

Crittografia e sicurezza delle reti

Crittografia a chiave pubblica

1. One way Trapdoor Functions
2. RSA Public Key CryptoSystem
3. Diffie-Hellman
4. El Gamal

RSA: Frammenti dall'articolo (1978)

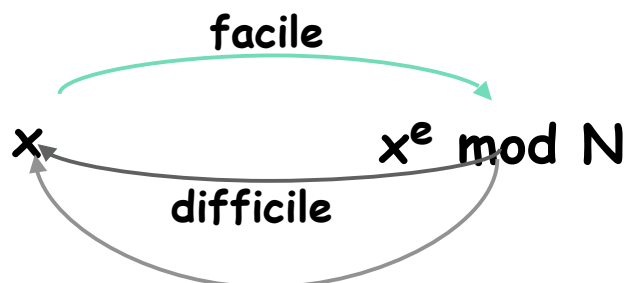
The era of "electronic mail" may soon be upon us; we must ensure that two important properties of the current "paper mail" system are preserved: (a) messages are *private*, and (b) messages can be *signed*. We demonstrate in this paper how to build these capabilities into an electronic mail system.

At the heart of our proposal is a new encryption method. This method provides an implementation of a "public-key cryptosystem," an elegant concept invented by Diffie and Hellman. Their article motivated our research, since they presented the concept but not any practical implementation of such system.

Crittografia a chiave pubblica: Schema di base

- Uno schema crittografico a chiave pubblica include:
 - Una chiave privata k
 - Una chiave pubblica k'
 - Un metodo di codifica che sia una trap door OWF. La chiave privata rappresenta l'informazione trap-door
- La chiave pubblica è distribuita (ognuno può codificare)
- La decodifica richiede la chiave privata

One Way Function (OWF)



facile: esiste algoritmo Probabilistico con Tempo Polinomiale (PPT) A t.c. $A(x) = f(x)$ per ogni x

difficile: per ogni alg. PPT B per ogni k suff. grande

$\text{Prob}(B(f(x)) = x' \text{ t.c. } f(x) = f(x'))$ con $|x| = k$ è pari a quella di gettare monete

OWF: definizione

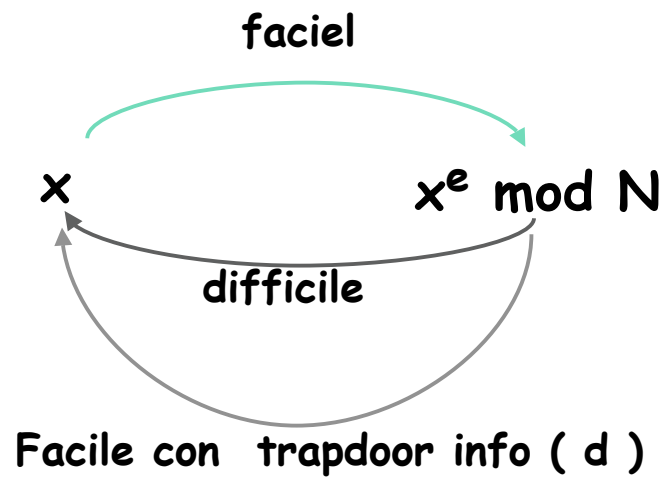
Definizione: $f: D \rightarrow R$ è *one way function* se è

- facile da calcolare
- difficile da invertire

Cosa vuol dire difficile:

- con tempo polinomiale
- inversione possibile raramente anche in modo probabilistico

One Way Trapdoor Function (OWF)



Trap-door OWF

Definizione: $f: D \rightarrow R$ è una *trap-door one way function* se esiste trap-door s tale che

- Senza conoscenza di s , la funzione è one way
- Dato s , invertire f è facile

Esempio: $f_{g,p}(x) = g^x \bmod p$ non è trap-door one way function (non esiste trap door).

