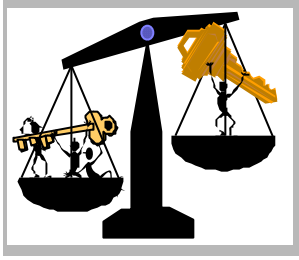
Capitolo 5



Meccanismi asimmetrici

5.1 Proprietà delle chiavi ….....……..……88

5.2 Robustezza degli algoritmi asimmetrici ……..……89

- Funzioni unidirezionali e pseudo-unidirezionali
- Problemi difficili

5.3 Autenticità della chiave pubblica ……..……..…91

- Attacco dell'uomo in mezzo
- Il certificato
- L'Autorità di certificazione
- Sicurezza in rete: l'uso dei certificati

5.4 Cifrari asimmetrici …..……...97

- Il Cifrario RSA
- Il Cifrario di ElGamal
- Il Cifrario ibrido
- Il Cifrario a flusso di Blum-Goldwasser

5.5 Schemi di firma digitale ..……..……...108

- Firma con RSA
- Firma con ElGamal e con DSS

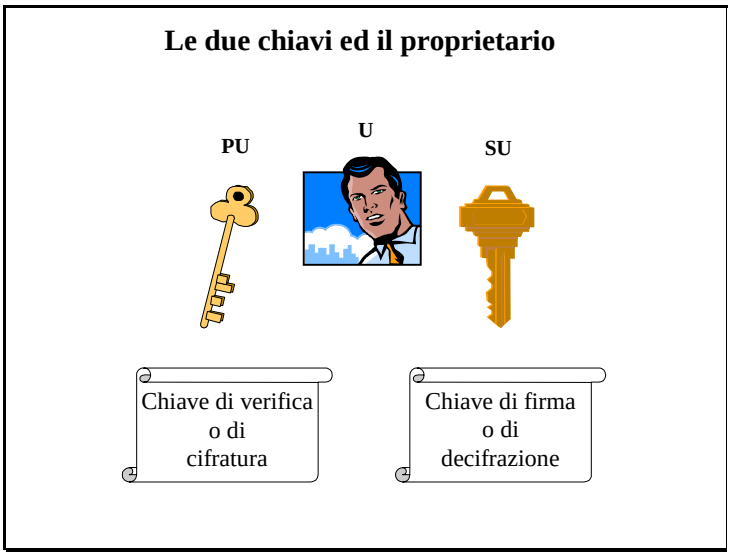
5.6 Protocolli d'identificazione ...…….115

- Protocollo challenge/response
- Zero-knowledge

5.1 Proprietà delle chiavi

Oggetti di studio della Crittografia a chiave pubblica sono i Cifrari asimmetrici, i Generatori di bit pseudocasuali, gli Schemi di firma digitale ed i Protocolli d'identificazione attiva.

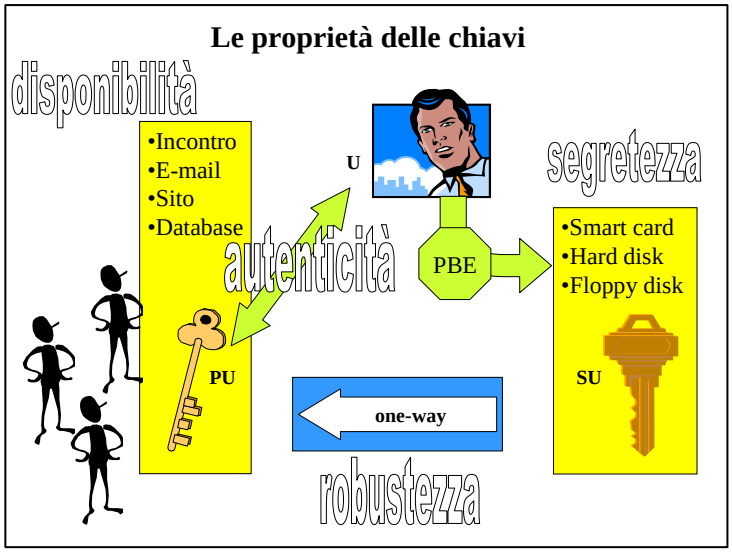
I meccanismi asimmetrici per la riservatezza e per l'autenticazione introducono in Crittografia tre aspetti di importanza fondamentale dal punto di vista della gestione delle chiavi.



1. Una volta scelto un Cifrario asimmetrico, **una sola chiave**, quella **privata**, consente ad ogni utente di decifrare i messaggi riservati che **chiunque altro gli invia** cifrati con la corrispondente chiave pubblica; chi non conosce la chiave privata, non riesce a decifrare.
2. Una volta scelto uno Schema di firma, **una sola chiave**, quella **privata**, consente ad ogni utente di firmare messaggi di cui chiunque altro potrà verificare l'autenticità usando la corrispondente chiave pubblica; chi non conosce la chiave privata non riesce ad imitare la firma del suo proprietario.
3. La chiave pubblica di un utente può essere comunicata a tutti gli altri **senza alcuna forma di riservatezza**. Il canale può essere la voce, o uno scritto, ma la soluzione migliore, data la dimensione, è un **file**.

ESEMPIO – La chiave pubblica PGP di Roberto Laschi può essere scaricata dal sito del Corso o dal server **https//keyserver2.pgp.com/vkd/SubmitSearch.event** impiegando rlaschi@deis.unibo.it come "User ID".

Quattro sono gli aspetti a carico di ogni utente U di un meccanismo asimmetrico.



- **Disponibilità della chiave pubblica PU**: chiunque ne ha bisogno deve poterla facilmente reperire; a tal fine possono essere impiegati o un incontro diretto tra gli interessati, o un'e-mail inviata dal proprietario ai suoi potenziali corrispondenti, o un file che questi ultimi possono scaricare dal sito del proprietario, o, ed è questa la soluzione migliore, un record che chiunque può ottenere da un **database**.

- **Segretezza della chiave privata SU**: solo il proprietario deve poterne disporre; a tal fine deve conservarla cifrata (con la tecnica PBE) in una memoria elettronica con controllo d'accesso (hard disk, o floppy, o, molto meglio, una **smart card**).

- **Robustezza agli attacchi di Crittanalisi**: dopo aver scelto un meccanismo per cui sono noti **solo** algoritmi di attacco non polinomiali, il proprietario deve attribuire alle sue chiavi una **dimensione** atta a renderli computazionalmente non eseguibili; approfondiremo questo aspetto nel paragrafo 5.2.

- **Autenticità della chiave pubblica**: il proprietario deve rendere di pubblico dominio anche informazioni atte a consentire a chiunque di verificare, in modo sicuro ed inequivocabile, la sua **identità**; approfondiremo questo aspetto nel paragrafo 5.3.

5.2 Robustezza degli algoritmi asimmetrici

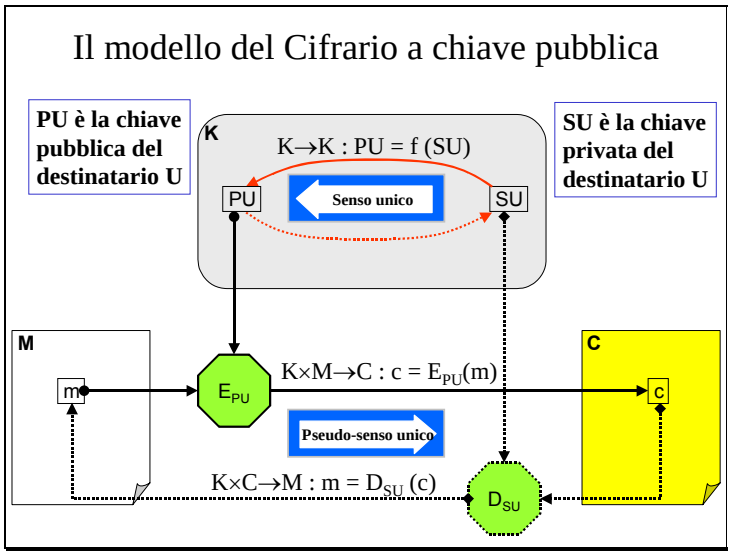
5.2.1 Funzioni unidirezionali e pseudo-unidirezionali

Nel cap. 2, a pag. 22, abbiamo introdotto il modello della **funzione unidirezionale**, evidenziando il suo importantissimo ruolo nella Crittografia simmetrica ed asimmetrica: sempre nel cap. 2, a pag. 25, abbiamo poi subito anticipato una **prima esigenza**:
"la funzione **K → K: PU = f (SU)** tra le chiavi **PU**, **SU** di un qualsiasi algoritmo asimmetrico non deve poter essere invertita, cioè deve essere una di quelle "candidate" ad essere considerate unidirezionali".

ESEMPIO - Nello scambio DH (si noti che non si tratta né di un Cifrario, né di uno Schema di firma digitale) la funzione candidata ad essere unidirezionale è l'esponenziazione modulo un grande numero primo.

Abbiamo infine segnalato una **seconda esigenza**:
"le due chiavi devono selezionare trasformazioni corrispondenti e la trasformazione segreta (la decifrazione nel caso della riservatezza, la firma nel caso dell'autenticazione) deve essere facile da eseguire solo per chi conosce la chiave privata".
Occorrono dunque anche le funzioni **pseudo-unidirezionali**.

Anche in questo caso nessuno ha fornito una dimostrazione di esistenza, ma molte sono le funzioni che presentano tale proprietà, sfruttando la presunta difficoltà di problemi della Teoria dei numeri.



Il ruolo dei due modelli è illustrato in figura facendo riferimento ad un Cifrario asimmetrico, un "crittosistema" i cui tre algoritmi ***G***, ***E***, ***D***, hanno tutti **tempo polinomiale**.

U impiega ***G*** per generare la sua coppia di chiavi. Tale algoritmo deve garantire la **unidirezionalità** della funzione

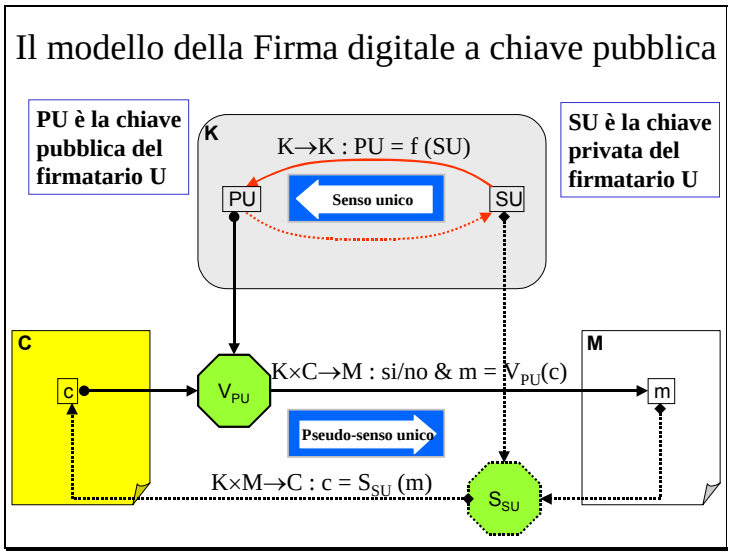
K → K: PU = f (SU).

Chiunque vuole inviare a **U** un messaggio riservato impiega l'algoritmo ***E***. Tale algoritmo deve garantire la **pseudo-unidirezionalità** della funzione di cifratura

K × M → C: c = E_{PU}(m).

U, infine, impiega ***D*** per decifrare. La conoscenza di **SU** deve costituire l'unico mezzo per rendere facile il calcolo della funzione

K × C → M : m = D_{SU} (c).



Facciamo riferimento ora ad uno Schema di Firma, un crittosistema formato da tre algoritmi ***G***, ***S***, ***V*** , tutti di facile esecuzione.

Anche in questo caso, ***G*** garantisce la **unidirezionalità** della funzione **PU = f (SU)**.

Consideriamo qui il caso di un algoritmo ***V*** che mette in chiaro il crittogramma che gli viene fornito in ingresso e che risponde "vero" se e solo se corrisponde al testo in chiaro del messaggio firmato da U.

Con questa ipotesi (ma vedremo che esistono altri algoritmi che giudicano l'autenticità senza rendere disponibile all'esterno ciò che hanno decifrato) deve essere **pseudo-unidirezionale** la funzione **K × C → M: m = V_{PU}(c)**

U, che conosce la ***trapdoor***, può così essere l'unico a riuscire a calcolare la funzione inversa con l'algoritmo ***S***.

5.2.2 Problemi difficili

Le funzioni unidirezionali e pseudo-unidirezionali impiegate dalla Crittografia asimmetrica si basano dunque sull'assunzione che per certi problemi della Teoria dei Numeri non si troveranno mai algoritmi di soluzione con tempo polinomiale. E' utile disporre di una panoramica, anche se poi in questo corso chiameremo esplicitamente in causa soltanto i primi tre.

- **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**, $1 \le x \le p-2$, tale che $g^x \bmod p = c$, o anche $\mathbf{g^x \equiv c \pmod p}$.

COMMENTI – Abbiamo già visto che su P1 si basa la sicurezza dello scambio DH e che l'impossibilità di calcolo si ottiene oggi con numeri primi formati da almeno un migliaio di bit. Si ricordi che lo scambio, con la stessa robustezza, può essere definito su GF(p), su un suo sottogruppo di grande ordine e su $GF(p^n)$.

Vedremo tra poco che l'intrattabilità di P1 è il presupposto che rende pseudo-unidirezionale la funzione di cifratura del Cifrario di ElGamal e la funzione di verifica dello Schema di firma digitale DSS.

- **P2: problema della radice e-esima** – Calcolare $\mathbf{m \equiv \sqrt[e]{c} \pmod n}$ con **n** prodotto di due numeri primi ed **e** coprimo con $\Phi(\mathbf{n})$.

COMMENTI – Il problema è oggi risolvibile solo con algoritmi di complessità sub-esponenziale. E' invece facile se n è un numero primo o se si conoscono i fattori di n; vedremo che su quest'ultimo fatto si basa la trapdoor dell'algoritmo RSA, sia quando è impiegato come Cifrario, sia quando è impiegato come Schema di firma digitale.

- **P3: problema della fattorizzazione intera -** Dato un intero composto **n**, trovare i numeri primi $\mathbf{p_i}$ ed i coefficienti $\mathbf{e_i} \ge 1$ tali che $\mathbf{n = p_1^{e1} \times ... \times p_k^{ek}}$.

COMMENTI – La dimensione di **n** che rende impossibile la fattorizzazione si è modificata nel tempo.

- Gauss e Fermat sono stati i primi a segnalare la difficoltà di fattorizzare numeri con più di 20 cifre decimali.
- Nel 1970 era considerato difficile fattorizzare con un calcolatore un numero di 41 cifre decimali.
- Nel 1977 Rivest valutò che un main frame di quell'epoca avrebbe impiegato 10^{15} anni per fattorizzare un numero di 125 cifre decimali.
- Nel 1994 sono bastati 8 mesi per fattorizzare un numero di 129 cifre decimali utilizzando circa 1.600 computer collegati ad Internet.
- Nel 2000 si è stimato che occorra una macchina parallela da 10 milioni di dollari per riuscire a fattorizzare in un anno un numero di 150 cifre decimali (circa 500 bit se il numero è binario). Si era inoltre previsto di poter fattorizzare numeri di 1024 bit nel 2004 (ma non è stato così) e numeri di 1500 bit nel 2014.
- Nel 2004 A. Shamir ha realizzato il prototipo di una macchina optoelettronica, TWINKLE, potenzialmente in grado di incrementare di due o tre ordini di grandezza l'efficienza degli attuali algoritmi di fattorizzazione; per ora però il costo è proibitivo,

L'istanza del problema che interessa l'algoritmo RSA è $n = p \times q$, con $p \ne q$.

Il più intuitivo algoritmo di rottura è "la divisione di prova": si divide n per 2, 3, 5, 7 .. fino a quando non si trova un quoziente intero. Il più piccolo divisore di n è sicuramente minore di $n^{1/2}$; la complessità della divisione di prova è dunque: $O(\exp(\frac{1}{2}(\log n)))$.

Non è questo però l'attacco più pericoloso. Anche per la fattorizzazione, infatti, sono stati individuati algoritmi sub-esponenziali; il più efficace è GNFS (*General Number Field Sieve*) ed ha complessità

$$O(\exp((\ln n)^{1/3} (\ln \ln n)^{2/3})).$$

Per avere sicurezza occorrono dunque oggi chiavi di almeno 1000 bit.

Non bisogna però dimenticare che esistono algoritmi estremamente efficaci in casi particolari. Per evitare di cadere in queste situazioni si può imporre vincoli a p e a q.

Una proposta è quella di usare *strong primes* (un numero primo p è *strong* se p-1 e p+1 hanno grandi fattori primi r, con r-1 dotato a sua volta di un grande fattore primo t); un'altra è che $|p-q|$ sia più grande di $n^{1/2}$. In pratica però è considerato sicuro anche sceglierli a caso.

- **P4: problema del fusto** - Dato un insieme di **p** interi positivi $\mathbf{\{a_1, a_2, .., a_n\}}$ ed un intero positivo **s** determinare se esiste o no un sottoinsieme di $\mathbf{a_j}$ la cui somma è **s**.

COMMENTI – Il problema, detto anche "dello zaino", è NP-completo ed è stato per la prima volta chiamato in causa dal Cifrario di Merkle-Hellman. Il Cifrario si è però dimostrato insicuro; da esso è poi disceso il Cifrario di Chor-Rivest, che finora non ha mostrato alcun punto debole.

- **P5: problema della radice quadrata modulare -** Dato un intero composto **n** ed un elemento $\mathbf{a} \in \mathbf{Q_n}$ (insieme dei residui quadratici modulo n) trovare la radice quadrata di a modulo n, cioè un **x** tale che $\mathbf{x^2 \equiv a \pmod n}$.

COMMENTI – Il già visto PRNG BBS ed il Cifrario di Rabin (che non discuteremo) usano $\mathbf{n = p \times q}$, con p e q distinti numeri primi congruenti a 3 modulo 4: tale **n**, detto intero di Blum, ha la proprietà notevole che uno solo dei quattro distinti **x** che soddisfano $\mathbf{x^2 \equiv a \pmod n}$ appartiene a sua volta a $\mathbf{Q_n}$. Quando p e q sono noti, la soluzione è facile; se li si tiene segreti, questa è la trapdoor, si mette l'avversario di fronte al problema della fattorizzazione.

- **P6: problema della residuosità quadratica –** Dato un numero intero composto e dispari **n** ed un elemento $\mathbf{a} \in \mathbf{J_n}$ (insieme degli interi con simbolo di Jacobi uguale a 1) determinare se **a** è o non è un quadrato mod n.

COMMENTI – Il primo ad accorgersi della difficoltà del problema è stato Gauss. L'apparente intrattabilità di P6 quando n non è primo è alla base della sicurezza del Cifrario di Goldwasser-Micali (qui non lo studieremo) e del PRNG BBS (v. pag. 31). In entrambi si considera l'istanza in cui n è un intero di Blum.

Da una decina d'anni è stato preso in considerazione un ulteriore e piuttosto complesso contesto matematico[45], su cui sviluppare crittosistemi a chiave pubblica. La sicurezza chiama in causa il seguente problema.

