Sistemi Di Elaborazione: Sicurezza Su Reti **Anno Acc. 1997-98**
PROF. ALFREDO DE SANTIS

PROTOCOLLI SICURI

a cura di Marinella Mazza,Daniela Scala,Rosita Auriemma *29 Ottobre 1998*

Indice

1 Poker Mentale

Supponiamo, che tre giocatori, Alice, Bob e Carol (li chiameremo **A**, **B**, **C**), vogliano giocare a poker su rete. Cerchiamo di costruire un protocollo che possa realizzare questa partita. Un primo approccio è quello di chiamare in causa una quarta persona che faccia da garante del gioco, con il compito di mischiare le carte e distribuirle ai partecipanti con comunicazioni private.

Noi ci interesseremo di un protocollo che non faccia uso del garante e che preservi la caratteristica che nessuno dei giocatori si fidi degli altri. Faremo uso di un sistema di cifratura a chiave pubblica che gode della proprietà commutativa. Diamo un esempio di cifratura che fa uso dell' **RSA**, e che soddisfa la proprietà commutativa, (ricordiamo che nell'**RSA**, la chiave pubblica è rappresentata dalla coppia (e, n), mentre quella privata da (d, n)) :
$E_A(x) = x^{e_A} \mod n$
$E_B(E_A(x)) = (x^{e_A})^{e_B} \mod n$
$D_A(E_B(E_A(x))) = ((x^{e_A})^{e_B})^{d_A} \mod n = x^{e_A e_B d_A} \mod n = x^{e_B} \mod n = E_B(x)$.
Il primo protocollo che fu proposto funziona in questo modo:

1. **A, B, C** costruiscono una coppia di chiavi pubblica-privata.

2. Il gioco comincia da **A** che costruisce 52 stringhe diverse $x_1, x_2, \cdots, x_{52}$ a caso, una per ogni carta del mazzo, e poi le cifra con la sua chiave pubblica ottenendo le stringhe: 7¡

$$E_A(x_1),\ E_A(x_2), \cdots,\ E_A(x_{52}).$$

   Quindi invia le 52 carte cifrate a **B** (chiaramente a **B** arrivano valori senza nessun tipo di ordinamento, cioè mischiate).

3. Poichè **B**, non è in grado di leggere nessuno dei 52 messaggi inviatigli da **A**, dopo aver scelto a caso le sue 5 carte, non può chiedere ad **A** di farsele decifrare perchè in questo modo **A** conoscerebbe le carte di **B**, quindi **B** farà come segue: valuta $E_B(E_A(x_i))$ per ognuna delle 5 carte che ha preso e poi invia questi valori ad **A**.

4. **A** riceve i 5 valori $E_B(E_A(x_i))$. Applica ad essi la sua funzione di decifratura, valuta $D_A(E_B(E_A(x_i)))$ per ognuna delle 5 carte ed invia tali valori a **B**. **B**, riceverà $E_B(x_i)$ e puó, quindi, decifrare le sue carte con la sua chiave privata.

5. A questo punto bisogna dare le carte al giocatore **C**, che dovrà sceglierle tra le restanti 47. Le 47 carte non possono tornare in mano ad **A** in quanto potrebbe risalire alle carte di **B** (il mazzo era stato cifrato con E_A), quindi è **B** che invia a **C** le 47 carte rimanenti.

6. **C** fa la stessa cosa di **B**:

   (a) Sceglie le sue 5 carte, le codifica con la sua chiave pubblica costruendo $E_C(E_A(x_i))$ e le invia ad **A**.

   (b) **A** applica la sua funzione di decifratura: $D_A(E_C(E_A(x_i)))$ per ognuna delle 5 carte di **C** e invia a **C** i valori $E_C(x_i)$.

   (c) **C** applica la sua funzione di decifratura ai valori pervenuti da **A** e ottiene le sue carte in chiaro.

7. Mancano le 5 carte al giocatore **A**. **C** sceglie 5 delle restanti 42 carte e le invia ad **A**. **A** calcola la funzione $D_A(E_A(x_i))$ ottenendo le sue 5 carte in chiaro.

I giocatori potrebbero aver bisogno di altre carte (cioè effettuare lo scarto come da regola del poker). Se **B** o **C** volessero altre carte possono prenderle dal mazzo cifrato tra le restanti 42 e riavviare il protocollo con **A**, se **A** volesse le carte saranno gli altri ad inviargliele (come ha fatto **C** nel passo 7).
Per verificare un possibile imbroglio ci sono due possibilità:

1. Alla fine della mano, tutti i partecipanti devono rivelare le carte e le loro chiavi. Questo può essere inserito come ultimo passo nel protocollo, ma a nessun giocatore fa piacere rivelare le carte in quanto si rivelerebbe un possibile bluff. Supponendo che esista un tale passo, si avrebbero le seguenti situazioni.

   (a) **A** vince, rivela le carte e le sue chiavi. **B** può usare la chiave privata di **A** per confermare che ella ha effettuato correttamente il passo 2, cioè gli ha inviato 52 carte differenti. **C** può confermare che **A** non ha barato in quanto, può prendere la chiave privata di **A** e verificare se le carte in suo possesso sono quelle che le ha inviato nel passo 7.

