Tecniche di Progettazione Automatica **Anno Acc. 1995-96**

PROF. ALFREDO DE SANTIS

GESTIONE DI CHIAVI

a cura di Blandi Luciano e Muto Nicola *22 Luglio 1996*

Indice

1 Introduzione **2**

2 Distribuzione di Chiavi **4**

2.1 Predistribuzione di Chiavi 4

2.1.1 Sicurezza Totale 5

2.1.2 Schema di Blom 6

2.2 Distribuzione di chiavi con Server On-Line 16

2.2.1 Needham-Schroeder 18

2.2.2 Kerberos 19

3 Accordo di Chiavi **24**

3.1 Accordo tramite protocollo a chiave pubblica 24

3.2 Schema di Diffie-Hellman 25

3.2.1 Il Protocollo 25

3.2.2 Equivalenza tra ElGamal e Diffie-Hellman 28

3.2.3 Chiavi di Conferenza 29

3.3 Schema di Merkle 31

1 Introduzione

Quando si parla di sicurezza in una rete, il problema fondamentale riguarda principalmente la possibilità di poter permettere a una qualsiasi coppia di utenti lo scambio di messaggi, senza correre il rischio che gli altri possano carpire le informazioni trasmesse.

Si potrebbe utilizzare un sistema a chiave pubblica, il quale permette ad ogni utente di poter comunicare in modo sicuro con qualsiasi altro utente mantenendo solo la propria chiave segreta mentre la chiave pubblica di ognuno è mantenuta in un database pubblico.

Oltre al vantaggio dovuto al fatto che tale sistema non necessita, come quello a chiave privata, di un canale sicuro per lo scambio di chiavi segrete, consentendo anche al singolo utente di mantenere un basso numero di informazioni (solo la propria chiave privata), esso offre inoltre una grande flessibilità di gestione. Infatti, quando un nuovo utente viene aggiunto alla rete, è sufficiente che esso generi solo le sue due chiavi, pubblica e privata, e inserisca la prima nel file pubblico.

Sfortunatamente però questi metodi di codifica sono molto più lenti dei sistemi a chiave privata. Quindi quello che si fa in pratica è di utilizzare le tecniche di codifica a chiave pubblica per messaggi di piccole dimensioni, e codificare le informazioni con una tecnica a chiave privata se la dimensione delle stesse è elevata.

Analizziamo a questo punto le problematiche inerenti l'utilizzo di quest'ultimo metodo di codifica.

In un sistema a chiave privata, due utenti utilizzano un'unica chiave comune per scambiarsi messaggi codificati. Ovviamente, affinché i dati siano protetti da eventuali nemici, la chiave deve essere nota solo alle due parti.

Questo approccio al problema di codifica, presenta però un inconveniente. Come detto prima la condizione necessaria affinché il sistema di codifica possa funzionare in modo sicuro è che le chiavi siano note solo agli utenti.

In generale, quindi, per una rete con n utenti, devono essere generate $\binom{n}{2} = \frac{n(n-1)}{2}$ chiavi distinte.

Al problema di codifica/decodifica **sicura** dei messaggi all'interno di una rete non sicura, si affianca quindi il problema della Gestione di Chiavi nel momento in cui si utilizza un sistema a chiave privata in luogo di un sistema a chiave pubblica. In questo caso infatti sorge il problema di distribuire in modo sicuro a tutte le parti le chiavi, tenendo conto che:

- abbiamo a disposizione una rete non sicura;
- tutte le possibili coppie di utenti devono essere in possesso degli strumenti per poter comunicare tra loro;
- nessun utente non autorizzato deve poter entrare in possesso di una chiave comune ad altri utenti.

È importante sottolineare che le tecniche di Gestione di Chiavi, si occupano solamente del problema di trasferire le chiavi agli utenti in modo sicuro. Queste stesse chiavi, verranno poi utilizzate dagli utenti per comunicare tra loro, utilizzando un qualsiasi algoritmo di codifica (ad esempio il DES).

La soluzione del problema della Gestione di Chiavi può essere affrontato in vari modi. Si identificano però due grandi famiglie di metodi:

- **Distribuzione di Chiavi**

- **Accordo di Chiavi**

La distribuzione di chiavi viene definita come un meccanismo tramite il quale una parte sceglie una chiave segreta e la trasmette ad un'altra parte o a più parti. L'accordo di chiavi specifica un meccanismo con cui due o più parti stabiliscono insieme una chiave segreta che consenta loro di comunicare su un canale pubblico. Va ricordato che in uno schema di accordo di chiavi il valore della chiave è determinato come una funzione degli input forniti da entrambe le parti.

In entrambi i casi, gli utenti seguono un *Protocollo* comune di gestione delle chiavi. Essi cioè eseguono un insieme predefinito di passi che deve godere delle seguenti proprietà:

- tutti i partecipanti devono conoscere tutti i passi che costituiscono il protocollo;
- tutti i partecipanti devono seguire correttamente il protocollo;
- il protocollo deve essere non ambiguo;
- il protocollo deve specificare le operazioni da eseguire per tutti le possibili situazioni.

Visto inoltre, che la rete non è sicura, dobbiamo tener conto dei possibili attacchi che possono essere portati al nostro protocollo dai *nemici*, cioè da utenti non appartenenti alla rete che intercettano i messaggi cifrati scambiati durante le fasi di un protocollo cercando di scoprirne il contenuto. Questi possono essere divisi in due categorie principali:

- **nemici passivi**: possono solamente intercettare il messaggio tra due utenti senza possibilità di interferire nella comunicazione;
- **nemici attivi**: questi hanno la possibilità di alterare il messaggio sulla linea, salvare il messaggio per riutilizzarlo in un secondo momento ed essere visti nella rete come un qualsiasi utente.

L'obiettivo di un nemico attivo potrebbe essere uno dei seguenti:

- ingannare due utenti A e B inducendoli ad accettare come valida una chiave "non valida" (una chiave non valida potrebbe essere una vecchia chiave scaduta, cioè che ha superato il suo periodo di validità, oppure una chiave scelta dal nemico);
- far credere agli utenti A e B che si sono scambiati una chiave quando invece non lo hanno fatto.

Nell'analisi della sicurezza di un protocollo dobbiamo tener presente anche gli attacchi che possono essere portati dagli utenti della stessa rete. L'ipotesi forte che facciamo inoltre è di utilizzare un sistema sicuro di codifica e uno schema sicuro per la generazione delle chiavi. Non ci interesseremo, cioè, della possibilità di rompere il sistema di cifratura, né dei possibili buchi nello schema di generazione delle chiavi, ma solo dei possibili attacchi alla struttura del protocollo, in ogni suo passo.
In alcuni casi, infatti, la sicurezza del protocollo prescinde (ad es. Predistribuzione di Chiavi) dalla sicurezza del metodo di codifica, in quanto, le chiavi vengono comunicate su canali sicuri, senza bisogno quindi di codificare le stesse. In altri casi, invece, le chiavi debbono essere codificate con un algoritmo sicuro, utilizzando informazioni comuni alle due parti. Vedremo che,

a seconda del tipo di approccio, le due parti possono essere un utente ed un utente "Speciale" (Distribuzione di Chiavi con server on-line), oppure due utenti (Accordo di Chiavi). In ogni caso, le due parti codificano la chiave che utilizzeranno per la trasmissione, con informazioni che hanno stabilito precedentemente. Studieremo particolarmente quindi, il modo di stabilire le informazioni comuni che permettono, in un secondo momento, lo scambio di chiavi che verranno poi utilizzate per codificare le informazioni che gli utenti si scambieranno.

2 Distribuzione di Chiavi

In tutti gli approcci di questo tipo si identifica nella rete una figura responsabile della scelta e trasmissione delle chiavi, e della verifica dell'identità degli utenti.

Si associa a questa figura il nome di *Autorità Fidata* o *TA* (Trusted Authority); questo nome deriva dal fatto che la TA conosce le chiavi di tutti gli utenti che si trovano nella rete e quindi, deve avere la fiducia di questi, dato che se un utente riuscisse a corrompere questa autorità, potrebbe accedere incondizionatamente alle informazioni che gli altri utenti si scambiano. Praticamente, il ruolo della TA viene effettuato, in modo automatico, da una macchina all'interno della rete, detta *Server*; identificheremo d'ora in poi col nome Server la figura della TA.
Vi sono due approcci per la distribuzione delle chiavi:

- **Predistribuzione di Chiavi**
  In una prima fase, il Server distribuisce a tutti gli utenti della rete dei pezzi di informazione tramite i quali, successivamente, una qualsiasi coppia di utenti riuscirà a calcolare la chiave segreta comune con cui comunicare in maniera sicura.
  Dunque, nella seconda fase, gli utenti possono comunicare tra loro senza fare più riferimento al Server, il quale, non partecipando in alcun modo alla comunicazione, è come se scomparisse dalla rete.
  Si dice, in questo caso, che il Server è *off-line*.

- **Distribuzione di chiavi con Server on-line**
  Nella prima fase il Server distribuisce ad ogni utente una chiave segreta; ogni utente, quindi, condivide con il Server una propria chiave non nota a qualunque altro utente della rete.
  Nella fase successiva, due utenti che vogliano comunicare tra loro non possono farlo indipendentemente dal Server, ma devono rivolgersi ad esso per ottenere la chiave di sessione, ossia la chiave con cui poter comunicare in maniera sicura.
  Ad ogni richiesta di comunicazione il Server fornisce una nuova chiave alle due parti; quindi, il Server è funzionante per tutto il protocollo.
  Si dice, in questo caso, che il Server è *on-line*.

Analizziamo ora meglio questi due approcci fornendo dei protocolli per la gestione delle chiavi.

2.1 Predistribuzione di Chiavi

Come abbiamo appena detto, in questo approccio il Server è off-line, ossia distribuisce in modo sicuro dei pezzi di informazione agli utenti della rete, dopodiché esso non partecipa in nessun modo alla comunicazione.
Vediamo ora alcuni protocolli di distribuzioni di chiavi con Server off-line.

2.1.1 Sicurezza Totale

Supponiamo di avere una rete con n utenti.
Il Server comincia col generare una chiave per ogni coppia di utenti, diversa da tutte le altre in maniera tale che possa essere usata solo da questi per codificare i propri messaggi. In totale, quindi vi sono $\binom{n}{2} = \frac{n(n-1)}{2}$ chiavi distinte e queste devono essere state generate indipendentemente l'una dall'altra.

Dopo aver generato tutte le chiavi il Server dovrà opportunamente distribuirne $n-1$ ad ogni utente, che permetteranno a quest'ultimo di comunicare con gli altri $n-1$ utenti. È chiaro che una volta distribuite queste chiavi, gli utenti della rete non hanno più bisogno del Server per comunicare tra di loro e quindi la figura del Server può a questo punto scomparire.

Per quanto riguarda la sicurezza del protocollo appena descritto, possiamo affermare che questo è il più sicuro che si possa avere.

Infatti, la chiave comune a qualunque coppia di utenti non dipende dalle informazioni che hanno tutti gli altri utenti, proprio perché ogni chiave è stata generata indipendentemente dalle altre.
Quindi, anche se altri $n-2$ utenti si accordassero per cercare di capire quale sia questa chiave, qualsiasi cosa facessero non scoprirebbero nulla. Da questo punto di vista, dunque, la sicurezza del protocollo è totale.
Vediamo ora gli svantaggi di un simile protocollo. Il problema di base è che ogni utente deve mantenere $n-1$ chiavi, ossia deve memorizzare un gran numero di informazioni, specie se n è abbastanza grande; il Server, inoltre, deve distribuirne $n-1$ ad ogni utente della rete, quindi un totale di $n(n-1)$ chiavi (questo è chiamato spesso "problema n^2"), e deve farlo in modo sicuro, ad esempio attraverso un canale privato.