- **P7: problema del logaritmo discreto su una curva ellittica –** Data la curva ellittica formata da punti le cui coordinate **x**, **y** soddisfano l'equazione

$$\mathbf{y^2 = x^3 + ax + b \pmod p}, \text{ con } \mathbf{p} \text{ primo,}$$

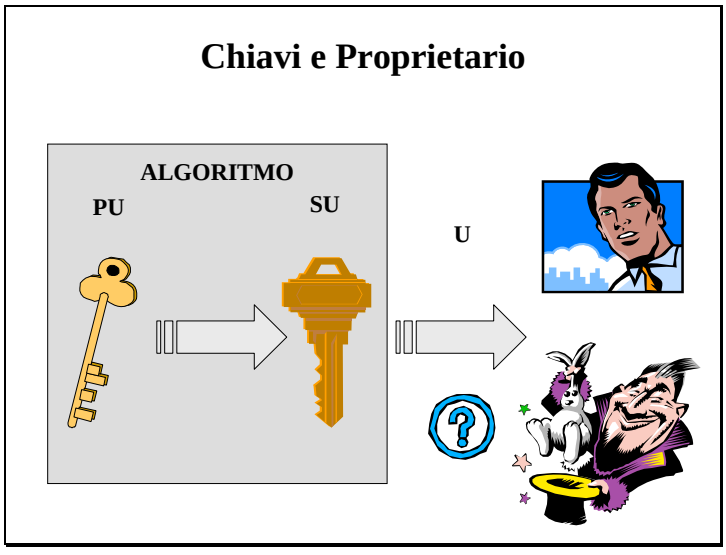
e due suoi punti **P**, **Q** tali che $\mathbf{Q = n \times P}$, determinare **n**.

COMMENTI - L'aspetto interessante di P7 è che per ora nessuno ha trovato algoritmi di rottura con complessità inferiore a $\mathbf{O(\exp(\frac{1}{2}(\log p)))}$. Molti sono però convinti che il problema non sia stato ancora analizzato a sufficienza. Il sito www.certicom.com presenta molte interessanti considerazioni sull'impiego della *Elliptical Curve Cryptography* (**ECC**).

Una chiave di soli 160 bit sembra dunque essere adeguata: ciò consente di cifrare, decifrare e firmare anche in ambienti di elaborazione, come le smart card, in cui si hanno vincoli abbastanza stringenti sulla potenza di calcolo, sul consumo di potenza elettrica, sulla capacità di memoria e sulla lunghezza dei dati da elaborare.

5.3 Autenticità della chiave pubblica

La conoscenza di **PU** è una condizione necessaria, ma non sufficiente, per proteggere la riservatezza di un messaggio da inviare a **U** o per attribuire a **U** la paternità di un documento firmato con **SU**.



Chi usa **PU** ad un'estremità del canale è, infatti, ben consapevole che all'altra estremità è stata o dovrà essere impiegata la chiave **SU**, ma non ha alcuna garanzia che la coppia di numeri **SU**, **PU** sia proprio dell'utente di nome **U** e non di un impostore che vuole spacciarsi per **U**.

In queste condizioni il pericolo è grosso: si può, infatti, o inviare ad un intruso informazioni riservate destinate solo ad **U**, o accettare come originate da **U** informazioni false predisposte dall'intruso.

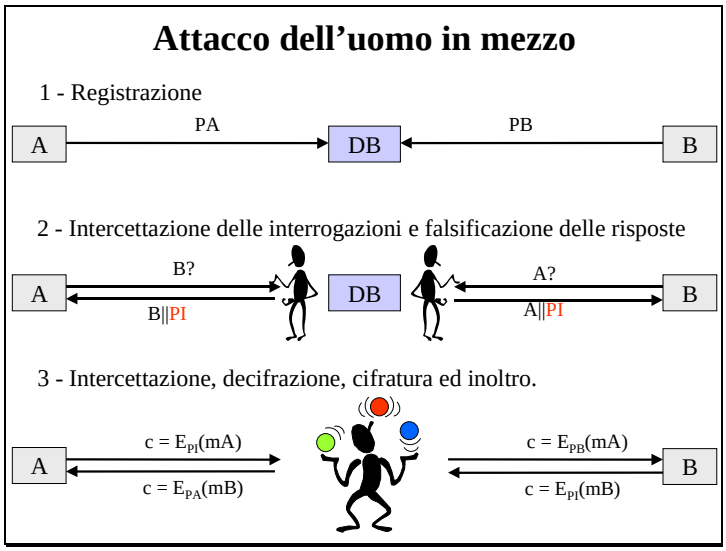
Nei paragrafi successivi vedremo dapprima cosa può fare l'intruso per creare queste condizioni a lui favorevoli e poi cosa devono fare gli utenti per vanificare i suoi attacchi. Possiamo però già fin d'ora tirare una conclusione dettata solo dal buon senso e dal nostro usuale modo di comportarci (ad es. quando dobbiamo decidere se accettare o no un assegno):

- R28: "*prima d'impiegare una chiave pubblica bisogna o essere personalmente certi dell'identità del suo proprietario o disporre di un documento che l'attesta, redatto da un'Entità di cui ci si fida*".

[45] Per un primo approfondimento si veda [6], cap. 10, pag. 317-328. Nel sito del corso è disponibile materiale didattico predisposto da G. Dionigi

5.3.1 Attacco dell'uomo in mezzo

La sicurezza d'uso di una chiave pubblica è minacciata da un particolare tipo di attacco attivo, detto **attacco dell'uomo in mezzo**.



Consideriamo il caso di un Cifrario asimmetrico e supponiamo che due utenti **A** e **B** abbiano inserito in un database DB le loro chiavi pubbliche **PA** e **PB**.

L'intruso **I** che vuole violare la riservatezza delle loro comunicazioni deve fare solo tre cose, neanche troppo difficili:

1 - fa apparire la sua chiave pubblica **PI** al posto di quelle di **A** e di **B**, o modificando il contenuto di DB o dirottando tutte le richieste di accesso sul suo calcolatore;

2 - quando **A** invia un $c = E_{PI}(m)$ a **B**, lo intercetta, lo decifra, lo cifra con **PB** e lo inoltra a **B**;

3 – quando **B** risponde ad **A** ripete lo stesso attacco attivo, completando così la violazione della riservatezza dei messaggi tra **A** e **B**.

La minaccia dell'uomo in mezzo riguarda ovviamente anche gli algoritmi asimmetrici di firma.

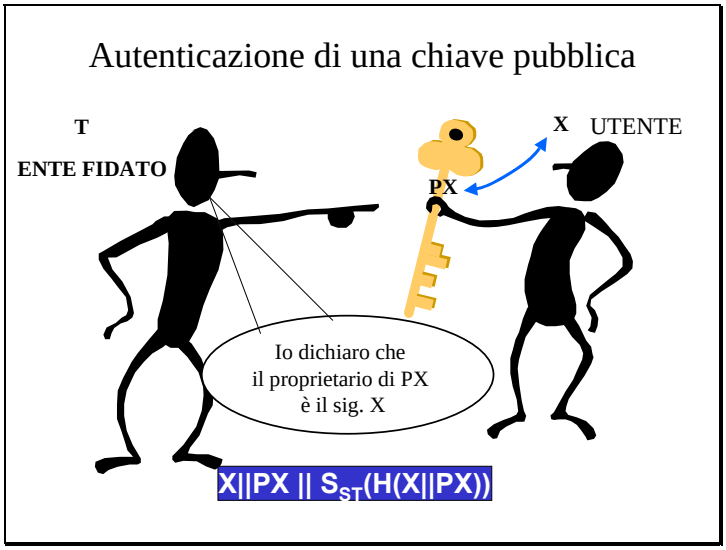
La contromisura è comunicare chiavi pubbliche di cui è possibile verificare l'origine, naturalmente senza perdere i due grandi vantaggi della Crittografia asimmetrica:

- la comunicazione della chiave deve essere non riservata,
- l'identificazione del proprietario della chiave deve poter essere fatta da chiunque.

5.3.2 Il certificato

Due che si conoscono non hanno particolari problemi ad autenticare le loro chiavi pubbliche.

Una soluzione ovvia è che se le scambino durante un incontro diretto. Una seconda soluzione è che se le inviino per posta ed usino poi una telefonata per confermarne a voce il valore (v. esempio a pag. 16).



In generale, però, chi deve usare una chiave pubblica non conosce il suo proprietario.

In questo caso la certificazione della associazione **chiave-proprietario** deve essere:

- fatta da qualcuno di cui ci si può **fidare**,
- comunicata con un messaggio di cui sia possibile verificare **integrità** ed **origine**.

Il messaggio che attesta l'identità del proprietario di una chiave pubblica è detto **certificato**.

L'Ente **T** che produce certificati è detto **terza parte fidata**.

La **firma digitale** di **T** apposta sul certificato tramite un algoritmo asimmetrico è l'accorgimento efficiente che consente a tutti gli interessati di verificare l'integrità e l'origine dell'attestato. Il presupposto è che tutti abbiano una copia autentica di **PT**.

Il **certificato** della chiave pubblica di un generico utente **X** da parte di **T** è formato da un testo in chiaro **m**, affiancato dal suo hash firmato con **ST**: **cert (PX, T) = m||S_{ST}(H(m))**.

Il testo in chiaro è, a sua volta, composto di diverse parti: **m = X||iX||PX||iPX||iT** ove

- **X** è il cognome-nome dell'utente, **iX** i dati che lo contraddistinguono (attività, indirizzo di e-mail, ecc.),
- **PX** è la chiave, **iPX** le informazioni per usarla (gli algoritmi di hash e di firma, il periodo di validità, ecc.),
- **iT** sono le informazioni che contraddistinguono **T** (l'identificativo, gli algoritmi di hash e di firma impiegati) e che consentiranno di svolgere la verifica d'integrità e d'origine del certificato.

Una volta predisposto il certificato di **X**, **T** può consegnarlo

- ad un altro utente **Y** che gli ha direttamente richiesto l'autenticazione di **PX**,
- allo stesso utente **X**, che provvederà poi ad inoltrarlo a chi glielo richiederà,
- ad un **database** a libero accesso, da cui chiunque potrà estrarlo.

Chi riceve un certificato emesso da **T**, si procura **PT** in modo sicuro (se non l'ha già), calcola

$$u = V_{PT}(cert(PX,T)) = vero/falso$$

e solo in caso di verifica positiva **trasferisce su PX la "fiducia" che ha su PT**.

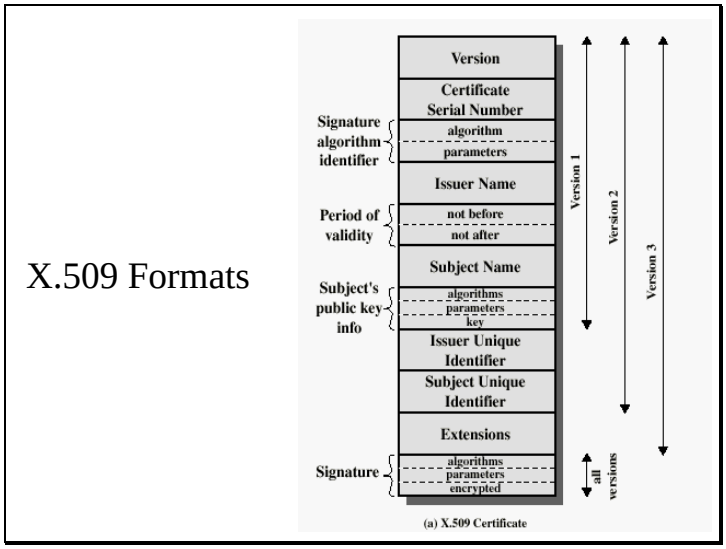
Le azioni di **T** e di **Y** devono poter essere eseguite in modo automatico: occorrono quindi standard ben precisi che definiscano il formato del **certificato**.

Occorre anche decidere chi può ricoprire il ruolo di **T**; due sono i modelli in uso.

1. **T** è un Ente ufficialmente riconosciuto come fidato ed istituzionalmente incaricato di emettere certificati.
2. **T** è un qualsiasi utente che dichiara spontaneamente di conoscerne bene un altro; chi si fida di lui, si fida anche della sua dichiarazione.

La prima soluzione, detta della **Autorità di certificazione** (o **CA**), è formalizzata nello standard **ISO X.509** ed è impiegata sia in applicazioni sicure, sia nelle reti sicure; la seconda soluzione, propugnata dal **PGP**, è il contesto preferito da tutti quelli che non vogliono un Grande Fratello[46] sopra di loro.

1 - Il certificato CCITT ISO X.509

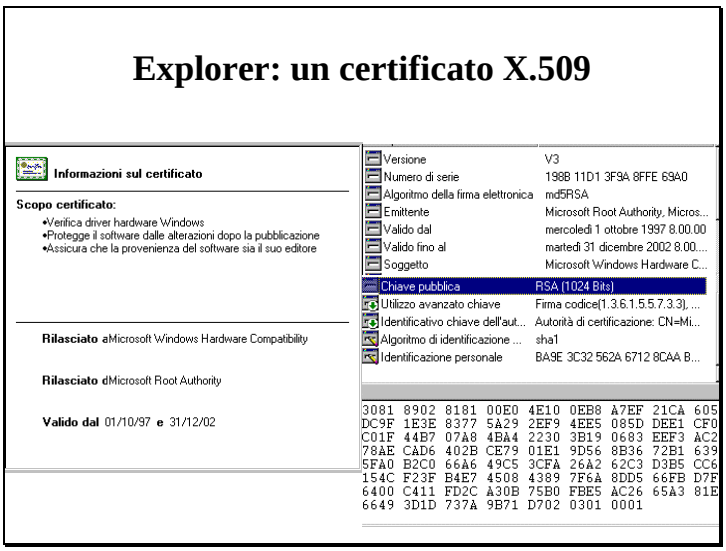


Lo standard prevede una suddivisione del certificato in diversi campi in chiaro, seguiti dalla firma del loro hash con la chiave privata di **CA** (in figura, l'ultimo campo in basso). La versione dello standard è indicata nel primo campo in alto (attualmente è in uso la v3, ma nel seguito, per semplicità, prenderemo in considerazione solo la 1).

I tre campi sottostanti contengono informazioni sulla **CA** (n° d'ordine del certificato, algoritmo di firma usato, nome e identificativo della autorità).

Altri tre campi infine forniscono il periodo di validità del certificato, il nome del proprietario della chiave pubblica certificata, il suo valore e le informazioni per impiegarla.

Il file di un certificato X.509 ha di norma l'estensione **.cer** ed una dimensione di circa 1KB.



Si noti che il certificato è stato prima compresso con l'algoritmo di hash MD5 e poi firmato con l'algoritmo RSA (che studieremo più avanti).

Si noti anche l'apparente incongruenza tra la lunghezza in bit della chiave (il numero 1024 affiancato alla sigla RSA sulla riga "Chiave pubblica") e la successiva stringa di 280 caratteri esadecimali (1120 bit complessivamente) che la rappresenta.

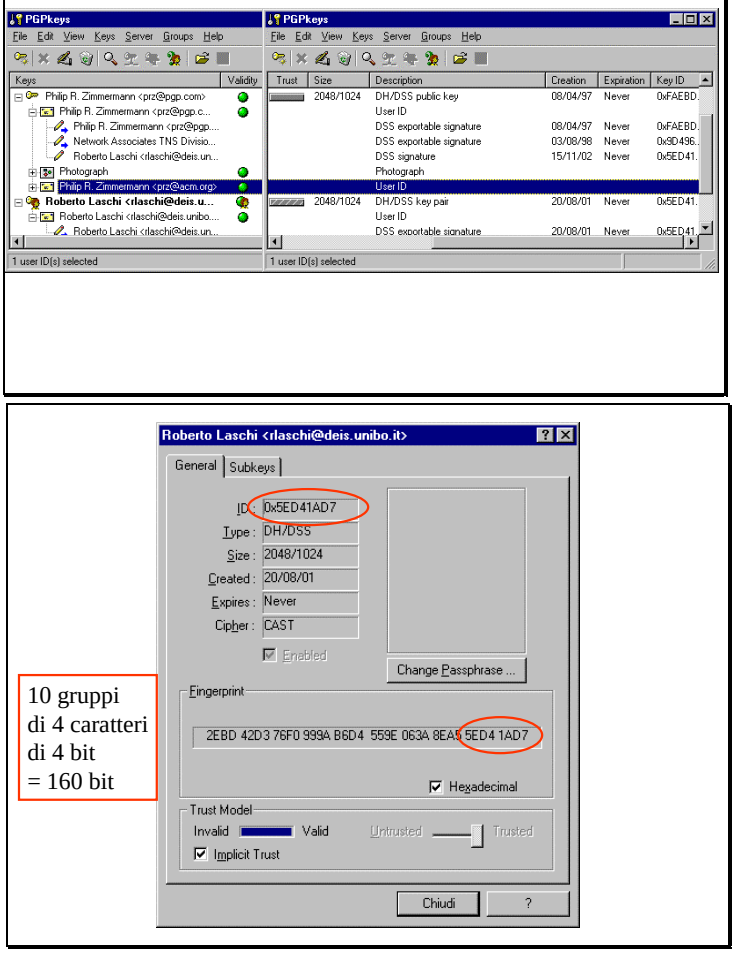
La giustificazione sarà data, quando prenderemo atto di come funziona questo algoritmo.

[46] Scaricare il PGP dal sito www.pgpi.com (la versione 7 è free), aprire il file della documentazione e leggere il capitolo 2 dell'introduzione.

2 - Il certificato del PGP

Anche il certificato del PGP ha diversi campi in chiaro, seguiti dalla firma del proprietario (autocertificazione) ed, eventualmente, di altri utenti. Per prendere atto della struttura di questi dati è sufficiente esaminare una riga del nostro portachiavi pubblico (a tal fine si deve selezionare **PGPkeys**).

PGPkeys: un certificato PGP



10 gruppi di 4 caratteri di 4 bit = 160 bit

Nel campo **Keys** di ogni chiave è visibile la descrizione del proprietario e di chi l'ha autenticata; nei successivi campi **Validity**, **Trust**, **Size**, **Description**, **Espiration** sono indicati rispettivamente il livello di fiducia sul proprietario, il livello di fiducia sull'autenticatore, la dimensione in bit della chiave, la sigla dell'algoritmo che la impiega, la data di creazione e quella di scadenza.

Nel campo **Key ID** appaiono infine, in esadecimale, i 32 bit meno significativi dei 160 bit dell'impronta della chiave.

Tramite un comando a menù è possibile visualizzare un riassunto delle caratteristiche della chiave, l'intera impronta (in esadecimale o come sequenza di frasi da pronunciare per una comunicazione vocale) e, se esiste, la fotografia del proprietario.

Ad ogni nuova chiave pubblica che un utente del PGP inserisce nel suo portachiavi è automaticamente attribuito **un livello di fiducia** (il campo **Trust**) basato appunto sull'elenco dei firmatari.

