



Approfondimento Esame Orale

Sicurezza Sistemi Informatici e Crittografia

Algoritmo Diffie – Hellman

Daide Carnevali
Matricola 048293
carned@fastwebnet.it

1. Introduzione
2. Funzionamento
 - Proprietà Matematiche
 - Sicurezza
3. Implementazione
4. Svantaggi
 - Falla di sicurezza
5. Diffie – Hellman Autenticato – STS
 - Notazioni
 - Piccola Variante : Conferma della Chiave
6. Protocollo MTI
 - Autenticazione Implicita
7. Riferimenti Bibliografici

1. Cos'è Diffie – Hellman

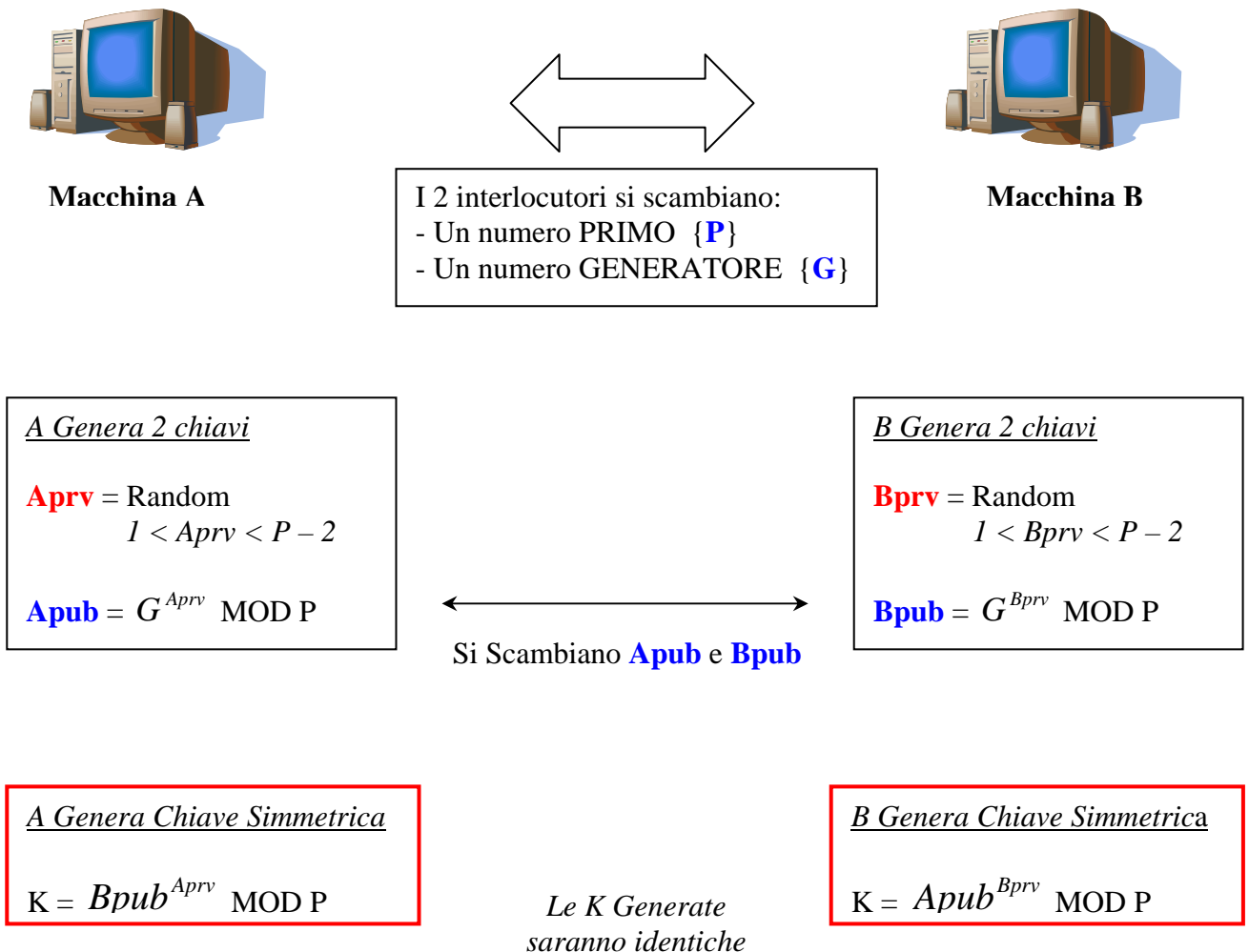
Diffie-Hellman è uno dei primi algoritmi (1976) per la distribuzione di chiavi segrete.

