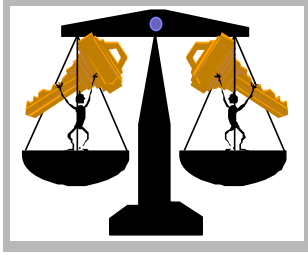
Capitolo 4



Meccanismi simmetrici

4.1 Cifrari simmetrici

Il principale oggetto di studio della moderna **Crittografia** simmetrica è il **Cifrario a chiave segreta**.

Gli attuali Cifrari simmetrici sono anche detti **convenzionali** essendo i discendenti diretti di quelli messi a punto nel periodo classico; tipicamente sono usati per la difesa della riservatezza, ma sono impiegati anche come meccanismi per la generazione di numeri pseudocasuali, per l'autenticazione e per l'identificazione.

I Cifrari attualmente in uso sono:

- **robusti** (resistono a tutti gli attacchi di crittanalisi finora individuati),
- **veloci** (nelle realizzazioni con Hw special purpose elaborano diversi milioni di bit al secondo),
- **efficaci** (gestiscono stringhe binarie di lunghezza arbitraria),
- **efficienti** (il testo cifrato è praticamente lungo quanto il testo in chiaro).

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 $\mathbf{E_k}$ e decifrata da $\mathbf{D_k}$.

Le attività che **A** e **B** devono svolgere sono:

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

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.

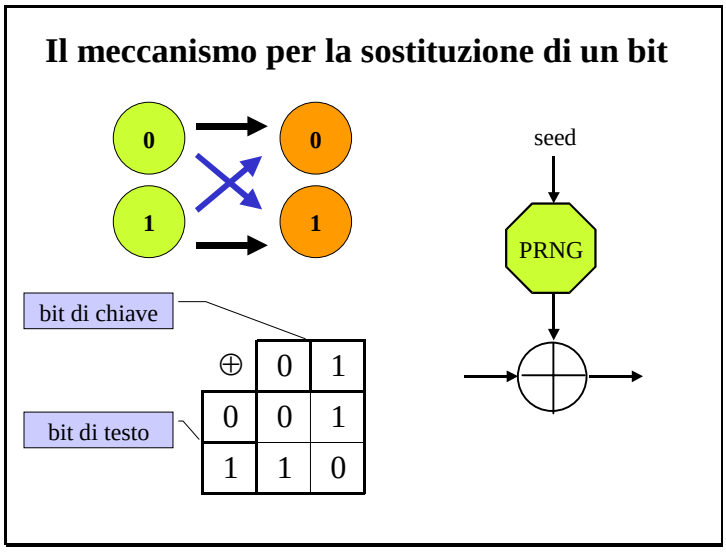
- Il **Cifrario a flusso** (*stream cipher*) si ispira a **One-time pad**, trasforma pochi bit alla volta (tipicamente un solo bit od un byte) e risulta particolarmente adatto quando occorre proteggere singoli dati, generati uno dopo l'altro (trasmissione seriale di caratteri ASCII, comunicazione fonica digitalizzata, ecc.).
- Il **Cifrario a blocchi** (*block cipher*) si ispira al **Cifrario poligrafico** ed al **Cifrario composto**, 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.).

Vedremo che la demarcazione non è poi così netta. E' comunque importante prendere atto fin d'ora che

- il cifrario a flusso è più veloce del cifrario a blocchi,
- il cifrario a blocchi è più sicuro del cifrario a flusso.

4.2 Cifrari a flusso

I Cifrari a flusso realizzano una trasformazione **variabile al progredire del testo**.



Discutendo il Cifrario di Vernam, è stato evidenziato che l'hardware necessario per trasformare e ritrasformare un bit alla volta è un semplice gate EX-OR.

Un bit di testo (in chiaro o cifrato) ha, infatti, due sole possibili sostituzioni:

$0 \to 0$ $\quad$ $0 \to 1$

$1 \to 1$ $\quad$ $1 \to 0$

Per eseguire o l'una o l'altra è sufficiente sommarlo **modulo due** con un corrispondente bit di chiave. Si ha dunque:

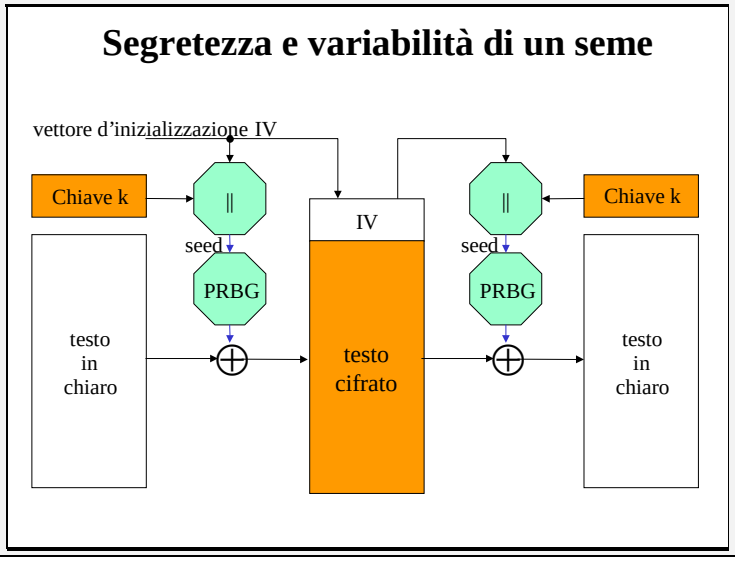
CIFRATURA: $\mathbf{c_i = m_i \oplus k_i}$

DECIFRAZIONE: $\mathbf{c_i \oplus k_i = m_i \oplus k_i \oplus k_i = m_i}$

La sicurezza del metodo è concentrata nella realizzazione di una stringa di **bit aleatori** $\mathbf{k_i}$ lunga quanto il testo.

In realtà i bit di chiave non possono che essere **pseudocasuali** ed è questo il fatto che non consente di ottenere una sicurezza perfetta: in trasmissione ed in ricezione occorre, infatti, impiegare due **PRNG**, detti **generatori di flusso di chiave**, per garantire la perfetta sincronizzazione dei due flussi.

Abbiamo già osservato in cap. 2 che la stringa dei bit di chiave deve avere un periodo grandissimo (almeno 10^{50}). Ciò consente di suddividere la sequenza in moltissime sottosequenze e di rendere imprevedibile quella che viene di volta in volta impiegata tramite un seme **casuale**, concordato in **segreto** dai due corrispondenti.



ESEMPIO - In figura è mostrata la soluzione adottata nel protocollo WEP (*Wired Equivalent Privacy*) per reti wireless.

Il mittente sceglie di volta in volta un diverso vettore di inizializzazione, che concatena alla chiave segreta per determinare il seed e che trasmette in chiaro in testa al cifrato.

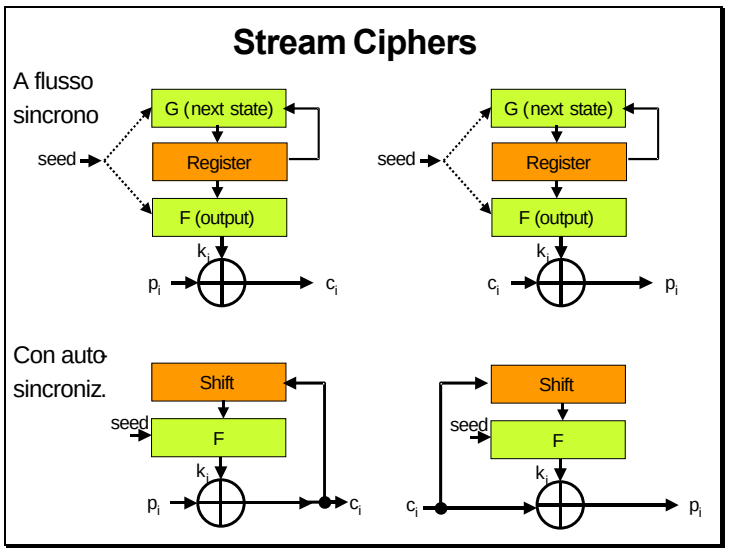
Il destinatario estrae IV, lo concatena con la chiave segreta che condivide con il mittente ed inizializza così la generazione del flusso di chiave esattamente dallo stesso punto da cui è partito il corrispondente.

La soluzione è però oggi giudicata insicura.

Una volta fissata la dimensione del seed, la necessità di concatenare **k** e **IV** limita la dimensione della chiave, rendendola più facilmente prevedibile.

Esistono due differenti tipi di Cifrario a flusso:

- **a flusso sincrono**,
- con **auto-sincronizzazione**.



Nei Cifrari a flusso sincrono il flusso dei bit di chiave è indipendente dal flusso dei bit di testo; la chiave, il seed, può interessare o la F, o la G; la funzione interessata deve essere **unidirezionale**. Per un corretto funzionamento i generatori di trasmissione e di ricezione devono mantenersi al passo: quando si verifica un disallineamento, sorgente e destinazione devono far ripartire i due generatori, curando anche di scegliere un diverso punto di inizio della sequenza.

Nei Cifrari a flusso con auto-sincronizzazione il flusso dei bit di chiave **dipende dal flusso dei bit di testo cifrato**: si noti in figura il ruolo svolto dai due **registri a scorrimento**.

Un'eventuale perdita di sincronismo non richiede alcun intervento da parte degli utenti: dopo un breve transitorio (il tempo necessario a ricaricare in modo corretto lo shift di ricezione) i due generatori si rimettono al passo da soli.

La più comune causa di disallineamento è la **perdita d'integrità** del testo cifrato, quale può essere generata o da un evento casuale o da un attacco intenzionale.

Consideriamo dapprima l'eventualità che sia **cancellato** o **inserito** un bit. Le stazioni di un Cifrario a flusso sincrono perdono il sincronismo e l'errore si propaga su tutta la restante parte del messaggio; le stazioni del Cifrario con auto-sincronizzazione perdono il sincronismo solo temporaneamente e consentono di riprendere correttamente la decifrazione al termine del transitorio.

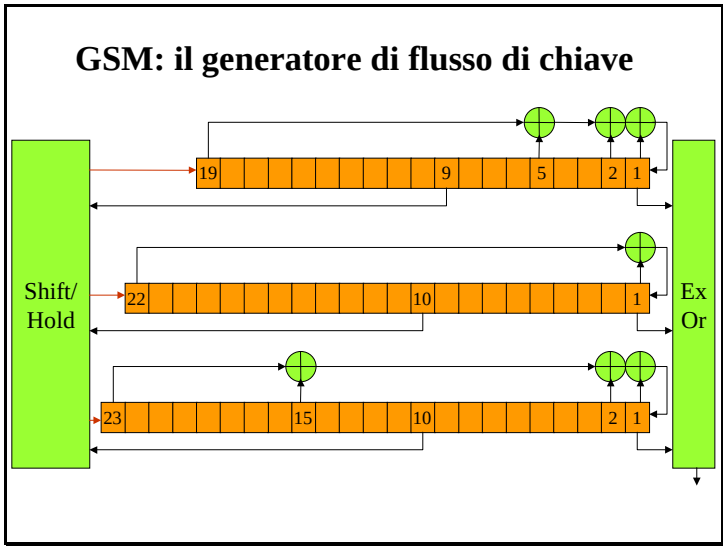
Consideriamo ora l'eventualità che sia **modificato** un bit. Nei Cifrari a flusso sincrono la violazione dell'integrità non si propaga ai bit successivi e non è rilevabile; i Cifrari con auto-sincronizzazione decifrano invece in modo errato finché il bit modificato è presente nello shif di ricezione.

I Cifrari a flusso sincrono sono più usati di quelli con auto-sincronizzazione. Molti sono stati gli studi indirizzati ad individuare PRNG veloci e di facile realizzazione. Per allargare il quadro delineato nel cap. 2, esamineremo due ulteriori soluzioni, una basata sullo scorrimento di bit e l'altra sullo scambio di posizione di byte.

Oggi si preferisce impiegare la funzione E di un Cifrario simmetrico a blocchi: la modalità CTR è stata discussa in 2.2.1, le modalità OFB e CFB verranno esaminate in 4.3.7.

4.2.1 L'algoritmo A5 del GSM

Lo standard **GSM** per i telefoni mobili impiega un Cifrario a flusso sincrono, detto algoritmo **A5**, per proteggere la riservatezza delle comunicazioni sulle tratte radio.



Il PRN è formato da tre registri a scorrimento, ciascuno con retroazione lineare e con un comando di shift/ hold che gli indica, in ogni intervallo di clock, se deve o non deve attuare lo scorrimento.

Le retroazioni calcolano l'EX-OR dei bit la cui posizione nello shift compare come esponente nei seguenti **polinomi primitivi**:

$$x^{19} + x^5 + x^2 + x + 1$$
$$x^{22} + x + 1$$
$$x^{23} + x^{15} + x^2 + x + 1$$

L'uso dei polinomi primitivi attribuisce ad ogni registro un **periodo di massima lunghezza**: in opportuna sequenza il registro assume, infatti, tutti gli stati interni possibili ad eccezione dello stato di tutti zeri.

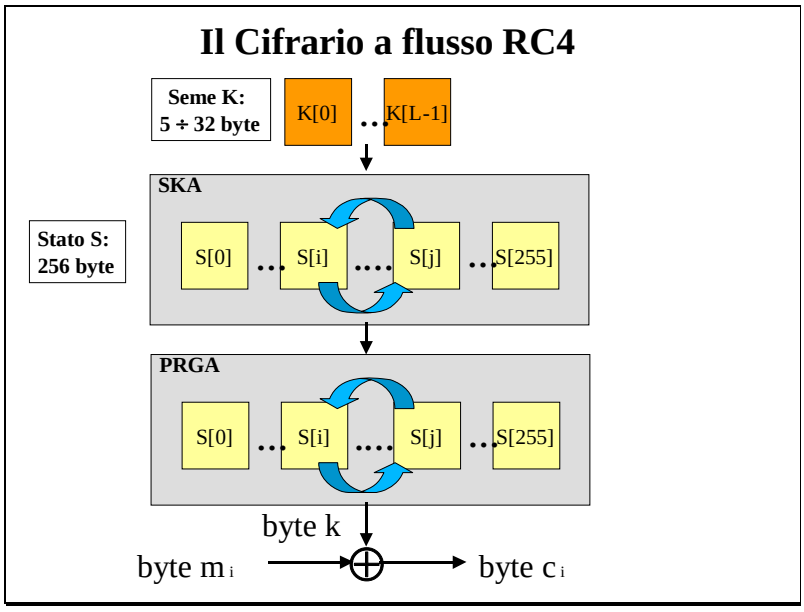
Ad ogni passo ciascun registro è shiftato solo se il suo bit "centrale" concorda con il valore di maggioranza dei bit centrali dei tre registri. Il fatto che almeno due registri alla volta vengano shiftati attribuisce al loro insieme un comportamento che dipende in modo **non lineare** dallo stato interno e quindi dal seme di inizializzazione.

La segretezza e la lunghezza della chiave (64 bit, ma in realtà, per motivi non del tutto chiari, gli ultimi 10 bit hanno sempre valore 0) dovrebbero garantire la non riproducibilità del flusso di chiave da parte di un intruso. Ross Andersen ha dimostrato che non è così (v. http://www. chem.leeds.ac.uk/ICASM/people/jon/a5.html). Molti altri ricercatori hanno individuato tecniche di decrittazione abbastanza efficaci (da pochi secondi a qualche minuto, secondo la quantità di dati intercettati e della memoria a disposizione per sottoporli a pre-elaborazioni).

Nei videotelefoni di nuova generazione (standard UMTS) il flusso di chiave è generato in modo più sicuro dal Cifrario a blocchi Kasumi. A differenza del GSM, UMTS impiega flussi diversi in trasmissione ed in ricezione.