Un altro inconveniente è che l'upgrade di questo sistema risulta enormemente costosa.
Infatti, se la rete è di tipo dinamico e vi è un $(n+1)-esimo$ utente che vuole far parte della rete, il Server deve generare n nuove chiavi diverse dalle precedenti, fornirle al nuovo utente e distribuirne una ciascuno ai vecchi utenti della rete.

Il grande numero di chiavi, la grande quantità di informazioni da trasferire in modo sicuro insieme al problema della non flessibilità del sistema rispetto all'aggiunta di un nuovo utente, rende questa politica di gestione non utilizzabile se non per reti di piccole dimensioni. La sicurezza totale, inoltre, potrebbe non essere un requisito essenziale: si potrebbe infatti assumere che i "cattivi" della rete siano non più di un certo numero fissato. Quindi, da questo punto di vista, la sicurezza di questo schema è maggiore di quella che il più delle volte si rende necessaria.

Si cercherà allora di ridurre la quantità di informazione che si deve trasmettere e memorizzare, consentendo ancora che ciascuna coppia di utenti A e B possa calcolarsi indipendentemente una chiave segreta $K_{A,B}$ per comunicare. Uno schema elegante (chiamato **Schema di Predistribuzione di Chiavi di Blom**) che realizza questo viene discusso nel prossimo paragrafo.

2.1.2 Schema di Blom

Supponiamo sempre che nella rete vi siano n utenti.
In questo schema, il Server di tipo off-line distribuisce dei pezzi di informazioni ad ogni utente, in base alle quali ognuno di essi può calcolare le chiavi segrete che gli permettono di comunicare con qualsiasi altro utente in maniera sicura.

Nello schema di Blom, vengono fissati inizialmente due parametri: b e c. Il primo definisce il numero di utenti che possono comunicare tra loro utilizzando un'unica chiave detta *chiave di conferenza*, e può assumere valori compresi tra 2 ed n; il secondo identifica il grado di sicurezza dello schema adottato e può assumere valori compresi tra 1 ed $n - b$.

Definiamo ora meglio in cosa consiste la sicurezza di questo schema:

Definizione 2.1 (c-Sicurezza) *Un protocollo si dice c-sicuro se considerato un qualsiasi insieme di al più c utenti, questi non possono in alcun modo, con le informazioni ricevute dal Server e con quelle che sono state rese pubbliche, ottenere alcun tipo d'informazione sulla chiave di conferenza di un qualsiasi altro insieme di b utenti disgiunto dal precedente.*

Se si accordassero più di c utenti allora questi potrebbero ricavare tale chiave.
Notiamo, inoltre, che qui si sta parlando di *c-sicurezza* incondizionata (come verrà dimostrato in seguito).

Quindi, la sicurezza di questo schema non è di tipo totale, ma è una sicurezza limitata dall'assunzione che nella rete non vi siano più di c utenti *cattivi* che vogliono carpire le informazioni di altri b utenti *buoni*.
Vediamo ora come sia possibile realizzare un simile protocollo.

Lo schema di Blom si basa sulle proprietà dei polinomi simmetrici.

Definizione 2.2 (Polinomio Simmetrico) *Sia p un intero positivo. Un polinomio*

$$f(x_1, x_2, \ldots, x_n)$$

con coefficienti in Z_p si dice simmetrico se risulta:

$$f(x_1, x_2, \ldots, x_n) = f(x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})$$

per ogni permutazione σ di $\{1, 2, \ldots, n\}$ e per ogni n-pla $(x_1, x_2, \ldots, x_n) \in Z_p^n$.

Consideriamo inizialmente il caso speciale della comunicazione tra una coppia di utenti ($b = 2$), utilizzando uno schema *1-sicuro* ($c = 1$), cioè uno schema in cui il Server distribuisce delle informazioni ad ogni utente in modo tale che ogni coppia di essi riesce a calcolarsi la propria chiave, ed un terzo utente che voglia scoprire tale chiave non vi riuscirà in alcun modo, mentre una coalizione di più utenti potrebbe riuscirvi.
Estenderemo poi il protocollo al caso della comunicazione tra b utenti nello schema *c-sicuro.*

Riportiamo di seguito il protocollo di Blom per *b=2* e *c=1* in ogni suo passo:

Protocollo di Blom

1. Viene scelto un numero primo p maggiore di n; ad ogni utente **U** si associa un identificativo $ID_U \in Z_p$ unico. Queste informazioni vengono rese pubbliche.

2. Il Server sceglie a caso tre elementi $r, s, t \in Z_p$ (non necessariamente distinti) e costruisce il seguente polinomio:
$$f(x, y) = r + s(x + y) + txy \bmod p. \tag{1}$$

3. Per ogni utente **U** il Server calcola il polinomio
$$\psi_U(x) = f(x, ID_U) \bmod p \tag{2}$$
e trasmette tale polinomio all'utente **U** tramite un canale sicuro. Si noti che $\psi_U(x)$ è un polinomio lineare in x e può essere scritto come
$$\psi_U(x) = a_U + b_U x,$$
dove
$$a_U = r + sID_U \bmod p$$
e
$$b_U = s + tID_U \bmod p.$$

4. Se due utenti **A** e **B** vogliono comunicare tra loro, utilizzano la seguente chiave comune:
$$k_{B,A} = \psi_B(ID_A) = \psi_A(ID_B) = k_{A,B}. \tag{3}$$

Illustriamo lo schema di Blom con $c = 1$ nel seguente esempio.

Esempio 2.3 (Schema di Blom 1-Sicuro) *Supponiamo che p=17 e che gli identificativi dei tre utenti* **A**, **B** *e* **C** *siano rispettivamente* $ID_A = 12$, $ID_B = 7$ *e* $ID_C = 1$. *Supponiamo che il Server scelga* $r = 8$, $s = 7$ *e* $t = 2$, *il polinomio simmetrico* f *è*
$$f(x, y) = 8 + 7(x + y) + 2xy.$$

I polinomi ψ *dati ai tre utenti sono*
$$\begin{aligned}\psi_A(x) &= 7 + 14x,\\ \psi_B(x) &= 6 + 4x,\\ \psi_C(x) &= 15 + 9x.\end{aligned}$$

Quindi le tre chiavi sono
$$\begin{aligned}K_{A,B} &= 3,\\ K_{A,C} &= 4,\\ K_{B,C} &= 10.\end{aligned}$$

A *calcola* $K_{A,B}$ *come*

$$\psi_A(ID_B) = 7 + 14 \times 7 \bmod 17 = 3.$$

B *calcola* $K_{A,B}$ *come*

$$\psi_B(ID_A) = 6 + 4 \times 12 \bmod 17 = 3.$$

Il calcolo delle altre chiavi è del tutto analogo. □

Analizziamo meglio ogni passo di questo protocollo:

1. Al primo passo viene scelto un primo p maggiore di n. Il motivo per il quale p debba essere maggiore di n sta nel fatto che ad ogni utente è associato un identificativo tramite il quale l'utente stesso viene riconosciuto (identificato) sia dal Server che dagli altri utenti della rete. Se a due utenti venisse associato lo stesso identificativo allora il Server trasmetterebbe ad entrambi lo stesso polinomio $\psi(x)$ e quindi i due utenti calcolerebbero le stesse chiavi; ciò permetterebbe ad entrambi di decifrare i messaggi che l'altro spedisce sulla rete, e ciò deve essere evitato.
   È essenziale quindi che ogni identificativo sia unico, cioè diverso da tutti gli altri. Ovviamente, poiché un identificativo è un valore appartenente a Z_p, la condizione necessaria affinché il Server possa scegliere identificativi diversi è appunto quella di fissare il valore di p maggiore del numero n di utenti presenti nella rete.
   Il parametro p indica inoltre la base dell'aritmetica modulare utilizzata per il calcolo delle chiavi. Queste ultime avranno quindi la stessa lunghezza di p. Dunque p deve essere tanto grande da permettere di ottenere delle chiavi di lunghezza opportuna.
   Ad esempio, se due utenti decidono di trasmettersi dei messaggi utilizzando il DES allora p deve essere un valore di almeno 56 bit; il fatto che p possa essere più grande di 56 bit non è un problema perché i due utenti possono calcolarsi la loro chiave e decidere ad esempio di utilizzare come chiave DES solo i primi 56 bit di tale chiave.

2. Il polinomio costruito dal Server ha la seguente forma:

   $$f(x, y) = r + s(x + y) + txy \bmod p.$$

   Questo è un polinomio in due variabili ed è di primo grado. Infatti, le variabili x e y compaiono sempre con esponente al più uguale a 1.
   Inoltre, questo polinomio è simmetrico. Per provarlo basta verificare che $f(x, y) = f(y, x)$ *per ogni* $x, y \in Z_p$.
   Fissiamo dunque $x, y \in Z_p$. Risulta che

   $$\begin{aligned} f(x,y) &= r + s(x+y) + txy \bmod p \\ &= r + s(y+x) + tyx \bmod p \\ &= f(y,x). \end{aligned}$$

   Infatti le operazioni di somma e di prodotto modulare godono della proprietà commutativa.
   Osserviamo infine che il polinomio è stato scelto in maniera uniforme tra tutti i possibili polinomi di questo tipo. Per far ciò basta infatti scegliere a caso i tre valori $r, s, t \in Z_p$, e questo viene realmente fatto dal Server.

3. In questo protocollo il Server distribuisce ad ogni utente **U** solo il polinomio $\psi_U(x)$, mentre tutte le altre informazioni (identificativi degli utenti della rete, parametro p) sono pubbliche.
   Il polinomio $\psi_U(x)$ ha la seguente forma:

$$\begin{aligned} \psi_U(x) &= f(x, ID_U) \\ &= r + s(x + ID_U) + txID_U \\ &= (r + sID_U) + (s + tID_U)x \\ &= a_U + b_U x \bmod p, \end{aligned}$$

   dove si è posto $a_U = r + sID_U$ e $b_U = s + tID_U$.
   Quindi, l'informazione reale che ogni utente **U** riceve dal Server è la coppia (a_U, b_U); il parametro p è invece noto a tutti. Una volta ricevuta questa coppia, l'utente **U** può costruirsi il proprio polinomio $\psi_U(x) = a_U + b_U x \bmod p$ tramite il quale può in ogni momento calcolare le chiavi per comunicare con gli altri utenti. La coppia (a_U, b_U) rappresenta quindi la chiave segreta per l'utente **U**. È dunque necessario che le varie coppie siano comunicate dal Server ai vari utenti su un canale sicuro, dopodiché il Server diventa off-line.

4. Se due utenti **A** e **B** vogliono comunicare tra di loro, allora questi calcolano rispettivamente i valori $\psi_A(ID_B) = f(ID_B, ID_A) \in Z_p$ e $\psi_B(ID_A) = f(ID_A, ID_B) \in Z_p$.
   Poiché il polinomio $f(x, y)$ è simmetrico, risulta che

$$\begin{aligned} \psi_A(ID_B) &= f(ID_B, ID_A) \\ &= f(ID_A, ID_B) \\ &= \psi_B(ID_A). \end{aligned}$$

   Questo valore può quindi essere usato come chiave segreta da **A** e **B** per comunicare tra di loro. In questo modo, entrambi gli utenti, senza scambio reciproco di informazioni e senza chiedere aiuto al Server o ad altri utenti, riescono a calcolare la propria chiave.
   Quindi il protocollo di Blom si basa molto sul fatto che il polinomio $f(x, y)$ sia simmetrico.

