di A e messaggio generato da A
- Il messaggio è destinato a B
- Integrità e originalità del messaggio.

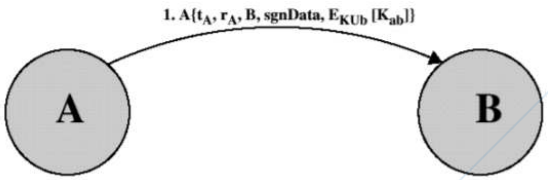


Figura 207: Autenticazione One-way

Nella Figura 207 t_A rappresenta il timestamp, per verificare l'expiration date. r_A rappresenta un nonce unico fino all'expiration time, per evitare attacchi di replay. $sgnData$ e K_{AB} sono campi opzionali e rappresentano rispettivamente le informazioni per B e la chiave di sessione.

Inoltre, l'autenticazione a due vie stabilisce quanto segue:

- Identità di B e che il messaggio di risposta è generato da B (la destinazione del primo messaggio)
- Il messaggio è destinato ad A
- Integrità e originalità della risposta

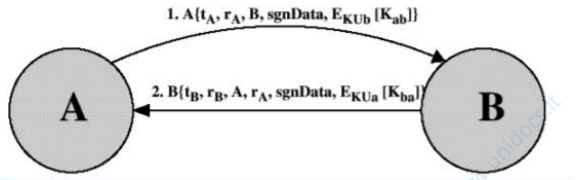


Figura 208: Autenticazione Two-way

Nell'autenticazione a tre vie il messaggio finale da A a B contiene una copia firmata del nonce (r_B) ricevuto da B. Questa aggiunta elimina la necessità di controllare i timestamp e viene utilizzata quando non sono disponibili orologi sincronizzati.

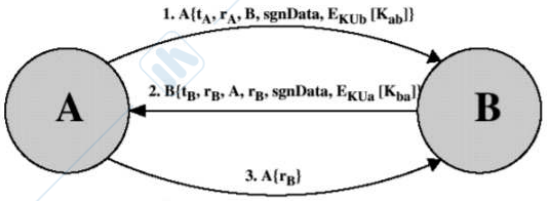


Figura 209: Autenticazione Three-way

La versione 3 del certificato X.509 è stata introdotta per soddisfare le inegatuezze della versione 2:

- Subject field non adeguato: nomi X.509 sono corti, e mancano dettagli identificativi che potrebbero essere utili
- Subject field non adeguato per le applicazioni che riconoscono entità dall'indirizzo email, URL
- Vi è necessità di indicare politiche di sicurezza
- Vi è necessità di limitare il danno che potrebbe fare una CA maliziosa, ponendo vincoli all'applicabilità di un particolare certificato
- É importante distinguere chiavi diverse usate dallo stesso utente in tempi diversi

La soluzione introdotta due approcci: aggiungere campi al formato versione 2 o aggiungere campi di estensione opzionali. L'introduzione di estensioni opzionali nella versione 3 è una soluzione flessibile ed meglio dell'aggiungere altri campi fissi alla versione 2. Ogni estensione contiene:

- Identificatore estensione
- Indicatore di criticità
- Valore estensione

Esistono tre categorie principali per le estensioni:

- Key and Policy Information: Informazioni sulle chiavi del soggetto e dell'emittente. Si tratta di indicatori della politica di certificazione. Ha i seguenti campi di estensione:
  - Authority key identifier: indica quale di più chiavi pubbliche della CA usare per verificare la firma di un certificato o della CRL
  - Subject key identifier: identifica quale di più chiavi pubbliche viene certificata
  - Key usage: restrizione sull'uso della chiave certificata, come scopo: (digital signature, key encryption, data encryption, key agreement, CA signature verification on certificates, CA signature verification on CRL)
  - Private-key usage period: periodo uso della chiave privata (per la firma, diverso periodo per chiave privata e pubblica)
  - Certificate policy: insieme di regole che indica l'applicabilità di un certificato ad una comunità e/o classi di applicazioni con requisiti di sicurezza comuni
  - Policy mappings: usato solo per CA da altre CA. Permette ad una CA di indicare che una propria politica può essere considerata equivalente ad un'altra politica usata dalla CA soggetto.
- Certificate Subject and Issuer Attributes: nomi alternativi per l'oggetto del certificato o l'emittente del certificato. Ha i seguenti campi di estensione:
  - Subject alternative name: contiene uno o più nomi alternativi, in formati alternativi. Importante per le applicazioni che hanno formati propri per i nomi (ad es., email, IPSec).
  - Issuer alternative name: contiene uno o più nomi alternativi, in formati alternativi.
  - Subject directory attributes: contiene attributi della directory X.500 per il soggetto del certificato.
- Certification Path Constraints: fornisce vincoli per i certificati emessi dalle autorità di certificazione per altre autorità di certificazione. Ha i seguenti campi di estensione:

- Basic constraints: indica se il soggetto può agire come CA. Se si, si possono specificare vincoli sulla lunghezza della certification path
- Name constraints: indica uno spazio dei nomi in cui tutti i seguenti certificati in un certification path devono essere
- Policy constraints: inibisce policy mappings per la parte rimanente della certification path

14 Accordo su chiavi

Per permettere un accordo sulle chiavi vedremo due schemi:

- Diffie-Hellman: basato sull'intrattabilità del problema del logaritmo discreto
- Puzzle di Merkle: non basato su alcuna assunzione computazionale

14.1 Scambio di chiavi Diffie-Hellman

Lo scambio di chiavi Diffie-Hellman (in inglese Diffie-Hellman key exchange) è un protocollo crittografico che consente a due entità di stabilire una chiave condivisa e segreta utilizzando un canale di comunicazione insicuro (pubblico) senza la necessità che le due parti si siano scambiate informazioni o si siano incontrate in precedenza. La chiave ottenuta mediante questo protocollo può essere successivamente impiegata per cifrare le comunicazioni successive tramite uno schema di crittografia simmetrica. Sebbene l'algoritmo in sé sia anonimo (cioè non autenticato) è alla base di numerosi protocolli autenticati ed è usato anche in alcune modalità di funzionamento del protocollo TLS. Lo schema del protocollo Diffie-Hellman fu pubblicato per la prima volta nel 1976 nell'ambito di una collaborazione tra Whitfield Diffie e Martin Hellman e fu il primo metodo pratico per due interlocutori di accordarsi su un segreto condiviso (la chiave) utilizzando un canale di comunicazione non protetto.

In matematica, in particolare in aritmetica modulare, un generatore modulo n o radice primitiva modulo n o semplicemente generatore se è chiaro il contesto, è un intero g le cui potenze modulo n sono congruenti con i numeri coprimi ad n. I generatori modulo n rivestono un'importanza considerevole in crittografia. Se $n \geq 1$ è un intero, i numeri coprimi ad n, considerati modulo n, costituiscono un gruppo rispetto all'operazione di moltiplicazione; esso viene generalmente indicato con $(\mathbb{Z}/n\mathbb{Z})^*$ oppure Z_n^*. Esso è un gruppo ciclico se e solo se n è uguale a 2, 4, p^k o $2p^k$ per un numero primo dispari p e $k \geq 1$. Un generatore di questo gruppo ciclico è chiamato anche elemento primitivo di Z_n^*. Si consideri per esempio $n = 14$. Gli elementi di $(\mathbb{Z}/14\mathbb{Z})^*$, sono le classi di congruenza di 1, 3, 5, 9, 11 e 13. Si ha che 3 è un generatore modulo 14, perché $3^2 = 9, 3^3 = 13, 3^4 = 11, 3^5 = 5$ e $3^6 = 1 \pmod{14}$. L'unica altra radice primitiva modulo 14 è 5.

Nell'implementazione originale (e più semplice) del protocollo si considera inizialmente un numero g, generatore del gruppo moltiplicativo degli interi modulo p, dove p è un numero primo. Uno dei due interlocutori, ad esempio Alice, sceglie un numero casuale "a" e calcola il valore $A = g^a \pmod{p}$ (dove mod indica l'operazione modulo, ovvero il resto della divisione intera) e lo invia attraverso il canale pubblico a Bob (l'altro interlocutore), assieme ai valori g e p. Bob da parte sua sceglie un numero casuale "b" , calcola $B = g^b \pmod{p}$ e lo invia ad Alice. A questo punto Alice calcola $K_A = B^a \pmod{p}$, mentre Bob calcola $K_B = A^b \pmod{p}$. I valori calcolati sono gli stessi, in quanto $B^a \pmod{p} = A^b \pmod{p}$. A questo punto i due interlocutori sono entrambi in possesso della chiave segreta e possono cominciare ad usarla per cifrare le comunicazioni successive. Un attaccante può ascoltare tutto lo scambio, ma per calcolare i valori a e b avrebbe bisogno di risolvere l'operazione del logaritmo discreto, che è computazionalmente onerosa e richiede parecchio tempo, in quanto sub-esponenziale (sicuramente molto più del tempo di conversazione tra i 2 interlocutori).