4.2.2 L'algoritmo RC4

L'algoritmo RC4[29], inventato da R. Rivest nel 1987 e tenuto segreto, è stato divulgato su Internet nel 1994. Inizialmente considerato molto robusto (ma poi la crittanalisi ha evidenziato diverse vulnerabilità), ha avuto diverse applicazioni (Lotus Notes, Oracle, Secure SQL, SSL, WEP). RC4 impiega un **registro di stato interno** formato da 256 celle **S[i]**, in cui sono inizialmente sistemati, in ordine crescente, i 256 possibili valori di un byte; è poi operata una permutazione definita da un seme di lunghezza **L**, variabile tra 40 e 256 bit.



A tale attività provvede l'algoritmo **SKA**, che esegue scambi di contenuto (*swap*) tra coppie di celle:

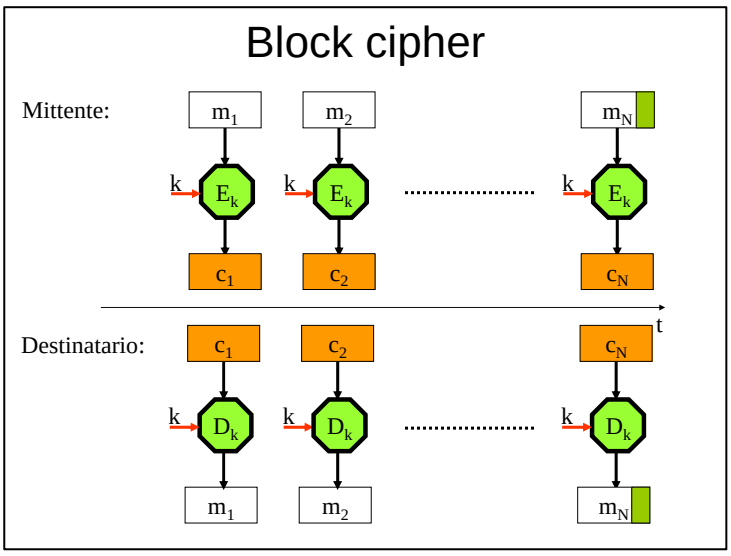
```
for i = 0 to 255
  S[i] = i;
j = 0;
for i = 0 to 255
  j = (j + S[i] + K[i mod L]) mod 256
  scambia i valori di S[i] e S[j]
```

Il flusso di byte pseudocasuali **k** (sono più di 10^{100}) è generato dall'algoritmo **PRGA**, anch'esso basato sul paradigma dello scambio:

```
i = 0, j = 0
loop: i = (i +1) mod 256
      j = (j + S[i]) mod 256
      scambia i valori di S[i] e S[j]
      k = S[(S[i] + S[j]) mod 256]
```

[29] Nel sito del corso è disponibile il file RC4.zip, contenente un eseguibile ed i relativi sorgenti; l'approfondimento è stato fatto da Ginevra Cicconi

4.3 Cifrari a blocchi



Il testo da cifrare viene suddiviso in blocchi m_1, m_2, .., m_N di lunghezza prefissata L.

Se la lunghezza del testo in chiaro non è un multiplo intero della lunghezza del blocco, il mittente aggiunge all'ultimo blocco simboli di riempimento privi di significato (*padding*).

La regola di trasformazione dei blocchi, fissata dalla chiave **k**, è una sostituzione mono-alfabetica, ma la grossa dimensione dei blocchi (nel caso di un testo letterario codificato in ASCII un blocco include la codifica di almeno otto caratteri) la rende immune da un attacco con statistiche.

Anche la decifrazione procede a blocchi ed è fissata dalla chiave **k**. In conclusione si ha:

CIFRATURA: $c_i = E(m_i, k)$, per i =1,2, ..

DECIFRAZIONE: $D(c_i, k) = m_i$, per i =1,2, ..

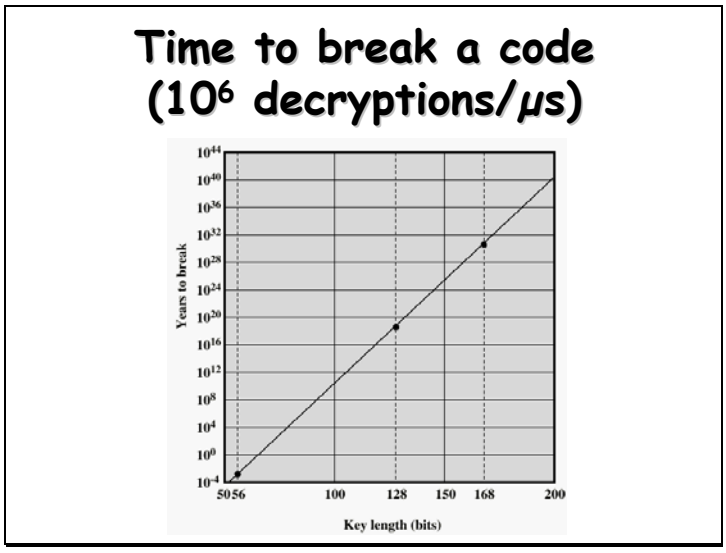
Sono in uso diversi standard per consentire alla macchina del destinatario di eliminare automaticamente i bit di padding aggiunti dalla macchina del mittente. PKCS#7, ad esempio, prevede che il completamento dell'ultimo blocco sia fatto da **n byte di valore n**, se **8 × n** è il numero di bit che mancano nel testo in chiaro.

4.3.1 Lunghezza della chiave

n° di bit	n° di chiavi
56	$2^{56} = 7{,}2 \times 10^{16}$
128	$2^{128} = 3{,}4 \times 10^{38}$
168	$2^{168} = 3{,}7 \times 10^{50}$
192	$2^{192} = 6{,}3 \times 10^{57}$

La tabella a lato riporta una valutazione dello spazio delle chiavi (2^N) in corrispondenza di alcuni valori della loro lunghezza N.

Tutti i numeri nella seconda colonna sono oggettivamente molto grandi, ma potrebbero non spaventare un attacco con forza bruta: per condurlo basta, infatti, disporre di un testo cifrato e scrivere un programma che lo decifri provando uno dopo l'altro tutti i possibili valori della chiave. Per decidere se la chiave in prova è quella giusta occorre come minimo conoscere il linguaggio impiegato nel testo in chiaro: l'analisi statistica del testo decifrato fornisce già una buona indicazione se prenderla o no in considerazione. Le cose sono ancora più semplici se si conosce anche il corrispondente testo in chiaro.



Il tempo necessario per rompere il Cifrario dipende naturalmente dalla potenza della macchina a disposizione dell'intruso.

In figura[30] si è fatta l'ipotesi che l'intruso non sia un *hacker* estemporaneo, ma un membro di una grossa Organizzazione (o criminale, o governativa) dotata di un super calcolatore in grado di eseguire 10^{12} decifrazioni al secondo.

In media la macchina deve controllare solo la metà dello spazio delle chiavi; il tempo medio di esecuzione dell'attacco è dunque:

$$T = 2^{N-1} / 10^{12} \text{ sec} = 2^{N-1} / (3{,}1 \times 10^{19}) \text{ anni}$$

Per individuare una chiave di 56 bit una macchina di questo tipo impiega circa 10 ore; con 168 bit occorrono circa 6.10^{30} anni.

ESEMPI - Nel 1998 il **DES Cracker**, una macchina parallela costata "solo" 250.000 $, è riuscito ad individuare in meno di 3 giorni una chiave di 56 bit. Fino a poco tempo fa, per venire incontro alla richiesta di FBI e CIA, gli Stati Uniti hanno consentito l'esportazione di algoritmi crittografici solo se dotati di una chiave non più lunga di 40 bit.

[30] slide prelevata dal sito di William Stallings

Il periodo di tempo in cui un'informazione deve rimanere segreta può variare da poche ore a molti anni: anche questo dato deve dunque essere messo in conto quando si vuole un dimensionamento sicuro della chiave di un algoritmo simmetrico a blocchi.

Nel 1996, durante un summit dei più famosi crittografi del mondo[31], è stata suggerita la seguente regola.

- R28: "*per avere sicurezza a breve termine occorre una chiave con 75 bit; per mantenere lo stesso livello di robustezza la lunghezza della chiave deve essere incrementata di 14 bit ogni vent'anni*".

Il continuo incremento della lunghezza della chiave è un aspetto caratteristico dei Cifrari a blocchi moderni. La pericolosità di particolari attacchi con testo noto o scelto (li citeremo più avanti) ha fatto parallelamente incrementare anche la dimensione dei blocchi.

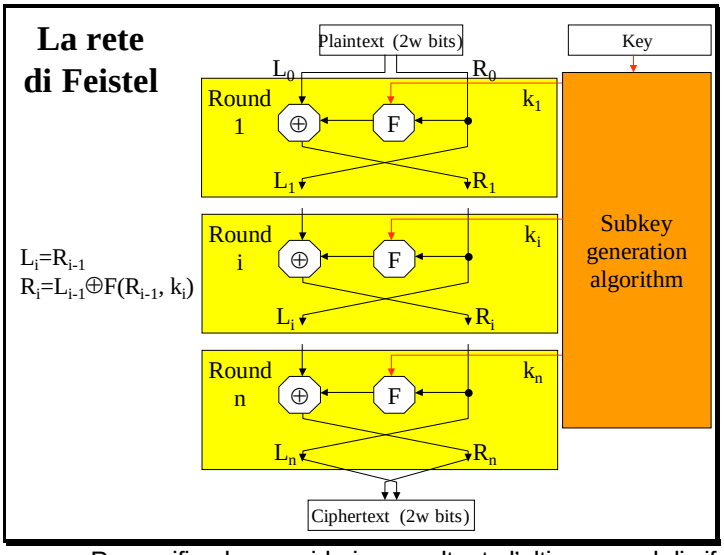
ESEMPI - Diversi standard di **Cifrari a blocchi** si sono succeduti nel tempo:
- **DES** (56 bit di chiave e 64 bit di blocco): ha avuto un larghissimo impiego negli anni '80 e '90;
- **TDES** (112 o 168 bit di chiave e 64 bit di blocco): versione più robusta del DES usata negli anni '90;
- **AES**-**Rijndael** (da 128 a 256 bit di chiave con blocchi di 128,192,256 bit): il vincitore della gara citata a pag. 22, per il quale è stata prevista una vita almeno trentennale.

Gli utenti devono inoltre rispettare le regole R12 (i bit della chiave devono essere generati a caso) e R24 (la chiave in uso deve essere frequentemente modificata, come peraltro aveva a suo tempo sostenuto Kerckhoffs).

4.3.2 Il modello di Feistel

Quasi tutti i Cifrari adottano, con qualche variante, il **modello di Feistel**[32], che a sua volta discende dal modello del Cifrario composto proposto da Shannon.

Ogni blocco di testo cifrato è ottenuto dal corrispondente blocco di testo in chiaro (nel seguito indicheremo con **2w** la loro dimensione in bit) iterando diverse volte l'esecuzione di una sostituzione e di una trasposizione.



La generica i-esima iterazione, detta ***round***, genera due vettori di **w** bit, (L_i, R_i) a partire dai risultati del round i-1-esimo (L_{i-1}, R_{i-1}):

$$\mathbf{L_i = R_{i-1}}$$
$$\mathbf{R_i = L_{i-1} \oplus F(R_{i-1}, k_i)}.$$

La proprietà di "confusione" è ottenuta con la sostituzione operata dalla **funzione non lineare F**. Ogni round ha una sua regola di sostituzione, stabilita di volta in volta da una **sottochiave k_i** che un apposito **generatore** calcola a partire dalla chiave **k** concordata dagli utenti. La proprietà di "diffusione" discende dal continuo scambio tra **L** e **R**. Si noti che in questo modello la trasposizione è fissa: in altre parole la P-box è priva di chiave.

Una proprietà notevole del modello è che la decifrazione si esegue con lo stesso algoritmo, invertendo l'ordine delle sottochiavi.

Per verificarla, consideriamo soltanto l'ultimo round di cifratura ed il primo di decifrazione.

In ingresso alla decifrazione si ha:

$$\mathbf{L_0 = R_n = L_{n-1} \oplus F(R_{n-1}, k_n)}$$
$$\mathbf{R_0 = L_n = R_{n-1}}.$$

In uscita dal primo round si ha:

$$\mathbf{L_1 = R_0 = R_{n-1}}$$
$$\mathbf{R_1 = L_0 \oplus F(R_0, k_n) = L_{n-1} \oplus F(R_{n-1}, k_n) \oplus F(R_{n-1}, k_n) = L_{n-1}}.$$

Arrivati a questo punto sono stati dunque annullati gli effetti dell'ultimo round di cifratura.

[31] M. Blaze, W. Diffie, R. Rivest, B. Schneier, T. Shimomura, E. Thompson, M. Wiener, " Minimal Key Length for Symmetric Ciphers to Provide Adequate Commercial Security", *Report,January 1996.*

[32] All'inizio degli anni '70 H. Feistel ha guidato il gruppo che ha progettato il Cifrario Lucifer per i calcolatori IBM-Serie 360 e ne ha poi descritto il modello sulla rivista *Scientific American* (maggio 1973).

4.3.3 Data Encryption Standard (DES)

Il capostipite dei moderni Cifrari a blocchi è stato il **DES**, voluto a metà degli anni '70 dal Governo Federale USA come *standard* per la riservatezza di dati "non classificati" e realizzato poi dalla IBM su invito del NIST.

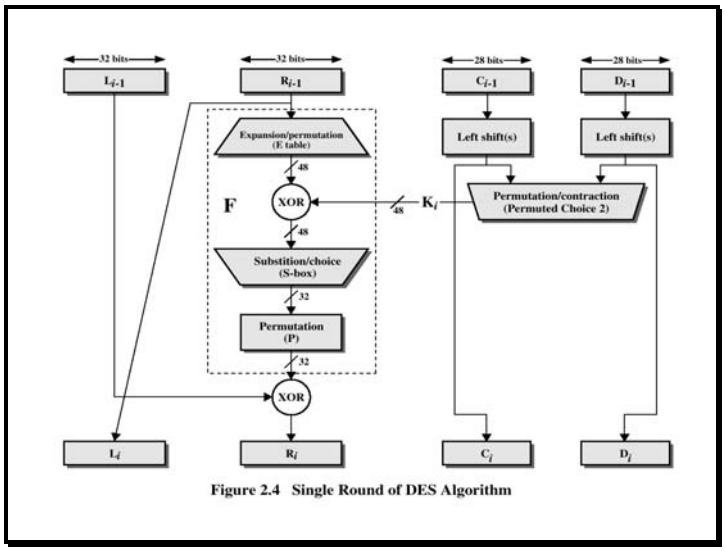


Figure 2.4 Single Round of DES Algorithm

I round sono 16, i blocchi contengono 64 bit e la chiave è di 64 bit, o meglio di 56 perché 8 sono di parità.

Si noti in figura[33] che la funzione F (uno dei punti più delicati della progettazione di tutti i Cifrari a blocchi) è stata ottenuta disponendo in cascata:

- una "espansione&permutazione" da 32 a 48 bit (effettuata da 8 **E-box**),
- la somma modulo due del risultato con altrettanti bit di sottochiave,
- una "sostituzione&scelta" (effettuata da 8 **S-box**) che fornisce un dato di 32 bit,
- una permutazione senza chiave (**P-box**) dei bit prima ottenuti.

Si noti anche la relativamente complessa modalità di generazione delle sottochiavi (un secondo punto delicato della progettazione).

Il DES è stato concepito per una realizzazione Hw (con un chip VLSI si è arrivati a cifrare 120 milioni di byte il secondo), ma ha poi trovato un impiego grandissimo con le più svariate realizzazioni Sw (le migliori cifrano 100 mila byte il secondo).