Vediamo ora alcuni vantaggi e svantaggi del protocollo.

Abbiamo appena visto che ogni utente **U** può comunicare con qualsiasi altro utente della rete mantenendo solamente il proprio polinomio $\psi_U(x) = a_U + b_U x \bmod p$ e conoscendo gli identificativi degli altri utenti (che sono pubblici).
Dunque, l'informazione segreta che l'utente **U** deve memorizzare per poter utilizzare questo protocollo è rappresentata solo dai due coefficienti a_U e b_U del polinomio $\psi_U(x)$, essendo p un parametro pubblico. Poiché a_U e b_U sono due valori modulo p, ossia due valori aventi la stessa grandezza di una qualunque chiave, allora **U** deve memorizzare il doppio dell'informazione della chiave che riesce a instaurare con un altro utente. Si è così ottenuto dal punto di vista della memoria, un notevole miglioramento rispetto allo schema in cui si ricercava la sicurezza totale, perché in precedenza ogni utente doveva memorizzare un fattore di $n-1$ volte la lunghezza della chiave, mentre ora il fattore è soltanto del doppio.

Il problema di questo schema sta nel fatto che esso è sicuro solo rispetto ad ogni singolo utente, mentre è vulnerabile agli attacchi di coalizioni di utenti.

Osserviamo che questo protocollo può essere attaccato solo da utenti della rete. Infatti un nemico esterno non è in grado di carpire alcun segreto alle parti perché i polinomi sono comunicati dal Server ai vari utenti su un canale sicuro ed inoltre, due parti che vogliono comunicare tra di loro calcolano la loro chiave in base ad informazioni pubbliche ed, ovviamente, al proprio polinomio, senza passarsi alcuna informazione.

Cominciamo col vedere che il protocollo gode di una proprietà di sicurezza forte rispetto agli attacchi di ogni singolo utente della rete, dove per sicurezza forte si intende che questo utente *cattivo* qualunque cosa faccia non riuscirà a sapere **nulla** sulla chiave tra altri due utenti, ossia non riuscirà a calcolarsi neanche un singolo bit di tale chiave.
Sussiste infatti il seguente

Teorema 2.4 *Lo schema di Blom definito sopra con b=2 e c=1 è incondizionatamente sicuro contro gli attacchi portati da un singolo utente.*

Dimostrazione Supponiamo che un utente **C** voglia calcolare la chiave comune a due utenti **A** e **B**.
Le informazioni relative agli utenti **A** e **B** che **C** ha in suo possesso sono le seguenti:

1. ID_A, ID_B, ID_C, p informazioni pubbliche;
2. $\psi_C(x) = a_C + b_C x \bmod p$ comunicatogli dal Server.

Inoltre, dai vari passi del protocollo, **C** sa che il suo polinomio è stato costruito dal Server nel seguente modo:

$$\begin{aligned}\psi_C(x) &= r + s(x + ID_C) + txID_C \bmod p && \text{per la (1) e la (2)}\\ &= r+sx+sID_C+txID_C \bmod p\\ &= \underbrace{(r + sID_C)}_{a_C} + \underbrace{(s + tID_C)}_{b_C} x \bmod p.\end{aligned}$$

Quindi **C** conosce i valori di a_C e b_C e sa che questi coefficienti hanno la seguente forma:

$$a_C = r + sID_C \bmod p, \qquad b_C = s + tID_C \bmod p. \tag{4}$$

Inoltre, dalla (3), **C** sa anche che la chiave $k_{A,B}$ ha la seguente forma:

$$\begin{aligned}k_{A,B} &= \psi_A(ID_B)\\ &= \psi_B(ID_A)\\ &= f(ID_A, ID_B)\\ &= r + s(ID_A + ID_B) + tID_A ID_B \bmod p,\end{aligned}$$

ossia che la chiave $k_{A,B}$ soddisfa la seguente relazione:

$$k_{A,B} = r + s(ID_A + ID_B) + tID_A ID_B \bmod p. \tag{5}$$

L'utente **C** può quindi costruire il seguente sistema:

$$\begin{cases} k_{A,B} &= r + s(ID_A + ID_B) + tID_A ID_B \bmod p\\ a_C &= r + sID_C \bmod p\\ b_C &= s + tID_C \bmod p\end{cases}$$

che scritto in forma matriciale diventa

$$\begin{pmatrix} 1 & ID_A + ID_B & ID_A ID_B \\ 1 & ID_C & 0 \\ 0 & 1 & ID_C \end{pmatrix} \begin{pmatrix} r \\ s \\ t \end{pmatrix} = \begin{pmatrix} k_{A,B} \\ a_C \\ b_C \end{pmatrix}.$$

Le uniche incognite del sistema sono r, s, t e $k_{A,B}$. Se l'utente **C** fissa un valore per $k_{A,B}$ allora il sistema precedente diventa un sistema lineare nelle tre variabili r, s e t. Ricordiamo che una condizione necessaria e sufficiente affinché questo sistema ammetta un'unica soluzione è che la matrice dei coefficienti abbia determinante non nullo.
Nel nostro caso, detta **W** la matrice dei coefficienti, risulta:

$$\begin{aligned} \det W &= ID_C^2 - [(ID_A + ID_B)ID_C - ID_A ID_B] \\ &= ID_C(ID_C - ID_A) - ID_B(ID_C - ID_A) \\ &= (ID_C - ID_A)(ID_C - ID_B) \bmod p. \end{aligned}$$

Poiché abbiamo supposto che gli identificativi siano tutti diversi tra di loro (in particolare $ID_C \neq ID_A$ e $ID_C \neq ID_B$), risulta che $\det W \neq 0$, da cui si deduce che il sistema ammette un'unica soluzione.
Dunque, fissato un valore per la chiave $k_{A,B}$, esiste un' unica (r, s, t) compatibile con le informazioni che ha **C**.
Se **C** riuscisse a stabilire i valori scelti dal Server, allora sarebbe in grado di calcolarsi il polinomio associato ad ogni utente e quindi riuscirebbe a ricavare le chiavi di tutti gli utenti della rete. Risulta chiaro che il problema di fondo sta nel fatto che **C** non conosce la chiave $k_{A,B}$, e con le informazioni in suo possesso può solo fissare un valore per $k_{A,B}$ e ricavare in modo univoco una tripla di valori (r, s, t) compatibile con le sue informazioni.
Poiché per ogni possibile valore di $k_{A,B}$ esiste una tripla (r, s, t) e, viceversa, per ogni tripla esiste una possibile chiave (ciò risulta dal sistema), e giacché il Server ha scelto $r, s, t \in Z_p$ in maniera indipendente e uniforme, allora tutte le triple ricavate da **C** hanno la stessa probabilità di essere state scelte dal Server, e ciò implica che le chiavi che **C** può ricavare, che sono tutte le possibili chiavi, hanno tutte la stessa probabilità di essere la vera chiave $k_{A,B}$, e quindi **C** non sa assolutamente nulla su $k_{A,B}$, neanche su un singolo bit di questa chiave. $\square$

Abbiamo quindi visto che vi è una sicurezza forte per quanto riguarda attacchi di singoli utenti; facciamo ora vedere come una coalizione tra più utenti possa attaccare il protocollo ricavando semplicemente i valori di r, s e t.

Supponiamo che la coalizione di utenti cattivi sia formata dagli utenti **C** e **D**.
Le informazioni in loro possesso sono:

1. ID_C, ID_D, p — informazioni pubbliche;
2. a_C, b_C, a_D, b_D — informazioni ottenute dal Server;

ed inoltre sanno che valgono le seguenti equazioni:

$$\begin{aligned} a_C &= r + s \cdot ID_C \bmod p, \\ b_C &= s + t \cdot ID_C \bmod p, \\ a_D &= r + s \cdot ID_D \bmod p, \\ b_D &= s + t \cdot ID_D \bmod p. \end{aligned}$$

Esistono quindi quattro equazioni lineari in tre incognite e ciò permette di ricavare i valori di r, s e t. Si potrebbe infatti procedere sottraendo la terza relazione dalla prima, ottenendo

$$a_C - a_D = s(ID_C - ID_D) \bmod p,$$

da cui

$$s = (a_C - a_D)(ID_C - ID_D)^{-1} \bmod p.$$

Ricordiamo che essendo p un numero primo, esiste l'inverso di qualunque elemento appartenente a Z_p^*.
Sostituendo il valore di s nella prima e nella seconda relazione si ottiene:

$$\begin{aligned} r &= a_C - ID_C(a_C - a_D)(ID_C - ID_D)^{-1} \bmod p, \\ t &= [b_C - (a_C - a_D)(ID_C - ID_D)^{-1}]ID_C^{-1} \bmod p. \end{aligned}$$

C e **D**, a questo punto, possono ricavare il polinomio $f(x, y)$ e computare le chiavi di tutti gli utenti della rete.

Affinché uno schema che prevede la comunicazione tra coppie di utenti riesca a sopportare coalizioni di utenti maliziosi, è sufficiente che il Server utilizzi polinomi simmetrici di grado maggiore di 1.

Si può dimostrare che affinché lo schema di Blom sia c-sicuro, è sufficiente che il Server calcoli il polinomio $f(x, y)$ nel seguente modo:

$$f(x, y) = \sum_{i=0}^{c} \sum_{j=0}^{c} a_{i,j} x^i y^j \bmod p,$$

con

$$a_{i,j} \in Z_p, \qquad a_{i,j} = a_{j,i} \qquad \textit{per ogni } i, j. \tag{6}$$

La relazione (6) è necessaria affinché il polinomio sia simmetrico.
Questo schema si può ulteriormente generalizzare.

Supponiamo a questo punto che ci siano b utenti $A_1, A_2, \ldots, A_b$ che vogliano comunicare tra loro utilizzando un'unica chiave di conferenza, mantenendo lo schema c-sicuro.
Ciò significa che si vuole uno schema in cui b utenti possono calcolare una chiave che solo loro conoscono e che altri c utenti della rete non riescono a sapere nulla su questa chiave.
L'estensione del metodo di Blom a questo caso, prevede che il Server scelga un polinomio simmetrico in b variabili:

$$f(x_1, x_2, \ldots, x_b) = \sum_{i_1=0}^{c} \sum_{i_2=0}^{c} \cdots \sum_{i_b=0}^{c} a_{i_1,i_2,\ldots,i_b} x_1^{i_1} x_2^{i_2} \cdots x_b^{i_b} \bmod p. \tag{7}$$

Il requisito di simmetria del polinomio (7) viene ottenuto scegliendo i valori dei coefficienti $a_{i_1,i_2,\ldots,i_b}$ in modo tale che soddisfino la seguente proprietà:

$$a_{i_1,i_2,\ldots,i_b} = a_{i_{\sigma(1)},i_{\sigma(2)},\ldots,i_{\sigma(b)}}, \tag{8}$$

per ogni permutazione σ di $\{1, 2, \ldots, b\}$.
Il polinomio associato ad ogni utente sarà quindi un polinomio in $b - 1$ variabili:

$$\psi_{A_i}(x_1, x_2, \ldots, x_{b-1}) = f(x_1, x_2, \ldots, x_{b-1}, ID_{A_i}) \bmod p.$$

Se b utenti $A_1, A_2, \ldots, A_b$ vogliono comunicare tra loro, utilizzeranno la seguente chiave:

$$k_{A_1,A_2,\ldots,A_b} = \psi_{A_i}(ID_{A_1}, ID_{A_2}, \ldots, ID_{A_{i-1}}, ID_{A_{i+1}}, \ldots, ID_{A_b}) \qquad \text{per } i = 1, 2, \ldots, b.$$

Per quanto riguarda la quantità di informazione che ogni utente deve memorizzare, questa dipende dai valori dei parametri b e c.