   (b) È **B** oppure **C** a vincere. **A** può legittimare la vincita in quanto, nel momento in cui **B** o **C** scoprono le carte e rivelano le loro rispettive chiavi private, può verificare se sono le carte che le hanno inviato quando le hanno chiesto di applicare la sua chiave privata (passi, 3 e 6 rispettivamente).

   Questo protocollo non è sicuro se ci sono 2 giocatori che si coalizzano per giocare contro il terzo. I due giocatori riescono a truffare l'altro senza essere scoperti.

2. Un altro modo è quello di dare delle prove a **conoscenza zero**, ogni utente può dimostrare agli altri che lui si è comportato in maniera corretta, senza scoprire le carte.
   La cifratura viene effettuata mediante chiavi private, cosicchè se esse venissero rese note, ognuno potrebbe verificare se il protocollo si è svolto correttamente. Quindi, è possibile **verificare** se un giocatore si è comportato in maniera corretta. Allora questa diventa una proposizione appartenente alla classe di problemi **NP** dove NP è l'insieme dei linguaggi verificabili in tempo polinomiale. In questo caso rappresenta l'insieme delle proposizioni su cui esiste una verifica. Ricapitolando, la proposizione **A** si è comportata in maniera corretta, è appartenente ad NP (perchè se **A** rendesse note le sue chiavi private, tutti potrebbero verificare se si è comportata correttamente). Se questa venisse trasformata in una istanza del problema della 3-colorabilità che è un problema **NP-completo**, **A** proverebbe agli altri che questa istanza è **3-colorabile** dimostrando la sua onestà senza scoprire le carte.

L'uso dell'**RSA** può sollevare qualche problema riguardante i bit i quali non tutti sono difficili. Vi sono dei bit facili e quindi qualche carta può essere considerata truccata, perchè quando è possibile calcolare dei bit significa che in qualche modo si hanno delle informazioni sulle variabili. Quindi, questo protocollo non va bene. Comunque esistono altri protocolli più complessi che risolvono il problema.

2 Lancio di una Moneta

Supponiamo che Alice e Bob vogliano giocare al lancio di una moneta in rete. Possono utilizzare il seguente protocollo:

1. **A** lancia una moneta e nasconde il risultato $b \in \{0, 1\}$ in una busta che poi spedisce a **B**.

2. **B** cerca di indovinare il valore del lancio effettuato da **A** quindi lancia una moneta e ottiene $c \in \{0, 1\}$.

3. Adesso **A** apre la busta e mostra il risultato b a **B**.

4. **B** effettua la seguente operazione:
$$b \oplus c$$
se il risultato è 0 ha indovinato, se il risultato è 1 ha sbagliato.

A oppure **B** non possono imbrogliare in quanto, qualunque cosa faccia **A** o **B**, il valore del bit finale è sempre 0 o 1 con probabilità $\frac{1}{2}$.
Al posto della busta può essere usato nel protocollo qualsiasi tipo di cifratura o il **bit commitment**.

3 Oblivious Transfer

I protocolli che analizzeremo in questo paragrafo, vengono usati per la trasmissione di insiemi di messaggi "segreti" tra due utenti [2]. La particolarità di questi protocolli rispetto ad altri, può essere compresa con un esempio. Supponiamo che una spia abbia carpito dei segreti e si trovi nella situazione di non poterli rivelare subito alla persona per la quale lavora, in quanto ha paura che questi lo abbandoni, e nemmeno può pensare di rivelarne una parte, perchè il ricevente potrebbe pensare che quelli inviati sono gli unici che la spia ha carpito.
La soluzione a questo problema viene fornita dai protocolli di trasferimento **oblivi** che consentono al ricevente di decifrare i messaggi nel seguente modo.

1. Riesce a decifrare ognuno dei messaggi con probabilità $\frac{1}{2}$, e il trasmettente non sa se il ricevente li ha decifrati o meno.

2. Riesce a decifrare uno dei messaggi che il trasmettente ha inviato e, anche in questo caso, colui che trasmette non sa quale messaggio inviato il ricevente ha decifrato.

Nell'esempio di sopra, la spia userà la seconda modalità di trasferimento, dove il ricevente è sicuro della presenza di tutti i segreti ma poichè riesce a decifrarne solo una parte, sarà costretto a tutelarla fino alla fine della missione. Vediamo in dettaglio i protocolli che consentono questi tipi di trasmissione.

3.1 Oblivious transfer con probabilità $\frac{1}{2}$

Siano **A** e **B** due utenti, e **A** possiede un segreto S. Si vuole realizzare un protocollo tale che **B** computa S con probabilità $\frac{1}{2}$ ed **A** non sa se **B** ha calcolato S. Analizziamo il protocollo.

1. **A** valuta un intero n come prodotto di due numeri primi $n = pq$, e lo invia a **B**
2. **B** valuta un valore casuale $r \in Z_n^*$. **B** calcola: $x = r^2 \mod n$, e lo invia ad **A**.
3. **A** conoscendo p e q, può valutare le quattro radici di x e sceglie un w tale che $w^2 = x \mod n$.
   A invia a **B** w.