A differenza degli altri algoritmi asimmetrici, l'algoritmo Diffie-Hellman consente a due interlocutori di "scambiarsi" in maniera sicura una chiave simmetrica attraverso un canale pubblico, per definizione non sicuro.

In realtà i due interlocutori non si scambiano la chiave, ma la generano sfruttando alcune proprietà matematiche dei numeri primi e numeri generatori.

La sicurezza dell'algoritmo si basa principalmente sulla complessità computazionale del calcolo del logaritmo discreto.

2. Funzionamento



2.1 Proprietà Matematiche

P : Numero primo molto elevato (es. 1024bit)

P deve essere nella forma $P = 2Q + 1$, dove Q è un altro numero primo generato casualmente.

G : Numero generatore di P, che sicuramente esiste dato che P è primo.

Essendo P primo, il numero generatore G sarà un valore che elevato successivamente a potenza genera tutti i numeri compresi tra 1 e P-1.

K : La chiave generata sarà identica in quanto per entrambi vale la proprietà:

$$K = G^{A_{prv} * B_{prv}} \text{ MOD } P$$

Essendo $A_{pub} = G^{A_{prv}} \text{ MOD } P$ e $B_{pub} = G^{B_{prv}} \text{ MOD } P$ vale la formula sopra descritta.

2.2 Sicurezza

L' algoritmo si può definire sicuro in quanto un attaccante C che intercetta i 4 valori scambiati tra A e B { **P**, **G**, **Apub**, **Bpub** } non è in grado di generare la chiave **K** non essendo a conoscenza dei valori privati **Aprv** e **Bprv**.

Teoricamente l'attaccante potrebbe risalire alle 2 chiavi private, in quanto

$$\mathbf{Aprv} = \text{Log}_G \text{Apub} \text{ MOD } P \text{ e } \mathbf{Bprv} = \text{Log}_G \text{Bpub} \text{ MOD } P$$

ma essendo il calcolo del logaritmo discreto molto complesso dal punto di vista computazionale per numeri molto grandi (es. 1024bit) è pressoché impossibile.

Per questo motivo, dal punto di vista matematico, dopo 30anni è considerato ancora un algoritmo molto valido, infatti alcune varianti (di cui parleremo in seguito) sono implementate in protocolli di crittografia odierna (es. SSH).

3. Implementazione

Questa implementazione è basata sul linguaggio java, e ha il solo scopo di voler dimostrare che la chiave generata da due interlocutori è identica, cioè che le proprietà matematiche descritte sinora siano verificate.

Generazione numero Primo P

Genero un numero primo P nella forma $P = 2Q + 1$

```
while(!isPrime(p)) {
    q = generatorPrime(Long.MAX_VALUE/2);
    p = 2*q + 1;
}

static long generatorPrime(long max){
    long n = ((long) (Math.random() * max ));
    while (!isPrime(n)) {
        n++;
    }
    return n;
}

static boolean isPrime(long n) {
    //Se è un numero pari, allora non è primo.
    if (n % 2 == 0) {
        return false;
    }
    //Altrimenti controllo che non sia divisibile per tutti numeri dispari che ci sono tra
    //3 e la radice quadrata del numero
    for (long i = 3, end = (long)Math.sqrt(n); i <= end; i += 2) {
        if (n % i == 0) {
            return false;
        }
    }
    return true;
}
```

Trovare numero Generatore G di P

Se il numero primo P è nella forma $P = 2Q + 1$

Allora un numero generatore di P ha le seguenti proprietà:

$$G^Q \text{ MOD}(P) \neq 1 \text{ e } G^{2Q} \text{ MOD}(P) \neq 1$$

```
g = p-1;
while ( (potMod(g,q,p)==1.0) || (potMod(g,2,p)==1.0) ){
    g--;
}
```

X elevato alla Y MOD Z

Questa funzione risulta utile per il trovare il numero generatore, A_{pub} , B_{pub} e K .