Teorema 2.5 *Sia $N(c,b)$ il numero di coefficienti distinti che il server deve generare per costruire un polinomio simmetrico per il calcolo di chiavi di conferenza per b utenti in uno schema c-sicuro. Allora*

$$N(c,b) = \binom{c+b}{c}.$$

Dimostrazione Consideriamo l'insieme $\mathcal{S}$ di tutte le sequenze di b numeri che possono assumere valori da 0 a c.
Come già sappiamo la cardinalità di tale insieme è $(c+1)^b$.
Sia $\approx$ una relazione binaria su $\mathcal{S}$ definita nel seguente modo: $s_1, s_2 \in \mathcal{S}$,

$$s_1 \approx s_2 \iff \text{esiste una permutazione } \sigma \text{ tale che } \sigma(s_1) = s_2,$$

in parole, s_2 può essere ottenuta permutando gli elementi di s_1.
Come si può facilmente verificare, la relazione sopra definita è una relazione di equivalenza e, come tale, partiziona l'insieme $\mathcal{S}$ in classi disgiunte.
Vi è una corrispondenza biunivoca tra le sequenze di $\mathcal{S}$ e i coefficienti del polinomio (7); più precisamente, ogni sequenza rappresenta l'indice di un coefficiente. L'idea è quella di ragionare sugli indici dei coefficienti piuttosto che sui loro effettivi valori. Dunque, contare il numero dei coefficienti distinti del polinomio (7) si traduce nel calcolare il numero delle classi di equivalenza in cui viene partizionato $\mathcal{S}$ dalla relazione $\approx$.
Notiamo infatti che ogni classe di equivalenza non è altro che l'insieme degli indici dei coefficienti che sono uguali per la proprietà di simmetria del polinomio (7): tali coefficienti infatti dovranno dare un contributo unitario al conteggio che dobbiamo effettuare.
Per stabilire quante sono queste classi cominciamo con l'imporre che ogni classe sia rappresentata dalla sequenza che, intesa come valore numerico, rappresenta l'elemento più piccolo della classe stessa. Chiameremo questa seguenza il *minimo rappresentante* di una classe. Possiamo perciò identificare ogni classe con il suo minimo rappresentante, ottenendo così un nuovo insieme di sequenze che denotiamo con $\mathcal{S}'$.
Le cifre che compongono le sequenze di $\mathcal{S}'$ sono tali da presentarsi, da sinistra verso destra, in ordine crescente; in tal senso $\mathcal{S}'$ è un insieme di sequenze ordinate.
Ad esempio la sequenza <0,0,1,2,0,0> non è ordinata in quanto scorrendola da sinistra verso destra si incontra uno 0 dopo un 2. Possiamo osservare che la sequenza in questione non è un minimo rappresentante; infatti la classe di cui fa parte, è rappresentata dalla sequenza <0,0,0,0,1,2>, che è una sua permutazione.
Il problema di partenza può dunque essere riformulato nel seguente modo: stabilire il numero di sequenze ordinate di lunghezza b composte da cifre che assumono valori compresi tra 0 e c.
Per contare tali sequenze consideriamole in cascata, ordinate in base al valore numerico che rappresentano. Per esempio, se $b = 3$ e c qualsiasi, le possibili sequenze sono quelle riportate in Figura 1*(a)*.

Queste sequenze possono essere viste come suddivise in gruppi, dove ogni gruppo contiene le sequenze che iniziano con la stessa cifra. La Figura 1*(b)* mostra quali sono questi gruppi se

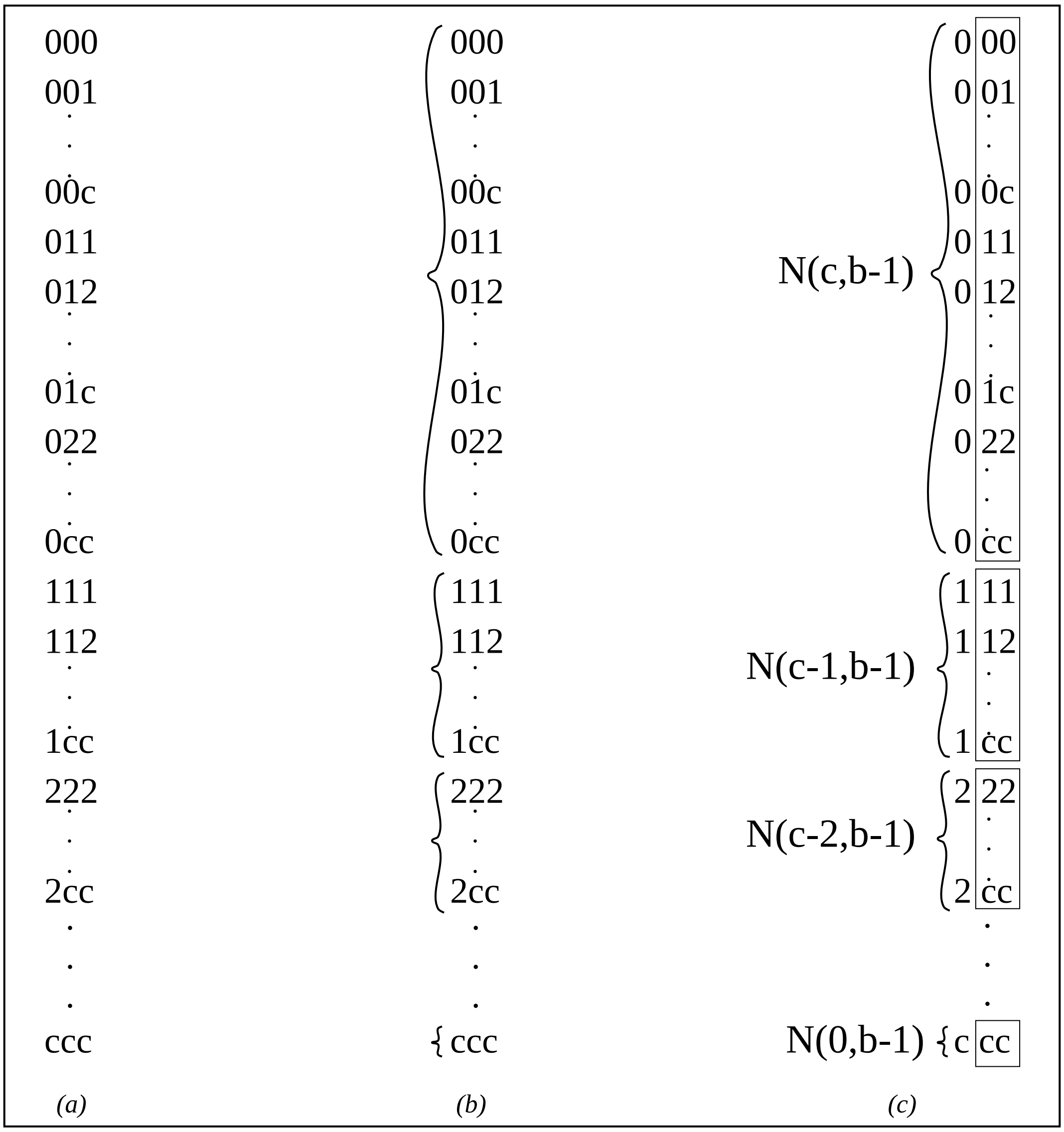

Figura 1: Esempio di sequenze ordinate quando b=3 e c qualsiasi

ci si riferisce all'esempio di Figura 1*(a)*.
L'idea è quella di contare le sequenze di ogni gruppo e poi sommare gli addendi così ottenuti. Se ci riferiamo alla Figura 1*(c)*, possiamo osservare che la cardinalità del gruppo contenente sequenze che iniziano con i ($0 \leq i \leq c$) è esattamente $N(c-i, b-1)$.
Il valore $N(c,b)$ cercato può quindi essere espresso dalla seguente relazione di ricorrenza:

$$N(c,b) = \begin{cases} c+1 & se \;\; b=1,\; c \geq 0 \qquad (9) \\ \sum_{i=0}^{c} N(c-i, b-1) & se \;\; b \geq 2,\; c \geq 0 \qquad (10) \end{cases}$$

Dimostriamo per induzione su b che la soluzione di questa ricorrenza è proprio

$$N(c,b) = \binom{c+b}{c}.$$

Faremo uso della seguente formula

$$\binom{n-1}{k-1} + \binom{n-1}{k} = \binom{n}{k}. \qquad (11)$$

BASE Sia $b = 1$ e $c \geq 0$. Allora

$$\begin{aligned} N(c,1) &= c+1 && \text{per la (9)} \\ &= \binom{c+1}{1} \\ &= \binom{c+1}{c} \\ &= \binom{c+b}{c}. \end{aligned}$$

La base è vera.

IPOTESI Supponiamo per ipotesi induttiva che la tesi sia vera per ogni $1 \leq t \leq b-1$, cioè

$$\textit{per ogni } c \geq 0 \qquad N(c,t) = \binom{c+t}{c}, \qquad \text{con } 1 \leq t \leq b-1.$$

INDUZIONE Proviamo la verità della tesi per t=b.

$$\begin{aligned} N(c,b) &= \sum_{i=0}^{c} N(c-i, b-1) && \text{per la (10)} \\ &= \sum_{i=0}^{c} N(i, b-1) \\ &= \sum_{i=0}^{c} \binom{i+b-1}{i} && \text{per l'ipotesi induttiva} \\ &= \binom{b-1}{0} + \binom{b}{1} + \binom{b+1}{2} + \binom{b+2}{3} + \cdots + \binom{b+c-1}{c} \\ &= \binom{b}{0} + \binom{b}{1} + \binom{b+1}{2} + \binom{b+2}{3} + \cdots + \binom{b+c-1}{c} && \left[\binom{b-1}{0} = 1 = \binom{b}{0}\right] \\ &= \binom{b+1}{1} + \binom{b+1}{2} + \binom{b+2}{3} + \cdots + \binom{b+c-1}{c} && \text{per la (11)} \\ &= \binom{b+2}{2} + \binom{b+2}{3} + \cdots + \binom{b+c-1}{c} && \text{per la (11)} \\ &= \binom{b+3}{3} + \cdots + \binom{b+c-1}{c} && \text{per la (11)} \\ &\vdots \\ &= \binom{b+c-1}{c-1} + \binom{b+c-1}{c} && \text{per la (11)} \\ &= \binom{b+c-1+1}{c} && \text{per la (11)} \\ &= \binom{b+c}{c}. && \text{per la (11)} \end{aligned}$$

Pertanto la tesi è vera per il principio d'induzione.

$\square$

Osservazione 1 Il valore $N(c, b)$ è uguale al numero di possibili modi per estrarre con ripetizione b oggetti (gli indici $i_1, i_2, \ldots, i_b$) tra $c+1$ possibili valori (ogni indice varia da 0 a c). ▯

Osservazione 2 È possibile esprimere $N(c, b)$ tramite un'altra relazione di ricorrenza in cui non compare la sommatoria:

$$N(c,b) = \begin{cases} 1 & se \;\; b \geq 1, c = 0 \\ c+1 & se \;\; b = 1, c \geq 1 \\ N(c-1,b) + N(c,b-1) & se \;\; b \geq 2, c \geq 1 \end{cases}$$

Tale relazione discende direttamente dalla (11). ▯

Ogni utente, ricevendo dal server un polinomio in $b-1$ variabili di grado c, deve memorizzare $\binom{c+b-1}{c-1}$ coefficienti. Questo coefficiente binomiale diventa molto grosso appena i termini b e c diventano grandi, per cui lo schema di Blom è consigliabile adottarlo solo per valori abbastanza piccoli di b e di c; in genere tale schema si usa quando $b = 2$ e c è abbastanza piccolo.