Ricordiamo che un quadrato modulo n ha quattro radici. Supponiamo che siano r ed $n - r$, u ed $n - u$, allora:

1. Se $w = r$ oppure $w = n - r$, **B** non ottiene nessun tipo di informazione.
2. Se $w = u$ oppure $w = n - u$, **B** viene a conoscenza delle quattro radici di x.

Nel secondo caso, **B** può fattorizzare n. **A** dovrebbe 17 quindi mandare S a **B** in maniera oblivia usando questo protocollo e utilizzando n.
In realtà **A** manda a **B** il valore cifrato di S usando n, prima dell'inizio del protocollo.
Passo 0: **A** cifra con n il messaggio segreto S e lo invia a **B**. Adesso, se **B** riesce a fattorizzare n, decifra il messaggio S altrimenti no.
La decifratura è possibile ad esempio usando **RSA**.
Esempio:

$$y = S^e \mod n$$

$$S = y^d \mod n$$

quindi se **B** fattorizza n valuta d ed ottiene S.

3.2 Oblivious transfer 1 su 2

Supponiamo di dover risolvere il seguente problema: Alice trasmette un gruppo di messaggi a Bob, Bob riceve un sottoinsieme di questi messaggi, ma Alice non sa quali tra quelli che gli ha trasmesso Bob ha ricevuto.
Analizziamo un semplice protocollo che ha un reale utilizzo in pratica. Supponiamo che ci siano due utenti **A** e **B**, **A** possiede due segreti S_1 ed S_2 facciamo in modo che sia rispettata la seguente proprietà:
alla fine del protocollo **B** riesce a sapere uno dei due segreti, però **A** non sa che cosa ha scoperto **B**. Questo viene definito come un trasferimento in maniera oblivia.
Protocollo:

1. **A** genera 2 coppie di chiavi pubblica-privata, cioè genera:

   $$(E, D) \text{ e } (E', D')$$

   poi **A** manda a **B** le chiavi pubbliche E ed E'.

2. **B** sceglie una chiave k (ad esempio usando il **DES**), sceglie casualmente una delle due chiavi pubbliche di **A**, ed invia ad **A** un valore y derivato da una di queste due cifrature:

$$y = E(k) \text{ oppure } y = E'(k)$$

non rivelando ad **A** quale di queste due ha usato.

3. Adesso **A** riceve y ma non sa se è stato ottenuto usando E oppure E'.
Cioè, se k fosse stato cifrato con **RSA** allora in un caso si avrebbe: $y = k^e \mod n$ mentre nell'altro caso $y = k^{e'} \mod n$.
Poichè k è un valore scelto a caso, anche k^e e $k^{e'}$ divengono valori scelti a caso.
Quindi **A** non ha informazioni sul valore di y e lo decifra con entrambi gli algoritmi D e D', calcola le chiavi:

$$k_1 = D(y) \text{ e } k_2 = D'(y).$$

(dove k_1 sarebbe k se **B** avesse usato il primo sistema di cifratura E oppure, è k_2 ad essere k se **B** avesse usato il secondo sistema E').

4. A questo punto **A** invia a **B** due valori:

$$k_1 \oplus S_1 \text{ e } k_2 \oplus S_2$$

5. **B** adesso riesce a scoprire solo uno dei due messaggi (che è cifrato con la sua chiave DES k) di **A**, perchè

se $k = k_1$ allora recupera il messaggio S_1, altrimenti è $k = k_2$ e quindi recupera S_2 ma non può recuperarli entrambi.

Per verificare se **A** e **B** hanno agito correttamente si può pensare di far svelare agli utenti i loro segreti usati durante il protocollo, S_1, S_2 e gli algoritmi di decifratura (inseribile come ultimo passo nel protocollo stesso), o in alternativa utilizzare prove a **conoscenza zero**.
Il protocollo è sicuro da un attacco da parte di **A** in quanto **A** non conosce quale chiave k è stata usata da **B** per costruire y nel passo 2. (Perchè **A** può risalire alle due chiavi k_1 e k_2, una delle due è sicuramente la chiave DES k usata da **B**, ma non sa quale).
Il protocollo è anche sicuro da un attacco da parte di **B** perchè **B** può rompere il protocollo solo se conoscesse le chiavi private di **A**.

4 Key Escrow e Clipper

4.1 Introduzione

L'obiettivo del modello Key Escrow [3] e del relativo programma Clipper, elaborati agli inizi degli anni '90 dall'NSA, è quello di fornire al governo americano uno strumento per operare nel campo dell'intercettazioni telefoniche.
Nel 1993, la **AT&T**, una compagnia americana che gestisce le telecomunicazioni, ha commercializzato un dispositivo telefonico che permette di convertire in forma digitale la conversazione e cifrarla. Tale dispositivo possiede una chiave privata, e per ogni nuova comunicazione, genera una diversa chiave di sessione, la cifra e la trasmette al dispositivo ricevente.