Il basso numero di bit di chiave (il predecessore Lucifer ne impiegava 128) e l'iniziale segretezza delle S-box avevano fatto sorgere il sospetto che il progetto contenesse punti di vulnerabilità dettati dal NIST. Dopo molti anni di polemica si arrivò alla conclusione che il sospetto era infondato.

A causa del limitato spazio delle chiavi, era stato comunque previsto di dare al DES non più di dieci anni di vita, anche se poi in realtà è stato usato per più di trent'anni.

4.3.4 Crittanalisi differenziale e lineare

Il DES è stato un'importante palestra per i crittanalisti, che hanno provato tutti i modi possibili per romperlo, senza però mai riuscirci. Da questi sforzi sono però discese due nuove e potenti tecniche di crittanalisi:

- la crittanalisi **differenziale**[34] (un attacco con testo in chiaro scelto alla sottochiave di round, in cui si cercano coppie di dati d'ingresso della funzione F la cui somma modulo due, o **differenza**, sia riscontrabile anche nelle corrispondenti coppie di dati d'uscita),
- la crittanalisi **lineare**[35] (un attacco con testo in chiaro noto, in cui si cerca di approssimare la trasformazione eseguita dal Cifrario con un'**equazione lineare**, da cui poi dedurre singoli bit di chiave).

Entrambe si sono dimostrate particolarmente efficaci, quando il numero di round del Cifrario a blocchi è basso e quando la sua funzione F non è ben progettata.

Nel caso del DES si è però visto che è necessario procurarsi $\mathbf{2^{47}}$ coppie di testo, attività di complessità più bassa di quella di un attacco con forza bruta ($\mathbf{2^{55}}$), ma ancora troppo alta per poterla realmente mettere in atto. Uno dei progettisti del DES ha recentemente dichiarato che fin dall'inizio avevano tenuto conto della pericolosità dell'attacco denominato, 15 anni dopo, crittanalisi differenziale.

Da tutti questi studi sono discesi alcuni principi empirici[36], che è molto opportuno rispettare nella progettazione di un Cifrario a blocchi. In questa sede ci limitiamo ad elencarli.

- **SAC** (***Strict Avalanche Criterion***) – Ogni bit d'uscita di una S-box, o più in generale di una funzione F, deve cambiare con probabilità 0,5, quando viene invertito il valore di uno qualsiasi dei bit d'ingresso.
- **BIC** (***Bit Independence Criterion***) – Due qualsiasi bit d'uscita di una funzione F devono modificarsi in modo apparentemente indipendente, quando è invertito il valore di uno qualsiasi dei bit d'ingresso.
- **GA** (***Guaranteed Avalanches***) – Ad ogni variazione di un bit d'ingresso di una S-box, da 2 a 5 bit d'uscita devono modificare il loro valore.

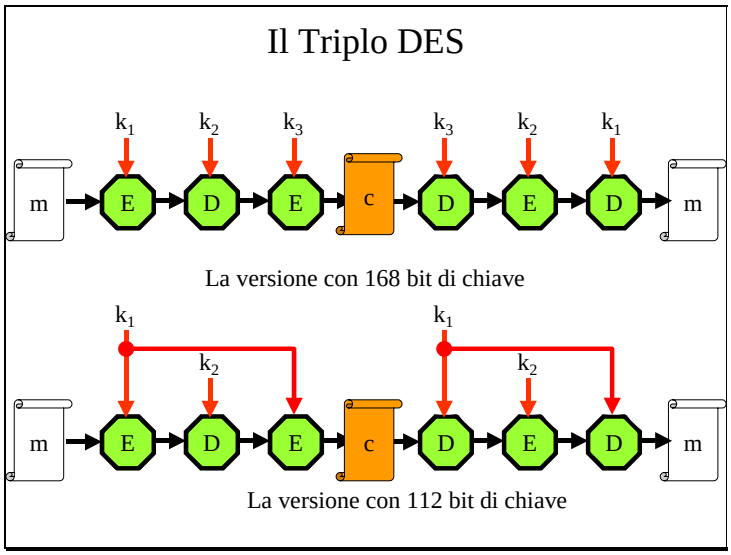
[33] da [6], pag.81
[34] E.Bhiam, A.Shamir "Differential cryptanalysis of DES-like cryptosystems" *Journal of Cryptology*, 4, 1991, pp. 3-72.
[35] M.Matsui "Linear Cryptanalysis Method for DES Chipher" Advances in Cryptology – EUROCRYPT '93 Proceedings, Springer-Verlag 1994.
[36] v. [6], pag. 93-94.

4.3.5 I successori del DES

Tutti i Cifrari sviluppati negli anni '90 (IDEA, Blowfish, CAST, ecc.) hanno impiegato chiavi di 128 o più bit e strutture più adatte ad una realizzazione Sw.

Le sostituzioni, che il DES affidava o a blocchi hardware o alla tecnica della *table look-up*, sono diventate sequenze di operazioni direttamente svolte dalla ALU di un calcolatore; le trasposizioni, che il DES otteneva scambiando l'ordine dei segnali, sono parallelamente diventate sequenze di rotazioni.

ESEMPIO – Il Cifrario **IDEA** (1992) prevede l'esecuzione di operazioni definite su tre differenti gruppi algebrici:
- la somma modulo 2 tra vettori di 16 bit,
- l'addizione di interi modulo 2^{16},
- la moltiplicazione di interi modulo $2^{16}+1$.



Chi ha voluto/dovuto mantenere in servizio il DES (tipicamente l'ambiente bancario) ha usato il Cifrario **TDEA** (o TriploDES, o EDE), che sottopone il testo in chiaro a tre successive trasformazioni DES: una cifratura, una decifrazione ed un'ulteriore cifratura.

Nella versione più robusta la sorgente e la destinazione concordano 168 bit di chiave, o meglio tre chiavi di 56 bit (k1, k2, k3).

Nella versione meno robusta k1 e k3 coincidono.

Alle due estremità del canale sono svolti i seguenti calcoli:

SORGENTE: $c = E_{k3}(D_{k2}(E_{k1}(m)))$

DESTINAZIONE: $m = D_{k3}(E_{k2}(D_{k1}(c)))$

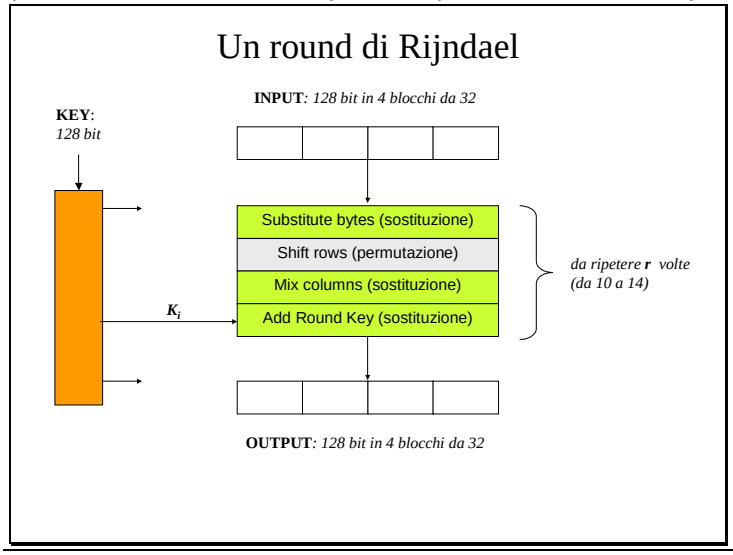
4.3.6 Advanced Encryption Standard (AES)

AES è lo standard voluto dal NIST per il terzo millennio (www.csrc.nist.gov/encryption).

Dopo una lunga e pubblica valutazione della robustezza, cinque dei sedici candidati sono stati giudicati ottimi ed ammessi alla fase finale della selezione[37]:

Alla fine è stato dichiarato vincitore unico (molti avrebbero preferito più vincitori) il Cifrario **Rijndael**.

Le lunghezze della chiave e del blocco partono dal valore minimo di 128 bit e possono essere poi incrementate per multipli di 32 bit. Tutte le trasformazioni previste sono operazioni definite su un **campo di Galois** (usualmente indicato con **GF**) e sono quindi ricondotte a semplici sequenze di somme modulo 2 e di scorrimenti.



La struttura[38], molto "pulita" ed innovativa rispetto al modello di Feistel (non c'è più la suddivisione del blocco in due parti uguali), è detta *square* ed è schematizzata in figura nel caso di blocchi di 128 bit.

Ogni round è formato da quattro successive trasformazioni, tre di sostituzione ed una di permutazione.

- Il blocco **Substitute bytes** è una S-box che opera byte per byte.
- Il blocco **Shift Row** esegue una permutazione.
- Il blocco **Mix columns** esegue operazioni aritmetiche su GF(2^8).
- Il blocco **Add Round Key** somma modulo due i dati d'ingresso e la chiave di round.

[37] MARS della IBM, RC6 di Rivest, Serpent di Biham, Twofish di Schneier e Rijndael di V. Rijmen e J. Daemen dell'Università di Leuven (www.esat.kuleuven.ac.be/~rijmen/rijndael).

[38] Si veda anche, scaricandola dal sito del corso, la bella presentazione fatta dallo studente uruguaiano Enrique Zabala.

Il Cifrario non ha presentato alcun punto debole ed ha dimostrato:

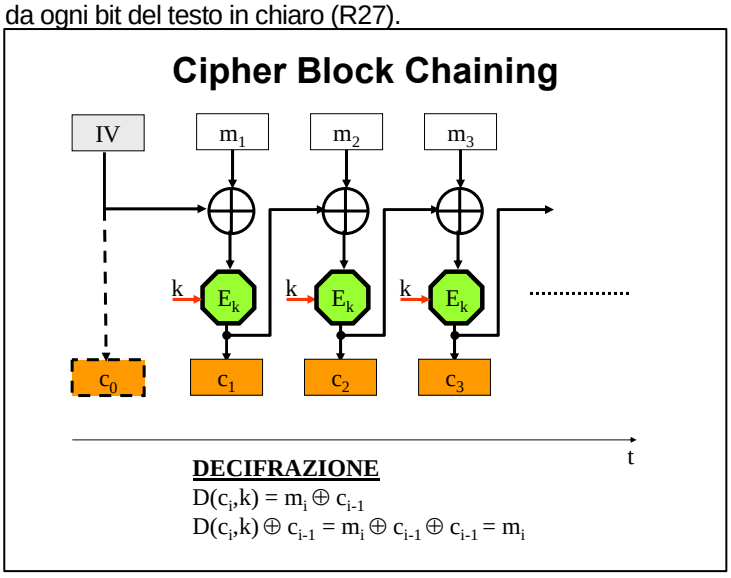
- facilità di implementazione su tutte le piattaforme (dai *main frame* alle *smart card*),
- ottime prestazioni (sui processori a 32 bit si è dimostrato molto più veloce di ogni altro cifrario a blocchi),
- alto grado di parallelismo nell'esecuzione delle trasformazioni di round,
- buon margine di sicurezza a fronte di ogni attacco conosciuto,
- bassa richiesta di memoria (ROM e RAM),
- veloce procedura di *key setup*.

4.3.7 Modalità di cifratura

La cifratura di un solo blocco di bit alla volta, evidenziata a pag. 67 e detta **modalità ECB** (*Electronic Code Book mode*), ha il pregio della **non propagazione degli errori**, ma ha anche una pericolosa vulnerabilità: "**blocchi identici di testo in chiaro producono blocchi identici di testo cifrato**".

ESEMPIO – Supponiamo che due banche A e B trasferiscano fondi con messaggi cifrati di formato fisso e che l'intruso sappia in quali blocchi è comunicato il n° del conto corrente di destinazione. Se l'intruso è in grado di svolgere attacchi passivi ed attivi può diventare ricco. Come prima cosa apre un conto presso A, uno presso B ed ordina ad A di trasferire un po' di danaro sul suo conto presso B. A questo punto intercetta il messaggio, prende nota della cifratura del suo n° di conto corrente e lo sostituisce poi a quello contenuto in tutti gli ordini di trasferimento emessi successivamente da A.

Per eliminare questo punto debole sono state previste le modalità **CBC** (*Cipher Block Chaining*), **CFB** (*Cipher Feedback*), **OFB** (*Output Feedback*) e **CTR** (*Counter*), che realizzano tutte trasformazioni variabili al progredire del testo e che rispettano quindi, se pur non perfettamente, la regola di far dipendere ogni bit del cifrato da ogni bit del testo in chiaro (R27).



Con **CBC** (la modalità più usata) ogni blocco di bit del testo in chiaro è dapprima sommato modulo 2 con il blocco di bit del testo cifrato precedente, poi cifrato a sua volta.

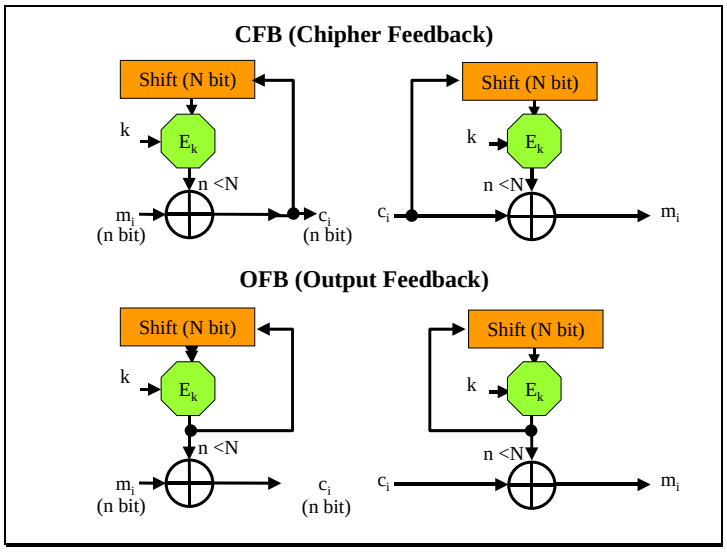
In questo modo **ogni blocco di testo cifrato viene a dipendere da tutti i precedenti blocchi in chiaro**.

Il primo blocco di testo in chiaro viene sommato modulo 2 con un valore casuale non segreto, detto vettore di inizializzazione **IV**, per impedire all'intruso di scoprire se il messaggio appena intercettato è uguale o ha la stessa intestazione di un messaggio precedente.

IV può essere o concordato in precedenza, o inviato come primo blocco del cifrato. In questo caso si ha:

Sorgente: $\mathbf{c_0 = IV}$, $\mathbf{c_i = E_k\,(c_{i-1} \oplus m_i)}$

Destinazione: $\mathbf{IV = c_0}$, $\mathbf{m_i = D_k\,(c_i) \oplus c_{i-1}}$



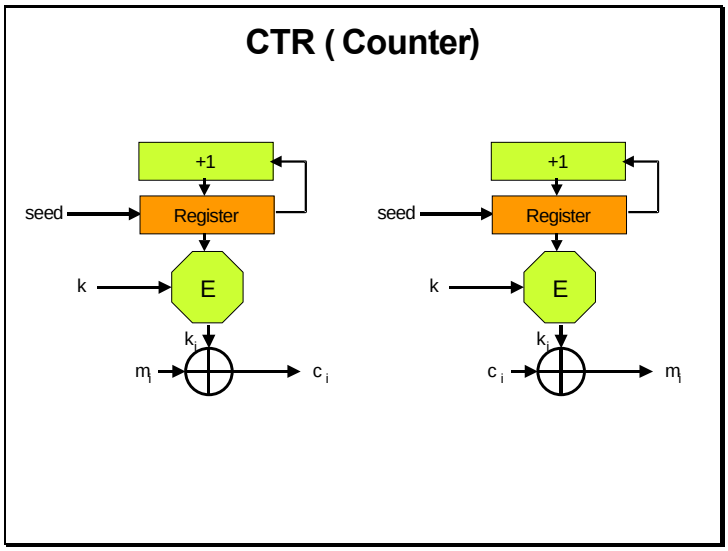
Le modalità **CFB** e **OFB** prevedono di cifrare e di decifrare **n** bit alla volta, con **n** notevolmente più piccolo della originaria lunghezza **N** del blocco.