Analizziamo infine il problema dell'inserimento di un nuovo utente nella rete. Una volta generato l'identificativo ID_U del nuovo utente U, identificativo diverso da tutti gli altri, il Server deve calcolare il polinomio $\psi_U(x)$ associato all'utente e comunicarglielo tramite un canale sicuro. Come si vede, in questo caso, l'aggiunta di un nuovo utente non genera nessun cambiamento sostanziale nella rete. Infatti ogni vecchio utente, non ha necessità di cambiare il proprio polinomio, se non nel caso in cui aggiungendo il nuovo utente, n supera il valore di p, condizione questa che si vuole evitare perché, come detto precedentemente, ciò provocherebbe l'esistenza di almeno due identificativi uguali, con corrispondente perdita di sicurezza dei dati. In questo caso, il server dovrebbe rientrare in funzione e ridistribuire i polinomi chiave a tutti gli utenti, con il valore di p modificato.
Questo inconveniente è comunque facilmente superabile scegliendo opportunamente il valore di p abbastanza grande, in maniera tale che ben difficilmente nella rete si possa avere un numero superiore di utenti. Ad esempio, p potrebbe essere un valore di 60 bit, perché non è possibile, dal punto di vista pratico, che una rete raggiunga una dimensione tale da dover gestire 2^{60} utenti.

2.2 Distribuzione di chiavi con Server On-Line

L'approccio visto precedentemente è un approccio statico, nel senso che il Server distribuisce le chiavi e non partecipa più alla gestione delle stesse. Quello che succede quindi è che se un nemico viene a conoscenza della chiave comune a due utenti, questi non potranno più comunicare in modo sicuro, fino a quando non verranno ridistribuite le chiavi.

Le tecniche di Gestione, che analizzeremo in questa parte, prevedono invece un Server *on-line* che, ad ogni nuova richiesta di comunicazione, fornisce una nuova chiave di sessione alle due parti (questa proprietà è chiamata *key freshness*). Dunque, dopo aver utilizzato questa chiave, le due parti possono dimenticarla del tutto.
Ogni utente, quindi, deve mantenere solo la chiave segreta che gli permette di comunicare in modo sicuro col Server, visto che per ogni comunicazione con gli altri utenti, la chiave utilizzata è fornita dal Server stesso. Il Server, a sua volta, deve mantenere un database in cui conservare le chiavi che gli permettono di comunicare con tutti gli utenti.

Ovviamente la chiave che ogni utente condivide col Server deve averla ricevuta attraverso un canale sicuro, mentre la distribuzione della chiave di sessione avviene sulla rete, quindi su un canale non sicuro.

Nello schema generale che mostreremo, ogni utente **U** è mappato nella rete con un identificativo unico ID_U. Indicheremo di seguito con $e_k(x)$ la funzione di cifratura con chiave privata k, e con $d_k(x)$ la corrispondente funzione di decifratura. Per quel che riguarda le chiavi, utilizzeremo la seguente notazione:

- k indica la Chiave di Sessione
- k_A indica la Chiave comune a **S** ed **A**
- k_B indica la Chiave comune a **S** e **B**

Lo schema generale di comunicazione tra due utenti **A** e **B** è il seguente:

Schema Generale

1. **A** manda al server **S** la coppia (ID_A, ID_B)
2. **S** manda ad **A**
$$m_1 = e_{k_A}(k), \qquad m_2 = e_{k_B}(k),$$
dove k è una chiave scelta a caso nello spazio delle chiavi **K**.
3. **A** decodifica m_1, ottenendo $k = d_{k_A}(m_1)$ e manda m_2 a **B**
4. **B** decodifica m_2 ottenendo così la chiave $k = d_{k_B}(m_2)$

Anche in questo caso, le schiavi comuni tra il Server e gli utenti devono essere comunicate su un canale sicuro, mentre la distribuzione della chiave di sessione avviene sulla rete, quindi su un canale non sicuro. Chiunque assiste a questa comunicazione non riesce a sapere nulla sulle informazioni che il Server **S** e i due utenti **A** e **B** si trasmettono; infatti, i messaggi che viaggiano sulla rete sono crittografati con le chiavi segrete di **A** e **B**, e quindi solo questi riescono a decifrarli. L'unica informazione che il nemico riesce a reperire è la richiesta iniziale di **A** che vuole comunicare con **B**, in quanto può intercettare la coppia (ID_A, ID_B) che **A** spedisce al Server. Tale intercettazione non gli fornisce però alcuna informazione utile in quanto gli identificativi degli utenti sono comunque dei valori pubblici e quindi noti a tutti. Dopo aver seguito questo schema, i due utenti **A** e **B** potranno utilizzare la chiave k per comunicare tra di loro.

Uno dei vantaggi principali di questa tecnica è il ricambio continuo delle chiavi. Infatti se la chiave utilizzata per la comunicazione tra due utenti, è utilizzata più volte, la probabilità che questa venga scoperta da un nemico cresce. Con questa tecnica di comunicazione, questa probabilità diventa estremamente bassa in quanto la chiave di sessione viene utilizzata solo per un'unica sessione, quindi il suddetto problema risulta inesistente.

2.2.1 Needham-Schroeder

Una prima modifica allo schema generale la possiamo vedere con lo schema di Needham-Schroeder.
Come si può osservare, in questo protocollo compare un controllo, prima dell'utilizzo della chiave k, che tende a verificare se le parti hanno ricevuto correttamente il valore di k.

Quello che si può dire è che questo protocollo può essere soggetto agli attacchi di *Replay*. Questo tipo di attacco viene portato da utenti della rete stessa. L'ipotesi è che l'utente **A** abbia diritto ad un servizio da **B** per un periodo di tempo limitato. Ottiene quindi legalmente la chiave k da **S**.
Nel momento in cui **A** perde il diritto ad accedere al servizio, questi può cercare di riutilizzare la chiave k per comunicare con **B**. Questi infatti non riceve nessuna informazione dal Server **S**. Tutte le informazioni necessarie alla comunicazione vengono trasmesse da **A**.

Per questo motivo, questo protocollo non è utilizzato.

Schema di Needham-Schroeder

1. **A** genera un messaggio casuale R_A e manda al Server **S** la tripla (A, B, R_A)
2. **S** genera a caso una chiave di sessione k e manda ad **A**
$$m_1 = e_{k_A}(R_A, B, k, e_{k_B}(k, A)).$$
3. **A** decodifica m_1 e manda $e_{k_B}(k, A)$ a **B**
4. **B** decodifica il messaggio, genera un messaggio casuale R_B e manda ad **A**
$$m_2 = e_k(R_B).$$
5. **A** decodifica il messaggio m_2 e calcola il valore $R_B - 1$ spedendo a **B** $m_3 = e_k(R_B - 1)$.
6. **B** decodifica il messaggio ricevuto e controlla che
$$R_B - 1 = d_k(m_3).$$

2.2.2 Kerberos

Cos'è Kerberos[1].
Nel 1983, in collaborazione con la IBM e con la Digital Equipment Corporation, il MIT (Massachusetts Institute of Technology) varò il progetto ATHENA, che doveva fornire risorse per il calcolo avanzato a studenti e a docenti.

Il progetto Athena non è un sistema operativo per la gestione della rete del MIT (MITnet), ma si pone l'obiettivo di affiancarlo per la gestione del traffico sulla rete. Tuttavia, l'efficienza della MITnet deriva dal successo del progetto Athena.

MITnet si estende su tutto il coprensorio del campus su circa due miglia quadrate, e su esso circolano 100 Mb/s. Il modello di rete è quello client/server ma per una rete così grande è stato necessario un modello spine/gateway, preso poi come modello dalla comunità Internet.

Il modello spine/gateway può essere visto come un agglomerato di subnet (sottoreti) con una propria struttura indipendente ma che comunicano tra loro grazie ai gateway (nodi di comunicazione) per mezzo di indirizzi elettronici.

Quando è stato progettato Athena, si era preso in considerazione che la struttura a rete rappresentava una seria minaccia alla sicurezza, e senz'altro gli studenti avrebbero subito scoperto come programmare le stazioni di lavoro dell'università in modo da farne efficienti dispositivi per il furto di passwords. Per ridurre il rischio fu progettato Kerberos, un sistema di autenticazione che evita all'utente di inviare continuamente parole d'ordine leggibili su collegamenti di rete vulnerabili. Questo omonimo del formidabile cane a tre teste custode degli inferi è il cuore della sicurezza di ATHENA. A differenza del suo mitico predecessore, tuttavia, il nostro guardiano cibernetico non è ancora venuto meno al suo compito di sorvegliante.

Per quanto riguarda il problema della distribuzione delle chiavi, Kerberos utilizza l'approccio con chiave simmetrica.
Poiché ogni protocollo per la distribuzione simmetrica delle chiavi ha alla base un sistema di cifratura a chiave segreta ci aspettiamo che anche Kerberos abbia tutto ciò.

Inoltre, essendo Kerberos un protocollo per l'autenticazione su rete, esso ha al suo interno una funzione per firmare le comunicazioni: in particolare tale funzione è realizzata usando l'algoritmo di cifratura **DES** in congiunzione con il metodo di checksum.

Il centro di distribuzione delle chiavi per ogni dominio condivide una chiave segreta con ogni nodo nella rete. La prova della conoscenza di questa chiave segreta è usata per verificare l'autenticità del nodo.

Da quanto detto si evince che Kerberos è un protocollo con distribuzione decentralizzata, in quanto l'autenticazione del Server e del client è ottenuta dalla collaborazione di due nodi per ogni dominio della rete.
Per distribuzione decentralizzata si intende che la distribuzione delle chiavi nella rete è controllata da più computer della rete e non da un singolo computer (distribuzione centralizzata).

Il protocollo Kerberos ha la peculiarità di derivare le chiavi degli utenti dalle loro passwords; questo ha il vantaggio di trasparenza in quanto gli utenti non devono continuamente digitare la loro chiave segreta. Allo stesso tempo può essere un punto debole del sistema per cui è desiderabile che la funzione che trasforma la password in chiave sia non invertibile, e la sua

[1]Kerberos (o Kerboros, come riportato da Stinson), in italiano Cerbero, è una figura mitologica greca. Dall'unione di Echidna e Tifone nacque ≪il prodigioso indicibile Cerbero, carnivoro dell'Ade, cane dalla bronzea voce, dalle cinquanta teste, impudente e vigoroso≫. In genere però gli vengono attribuite soltanto tre teste. A Cerbero era commesso l'incarico di vietare ai morti di oltrepassare i confini dell'eterna notte, e d'impedire ai vivi di penetrare nel suo regno tenebroso: solo Ercole, Orfeo ed Enea ardirono di affrontarlo. (cfr. A. Morelli, DEI E MITI Enciclopedia di mitologia universale, ed. EDIZIONI LIBRARIE ITALIANE)

rappresentazione deve essere diversa in ogni dominio. Questo è importante per gli utenti che sono registrati in più domini, perché spesso usano la stessa password per ogni dominio e quindi se è compromesso il server Kerberos di un dominio questo non compromette gli altri domini in cui il client è registrato. La proprietà per un utente di non avere la stessa chiave su domini diversi, anche se ha la stessa password, è garantita dal fatto che l'algoritmo di derivazione prende in input la password dell'utente insieme a delle informazioni contenute nei file **krb.conf** e **kerb.realms**, dove risiedono le informazioni di configurazione del dominio. Essi fisicamente stanno nella directory **/krb5** alla quale possono accedere solo i super utenti.