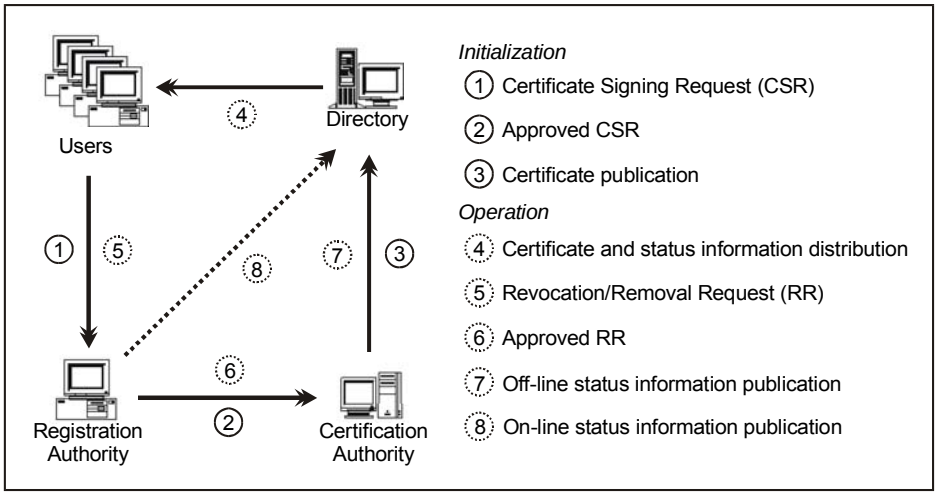
Se almeno uno dei firmatari è fidato (cioè se la sua chiave è stata in precedenza firmata dal proprietario del portachiavi) il livello di fiducia della nuova chiave è automaticamente portato al massimo livello.

L'utente può, se vuole, firmare la chiave ricevuta ed inviare poi la sua certificazione al proprietario della chiave e/o ad un database di chiavi PGP.

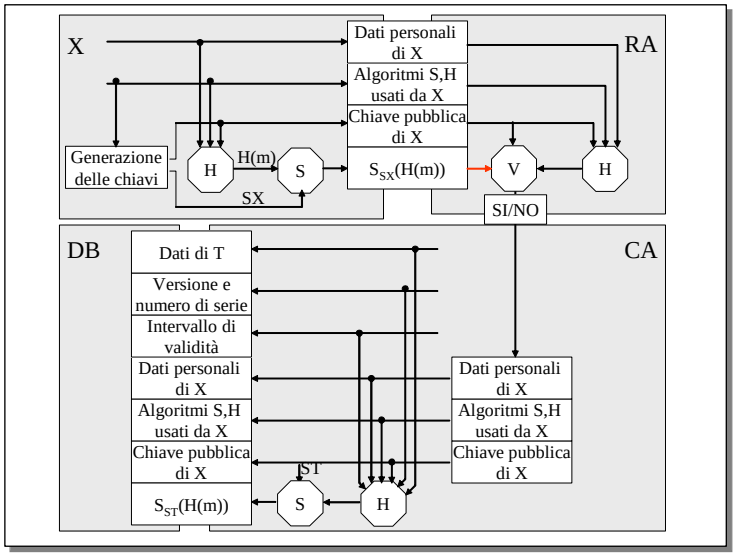
Da questa spontanea collaborazione tra gli utenti discende una **rete di fiducia**, che consente di trasferire da una chiave all'altra la confidenza sulla sua autenticità.

5.3.3 L'Autorità di certificazione

La struttura di un Ente che emette certificati è in generale formata da tre parti: la **Registration Authority**, la **Certification Authority** e la **Directory**.



1 - Richiesta ed emissione del certificato (attività 1, 2, 3, 4)



Supponiamo che **X** voglia farsi certificare una chiave di firma. Come prima cosa (in alto a sinistra) genera un autocertificato della sua **PX**, per dimostrare di essere il proprietario di **SX**.

A questo punto (in alto a destra) **X** fornisce a **RA** i suoi dati (tramite un'*e-mail* o una *form*, quando è richiesto un basso livello di sicurezza, presentandosi invece con documenti autenticati, quando il livello di sicurezza deve essere alto). **RA** verifica la firma e, in caso positivo, passa i dati a **CA**.

Per sicurezza la **CA** (in basso in figura) non è in rete e comunica con **RA** tramite un canale sicuro. **CA** organizza secondo lo standard i dati raccolti da **RA**, aggiunge quelli di sua competenza, calcola l'hash del tutto e lo firma.

Una prima copia del certificato è consegnata a **X** ed una seconda al *repository* **DB** accessibile dalla rete: si noti che un database di questo tipo non richiede né **controllo degli accessi**, né alcuna particolare **forma di protezione dei dati** in memoria (LDAP è il tipico protocollo d'accesso a DB).

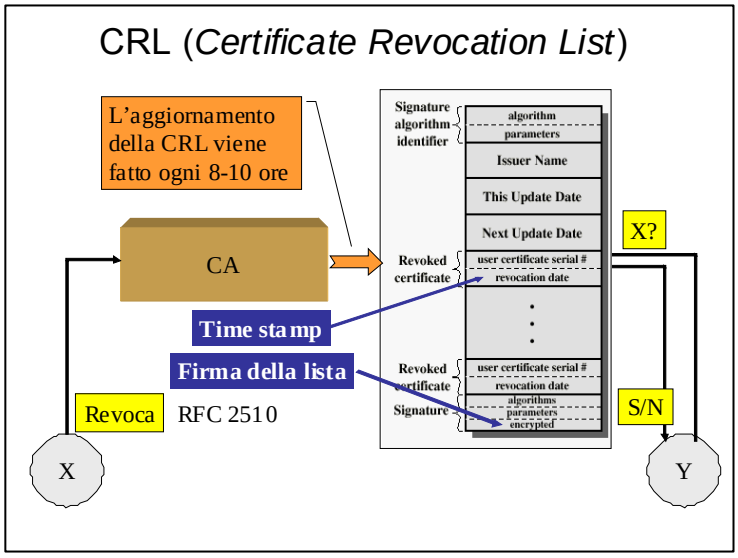
L'intervallo di validità del certificato è tipicamente è **un anno**: prima della scadenza, l'utente ne deve richiedere il rinnovo.

2 - Ripudio della chiave pubblica e revoca del suo certificato (attività 5, 6, 7, 8)

Su tutti i meccanismi basati su un segreto incombe il pericolo che qualcuno riesca o ad indovinarlo o a rubarlo. La difesa che gli utenti devono adottare è espressa nella seguente regola.

- R29: "*quando uno ha il sospetto che la sua chiave privata sia stata violata, deve rinunciare immediatamente ad impiegarla, generarsene una nuova e farsi certificare la corrispondente chiave pubblica*".

Può essere necessario rinunciare ad usare una chiave, anche se non è stata compromessa: ciò capita quando l'utente, per una qualsiasi ragione, non ha più lo "stato" o il "ruolo" descritto nel certificato.



Il **ripudio** è l'azione con cui un utente, prima della scadenza, spezza l'associazione chiave pubblica/proprietario sancita da un certificato. E' indispensabile che questo evento sia **tempestivamente** notificato a tutti gli altri utenti.

La raccolta delle notifiche di ripudio è fatta dalla **RA**; alla **CA** compete la successiva azione di **revoca del certificato**.

Per rendere edotti tutti gli altri utenti che una chiave non deve essere più usata, **CA** mantiene *on line* una lista **autenticata** dei certificati revocati (**CRL**).

Chiunque è dunque così in grado di sapere se è ancora valida la chiave pubblica di chiunque altro. Fino a poco tempo fa occorreva reperire la CRL (dove è indicato nel certificato), scaricarla e poi interrogarla; ora alcune CA supportano *query* dirette.

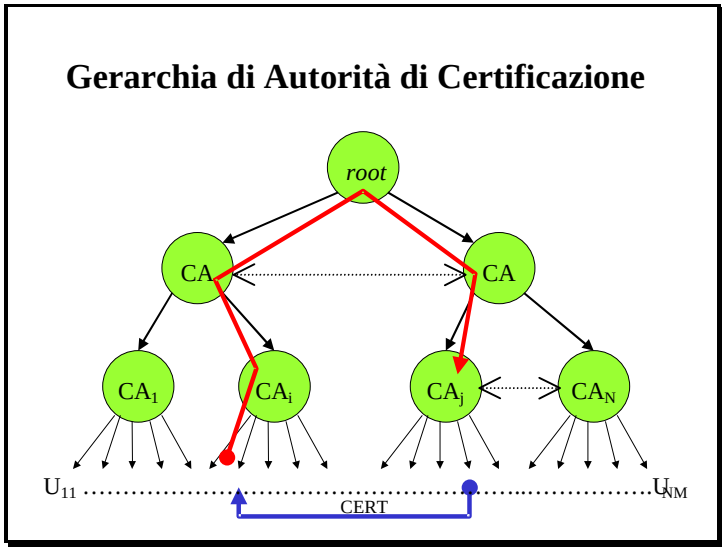
Presupposto essenziale è che a CA non passi mai per la testa l'idea di fare "l'uomo in mezzo".

ESEMPI – **Verisign** è una delle Autorità di certificazione più note negli Stati Uniti. I certificati dei docenti della nostra Università sono emessi dalla S.p.A. **Actalis**. Altre Autorità italiane sono **Trustitalia** e **E-trustcom.intesa**.

L'insieme formato da un'autorità di certificazione, da un'autorità di registrazione e dai depositi per i certificati e per le revoche è detto **PKI** (*Public Key Infrastructure*): **RFC 2510** fornisce le specifiche delle diverse funzionalità. Il software (open e free) per una **PKI** è reperibile in Internet; meritano una visita i progetti **OpenCA** (www.openca.org), **IDX-PKI** (http: //idx-pki.idealx.org) e **OpenSSL** (www.openssl.org).

3- La gerarchia delle Autorità di certificazione

Sicuramente non può bastare un unico **certificatore** per tutti gli utenti del mondo. Lo standard X.509 prevede una **gerarchia di Autorità**.



Alla base della piramide si trovano le Autorità che certificano ciascuna le chiavi di un certo numero di utenti.

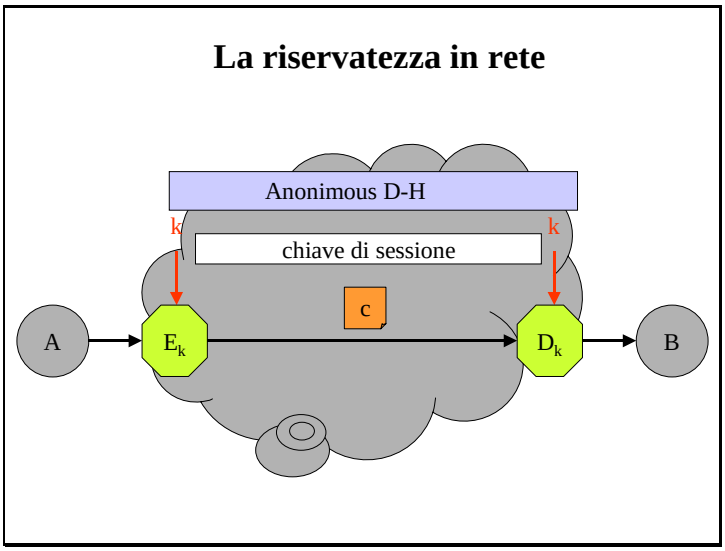
Al di sopra si trovano Autorità che certificano le chiavi di Autorità sottostanti e spesso anche di Autorità dello stesso livello.

Nel punto più alto si trovano le Autorità **radice**, che si autocertificano, essendoci le condizioni per considerarle fidate.

Risalendo la gerarchia e controllando la **catena dei certificati** un utente di una $\mathbf{CA_i}$ può verificare il certificato di un altro, anche se è stato emesso da una $\mathbf{CA_j}$ diversa dalla sua. Una seconda possibilità è che ognuno metta a disposizione degli altri la catena di certificati da usare per avere fiducia sulla autenticità della sua chiave.

ESEMPIO – In Italia il DPR 513/97 ha assegnato all'**AIPA,** oggi **cnipa.gov.it**, il ruolo di Autorità radice.

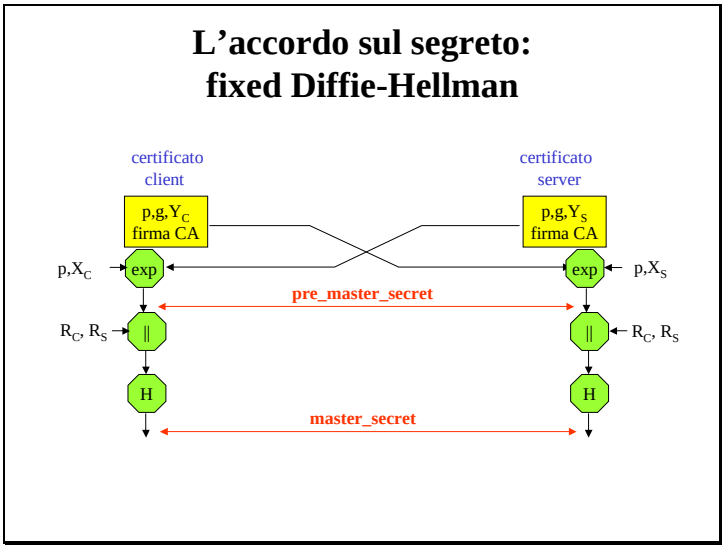
5.3.4 Sicurezza in rete: l'uso dei certificati



Per difendere la riservatezza delle loro comunicazioni in rete è usuale che i due corrispondenti impieghino un Cifrario simmetrico, dopo aver preliminarmente concordato una chiave di sessione con il protocollo DH.

Il tutto è naturalmente affidato alla suite dei protocolli di rete per renderlo "trasparente" agli utenti ed alle applicazioni.

In figura è indicato l'uso dello **scambio anonimo**, ma questa è solo una tra diverse possibilità previste ad esempio nel protocollo **SSL/TLS**. Questo protocollo, che vedremo meglio nel cap. 6, supporta tipicamente applicazioni *client*/*server*, attribuisce ad un suo protocollo interno (l'*Handshake*) l'esecuzione dei calcoli necessari per la condivisione del segreto e può avvalersi di certificati per rendere **non anonimo** lo scambio.

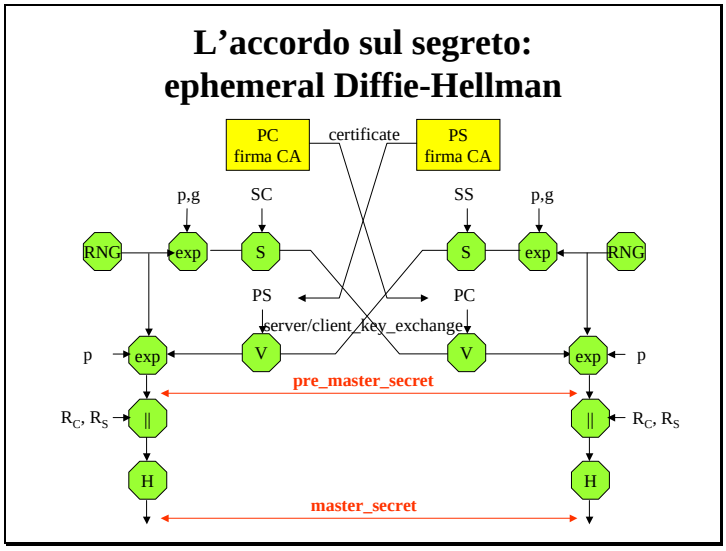


La modalità detta **fixed DH** prevede che il dato pubblico <**p**, **g**, **Y**> di ciascun utente sia comunicato all'altro tramite un certificato **X.509v3**. Una volta ricevuto questo dato, entrambi calcolano l'esponenziazione modulare, in modo del tutto analogo a quello dell'anonimous DH.

Il **pre_master_secret** è dunque sempre lo stesso per ogni coppia di utenti.

Per non dover usare sempre lo stesso segreto, i due partecipanti si scambiano anche due numeri a caso $\mathbf{R_C}$, $\mathbf{R_S}$, che consentono di variare ad ogni sessione il **master_secret**.

La funzione hash disposta a valle della concatenazione consente di trasmettere in chiaro i due nonce.



La modalità detta **ephemeral DH** prevede che, all'inizio di ogni sessione di comunicazione, i due partecipanti

- scelgano a caso il loro dato privato X,
- calcolino il dato pubblico Y,
- lo autentichino firmandolo con una chiave privata,
- lo comunichino al corrispondente unitamente ad un certificato della loro chiave pubblica.

In ricezione è così possibile verificare l'integrità e l'origine di Y.

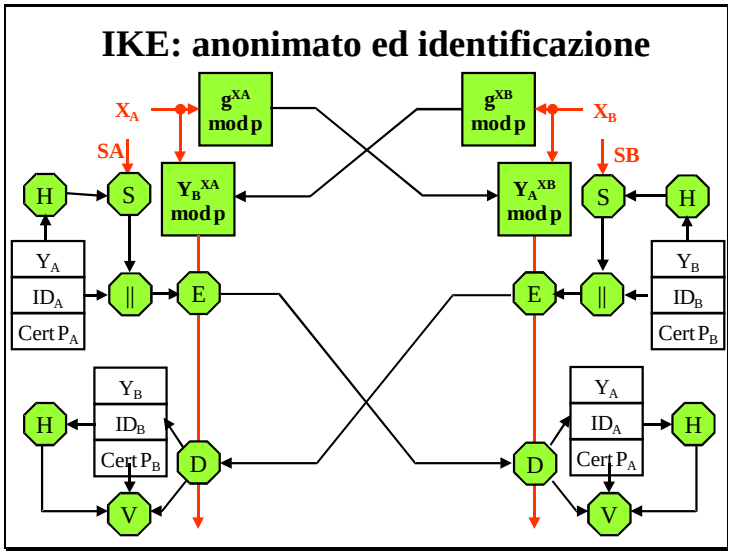
Il valore del pre_ master_secret varia inoltre da una sessione all'altra. Per mantenere compatibilità con la modalità "fixed", anche in questo caso si ha uno scambio di nonce ed un loro impiego nel calcolo del master_secret.

Un'ulteriore ed interessante modalità di accordo è prevista nell'architettura **IPSec**.

Abbiamo già detto nel cap.1 che l'obiettivo di IPSec è l'inserimento dei meccanismi di sicurezza a livello IP; come lo vedremo più in dettaglio nel cap.6, ma è utile anticipare qui la modalità con cui viene istituito un canale sicuro tra due estremità della rete.

Il compito è affidato al protocollo **IKE** (*Internet Key Exchange*, RFC 2409) che intende garantire ai due partecipanti sia l'**identificazione mutua**, sia l'**anonimato** a fronte della minaccia che qualcuno intercetti i messaggi contenenti la loro identità.

IKE prevede due fasi di scambio. All'inizio della prima fase avviene la **negoziazione**: i due corrispondenti, scambiandosi due pacchetti identificati da una loro peculiare intestazione, si accordano sugli algoritmi per la sicurezza che impiegheranno (Cifrario, Firma, Hash, DH anonimous/fixed/ephemeral) e sui parametri pubblici da utilizzare nei calcoli.



Successivamente (v. figura), i due nodi impiegano altri due pacchetti per comunicarsi due nonce e per arrivare a condividere una chiave di sessione k con uno **scambio anonimo DH**.