RSA è considerata trap-door OWF

- Trovare d , dati p e q è facile
- Trovare d dati solo N e e è considerato *difficile* (*assunzione di RSA come OWF*)

Collezione di Trap-door OWF

Definizione: Sia I un insieme di indici e D_i un insieme finito. Una collezione di funz. trapdoor è una collezione di

$f_i: D_i \rightarrow R_i$ è una *trap-door one way function*

Generazione: Esiste un algoritmo PPT G che con input parametro di sicurezza k seleziona una funzione casuale f_i in F insieme ad una informazione trapdoor t_i (con $|i| = k$)

Proprietà trapdoor: Esiste algor. PPT I tale che

$I(f_i(x), t_i) = x'$ tale che $f(x) = f(x')$ (ad es. $x=x'$)

Sicurezza

- idea bellissima (Diffie Hellman)
- chiaramente se F è una collezione di funzioni trapdoor per la proprietà di di essere difficili da invertire (senza conoscenza trapdoor) è computazionalmente difficile invertire il codice crittografico
- E' SUFFICIENTE?

Problemi di sicurezza

- Insieme di messaggi speciali: il fatto che f sia una funz. trapdoor non implica che sia difficile da invertire quando il mess. appartiene ad un insieme limitato
- Informazione parziali: il fatto che f sia trapdoor non implica che sia possibile avere informazioni parziali su x
- Relazioni fra codifiche: si possono ottenere relazioni fra messaggi correlati (es. lo stesso mess. è inviato due volte)

Possibili sorgenti di problemi difficili

- Teoria dei numeri: fattorizzazione di interi
- Teoria dei codici: Decodifica di codici lineari
- Geometria algebrica: Problemi computazionali su curve ellittiche
- Geometria dei numeri: problemi computazionali su reticoli di interi

Come mostrare una funzione è Trapdoor One Way

Usare la definizione

- mostrare algoritmo per generare le chiavi
- mostrare algoritmo veloce per calcolo
- **dimostrare** che non esiste algoritmo veloce di inversione

in molti casi possiamo solo **congetturare** che non esiste inversione veloce (cioè che l'inversione è computazionalmente difficile). Non abbiamo limitazioni inferiori

Come credere che un problema sia difficile?

Caso peggiore

- esperienza
- riduzioni: se trovo un algoritmo veloce allora $P=NP$ o $NP= BPP$

Caso medio

- provare che se A è difficile nel caso peggiore allora B è difficile nel caso medio (utilizzo di riduzioni probabilistiche)

RSA Public Key Cryptosystem

Definizione Sia $\phi(N)$ il numero di interi primi con N (se N è primo allora $\phi(N) = N-1$)

RSA

- Sia $N=pq$ il prodotto di due primi
- Sia e tale che $\gcd(e, \phi(N))=1$
- Sia d tale che $de \equiv 1 \pmod{\phi(N)}$

La chiave pubblica è (N, e) , la chiave segreta d

- Codifica di $M \in \mathbb{Z}_N^*$: $C = E(M) = M^e \pmod{N}$
- Decodifica di $C \in \mathbb{Z}_N^*$: $M = D(C) = C^d \pmod{N}$

RSA: esempio

Sia $p=47$, $q=59$, $N=pq=2773$, $\phi(N) = 46*58=2668$.

Scegli $e=157$, allora $157*17 - 2668 = 1$, quindi $d=17$ è l'inverso di 157 mod 2668.

Per $N = 2773$ codifichiamo due lettere per blocco usando due cifre per lettera:

blank=00, A=01, B=02, ..., Z=26.

Messaggio: ITS ALL GREEK TO ME si codifica

0920 1900 0112 1200 0718 0505 1100 2015 0013 0500

RSA: esempio

$N=2773$, $e=17$ (10001 in binario).

ITS ALL GREEK TO ME si codifica

0920 1900 0112 1200 0718 0505 1100 2015 0013 0500

Primo blocco $M=0920$ viene crittato

$$M^e = M^{17} = (((M^2)^2)^2)^2 * M = 948 \pmod{2773}$$

L'intero messaggio (10 blocchi) è crittato

0948 2342 1084 1444 2663 2390 0778 0774 0219 1655

Infatti $0948^d = 0948^{157} = 920 \pmod{2773}$, etc.