In java esiste già una funzione che realizza l'elevamento a potenza : `Math.pow(x,y)`, ma è risultata poco precisa nell'operare su variabili di tipo `LONG` in quanto ritorna un `DOUBLE`.

```
static long potMod(long x, long y, long N){
    long tmp = 0;
    if (y==1) return (x % N);

    if (((int)y&1)==0){
        tmp = potMod(x,y/2,N);
        return ((tmp * tmp) % N);
    }
    else{
        tmp = potMod(x,(y-1)/2,N);
        tmp = ((tmp * tmp) % N);
        tmp = ((tmp * x) % N);
        return (tmp);
    }
}
```

4. Svantaggi – Falla di sicurezza

Il principale problema dell'algoritmo Diffie-Hellman consiste nel fatto che non fornisce alcun meccanismo di autenticazione dei partecipanti.

Questo problema genera la vulnerabilità all'attacco *Man-in-the-Middle*.

Quando i 2 interlocutori si scambiano le proprie chiavi pubbliche (A_{pub} e B_{pub}) un attaccante potrebbe intercettarle e interferendo nella comunicazione inviare la propria chiave pubblica (C_{pub}) sia ad A che a B fingendosi di essere rispettivamente B ed A .

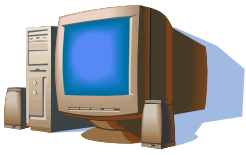
In questa maniera C genera una chiave segreta $K1$ con A , e un'altra chiave simmetrica $K2$ con B , riuscendo così ad intercettare e leggere tutta la comunicazione tra A e B .

Logicamente durante una comunicazione (es. da A a B), C dovrà decriptare con $K1$ (per poter leggere il messaggio) e criptare con $K2$ prima di rinviare il messaggio a B , così da rendersi invisibile ai 2 interlocutori.

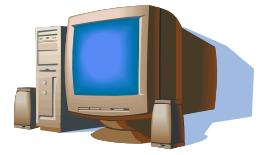
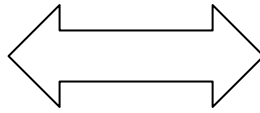
5. Diffie Hellman Autenticato – STS

Nel 1992 è stata sviluppata una versione modificata dell'algoritmo, detta *authenticated Diffie-Hellman key agreement protocol* o *protocollo STS (Station-to-Station)* che utilizza la firma dei messaggi ed i certificati digitali ovviando le problematiche descritte precedentemente.

Prima di eseguire lo scambio di chiavi, A e B sono ciascuno in possesso di una coppia di chiavi (pubblica/privata) ed un certificato digitale relativo alla chiave pubblica. Durante lo scambio, A esegue una firma dei messaggi inviati che contengono la sua chiave pubblica A_{pub} e lo stesso fa B . In questo modo anche se C riceve i messaggi di A e B non è in grado di firmarli a nome degli altri, quindi se li modifica, A e B si accorgono che i messaggi ricevuti non corrispondono a quelli inviati dal mittente.



Macchina A



Macchina B

I 2 interlocutori si scambiano:
 - Un numero PRIMO {P}
 - Un numero GENERATORE {G}

A Genera 2 chiavi
Aprv = Random
 $1 < Aprv < P - 2$
Apub = $G^{Aprv} \text{ MOD } P$

B Genera 2 chiavi, Identità, Firma
Bprv = Random
 $1 < Bprv < P - 2$
Bpub = $G^{Bprv} \text{ MOD } P$
Y(B) = $SIG_B(Apub, Bpub)$
C(B)

A Verifica Identità e Firma di B
C(B) = ?
Y(B) = ?
A Genera Identità e Firma
Y(A) = $SIG_A(Apub, Bpub)$
C(A)

B Verifica Identità e Firma di A
C(A) = ?
Y(A) = ?