Si noti in figura che in entrambi i casi, in ricezione, occorre impiegare ancora **E** e non **D**.

Per quanto detto in 4.2, **OFB** realizza un Cifrario a flusso sincrono, **CFB** un Cifrario a flusso con auto-sincronizzazione: il ruolo attribuito ad **E** ed al **registro a scorrimento** è dunque quello di un **generatore di flusso di chiave**.

Una volta inizializzati gli shift con lo stesso seed, in trasmissione ed in ricezione si ottengono flussi di chiave identici; il tutto è però non riproducibile dall'intruso, che non conosce la chiave **k**.

Per avere flussi di chiave sempre diversi, il mittente deve scegliere ogni volta a caso un vettore di inizializzazione dello shift (il seed) e comunicarlo poi al destinatario in testa al messaggio cifrato.



Anche la modalità **CTR**, già esaminata a pag. 31, impiega la sola trasformazione di cifratura per generare un flusso di L (N.B. L è la lunghezza del blocco del Cifrario) bit casuali da sommare modulo 2 con un pari numero di bit del testo in chiaro (lato sorgente) e cifrato (lato destinazione). Tali bit sono ottenuti cifrando lo stato di un contatore a L bit, incrementato di un'unità prima di procedere all'elaborazione di un nuovo blocco. Il comportamento è dunque quello di un Cifrario a flusso sincrono.

La segretezza della chiave di cifratura impedisce all'intruso di prevedere il dato di volta in volta fornito agli EX-OR.

Il seed, al solito trasmesso in chiaro in testa al messaggio cifrato, consente ai due corrispondenti di modificare il punto da cui iniziare il flusso di chiave.

4.4 Meccanismi per l'autenticazione di un documento

Si potrebbe pensare di usare un Cifrario simmetrico (a flusso o a blocchi) anche come meccanismo per autenticare l'origine di un documento informatico. La giustificazione è intuitiva: "**se si riesce ad ottenere un testo dotato di "significato" decifrando con una certa chiave un messaggio cifrato, allora si può confidare che tale testo sia stato inviato dal corrispondente con cui si condivide il segreto su quella chiave**".

Questa tecnica di autenticazione presenta aspetti problematici o dal punto di vista dell'efficienza, o da quello della sicurezza.

1. I documenti non riservati devono essere comunque prima cifrati e poi decifrati.
2. L'autenticazione di un documento non riservato da comunicare a più destinatari richiede la creazione e l'invio di altrettanti messaggi cifrati.
3. La condivisione della chiave segreta non consente di risalire in modo univoco a chi ha originato un documento.

ESEMPIO - A può "ripudiare" la paternità di un messaggio che ha realmente inviato sostenendo che B se lo è creato per conto suo. B può realmente forgiare un cifrato e sostenere di averlo ricevuto da A.

4. Un testo cifrato non più integro può generare un testo in chiaro ancora "significativo", se pur diverso dall'originario.

ESEMPI – Un messaggio cifrato con modalità ECB è decifrabile anche se un intruso replica alcuni blocchi, o li elimina, o li dispone in ordine diverso, o ne inserisce dei nuovi prelevandoli da precedenti messaggi cifrati con la stessa chiave. Un attacco attivo a OTP (v. pag. 57) può modificare il testo in chiaro in modo non rilevabile.

5. Non è detto che un messaggio debba necessariamente avere un "significato".

ESEMPIO – A può mandare a B una stringa casuale di bit da usare come chiave in messaggi successivi.

Gli aspetti enunciati nei punti 1, 2, 3 non sono critici

- se il documento è per sua natura riservato,
- se solo B deve poterlo mettere in chiaro,
- se A e B si fidano reciprocamente uno dell'altro.

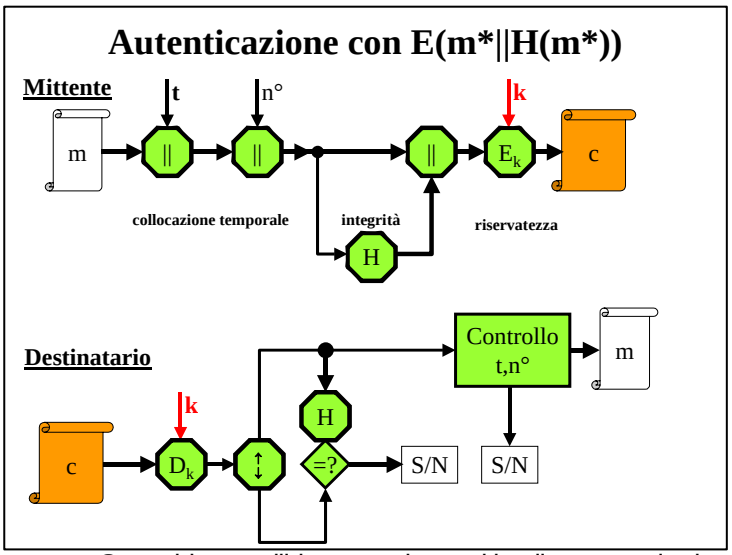
Nel prossimo paragrafo studieremo questo contesto, rimandando al successivo il problema dell'autenticazione di documenti non riservati; nel paragrafo 4.4.3 ci occuperemo infine degli utenti che non vogliono, o non possono, fidarsi uno dell'altro.

4.4.1 Integrità ed origine di un testo cifrato

Per la difesa dell'integrità dei messaggi riservati il destinatario B, o ancor meglio la macchina usata da B, deve poter verificare se il testo cifrato ricevuto ha subito o no alterazioni durante la trasmissione.

La semplice soluzione delineata in 2.1.3 non è sufficiente, dato che non protegge da **attacchi con replica**. A tal fine bisogna aggiungere ad ogni messaggio informazioni puntuali sul suo contesto: tipicamente l'istante **t** (*time stamp*) in cui è stato inviato, la sua posizione **n°** nel flusso di comunicazioni tra i due corrispondenti, ma possono essere utili anche altri dati, come l'**identificativo** del mittente, la **versione** del protocollo, il **formato** dei campi del messaggio composto, ecc..

L'incremento della sicurezza si ottiene anche in questo caso a spese dell'efficienza: lo spazio dei messaggi da inviare è, infatti, **più piccolo** dello spazio dei messaggi che vengono inviati.



La sorgente A deve dunque prima attribuire a **m** una particolare **struttura dati**, ad es. **m*=m||t||n°**, ed affiancarla poi con l'etichetta **H(m*)** che ne attesta l'integrità; non è necessario che H sia una funzione sicura.

La marca temporale **t** è generata dalla macchina di A, a cui spetta anche il compito di aggiornare **n°** ad ogni trasmissione.

La destinazione B deve ovviamente conoscere la struttura di ogni messaggio che riceve e può avvalersi di questa ridondanza per verificare se ciò che ha decifrato è integro e "fresco":

- calcola **H(m*)** e lo confronta;
- aggiorna il suo **n°** e lo confronta;
- confronta **t** con l'istante di ricezione e valuta se è ammissibile, mettendo in conto il ritardo della rete ed il disallineamento degli orologi.

Se tutti i controlli hanno esito positivo il messaggio ricevuto è considerato autentico: l'intruso, per ipotesi, non conosce la chiave segreta e non può quindi essere l'autore del messaggio strutturato che la destinazione ha ottenuto con la decifrazione.

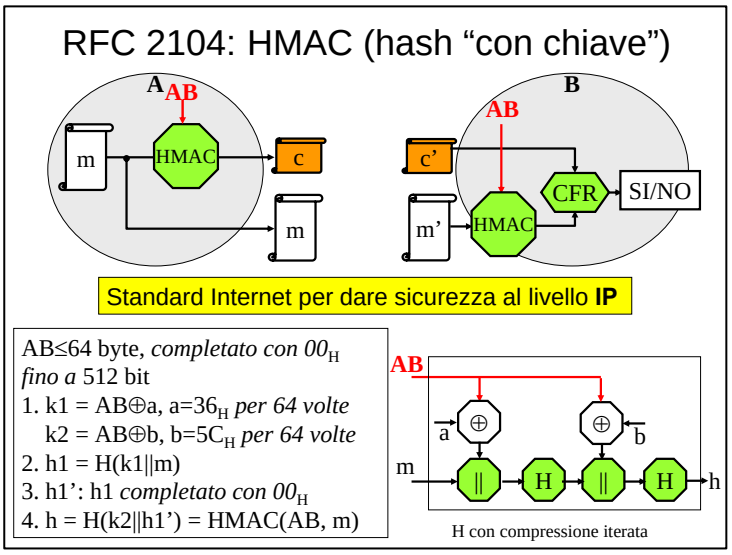
4.4.2 Integrità ed origine di un testo in chiaro

L'aggiunta di un riassunto va bene anche quando occorre autenticare un documento in chiaro, a patto però di tener conto che l'intruso può svolgere attacchi attivi su entrambe le componenti del messaggio.

La Crittografia simmetrica ha individuato molte soluzioni; parallelamente la Crittanalisi ha mostrato l'esistenza di vulnerabilità. In questa sede ci limitiamo a discutere le due soluzioni più usate: HMAC e MAC.

HMAC (*Hash Message Authentication Code*)

Lo standard Internet per l'autenticazione è HMAC (v. 2.1.3). Un pregio dello standard è che gli utenti possono scegliere l'algoritmo di hash che ritengono più sicuro. In figura è richiamato il principio di funzionamento.



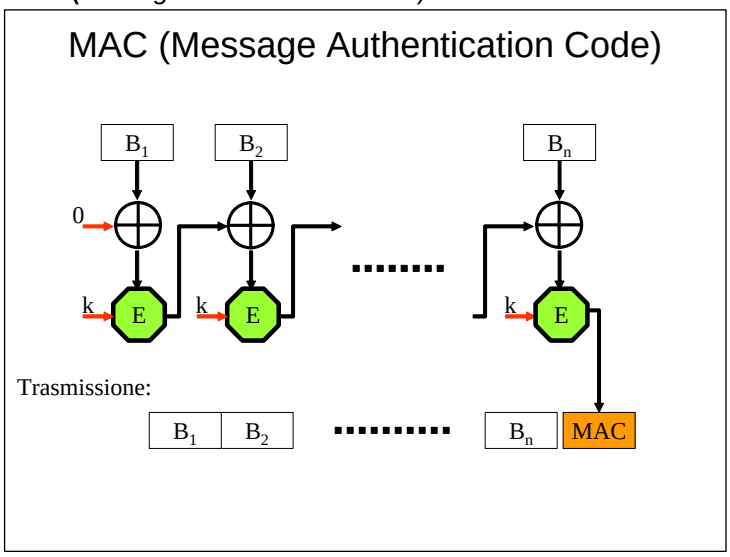
A e B concordano un **dato segreto s** di non più di 64 bit e si accordano sull'uso di una certa **funzione hash sicura**.

A trasmette il messaggio (v. figura)
m || HMAC(m, s).

B, conformemente a quanto previsto dallo standard, suddivide il messaggio che riceve nelle due parti **m'** e **HMAC'(m, s)**. Calcola poi **HMAC(m', s)** e confronta il risultato con **HMAC'(m, s)**. Il messaggio m' è considerato autentico solo nel caso di eguaglianza.

La sicurezza è connessa alla **non invertibilità** della funzione hash (R21): l'intruso che conosce **m** e **H** non può dunque risalire in alcun modo al segreto **s**. L'impossibilità di individuare collisioni (R19, R20) garantisce inoltre che non potrà utilizzare **H(m||s)** con un **m' ≠ m**.

MAC (*Message Authentication Code*)



Una seconda soluzione si basa sull'impiego di un Cifrario a blocchi.

Il mittente **struttura** il testo in chiaro (v. 4.4.1) e lo sottopone ad una **cifratura a blocchi con modalità CBC** (IV = 0) usando la chiave segreta **k** che condivide con la destinazione. Il mittente trattiene solo l'ultimo blocco di testo cifrato, il **MAC**, e lo trasmette insieme al messaggio in chiaro.

Si noti la similitudine tra questo modo di operare ed il principio della compressione iterata impiegato dagli algoritmi di hash.

La destinazione esegue lo stesso calcolo sul messaggio ricevuto e confronta il risultato con il MAC inviatole dalla sorgente: se coincidono il messaggio è integro e la sua origine non può che essere la persona con cui il destinatario ha concordato la chiave segreta.

L'attacco attivo è difficile: l'intruso non ha problemi a modificare il testo in chiaro, ma non riesce poi, non conoscendo la chiave, a adattarlo al MAC originario o ad affiancargli un MAC in grado di superare la verifica cui sarà sottoposto. La sicurezza è tanto più alta quanto maggiore è la dimensione del MAC.

ESEMPIO – Nell'ambiente bancario è stato impiegato il **DES** (standard ANSI X9.17), ma i suoi 64 bit di blocco sono oggi ritenuti insufficienti. Attualmente si usa **AES** che genera autenticatori o di 128, o di 256 bit

4.4.3 Integrità, origine e non ripudio di un testo in chiaro

Esistono documenti informatici che tutti devono poter inequivocabilmente attribuire ad una persona ben precisa; l'autore non deve inoltre poterli ripudiare e nessuno, neppure l'autore, deve riuscire a spacciare per buone modifiche che ha apportato successivamente.

Al di fuori dell'ambiente telematico la soluzione del problema è ben nota: chi vuole dimostrare o di essere l'autore di un documento o di concordare con il suo contenuto, deve firmarlo e consegnare poi l'originale alla controparte o ad una terza parte fidata.

Firma digitale

La firma digitale di un documento informatico deve:

1- consentire a **chiunque** di identificare **univocamente** il firmatario,

2- non poter essere **imitata** da un impostore,

3- non poter essere **trasportata** da un documento ad un altro,

4- non poter essere **ripudiata** dall'autore,

5- rendere **inalterabile** il documento in cui è stata apposta.

Il paradigma della "firma autografa", una volta trasportato nel mondo telematico, è detto **firma digitale di un messaggio**.

Per avere una sicurezza paragonabile, o meglio superiore, alla firma autografa, gli algoritmi di firma digitale devono presentare le cinque proprietà indicate in figura.

Nel prossimo capitolo vedremo che la Crittografia a chiave pubblica è in grado di garantire il loro conseguimento in modo semplice e sicuro.

Anche la Crittografia a chiave segreta può farlo, a patto, però, di adottare accorgimenti abbastanza complessi che le consentano di superare il limite del contesto "uno a uno", proprio dei procedimenti di autenticazione basati sulla condivisione di un segreto.

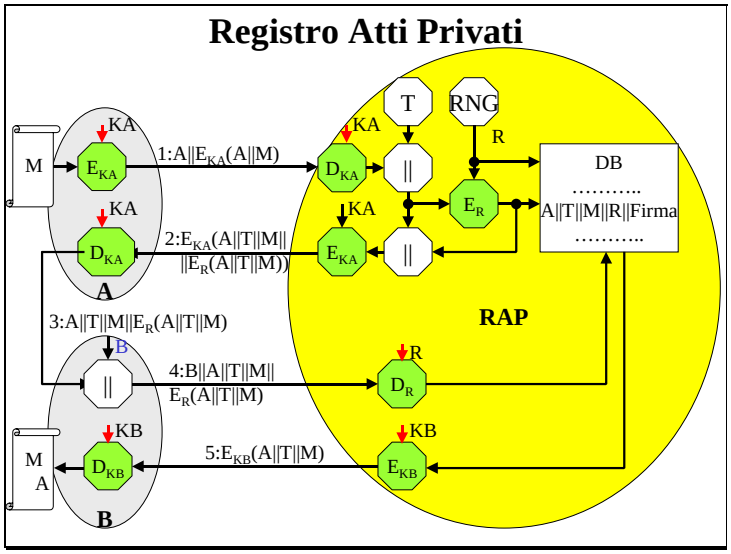
Per ottenere un contesto "uno a molti" sono necessari un'**Autorità** e diversi prerequisiti:

- tutti devono fidarsi dell'Autorità;
- l'Autorità deve poter autenticare ogni messaggio che riceve;
- tutti devono poter autenticare i messaggi che provengono dall'Autorità.