Per il governo sorse quindi il problema di trovare un modo per rompere questo processo di cifratura senza intaccarne la sicurezza. Una soluzione abbastanza semplice poteva essere quella di permettere ad un ente statale di venire a conoscenza, di volta in volta, di tutte le chiavi di sessione generate dagli utenti della rete telefonica. Tale soluzione venne subito scartata in quanto avrebbe incontrato l'ovvia opposizione della pubblica opinione e venne invece elaborato il modello **Key Escrow**.

4.2 Key Escrow

Questo modello introduce la figura delle **agenzie fidate**, tali agenzie custodiscono tutte le chiavi di cifrature e dietro autorizzazione le svelano alle autorità che intendono effettuare una intercettazione. Per proteggere la sicurezza della cifratura, la chiave può essere suddivisa in più parti, e ogni parte viene spedita ad una agenzia diversa. In questo modo solo se tutte le agenzie collaborano si può recuperare la chiave di cifratura e nessuna di esse da sola ha sufficienti informazioni per ricostruirsi la chiave.

4.3 Clipper

Il chip Clipper è stato progettato dal **NSA** e implementa l'algoritmo di cifratura detto **SKIPJACK**. Tale algoritmo è rimasto segreto fino al 24 giugno 1998, data in cui il **NIST** lo ha reso pubblico. La descrizione dell'algoritmo **SKIPJACK** si può trovare presso il sito web del **NIST** [8]. Le caratteristiche principali sono le seguenti:

1. È un processo iterativo come il DES.
2. Usa 32 rounds, a differenza del DES che ne ha solo 16.
3. Usa una chiave di 80 bit e cifra blocchi di 64 bit.

Dati un messaggio M e una chiave k, il processo di cifratura gode della seguente proprietà :

$$\mathbf{D}(\mathbf{E}(M,k),k) = M$$

dove $\mathbf{E}$ è l'algoritmo di cifratura e $\mathbf{D}$ è l'algoritmo di decifratura. Il chip Clipper crea prima di inizialmente un campo detto **LEAF** (Law Enforcement Access Field), che contiene la chiave di sessione k cifrata con la chiave privata dell'utente, un numero n identificativo dell'utente, e un valore a di autenticazione.
Vediamo come viene implementato l'algoritmo.
Immaginiamo di avere due utenti **A** e **B** che intendono comunicare tra loro. L'utente **A** possiede una chiave privata u di 80 bit, essa viene suddivisa in due parti u_1 e u_2, tali che:

$$u = u_1 \oplus u_2$$

e manda u_1 alla prima agenzia e u_2 alla seconda agenzia.
Sul chip Clipper di **A** vi sono memorizzate:

- La chiave privata u.
- Un identificativo n di 80 bit.

- Un'ulteriore chiave f di cifratura di 80 bit che è comune a tutti i chip Clipper.

La chiave f è nota anche alle autorità, mentre u è noto alle due agenzie fidate o meglio ciascuna agenzia ne conosce una parte.
A, prima di iniziare la comunicazione con **B**, cifra la chiave di sessione k con la sua chiave privata u, alla stringa cosıottenuta si aggiunge l'identificativo n e il valore a di autenticazione. Infine cifra la risultante stringa con la chiave f:

$$\mathbf{E}((\mathbf{E}(k, u)\ , n\ , a),\ f)$$

Questo valore viene inviato a **B**.
Quando l'autorità vuole inserirsi su una linea telefonica esegue i seguenti passi:

1. Decifra $\mathbf{E}(\mathbf{E}((k, u)\ , n),\ f)$ per ottenere n.

2. Conoscendo n, chiede alle due agenzie le due chiavi u_1 e u_2 corrispondenti ad n.

3. Ricostruisce u:
$$u = u_1 \oplus u_2$$

4. Usa u per decifrare $\mathbf{E}(k, u)$ ottenendo k, cioè la chiave di sessione con la quale la comunicazione è stata cifrata.

5. Decifra la comunicazione adoperando k.

Il valore dell'autenticatore a è calcolato tramite funzioni hash e il suo ruolo è quello di autenticare la comunicazione tra **A** e **B**, cioè testimoniare la correttezza del formato della comunicazione.

4.4 Sicurezza di Skipjack

Sebbene coloro che svilupparono l'algoritmo inizialmente non volevano renderlo pubblico, per dimostrarne la sicurezza sottoposero l'algoritmo all'analisi di un gruppo di esperti, i quali trassero le seguenti conclusioni [7]:

1. Se si assume che il costo della potenza computazionale degli elaboratori si dimezza ogni diciotto mesi, allora sono necessari trentasei anni prima che il costo per rompere Skipjack tramite una ricerca esaustiva sia uguale al costo per rompere il **Des** oggi. In altre parole non ci sono rischi significativi che Skipjack venga rotto tramite la ricerca esaustiva nei prossimi trenta o quaranta anni.