Invia **Apub**

Invia **C(B)**, **Bpub**, **Y(B)**

Invia **C(A)**, **Y(A)**

B Genera Chiave Simmetrica
KEY = $Apub^{Bprv} \text{ MOD } P$

B Genera Chiave Simmetrica
KEY = $Apub^{Bprv} \text{ MOD } P$

E' Proprio l'uso delle firme digitale che contrasta l'attacco *Man-In-The-Middle*.

5.1 Notazioni - STS

Certificato $C(U) = \{ID(U), ver_U, Sig_{TA}(ID(U), ver_U)\}$

1. ID è un numero univoco che ciascun interlocutore possiede.
2. Ver_U è l'algoritmo di verifica che viene utilizzato per controllare la firma Sig_U
3. Sig_{TA} è un numero ottenuto dal Central Trusted Authority, in pratica la firma dell'identificazione ID e dell'algoritmo Ver_U .

Y(A) e Y(B) Firma digitale delle 2 chiavi pubbliche.

Y(B) = ? A verifica Y(B) utilizzando l'algoritmo Ver_B

C(B) = ? A verifica C(B) utilizzando l'algoritmo Ver_{TA}

5.2 Piccola Variante

Il Protocollo sopra descritto non prevede la conferma della chiave generata (**KEY**). Tuttavia è facile introdurre la conferma ridefinendo Y(A) e Y(B) nel seguente modo:

$$Y(A) = E_{KEY}(SIG_A(Apub, Bpub))$$

$$Y(B) = E_{KEY}(SIG_B(Apub, Bpub))$$

Chiaramente la chiave KEY non viene più generata alla fine della comunicazione ma prima di inviare i propri certificati.

6. Protocollo MTI

Matsumoto, Takashima e Imai hanno realizzato un'interessante modifica al protocollo STS eliminando l'uso delle firme digitali, ma limitando comunque gli attacchi di tipo *Man-In-the-Middle*. Una volta scambiato il numero primo **{P}** e il valore generatore **{G}** ogni utente genera 2 chiavi pubbliche ($Pub1$ $Pub2$) e 2 private ($Prv1$ $Prv2$) anziché di una come nel protocollo STS che nell'algoritmo originale. Le chiavi vengono generate nella stessa maniera degli algoritmi precedenti.

Ogni interlocutore possiede un certificato

$$C(U) = \{ID(U), Pub1_U, Sig_{TA}(ID(U), Pub1_U)\}$$

ove $Pub1_U$ è una delle 2 chiavi pubbliche generate.

La chiave simmetrica sarà generata nella seguente maniera

$$KEY_A = Pub2_A^{Prv1_B} * Pub1_A^{Prv2_B} \text{ MOD } P$$

$$KEY_B = Pub2_B^{Prv1_A} * Pub1_B^{Prv2_A} \text{ MOD } P$$

Chiaramente KEY_A e KEY_B saranno identiche.

6.1 Autenticazione Implicita

Il non utilizzo delle firme da nuovamente la possibilità ad un attaccante di porsi in mezzo alla comunicazione e alterare i valori scambiati dagli utenti, ma l'uso dei certificati limita l'attacco in quanto non è possibile "falsificare" il mittente, cosa che avviene nell'attacco *Man-In-the-Middle*.

Se l'attaccante altera le chiavi pubbliche scambiate, i due utenti non genereranno la stessa chiave simmetrica (che naturalmente sarà inutile a loro), ma nessuna di queste chiavi sarà computabile dall'attaccante.

Quindi A e B si assicurano che l'altro sia l'unico utente nella rete che possa computare la chiave che hanno computato. Tale proprietà è detta *Autenticazione Implicita*.

7. Riferimenti Bibliografici

Cryptography: Theory and Practice

by Douglas Stinson
ISBN: 0-84-938521-0
Pub Date: 03/17/95

Modern Cryptography: Theory and Practice

by Wenbo Mao Hewlett-Packard Company
Publisher: Prentice Hall PTR
ISBN: 0-13-066943-1
Pub Date: July 25, 2003

Generating Prime Numbers and When Not to Overload Methods

Technical Articles and Tips

<http://java.sun.com/developer/JDCTechTips/2002/tt0806.html>

RFC 2631 - Diffie-Hellman Key Agreement Method

Internet RFC/STD/FYI/BCP Archives
<http://www.faqs.org/rfcs/rfc2631.html>