Riportiamo, nella pagina seguente, l'algoritmo che costruisce una chiave DES di 8 byte partendo dalla password dell'utente e, di seguito, lo pseudocodice che lo realizza.

```
pwd_key_proc(string,realm,name)
{
 odd=1;
 s=string+realm;
 for(each component in name){
   s=s+component;
  }
 tempkey=NULL;
 pad(s); /* with nulls to 8 byte boundary */
 for(8byteblock in s){
   if(odd==0){
     odd=1;
     reverse(8byteblock);
    }
   else odd=0;
   tempkey=tempkey xor 8byteblock;
  }
 fixparity(tempkey);
 key=DES_CBC_check(s,tempkey);
 fixparity(key);
 return key;
}
```

I tre parametri della procedura contengono le seguenti informazioni:

- string: la pasword dell'utente;
- realm: informazioni contenute nel file kerb.realms;
- name: un record i cui campi contengono informazioni contenute nel file krb.conf.

È interessante notare che, quando la funzione di trasformazione da password a chiave (pwd_key_proc) deriva la chiave utente, non la copia in nessun file ne tanto meno in zone di memoria RAM condivisa. La chiave si trova esclusivamente nella zona di memoria locale dell'utente (PCB) e quando il processo muore tale chiave viene distrutta insieme al PCB.

Allo stesso tempo il protocollo ha la peculiarità di non far viaggiare passwords in forma chiara sulla rete.

Trasformazione da password a chiave DES

1. Concateniamo la password dell'utente con le informazioni relative allo stesso utente contenute nei file **krb.conf** e **kerb.realms** e con una sequenza di zeri tale da rendere la stringa risultante divisibile in blocchi di 8 byte ciascuno. La stringa s così ottenuta sarà quindi composta da un certo numero n di blocchi:

$$s = B_1 B_2 ... B_n,$$

ogni blocco composto di 8 byte:

$$B_i = b_0^{(i)} b_1^{(i)} ... b_7^{(i)}, \qquad i = 1, 2, ..., n.$$

2. Elaboriamo ogni blocco con la seguente regola:

   - i blocchi in posizione dispari $(B_1, B_3, ...)$ restano inalterati;
   - i blocchi in posizione pari $(B_2, B_4, ...)$ vengono modificati invertendo l'ordine degli 8 byte di cui sono composti; per esempio il generico blocco in posizione pari

   $$B_{2k} = b_0^{(2k)} b_1^{(2k)} ... b_7^{(2k)}$$

   viene così modificato

   $$B'_{2k} = b_7^{(2k)} b_6^{(2k)} ... b_0^{(2k)}.$$

3. Calcoliamo lo XOR bit a bit dei blocchi $B_1, B'_2, B_3, B'_4, ...$

$$\begin{array}{c} b_0^{(1)} b_1^{(1)} ... b_7^{(1)} \oplus \\ b_7^{(2)} b_6^{(2)} ... b_0^{(2)} \oplus \\ b_0^{(3)} b_1^{(3)} ... b_7^{(3)} \oplus \\ b_7^{(4)} b_6^{(4)} ... b_0^{(4)} \oplus \\ \vdots \end{array}$$

ottenendo una stringa di 64 bit che chiamiamo *tempkey*.

4. Aggiustiamo il bit di parità di ciascuno degli 8 byte di *tempkey*.

5. Calcoliamo il checksum di 64 bit della stringa s utilizzando il DES in modalità CBC con la chiave *tempkey*:

$$check = \text{DES\_CBC\_checksum}(s, tempkey).$$

6. Aggiustiamo nuovamente il bit di parità di ciascuno degli 8 byte di *check*. Questa è la nostra chiave DES.

In ogni caso, però, nel passaggio tra il terminale dell'utente e la macchina client, la password è in forma chiara. Per terminali più potenti come gli Xterm che hanno una capacità elaborativa propria, non sfruttabile dall'utente ma solo dal costruttore dell'hardware, c'è la possibilità di eseguire questa funzione dal terminale stesso. In pratica la funzione di login tradizionale viene sostituita dalla klogin (k sta per Kerberos) che è fatta in modo da interfacciare le procedure incapsulate nel terminale.

È importante notare che ogni messaggio che viaggia tra host di un dominio o su domini diversi ha nel suo interno una struttura chiamata **EncryptedData**, costituita da:

1. etype (tipo di cifratura)
2. kvno (il numero di versione della chiave sotto cui i dati sono cifrati)
3. cipher (testo cifrato)

Per ottimizzare le misure di sicurezza viene aggiunto un checksum e un confonditore al testo che dobbiamo cifrare. In pratica, di ogni messaggio si fa un *digest* che accompagna il messaggio stesso. Ovviamente l'algoritmo che effettua questa funzione deve garantire che due messaggi differenti non producano lo stesso digest; questa proprietà è detta collision-free.

Il protocollo Kerberos ha scelto come algoritmi di checksum quelli di Ron Rivest, RSA-MD4 e RSA-MD5, i quali si suppongono collision-free. Per migliorare ulteriormente la sicurezza, il risultato del checksum viene cifrato in DES con una variante della chiave di sessione. Questa variante è ottenuta facendo l'operazione di **XOR** tra la chiave di sessione e la costante $F0F0F0F0F0F0F0F0$; successivamente si esegue la cifratura normalmente.

Situazione attuale del sistema Kerberos al MIT

Attualmente al MIT sono disponibili due versioni del sistema Kerberos: la versione 4 e la versione 5 (succintamente descritta sopra).

La versione 4 è nella sua forma finale: il MIT infatti non rilascerà ulteriori release di tale versione. L'ultima release risale al 10 dicembre 1992. Per quanto riguarda la versione 5, la release più aggiornata è la *beta 5*, rilasciata il 5 maggio 1995. Essa contiene sostanziali cambiamenti, un nuovo "administration server" e una portabilità migliorata.

Le due versioni non sono compatibili. Esse infatti sono basate su protocolli completamente differenti. Comunque la distribuzione Kerberos V5 contiene alcune parti di codice che permettono una minima compatibilità con la versione precedente. Infatti:

a) il server Kerberos può opzionalmente servire richieste fatte seguendo la versione 4;

b) esiste un programma per convertire un database dal formato della versione 4 al formato della versione 5;

c) è fornito un "administration server" che accetta richeste fatte secondo la versione 4 ma opera su un database della versione 5.

Distribuzione di chiavi secondo Kerberos

In questo paragrafo analizziamo il modo in cui nel sistema Kerberos viene gestita la distribuzione delle chiavi agli utenti della rete.
Il protocollo Kerberos è descritto nel seguente diagramma:

Protocollo Kerberos

1. **A** richiede a **S** una *Chiave di Sessione* per comunicare con **B**.

2. **S** sceglie a caso una chiave k tra tutte le possibili chiavi, definisce il periodo di validità L e un valore T, detto *Timestamp*, che indica l'inizio del periodo di validità della chiave stessa.

3. **S** calcola
$$m_1 = e_{k_A}(k, ID_B, T, L), \qquad m_2 = e_{k_B}(k, ID_A, T, L)$$
e manda m_1 e m_2 ad **A**.

4. **A** utilizza la sua funzione di decodifica d_{k_A} per calcolare k, T, L da m_1, quindi computa
$$m_3 = e_k(ID_A, T)$$
e manda m_2 ed m_3 a **B**.

5. **B** utilizza la funzione di decodifica d_{k_B} per decodificare m_2, calcolando k, T, L, ID_A, ed utilizza la funzione d_k per decodificare m_3 ricavando il valore di T ed ID_A. Controlla che i due valori di T ed ID_A siano coincidenti. In caso positivo calcola:
$$m_4 = e_k(T + 1)$$
e lo manda ad **A**.

6. **A** decodifica m_4 verificando che $T + 1 = d_k(m_4)$.

La novità sostanziale di questo protocollo, rispetto allo schema generale visto precedentemente, è rappresentata dal fatto che il Server ad ogni richiesta genera una chiave di sessione e ne fissa anche la validità fornendo, oltre la chiave stessa, un *timestamp* T che indica l'inizio del periodo di validità della chiave, e un valore L che indica la durata di questo periodo. Trascorso questo periodo di validità, la comunicazione tra le due parti viene fatta cadere e, per poter continuare a dialogare, le due parti devono richiedere al Server che venga aperta loro una nuova sessione.

Nel passo 1 **A** richiede a **S** una chiave di sessione per comunicare con **B**. Questa fase è l'unica che può essere fatta in chiaro. Infatti, anche se un eventuale nemico venisse a conoscenza delle informazioni trasmesse in questo passo, non potrebbe ricavare informazioni sufficienti per poter decodificare i messaggi che **A** e **B** si scambiano. Questo sempre nell'ipotesi che le chiavi k_A e k_B siano chiavi sicure.

Come si può notare, il Server **S** codifica tra le informazioni che passa alle due parti, due parametri L e T; il loro valore viene fissato dal Server in base al tipo di comunicazione e alla richiesta che viene fatta da **A**. L'aggiunta dei campi L e T ad m_2 dà una validità temporale alla chiave k, validità che **A** non può modificare non conoscendo k_B.

I messaggi m_3 ed m_4 vengono utilizzati come ulteriore test, tramite il quale **A** e **B** si

assicurano che la comunicazione avverrà tra loro due evitando così che una terza persona provi ad immedesimarsi in uno di loro.
Alla fine di questo protocollo **A** e **B** conoscono la chiave k e possono iniziare a dialogare.

Il modo in cui il protocollo Kerberos gestisce le chiavi presenta vantaggi e svantaggi rispetto ad un sistema a chiave pubblica: il vantaggio è che non utilizza protocolli elaborati che si basano su assunzioni computazionali quali possono essere la difficoltà del calcolo del logaritmo discreto, della fattorizzazione, o di altro ancora.
Se tutte le trasmissioni sulla rete sono implementate tramite DES, nel protocollo non compaiono delle chiavi pubbliche e la sicurezza del protocollo stesso si basa interamente sulla sicurezza del sistema DES.

Questo protocollo, però, può essere adottato solo se si ha a disposizione un Server centrale on-line.
Nella prossima evoluzione del Kerberos è previsto l'utilizzo di un sistema a chiave pubblica per risolvere il problema della gestione delle chiavi.

3 Accordo di Chiavi

Come abbiamo visto nei precedenti paragrafi, le tecniche di Distribuzione prevedono che il Server conosca le chiavi utilizzate dagli utenti per codificare i propri messaggi. L'ipotesi forte è che il Server sia "fidato" per tutti gli utenti della rete. In generale però se un nemico riesce a corrompere il Server o, in qualche modo, riesce ad ottenere informazioni riservate, la sicurezza della rete può essere parzialmente (Distribuzione di chiavi con Server on-line) o totalmente (Schema di Blom) compromessa.

Questo tipo di approccio, invece, non prevede la presenza di un supervisore per la gestione delle chiavi, ma quando una coppia di utenti vuole comunicare in modo sicuro, si accorda su una chiave comune e la utilizza, ad esempio come chiave DES o di un qualsiasi altro cifrario a blocchi, per codificare le informazioni e trasmetterle sul canale pubblico.

Il problema di fondo sta nel fatto che, mentre nel caso di Distribuzione ogni utente aveva a sua disposizione delle informazioni che gli permettevano di comunicare in modo sicuro col Server o direttamente con il partner, nel caso di Accordo si parte dall'ipotesi che non vi siano informazioni comuni alle due parti e che tutti i messaggi vengono trasmessi attraverso il canale pubblico, per cui un eventuale nemico può intercettare le informazioni che le due parti si passano per accordarsi sulla chiave, e calcolarla egli stesso.
Per evitare questo si può procedere in vari modi; di seguito ne proponiamo qualcuno.