A questo punto i nodi si avvalgono della chiave k per comunicarsi, in modo **cifrato** e con altri due pacchetti, tutte le informazioni atte ad autenticare lo scambio precedente.

A tal fine entrambi firmano

- il parametro pubblico Y precedentemente impiegato,
- il loro identificativo ID (l'indirizzo IP o il nome dell'host),
- il certificato che autentica la loro chiave pubblica di firma.

Al termine dello scambio i due nodi decifrano con la chiave k e verificano l'autenticità del messaggio impiegando la chiave pubblica contenuta nel certificato che hanno appena ricevuto.

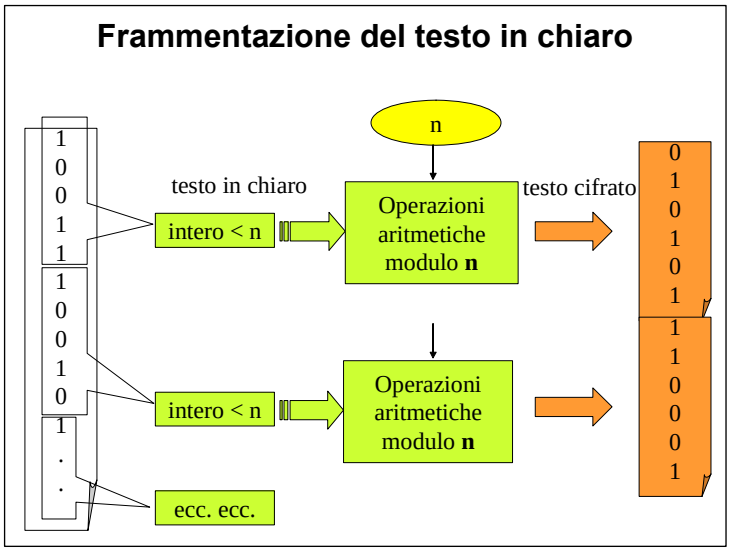
Gli utenti non interessati all'anonimato possono avvalersi di un secondo modo di esecuzione della prima fase, detto **aggressivo**, che richiede la trasmissione di soli tre pacchetti, sfruttando il fatto che identificativi e certificati possono essere comunicati in chiaro.

Nella seconda fase i due nodi si avvalgono della chiave segreta concordata in precedenza per avviare una nuova fase di negoziazione e per calcolare due nuove chiavi, che impiegheranno per cifrare e/o autenticare il traffico nell'una e nell'altra direzione.

5.4 Cifrari asimmetrici

In un Cifrario asimmetrico si deve cifrare il testo in chiaro con la **chiave pubblica del destinatario**; la corrispondente **chiave privata** serve per decifrare. Prima di esaminare i casi più importanti è utile mettere in discussione gli aspetti che portano a classificare tali meccanismi in differenti categorie.

1 – Cifrari a blocchi ed a flusso



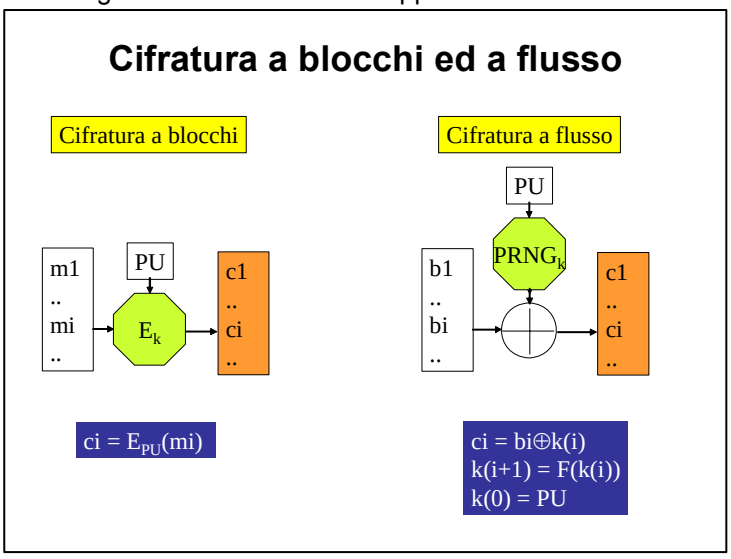
I Cifrari asimmetrici a blocchi devono suddividere i bit della stringa che rappresenta un messaggio **M** in pacchetti, atti ciascuno a rappresentare un numero binario intero minore del modulo **n**, un grande numero primo o composto, impiegato nelle operazioni della aritmetica modulare a cui verranno sottoposti:
$\mathbf{M = m_1 || m_2 || \ldots || m_k}$ con $\mathbf{m_i \in Z_n = \{0,..,n-1\}}$.

La funzione ***E*** trasforma un blocco dietro l'altro generando, di volta in volta testi cifrati corrispondenti a numeri minori di **n**.

Con queste ipotesi, la funzione ***D*** è in grado di ripristinare **in modo univoco** il testo in chiaro di ogni blocco. Al termine i blocchi vengono affiancati per restituire **M**.

Nei due casi che discuteremo, il Cifrario RSA ed il Cifrario di ElGamal, cifratura e decifrazione sono esponenziazioni modulari: la complessità del calcolo di $\mathbf{E_{PU}(m_i)}$ e di $\mathbf{D_{SU}(c_i)}$ è dunque $\mathbf{O((\log n)^3)}$ e la velocità operativa è di conseguenza bassa (da qualche decina a qualche centinaio di Kbit/sec).

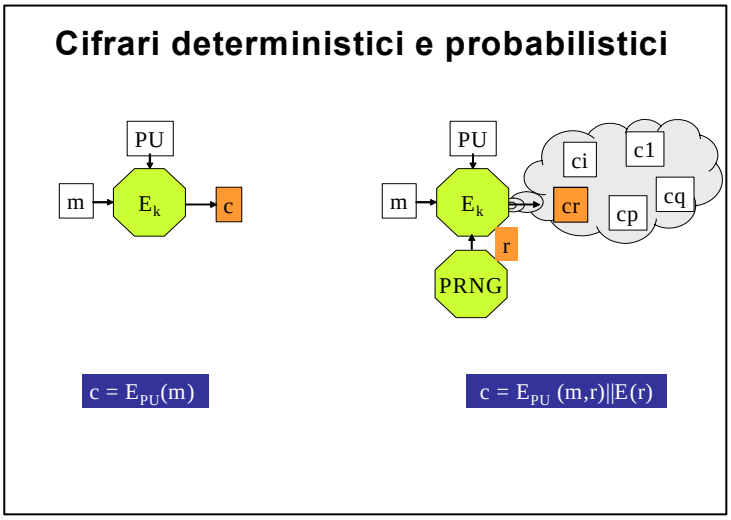
Un Cifrario asimmetrico è almeno **mille volte più lento** di un Cifrario simmetrico ed è quindi fortemente auspicabile impiegarlo con messaggi "corti". Ciò non ne limita l'utilità, dato che il suo uso tipico è la comunicazione della **chiave di un Cifrario simmetrico** (v. 5.4), cioè di qualche centinaia di bit a fronte delle migliaia di bit che devono rappresentare il modulo.



Esistono anche Cifrari asimmetrici **a flusso**, in cui le trasformazioni del testo in chiaro e del testo cifrato sono semplici operazioni di somma modulo due su uno o su pochi bit. Per rendere tali trasformazioni variabili al progredire del testo, occorrono, come sappiamo dalla Crittografia simmetrica, due PRNG mantenuti in perfetto sincronismo.

Nella Crittografia asimmetrica i bit di chiave sono generati eseguendo moltiplicazioni ed esponenziazioni modulari e quindi, anche in questo, con una velocità operativa relativamente bassa. L'aspetto interessante è che, nota PU, il mittente può scegliere a caso un seed per il suo generatore; il destinatario, tramite SU, è l'unico in grado di individuare quale seed è stato usato, cosa che gli consentirà di sincronizzare il suo generatore.

2 – Cifrari deterministici e probabilistici



Un Cifrario asimmetrico è detto

- **deterministico** se, una volta fissati la chiave di cifratura ed il testo in chiaro, il testo cifrato è unico,
- **probabilistico** se, a parità di chiave e di testo in chiaro, il testo cifrato è scelto di volta in volta tra molti possibili da un numero generato a caso durante il processo di cifratura.

E' giusto chiedersi subito se sono "migliori" i primi od i secondi.

Se si considera la sola **efficienza** potrebbero sembrare migliori i Cifrari **deterministici**.

Per consentire una corretta decifrazione, i Cifrari probabilistici devono, infatti, cifrare e decifrare una stringa più lunga di

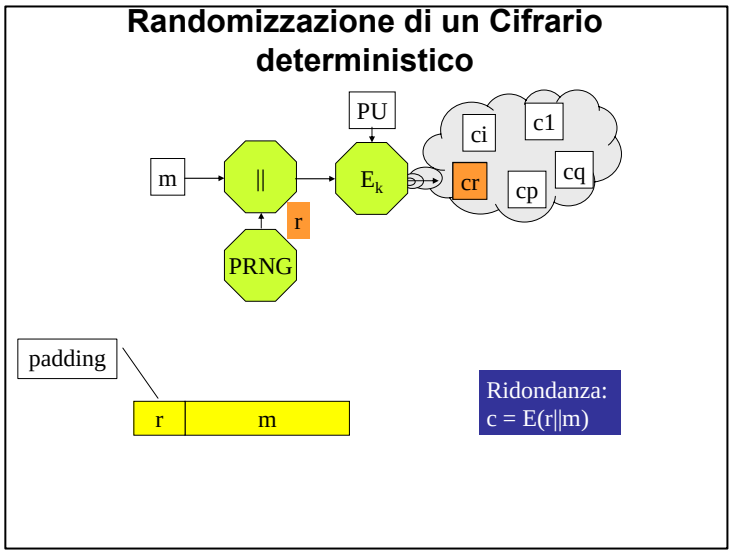
quella del messaggio, dato che il testo cifrato deve contenere anche il valore del numero a caso impiegato durante la cifratura.

Se si mette invece in conto anche la **sicurezza** sono di gran lunga da preferire i Cifrari **probabilistici**. I Cifrari deterministici operano, infatti, una sostituzione monoalfabetica ed hanno quindi due vulnerabilità:

1. **testi in chiaro identici generano testi cifrato identici**,
2. **in certi casi il testo cifrato può risultare identico al testo in chiaro**.

La non segretezza della chiave pubblica consente inoltre a chiunque di fare agevolmente **attacchi con testo in chiaro noto**.

ESEMPIO – In un referendum bisogna rispondere o SI, o NO ed inviare il proprio voto cifrato con la chiave pubblica dell'Ente che eseguirà lo scrutinio. Se il Cifrario è deterministico, chi intercetta un testo cifrato riesce a violare la segretezza e l'anonimità del voto: gli basta cifrare le due possibili risposte.



Per eliminare questi punti deboli, il Cifrario deterministico deve essere "**randomizzato**" dall'utente.

Se il messaggio è più lungo del modulo, si impiega tipicamente la modalità **CBC** ed un **IV** casuale.

Se il messaggio è più corto del modulo, ed è questo il caso di maggiore interesse come è stato detto nella pagina precedente, il mittente, seguendo uno standard ben preciso indicatogli dal destinatario, gli dispone in testa un ***padding*** contenente un **numero a caso** e poi cifra il tutto.

Il destinatario decifra, controlla il formato del padding, e, se tutto va bene, trattiene il solo messaggio.

3 – Problemi difficili e standard d'impiego

Cifrari asimmetrici

Cifrario	Proprietà	Tipo	Problema
RSA	deterministico	a blocchi	P2, P3
ElGamal	probabilistico	a blocchi	P1
Rabin	deterministico	a blocchi	P3, P5
Golwasser-Micali	probabilistico	a flusso	P6
Blum-Goldwasser	probabilistico	a flusso	P3, P5
Chor-Rivest	deterministico	a blocchi	P4

La chiave privata non deve essere deducibile né dalla chiave pubblica, né dal testo cifrato che con essa è stato prodotto.

A tal fine sono stati impiegati tutti i problemi difficili che abbiamo discusso nel paragrafo 5.2.2, come si può notare nella tabella di figura, ove sono elencati i Cifrari asimmetrici più noti e gli aspetti per i quali differiscono.

Per concludere questa presentazione generale occorre mettere in evidenza un ultimo aspetto importantissimo nella realizzazione di tutti gli algoritmi asimmetrici.

Chi progetta meccanismi e servizi deve, infatti, rispettare regole ben precise per garantire l'interoperabilità dei suoi prodotti con quelli messi a punto da altri progettisti.

Molti sono gli standard attualmente in uso e molti gli Enti che li hanno promulgati.

In questa sede ci limiteremo a citare le tredici raccomandazioni d'uso date dalle Ditte che per prime hanno prodotto meccanismi asimmetrici. Tali standard sono individuati da un numero preceduto dalla sigla **PKCS** (*Public Key Cryptography Standards*).

Per consultarli, si può ad esempio accedere al sito www.rsasecurity.com, da cui è stata presa la seguente succinta presentazione[47].

[47] Inizialmente c'erano anche PKCS#2 e PKCS#4, ma poi sono stati inclusi in una successiva versione di PKCS#1. Una visita periodica al sito di rsasecurity.com è comunque indispensabile per chi vuole mantenersi aggiornato sulle tecnologie per la sicurezza; dal sito è possibile anche scaricare una raccolta molto istruttiva di domande/risposte sulle tecniche crittografiche (labs_faq.pdf).

PKCS # 1	The RSA encryption standard. This standard defines mechanisms for encrypting and signing data using the RSA public key system.
PKCS # 3	The Diffie-Hellman key-agreement standard. This defines the Diffie-Hellman key agreement protocol.
PKCS # 5	The password-based encryption standard (PBE). This describes a method to generate a Secret Key based on a password.
PKCS # 6	The extended-certificate syntax standard. This is currently being phased out in favor of X509.
PKCS # 7	The cryptographic message syntax standard. This defines a generic syntax for messages which have cryptography applied to it.
PKCS # 8	The private-key information syntax standard. This defines a method to store Private Key Information.
PKCS # 9	This defines selected attribute types for use in other PKCS standards.
PKCS # 10	The certification request syntax standard. This describes a syntax for certification requests.
PKCS # 11	The cryptographic token interface standard. This defines a technology independent programming interface for cryptographic devices such as smartcards.
PKCS # 12	The personal information exchange syntax standard. This describes a portable format for storage and transportation of user private keys, certificates etc.
PKCS # 13	The elliptic curve cryptography standard. This describes mechanisms to encrypt and sign data using elliptic curve cryptography.
PKCS # 14	This covers pseudo random number generation (PRNG). This is currently under active development.
PKCS # 15	The cryptographic token information format standard. This describes a standard for the format of cryptographic credentials stored on cryptographic tokens.

5.4.1 Il Cifrario RSA

RSA[48], il primo Cifrario asimmetrico ad essere stato individuato (1978), è deterministico, opera a blocchi ed è universalmente considerato sicuro, anche se ogni tanto salta fuori qualcuno che afferma, erroneamente, di poterlo facilmente violare (l'ultimo caso è capitato nel 2004, con grande clamore dei media, dato che RSA è anche il Crittosistema più usato per la firma digitale) [49].

1 – L'algoritmo *E*

La trasformazione operata sul testo in chiaro **m** è l'esponenziazione modulare:

$$\mathbf{c = m^e \bmod n}$$

ove $\mathbf{n = p \times q}$ è il prodotto di due distinti **numeri primi p** e **q** ed **e** è **coprimo** con $\Phi(\mathbf{n}) = \mathbf{(p-1) \times (q-1)}$.

N.B. – La precedente formulazione dell'algoritmo non è l'unica possibile. Per trattare valori più piccoli dell'esponente, si può anche imporre **e** coprimo con $\lambda\mathbf{(n) = mcm(p-1, q-1)= \Phi(n)/MCD(p-1,q-1)}$. Per rendere più veloci i calcoli di decifrazione (v. pag. 105) **n** può essere il prodotto di più di due primi.

La coppia di numeri **P = {e, n}** è quindi la **chiave pubblica del destinatario**. Per l'esecuzione dei calcoli si ricorre usualmente ad uno dei due algoritmi con *repeated square-and-multiply* (v. pag. 82).

2- Aritmetica modulo $\mathbf{n = p \times q}$, con p e q numeri primi

Per comprendere il funzionamento di RSA sono utili alcuni teoremi della Teoria dei numeri. Alla base di tutto c'è la generalizzazione, individuata da Eulero, del piccolo teorema di Fermat (v. T4, pag.80).

- **T11:** "per ogni **n** (N.B. non importa se primo o composto) e per ogni **x coprimo** con **n** si ha $\mathbf{x^{\Phi(n)} \equiv 1 \pmod n}$ ".

Corollari

T11.1: "se $\mathbf{n = p \times q}$ e se $\mathbf{r \equiv s \pmod{\Phi(n)}}$, allora $\mathbf{a^r \equiv a^s \pmod n}$ per ogni intero **a**".

COMMENTO – Per non complicare inutilmente i calcoli, in RSA si deve quindi scegliere un $\mathbf{e \leq \Phi(n)}$.

[48] La sigla discende dai cognomi degli autori (Rivest, Shamir e Adleman); gli iniziali "detrattori" dei cifrari asimmetrici sostenevano invece che discende da *Really Slow Algorithm*.

[49] Una osservazione scherzosa, ma non troppo: se mai uno ci fosse riuscito, è certo che non andrebbe in giro a raccontarlo.

T11.2: "per ogni **n = p×q**, con p e q **primi,** e per **ogni x > 0** si ha $\mathbf{x^{\Phi(n)+1} \equiv x \pmod n}$ ".

COMMENTO – Per T11.1 si ha anche $\mathbf{x^{k\,\Phi(n)+1} \equiv x \pmod n}$.

- **T12:** "per ogni **n**, $\mathbf{Z^*_n}$ e la **moltiplicazione mod n** formano un **gruppo moltiplicativo**; se **n = p×q** ed **e** è **coprimo con Φ(n)**, $\mathbf{m^e}$ **mod n**, calcolata per ogni $\mathbf{m \in Z_n}$, produce una permutazione di $\mathbf{Z_n}$".

COMMENTO – Per la decifrabilità di RSA, ad ogni testo cifrato deve corrispondere un solo testo in chiaro. Per ottenere una permutazione di $\mathbf{Z_n}$ deve dunque essere $\mathbf{e \in Z^*_{\Phi(n)}}$; è ovvio che il caso **e = 1** non interessa.

E' sicuramente utile rivedere le precedenti proprietà in un caso semplice[50].