Per consentire l'autenticazione dei messaggi, tutti gli interessati devono avere dunque preliminarmente e segretamente concordato una chiave con l'Autorità.

Nel seguito indicheremo con **RAP** l'Autorità (essendo il suo modo di operare molto simile a quello di un **Registro di Atti Privati**), con **A** l'autore di un documento **M**, con **B** l'utente interessato a verificarne l'autenticità e con **KA**, **KB** le chiavi che **A** e **B** hanno concordato con **RAP**.

A, **B** e **RAP** devono eseguire il seguente protocollo.



1 - **A** cifra **M** con **KA** e lo invia a **RAP** con alcune informazioni in chiaro (come minimo la sua identità). **A** "deposita" così il suo atto; l'uso della chiave **KA** consente a **RAP** di identificare **A**, di rimettere in chiaro **M** e di formare la stringa **A||T||M**, ove **T** è la data/ora di ricezione.
2 - **RAP** sceglie a caso un numero **R** e lo impiega come chiave per costruire il messaggio cifrato $\mathbf{E_R(A||T||M)}$, che registra nel suo **DB** ed invia ad **A** come "ricevuta" del deposito di **M**.
3 - **A** invia a **B** il documento **M** ed una copia della sua ricevuta, fermo restando il fatto che né lui, né il destinatario, potranno decifrarla, poiché la chiave **R** la conosce solo **RAP**.
4 - **B** chiede a **RAP** di decifrare la ricevuta.
5 - **RAP** mette in chiaro la ricevuta ed invia a **B** $\mathbf{E_{KB}(A||T||M)}$. L'uso di **KB** garantisce a **B** che la dichiarazione di autenticità proviene da **RAP**.

Tutti possono dunque convincersi dell'autenticità di un documento in chiaro e nessuno può negare di averne depositato una copia presso l'Autorità (tale informazione è quindi **non ripudiabile**). La soluzione ha però pesanti implicazioni. L'Autorità

- deve essere sempre *on-line*,
- non deve costituire un *collo di bottiglia* per il traffico tra gli utenti,
- non deve creare documenti falsi,
- deve custodire le chiavi degli utenti in una memoria *a prova di intrusione*.

4.5 Gestione delle chiavi

L'uso di un Cifrario simmetrico presuppone

- che ogni utente, tramite un canale sicuro, stabilisca una chiave con chiunque voglia corrispondere con lui,
- che ogni coppia di corrispondenti ripeta periodicamente l'accordo per modificare la chiave in uso.

Nel periodo della Crittografia classica l'**accordo sulla chiave** (oggi spesso indicato con la sigla **KA** da *Key Agreement*) si è basato su due modalità abbastanza pesanti: l'**incontro diretto** e l'impiego di un **corriere fidato**.

La **comunicazione in rete** è la terza e ben più efficiente modalità introdotta dalla Crittografia moderna.

La Società dell'informazione presuppone che tutti possano comunicare con tutti in modo riservato. Il conseguimento di tale obiettivo richiede di risolvere il problema della **gestione delle chiavi** massimizzando l'efficienza e la sicurezza. Ciò è avvenuto in tre tappe, che è didatticamente utile ripercorrere nell'ordine:

1. la **chiave che cifra chiavi**, un accorgimento che consente a due corrispondenti di comunicarsi in modo riservato un nuovo valore di chiave;
2. il **centro di distribuzione delle chiavi**, un sistema che consente di assegnare un'unica chiave segreta ad ogni utente di una Comunità in cui tutti devono poter comunicare con tutti in modo riservato;
3. lo **scambio di Diffie-Hellman**: la tecnica asimmetrica che per prima ha consentito di concordare una chiave segreta su un canale insicuro senza presupporre l'esistenza di precedenti accordi tra i due corrispondenti.

4.5.1 La chiave che cifra chiavi

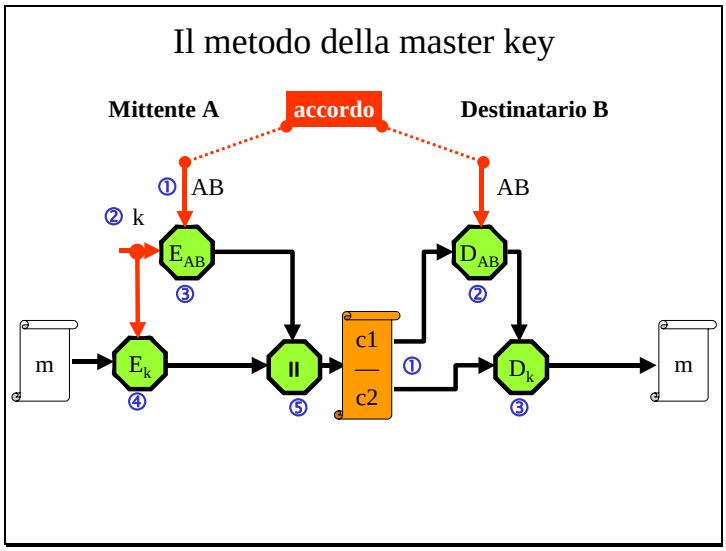
Per modificare la chiave in uso, uno dei due corrispondenti potrebbe impiegarla un'ultima volta per comunicare all'altro, in modo riservato, un nuovo valore. Il motivo per il quale si cambia frequentemente la chiave è il timore che un suo uso prolungato abbia consentito all'intruso di identificarla: se questo si è verificato la nuova chiave nasce già compromessa.

Un metodo più sicuro è accordarsi su due differenti chiavi:

- la **master key**, che gli utenti impiegheranno solo per cifrare nuove chiavi,
- la vera e propria chiave di cifratura/decifrazione dei messaggi che i due intendono scambiarsi.

La master key cifra pochi bit alla volta e può avere una vita più lunga di quella attribuita alla chiave dei messaggi. Ogni tanto va comunque rinnovata, con un incontro o con un corriere.

Il modo più sicuro per impiegare la chiave dei messaggi è quello di one-time pad: **usarla una volta sola**.



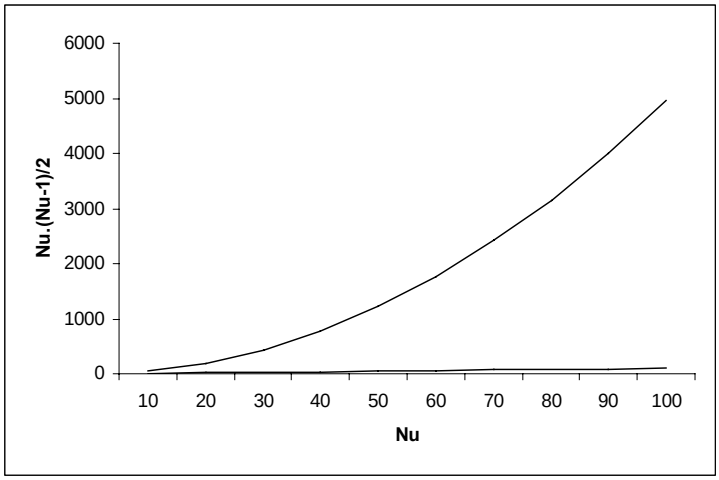
In figura è mostrato cosa devono fare due utenti **A** e **B**, che condividono la master key **AB**, per comunicarsi messaggi cifrati con una sempre diversa **chiave di sessione k** (*session key* o *one-time key*):

1. **A:** sceglie a caso la chiave **k**,
2. **A:** calcola **c1 = E_{AB}(k)**,
3. **A:** calcola **c2 = E_k(m)**,
4. **A:** trasmette a **B** il testo cifrato **c = c1 ll c2**.

1. **B:** separa **c1** e **c2**,
2. **B:** calcola **k = D_{AB}(c1)**,
3. **B:** calcola **m = D_k(c2)**.

ESEMPIO – Nel **file system sicuro** (v.pag. 27), **s** è la master key (conservata in forma cifrata nel portachiavi) che l'utente impiega per cifrare/decifrare la session key **k** (alloggiata, già cifrata, nell'intestazione del testo cifrato di ogni file).

4.5.2 Il Centro di distribuzione delle chiavi



Si è detto in precedenza che nella Società dell'informazione è impossibile che ogni utente stabilisca una chiave segreta con ognuno degli altri. Vediamo di convincerci con un semplice calcolo, ipotizzando una Comunità formata da **Nu** utenti che si accordano sulla chiave con incontri diretti.

Ciascuno di loro deve

- fare **Nu-1** incontri riservati,
- memorizzare **Nu-1** chiavi segrete,
- darsi da fare ogniqualvolta nella comunità si aggiunge un nuovo utente.

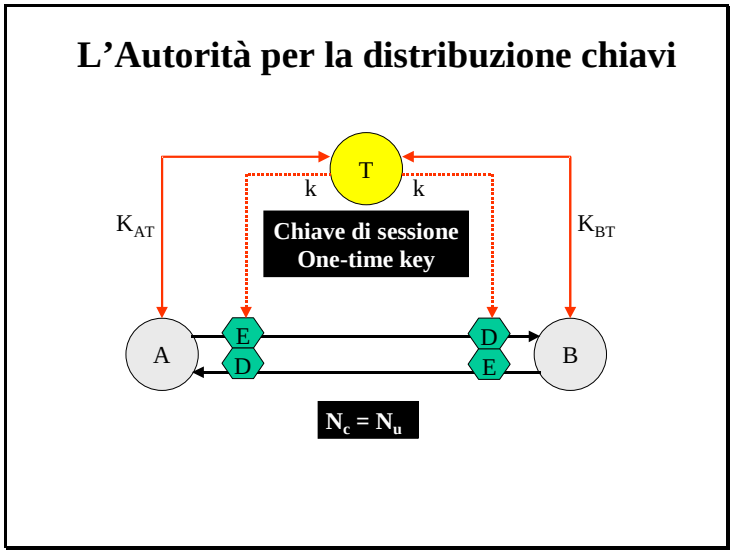
Occorrono dunque

Nc = Nu (Nu-1)/2

canali "sicuri"; altrettante sono le chiavi segrete in circolazione.

Si noti nella precedente figura che **Nc** cresce molto più rapidamente di **Nu**.

ESEMPIO - Con **Nu = 10^3** si ha già **Nc = 5.10^5**; ogni utente deve inoltre memorizzare **999** chiavi segrete.



L'accorgimento di far incontrare "tutti con tutti", ammissibile in piccole Comunità chiuse, diventa ingestibile al crescere di **Nu** (in gergo si dice che **non** è **scalabile**).

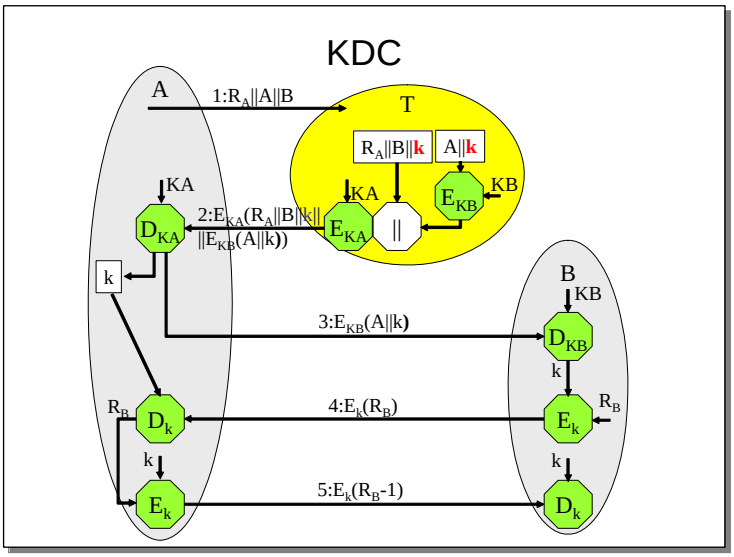
L'obiettivo da conseguire è **Nc = Nu**: ogni utente deve avere un'unica chiave segreta e deve usare di conseguenza un solo canale sicuro per concordarla.

La Crittografia simmetrica ha risolto il problema prevedendo un'**Autorità fidata**, detta **KDC** da *Key Distribution Center*, con cui tutti gli utenti della Comunità concordano individualmente una chiave segreta.

A tale Autorità compete sia la generazione di una chiave di sessione per ogni coppia di utenti che intendono comunicare, sia il suo invio in modo riservato agli interessati.

Nella figura sottostante è illustrato il protocollo[39] che, per primo, è riuscito a conseguire i precedenti obiettivi, garantendo anche ad ogni parte in causa il riconoscimento della controparte tramite "sfide" e "risposte".

Indichiamo con **KA** e **KB** le chiavi che **A** e **B** hanno concordato con l'Ente "fidato" **T**.



1 - **A** trasmette in chiaro a **T** la sua intenzione di comunicare con **B** ed aggiunge un numero a caso $\mathbf{R_A}$ per poterne autenticare la risposta.
2 - **T** sceglie a caso la chiave di sessione **k**, la concatena con $\mathbf{R_A}$**||B** (per garantire ad **A** che si sta riferendo alla sua precedente richiesta), aggiunge il messaggio $\mathbf{E_{KB}(A||k)}$ (che solo B potrà mettere in chiaro) e cifra il tutto con **KA** per permettere solo ad **A** di leggerlo e per farsi identificare.
3 - **A** decifra, memorizza **k** (che gli servirà successivamente) ed invia a **B** la parte che **T** ha predisposto per lui. **B** lo mette in chiaro con la sua chiave **KB**: il fatto che sia stato cifrato con KB gli garantisce che **A** è stato identificato da **T** e che **T** ha stabilito **k**.
4,5 - **B** ed **A** si identificano reciprocamente e dimostrano di aver già installato la chiave **k**.

E' stato osservato che questo protocollo presenta una possibile via di attacco. Un intruso **I** che è riuscito ad intercettare il messaggio $\mathbf{E_{KB}(A||k)}$, può ripresentarlo a **B** dopo un po' di tempo, iniziando praticamente il protocollo dal passo 3. Se **I** è riuscito a decifrare **k** ed è in grado di svolgere un attacco attivo completo sul canale che connette **A** e **B**, può impersonare **A** nel passo 5 e catturare ogni informazione inviata successivamente.

Sono state però anche individuate due possibili contromisure:

- **B** memorizza tutte le chiavi che ha ricevuto ed evita di riusarle,
- **KDC** attribuisce un tempo di vita limitato ad ogni chiave di sessione.

Il modello del **KDC**, pur ponendo tutti i problemi di efficienza e di sicurezza che abbiamo già citato per RAP (v. pag. 73), ha una sua indiscutibile validità, quando il numero degli utenti è relativamente limitato. Molti sono i progetti che lo hanno assunto come riferimento (il prototipo **KryptoKnight**, il servizio di autenticazione **Kerberos** realizzato dal MIT, lo standard **DCE** per ambienti distribuiti e, ultimamente, **Active Directory** di Windows 2000).

4.5.3 Lo scambio di Diffie-Hellman

Si è già detto più volte che nella Società dell'informazione chiunque deve poter instaurare comunicazioni riservate con chiunque altro: molto spesso, però, i due che intendono corrispondere non si sono mai incontrati prima e non sanno neppure in quale parte del Globo vive il loro interlocutore.