2. La probabilità che Skipjack venga rotto attraverso un metodo di attacco non esaustivo è bassa.

3. La forza dell'algoritmo contro gli attacchi non dipende dalla sua segretezza.

In seguito alla divulgazione di **SKIPJACK**, Eli Biham, Adi Shamir e altri criptoanalisti, il 25 giugno 1998, tramite un attacco differenziale, hanno rotto una versione dell'algoritmo che utilizza solo 16 round. Alcuni giorni dopo sono riusciti a rompere una versione dell'algoritmo che utilizza 32 round ma 4 XOR in meno[9]. Attualmente sono allo studio nuovi attacchi.

4.5 Nomi

Inizialmente il nome dell'algoritmo di cifratura era **Skipjack**, il nome del chip che implementava l'algoritmo e generava il LEAF era **Clipper**, mentre **Capstone** era il nome del chip che implementava tutte le funzioni di **Clipper** e lo interfacciava con le altre macchine.
Successivamente Clipper venne ridenominato **Mosaic**, il chip Capstone **Fortezza** o **Tessera**.

5 Fair Cryptosystem

In alternativa alla proposta del NSA, Silvio Micali nel 1993 propose un protocollo chiamato **Fair Cryptosystem** [4], ovvero *criptosistema giusto* che riprendeva alcuni concetti del **Key Escrow** aggiungendo delle nuove propietà .
In questo crittosistema la chiave privata è frazionata in più parti e distribuita a differenti agenzie. Le autorità possono rimettere insieme le varie parti e ricostruirsi la chiave privata. La nuova proprietà di cui gode questo sistema è che ogni agenzia singolarmente può verificare la correttezza della parte di chiave in suo possesso, senza ricostruire la chiave privata.
Supponiamo che ci siano cinque agenzie :

$$AG_1,\ AG_2,\ AG_3,\ AG_4,\ AG_5$$

e due utenti **A** e **B** che vogliono comunicare. **A** prima di iniziare la comunicazione manda a ciascuna agenzia una parte diversa della propria chiave privata. Il protocollo assicura che:

- ogni agenzia può verificare che la parte di chiave che ha ricevuto è stata calcolata in modo corretto,
- mettendo insieme al più 4 agenzie non si riesce a sapere nulla sulla chiave,
- solo consultando tutte le 5 agenzie si può recuperare la chiave privata.

Vediamo ora come realizzare un **Fair Cryptosystem** utilizzando il protocollo di Diffie Hellmann. Ricordiamo che nel protocollo di Diffie-Hellman le parti **A** e **B** si accordano su due valori, p e α , dove p è un numero primo e α è un generatore di ${Z_p}^*$, (questi due valori possono essere noti a tutti o generati da uno dei due utenti **A** e **B** e poi comunicati all'altro).
A sceglie a caso un valore $s \in Z_{p-1}$ che sarà la sua chiave privata e rende pubblico il valore $\alpha^s \bmod p$; allo stesso modo **B** sceglie a caso un valore $r \in Z_{p-1}$ che sarà la sua chiave privata e rende pubblico il valore $\alpha^r \bmod p$. La loro chiave comune sarà :

$$K = \alpha^{sr} \bmod p.$$

Il protocollo seguente è un Fair Cryptosystem ottenuto dal protocollo di Diffie Hellman.

1. **A** sceglie un primo p ed un generatore α di ${Z_p}^*$, sceglie a caso un $s \in Z_{p-1}$ e poi sceglie cinque interi, $s_1, s_2, \cdots, s_5$ ognuno minore di $p-1$, tali che

   $$s = (s_1 + s_2 + \cdots + s_5) \bmod p - 1.$$

   A inoltre calcola: $t_i = \alpha^{s_i} \bmod p$ per $i = 1, 2, \cdots, 5$.
   A inserisce in un file pubblico:

   $$\alpha,\ p,\ \alpha^s \bmod p,\ t_1, \cdots,\ t_5.$$

   La chiave privata di **A** è il valore s ed i cinque interi $s_1, s_2, \cdots, s_5$.

2. **A** invia, in forma privata, il valore s_i all'agenzia AG_i, per $i = 1, 2, \cdots, 5$.

3. Ogni agenzia ricevuto s_i verifica che:

   $$t_i \ = \alpha^{s_i} \bmod p$$

4. Ogni agenzia AG_i è sicura che sommando la parte di chiave in suo possesso, il valore s_i, con le altre quattro chiavi in possesso alle restanti agenzie, ottiene proprio s, questo perchè ogni agenzia con le informazioni contenute nel file pubblico può verificare che:

   $$\alpha^s \bmod p = t_1 \times t_2 \times \cdots \times t_5.$$

Verifichiamo se questo protocollo va bene.
Gli s_i vengono inviati ad un server tramite un canale privato. Questo server, provvederà a distribuirli alle agenzie. Il file pubblico conterrà i seguenti valori: $p,\ s,\ \alpha,\ \alpha^s \bmod p,\ t_1,\ t_2, \cdots,\ t_5$. Ogni agenzia ha potuto calcolare se t_i coincide con $\alpha^{s_i} \bmod p$, quindi il sistema va bene. Notiamo che quattro agenzie non sono in grado di calcolarsi s in quanto dovrebbero calcolarsi il logaritmo discreto di $\alpha^s \bmod p$ oppure il logaritmo discreto di t_i, nell'ipotesi che l'agenzia i non sia presente. Solo se le cinque agenzie mettono insieme i loro s_i si può risalire alla chiave privata s, rompendo la comunicazione di **A** con ogni altro utente. Qualsiasi crittosistema a chiave pubblica può essere usato per costruire un Fair Cryptosystem.