ESEMPIO - Tabuliamo i risultati di $\mathbf{m^e}$ **mod n** per ogni **m, e < n** nel caso di **n = 3 × 11 = 33**.

e

m exp	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	26	27	28	29	30	31	32
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	4	8	16	32	31	29	25	17	1	2	4	8	16	32	31	29	25	17	1	2	4	8	16	32	31	29	25	17	1	2	4
3	3	9	27	15	12	3	9	27	15	12	3	9	27	15	12	3	9	27	15	12	3	9	27	15	12	3	9	27	15	12	3	9
4	4	16	31	25	1	4	16	31	25	1	4	16	31	25	1	4	16	31	25	1	4	16	31	25	1	4	16	31	25	1	4	16
5	5	25	26	31	23	16	14	4	20	1	5	25	26	31	23	16	14	4	20	1	5	25	26	31	23	16	14	4	20	1	5	25
6	6	3	18	9	21	27	30	15	24	12	6	3	18	9	21	27	30	15	24	12	6	3	18	9	21	27	30	15	24	12	6	3
7	7	16	13	25	10	4	28	31	19	1	7	16	13	25	10	4	28	31	19	1	7	16	13	25	10	4	28	31	19	1	7	16
8	8	31	17	4	32	25	2	16	29	1	8	31	17	4	32	25	2	16	29	1	8	31	17	4	32	25	2	16	29	1	8	31
9	9	15	3	27	12	9	15	3	27	12	9	15	3	27	12	9	15	3	27	12	9	15	3	27	12	9	15	3	27	12	9	15
10	10	1	10	1	10	1	10	1	10	1	10	1	10	1	10	1	10	1	10	1	10	1	10	1	10	1	10	1	10	1	10	1
11	11	22	11	22	11	22	11	22	11	22	11	22	11	22	11	22	11	22	11	22	11	22	11	22	11	22	11	22	11	22	11	22
12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
13	13	4	19	16	10	31	7	25	28	1	13	4	19	16	10	31	7	25	28	1	13	4	19	16	10	31	7	25	28	1	13	4
14	14	31	5	4	23	25	20	16	26	1	14	31	5	4	23	25	20	16	26	1	14	31	5	4	23	25	20	16	26	1	14	31
15	15	27	9	3	12	15	27	9	3	12	15	27	9	3	12	15	27	9	3	12	15	27	9	3	12	15	27	9	3	12	15	27
16	16	25	4	31	1	16	25	4	31	1	16	25	4	31	1	16	25	4	31	1	16	25	4	31	1	16	25	4	31	1	16	25
17	17	25	29	31	32	16	8	4	2	1	17	25	29	31	32	16	8	4	2	1	17	25	29	31	32	16	8	4	2	1	17	25
18	18	27	24	3	21	15	6	9	30	12	18	27	24	3	21	15	6	9	30	12	18	27	24	3	21	15	6	9	30	12	18	27
19	19	31	28	4	10	25	13	16	7	1	19	31	28	4	10	25	13	16	7	1	19	31	28	4	10	25	13	16	7	1	19	31
20	20	4	14	16	23	31	26	25	5	1	20	4	14	16	23	31	26	25	5	1	20	4	14	16	23	31	26	25	5	1	20	4
21	21	12	21	12	21	12	21	12	21	12	21	12	21	12	21	12	21	12	21	12	21	12	21	12	21	12	21	12	21	12	21	12
22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22
23	23	1	23	1	23	1	23	1	23	1	23	1	23	1	23	1	23	1	23	1	23	1	23	1	23	1	23	1	23	1	23	1
24	24	15	30	27	21	9	18	3	6	12	24	15	30	27	21	9	18	3	6	12	24	15	30	27	21	9	18	3	6	12	24	15
25	25	31	16	4	1	25	31	16	4	1	25	31	16	4	1	25	31	16	4	1	25	31	16	4	1	25	31	16	4	1	25	31
26	26	16	20	25	23	4	5	31	14	1	26	16	20	25	23	4	5	31	14	1	26	16	20	25	23	4	5	31	14	1	26	16
27	27	3	15	9	12	27	3	15	9	12	27	3	15	9	12	27	3	15	9	12	27	3	15	9	12	27	3	15	9	12	27	3
28	28	25	7	31	10	16	19	4	13	1	28	25	7	31	10	16	19	4	13	1	28	25	7	31	10	16	19	4	13	1	28	25
29	29	16	2	25	32	4	17	31	8	1	29	16	2	25	32	4	17	31	8	1	29	16	2	25	32	4	17	31	8	1	29	16
30	30	9	6	15	21	3	24	27	18	12	30	9	6	15	21	3	24	27	18	12	30	9	6	15	21	3	24	27	18	12	30	9
31	31	4	25	16	1	31	4	25	16	1	31	4	25	16	1	31	4	25	16	1	31	4	25	16	1	31	4	25	16	1	31	4
32	32	1	32	1	32	1	32	1	32	1	32	1	32	1	32	1	32	1	32	1	32	1	32	1	32	1	32	1	32	1	32	1

$\mathbf{m^e \bmod 33}$

Prendiamo atto che **Φ(33) = (3-1) × (11-1) = 20** ed evidenziamo alcune proprietà notevoli della tabella.

1. Le colonne da 21 a 32 (**e > Φ(33)**) sono identiche alle colonne da 1 a 12; se si proseguisse la tabulazione si potrebbe verificare che le colonne da 1 a 20 sono identiche a quelle da 21 a 40, da 41 a 60, ecc..
2. Solo nelle colonne 1,3,7,9,11,13,17,19 (tutti e soli i numeri minori di 20 e coprimi con esso) l'esponenziazione produce una permutazione degli elementi di $\mathbf{Z_n}$.
3. La colonna 20 (**e = Φ(33)**) fornisce risultato **1** se e solo se **m** è coprimo con **33**.
4. Nella colonna 21 (**e = Φ(33)+1**) si ha $\mathbf{m^{\Phi(33)+1}}$ **mod 33 = m** per ogni **m**.
5. Quanto evidenziato vale anche se ci si limita ad esaminare le colonne da 1 a 10; in questo esempio si ha infatti **λ(33) = Φ(33)/MCD(2,10) = 10**.

[50] Per fare verifiche su RSA con Mathematica, è possibile scaricare dal sito del corso RSA.nb e RivestShamirAdleman.nb

3 – L'algoritmo *D*

L'operazione inversa della cifratura RSA è l'**estrazione della radice e-esima modulo n del testo cifrato c**. Per eseguirla occorre saper risolvere il problema difficile P2, ma abbiamo già detto che il calcolo è attualmente impossibile se **n** è formato da almeno un migliaio di bit.

Il destinatario del messaggio riservato, cioè chi ha generato la chiave pubblica **P**, dispone di una **trapdoor**.

Sia **S = {d, n}** la sua chiave privata, con $\mathbf{d = e^{-1} \bmod \Phi(n)}$ **inverso moltiplicativo** di **e** modulo **Φ(n)**. Per ottenere il testo in chiaro gli è sufficiente eseguire un'esponenziazione; si ha, infatti:

$$\mathbf{c^d \bmod n = (m^e)^d \bmod n = m^{e\,d} \bmod n}.$$

ESEMPIO – Riprendiamo il caso **n = 33** e tabuliamo il **prodotto mod 20** tra numeri minori di **Φ(33)** e coprimi con esso.
Si noti ad esempio che **3** e **7** sono uno l'inverso moltiplicativo dell'altro modulo 20. Dalla tabella di pag. 97 si ottiene

m	$c = m^3 \bmod 33$	$c^7 \bmod 33$
1	1	1
2	8	2
.....	.....	.....
31	25	31
32	32	32

*	1	3	7	9	11	13	17	19
1	1	3	7	9	11	13	17	19
3	3	9	1	7	13	19	11	17
7	7	1	9	3	17	11	19	13
9	9	7	3	1	19	17	13	11
11	11	13	17	19	1	3	7	9
13	13	19	11	17	3	9	1	7
17	17	11	19	13	7	1	9	3
19	19	17	13	11	9	7	3	1

Il corretto funzionamento di RSA può anche essere verificato[51] formalmente.

Per n = p×q si ha: $\mathbf{\Phi(n) = \Phi(p \times q) = \Phi(p) \times \Phi(q) = (p-1) \times (q-1)}$.

Da T11.2 si ha: $\mathbf{m^{k\,\Phi(n)+1} = m^{k\,(p-1)(q-1)\,+1} \equiv m \pmod n}$ per ogni 0 < m < n.

Per la decifrabilità di ogni c deve essere: $\mathbf{c^d \bmod n = m^{e\,d} \bmod n = m}$ con $m \in Z_n$.

La precedente congruenza è vera se: $\mathbf{e\,d = k(p-1)(q-1)+1}$,
o anche $\mathbf{e\,d - k(p-1)(q-1) = 1}$ (*)

Un teorema, dovuto a Bézout, evidenzia che il massimo comune divisore di due numeri interi **a** e **b** può sempre essere espresso da una loro combinazione lineare a coefficienti interi **x** e **y**:

$$\mathbf{MCD(a, b) = x\,a + y\,b}.$$

La (*) equivale dunque a $\mathbf{MCD\,(e, \Phi(n)) = 1}$
ed operando mod Φ(n) anche a $\mathbf{e\,d \equiv 1 \pmod{\Phi(n)}}$

In altre parole il corretto funzionamento di RSA è garantito se e solo se il numero **e** è relativamente primo con **Φ(n)** ed il numero **d** è il suo inverso moltiplicativo in $\mathbf{Z^*_{\Phi(n)}}$.
Ma queste sono proprio le ipotesi da cui siamo partiti nella presentazione degli algoritmi E, D.

4 – L'algoritmo *G*

Ogni utente **U** di RSA deve eseguire il seguente algoritmo.

Generazione delle chiavi di RSA:
1. si generano a caso due grandi numeri primi **p**, **q** di circa egual dimensione
2. si calcola $\mathbf{n = p \times q}$
3. si calcola $\mathbf{\Phi(n) = \Phi(p) \times \Phi(q) = (p-1)(q-1)}$
4. si sceglie un intero **e**, $\mathbf{1 < e < \Phi(n)}$, tale che $\mathbf{MCD(\Phi(n), e) = 1}$
5. si calcola $\mathbf{d = e^{-1} \bmod \Phi(n)}$
6. si rende di dominio pubblico la chiave **{e, n}**
7. si impiega come privata la chiave **{d, n}**

Tutti questi calcoli hanno complessità polinomiale. Verifichiamolo in due casi.

CALCOLO DELL'ESPONENTE PUBBLICO **e**: occorre scegliere un **e** tale che $\mathbf{MCD(e, \Phi(n)) = 1}$.

L'**algoritmo di Euclide** è un modo efficiente per calcolare il massimo comun divisore di due interi **a** e **b**: la complessità è di $\mathbf{O((\lg n)^2)}$ operazioni su bit.

[51] Giulio C. Barozzi "I codici crittografici a chiave pubblica" Conferenza, Dallas, 2000.

MCD(a,b) o anche solo **(a,b)**
a, b interi positivi; a≥b>0
Finché b>0
calcola r = a mod b
poni a = b
poni b = r
Restituisci **a**

ESEMPIO: MCD(240,42) = 6

passo	a	b	r
1	240	42	30
2	42	30	12
3	30	12	6
4	12	6	0
	6		

Per il calcolo di **e** si può dunque scegliere a caso un **r < Φ(n)** e poi verificare se **MCD(r, Φ(n))** = **1**. In caso contrario si itera il procedimento. Di solito bastano poche prove, essendo

(*Prob*(MCD(r, Φ(n) =1) = Φ(Φ(n))/Φ(n).

Usualmente però l'esponente **e** viene **fissato** in modo da rendere semplice il calcolo di m^e mod n: a tal fine deve essere un numero binario con "pochi uni" [52]. Valori in uso sono **3** e $\mathbf{2^{16}+1}$ (ritenuto più sicuro).

CALCOLO DELL'ESPONENTE PRIVATO **d:** deve essere $\mathbf{d = e^{-1} \bmod \Phi(n)}$.

Dati due interi qualsiasi a, b il calcolo del loro massimo comun divisore e dei due coefficienti x,y della combinazione lineare che lo individua (th. di Bézout) può essere efficientemente fatto con l'algoritmo **esteso di Euclide**. La complessità è $\mathbf{O((\lg n)^2)}$ operazioni su bit:

MCD(a,b) = x a + y b
a, b interi positivi; a≥b>0
Poni $x_2 = 1, x_1 = 0, y_2 = 0, y_1 = 1$
Finché b>0
calcola $q = \lfloor a/b \rfloor$, r = a - qb
calcola $x = x_2 - qx_1$, $y = y_2 - qy_1$
poni a = b, b = r, $x_2 = x_1$, $x_1 = x$, $y_2 = y_1$, $y_1 = y$
Restituisci **MCD(a,b) = a, $\mathbf{x = x_2}$, $\mathbf{y = y_2}$**

ESEMPIO – MCD(240,42) = 6 = 3×240 – 17×42

passo	a	b	x_2	x_1	y_2	y_1	q	r	x	y
1	240	42	1	0	0	1	5	30	1	-5
2	42	30	0	1	1	-5	1	12	-1	6
3	30	12	1	-1	-5	6	2	6	3	-17
4	12	6	-1	3	6	-17	2	0		
	6		**3**		**-17**					

Nel caso particolare di RSA, occorre dunque soltanto fornire in ingresso a tale algoritmo **Φ(n), e**; in uscita si otterrà (**Φ(n)**,**e**) = 1 (ma lo si sapeva già), **d** (il coefficiente di **e**) ed un intero che non interessa.

5 - Decifrazione con il teorema cinese dei resti

Dal punto di vista della sicurezza, sarebbe molto opportuno che chi ha generato una coppia di chiavi RSA eliminasse ogni traccia dei due numeri primi impiegati.

Dal punto di vista dell'efficienza è invece utile che li conservi: esiste, infatti, un metodo, basato proprio sulla conoscenza di **p** e di **q**, che consente di ridurre di **circa quattro volte** il carico di lavoro della decifrazione.

Per giustificarlo è necessario richiamare il **teorema cinese dei resti**, particolarmente utile anche in altre elaborazioni richieste dalla Crittografia asimmetrica.

- **T13: "**se gli interi $\mathbf{n_1}$, $\mathbf{n_2}$,..$\mathbf{n_k}$ sono **coprimi** a due a due, allora il sistema di congruenze
  $\mathbf{x \equiv a_1 \pmod{n_1}}$,
  $\mathbf{x \equiv a_2 \pmod{n_2}}$,
  … ,
  $\mathbf{x \equiv a_k \pmod{n_k}}$

ha un'unica soluzione modulo $\mathbf{n = n_1 \times n_2 \times \ldots \times n_k}$ **".**

[52] In questi casi la coppia **e**,**n** può essere assegnata con una stringa binaria più lunga di quella che rappresenta il solo n, ma più corta di 2 log n: ecco la spiegazione dell'incongruenza segnalata a pag. 93.

Una conseguenza importante. Quando si deve elaborare un certo numero **y** è possibile
1. passare alla sua **rappresentazione modulare** $\mathbf{v(y) = (y \bmod n_1, y \bmod n_2, ..., y \bmod n_k)}$,
2. elaborare **separatamente** le k coordinate,
3. restituire un risultato con la **rappresentazione originaria**.

CASO DI STUDIO - Nella decifrazione di RSA, la nuova rappresentazione di **c** si ottiene impiegando come moduli p e q (primi e quindi anche coprimi): **v(c) = (c mod p, c mod q)**

In tale rappresentazione il calcolo di $\mathbf{c^d \bmod n}$ richiede due esponenziazioni modulari:

$$\mathbf{v(c^d) = ((c \bmod p)^d \bmod p, (c \bmod q)^d \bmod q)}$$

Il già citato vantaggio computazionale di procedere in questo modo è tutto nel fatto che i numeri da trattare sono di dimensione più piccola (½ log n). La complessità del calcolo è quindi $2(½ \log n)^3 = ¼ (\log n)^3$

Esistono due metodi per passare da una rappresentazione modulare **v(y)** alla rappresentazione originaria di **y**. Il primo, dovuto a Gauss, richiede di calcolare una combinazione lineare a coefficienti interi delle coordinare

CASO DI STUDIO **–** Indichiamo con **a** e con **b** i coefficienti che rimettono in chiaro ciò che è stato decifrato nella rappresentazione modulare. Deve dunque essere:

$$\mathbf{m = c^d \bmod n = \{a [(c \bmod p)^d \bmod p] + b [(c \bmod q)^d \bmod q]\} \bmod n}$$

I due coefficienti possono essere agevolmente calcolati con **T13** dovendo soddisfare le congruenze

$$\mathbf{a \equiv 1 \pmod p} \qquad \mathbf{b \equiv 0 \pmod p}$$
$$\mathbf{a \equiv 0 \pmod q} \qquad \mathbf{b \equiv 1 \pmod q}$$

ESEMPIO – Sia p = 3, q = 11 e n = 33. Per x = 17 si ha v(x) = (2,6).
Verifichiamo che a = 22, b = 12 soddisfano le congruenze indicate nel caso di studio:
22 mod 3 = 1, 22 mod 11 = 0, 12 mod 3 = 0, 12 mod 11 = 1.
Si ha inoltre: (22×2 + 12×6) mod 33 = 116 mod 33 = 17

Il passaggio dalla rappresentazione modulare alla rappresentazione originaria può essere oggi fatto in modo ancora più efficiente.

A**lgoritmo di Garner** - Sia **n = p×q**, con **p** e **q** primi
Sia inoltre assegnata la rappresentazione modulare di **y**

$$\mathbf{v(y) = (v1 = y \bmod p, v2 = y \bmod q)}, \text{ con } \mathbf{0 \le y < n}.$$

Il calcolo di **y** richiede tre passi:
1. $\mathbf{C = p^{-1} \bmod q}$
2. $\mathbf{u = ((v2 - v1) \times C) \bmod q}$
3. $\mathbf{y = v1 + u \times p}$

Formula di Garner – Per sostituzione si ha anche: $\mathbf{y = v1 + (((v2 - v1) (p^{-1} \bmod q) \bmod q) \times p}$

CASO DI STUDIO **–** Nelle implementazioni di RSA basate sul teorema cinese del resto (CRT) viene oggi usato l'algoritmo di Garner. Per rendere i calcoli efficienti, la chiave privata è formata da 5 componenti:
- **n**, il modulo dell'esponenziazione
- **p**, il primo fattore di n
- **q**, il secondo fattore di n
- $\mathbf{C = p^{-1} \bmod q}$, la costante chiamata in causa dall'algoritmo
- **d1= d mod (p-1)**, l'esponente più piccolo per il calcolo della prima coordinata (v. T4.1)
- **d2= d mod (q-1)**, l'esponente più piccolo per il calcolo della seconda coordinata

La decifrazione di un testo cifrato **c** richiede dunque i seguenti calcoli:

$\mathbf{v1 = c^{d1} \bmod p}$
$\mathbf{v2 = c^{d2} \bmod q}$
$\mathbf{u = ((v2 - v1) \times C) \bmod q}$
$\mathbf{m = v1 + u \times p}$

ESEMPIO – Sia n = 33, p = 3, q = 11, e = 3, d = 7.
I parametri della chiave privata sono: C = 4, d1 = 1, d2 = 7.
Per il testo in chiaro m =8 la cifratura ha generato il testo cifrato c = 17
In decifrazione si ha v1 = 2, v2 = 8, u = (6 x 4) mod q = 2 e quindi m = 2 + 2 x 3 = 8.

NOTA IMPORTANTE - Lo standard PKCS#1 v2 prevede che gli utenti possano anche usare **più di due primi** nel calcolo del modulo di RSA. Ciò risponde a due differenti esigenze:

- poter fare calcoli rispetto ad un modulo sempre più grande per fronteggiare la probabile evoluzione degli algoritmi di fattorizzazione e delle macchine che li eseguono,
- poter ottenere dall'uso del teorema cinese del resto un'efficienza ancora maggiore quando si deve o decifrare un messaggio riservato o, come vedremo, apporre la firma digitale ad un documento. L'algoritmo di Garner, con poche modifiche (v. [2], pag. 612), si applica infatti anche a questo caso.