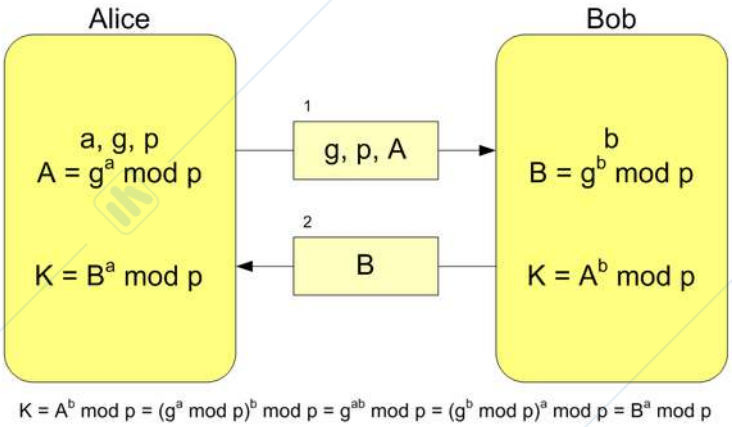


Figura 210: Diffie-Hellman

Il miglior algoritmo conosciuto calcola prima il logaritmo discreto $x \leftarrow log_{g,p}(g^x \pmod p)$, ma non assicura che i valori trovati e quelli reali siano equivalenti.

14.2 Puzzle di Merkle

In crittografia l'algoritmo del puzzle è un esempio istruttivo di algoritmo di crittografia a chiave pubblica. Sebbene sia praticamente irrealizzabile contiene molte delle idee di base di algoritmi più complessi, in particolare risolve il problema dello scambio della chiave, cioè consente a due persone di scambiare messaggi segreti anche se non si sono mai scambiate un segreto prima di allora (la chiave). L'algoritmo fu proposto da Merkle nell'ottobre del 1974 e pubblicato nel 1978.

Supponiamo che Alice e Bob vogliano comunicare in modo sicuro, al riparo di Eva che ha accesso al canale di comunicazione e quindi potrebbe intercettare i loro scambi di informazione. Bob e Alice devono scambiarsi la chiave, ma non hanno un canale sicuro per farlo. Alice crea dei puzzle ovvero crea tanti messaggi codificati con un algoritmo facile da forzare (il puzzle è una metafora, è abbastanza facile da risolvere come è abbastanza facile forzare il messaggio cifrato). Bob riceve questi puzzle (la chiave pubblica), diciamo qualche milione, e ne sceglie uno a caso. Lo risolve, ovvero forza la cifratura. A questo punto Bob legge il messaggio contenuto nel puzzle che dice qualcosa del genere: "Sono il puzzle numero x, la chiave numero x è y". A questo punto Bob comunica ad Alice il numero del puzzle: x. In questo modo Alice e Bob hanno una chiave in comune: y. Eva avendo intercettato lo scambio dei puzzle ha anche lei la chiave y, ma è mischiata tra un milione di altre chiavi; inoltre conosce il numero della chiave. A questo punto potrebbe provare a risolvere tutti i puzzle fino a trovare il puzzle numero x, ma il numero dei puzzle è scelto in modo che questa operazione sia computazionalmente troppo lunga.

Supponiamo che Alice mandi 2^{20} (circa un milione) di puzzle a Bob. Mediamente Eva per trovare il puzzle che contiene la chiave usata tra Bob e Alice dovrà risolvere la metà dei puzzle: 2^{19}. Se per risolvere un puzzle Bob ci mette un minuto, Eva impiegherà un anno a risolverne la metà. Quindi in media Eva ha bisogno di un anno per decifrare il messaggio intercettato.

Il metodo non è considerato sufficientemente sicuro perché il tempo che ci metterà Eva a trovare la chiave cresce quadraticamente rispetto a quello impiegato da Bob. I moderni protocolli di crittografia asimmetrica e di scambio di chiave richiedono algoritmi di attacco e legittimi con complessità computazionali che divergono esponenzialmente.

Figura 211: Computazione per puzzle di Merkle

15 Secret Sharing

Il secrete sharing (detta anche suddivisione segreta) si riferisce ai metodi per distribuire un segreto tra un gruppo di partecipanti, a ciascuno dei quali viene assegnata una parte del segreto. Il segreto può essere ricostruito solo quando un numero sufficiente, di possibilmente diversi tipi, di azioni viene combinato insieme; le singole azioni non sono utili da sole.