Incontri diretti e corrieri sono da escludere a priori. Una rete mondiale di KDC è improponibile. L'accordo sulla chiave di sessione deve dunque poter essere preso in assenza di ogni altro precedente accordo e l'unico modo per prenderlo è di farlo tramite Internet, tenendo però presente che tutte le comunicazioni sono intercettabili.

Il grande merito di Diffie ed Hellman è quello di aver individuato non solo un metodo di grande efficacia e sicurezza, lo **scambio** detto appunto **DH** dai loro cognomi, ma anche una nuova direzione su cui avviare gli studi e le applicazioni della Crittografia[40]. Tale direzione è stata immediatamente seguita da tutto il modo scientifico ed ha oggi un nome: **crittografia a chiave pubblica**.

L'obiettivo dello scambio DH è far si che due utenti qualsiasi **A** e **B**, senza aver preso alcun precedente accordo segreto, riescano a condividere un dato segreto **K** dopo aver calcolato ed essersi scambiati senza alcuna segretezza due dati $\mathbf{Y_A}$ e $\mathbf{Y_B}$.

A tal fine è necessario che ciascuno dei due utenti scelga **a caso** un numero **X** (che terrà segreto) ed usi poi una funzione unidirezionale **F** per calcolare il numero **Y = F(X)** da inviare al corrispondente: in questo modo, infatti, l'intruso che riesce ad intercettare **Y** non dispone di algoritmi efficienti per calcolare $\mathbf{X = F^{-1}(Y)}$.

Una volta avvenuto lo scambio, il metodo prevede che **A** e **B** dispongano di una particolare funzione **G** che garantisca ad entrambi di ottenere lo stesso risultato **K** a partire dai dati in loro possesso:

$$\mathbf{G(X_A,Y_B) = G(X_B,Y_A) = K}.$$

[39] R.M. Needham, M.D. Schroeder, "Using Encryption for Authenticaton in Large Networks of Computers", *Communications of the ACM*, v.21, n.12, Dec. 1978, pp.993-999. $\mathbf{Z^*_n}$

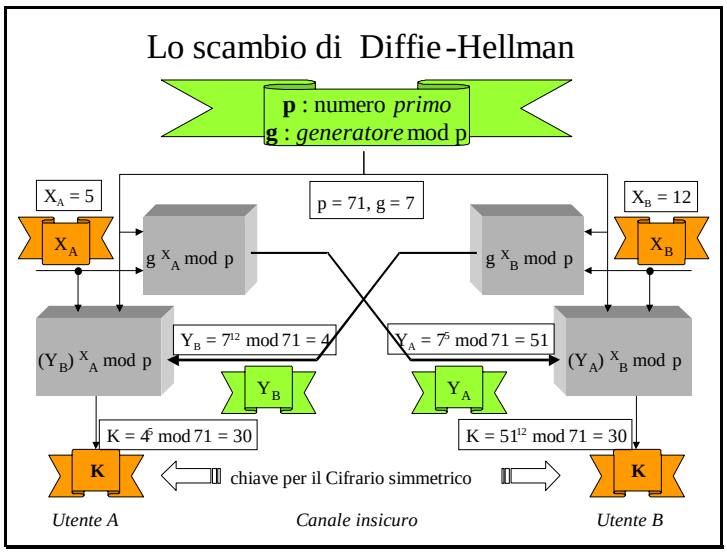
[40] W.Diffie, M.E. Hellman " *New Directions in Cryptography*", IEEE Trans. on Information Theory,v.IT-22,n.6, Jun 1977, pp.74-78. Un indipendente e contemporaneo contributo è stato dato da R.C. Merkle "*Secure Communications over Insecure Channels*" Comm. ACM, v.21,n.4,1978, pp.294-299. Di recente è emerso che i Servizi Segreti inglesi avevano individuato diversi anni prima le tecniche d'uso delle chiavi asimmetriche.

Diffie ed Hellman hanno suggerito di usare l'esponenziazione modulare per entrambe le funzioni:

$$Y = g^x \bmod p \qquad K = Y^x \bmod p$$

ove **p** è un **numero primo** e **g<p** un numero il cui elevamento a potenza (mod p) per **x=1,2,..,p-1** fornisce una permutazione di **Z*p** (nel seguito diremo più propriamente che **g** è un **generatore del campo di Galois GF(p)**).

Il protocollo prevede due passi per entrambi i corrispondenti. I dati **p** e **g**, non segreti, devono essere noti ad entrambi (ad esempio possono essere stabiliti da un'Autorità, oppure chi inizia il protocollo può comunicarli al corrispondente insieme al suo dato Y).



1: Generazione delle chiavi segrete X_A, X_B e pubbliche Y_A, Y_B – A e B, dopo aver generato a caso rispettivamente i numeri

$$1 < X_A < p-1$$
$$1 < X_B < p-1$$

(che tengono segreti), calcolano

$$Y_A = g^{X_A} \bmod p$$
$$Y_B = g^{X_B} \bmod p$$

e si scambiano i risultati in modo non riservato.

2: Calcolo del segreto K - A e B calcolano rispettivamente:

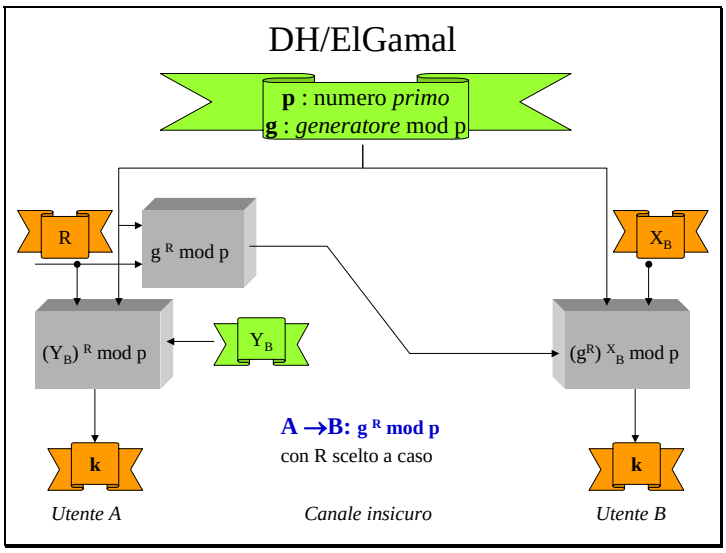
$$K_A = Y_B^{X_A} \bmod p = (g^{X_B})^{X_A} \bmod p$$
$$K_B = Y_A^{X_B} \bmod p = (g^{X_A})^{X_B} \bmod p$$

Per una proprietà dell'aritmetica modulare si ha:

$$K = K_A = K_B.$$

In figura la proprietà è stata verificata in un caso molto semplice.

Se **K** è un numero molto grande, come vedremo deve essere, **A** e **B** ne prendono una parte (ad es. i 128 bit più pesanti) e la usano come chiave segreta per cifrare i loro successivi messaggi riservati.



Questa originaria versione dell'algoritmo **DH** è detta **anonima**: ciascuno dei due utenti riceve, infatti, un numero cui non è associata alcuna indicazione sicura su chi l'ha inviato. Una variante, proposta da ElGamal ed adottata dal PGP, prevede che **A**, l'iniziatore del protocollo, abbia già a disposizione Y_B, ottenuto in precedenza ed in modo sicuro dal corrispondente **B**. In questo caso è sufficiente il solo messaggio

A → B: $Y_A = g^R \bmod p$ (con **R** scelto a caso)

per consentire ad entrambi di condividere un segreto *one-time* ($K = g^{R\,X_B} \bmod p$).

A è inoltre sicuro di averlo in comune proprio con **B**; **B** può solo …sperarlo!

Vedremo più avanti che due ulteriori varianti, dette rispettivamente **fixed DH** ed **ephemeral DH**, consentono a tutti e due i partecipanti di essere sicuri sull'origine della chiave pubblica ricevuta.

La segretezza di **K** si basa sull'assunzione che la funzione $y = g^x \bmod p$ sia **unidirezionale**. Per il calcolo dell'esponenziazione esistono algoritmi polinomiali (li esamineremo a pag.80). Per il calcolo della funzione inversa, detta estrazione del **logaritmo discreto** ed espressa da $x = \log_g y \bmod p$, si conoscono **solo algoritmi sub-esponenziali** (li esamineremo a pag.82).

ESEMPIO - Dato un primo **p** formato da 150 cifre decimali (circa 500 bit), un calcolatore in grado di eseguire 10^6 moltiplicazioni/sec calcola un'esponenziazione in circa 0,5 msec.
Un calcolatore 10^6 volte più veloce calcola un logaritmo discreto in 10^{56} anni!

L'estrazione del logaritmo discreto è uno dei presunti problemi **difficili** di cui si è parlato nel cap. 2. Vediamone l'enunciato.

- **P1: problema del logaritmo discreto su un campo di Galois -** Dato un primo **p,** un generatore **g** ed un intero $c \in Z^*_p$, trovare l'intero **x,** $1 \le x \le p-1$, tale che $g^x \bmod p = c$, o anche $g^x \equiv c \pmod p$.

4.6 Campo GF(p)

L'**algebra astratta** introduce cinque diverse **strutture**, a crescente complessità, per operare su insiemi dotati di un numero finito di elementi.

Gruppo finito	Insieme con un numero finito di elementi su cui è definita un'operazione **chiusa**, **associativa**, con **elemento identità** e con **elemento inverso**
Gruppo abeliano	Gruppo finito in cui l'operazione di gruppo è anche **commutativa**
Gruppo ciclico	Gruppo abeliano in cui ogni elemento è una **potenza con esponente intero** (cioè il risultato di un certo numero di applicazioni dell'operazione) di un elemento fisso del gruppo stesso, detto **generatore**
Anello	Insieme con un numero finito di elementi su cui sono definite **due** operazioni, una "somma" **chiusa**, **associativa**, con **elemento identità**, con **elemento inverso** ed una "moltiplicazione" **chiusa**, **associativa**, **commutativa**, **distributiva** rispetto alla somma, con **elemento identità**
Campo finito	Anello in cui ogni elemento $\neq 0$ ha il suo **inverso moltiplicativo**, o reciproco

Un **campo finito (o di Galois)** è definito o sull'insieme $\mathbf{Z_p}$ (con p primo), o sull'insieme $\mathbf{Z_q}$ (con $q = p^n$, p primo e n intero). Nel primo caso è indicato dalla notazione **GF(p)**, nel secondo da **GF(p^n)**.

In **GF(p)**, la struttura algebrica a cui vogliamo qui fare riferimento, gli elementi sono dunque gli interi minori di **p** (0, 1, ...p-1); le operazioni sono l'**addizione** e la **moltiplicazione** dell'**Aritmetica modulare**.

L'**addizione** di due qualsiasi numeri interi a,b **modulo** un intero **m**, primo o composto, è indicata dall'espressione (**a+b) mod m** e gode della proprietà notevole:

$$\mathbf{(a+b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod m}$$

Per il calcolo è conveniente ricondursi prima a due numeri congruenti con quelli assegnati, sommarli poi con le usuali regole dell'aritmetica e sottrarre infine **m** se il risultato è maggiore o uguale a **m**.

La complessità computazionale di questo algoritmo (v. pag. 20) è espressa da **O(log p)**.

La **moltiplicazione** di due qualsiasi numeri interi a,b **modulo** un intero **m**, primo o composto, è indicata dall'espressione **(a × b) mod m** e gode della proprietà notevole:

$$\mathbf{(a \times b) \bmod m = ((a \bmod m) \times (b \bmod m)) \bmod m}$$

Per il calcolo è conveniente ricondursi prima a due numeri congruenti con quelli assegnati, moltiplicarli poi con le usuali regole dell'aritmetica e dividere il risultato per **m** trattenendo il solo resto.

La complessità computazionale di questo algoritmo (v. pag. 20) è espressa da **O($(\log m)^2$)**.

4.6.1 Addizione in GF(p): a+b mod p

Per fissare le idee, tabuliamo il caso semplice di **p = 11** e di addendi interi già minori di p: **$0 \le a, b < 11$**.

	b										
a	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	0	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10	0
2	2	3	4	5	6	7	8	9	10	0	1
3	3	4	5	6	7	8	9	10	0	1	2
4	4	5	6	7	8	9	10	0	1	2	3
5	5	6	7	8	9	10	0	1	2	3	4
6	6	7	8	9	10	0	1	2	3	4	5
7	7	8	9	10	0	1	2	3	4	5	6
8	8	9	10	0	1	2	3	4	5	6	7
9	9	10	0	1	2	3	4	5	6	7	8
10	10	0	1	2	3	4	5	6	7	8	9

(a + b) mod 11

- Tutti i risultati appartengono a Z_{11} (proprietà di **chiusura**).
- Ogni riga ed ogni colonna contiene una **permutazione** (per l'esattezza una rotazione) di Z_{11}.
- Scambiando gli addendi si ottiene lo stesso risultato (proprietà **commutativa**).
- La somma di un elemento con 0 fornisce l'elemento stesso (0 è dunque l'**elemento identità**).
- In ogni riga ed in ogni colonna l'elemento 0 è presente una volta sola (esistenza dell'**inverso additivo**).

4.6.2 Moltiplicazione in GF(p): a × b mod p

Consideriamo di nuovo il caso semplice di **p = 11**, e tabuliamo **a × b mod 11** con **0 ≤ a, b < 11**.

	b										
a	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10
2	0	2	4	6	8	10	1	3	5	7	9
3	0	3	6	9	1	4	7	10	2	5	8
4	0	4	8	1	5	9	2	6	10	3	7
5	0	5	10	4	9	3	8	2	7	1	6
6	0	6	1	7	2	8	3	9	4	10	5
7	0	7	3	10	6	2	9	5	1	8	4
8	0	8	5	2	10	7	4	1	9	6	3
9	0	9	7	5	3	1	10	8	6	4	2
10	0	10	9	8	7	6	5	4	3	2	1

(a × b) mod 11

- Ogni riga della tabella (ad esclusione di a = 0) è una **permutazione** di Z_{11}.
- L'elemento identità della moltiplicazione è **1**.
- L'inverso moltiplicativo di ogni **a ≠ 0** è il valore di **b** che fornisce a × b mod 11 = 1; si noti che tale valore è presente una volta sola in ogni riga (ad esclusione di a = 0) ed in ogni colonna (ad esclusione di b = 0).

4.6.3 Esponenziazione in GF(p): $y = b^e \bmod p = (..(((b)b)b)..b) \bmod p$

← e volte →

La tabella riporta i risultati di $\mathbf{b^e \bmod m}$ nel caso semplice di **p = 11** e di **0 < e, b < p**.

	e									
b	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	1	1	1	1	1	1	1	1	1	1
2	2	4	8	5	10	9	7	3	6	1
3	3	9	5	4	1	3	9	5	4	1
4	4	5	9	3	1	4	5	9	3	1
5	5	3	4	9	1	5	3	4	9	1
6	6	3	7	9	10	5	8	4	2	1
7	7	5	2	3	10	4	6	9	8	1
8	8	9	6	4	10	3	2	5	7	1
9	9	4	3	5	1	9	4	3	5	1
10	10	1	10	1	10	1	10	1	10	1

b^e mod 11

Proprietà notevoli dell'esponenziazione modulo un primo p

- **T4** (piccolo teorema di Fermat): "per ogni primo **p** e per ogni $\mathbf{x} \in \mathbf{Z^*_p}$ si ha $\mathbf{x^{p-1} \bmod p = 1}$, cioè $\mathbf{x^{p-1} \equiv 1 \pmod p}$ o anche".

ESEMPIO – Si noti la presenza del solo numero **1** nella colonna e = 10 (N.B. **p-1**) della precedente tabella.

Corollari

T4.1: Per ogni $\mathbf{x} \in \mathbf{Z^*_p}$, se $\mathbf{s \equiv r \pmod{p-1}}$ allora si ha anche $\mathbf{x^s \equiv x^r \pmod p}$.