Teorema di Fermat

Se p è primo e a è intero non divisibile per p allora

$$a^{(p-1)} = 1 \pmod{p}$$

Prova (si assume, senza perdita di generalità, che $a < p$)

- per proprietà di mod abbiamo che

$$[a \cdot (2a \pmod{p}) \cdot (3a \pmod{p}) \dots \cdot ((p-1)a \pmod{p})] =$$

$$(a \cdot 2a \cdot 3a \dots \cdot (p-1)a) \pmod{p}$$

- poichè p è primo allora sequenza $a, 2a \pmod{p}, 3a \pmod{p}, \dots, (p-1)a \pmod{p}$ sono i numeri $1, 2, \dots, p-1$ in qualche ordine quindi $[a \cdot (2a \pmod{p}) \cdot (3a \pmod{p}) \dots \cdot ((p-1)a \pmod{p})] = [(p-1)!]$
- $(a \cdot 2a \cdot 3a \dots \cdot (p-1)a) \pmod{p} = ((p-1)! \cdot a^{(p-1)}) \pmod{p}$
- p primo implica che $(p-1)!$ non è multiplo di p , quindi si può cancellare fattore comune $(p-1)!$ ottenendo la tesi

Teorema di Eulero

Se a e n sono primi fra loro allora $a^{\phi(n)} = 1 \pmod{n}$

Prova Se n primo allora è vero da $[\phi(n) = n-1$ e vale Fermat]

n non primo: per def. a appart. al gruppo $Z_n^* = \{x_1, x_2, \dots, x_{\phi(n)}\}$

(n) ; a primo con n implica che i numeri $a \cdot x_1 \pmod{n}, a \cdot x_2 \pmod{n}, \dots, a \cdot x_{\phi(n)} \pmod{n}$ sono primi con n e sono una permutazione di $x_1, x_2, \dots, x_{\phi(n)}$ (se $a \cdot x_i \pmod{n} = a \cdot x_j \pmod{n}$ allora $x_i = x_j$)

- quindi $(a \cdot x_1 \pmod{n}) \cdot (a \cdot x_2 \pmod{n}) \dots (a \cdot x_{\phi(n)} \pmod{n}) = x_1 \cdot x_2 \dots \cdot x_{\phi(n)}$ quindi
- $(a \cdot x_1) \cdot (a \cdot x_2) \dots (a \cdot x_{\phi(n)}) = (x_1 \cdot x_2 \dots \cdot x_{\phi(n)}) \pmod{n}$ quindi
- $a^{\phi(n)} (x_1 \cdot x_2 \dots \cdot x_{\phi(n)}) = (x_1 \cdot x_2 \dots \cdot x_{\phi(n)}) \pmod{n}$ quindi
- $a^{\phi(n)} = 1 \pmod{n}$

Il gruppo moltiplicativo Z_N^*

Dato N il gruppo moltiplicativo Z_N^* contiene tutti gli interi nell'intervallo $[1, pq-1]$ primi con N

Teorema Z_N^* è un gruppo

Prova: chiusura: se a e b sono primi con N allora

$$au + vn = 1 \quad \text{inoltre} \quad bw + zn = 1$$

moltiplicando le due equazioni insieme otteniamo

$$(au + vn)(bw + zn) = uw(ab) + (auz + vbw + vzn) = 1$$

quindi (ab) è primo con n e la tesi segue.

Il gruppo moltiplicativo Z_{pq}^*

Siano p e q due primi e sia $N = p \cdot q$ il loro prodotto.

Il gruppo moltiplicativo $Z_N^* = Z_{pq}^*$ contiene tutti gli interi nell'intervallo $[1, pq-1]$ primi sia rispetto a p che a q - che sono $\phi(pq) = (p-1)(q-1) = N - (p+q) + 1$