In un tipo di schema di condivisione segreta c'è un dealer e n player. Il dealer fornisce una parte del segreto ai player, ma solo quando sono soddisfatte determinate condizioni i player saranno in grado di ricostruire il segreto dalle loro azioni. Il dealer ci riesce dando a ciascun player una quota in modo tale che qualsiasi gruppo di k (per soglia) o più giocatori possano ricostruire insieme il segreto ma nessun gruppo di meno di k giocatori può farlo. Tale sistema è chiamato schema a soglia (k, n).

La condivisione segreta è stata inventata in modo indipendente da Adi Shamir [1] e George Blakley.

Si dividono in due gruppi: schema (n, n) e schema (k, n). Lo schema (n, n) divide il segreto tra n partecipanti, e per ricostruirlo sono necessarie le share di tutti gli n partecipanti. Lo schema (k, n) invece divide il segreto sempre tra n partecipanti, ma bastano $k < n$ partecipanti per ricostruirlo.

15.1 Schema (n, n)

Ecco che cosa occorre fare:

- scegliere un numero primo p
- scegliere il numero segreto S, con $S < p$
- scegliere casualmente $n - 1$ valori minori di p, e assegnarli alle a_{n-1} share: $a_1, a_2 ... a_{n-1}$.
- lo share a_n lo ottengo così: $a_n = S - a_1 - a_2 - ... - a_{n-1} \pmod p$
- distribuisco gli a ai miei n utenti
- per ricostruire, tutti gli utenti devono condividere i loro share: $S = a_1 + a_2 + ... + a_{n-1} + a_n \pmod p$.

La Figura 212 mostra una rappresentazione grafica della ricostruzione del segreto seg

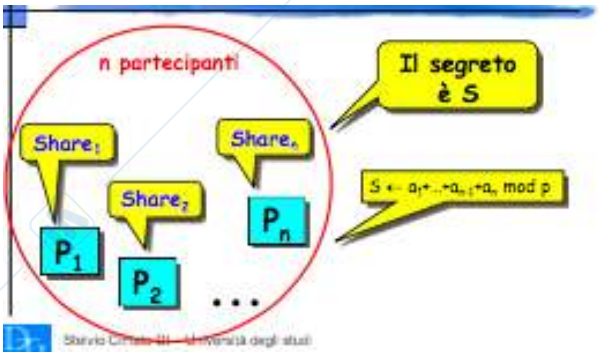


Figura 212: Ricostruzione del segreto

Voglio uno schema (5,5), cioè divido il segreto in 5 share. Ecco tutti i passaggi

- Scelgo un numero primo che più mi aggrada: $p = 7$.
- Decido che il segreto sia S = 3.
- Scelgo ora $n - 1$ valori casuali, per assegnarli agli share fino a a_{n-1}:
  - $a_1 = 4$
  - $a_2 = 5$
  - $a_3 = 2$
  - $a_4 = 4$
- É possibile anche avere valori ripetuti, non è un problema.
- Per trovare a_n devo risolvere questo calcolo: $a_n = S - a_1 - a_2 - ... - a_{n-1} \pmod p$. In questo caso, $a_n = 3 - 4 - 5 - 2 - 4 \pmod 7 = 2 \pmod 7$. Ho quindi tutte le 5 share:
  - $a_1 = 4$
  - $a_2 = 5$
  - $a_3 = 2$
  - $a_4 = 4$
  - $a_5 = 2$

  e le distribuisco ai miei 5 utenti. Per ricostruire il segreto S, avendo a disposizione solo le share, devo fare: $S = a_1 + a_2 + ... + a_{n-1} + a_n$. $S = 4 + 5 + 2 + 4 + 2 \pmod 7 = 17 \pmod 7 = 3 \pmod 7$. E infatti, il segreto era 3.

15.2 Schema (k, n)

Questo schema è più complesso: divido il segreto in n share, ma ne bastano $k < n$ per ricostruirlo. Come sopra, devo scegliere il primo p, il segreto $S < p$ e $k - 1$ valori: $a_1, a_2...a_{k-1}$, tutti minori di p. In questo caso, questi $k - 1$ valori non sono le share. Infatti, le share le ottengo calcolando la seguente espressione:

for $(i = 1; i <= n; i++)$ {

$$y_i = S + a_1 x + a_2 x^2 + ... + a_{k-1} x^{k-1} \pmod p$$

}

Questo vuol dire che l'i-esima share è ottenuta a partire da questa formula, in cui i valori a_k li posseggo, perché li ho scelti a caso sopra al posto delle x sostituisco il valore di i:

- nella share numero 1, $x = 1$
- nella share numero 2, $x = 2$
- e così via per tutte le n share

Distribuisco quindi i vari y_i, assieme alla i stessa: ovvero, l'utente conosce il valore della share ed il numero i della sua share.+

Per ricostruire il segreto, mi bastano k share, perché mi permettono di costruire un sistema così:

$$y_1 = S + a_1 x + a_2 x^2 + ... + a_{k-1} x^{k-1} \pmod p$$

$$y_2 = S + a_1 x + a_2 x^2 + ... + a_{k-1} x^{k-1} \pmod p$$

...

$$y_k = S + a_1 x + a_2 x^2 + ... + a_{k-1} x^{k-1} \pmod p$$

In questo caso: le varie y_i sono note, le x le sostituisco come prima, con il valore di i. Le incognite sono quindi S e i $k-1$ valori di a: k incognite, k equazioni: il sistema si può risolvere. La Figura 213 mostra una rappresentazione grafica della ricostruzione del segreto secondo questo schema.

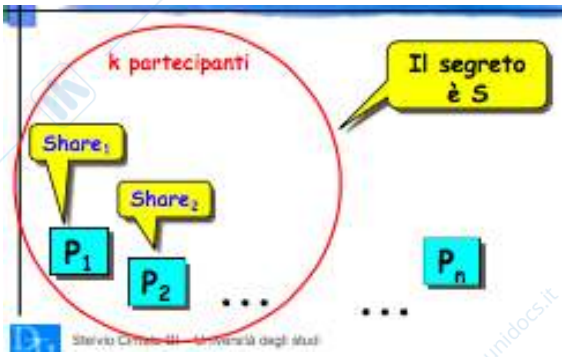


Figura 213: Ricostruzione del segreto

Voglio uno schema (3,5). Scelgo $p = 7$ e $S = 3$ come prima. Scelgo a caso i $k-1$ valori di a:

- $a_1 = 4$
- $a_2 = 3$

Calcolo le share secondo la formula $y_i = S + a_1 x + a_2 x^2 + ... + a_{k-1} x^{k-1} \pmod p$:

- $y_1 = 3 + 4 * 1 + 3 * 1^2 \pmod 7 = 3 \pmod 7$
- $y_2 = 3 + 4 * 2 + 3 * 2^2 \pmod 7 = 2 \pmod 7$
- $y_3 = 3 + 4 * 3 + 3 * 3^2 \pmod 7 = 0 \pmod 7$
- $y_4 = 3 + 4 * 4 + 3 * 4^2 \pmod 7 = 4 \pmod 7$
- $y_5 = 3 + 4 * 5 + 3 * 5^2 \pmod 7 = 0 \pmod 7$

Posso distribuire le share:

- $y_1 = 3$
- $y_2 = 2$
- $y_3 = 0$
- $y_4 = 4$
- $y_5 = 0$

Per ricostruire, devo impostare il sistema con k di questi valori. Ad esempio, scelgo y_1, y_3, y_5:

- $y_1 = S + a_1 * 1 + a_2 * 1^2 \pmod 7$
- $y_3 = S + a_1 * 3 + a_2 * 3^2 \pmod 7$
- $y_5 = S + a_1 * 5 + a_2 * 5^2 \pmod 7$

Lo stesso schema può essere visto utilizzando la formula di Lagrange per l'interpolazione polinomiale: un insieme di n punti individua univocamente una curva di ordine $n-1$. Ad esempio, dati due punti esiste una sola retta passante. Per uno schema a soglia k utilizzo un polinomio di grado $k-1$: $y = S + a_1 i + ... + a_{k-1} i^{k-1}$. Distribuisco a ciascun partecipante un punto. Solo k partecipanti avranno k punti e possono ricostruire la curva, risolvendo k equazioni in k incognite. La Figura 214 mostra la procedura per il calcolo del segreto.

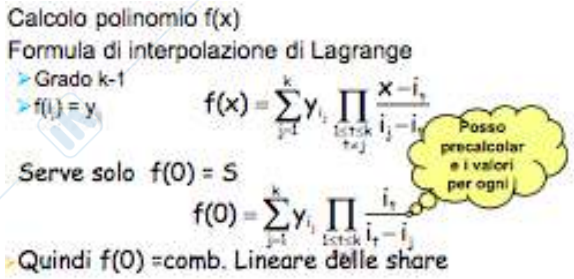


Figura 214: Ricostruzione del segreto