6- Robustezza di RSA

Ipotesi fondamentale per la sicurezza di RSA è che il proprietario della chiave privata **S =** {**d, n**} tenga **segreti d** ed i fattori **p** e **q** di **n**.

Vediamo quali attacchi può condurre un avversario e quali contromisure si devono adottare per fronteggiarli.

6.1 - Chi conosce la sola chiave pubblica di un utente di RSA può individuare la corrispondente chiave privata fattorizzando **n** e calcolando poi $\Phi(n)$ e $d = e^{-1} \bmod \Phi(n)$. La presunta intrattabilità di P3 e la scelta di primi adeguatamente "grandi" rendono questo attacco computazionalmente impossibile.

Altri attacchi basati sulle proprietà matematiche di RSA hanno tutti mostrato la stessa complessità computazionale della fattorizzazione.

Sfruttare la conoscenza di un testo cifrato intercettato non fornisce alcun vantaggio computazionale.

ESEMPI – L'estrazione della radice e-esima ha la stessa complessità computazionale della fattorizzazione. Di pari complessità computazionale è il "**cycling attack**", che prevede di iterare la cifratura a partire da un testo intercettato **c** fino a quando non lo si riottene: se dopo **k** passi si ha $(..((c^e)^e)..^e) = c$, allora al passo precedente si ha **m**.

6.2 - RSA è un Cifrario **deterministico** ed ha quindi la vulnerabilità che a blocchi identici di testo in chiaro corrispondono blocchi identici di testo cifrato. Presenta inoltre una ulteriore vulnerabilità: per alcuni testi in chiaro (in realtà sono pochissimi e ben noti), si ha $m \equiv m^e \pmod n$.

ESEMPIO - Per e = 3 (v. tabella di pag. 101) sono "non nascosti" i messaggi 1, 10, 11, 12, 21, 22, 23, 32.
Gli stessi messaggi sono non nascosti anche per e = 7.

La contromisura è, come si è detto a pag. 99, quella di rendere RSA **probabilistico**. A tal fine gli utenti, prima della cifratura, devono concatenare un numero a caso al testo in chiaro; quest'ultimo deve quindi avere una dimensione adeguatamente inferiore a quella del modulo.

PKCS#1 prevede due soluzioni che riassumiamo qui succintamente[53].

La prima premette a **m** un padding di 88 bit:

0x**00 ||** 0x**02 || r ||** 0x**00** || **m** , con **r** stringa di 64 bit casuali.

La seconda, detta OAEP (*Optimal Asymmetric Encryption Padding*) è decisamente più complessa, ma è anche considerata più sicura; il padding è ottenuto concatenando al prefisso 00 due dati casuali ottenuti sottoponendo **m** e **r** a funzioni hash ed a somme modulo due:

0x**00** || **maskedSeed** || **maskedDB**

6.3 - RSA può essere rotto con un attacco "con testo cifrato scelto". In questo contesto si suppone che l'intruso possa richiedere al proprietario della chiave privata la decifrazione di qualsiasi messaggio di sua scelta, ad esclusione del messaggio **c** di suo interesse.

L'attacco si basa sulla **proprietà moltiplicativa** di RSA. Consideriamo un messaggio $m = m_1 \times m_2$. La sua cifratura è:

$$c = m^e \bmod n = ((m_1^e \bmod n) \times (m_2^e \bmod n)) \bmod n$$

La cifratura del prodotto di due messaggi è uguale al prodotto dei due testi cifrati. Analogamente si ha che **la decifrazione di un testo cifrato ottenuto moltiplicando due testi cifrati è uguale al prodotto dei due corrispondenti testi in chiaro.**

Un intruso che ha intercettato **c** può dunque generare un numero a caso **r**, costruire $c1 = c \times r^e$, calcolare $m2 = r^{-1}$, ottenere la decifrazione **m1** di **c1** e moltiplicare infine **m1** e **m2**.

Pur non avendo inviato **c**, l'intruso ha così ottenuto la decifrazione che gli interessa!

[53] Per maggiori dettagli, è sufficiente consultare lo standard.

Un caso particolare, detto *side-channel attack*, si ha quando l'intruso può richiedere la decifrazione di messaggi di sua scelta ed ottenere informazioni sui calcoli che vengono eseguiti dal destinatario (ad esempio il tempo di esecuzione, o il consumo di potenza, o l'emissione di onde elettromagnetiche, o la generazione di avvisi sonori). In queste situazioni oggetto dell'attacco non è l'algoritmo crittografico, ma la sua implementazione.

A prima vista lo svolgimento di questi attacchi può sembrare estremamente difficile, ma non è così: basti pensare alla facilità con cui un intruso può misurare le radiazioni emesse dallo schermo di un PC, o il consumo di potenza di una smart card lasciata incustodita, o il tempo di risposta di un server che esegue un protocollo interattivo in situazioni di basso carico.

ESEMPIO - Il "**timing attack**" si basa sulla misura del tempo di esecuzione per dedurre il parametro segreto di un algoritmo crittografico[54]. Nel caso di RSA (o anche dello scambio DH) si sfrutta il fatto che l'esponenziazione modulare è di norma realizzata con uno degli algoritmi con *repeated square and multiply*: Ciò consente all'attaccante di discriminare due casi: se uno qualsiasi dei bit dell'esponente segreto è 0, l'algoritmo esegue solo un elevamento al quadrato, se è 1, viene eseguita anche una moltiplicazione. In seguito, in dipendenza dal risultato ottenuto e quindi dalla base dell'esponenziazione, venire eseguita o meno una riduzione in modulo.

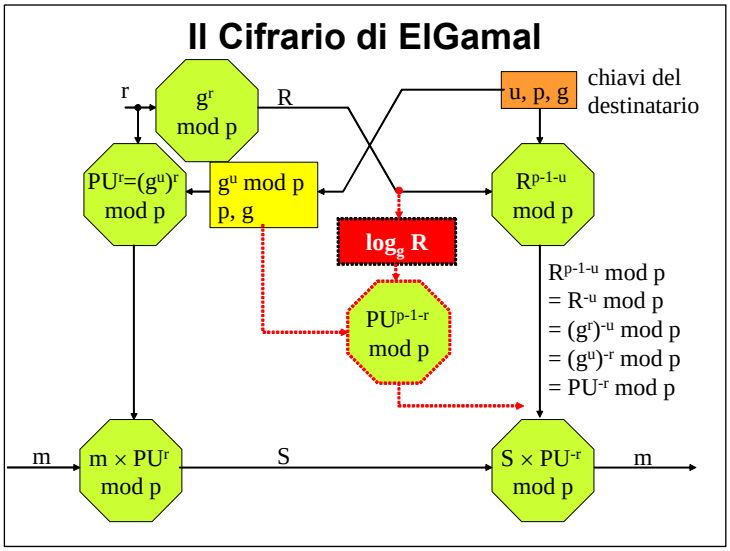
E' stato dimostrato che sulla base di queste differenze di comportamento è possibile preparare un certo numero di testi cifrati, richiederne la decifrazione ed usare la misura del tempo di esecuzione per ricostruire, bit a bit, l'intero esponente.

Ciò ha imposto di correre ai ripari. La contromisura attualmente più diffusa è quella di impiegare la proprietà moltiplicativa di RSA: prima della decifrazione si moltiplica **c** per $\mathbf{r^e}$, con **r** numero a caso; dopo la decifrazione si moltiplica il risultato per $\mathbf{r^{-1}}$.

5.4.2 Il Cifrario di ElGamal

Il secondo Cifrario a blocchi della Crittografia asimmetrica è stato proposto da **ElGamal** (1985) ed opera sugli elementi di un **qualsiasi gruppo ciclico**; in questa sede ci limitiamo alla formulazione sul campo **GF(p)**, indicando al solito con **g** il generatore impiegato come base delle esponenziazioni.

Algoritmo *G* - Ogni utente **U**, dopo aver scelto un numero primo **p**, un suo generatore **g** ed un numero a caso **u**, con $1 \le u \le p-2$, assume **SU = {u}** come sua chiave privata e **PU = {p, g, g^u mod p}** come sua chiave pubblica. Nel seguito, per semplificare le notazioni, porremo **PU = g^u mod p**, dando per scontato che chi impiega la chiave conosca già i valori di p e di g[55].



Algoritmo *E* - Chi possiede una copia autentica di PU e vuole mandare ad U un messaggio riservato **m**, con $\mathbf{m} \in \mathbf{Z_p}$, deve

1. eseguire la moltiplicazione modulare
   $$\mathbf{S = m \times (PU)^r \bmod p}$$
   in cui **r** è un numero a caso minore di **p**, scelto appositamente per il messaggio **m**,
2. eseguire l'esponenziazione modulare
   $$\mathbf{R = g^r \bmod p},$$
3. trasmettere **S||R**.

Algoritmo *D* - Per ripristinare **m** da **c** occorre moltiplicare **S** per l'inverso moltiplicativo di $\mathbf{(PU)^r}$. In tal caso si ha infatti:

$$\mathbf{S \times ((PU)^r)^{-1} \bmod p = m}.$$

Il calcolo dell'inverso moltiplicativo del risultato di un'esponenziazione discende da **T5**:

$$\mathbf{(PU)^{-r} \bmod p = (PU)^{p-1-r} \bmod p}.$$

Per fare questo calcolo occorre però conoscere **r**, cioè saper calcolare il logaritmo discreto di **R** rispetto alla base **g**, modulo **p**. L'intercettatore non può dunque farcela.

Per il legittimo destinatario è invece tutto facile: solo nel suo caso, infatti, il calcolo dell'inverso moltiplicativo di $\mathbf{(PU)^r}$ può essere fatto con una trapdoor, che richiede l'esponenziazione, ma non l'estrazione del logaritmo.

[54] P. Kocher "*Timing attacks on implementation ..* ecc." in http://www.cryptography.com/resources/whitepapers/TimingAttacks.pdf
[55] Con il Cifrario di ElGamal, come con lo scambio DH, **p** e **g** possono essere condivisi da una comunità di utenti.

Si ha infatti: $(PU)^{-r} \bmod p = g^{-u\,r} \bmod p = R^{-u} \bmod p = R^{p-1-SU} \bmod p$

Il legittimo destinatario, per rimettere in chiaro un testo cifrato ricevuto, deve calcolare[56]

1. $R^{p-1-SU} \bmod p$,
2. $S \times R^{p-1-SU} \bmod p$.

Lo svantaggio più grosso del Cifrario di ElGamal è quello di **espandere** il testo in chiaro di un fattore 2: ogni **m** di **log p** bit è trasformato nella coppia di testi cifrati **c**, **R** , ciascuno di **log p** bit.

Un vantaggio è che non è possibile capire dai testi cifrati se i corrispondenti testi in chiaro sono identici: l'uso del numero a caso **r** attribuisce infatti al Cifrario un comportamento **probabilistico**.

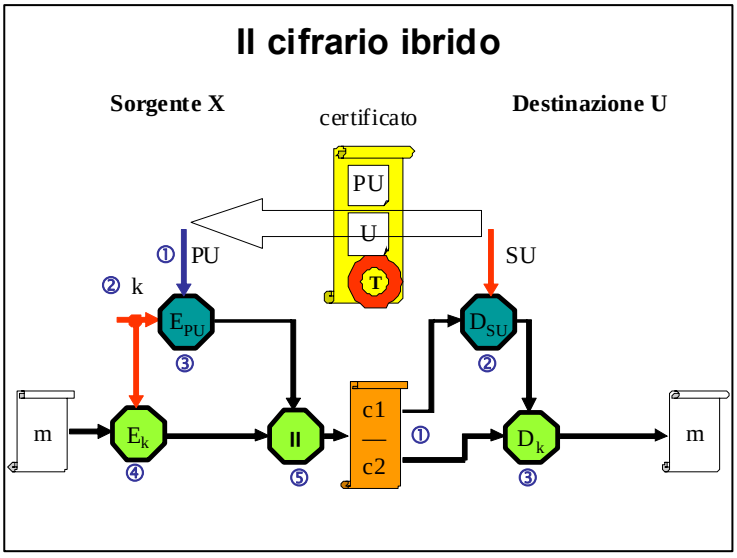
5.4.3 Il Cifrario ibrido

Due utenti che non hanno avuto in precedenza alcun accordo segreto possono comunicare in modo riservato (v. pag. 96) impiegando dapprima lo **scambio DH** per arrivare a condividere il valore di una **chiave segreta** ed usandola poi con un **Cifrario simmetrico** per cifrare e decifrare i loro messaggi.

Al posto dello scambio DH, si può usare un Cifrario asimmetrico. In questo caso si parla di Cifrario **ibrido**, dato che si devono impiegare due Cifrari di tipo diverso.

Supponiamo che un **X** debba mandare un messaggio riservato **m** ad un **U**, di cui conosce con sicurezza (tipicamente mediante un Certificato) la chiave pubblica **PU** ed il Cifrario asimmetrico **CA** per cui è stata generata: **CA** sarà dunque il Cifrario che X e U impiegheranno, il primo per cifrare, il secondo per decifrare la chiave di sessione.

Sia infine **CS** il Cifrario simmetrico che **X** decide di impiegare per cifrare **m**: sarà sua cura comunicare tale scelta a **U** insieme alla chiave di sessione.



In questo contesto **X**

1. sceglie a caso una chiave *one-time* **k**,
2. calcola $E_{PU}(k||CS)$ con **CA**,
3. calcola $E_k(m)$ con **CS**,
4. invia il messaggio $E_{PU}(k||CS)||E_k(m)$.

A sua volta **U**

1. separa le due parti,
2. usa **CA** per calcolare $D_{SU}(E_{PU}(k||CS))$ e venire così a conoscenza di **CS** e di **k**,
3. calcola $D_k(E_k(m)) = m$ con **CS**.

Si noti la similitudine tra questo schema e quello della master key (v. pag. 76): in entrambi i casi esiste una **chiave che cifra chiavi**. C'è però una grossa differenza: in questo caso la chiave che cifra chiavi (**PU**) è unica per tutti i corrispondenti di **U** e può essere loro comunicata senza alcuna forma di riservatezza.

Il ricorso al Cifrario ibrido può far sorgere un dubbio: **non sarebbe meglio inviare solo $E_{PU}(m)$?**

Apparentemente si, ma a scapito dell'**efficienza**: per l'inviolabilità del Cifrario asimmetrico le chiavi devono essere molto lunghe, ma questo rende poi molto onerosi i calcoli di cifratura e di decifrazione (bisogna quindi non esagerare mai nel dimensionamento delle chiavi).

La chiave di sessione **k** è invece sempre un messaggio "corto" (cioè più piccolo del modulo di RSA o di ElGamal) e cifrarla con un algoritmo simmetrico o asimmetrico non fa grande differenza.

Esiste però una vulnerabilità dei Cifrari asimmetrici, se li si impiega *tout court* per cifrare messaggi molto più corti del modulo: i crittogrammi generati ricadrebbero sempre, infatti, in uno spazio più piccolo di quello a disposizione e ciò renderebbe più agevole il lavoro di crittanalisi.

ESEMPIO – Se si usa e = 3 per migliorare l'efficienza di RSA, può capitare che $\mathbf{k^3}$ sia ancora minore di **n**. In tal caso è **facile** estrarre la **radice cubica** di **c**.

[56] Dal sito del corso si può scaricare il notebook ElGamal.nb preparato da Ginevra Cicconi.

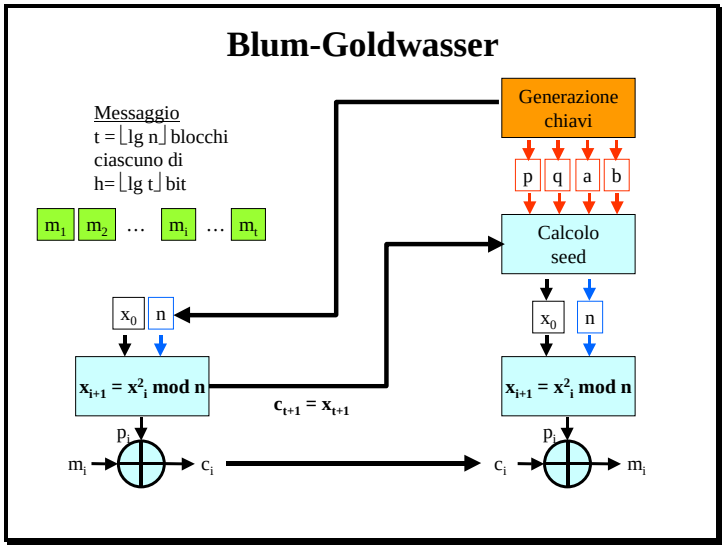
Prima di cifrare la chiave di sessione, occorre dunque affiancarle un padding casuale che porti la lunghezza complessiva a $\lfloor \log_2 n \rfloor$. Se si usa RSA, lo standard OAEP di **PKCS#1v2** (v. pag. 103) è idoneo allo scopo.

In [4, pag. 237] è proposta un'altra interessante soluzione, valida per qualsiasi Cifrario asimmetrico:

1. si sceglie a caso un **r** di lunghezza $\lfloor \log_2 n \rfloor$,
2. si invia $\mathbf{E_{PU}(r)}$,
3. si impiega **H(r)** come chiave.

5.4.4 Il Cifrario a flusso di Blum-Goldwasser

Per completare questa succinta presentazione dei Cifrari asimmetrici, è istruttivo esaminare anche quello a flusso inventato da Blum e da Goldwasser nel 1984: è infatti da tutti considerato il più efficiente schema probabilistico in termini di velocità e di espansione del testo in chiaro.



Ogni utente **U** del Cifrario a flusso di Blum-Goldwasser[57] rende pubblico un suo numero **n**, prodotto di due grandi e diversi numeri primi **p**, **q**, entrambi congruenti a 3 mod 4, e mantiene segreta la chiave {**p,q,a,b**} in cui **a**,**b** sono stati calcolati con l'algoritmo esteso di Euclide, dovendo essere

$$(ap + bq) \bmod \Phi(n) = 1.$$

Chi vuole inviare a **U** un messaggio **m**, sceglie **a caso** $\mathbf{x_0}$, uno dei residui quadratici modulo n, e genera altri **t** residui quadratici, con $\mathbf{t = \lfloor lg\ n \rfloor}$, iterando la funzione pseudo-unidirezionale

$$x_{i+1} = x_i^2 \bmod n.$$

La sequenza degli $\mathbf{x_i}$ è dunque quella di un generatore BBS. Il mittente suddivide inoltre il testo binario **m** in chiaro in **t** blocchi, ciascuno di $\mathbf{h = \lfloor lg\ t \rfloor}$ bit.

Ad ogni iterazione sono trasmessi i bit risultanti dalla somma modulo due degli **h** bit meno significativi di $\mathbf{x_i}$ e dei bit del blocco $\mathbf{m_i}$ (come in un Cifrario simmetrico a flusso sincrono).