3.1 Accordo tramite protocollo a chiave pubblica

Un primo schema è semplicemente quello in cui gli utenti **A** e **B** utilizzano un qualsiasi protocollo a chiave pubblica.
Questo accordo avviene nel seguente modo:

Accordo tramite protocollo a chiave pubblica

1. **A** genera una sua chiave pubblica e privata, rispettivamente Pub_A e $Priv_A$, e spedisce a **B** la chiave pubblica.

2. **B** riceve Pub_A, e sceglie a caso una chiave k fra tutte le possibili chiavi.

3. **B** calcola $m = e_{Pub_A}(k)$ e spedisce m ad **A**.

4. **A** utilizza la funzione di decodifica d_{Priv_A} per decodificare m e viene a conoscenza della chiave k, che può essere utilizzata dalle due parti per comunicare tra di loro.

Se un nemico intercetta tutta la comunicazione, vede la chiave pubblica di **A**, vede la cifratura di k tramite la chiave pubblica di **A** ma non riesce a ricavarsi k perché solo **A** può decifrare m tramite la sua chiave privata $Priv_A$. La sicurezza di questo schema è legata alla sicurezza del protocollo a chiave pubblica che viene utilizzato per generare la chiave pubblica e privata di **A**.

Lo schema risulta però lento e poco praticabile proprio perché ogni qual volta un utente vuole comunicare con un altro, è costretto a generare una sua chiave pubblica e privata.

3.2 Schema di Diffie-Hellman

3.2.1 Il Protocollo

Questo schema è stato realizzato nel 1976, addirittura un paio di anni prima di RSA, e sue versioni vengono concretamente utilizzate ai nostri giorni.

Lo schema prevede che i due utenti **A** e **B** si accordino su due valori, p e g: il parametro p è un numero primo che indica la base dell'aritmetica modulare utilizzata nel calcolo delle chiavi; il parametro g è uno dei generatori di Z_p^*. La scelta di p primo ci assicura che esiste almeno un elemento g che è radice primitiva di Z_p^*. Questi valori potrebbero essere gli stessi per tutti gli utenti della rete, quindi noti a tutti, oppure potrebbero essere generati da uno dei due utenti **A** o **B**, e comunicati all'altro.
Vediamo come è strutturato il protocollo:

Protocollo di Diffie-Hellman

1. Le parti si accordano sui valori di p e g.
2. **A** sceglie a caso $x_A \in Z_p$.
3. **B** sceglie a caso $x_B \in Z_p$.
4. **A** calcola
$$y_A = g^{x_A} \bmod p$$
e manda y_A a **B**.
5. **B** calcola
$$y_B = g^{x_B} \bmod p$$
e manda y_B ad **A**.
6. **A** e **B** calcolano la chiave k nel seguente modo:
$$k = g^{x_A x_B} \bmod p.$$

Mostriamo innanzitutto che entrambe le parti possono calcolare la chiave k partendo dalle informazioni che si sono scambiate.

L'utente **A** conosce p, g e x_A ma non conosce x_B; avendo però ricevuto y_B da **B**, può calcolare k nel seguente modo:

$$\begin{aligned}(y_B)^{x_A} \bmod p &= (g^{x_B})^{x_A} \bmod p\\ &= g^{x_B x_A} \bmod p\\ &= g^{x_A x_B} \bmod p\\ &= k.\end{aligned}$$

B può calcolare k in maniera analoga:

$$\begin{aligned}(y_A)^{x_B} \bmod p &= (g^{x_A})^{x_B} \bmod p\\ &= g^{x_A x_B} \bmod p\\ &= k.\end{aligned}$$

Illustriamo l'algoritmo con un piccolo esempio.

Esempio 3.1 (Accordo con Diffie-Hellman) *Supponiamo che $p = 25307$ e $g = 2$ siano pubblicamente conosciuti (p è primo e g è un generatore di Z_p^*). Supponiamo che* **A** *scelga $x_A = 3578$. Quindi egli calcola*

$$\begin{aligned}y_A &= g^{x_A} \bmod p\\ &= 2^{3578} \bmod 25307\\ &= 6113.\end{aligned}$$

Supponiamo che **B** *scelga* $x_B = 19956$ *e calcola*

$$\begin{aligned} y_B &= g^{x_B} \bmod p \\ &= 2^{19956} \bmod 25307 \\ &= 7984. \end{aligned}$$

Ora **A** *può calcolare la chiave*

$$\begin{aligned} k &= (y_B)^{x_A} \bmod p \\ &= 7984^{3578} \bmod 25307 \\ &= 3694. \end{aligned}$$

B *può calcolare la stessa chiave*

$$\begin{aligned} k &= (y_A)^{x_B} \bmod p \\ &= 6113^{19956} \bmod 25307 \\ &= 3694. \end{aligned}$$

A *e* **B** *possono ora comunicare in maniera sicura, per esempio con il DES.* □

Analizziamo a questo punto la sicurezza di questo schema. La domanda a cui occorre dare una risposta è: può un utente **C** calcolarsi la chiave k se $\mathbf{C} \neq \mathbf{A}, \mathbf{B}$? In altre parole, dati $g^{x_A} \bmod p$ e $g^{x_B} \bmod p$ (ma non x_A o x_B), è possibile calcolare $g^{x_A x_B} \bmod p$?
Questo è noto come il **Problema di Diffie-Hellman**.

Un nemico che cerca di attaccare il protocollo si troverà quindi a dover risolvere un' istanza del problema di Diffie-Hellman, il quale è formalmente definito come segue:

Problema di Diffie-Hellman

Istanza del Problema: $I = (p, g, y_A, y_B)$, dove p è primo, g è un generatore di Z_p^*, $y_A = g^{x_A} \in Z_p^*$ e $y_B = g^{x_B} \in Z_p^*$.

Obiettivo: Trovare l'unico valore $k \in Z_p^*$ tale che

$$k = g^{x_A x_B} \bmod p.$$

Molti ritengono che il problema di Diffie-Hellman sia tanto difficile quanto il problema del logaritmo discreto; tuttavia non esiste alcuna dimostrazione di equivalenza tra questi due problemi.
Come è semplice intuire, se si ha a disposizione un algoritmo efficiente per il calcolo del logaritmo discreto, allora si può risolvere in modo efficiente il problema di Diffie-Hellman. Infatti,

$$\begin{aligned} k &= g^{x_A x_B} \bmod p \\ &= y_B^{x_A} \bmod p \\ &= y_A^{x_B} \bmod p, \end{aligned}$$

dove

$$x_A = \log_g y_A \bmod p \qquad \text{e} \qquad x_B = \log_g y_B \bmod p.$$

Non è stato dimostrato il viceversa. Cioè non è detto che avendo a disposizione un algoritmo efficiente per la soluzione del problema di Diffie-Hellman, si possa risolvere in modo efficiente una istanza del logaritmo discreto.

Il tempo necessario alle parti per il calcolo delle chiavi, equivale alla soluzione del problema dell'esponenziazione modulare che ha complessità $O(\log p)$ operazioni in aritmetica modulo p oppure $O(\log^3 p)$ operazioni su bit. Il tempo necessario al nemico per il calcolo della chiave comune agli utenti **A** e **B** è uguale al tempo necessario per la soluzione di un'istanza del problema di Diffie-Hellman. Attualmente, gli algoritmi più veloci che risolvono questo problema, risolvono prima una istanza del problema del logaritmo discreto, calcolando uno dei due valori x_A o x_B come indicato nella seguente relazione:

$$x_A = \log_g y_A \bmod p, \qquad x_B = \log_g y_B \bmod p.$$

I migliori algoritmi che risolvono questo problema, prendono però un tempo proporzionale a:

$$e^{\sqrt{\log p \log \log p}}.$$

Il protocollo di Diffie-Hellman è più veloce rispetto all'accordo che fa uso di un protocollo a chiave pubblica in quanto non occorre calcolare alcuna chiave pubblica e privata, ma basta semplicemente accordarsi sui valori di p e g, i quali possono essere addirittura gli stessi per tutti gli utenti della rete.

L'esigua quantità di informazioni che gli utenti si scambiano pubblicamente è dovuta al fatto che questo protocollo basa la sua forza sulla relativa semplicità del problema di esponenziazione modulare e sull'intrattabilità del problema inverso ossia del calcolo del Logaritmo Discreto.
In sostanza il protocollo di Diffie-Hellman si basa su assunzioni computazionali quali ritenere che sia difficile risolvere un'istanza del problema di Diffie-Hellman e che sia difficile calcolare il logaritmo discreto.

3.2.2 Equivalenza tra ElGamal e Diffie-Hellman

Da quanto detto nel paragrafo precedente, il problema di Diffie-Hellman non è più difficile del problema del Logaritmo Discreto. Anche se non si riesce a dire quanto il problema di Diffie-Hellman sia difficile, si può relazionare la sua sicurezza a quella di un altro crittosistema, quello di ElGamal.

Teorema 3.2 *Rompere il crittosistema di ElGamal è equivalente a risolvere il problema di Diffie-Hellman.*

Dimostrazione Ricordiamo innanzitutto come lavorano le funzioni di cifratura e decifratura nel crittosistema di ElGamal. La chiave è $\mathcal{K} = (p, \alpha, a, \beta)$, dove $\beta = \alpha^a \bmod p$ (a è segreto e p, α e β sono pubblici).
Per un numero casuale $k \in Z_{p-1}$ segreto

$$e_{\mathcal{K}}(x, k) = (y_1, y_2),$$

dove

$$y_1 = \alpha^k \bmod p \qquad \text{e} \qquad y_2 = x\beta^k \bmod p.$$

Per $y_1,\ y_2 \in Z_p^*$

$$d_{\mathcal{K}}(y_1, y_2) = y_2(y_1^a)^{-1} \bmod p.$$

Supponiamo di avere un algoritmo **A** che risolve il problema di Diffie-Hellman e supponiamo di avere un messaggio (y_1, y_2) codificato con ElGamal. Applicando all'algoritmo **A** gli input p, α, y_1, β otteniamo:

$$\begin{aligned} \mathbf{A}(p, \alpha, y_1, \beta) &= \mathbf{A}(p, \alpha, \alpha^k, \alpha^a) \\ &= \alpha^{ka} \bmod p \\ &= \beta^k \bmod p. \end{aligned}$$

Quindi la decodifica di (y_1, y_2) può essere facilmente calcolata come:

$$x = y_2(\beta^k)^{-1} \bmod p.$$

Viceversa, supponiamo di avere un algoritmo **B** che rompe ElGamal; cioè dando in input a **B** i parametri $p, \alpha, \beta, y_1, y_2$ esso calcola la quantità

$$x = y_2(y_1^{log_\alpha \beta})^{-1} \bmod p.$$

Dando ora in input a **B** i parametri p, α, β e γ del problema di Diffie-Hellman, possiamo osservare che:

$$\begin{aligned} \mathbf{B}(p, \alpha, \beta, \gamma, 1)^{-1} &= 1((\gamma^{log_\alpha \beta})^{-1})^{-1} \bmod p \\ &= \gamma^{log_\alpha \beta} \bmod p. \end{aligned}$$

□

3.2.3 Chiavi di Conferenza

Abbiamo considerato finora solamente il caso di una coppia di utenti che utilizza il protocollo di Diffie-Hellman per accordarsi su una chiave. Estenderemo ora il protocollo ad una terna di utenti. La generalizzazione dello stesso ad una n-pla è immediata.