Teorema Per ogni $a \in Z_{pq}^*$, $a^{(p-1)(q-1)} = 1 \pmod N$

Prova: sia k ordine di a ; le potenze di a generano un

sottogruppo con k elementi; quindi k divide

$(p-1)(q-1)$ e esiste c t.c. $ck = (p-1)(q-1)$; per def. di k si ha

$a^k = 1 \pmod N$; quindi la tesi

Equivalentemente per ogni a $a^{\phi(n)+1} = a \pmod N$

Il gruppo moltiplicativo Z_{pq}^*

II Prova con aritmetica mod. di $a^{\phi(n)+1} = a \pmod N$

- se a primo con pq ($\gcd(a,pq) = 1$) vale Teor. Eulero
- se $\gcd(a,pq) \neq 1$ poichè p e q sono primi segue
(a multiplo di p) OR (a multiplo di q)
- $a < pq$ implica che a non è multiplo sia di p che di q
- supponi a multiplo di p (ma non di q): $a=hp$ e $\gcd(a,q) = 1$
- per Eulero $a^{\phi(q)} = 1 \pmod q$
- inoltre $\phi(pq) = (p-1)(q-1) = \phi(p)\phi(q)$
- quindi $a^{\phi(pq)} = [a^{\phi(q)}]^{\phi(p)} = 1 \pmod q$
- quindi esiste intero k t.c. $a^{\phi(pq)} = 1 + kq$
- moltiplicando ambo i lati per $a=hp$ si ha $a^{\phi(pq)+1} = a + kqhp = a \pmod{pq}$

Perché RSA è una codifica

Teorema Se e , $1 < e < (p-1)(q-1)$ è primo rispetto a

$(p-1)(q-1)$ allora $a \mapsto a^e$ è una funzione 1-1 in Z_{pq}^*

Prova: Poichè $\gcd(e, (p-1)(q-1)) = 1$, e appartiene a Z_{pq}^*

e , quindi, ha inverso moltiplicativo mod $(p-1)(q-1)$.

- Sia esso d , allora $ed = 1 + k(p-1)(q-1)$, k costante

- Sia $y = a^e$, allora $y^d = (a^e)^d = a^{1+k(p-1)(q-1)} = a^{1+k\phi(pq)}$

- $a^{(p-1)(q-1)} = 1 \pmod N$ implica che $a^{k\phi(pq)} = 1 \pmod N$ e la tesi segue

Equivalentemente $y \mapsto y^d$ è l'inverso di $x \mapsto x^e$ e

RSA è uno schema di codifica/decodifica

RSA: implementazione

1. Trovare p e q , numeri primi grandi
 - casuali (non prevedibili da un attaccante e diversi ogni volta che definisco una nuova chiave)
2. Definire e per ottenere una codifica veloce
 - si usa un algoritmo di esponenziazione basato su ripetute operazioni di elevamento al quadrato: la codifica binaria di e ha pochi bit pari a uno in posizioni strategiche (ad esempio $e=3$ oppure $e=2^{16}+1$)
 - la decodifica di solito più lenta (d ha un valore grande)
3. Calcolare d :
 - si utilizza Eulero: $e^{\phi(n)} = 1 \pmod N$ implica $e^{\phi(n)-1} = e^{-1} \pmod N$

RSA: implementazione

1. Trovare numeri primi grandi che siano non prevedibili

Algoritmo:

- scegli a caso un numero dispari grande
- verifica se è primo (prossimamente)

Nota:

- i numeri primi sono frequenti (tra N e $2N$ ci sono $\approx N/\log N$ numeri primi)
- scegliendo a caso mi aspetto di trovare un numero primo ogni $\log N$ tentativi

RSA: implementazione

2. Algoritmo per ottenere una codifica veloce

si usa un algoritmo di esponenziazione basato su ripetute operazioni di elevamento al quadrato e si moltiplicano fra loro quelli necessari per il calcolo del risultato (si usa notazione binaria di e)

- al massimo $O(\log N)$ operazioni
- numero costante di operazioni se la codifica binaria di e ha pochi bit pari a uno in posizioni strategiche

esempio:

- $e = 3$, si calcola $M^2 \bmod N$,
 $M^3 \bmod N = ((M \bmod N) * (M^2 \bmod N)) \bmod N$
- $e = 65537 (2^{16} + 1)$, si calcola $M^2 \bmod N, M^4 \bmod N, M^8 \bmod N, M^{16} \bmod N \dots M^{65536} \bmod N$ e poi $M^{65536} \bmod N$ (una moltiplicazione mod N)