Dopo **t+1** iterazioni si ottiene $\mathbf{x_{t+1}}$ e lo si trasmette al corrispondente. L'*espansione* del messaggio è dunque molto piccola: un testo in chiaro formato da **t** × **h** bit genera un testo cifrato di **t** × **(h+1)** bit.

La sicurezza si basa su **P3** e su **P5**: se **n** è formato da almeno un migliaio di bit, è computazionalmente impossibile per l'intruso o fattorizzare **n** o estrarre radici quadrate modulari con cui dedurre $\mathbf{x_0}$ da $\mathbf{x_{t+1}}$.

Il destinatario esegue invece solo calcoli facili: una volta individuato il *seed* $\mathbf{x_0}$ (chi vuol saper come veda ad es. [2], pag. 309) può, infatti, ricostruire il "flusso" di chiave con un generatore identico a quello usato in trasmissione ed impiegare **h** EX-OR per rimettere in chiaro i blocchi del testo cifrato.

5.5 Schemi di firma digitale

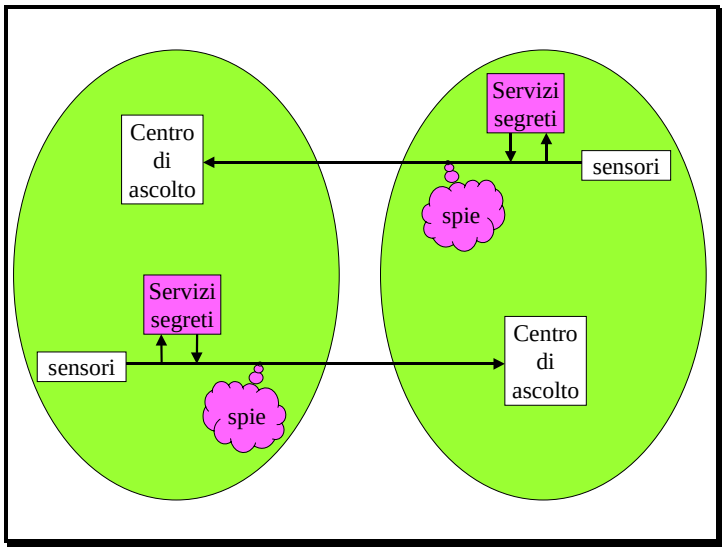
Nel cap. 4 abbiamo detto che il problema della firma digitale ha trovato la sua soluzione ottimale solo con l'avvento degli algoritmi asimmetrici.

Il grande pregio del meccanismo asimmetrico di firma è, infatti, quello di creare un crittogramma la cui origine può essere verificata con un dato reso di dominio pubblico dal firmatario: chiunque ha di conseguenza la possibilità di giudicare l'autenticità dei documenti prodotti da chiunque altro.

1 – Tecniche di autenticazione a confronto

Per sottolineare questa supremazia della firma con algoritmi asimmetrici vale la pena ricordare una storia curiosa (probabilmente un aneddoto): *prima di firmare un trattato che mettesse al bando gli esperimenti nucleari sotterranei, Russia e Stati Uniti concordarono che ciascuno installasse, nel territorio dell'altro, sia opportuni sensori, sia una linea di trasmissione su cui convogliare i dati di misura*.

[57] Dal sito del corso è possibile scaricare il notebook BlumGoldwasser.nb per fare delle verifiche.



Entrambi temevano due cose:

- *la **falsificazione dei dati** dei sensori,*
- ***l'uso improprio della linea** per l'invio di messaggi di spie.*

Per eliminare ogni sospetto si decise di chiamare in causa la Crittografia.

La cifratura simmetrica di data, luogo, misure e bit di controllo dell'integrità fu scartata subito: essa, infatti, consentiva ad uno dei due di verificare l'integrità e l'origine dei messaggi, ma non permetteva all'altro di verificare se la linea era usata anche per altri scopi.

La trasmissione in chiaro affiancata da un MAC fu considerata solo in parte soddisfacente: l'autenticità delle misure poteva essere verificata, ma rimaneva il sospetto che nel continuo flusso dei bit cifrati potessero starci anche messaggi illeciti. Per ridurre questa preoccupazione si decise di cambiare periodicamente la chiave e di comunicare all'altro il valore usato in precedenza: era così possibile verificare, se pure con lieve ritardo, tutto quello che era stato trasmesso.

*Solo con l'uso di messaggi in chiaro affiancati da hash firmati con un algoritmo asimmetrico entrambi poterono dichiararsi completamente soddisfatti: la disponibilità della chiave pubblica consentiva, infatti, il **controllo in tempo reale** di ogni bit cifrato trasmesso sulla linea.*

2 – Proprietà per la sicurezza di una firma digitale

Più importante degli aneddoti è l'avere ben chiaro lo scenario e le proprietà di sicurezza degli algoritmi asimmetrici di firma. L'uso del meccanismo richiede due attività, la prima svolta da un ben preciso utente **U** con la sua chiave privata **SU**, la seconda svolta da un qualsiasi altro utente **X** in possesso di una copia autentica di **PU**:

1. **U:** autentica il documento **m** calcolando **$c = S_{SU}(m)$** e rendendolo disponibile a chi è interessato;
2. **X:** verifica l'autenticità di **m** calcolando **$V_{PU}(c)$**, cosa che gli fornisce sia **m**, sia una risposta di tipo **vero/falso**.

Per la sicurezza occorrono due proprietà:

- per ogni **SU** e per ogni **m** la risposta di **$V_{PU}(S_{SU}(m))$** deve essere **"vero"**.
- per chiunque non abbia **SU** e per ogni **m** di sua invenzione deve essere computazionalmente infattibile trovare un **c** tale che la risposta di **$V_{PU}(c)$** è "**vero**".

L'ultima proprietà è essenziale per poter identificare con sicurezza l'origine di un documento; per ottenerla occorre naturalmente impiegare chiavi sufficientemente lunghe.

La firma asimmetrica di un documento elettronico crea dunque un attendibile e verificabile collegamento tra l'**informazione** ivi contenuta e la **persona** che l'ha fornita.

In realtà nel testo in chiaro è molto opportuno che ci siano anche l'**identità** ed il **ruolo** di chi ha apposto la firma, per consentire a chi riceve il documento di incrociare questi dati con quelli contenuti nel certificato.

Nel testo in chiaro deve inoltre essere esplicitamente indicato l'**istante temporale** in cui la firma è stata apposta. Il time-stamp deve essere **sicuro**: come renderlo tale sarà discusso nel paragrafo 6.1.3.

3 – Il ruolo del Giudice

Una **terza parte**, in possesso di una copia autentica di **V_{PU}** e cui è fornita una coppia **<m, c>**, può così dirimere due differenti dispute.

1. **X** sostiene che **m** è stato prodotto da **U**: se **$V_{PU}(c) = <m, SI>$**, **X** ha ragione, in caso contrario ha torto;
2. **U** nega di aver prodotto **m**: se **$V_{PU}(c) = <m, SI>$, U** ha torto, in caso contrario ha ragione.

Tanto meglio se la terza parte è un Tribunale, ma ciò richiede che ogni Stato legiferi in proposito.

ESEMPI - I primi ad attribuire valore legale alla firma digitale in contesti specifici sono stati gli USA (1989). Pochi mesi dopo un'analoga direttiva è stata data dalla Commissione ONU per le leggi sul commercio internazionale.

La legge n. 59/97 ed il successivo DPR n. 513/97 hanno disciplinato l'uso in Italia della firma digitale[58].

Due anni dopo la Comunità Europea ha emesso sull'argomento la direttiva 1999/93/CE.

Nel 2002 l'Italia ha approntato una norma (il testo è disponibile nel sito del corso) con cui ha recepito tale direttiva.

[58] Per una chiara presentazione della normativa italiana si veda P. Ridolfi "Firma elettronica: Tecniche, Norme, Applicazioni" Franco Angeli 2003

4 – Algoritmi di firma con recupero e con appendice

Algoritmi di firma a chiave pubblica

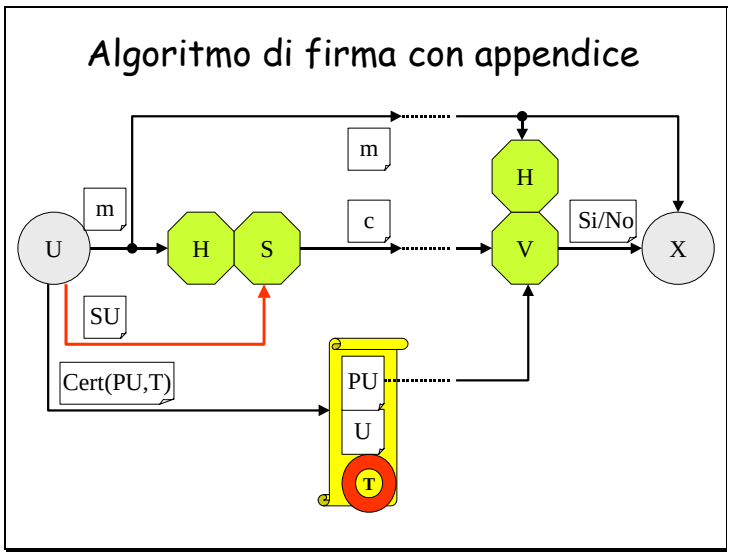
Nome	Tipo	Anno
RSA	ricupero	1978
Rabin	ricupero	1979
ElGamal	appendice	1984
DSA	appendice	1991
Schnorr	appendice	1991
Nyberg-Rueppel	ricupero	1993

Diversi sono gli algoritmi asimmetrici oggi a disposizione: alcuni sono **probabilistici**, altri **deterministici**. In figura sono elencati i più noti, unitamente ad una caratteristica che consente di classificarli in due grandi categorie.

Un algoritmo di firma può, infatti, essere o **con recupero**, o **con appendice**.

Negli **algoritmi con recupero** il documento **m** deve avere un valore numerico inferiore a quello del modulo impiegato nei calcoli; se non è così, il documento deve essere preventivamente suddiviso in blocchi.

Il firmatario trasforma dapprima ogni blocco con una **funzione di ridondanza R** (la scelta di **R** è critica dal punto di vista della sicurezza) e poi lo sottopone all'algoritmo di firma. Il verificatore controlla se il risultato appartiene al codominio di **R**: in caso affermativo, ottiene **m** tramite $\mathbf{R^{-1}}$.



Nel caso di messaggi "lunghi" è molto più efficiente l'impiego dei **meccanismi con appendice**. Questo metodo è anche il più usato, consentendo di autenticare messaggi in chiaro.

Il firmatario sottopone il messaggio **m** ad una funzione hash sicura, firma l'impronta con la sua chiave privata ed affianca a questa etichetta il messaggio in chiaro ed il certificato della sua chiave pubblica.

Dall'altra parte del canale il verificatore calcola l'impronta del messaggio ricevuto ed impiega l'algoritmo di verifica (la chiave è quella pubblica del firmatario, che ha estratto dal certificato) per confrontare le due impronte.

Si noti che tutti gli algoritmi con recupero possono essere impiegati anche come algoritmi con appendice, avendo l'hash una dimensione sicuramente inferiore a quella del modulo. Il caso tipico è quello dell'algoritmo **RSA**, che è anche il più usato nelle applicazioni: lo studieremo nel prossimo paragrafo.

Largamente impiegato è anche l'algoritmo con appendice **DSS**: lo studieremo nel paragrafo 5.5.2 unitamente all'algoritmo di **ElGamal** da cui discende.

5.5.1 Firma con RSA

L'algoritmo RSA è **reversibile**, cioè funziona anche scambiando la posizione delle chiavi.

Vediamo cosa succede se si opera in questo modo. Sia **m < n** un testo in chiaro dotato di un certo "formato", stabilito da un certo standard.

L'utente **U** impiega la sua chiave privata **SU = {dU, nU}** per calcolare

$$\mathbf{c = E_{SU}(m) = m^{dU} \bmod nU}$$

L'utente **X** che ha ricevuto **c** impiega la chiave pubblica **PU** di **U** ed ottiene

$$\mathbf{D_{PU}(c) = c^{eU} \bmod n = m^{dU\,eU} \bmod nU = m.}$$

La prima conclusione è che non deve essere in gioco la riservatezza di **m**, dato che **PU** è di dominio pubblico. La seconda conclusione è che così operando si ottiene l'autenticazione dell'origine di **m**: dal calcolo di $\mathbf{D_{PU}(c)}$, **X** può ottenere **m**, verificarne il "formato" e dedurre da questo se **c** proviene o no da **U** (a patto naturalmente che **SU** sia stata tenuta segreta e non sia stata ripudiata).

RSA a chiavi invertite è dunque un algoritmo di firma con recupero.

Il metodo è **sicuro** (chi vuole falsificare la firma di **U** deve saper fattorizzare **n**), ma **inefficiente** (si ricordi che un messaggio lungo deve essere suddiviso in blocchi di lunghezza inferiore a quella di **n**, ciascuno dei quali richiede poi calcoli di complessità non trascurabile).

Per ottenere efficienza è però sufficiente firmare solo l'impronta di **m** con una funzione hash sicura.

1. **U:** calcola $\mathbf{c = (H(m))^{dU} \bmod nU}$ e trasmette **m||c**;
2. **X:** estrae **m'** dal messaggio ricevuto, calcola **H(m')** ed accetta **m'** come un messaggio integro e generato da **U** se e solo se

$$\mathbf{H(m') = c^{eU} \bmod nU = H(m)}.$$

Con RSA, la dimensione in bit dell'appendice è dunque uguale a quella di **nU**.

1 - Padding

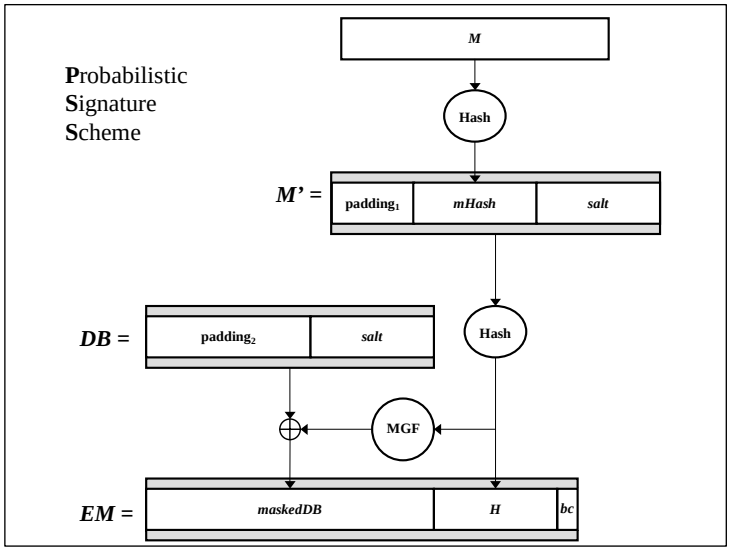
L'impronta ha però una lunghezza molto inferiore a quella del modulo ed è ritenuto insicuro trasformarla direttamente. Esistono diverse proposte di **padding**, atte ad attribuirgli la lunghezza desiderata: $\lfloor \mathbf{log_2\ n} \rfloor$, cioè un bit in meno di **n** e quindi un dato sicuramente minore di **n**.

Una soluzione, molto semplice e molto usata, è quella di impiegare un padding deterministico:

0x**00** 0x**01** 0x**FF**..... 0x**FF** 0x**00** || **H(m)**.

Una seconda soluzione, detta FDH (*Full Domain Hashing*), prevede l'impiego di una funzione hash ideale che genera impronte di lunghezza $\lfloor \mathbf{log_2\ n} \rfloor$. Tale funzione può in pratica essere realizzata concatenando un adeguato numero di impronte di **m** ottenute con un algoritmo di compressione crittografico:

H(c0.m)||H(c1.m)||H(c2.m)|| ...



Oggi però molti crittografi ritengono che sia più sicuro impiegare un padding contenente anche un numero a caso.

In PKCS#1v2, da cui è stata estratta la figura a lato ed a cui si rimanda per ogni dettaglio, è raccomandato l'uso dello standard PPS (*Probabilistic Signature Scheme*)

In [4] è caldeggiata una soluzione ancora diversa:
"si impiega **H(m)** come **seed** di un **PRNG** che fornisce numeri di lunghezza $\lfloor \mathbf{log_2\ n} \rfloor$ e si genera l'appendice di **m** firmando il primo valore così prodotto; il verificatore calcola l'impronta del messaggio ricevuto, avvia con questo seed il suo generatore e ne confronta l'uscita con il risultato dell'algoritmo di verifica".

2 – Firma cieca

La proprietà moltiplicativa di RSA (v. pag. 105) consente di eseguire la cosiddetta firma "**a occhi chiusi**", o "**con messaggio oscurato**"[59].

Consideriamo un messaggio $\mathbf{m = m_1 \times m_2}$. La sua firma da parte di **U** è:

$$\mathbf{c = m^{dU} \bmod nU = ((m_1^{dU} \bmod nU) \times (m_2^{dU} \bmod nU)) \bmod nU}$$

La firma del prodotto di due messaggi è dunque uguale al prodotto delle due firme.

Tale tecnica risulta particolarmente utile nella situazione in cui un utente **X** può richiedere ad un'Autorità **T** di autenticargli un messaggio **m**, di cui, però, non vuole rivelare il contenuto. Casi di questo tipo sono tutt'altro che infrequenti.

ESEMPI – Nei sistemi di **voto elettronico**, l'elettore si fa autenticare da un'Autorità il suo voto (in forma non leggibile per difenderne la segretezza) e lo deposita poi nell'"urna" in forma leggibile e verificabile. La firma dell'Autorità garantisce l'anonimità del votante, il suo diritto ad esprimere un voto ed il fatto che voterà una sola volta.

Nei sistemi di **commercio elettronico**, l'acquirente si fa autenticare da una Autorità il suo denaro elettronico (per un ammontare non leggibile) prima di consegnarlo al venditore in forma leggibile e verificabile. La firma dell'Autorità garantisce al venditore la copertura finanziaria ed al cliente l'anonimato.

[59] D. Chaum "Blind Signatures for Untraceable Payments" *Advances in Cryptology: Proceedings of Crypto82*, Plenum Press.
Dal sito del corso è possibile scaricare RSABBS.zip, contenente una demo sviluppata da Francesca Merighi

Per oscurare il suo messaggio **m**, **X** sceglie a caso un numero **r coprimo con nT** ed invia a **T**:

$$\mathbf{c1 = m \times r^{eT} \bmod nT}.$$

T firma **c1** e restituisce a **X** il testo cifrato

$$\mathbf{c2 = m^{dT} \times r^{eT\,dT} \bmod nT = m^{dT} \times r \bmod nT}$$

X moltiplica **c2** per l'inverso moltiplicativo di **r** (mod nT) ed ottiene **m** autenticato da **T**:

$$\mathbf{c3 = (m^{dT} \times r \times r^{-1}) \bmod nT = m^{dT} \bmod nT}$$

3 – Attacchi "matematici" alla firma RSA

3.1 - Se si impiega uno schema di firma digitale con recupero, chiunque può scegliere un numero a caso **r**, prendere la chiave pubblica RSA di chiunque altro, calcolare $\mathbf{y = r^e \bmod n}$ e produrre **r** come "firma" inconfutabile del "messaggio" **y**.