COMMENTI - Per ogni **e>p-1**, si ha dunque $\mathbf{x^e \equiv x^{e \bmod p-1} \pmod p}$. All'aumentare di **e**, la tabella si ripete ed è quindi inutilmente pesante calcolare l'esponenziazione modulo p per esponenti maggiori di **p-1**. Si noti che è anche inutilmente oneroso calcolare l'esponenziazione modulo p per basi maggiori di p-1: $\mathbf{b^x \bmod p = (b \bmod p)^x \bmod p}$

T4.2: $\mathbf{x^{(p-1)/2} \bmod p}$ fornisce o **+1**, o **–1**, cioè le due radici quadrate di 1; si noti in tabella la colonna e = 5.

In GF(p) è possibile eseguire divisioni: esiste infatti l'inverso moltiplicativo di ogni $\mathbf{x} \in \mathbf{Z^*_p}$, quale risulta definito dalla congruenza $\mathbf{x^{-1}\ x \equiv 1 \pmod p}$

- **T5**: "l'inverso moltiplicativo di ogni $\mathbf{x} \in \mathbf{Z^*_p}$ è valutabile con l'esponenziazione $\mathbf{x^{-1} = x^{p-2} \bmod p}$".

ESEMPIO – Moltiplicando mod 11 ogni elemento nella colonna 9 (N.B. **p-2**) per quello che denomina la riga in cui si trova, si ottiene sempre 1. Più avanti vedremo che è però più efficiente impiegare l'**algoritmo esteso di Euclide**.

Non tutte le righe sono **permutazioni** di $\mathbf{Z^*_{11}}$; la proprietà vale solo per b = 2, 6, 7, 8. In generale si ha:

- **T6**: "per ogni primo **p**, esiste almeno un **g < p,** detto **generatore mod p**, o **radice primitiva** di p, le cui potenze $\mathbf{g^1 \bmod p}$, $\mathbf{g^2 \bmod p}$, … $\mathbf{g^{p-1} \bmod p}$ forniscono tutti gli interi compresi tra **1** e **p-1** . Le radici primitive di **p** sono esattamente $\mathbf{\Phi(p-1)}$".

ESEMPIO – Trovare i generatori di un primo è un problema **difficile**. Facile invece è controllare se un elemento è un generatore. 2 è una radice primitiva di 11 (e di moltissimi altri numeri primi): $2^1 \bmod 11=2$, $2^2 \bmod 11=4$, $2^3 \bmod 11=8$, $2^4 \bmod 11=5$, $2^5 \bmod 11=10$, $2^6 \bmod 11=9$, $2^7 \bmod 11=7$, $2^8 \bmod 11=3$, $2^9 \bmod 11=6$, $2^{10} \bmod 11=1$.

- **T7**: "un $\alpha \in$ **GF(p)** è una radice primitiva se e solo se, per ogni **q** divisore primo di $\Phi(p)$, si ha $\alpha^{\Phi(p)/q} \neq 1$".

ESEMPIO - I generatori di **p = 11** sono quattro: $\mathbf{\Phi(10) = 10 \times (1-1/2) \times (1-1/5) = 4}$
Più precisamente i generatori sono:**2**, **6**, **7**,**8**
Verifica - $\Phi(11) = 10 = 2 \times 5$
In colonna e = 5 della tabella a lato si ha sempre $\mathbf{g^{(11-1)/2} = 10,}$ cioè –1. In colonna e = 2, $\mathbf{g^{(11-1)/5}}$ non è mai 1.

	e									
g	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
2	2	4	8	5	10	9	7	3	6	1
6	6	3	7	9	10	5	8	4	2	1
7	7	5	2	3	10	4	6	9	8	1
8	8	9	6	4	10	3	2	5	7	1

$\mathbf{g^e \bmod 11}$

Algoritmi per l'esponenziazione modulare

Esistono diversi algoritmi in grado di calcolare l'esponenziazione modulare
$\mathbf{y = b^e \bmod m}$ con $\mathbf{1 \le b,e \le m-1}$.

Il più inefficiente (ha tempo esponenziale!) si basa sulla definizione di esponenziazione:

$$\mathbf{b^e \bmod m = (..(((b)b)b)..b) \bmod m} \quad \text{(} \leftarrow \text{ e volte } \rightarrow \text{)}$$

Tutti gli algoritmi con tempo polinomiale usano la **rappresentazione binaria** dell'esponente **e**.
Posto $\mathbf{t = \lceil \log_2 m \rceil}$, si ha $\mathbf{e = e_{t-1}\, 2^{t-1} + e_{t-2}\, 2^{t-2} + \ldots\ldots + e_1\, 2^1 + e_0\, 2^0}$ e quindi

$$y = b^e = \prod_{i=0}^{t-1} b^{e_i 2^i} = (b^{2^0})^{e_0} \times (b^{2^1})^{e_1} \times \ldots \times (b^{2^{t-1}})^{e_{t-1}}$$

Dalla formula precedente discende il seguente algoritmo.

A0: INPUT: $b \in Z_m$, $0 \le e < m$, $e = (e_{t-1} .. e_0)_2$
OUTPUT: $y = b^e \bmod m$
1. si calcolano **b**, $\mathbf{b^2}$, $\mathbf{b^4}$, $\mathbf{b^8}$ ecc.,
2. si moltiplicano tra loro i $\mathbf{b^i}$ per i quali $\mathbf{e_i = 1}$
3. si divide il risultato per **m** e si trattiene il resto.

Valutiamone la complessità. Occorrono **t** elevamenti al quadrato (per calcolare i $\mathbf{b^i}$), al più **t** moltiplicazioni (per calcolare il prodotto dei $\mathbf{b^i}$) ed una divisione. Supponendo, per semplicità, che l'elevamento al quadrato e la divisione richiedano lo stesso tempo di calcolo di una moltiplicazione, nel caso peggiore l'algoritmo richiede l'esecuzione di **2t+1** moltiplicazioni.

Quanto modo di procedere ha però un difetto: elevamenti al quadrato e moltiplicazioni generano numeri sempre più grandi e richiedono di conseguenza tempo d'esecuzione e memoria sempre più grandi. Per eliminarlo, basta fare di volta in volta la riduzione in modulo dei risultati parziali.

Gli algoritmi con ***repeated square-and-multiply*** (v. [2], pag. 71 e pag. 615), conseguono maggiore efficienza non separando gli elevamenti al quadrato dalle moltiplicazioni: A1 scandisce i bit di **e** dal meno significativo al più significativo, A2 procede in ordine inverso.

A1: INPUT: $b \in Z_m$, $0 \le e < m$, $e = (e_{t-1} .. e_0)_2$
OUTPUT: $y = b^e \bmod m$

1. Set $y \leftarrow 1$. If $e = 0$ then return (y)
2. Set $A \leftarrow b$
3. If $e_0 = 1$ then set $y \leftarrow b$
4. For i from 1 to t-1 do the following:
   4.1 Set $A \leftarrow A^2 \bmod m$
   4.2 If $e_i = 1$ then set $y \leftarrow A \times y \bmod m$
5. Return (y)

A2: INPUT: $b \in Z_m$, $0 \le e < m$, $e = (e_{t-1} .. e_0)_2$
OUTPUT: $y = b^e \bmod m$

1. $y \leftarrow 1$
2. For i from t-1 down to 0 do the following:
   2.1 $y \leftarrow y^2 \bmod m$
   2.2 If $e_i = 1$ then $y \leftarrow y \times b \bmod m$
3. Return (y)

ESEMPIO A1 - **m = 11, b = 7, e = 5 =** (**101**)$_2$

	y	**A**
	1	**7**
e_0	**7**	7
e_1	7	**7 × 7 mod 11 = 5**
e_2	**3 × 7 mod 11 = 10**	**5 × 5 mod 11 = 3**

ESEMPIO A2 - **m = 11, b = 7, e = 5 =** (**101**)$_2$

	y
	1
e_2	**7**
e_1	**7 × 7 mod 11 = 5**
e_0	**5 × 5 mod 11 = 3** **3 × 7 mod 11 = 10**

La complessità di tali algoritmi è di **$O(\log_2 m)$** moltiplicazioni e quindi di **$O((\log m)^3)$** operazioni binarie.

NOTA – A parità di **b** e di **m**, il tempo di esecuzione dell'esponenziazione è proporzionale al n° di "uni" presenti in **e**; indicato con **t** il n° di bit di **e** e con **$1<n\le t$** il n° di bit con valore 1**,** il numero di moltiplicazioni modulari da eseguire è, infatti, **t+n** .

4.6.4 Logaritmo discreto

Nella precedente tabulazione di **$g^i \bmod 11$** si può notare che ogni elemento di **Z^*_{11}** compare in ogni riga una ed una sola volta. La proprietà notevole vale per qualsiasi primo e per qualsiasi sua **radice primitiva**.

- **T8**: "dati un primo **p**, un generatore **g** ed un qualunque intero **y** maggiore di 0 e minore di **p**, esiste un unico **x** (detto *logaritmo discreto* di **y** rispetto alla base **g**, modulo **p**) tale che **$y = g^x \bmod p$** con **$0 \le x \le (p-1)$** ".

ESEMPIO - Consideriamo **y = 3**: nella tabulazione di **$g^i \bmod 11$** tale valore è presente in ogni riga, una volta sola ed in una colonna sempre diversa.

	x									
g	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
2	2	4	8	5	10	9	7	3	6	1
6	6	3	7	9	10	5	8	4	2	1
7	7	5	2	3	10	4	6	9	8	1
8	8	9	6	4	10	3	2	5	7	1

$g^x \bmod 11$

Tutti gli algoritmi per il calcolo del logaritmo discreto finora individuati sono **non polinomiali**.

Ricerca esauriente – Si calcola **$g^x \bmod p$** per **x** = 0, 1, 2, .. fino a quando non si trova **y**. Il tempo atteso di esecuzione è **$2^{\lceil \log p \rceil - 1}$** esponenziazioni modulari.

La ricerca esauriente non è l'attacco più pericoloso.

Diversi anni fa è stata data una diversa formulazione al problema del logaritmo discreto: al posto della congruenza $\mathbf{y \equiv g^{x}}$ (mod p), si considera la congruenza $\mathbf{y \equiv g^{q\,t\,+\,r}}$ (mod p), in cui **q** e **r** sono, come indica la formula, il **quoziente** ed il **resto** della divisione di **x** per un certo intero **t < x**.

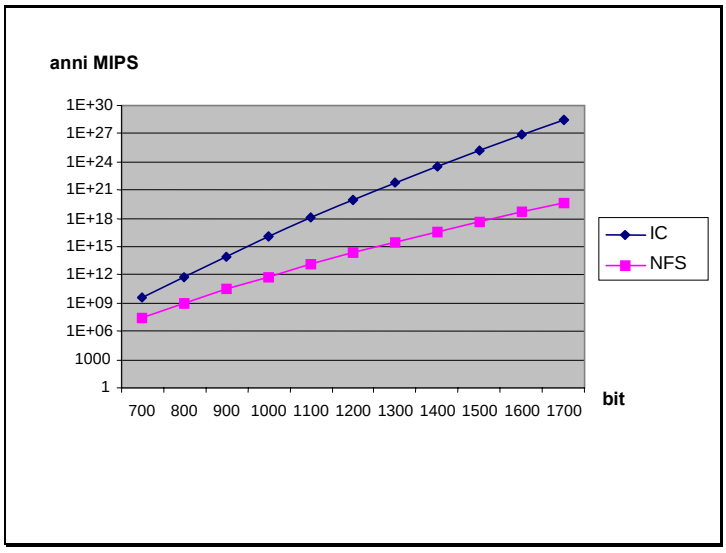
Il vantaggio rispetto alla forza bruta è che si deve eseguire un numero più basso di esponenziazioni con esponenti più piccoli di **x**. La contropartita è che occorre memorizzare **t** risultati parziali.

Un giusto equilibrio tra questi due aspetti si ottiene con $\mathbf{t = \lceil p^{1/2} \rceil}$.

Algoritmo passo del bambino-passo del gigante - Dalla precedente congruenza si ottiene $\mathbf{g^{r} \equiv y \times g^{-q\,t}}$ (mod p) e quindi anche $\mathbf{g^{r} \equiv y \times (g^{-t})^{q}}$ (mod p).

1. Si calcola $\mathbf{g^{r}\ mod\ p}$ per tutti i possibili valori di **r** ($\mathbf{0 \le r < t}$) e si costruisce una tabella in cui i risultati ottenuti, affiancati dal valore di **r** che li ha determinati, sono ordinati per valore crescente.
2. Si controlla se **y** è contenuto in una riga della tabella. Se ciò capita, si prende atto del valore di **r** che lo affianca, si restituisce **x = r** ed il calcolo termina; in caso contrario si pone **i = 1** e si calcola $\mathbf{z = y \times (g^{-t})\ mod\ p}$.
3. Si controlla se **z** è contenuto in una riga della tabella. Se ciò capita, si prende atto del valore di **r** che lo affianca, si restituisce $\mathbf{x = i \times t + r}$ ed il calcolo termina; in caso contrario si ripete con **i = i +1** e $\mathbf{z = z \times (g^{-t})\ mod\ p}$.

Sia $\mathbf{t = \lceil p^{1/2} \rceil}$. Dal punto di vista della complessità computazionale l'algoritmo richiede in questo caso **O(exp ((log p)/2))** operazioni ed altrettanti dati da memorizzare[41].



Algoritmi individuati successivamente[42] sono riusciti dapprima a ridurre le esigenze di memoria (Pohlig-Hellman, Pollard-ρ) e poi a rendere sub-esponenziale il tempo di esecuzione (Index-Calculus, Number Field Sieve, Function Field Sieve).

Con Index Calculus il tempo atteso di esecuzione ha ordine di grandezza asintotico

$$e^{((\ln p)^{1/2}(\ln(\ln p))^{1/2})}$$

Con Number Field Sieve si è successivamente riusciti ad ottenere

$$e^{((\ln p)^{1/3}(\ln(\ln p))^{2/3})}$$

Le curve quantitative indicate in figura vogliono evidenziare da un lato la differenza di questi tempi di esecuzione, da un altro lato l'evoluzione continua degli algoritmi di rottura.

COMMENTO – Per rendere impossibile il calcolo del logaritmo discreto oggi si usano primi di 1000 - 4000 bit.

Gli utenti di uno scambio DH devono però fare attenzione ai dati che impiegano: esistono, infatti, casi particolari in cui l'estrazione del logaritmo diventa facile!.

La scelta del numero segreto X può essere fatta a caso nell'intervallo 1 ÷ p-1, ma bisogna escludere i due estremi (v. pag. 78). Il motivo è molto semplice: se l'intruso intercettasse Y = g o Y = 1 non avrebbe bisogno di estrarre il logaritmo per arrivare alla conclusione che i corrispondenti valori di X sono 1 e p-1.

Anche la scelta di p è critica: Pohlig e Hellman hanno dimostrato che X può essere individuato con solo $2(\log p)^2$ moltiplicazioni, se i divisori di p-1 sono tutti numeri piccoli (v. [3] pag. 121)

4.6.5 Residui quadratici e sottogruppi ciclici

Per motivi di **efficienza** e di **sicurezza**, nelle applicazioni si impiega oggi una differente forma dello scambio di Diffie-Hellman. Il problema d'efficienza nasce dal fatto che per fronteggiare la crescente potenza di calcolo a disposizione dell'attaccante occorre via via i bit del modulo, ma purtroppo tale semplice accorgimento rende anche **sempre più oneroso** il calcolo che devono fare gli utenti. Passando ad esempio da 1024 bit di modulo (una dimensione oggi ancora in uso) a 2048 (il valore attualmente consigliato) il tempo di esecuzione dell'esponenziazione diventa otto volte più grande!