6 Protocollo del milionario

Questo protocollo è chiamato **problema del milionario di Yao** [5], in quanto permette di stabilire chi tra due persone è più ricco senza che nessuno dei due riveli a quanto ammonta la sua ricchezza. Indichiamo con l'intero i la ricchezza di Alice e con l'intero j quella di Bob. Alice e Bob desiderano conoscere se $i \leq j$ oppure se $i > j$ [6]. Assumiamo $i\ \ j \leq 100$. Bob ha una chiave pubblica e una chiave privata.

1. Alice sceglie un numero casuale x abbastanza grande, e lo cifra con la chiave pubblica di Bob $c = E_B(x)$.

2. Alice computa $c - i$ e manda il risultato a Bob.

3. Bob computa i seguenti 100 numeri: $y_u = D_B(c - i + u)$ per $1 \leq u \leq 100$ dove D_B è l'algoritmo di decifratura che utilizza la chiave privata di Bob.
   Egli sceglie un primo p abbastanza grande in maniera casuale, e computa i seguenti 100 numeri: $z_u = y_u \bmod p$, per $1 \leq u \leq 100$. Poi verifica che per ogni $u \neq v$
   $$|z_u - z_v| \geq 2$$
   e che per ogni u
   $$0 < z_u < p - 1$$
   Se ciò non è vero Bob sceglie un altro primo e ricomincia.

4. Bob manda ad Alice questa sequenza di numeri in questo esatto ordine:
   $$z_1, z_2, \cdots, z_j,\ z_{j+1} + 1,\ z_{j+2} + 1, \cdots,\ z_{100} + 1,\ p$$

5. Alice controlla se il numero i-esimo nella sequenza è congruo a $x \bmod p$. Se lo è conclude che $i \leq j$ altrimenti $i > j$.

6. Alice comunica a Bob il risultato.

Tutte le verifiche che Bob compie nel terzo passo hanno lo scopo di garantire che nessun numero appaia due volte nella sequenza generata al quarto passo. Altrimenti se $z_a = z_b$ Alice sa che $a \leq j \leq b$. L'unico svantaggio di questo protocollo è che Alice apprende il risultato prima di Bob. Inoltre nessuno vieta ad Alice di fermarsi prima di comunicare il risultato a Bob.

6.1 Esempio

Supponiamo di utilizzare come algoritmo di cifratura l'**RSA**, la chiave pubblica di Bob è 7 e la sua chiave privata è 23. Il valore segreto di Alice è $i = 4$ e quello di Bob è $j = 2$, $n = 55$. Assumiamo per semplicità che i e j possono assumere solo i valori $1,\ 2,\ 3,\ 4$.

1. Alice sceglie $x = 39$ e $c = E_B(39) = 19$

2. Alice computa $c - i = 19 - 4 = 15$ e spedisce il risultato a Bob.

3. Bob computa i seguenti 4 numeri:
   $$y_1 = D_B(15 + 1) = 26$$
   $$y_2 = D_B(15 + 2) = 18$$
   $$y_3 = D_B(15 + 3) = 2$$
   $$y_4 = D_B(15 + 4) = 39$$
   Bob sceglie $p = 31$ e calcola:
   $$z_1 = (26 \bmod 31) = 26$$
   $$z_2 = (18 \bmod 31) = 18$$
   $$z_3 = (2 \bmod 31) = 2$$
   $$z_4 = (39 \bmod 31) = 8$$
   Egli verifica che la sequenza sia corretta.