RSA - Attacchi

RSA è in generale difficile da attaccare (richiede fattorizzare)

MA

- alcuni N prodotto di due primi sono 'facili' da fattorizzare (alg. di Pollard)
- alcuni messaggi sono facili da decifrare

RSA e fattorizzazione

- Fatto 1: dato n, e, p e q è facile calcolare d
- Fatto 1: dato $n, e,$
 - calcolare $\phi(n)$ è tanto difficile quanto fattorizzare n
 - calcolare d è tanto difficile quanto fattorizzare n
- Conclusione:
 - se si può fattorizzare n si inverte RSA
 - se si inverte RSA si fattorizza n ? **PROBLEMA APERTO**

Attacchi a RSA

Fattorizzare $N=pq$. Questo è considerato difficile a meno che p, q verificano alcune proprietà "cattive". Per evitare questi primi si consiglia di

- Prendere p, q sufficientemente grandi (>100 cifre decimali ciascuno).
- Verificare che p, q non siano troppo vicini fra loro.
- Verificare che $(p-1)$ e $(q-1)$ abbiano fattori primi grandi (algoritmo di Pollard)

Attacchi a RSA

Fattorizzare $N=pq$: RSA challenges (sfide)

la soc. RSA Security pubblica sfide per la fattorizzazione di numeri grandi:

- RSA 426 bit, 129 digit:
 - pubblicato nel 1977
 - fattorizzato nel 1994 in 8 mesi con 1600 computer in internet (10000 Mips)
- RSA 576 bit, 173 digit: fattorizzato dic. 2003, 10000 \$
- RSA 640 (premio 20K\$), RSA 1024 (100K\$), RSA 2048 (200K\$)

RSA - Attacchi

- Messaggi facili da decifrare se $m=0,1,n-1$ allora $\text{RSA}(m) = m$: **SOLUZ: rari, uso salt**
- Inoltre se m è piccolo e $e=3$ allora può accadere che $m^3 < n$; quindi
$$m^3 \bmod n = m^3$$
attaccante calcola radice cubica e trova m

SOLUZ. aggiungere byte inizio del messaggio per evitare messaggi piccoli

RSA - Attacchi

Piccolo valore di e (ad es. $e=3$)

- Supponi che avversario abbia due codifiche di messaggi simili, ad es.

$$c_1 = m^3 \bmod n \text{ e } c_2 = (m+1)^3 \bmod n$$

- quindi in questo caso

$$m = (c_2 + 2c_1 - 1) / (c_2 - c_1 + 2)$$

- analogo per la coppia m e $(am + b)$

SOLUZ. scegliere e grande

RSA - Attacchi

Assumi $e=3$ e di inviare lo stesso messaggio 3 volte a tre utenti diversi con chiave pubbl.

$$(3, n_1) \text{ } (3, n_2) \text{ e } (3, n_3)$$

ATTACCO: Attaccante conosce chiavi pubbl. e conosce $m^3 \bmod n_1$, $m^3 \bmod n_2$, $m^3 \bmod n_3$

- usando Teorema resti cinese calcola $m^3 \bmod (n_1 * n_2 * n_3)$
- inoltre sa $m < n_1, n_2, n_3$; quindi $m^3 < n_1 * n_2 * n_3$ e $m^3 \bmod n_1 * n_2 * n_3 = m^3$
- attaccante calcola radice cubica e trova m

SOLUZ. aggiungere byte casuali nel messaggio per evitare messaggi uguali

RSA - Attacchi

- Se lo spazio dei possibili messaggi è piccolo si può procedere per enumerazione
es.: attaccante intercetta codifica c di m e sa che m è $m_1=10101010$ o $m_2=01010101$
attaccante codifica m_1 e m_2 con chiave pubblica e verifica
- SOLUZ. aggiungere stringa casuale nel messaggio

RSA - Attacchi

- Se due utenti hanno chiavi con lo stesso n (ma e e d diversi) allora il sistema è debole