[41] Per la dimostrazione si veda [3] pag. 130. Per fare qualche valutazione si può sfruttare il notebook DL.nb predisposto da Ivo Muccioli.

[42] A.M.Odlyzko, "Discrete logarithms: The past and future", July 1999.

Il problema di sicurezza discende da una sottile vulnerabilità della versione originale: **chi intercetta il dato pubblico Y è in grado d'individuare agevolmente il valore del bit meno significativo del dato privato X**.

Entrambi i problemi sono risolti in maniera soddisfacente se si lavora su un **opportuno sottoinsieme** di Z^*_p: per giustificare tale affermazione, dobbiamo approfondire le nostre nozioni sulla Teoria dei numeri.

Come prima cosa, ci serve una definizione: un numero **q** è detto **residuo quadratico modulo p** se esiste un intero **x** tale che $\mathbf{x^2 \equiv q \pmod p}$; in caso contrario, q è detto essere **non-residuo quadratico**.

Metà dei numeri minori di p sono residui quadratici, metà sono non-residui quadratici.

ESEMPIO – I residui quadratici modulo 11 sono 1,3,4,5,9 e si trovano tutti nella colonna e=2 della tabella di pag.80. I non-residui quadratici non compaiono nella colonna e=2 e sono 2,6,7,8,10

Come seconda cosa, ci serve una deduzione: **ogni radice primitiva modulo p è un non-residuo quadratico**. Se così non fosse, infatti, l'operazione $\mathbf{g^X \bmod p}$ per x = 0, 1, 2, .., p-1 genererebbe solo residui quadratici e quindi solo la metà degli elementi del campo.

ESEMPIO – I quattro generatori di 11 sono non-residui quadratici: 2,6,7,8 (v. pag.81).

Come terza cosa, dobbiamo precisare meglio una proprietà già messa in evidenza in T4.2: se **a** è un residuo quadratico modulo p, allora $\mathbf{a^{(p-1)/2} \bmod p}$ vale **1**; in caso contrario vale **-1**.

Su tale proprietà, si basa un **test**, rappresentato dal simbolo (a/p) introdotto da Legendre, che consente di sapere se un numero minore di p è, o non è, un residuo quadratico modulo p. Per eseguire il test non è necessario eseguire l'esponenziazione: un algoritmo con complessità $O((\lg p)^2)$ è indicato in [2], pag. 73.

Da queste premesse discende che chi intercetta un $\mathbf{Y \equiv g^X}$ (mod p), può sapere agevolmente se è, o non è, un residuo quadratico modulo **p**. Nel primo caso **X** è necessariamente **pari**, nel secondo **dispari**.

ESEMPIO – Riprendiamo in considerazione le radici di p = 11 Nelle colonne e = 2,4,6,8,10 appaiono solo i residui quadratici 1,3,4,5,9. Nelle colonne e = 1,3,5,7,9 appaiono solo i non-residui quadratici 2,6,7,8,10.

	e									
g	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
2	2	4	8	5	10	9	7	3	6	1
6	6	3	7	9	10	5	8	4	2	1
7	7	5	2	3	10	4	6	9	8	1
8	8	9	6	4	10	3	2	5	7	1

g^e mod 11

La contromisura è **non impiegare radici primitive** come base dell'esponenziazione**.**

Scelto un qualsiasi elemento **h** di GF(p), le esponenziazioni $h^1 \bmod p$, $h^2 \bmod p$, … $h^x \bmod p$ forniscono elementi di GF(p). Sia **q** il più piccolo esponente per cui si ha $\mathbf{h^q \bmod p = 1}$.

Per un tale **h**, detto **generatore di ordine q**, valgono alcune proprietà notevoli:

- per ogni $1 \le i \le q$ e per ogni intero k si ha $\mathbf{h^{i+kq} \bmod p = h^i \bmod p}$;
- l'insieme $\{h^1 \bmod p , h^2 \bmod p , \ldots h^q \bmod p\}$ è un **sottogruppo moltiplicativo di $\mathbf{Z^*_p}$** e contiene **q** elementi;
- **q** è un **divisore** di p-1.

E' dunque possibile adoperare come base dell'esponenziazione DH un residuo quadratico, ma il suo ordine deve essere "grande" per rendere impossibile il citato attacco di Pohlig-Hellman.

Scelto un **p** grande (e quindi dispari), esiste certamente un **h** di ordine **q = (p-1)/2**. Per non stare a cercarlo, i crittografi hanno scoperto che è più semplice cercare un primo di forma particolare.

E' detto *safe prime* un primo della forma **p = 2q + 1**, ove **q** è anch'esso un numero primo[43]. Per tali primi esistono soltanto due sottogruppi, uno di dimensione 2 (che ovviamente non interessa) ed uno di dimensione q (generato da uno qualsiasi dei residui quadratici modulo p, ad esclusione di 1).

ESEMPIO – $p = 11 = 2 \times 5 + 1$ è un safe prime. L'esponenziazione con base b=3,4,5,9 e con esponente e=1,2,3,4,5 fornisce risultati tutti inclusi nell'unico sottogruppo {1,3,4,5,9}

	e									
b	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
3	3	9	5	4	1	3	9	5	4	1
4	4	5	9	3	1	4	5	9	5	1
5	5	3	4	9	1	5	3	4	9	1
9	9	4	3	5	1	9	4	3	5	1

b^e mod1

[43] Nella Teoria dei numeri, q è noto come primo di Sophie Germain

Se il *safe prime* p è formato da **n** bit, **q** ed **e** ne richiedono **n-1**, ma questa piccola diminuzione della dimensione dell'esponente non risolve il problema dell'efficienza citato all'inizio di questo paragrafo.

A tal fine occorre impiegare sottogruppi più piccoli, ma pur sempre troppo grandi per consentire all'intruso di condurre un attacco con forza bruta. In [4], pag. 215, è indicato il seguente metodo per costruirsene uno:

1. si sceglie un primo **q** di dimensione non troppo grande (ad esempio 256 bit);
2. si costruisce un primo **p** della forma **Nq+1**, con N pari e più grande di 2;
3. si individua in Z^*_p un generatore di ordine q della forma $\mathbf{g=\alpha^N \bmod p}$, cercando a caso un α tale che $g \neq \mathbf{1}$ e $\mathbf{g^q \bmod p = 1}$.

ESEMPIO – q = 3, N = 4 e quindi p = 13. Il sottogruppo {1,3,9} ha due generatori: 3 e 9.

Ai fini dello scambio devono essere noti **p**, **g** e **q** o, perlomeno, $\lceil \mathbf{\log q} \rceil$ (v. standard PKCS#3); ogni utente, dopo aver scelto a caso un $\mathbf{X<q}$ (o di $\lceil \mathbf{\log q} \rceil$ bit, calcola e invia $\mathbf{Y= g^X \bmod p}$.

Quando **q** è noto, l'utente può anche verificare che il dato pubblico dell'altro appartenga realmente al sottogruppo di ordine q (o con il test di Legendre, o calcolando $\mathbf{Y^q \bmod p}$ per verificare se vale 1).

4.7 Numeri primi

Per la sicurezza dello scambio DH occorrono dunque **numeri primi molto grandi**. Nel prossimo capitolo vedremo che questa esigenza è molto sentita in tutta la Crittografia asimmetrica.

4.7.1 Distribuzione dei numeri primi

Lo studio delle proprietà dei numeri primi ha da sempre affascinato i matematici.

ESEMPI – **Euclide** ha dimostrato che i numeri primi sono infiniti.
Eratostene ha indicato un metodo per individuare tutti i primi minori o uguali ad un certo **N**:
1 - si costruisce una tavola dei numeri naturali inclusi nell'intervallo **2 ÷ N**;
2 - si eliminano dapprima tutti i **multipli di 2**, poi **di 3**, poi **di 5** e così via fino a quando non si trova un numero il cui quadrato sia maggiore di **N**. A questo punto la tavola contiene solo numeri primi.
Fermat ha fatto discendere da T4 un test che consente di verificare se un certo intero **x** è primo o composto:

```
for i from 1 to x-1
    y = i^(x-1) mod x
    if y ≠ 1 then return "x è composto"
return "x è primo"
```

Gauss ha formulato una congettura, dimostratasi poi molto ben approssimata, sulla distribuzione dei numeri primi.
Riemann ha studiato una funzione di variabile complessa che ha congetturato avere una profonda connessione con la distribuzione dei primi: la connessione è stata verificata in più casi, ma non è stata ancora dimostrata.

Due teoremi della Teoria dei numeri forniscono un'utile indicazione sulla distribuzione dei numeri primi.

- **T9**: "$\lim_{x \to \infty} \frac{\pi(x)}{x / \ln x} = 1$ *ove* $\pi(x)$ *è il n° di primi* $\leq x$ "

ESEMPI – Una buona approssimazione di $\pi(\mathbf{n})$, valida anche per **n** piccoli, è dunque **n/ln n**; ad esempio per $\pi(\mathbf{30})$, che vale **10**, si ha **30/ln 30** = **8,82**.

Nell'interessante sito http://primes.utm.edu, tutto dedicato alle proprietà dei numeri primi, è indicato che è ancora migliore l'approssimazione $\pi(\mathbf{x}) \cong \mathbf{x/(ln\ x\ -1)}$.

Da T9 discende che l'ennesimo numero primo è approssimativamente fornito dalla formula $\mathbf{p_n \cong n \ln n}$.

- **T10** (Dirichlet): "se **MCD(a, n) = 1**, allora la successione $\mathbf{p = a + k \times n}$, con k= 1,2,3,… contiene infiniti numeri primi congruenti ad **a** modulo **n**".

ESEMPI – Attentamente studiate dai Crittografi sono state le progressioni $\mathbf{4k \pm 1}$ (che forniscono tutti i numeri dispari e quindi tutti i primi escluso 2), $\mathbf{6k \pm 1}$ (che individuano, con maggiore efficienza, tutti i primi esclusi 2 e 3), $\mathbf{8k \pm 1}$ e $\mathbf{8k \pm 3}$ (che individuano, con ancora maggiore efficienza, tutti i primi ad esclusione di 2, 3 e 5), ecc..

La progressione **4k-1** contiene tutti i numeri primi congruenti a 3 modulo 4 (v. pag. 31), la progressione **6k-1** contiene i $p \equiv 5 \pmod 6$ ed in particolate tutti i primi di Sophie Germain e tutti i safe prime (v. pag.84).

4.7.2 Ricerca di numeri primi grandi

Scelto a caso un **x** grande, è dunque molto probabile trovare un primo in un suo intorno di ampiezza **ln x**. Per individuarlo servono solo un buon **test di primalità** e l'esecuzione del seguente algoritmo.

Algoritmo per la ricerca di un grande numero primo -
1. si genera a caso un numero intero grande e dispari;
2. si sottopone il numero ad un test di primalità;
3. se il test lo dichiara primo, si termina; in caso contrario o si torna al passo 1, o si incrementa il numero di 2 e si ritorna al passo 2.

I test di primalità di cui oggi disponiamo sono classificabili in due categorie.
1. Test **deterministici**: se **n** non lo supera è **composto**, se lo supera è **primo**.
2. Test **probabilistici**: se **n** fallisce il test è **composto**; se lo supera è **probabilmente** primo.

I test deterministici hanno complessità esponenziale[44], quelli probabilistici, polinomiale. Questi ultimi devono però essere ripetuti più e più volte per far tendere a 1 la probabilità di avere realmente individuato un primo.

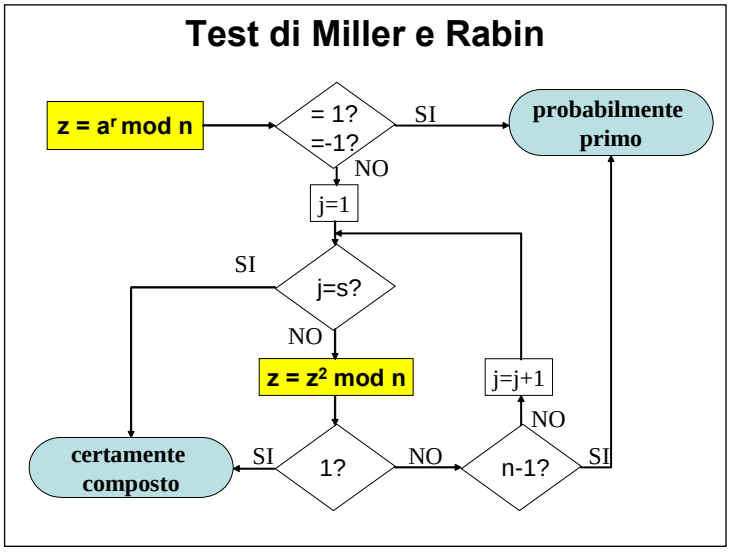
Il più usato è il **test di Miller-Rabin**. Il principio è di scegliere a caso un intero **a**, con **1 <a <n** e di calcolare $\mathbf{a^{n-1}}$ **mod n**: se il risultato è diverso da 1, allora **n** è certamente composto; se il risultato vale 1, **n** è detto **pseudoprimo in base a** e può anche essere un numero primo (per poter affermare con certezza che **n** è primo bisognerebbe verificare **T4** per ogni possibile **a**, come abbiamo visto nel test di Fermat citato in precedenza).

Sia **n** un numero dispari. Essendo **n-1** pari, si pone $\mathbf{n-1 = r \times 2^s}$ con **r** dispari e $\mathbf{s \geq 1}$. Si ha dunque:

$$a^{n-1} \bmod n = (a^r)^{2^s} \bmod n$$

Il test inizia con il calcolo di un'esponenziazione "più semplice" che quella richiesta dal test di Fermat:

$\mathbf{z = a^r \bmod n}$.



Se si ottiene **z = 1** o **z = n-1** (N.B. cioè –1) il test termina e **n** è dichiarato **probabilmente primo**: l'elevamento di z alla potenza 2^s è inutile essendo già certo che il risultato sarà 1.

In caso contrario si prosegue iterando al più s-1 volte il calcolo

$\mathbf{z = z^2 \bmod n}$.

Se in una qualsiasi delle iterazioni si ottiene **z = n-1** il test termina per le ragioni sopradette e **n** è dichiarato **probabilmente primo**.

Il test termina, anche se in una qualsiasi iterazione si ottiene $\mathbf{z^2 = 1}$: è, infatti, certo che solo un **numero composto**, diverso da 1 e da -1 può avere il quadrato uguale a 1 (v. [2], Fact 3.18). La stessa conclusione si può trarre se si arriva all'ultimo passo senza aver mai trovato né 1, né -1.

Per poter confidare sulla primalità di **n**, bisogna ripetere il test più volte con valori di **a** scelti a caso: dopo **t** test, tutti superati con la dichiarazione che **n** è probabilmente primo, la probabilità che non lo sia è più piccola di $\mathbf{2^{-2t}}$.

ESEMPIO – In [2], nell'ipotesi che il numero da testare sia stato generato a caso dall'utente, si dimostra che sono sufficienti 4-5 iterazioni.

In [4] si esamina il contesto di un primo ricevuto da una terza parte (che potrebbe essere un malintenzionato), ad esempio prima di procedere ad uno scambio DH: in questo caso viene suggerito di ripetere il test 64 volte per garantirsi un livello di sicurezza di 128 bit (se ci si fida e si usa come modulo un numero non primo, il dato privato non sarebbe, infatti, più protetto dall'impossibilità di calcolo del logaritmo discreto e la successiva chiave condivisa diventerebbe nota anche all'avversario).

[44] Nel 2002 Manindra Agrawal, Neeraj Kayal e Nitin Safena, tre ricercatori dell'Indian Institute of Technology di Kanpur, hanno per la prima volta individuato un algoritmo con complessità polinomiale. Studi successivi su AKS ne hanno ulteriormente ridotto la complessità computazionale.