Supponiamo di avere tre utenti **A**, **B** e **C** che vogliono accordarsi su una chiave comune. Il protocollo di Diffie-Hellman, viene modificato come segue:

Diffie-Hellman per Chiavi di Conferenza

1. Le parti si accordano sui valori di p e g.
2. **A**, **B** e **C** scelgono a caso i valori di x_A, x_B e x_C.
3. **A**, **B** e **C** calcolano rispettivamente
$$y_A = g^{x_A} \bmod p, \qquad y_B = g^{x_B} \bmod p, \qquad y_C = g^{x_C} \bmod p.$$
4. **A** manda y_A a **B**, **B** manda y_B a **C** e **C** manda y_C ad **A**.
5. **A**, **B** e **C** calcolano rispettivamente
$$y'_A = y_C^{x_A} \bmod p, \qquad y'_B = y_A^{x_B} \bmod p, \qquad y'_C = y_B^{x_C} \bmod p.$$
6. **A** manda y'_A a **B**, **B** manda y'_B a **C** e **C** manda y'_C ad **A**.
7. **A**, **B** e **C** calcolano rispettivamente
$$y''_A = (y'_C)^{x_A} \bmod p, \qquad y''_B = (y'_A)^{x_B} \bmod p,$$
$$y''_C = (y'_B)^{x_C} \bmod p.$$
8. **A** manda y''_A a **B**, **B** manda y''_B a **C** e **C** manda y''_C ad **A**.
9. **A**, **B** e **C** calcolano la chiave k come
$$k = g^{x_A x_B x_B} \bmod p.$$

Tenendo conto delle informazioni passate dagli utenti risulta:

$$\begin{aligned} y''_A &= (y'_C)^{x_A} \bmod p \\ &= (y_B^{x_C})^{x_A} \bmod p \\ &= ((g^{x_B})^{x_C})^{x_A} \bmod p \\ &= g^{x_A x_B x_C} \bmod p. \end{aligned}$$

Analogamente si dimostra che

$$y''_B = g^{x_A x_B x_C} \bmod p, \qquad y''_C = g^{x_A x_B x_C} \bmod p.$$

In generale, se n utenti vogliono utilizzare la stessa chiave di conferenza, ci sono $n-1$ round o passaggi di dati tra gli stessi. In questo caso, però non vi è la possibilità che un server mantenga le chiavi come nel caso del protocollo per coppie di utenti. Supponiamo, infatti, che un server mantenga le chiavi calcolate con la relazione al punto 4 del protocollo visto nel paragrafo 3.2.1, e che una terna di utenti **A**, **B** e **C** voglia comunicare utilizzando la stessa chiave di conferenza k.
L'utente **A** conosce il valore di x_A e prende dal server i valori di y_B e y_C. È evidente a questo

punto che per calcolare la chiave k deve risolvere una istanza del problema di Diffie-Hellman e calcolare almeno uno dei valori x_B o x_C.

A si trova quindi nelle stesse condizioni di un nemico che cerca di attaccare il protocollo per coppie di utenti, e ciò non è computazionalmente conveniente.

3.3 Schema di Merkle

Finora gli schemi che abbiamo visto si sono basati su delle assunzioni computazionali. Analizziamo ora uno schema che funziona senza basarsi su problemi che si reputano difficili ma che in realtà non è possibile dimostrare che siano tali.

Lo schema di Merkle consiste nel creare delle condizioni in cui la differenza tra il tempo necessario al partner ed al nemico, per calcolare la chiave, è tale da rendere inutile un attacco da parte di quest'ultimo.
Ad esempio, se **A** e **B** impiegano un minuto per calcolare la chiave, un eventuale nemico dovrà impiegarci un anno.

Questo tipo di approccio al problema di accordo, si basa su una filosofia molto semplice. Lo schema prevede infatti che, se un utente **A** vuole comunicare con un utente **B** in modo sicuro, genera un numero elevato di *puzzle*, ognuno dei quali contiene le informazioni per calcolare una chiave diversa, e li manda a **B** utilizzando la rete, quindi un canale non sicuro.

Abbiamo utilizzato il termine *puzzle* invece di *messaggio*, per indicare che la chiave non è contenuta in chiaro nelle informazioni che **A** manda a **B**. Nel puzzle, infatti, sono contenute solamente alcune informazioni che permettono il calcolo della chiave in un tempo prefissato t.
Indichiamo con e_k, d_k una coppia di funzioni di cifratura/decifratura sicura.
Vediamo in dettaglio il protocollo:

Schema di Merkle

1. L'utente **A** genera un numero elevato m di *puzzle*, contenenti ognuno una chiave distinta $k_1, k_2, \ldots, k_m$, e li manda a **B**.

2. **B** sceglie un *puzzle* a caso e lo risolve, ricavando in tempo t la chiave k in esso contenuta.

3. **B** cifra ID_B utilizzando la chiave k e manda il messaggio cifrato $y = e_k(IB_B)$ ad **A**.

4. **A** stabilisce la chiave k utilizzata da **B**, provando una per volta le chiavi $k_1, k_2, \ldots, k_m$ che aveva generato precedentemente, fino a che non trova la chiave k_i tale che

$$ID_B = d_{k_i}(y).$$

Per focalizzare il concetto di puzzle, consideriamo i seguenti due esempi:

Esempio 3.3 (Puzzle di Merkle) *Questa tecnica di costruzione del puzzle, è stata suggerita da Merkle stesso. L'idea è alquanto semplice e prevede che le chiavi che* **A** *genera, vengano codificate utilizzando chiavi di piccole dimensioni, ad esempio 10-20 bit, in modo tale* *che il tempo necessario a* **B** *per rompere un singolo puzzle non sia troppo lungo e, contemporaneamente, il tempo necessario a* **C** *per rompere mediamente* $m/2$ *puzzle sia abbastanza elevato. Insieme alla chiave vengono codificate anche alcune informazioni necessarie a* **B** *per controllare che la chiave calcolata sia effettivamente quella che* **A** *ha codificato.*

Possiamo considerare ad esempio il caso in cui **A** *e* **B** *utilizzino puzzle di* 96 *bit e funzioni di codifica/decodifica:*

$$e_k : \{0,1\}^{96} \rightarrow \{0,1\}^{96}, \qquad d_k : \{0,1\}^{96} \rightarrow \{0,1\}^{96}.$$

L'utente **A** *genera a caso* m *chiavi* $k_1, k_2, \ldots, k_m$ *di* 64 *bit e* m *chiavi* $h_1, h_2, \ldots, h_m$ *di 20 bit ciascuna, costruisce gli* m *puzzle* $y_i = e_{h_i}(k_i 0^{32})$ *e li manda a* **B**.
L'utente **B**, *dopo aver scelto un puzzle a caso, comincerà a generare tutte le possibili chiavi di* 20 *bit, fino a quando non otterrà un messaggio i cui* 32 *bit finali siano tutti nulli. A questo punto avrà identificato la chiave.*

La complessità di questo puzzle dipende dalla lunghezza delle chiavi h_i*; maggiore è la lunghezza di queste chiavi, più difficile diventa risolvere il singolo puzzle.* □

Esempio 3.4 *Consideriamo il caso in cui gli utenti utilizzano il DES per la codifica delle informazioni. Le* m *chiavi* k_i *che* **A** *genera sono, quindi, di* 56 *bit. Possiamo supporre che i puzzle che* **A** *manda a* **B** *siano strutturati come segue:*

$$y_i = (e_{k_i}(k_i), f_i).$$

dove con f_i *indichiamo i primi* 36 *bit di* k_i.
B *sceglie a caso un puzzle e genera tutte le possibili chiavi* k_i' *con il prefisso indicatogli da* **A** *e verifica, per ogni chiave, se* $k_i' = d_{k_i'}(e_{k_i}(k_i))$*. Nel momento in cui il test avrà esito positivo,* **B** *avrà identificato la chiave* k_i.

La complessità di questo puzzle dipende dal numero di bit di k_i *che vengono fissati; ovviamente, per ridurre la complessità del singolo puzzle occorre fissare un numero maggiore di bit e, viceversa, per aumentare la complessità occorre fissarne un numero inferiore.* □

Analizziamo ora i tempi necessari ad **A** e **B** per accordarsi su una chiave, ed il tempo necessario ad un nemico **C** che vuole scoprire quale sia questa chiave.

Supponiamo che il singolo puzzle prodotto da **A** possa essere risolto in t unità di tempo. Poiché **B** sceglie un puzzle a caso e lo risolve, il tempo che impiega per ottenere la chiave k è uguale al tempo di soluzione di un singolo puzzle, ossia è $\Theta(t)$.
Nel momento in cui **A** riceve il messaggio $y = e_k(ID_B)$ da **B**, **A** stabilisce la chiave scelta da **B** provando una per volta tutte le m chiavi che ha generato precedentemente, impiegando quindi un tempo che è $\Theta(m)$ volte la cifratura.

Un nemico passivo **C** che attacca questo protocollo, dovrebbe calcolare, nel caso peggiore, le chiavi contenute in **tutti** gli m puzzle che **A** manda a **B**. Mediamente, riuscirà a trovare la chiave scelta da **B**, rompendo solo la metà dei puzzle trasmessi e verificando quale sia la chiave effettiva. Il tempo necessario quindi al nemico per calcolare la chiave comune alle due parti sarà $\Theta(mt)$. Ricapitolando, indicando con *T(X)* il tempo necessario all'utente X per calcolare la chiave, risulta che

$$T(A) = \Theta(m), \qquad T(B) = \Theta(t), \qquad T(C) = \Theta(mt).$$

Scegliendo i valori di t ed m in modo tale che $m \simeq t$, e indicando con n il valore assunto da m e t, possiamo scrivere le seguenti relazioni:

$$T(A) = \Theta(n), \qquad T(B) = \Theta(n), \qquad T(C) = \Theta(n^2).$$

Questo schema presenta alcuni punti deboli che lo rendono non pratico da utilizzare.
Il problema è che la differenza computazionale tra n ed n^2 diventa molto grande nel momento in cui il valore di n è molto grande, perché n^2 è un valore polinomiale in n e non superesponenziale in n come succede ad esempio negli algoritmi di fattorizzazione, in cui il tempo per fattorizzare è $e^{\sqrt{\ln n \ln\ln n}}$.
Ciò implica che per ottenere un buon grado di sicurezza, tipo quello considerato nell'esempio, occorre generare un elevato numero di puzzle.

Dobbiamo inoltre considerare il fatto che esiste anche la possibilità che **C** abbia a disposizione risorse computazionali maggiori rispetto a quelle possedute dalle due parti. **C** potrebbe utilizzare infatti macchine parallele o, semplicemente, macchine più potenti, che ridurrebbero di molto il tempo per il calcolo della chiave comune ad **A** e **B**, e la differenza tra n ed n^2 non è tale da poter ovviare a problemi di questo tipo. Per questo motivo lo schema di Merkle non viene concretamente utilizzato.

Riferimenti

[1] R. Blom, *An optimal class of symmetric key generation schemes.* Lecture Notes in Computer Science, **209** (1985), 335-338. (Advances in Cryptology - EUROCRYPT '84.)

[2] W. Diffie and M. E. Hellman, *Multiuser cryptographic techniques.* AFIPS Conference Proceedings, **45** (1976), 109-112.

[3] J. Kohl and C. Neuman, *The Kerboros Network Authentication Service.* Network Working Group Request for Comments: 1510, September 1993.

[4] R. C. Merkle, *Secure communications over insecure channels.* Communications of the ACM, **21** (1978), 294-299.

[5] J. I. Schiller, *Secure distributed computing.* Scientific American, **271**(5) (1994), 72-76.

[6] *[ftp://athena-dist.mit.edu/pub/ATHENA/kerberos/KERBEROS.FAQ].*

[7] *[ftp://athena-dist.mit.edu/pub/ATHENA/kerberos/V5DRAFT3-RFC.PS].*