SOLUZ. ognuno si sceglie il proprio n (la prob. che scelgano lo stesso n è molto bassa perché ci sono solo molti numeri primi)

RSA - Attacchi

- Proprietà moltiplicativa di RSA:
 - Se $M = M_1 * M_2$ allora $(M_1 * M_2)^e \bmod N = ((M_1^e \bmod N) * (M_2^e \bmod N)) \bmod N$
Quindi attaccante può procedere per messaggi piccoli (attacco ciphertext)
 - Generalizzabile se $M = M_1 * M_2 * \dots * M_k$
 - Soluzione: padding su messaggi brevi

RSA - Attacchi

Proprietà moltiplicativa di RSA

Attacco chosen ciphertext:

- T conosce $c = M^e \bmod n$
- T sceglie a caso X e calcola $c' = c * X^e \bmod n$ e chiede a A di decodificare c'
- A calcola $(c')^d = c^d * (X^e)^d = M * X \bmod n !!$
- Soluzione: imporre struttura sui messaggi da codificare (A rifiuta di codificare se M non verifica quanto richiesto)

RSA - Attacchi sull'implementazione

Attaccare l'implementazione di RSA

- Timing: basato sul tempo richiesto per il calcolo di C^d
- Energia: basato sull'energia richiesta da una smart card per il calcolo di C^d

RSA - Attacchi: conclusione

La versione di RSA presentata nei libri NON è sicura

- non rispetta le definizioni base di sicurezza
- esistono molti attacchi

Usare una versione standard

- preprocessare M per ottenere M' e applicare RSA a M' (ovviamente il significato di M e M' è lo stesso)



Public-Key Crypto. Standard (PKCS)

Problema: inviare dati con RSA in formato standard per evitare attacchi; tante versioni diversi scopi (1-15).

PKCS-1: standard per formato messaggi (byte)

$m = 0 || 2 || \text{almeno 8 byte non zeri} || 0 || M$

(M rappresenta i dati da spedire)

- primo byte 0 implica che il messaggio è minore n
- secondo byte (2) indica codifica (1 indica firma) e implica messaggio grande
- byte casuali implicano
 - messaggio grande;
 - stesso messaggio inviato a più utenti è diverso;
 - attaccante che ha info su m non può codificare e verificare

RSA e Data integrity: OAEP

Problema: implementazione di PKCS-1 può creare un attacco (detto 1 in un milione)

Risposta OAEP (o anche PKCS-1 versione 2)

- codif. $c = \text{RSA}[M || 0^{k_1} \text{ xor } G(r)] || [H(\text{RSA}[M || 0^{k_1} \text{ xor } G(r)]) \text{ xor } r]$
- k_0, k_1 noti; G, H funzioni hash, note; r stringa casuale di k_0 bit (scelta dal mittente);
- decod. $cd = s || t$ (con t di k_0 bit); $u = t \text{ xor } H(s)$; $v = s \text{ xor } G(u)$; accetta se $v = m || 0^{k_1}$; altrimenti rifiuta
- r casuale implica che OAEP è robusto anche nel caso di attacco chosen ciphertext (assumendo H e G buone funzioni hash)

RSA: conclusioni

- Dopo tanti anni RSA è ancora metodo a chiave pubblica più usato
- Nella versione Standard: Robusto a tutti gli attacchi
- Problemi per uso effettivo velocità codifica/decodifica:
 - uso di RSA per inviare chiave segreta (uso per un solo messaggio)
 - limiti in uso per dispositivi con ridotte capacità di calcolo

Logaritmo Discreto

Se p è primo allora Z_p^* è un gruppo ciclico cioè esiste un generatore g tale che $G = \{g, g^2, g^3, \dots, g^{p-1}\}$

es. $p=7$ $g=5$

- Dato primo p e un generatore g di Z_p^* e y trova x t.c. $y = g^x \pmod p$ x è il *logaritmo discreto* di y in base g

Il logaritmo discreto è considerato un problema computazionalmente difficile

- dato x calcolare g^x è facile (computazionalmente veloce).
- dato g e g^x si crede sia difficile trovare x