4. Bob spedisce ad Alice questa sequenza di numeri in questo preciso ordine:

$$26,\ 18,\ 2+1,\ 8+1,\ 31$$

5. Alice controlla se il quarto numero nella sequenza è congruo a x mod p. Poichè $9 \neq 39$ mod 31 allora $i > j$.

6. Alice comunica il risultato a Bob.

7 Firma di un contratto

Supponiamo di avere due utenti di una rete, Alice e Bob, che vogliono firmare un contratto e nessuno dei due si fida dell'altro. Alice e Bob producono una firma digitale e quindi devono scambiarsi una stringa.
Poichè lo scambio simultaneo delle due firme non si può realizzare, uno dei due deve mandare per primo la sua firma e solo successivamente riceve la firma dell'altro. In questo modo uno dei due può approfittare dell'altro, ad esempio Bob potrebbe far firmare Alice e poi non mandargli la sua firma. Per evitare questo Alice e Bob possono decidere di scambiarsi un bit alla volta. Anche con questo metodo però c'è un problema: cosa accade se una persona non spedisce la firma ma una stringa qualsiasi? L'altro se ne accorgerà solo alla fine, cioè dopo che gli avrà inviato tutta la sua firma.
Si può ridurre la possibilità di imbrogliare utilizzando l'**oblivious transfer**.
Alice crea L_A e R_A che sono la parte sinistra e la parte destra della sua firma, allo stesso modo Bob crea L_B e R_B. Alice sceglie due chiavi DES: $(K_A)^L$ e $(K_A)^R$ con le quali cifra L_A ed R_A, producendo rispettivamente $(C_A)^L$ e $(C_A)^R$, Bob fa lo stesso producendo $(C_B)^L$ e $(C_B)^R$. Il contratto è valido se ha entrambe le firme (parte destra e parte sinistra).
Alice e Bob fanno l'**oblivious transfer 1 su 2** delle loro chiavi K^L e K^R, supponiamo che Bob ottiene $(K_A)^L$ e Alice $(K_B)^R$, allora Bob può ottenere $(C_A)^L$ e quindi conoscere la parte sinistra della firma di Alice; allo stesso modo Alice può ottenere $(C_B)^R$ e quindi conoscere la parte destra della firma di Bob.
Alice e Bob spediscono ognuno un bit della propria firma finchè tutti i bit sono spediti. In questa fase se uno dei due invia una stringa non corrispondente alla sua firma potrà essere scoperto prima della fine dello scambio con probabilità $\frac{1}{2}$ e quindi l' altro potrà decidere di fermarsi.

8 Elezioni

L'espressione di voto tramite computer è attuabile nelle elezioni politiche se esiste un protocollo in grado di mantenere la segretezza individuale dei votanti e di impedire la falsificazione del voto [1].
Un protocollo ideale deve avere almeno questi sei requisiti.

1. Possono votare solo coloro che sono autorizzati.

2. Nessuno può votare più di una volta.

3. Nessuno può determinare il voto degli altri.

4. Nessuno può duplicare il voto degli altri.

5. Nessuno può cambiare il voto di altri senza essere scoperto.

6. Tutti i votanti devono essere sicuri che il loro voto andrà ad incrementare un tabulato finale.

Inoltre, qualche schema di votazione più complesso, può richiedere anche un ulteriore requisito: ogni votante conosce chi ha votato e chi non lo ha fatto.
Analizziamo qualche semplice protocollo che fa uso di un server che chiameremo **CTF**, (acronimo di Central Tabulating Facility).

Primo protocollo di votazione semplificato con server.

1. Ogni votante, codifica il suo voto con la chiave pubblica del **CTF**.

2. Ogni votante, inserisce il suo voto nel **CTF**.

3. Il **CTF** decodifica i voti, ne effettua lo spoglio e rende i risultati pubblici.

Questo protocollo ha dei problemi. Il **CTF** non riesce a determinare se il voto proviene da un utente che ne ha diritto, e non sa se i votanti hanno votato più di una volta. Ciò che il protocollo non consente di fare è cambiare il voto degli altri, ma questo non è molto interessante in quanto nessuno si preoccupa di cambiare il voto di altri, quando per lui è più comodo votare più volte.

Secondo protocollo di votazione semplificato

1. Ogni votante firma il suo voto con la sua chiave segreta.

2. Ogni votante codifica il suo voto firmato con la chiave pubblica del **CTF**.

3. Ogni votante invia il suo voto al **CTF**.

4. Il **CTF** decodifica il voto, controlla la firma, spoglia i voti e rende noto il risultato.

Questo protocollo soddisfa le prime due proprietà del protocollo ideale (solo votanti autorizzati possono votare e nessuno può votare più di una volta). Infatti, il **CTF** registra i voti ricevuti nel terzo passo, ogni voto è firmato con la chiave segreta del votante, in questo modo il **CTF** conosce chi ha votato e chi no e quante volte ogni votante ha votato. Se il voto proviene da un utente che non ha diritto, oppure, il votante che ne ha diritto invia un voto doppio, il **CTF**, nel primo caso ignora il voto, nel secondo caso ignora solo il secondo voto. Nessuno può cambiare il voto degli altri in quanto verrebbe intercettato nel passo 2, perchè il votante deve apporre sul voto la sua firma digitale.
Il problema di questo protocollo deriva dal fatto che la firma è attaccata al voto e quindi, il **CTF** conosce chi ha votato e la preferenza che ha espresso, di contro nessun altro votante può

scoprire il voto degli altri, e nemmeno effettuarne il conteggio, in quanto i voti sono cifrati con la chiave pubblica del **CTF**.
Concludendo, l'aspetto negativo riguarda il fatto che ci si deve fidare completamente del **CTF**,e questo è paragonabile a dover esprimere il voto nella cabina di votazione in presenza del presidente del seggio.