Questo attacco non preoccupa però più di tanto, dato che il messaggio **y** è un numero a caso ed è veramente difficile che abbia un "significato" impegnativo per il proprietario di **d**.

3.2 - Se si impiega lo schema con etichetta, ed è questo il caso normalmente usato, il precedente attacco diventa praticamente impossibile: dopo aver scelto **r** e calcolato **y**, l'attaccante deve produrre il "messaggio" $z=\mathbf{H^{-1}(y)}$, ma ciò è computazionalmente impossibile se si usa una funzione hash sicura.

E se per un certo **y** l'attaccante ci riuscisse? E se inoltre fosse così fortunato da forgiare un dato z che gli preme attribuire al legittimo proprietario delle chiavi?

La probabilità che ciò si verifichi è bassissima, ma per fronteggiare attacchi di questo tipo è veramente opportuno rispettare la seguente regola:

☐ R30: "*il messaggio da firmare, qualsiasi sia il suo contenuto (il caso limite è un numero casuale), deve essere preventivamente dotato di una struttura ben precisa, quale, ad esempio, può essere conseguita inserendo in testa una data ed in coda cognome-nome del firmatario".*

3.3 - Il normale uso di RSA (come Cifrario e come Schema di firma) prevede di esponenziare **qualsiasi numero minore di n**. Dal punto di vista della sicurezza questo è un errore, per fortuna piccolo: la unidirezionalità della funzione $y = x^d \bmod n$ è, infatti, garantita **solo se x è coprimo con n**.

I numeri minori di **n** e multipli di **p** o di **q** sono $\mathbf{n- \Phi(n)}$. Con i moduli attualmente impiegati, il rapporto $\mathbf{(n- \Phi(n))/n}$ ha un valore così piccolo, da rendere altamente improbabile generare a caso un numero non coprimo con n ed impiegarlo poi per condurre un attacco alla fattorizzazione: nessuno quindi si preoccupa del fatto che ogni tanto capita di trasformare un numero non appartenente al gruppo moltiplicativo Z^*_n.

3.4 - Una più pericolosa vulnerabilità discende dalla proprietà moltiplicativa. Supponiamo che l'intruso possa richiedere al firmatario X di firmare qualsiasi messaggio di sua scelta, ad esclusione del messaggio **m** di suo interesse (questo scenario è detto ***chosen message attack***). Per ottenere la firma di **m** è sufficiente che l'intruso generi un numero a caso **r**, costruisca $\mathbf{m1 = m \times r}$, calcoli $\mathbf{m2 = r^{-1}}$, chieda ad X di firmare entrambi i messaggi e moltiplichi infine le firme di **m1** e di **m2** che gli vengono restituite.

☐ R31: "*la chiave usata per apporre firme ad occhi chiusi non deve essere impiegata in alcun altro contesto".*

Il precedente attacco può essere utilizzata anche da un intruso che ha intercettato un testo cifrato con RSA e destinato ad un utente X. L'intruso oscura c, se lo fa firmare da X (il protocollo a sfida/risposta costituisce un'ottima occasione, come vedremo tra poco), elimina il numero a caso ed ottiene il testo in chiaro che non doveva poter conoscere. Vale dunque la seguente regola.

❑ R32: "*chi impiega RSA per decifrare e per firmare deve utilizzare due differenti coppie di chiavi*".

5.5.2 Firma con ElGamal e con DSS

Il primo algoritmo probabilistico per la firma digitale è stato individuato da ElGamal, unitamente al Cifrario che abbiamo studiato a pag. 106.

Successivamente sono state messe a punto diverse varianti (Schnorr, Nyberg-Rueppel, ecc.) miranti a conseguire maggiore efficienza nei calcoli.

La variante oggi più usata è l'algoritmo DSA (*Digital Signature Algorithm*), individuato da D.W. Kravitz nel 1991 e poi ridenominato DSS (*Digital Segnature Standard*) nel 1994 quando, dopo alcune revisioni che ne hanno aumentato la robustezza, è stata accettata la proposta del NIST di adottarlo come

standard negli USA. L'uso obbligatorio di SHA-1 (si è già detto che sono state trovate collisioni) e di numeri primi di 1024 bit (troppo poco a fronte dell'inarrestabile evoluzione della tecnologia e degli algoritmi di rottura) fanno prevedere una prossima revisione di questo standard.

1 – L'algoritmo di firma di ElGamal

L'algoritmo è **probabilistico**, **con etichetta** ed impiega come modulo un **grande numero primo p** (oggi come minimo di 1000 bit). L'input è l'impronta **h=H(m)** del documento che deve essere autenticato (il valore numerico di h è dunque sicuramente minore di p).

L'output è formato da due etichette **R**, **S** di $\lceil \mathbf{log_2\ p} \rceil$ bit ciascuna.

La sicurezza si basa sulla presunta intrattabilità del problema del logaritmo discreto.

Algoritmo *G* - Ogni utente **U**, dopo aver scelto un primo **p** ed una sua radice primitiva **g**, genera a caso un numero a caso **a** (con $1 \leq a \leq p-2$) ed assume

- {**a**} come sua chiave privata
- {**p, g, y = g^a mod p**} come sua chiave pubblica.

L'impossibilità computazionale di risolvere il problema difficile del logaritmo discreto garantisce a U che il suo dato segreto **a** non può essere dedotto dal dato pubblico **y**.

Algoritmo *S* – Come si è detto, l'algoritmo è probabilistico. **Per ogni messaggio** il firmatario deve infatti scegliere a caso un **k** nell'intervallo 2 ÷ p-2 tale che **MCD(k, p-1)=1** e generare l'etichetta

$$\mathbf{R = g^k \bmod p.}$$

Anche in questo caso è in gioco il problema del logaritmo discreto: chi intercetta **R** non può individuare il valore di **k**.

Vediamo ora come si calcola l'etichetta **S**, cosa che ci consentirà anche di giustificare il precedente vincolo su **k**. Una firma di ElGamal è autentica se e solo se vale l'uguaglianza

$$\boxed{\mathbf{g^h \bmod p = y^R R^S \bmod p}}$$

e quindi $\mathbf{g^h \bmod p = g^{aR}\, g^{kS} \bmod p}$

o anche $\mathbf{g^{h-aR-kS} \equiv 1 \pmod p}$

Per T4 si ha: $\mathbf{h - aR - kS \equiv p-1 \pmod p}$

e quindi $\mathbf{h - aR - kS \equiv 0 \pmod{p-1}}$

da cui si ottiene $\mathbf{S = k^{-1}\,(h - aR) \bmod (p-1)}$

L'espressione di S è detta **equazione congruenziale di firma** e, per un noto teorema della Teoria dei numeri, ha un'unica soluzione solo se **k** è coprimo con **p-1**, cioè la condizione che abbiamo posto fin dall'inizio e che garantisce anche l'esistenza e l'unicità di $\mathbf{k}^{-1}$ (mod p-1).

Si noti che per il calcolo di **S** è richiesta la conoscenza di **a**: solo il legittimo proprietario della coppia di chiavi può dunque generare firme autentiche.

Algoritmo *V* - Chi vuole verificare l'autenticità del messaggio **m** affiancato dalla coppia **R**, **S** deve disporre di una copia autentica della chiave pubblica del firmatario.

Come prima cosa occorre verificare che sia $1 \leq R \leq p-1$ e $S \neq p-1$: esiste, infatti, un attacco particolare che può avere successo se non si opera questo controllo.

Una volta calcolata l'impronta **h=H(m)** del messaggio ricevuto, occorre fare i seguenti calcoli:

$$\mathbf{v1 = y^R R^S \bmod p}$$
$$\mathbf{v2 = g^h \bmod p}$$

Se si ottiene **v1 = v2** il messaggio **m** è integro ed è stato generato proprio da **U**.

2 – L'algoritmo di firma DSS

DSS, anch'esso **probabilistico** e **con appendice**, discende dall'algoritmo di firma di ElGamal e basa quindi la sua sicurezza sul problema difficile del logaritmo discreto.

Le differenze sono tre.
1. La dimensione di **p** è fissata dallo standard: $2^{512} < \mathbf{p} < 2^{1024}$.
2. I calcoli di **R** e di **S** sono fatti **mod q**, ove **q** è il più grande fattore primo di **p-1** tale che $2^{159} < \mathbf{q} < 2^{160}$: le due etichette sono dunque formate ciascuna da 160 bit.
3. L'impronta del messaggio **H(m)** deve essere di 160 bit e deve essere calcolata con **SHA-1**.

Per ottenere minore onerosità del calcolo dell'esponenziazione, l'algoritmo opera su un **sottogruppo moltiplicativo** di $\mathbf{Z_p}$ di ordine **q** (v. pag. 85).

Nel caso del DSS il sottogruppo ciclico di ordine **q** è unico.

Algoritmo *G* - Ogni utente **U** deve
1. scegliere a caso il suo primo **q**,
2. individuare un primo **p** tale che **q** sia un divisore di **p-1** (esiste in proposito un algoritmo suggerito dal NIST: v. [2], pag. 452),
3. individuare un generatore **g** del sottogruppo di ordine **q**,
4. scegliere a caso una chiave privata **a**, con $\mathbf{1 \le a \le q-1}$,
5. calcolare $\mathbf{y = g^a \bmod p}$ e rendere di dominio pubblico la chiave pubblica {**p**, **q**, **g , y**}.

Algoritmo *S* - Per ogni messaggio **m** da firmare (la lunghezza è arbitraria, dato che è autenticata la sola impronta), **U** deve scegliere un numero a caso **k**, con $\mathbf{1 \le k \le q-1}$ e calcolare due etichette di 160 bit ciascuna:

$$\mathbf{R = (g^k \bmod p) \bmod q}$$
$$\mathbf{S = [k^{-1} \times (H(m) + a \times R)] \bmod q}$$

Una firma autentica **m** deve dunque soddisfare l'equazione congruenziale

$$\mathbf{H(m) \equiv - a \times R + k \times S \ (mod\ q)}$$

Algoritmo *V* - Per verificare una firma di **U** occorre conoscere in modo sicuro **PU** (tipicamente estraendola da un certificato) e considerare la terna <**m**, **S**, **R**>.

La prima operazione da fare è controllare che entrambe le etichette ricevute siano maggiori di **0** e minori di **q**: in caso contrario la firma è sicuramente falsa.

Si devono poi fare i seguenti calcoli:

$$\mathbf{w = S^{-1} \bmod q}$$

$$\mathbf{u1 = [H(m) \times w] \bmod q}$$

$$\mathbf{u2 = (R \times w) \bmod q}$$

$$\mathbf{v = [(g^{u1} \times y^{u2}) \bmod p] \bmod q}.$$

Se si ottiene **v = R,** allora si è certi che **U** è il firmatario di **m**.

Giustificazione[60] – Supponiamo che le due etichette siano autentiche.

Dalla precedente equazione di firma si ha:

$$\mathbf{H(m) \equiv - a \times R + k \times S \ (mod\ q)}.$$

Dopo aver moltiplicato i due membri per **w** si ottiene:

$$\mathbf{w \times H(m) + w \times a \times R \equiv k \ (mod\ q)} \text{ e quindi}$$

$$\mathbf{u1 + a \times u2 \equiv k \ (mod\ q)}.$$

Impiegando le due parti come esponenti di **g** si ha:

$$\mathbf{g^{(u1 + a\ u2) \bmod q} \equiv g^{k \bmod q} \ (mod\ p)} \text{ e quindi}$$

$$\mathbf{(g^{u1} \times y^{u2} \bmod p) \bmod q \equiv (g^k \bmod p) \bmod q}, \text{ cioè}$$

$$\mathbf{v = R}.$$

[60] Utili verifiche possono essere fatte anche con DSS.nb, disponibile nel sito del corso

5.6 Protocolli d'identificazione

5.6.1 Protocollo challenge/response

Il protocollo a sfida/risposta può essere realizzato anche con un algoritmo di firma (v. pag. 40); la sfida è un messaggio "corto" e vanno quindi bene sia gli algoritmi con recupero, che quelli con etichetta.

L'identificazione di **A** da parte di **B** richiede tre passi.

1. **A** $\leftarrow$ **B: RB** (nonce)
2. **A** $\rightarrow$ **B: c = S_{SA}(RB)**
3. **B:** identifica **A** se **V_{PA}(c)** fornisce **RB** e la risposta **SI**.

Diversi accorgimenti consentono di rendere il protocollo più robusto.

- **A**, prima di firmare la sfida, concatena a **RB** un numero **RA** di sua scelta e lo trasmette anche in chiaro per consentire a **B** di verificare la risposta:

  A $\rightarrow$ B: RA||S_{SA}(RA||RB)

ESEMPIO- Chi impiega una stessa chiave privata RSA per decifrare e per firmare può così difendersi da un B disonesto che vuole farsi decifrare dei testi intercettati.
La legge italiana impone l'uso di chiavi distinte: una per la riservatezza, una per l'autenticazione ed una per l'identificazione.

- **A** allega alla risposta un certificato della sua chiave pubblica:

  A $\rightarrow$ B: Cert(PA,T)||RA|| S_{SA}(RA||RB)

ESEMPIO – L'invio del certificato impedisce l'attacco dell'uomo in mezzo, aiuta a scegliere l'algoritmo di verifica e consente di controllare l'intervallo di validità di PA ed il suo stato di non ripudio (per quest'ultimo aspetto occorre però interrogare la CRL dell'Autorità T che ha emesso il certificato).

- **A** firma il dato di sfida concatenato con una marca temporale **τA** e con l'identificativo di **B**:

  A $\rightarrow$ B: cert(PA, T)||τA|| S_{SA}(τA||RB||B)

ESEMPIO – L'uso del *time stamp* e del nome dello sfidante impedisce ad un intruso di tenere contemporaneamente aperti un protocollo con cui risponde a B ed uno con cui sfida A (il trucco del "gran maestro").

Il protocollo a sfida/risposta consente, come sappiamo, l'identificazione mutua. Occorrono tre passi, solitamente svolti al di sotto del livello applicazione:

1. **A $\leftarrow$ B: RB**
2. **A $\rightarrow$ B: cert(PA,T) || τA || B || S_{SA}(τA || RB || B)**
3. **B $\rightarrow$ A: cert(PB, T') || τB || S_{SB}(τB || τA || A)**

Unitamente, o subito dopo, a questo scambio di certificati, i due corrispondenti in generale procedono ad uno scambio DH non anonimo per arrivare a condividere una chiave segreta con cui difendere la riservatezza delle loro successive comunicazioni.

In molte applicazioni *client/server*, per maggiore sicurezza, all'utente *client* sono poi richieste ulteriori prove di conoscenza e di possesso.

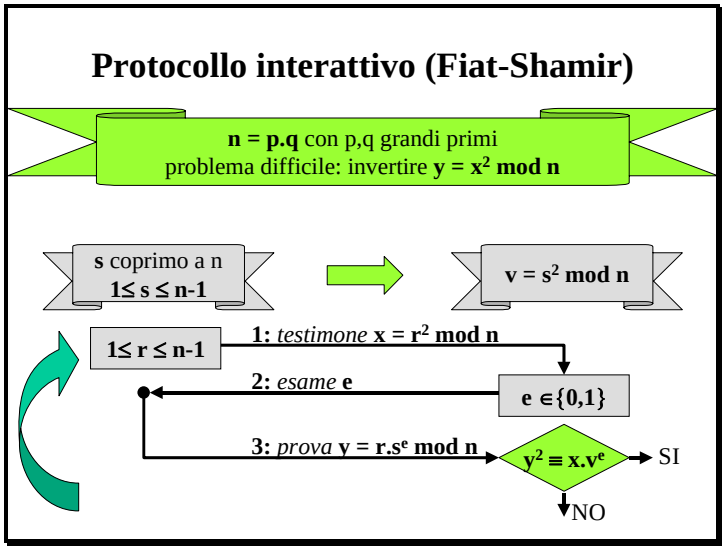
Il protocollo challenge/response ha comunque un difetto non eliminabile: il dato segreto che identifica un'entità deve partecipare alla formazione della risposta e non può quindi mai essere perfettamente protetto.

5.6.2 Zero-knowledge

I protocolli d'identificazione "a conoscenza zero" impiegano un **segreto diverso in ogni sessione**: è dunque perfettamente inutile cercare di individuarlo.

Le implicazioni del principio della *zero knowledge* sono molte, complesse e non saranno approfondite in questa sede; per fissare almeno le idee, si pensi ad uno che conosce un modo efficiente per risolvere un certo problema difficile e che lo dimostra ad un altro facendogli vedere di essere in grado di risolvere un diverso ed isomorfo problema difficile.

Istruttivo è comunque dare una prima occhiata al **protocollo interattivo**[61] che per primo ha applicato la conoscenza zero all'identificazione. Il problema difficile è quello della radice quadrata di un numero, modulo il prodotto di due fattori primi grandi (**P5**).



L'identificando **A** sceglie come chiave privata un numero **s** minore di **n = p × q** e coprimo con **n**; la chiave pubblica di **A**, consegnata da un arbitro al verificatore **B**, è

$$v = s^2 \bmod n$$

Passo 1: A sceglie a caso un numero **r** ed invia a **B** il numero $x = r^2 \bmod n$.

Passo 2: B sceglie a caso un bit **e**, lo invia ad **A** e gli chiede di rispondere con

- **r**, se **e = 0**,
- $r \times s \bmod n$, se **e = 1**.

Passo 3: l'identificando invia come risposta

$$y = r \times s^e \bmod n$$

A questo punto il verificatore calcola $y^2 \bmod n$, $x \times v^e \bmod n$ e controlla se i due risultati sono congruenti modulo **n** (si noti che $x \times v^e \bmod n = r^2 \times (s^2)^e \bmod n$).

Il legittimo proprietario del segreto non ha problemi a superare la verifica a fronte di entrambi i tipi d'esame.

L'impostore che ha scelto un qualsiasi **r** supera agevolmente l'esame di **e = 0**; non supera però quello di **e = 1**: non conosce **s** e non è quindi in grado di dedurlo da **v**. Per superare la prova **e = 1**, l'impostore potrebbe allora scegliere $x = r^2/v$, ma non sarebbe poi in grado di inviare una corretta risposta alla prova **e = 0**.

Per sicurezza i tre passi devono essere **iterati un certo numero di volte**, sempre con scelte casuali di **r, e**. Versioni successive[62] hanno ulteriormente migliorato l'efficienza del protocollo.

Da tale protocollo di identificazione, Feige, Fiat e Shamir hanno dedotto un algoritmo probabilistico di firma con appendice (v. [2], pag. 447), che basa la sua sicurezza sulla intrattabilità del calcolo di radici quadrate modulo un **n** prodotto di due grandi primi **p**,**q** scelti a caso.

A differenza di RSA, in questo algoritmo è richiesta la presenza di un Ente fidato, che deve non solo scegliere e tenere segreti i fattori del modulo n, ma anche calcolare la chiave pubblica e la chiave privata di ogni utente.

[61] A.Fiat, A.Shamir "How to prove yourself: practical solutions to identification and signature problems" Advances in Cryptology- CRYPTO 86 Proceedings, Springer-Verlag 1987, pp.186-194.

[62] Nel sito del corso è disponibile FiatFeigeShamir.nb per documentarsi e fare delle prove