Logaritmo Discreto come funzione one way

DLA (Assunzione del Logaritmo discreto) non è possibile in tempo polinomiale calcolare il logaritmo discreto

- migliore algoritmo noto: richiede tempo \gg di $p^{1/3}$

Nota Bene la lunghezza di p è data dal numero di bit di una codifica binaria di p e, quindi è $O(\log p)$

Se DLA vale (e assumiamo di sì) allora il log discreto è funzione one-way

Se varia p (e g) possiamo ottenere una collezione di funzioni trapdoor

Scambio di chiavi di Diffie-Hellman

Problema: stabilire una chiave segreta utilizzando un canale di comunicazione pubblico

Parametri pubblici: un primo p , e un elemento g (possibilmente un generatore del gruppo moltiplicativo Z_p^*)

- Alice sceglie a caso a in $[1..p-2]$ e manda $g^a \bmod p$ a Bob.
- Bob sceglie a caso b in $[1..p-2]$ e manda $g^b \bmod p$ a Alice.
- Alice e Bob calcolano la chiave $g^{ab} \bmod p$:

Bob conosce b e $g^a \bmod p$ calcola

$$(g^a \bmod p)^b \bmod p = g^{ab} \bmod p$$

Alice conosce a e $g^b \bmod p$ calcola

$$(g^b \bmod p)^a \bmod p = g^{ab} \bmod p$$

DH - Requisiti di Sicurezza

Meccanismo "costruttivo": la chiave segreta deve utilizzare sia informazioni pubbliche che segrete in modo opportuno.

- Requisito di sicurezza: la chiave segreta è una funzione one way dell'informazione pubblica e della informazione trasmessa.
- DH è almeno tanto difficile quanto il DL in Z_p . L'equivalenza formale non è nota anche se ci sono indicazioni parziali. Dopo 25 anni di attacchi è ancora considerato sicuro.
- Veloce anche con p di 512-1024 bit $O(\log^3 p)$.
- Per avere chiave DES uso primi 56 bit della chiave (perché non è possibile scegliere p di soli 56 bit?)

DH - Requisiti di Sicurezza

Attacco Man-in-the-middle:

Trudy si intromette tra A e B e effettua uno scambio di DH con Alice e Bob

Alice ----- Trudy ----- Bob

- Alice definisce una chiave con Trudy (pensando di farlo con Bob)
- Bob definisce una chiave con Trudy (pensando di farlo con Alice)
- Trudy ha due chiavi per colloquiare con A e B!!

L'attacco è letale: lo scambio di chiavi di DH Key è effettivo solo in presenza di un attaccante passivo.

DH - Requisiti di Sicurezza

Attacco Man-in-the-middle: soluzione

- autentica dei messaggi es. firma/autentica messaggio scambio;
- codifica segreta dello scambio

serve che A e B condividano una chiave segreta oppure si utilizza chiave pubblica

DH - Requisiti di Sicurezza

Chiavi Diffie Hellman pubbliche:

- stesso g e p per tutti gli utenti;
- ogni utente sceglie a una volta per tutte e lo usa sempre

serve un meccanismo analogo a autorità certificazione delle informazioni pubbliche

Algoritmo di El-Gamal

- Simile a Diffie Hellmann (poco usato)
- Alice pubblica p, g , g generatore gruppo Z_p
- Alice sceglie x come chiave privata e pubblica $y=g^x \bmod p$ come chiave pubblica
- Per codificare $m \in Z_p$ Bob sceglie a caso k e spedisce $(c=g^k \bmod p, d= my^k \bmod p)$

Nota: Richiede due esponenziazioni per ogni blocco trasmesso

El-Gamal: decodifica

- Alice sceglie x come chiave privata e pubblica $y=g^x \bmod p$ come chiave pubblica
- Per codificare $m \in Z_p$ Bob sceglie a caso k e spedisce $(c= g^k \bmod p, d= my^k \bmod p)$

Decodifica: Alice calcola $m = d / c^x$

Infatti: $c^x = (g^k)^x = (g^x)^k = y^k = d/m \pmod{p}$