8.1 Votazione con Blind Signature

Analizziamo un ultimo protocollo che fa uso di un server. Cerchiamo di dissociare il voto dal votante, mantenendo la firma del voto [1].
Quello che si vuole realizzare in questo protocollo è il consenso da parte del server dell'uso di una scheda, debitamente compilata dal votante, per la votazione. Tale scheda conterrà tutti i possibili modi per esprimere il voto ed un numero di identificazione che rappresenta la firma del votante. La particolarità del protocollo è questa:
il **CTF** vidima la scheda senza vederne il contenuto e quindi senza leggere la firma del votante. Per essere sicuro che il votante abbia compilato la scheda in modo corretto, fa quanto segue: si fa inviare dal votante un certo numero (ad esempio dieci) di schede (ricordiamo che queste devono contenere tutti i possibili modi per esprimere una preferenza, e la firma del votante), le apre tutte tranne una, controlla se sono state compilate correttamente, le vidima (in particolare vidima anche quella chiusa) e dopo aver memorizzato il nome del votante, gli invia indietro tutte le schede. A questo punto è chiaro che il votante con buona probabilità ha compilato bene le schede, (nell'esempio di dieci la probabilità che ha agito scorrettamente è $\frac{1}{10}$ e con l'aumentare del numero delle schede diventa sempre più bassa).
Analizziamo nei dettagli il protocollo.

1. Ogni votante genera dieci set di messaggi, ogni set contiene tutti i possibili voti (se ad esempio si tratta di un referendum, ogni set contiene due messaggi, uno che identifica l'espressione di voto **SI** e l'altro che identifica **NO**). Ogni messaggio contiene anche un numero di identificazione generato da ogni votante in modo casuale, grande a sufficienza in modo da evitare la duplicazione da parte di altri votanti.

2. Ogni votante nasconde i messaggi e li invia, con i loro numeri nascosti (blinding factors), al **CTF**.

3. Il **CTF** controlla il suo database per essere sicuro che il votante non abbia già chiesto la vidimazione della scheda precedentemente. Poi apre nove degli insiemi per controllare che siano stati propriamente scritti, se tutto è stato eseguito secondo le regole appone la **blind signature** sui messaggi nell'insieme che non è stato aperto, lo invia indietro al votante e memorizza il suo nome nel proprio database.

4. Il votante, apre i messaggi firmati dal **CTF**. (Tali voti sono firmati ma non cifrati, in questo modo il votante può facilmente vedere dove apporre la preferenza).

5. Il votante esprime la sua preferenza e la codifica con la chiave pubblica del **CTF**.

6. Il votante invia il suo voto.

7. Il **CTF** decodifica i voti, controlla il suo database per scoprire eventuali numeri di identificazione duplicati e spoglia i voti. Pubblica poi il risultato dello scrutinio, cioè i numeri di serie ed i voti ad essi associati.

Possibili attacchi al protocollo:

1. Votante scorretto che cerca di ottenere più schede per la votazione. Il **CTF** scoprirà la richiesta di una ulteriore scheda nel passo 3, infatti in tale passo memorizza nel suo database le richieste di schede da parte dei votanti che hanno diritto al voto.

2. Votante scorretto che vuole votare più volte. Il **CTF** lo scoprirà al passo 7, dove rileverà la presenza di numeri di identificazione uguali.

3. Nemico che vuole votare senza averne il diritto. Questo non è possibile in quanto non si conosce la vidimazione del **CTF**, che questi appone sulle schede al passo 3.

4. Nemico che intercetta il voto di altri e vuole cambiarlo. Tale attacco fallirà in quanto il voto inviato al passo 6 è cifrato e nessuno può risalire alla chiave privata del **CTF**.

Le proprietà di cui gode il protocollo sono le seguenti:

1. Unicità del voto. È assicurata dal passo 3 dove non è possibile richiedere più schede, e dal passo 7 dove vengono rilevati voti multipli che contengono lo stesso numero di identificazione.

2. Segretezza del voto. Questa proprietà è soddisfatta perchè l'insieme usato dal votante è rimasto chiuso e quindi nessuno può associare il voto con la firma. Però se il **CTF** effettuasse lo spoglio del voto man mano che i votanti gli inviano il voto, è in grado di associare il voto con la firma e quindi verrebbe violata questa proprietà. Il problema è risolto con l'assunzione di effettuare lo spoglio dei voti quando tutti hanno votato (cioè simulando un'urna chiusa che viene aperta alla fine della votazione).

3. Sicurezza sullo spoglio. Il protocollo al passo 7 rende pubblici i voti con il numero di serie associato quindi tutti possono controllare se il proprio voto è stato spogliato.

Esistono comunque dei problemi che nascono da un **CTF** scorretto.

1. Il **CTF** potrebbe generare voti validi senza essere scoperto in quanto nessuno può stabilire se il numero di identificazione che compare accanto alla preferenza appartiene o meno agli altri votanti (ricordiamo che i votanti sono solo in grado di verificare se compare accanto al voto il proprio numero di identificazione).

2. Il **CTF** potrebbe anche cambiare il voto che gli è stato inviato senza modificare il numero di identificazione. Il votante interessato si accorgerà dell'imbroglio ma non potrà dimostrare la scorrettezza del **CTF**.

8.2 Votazione senza uso di server

Analizzeremo un protocollo che mantenga tutte le caratteristiche di una votazione corretta e che non utilizza il server.
Supponiamo che per le elezioni vi siano solo due candidati **X** e **Y** e cinque votanti **A, B, C, D, E**:
protocollo:

1. Viene scelto un numero primo p maggiore del numero degli utenti della rete.
2. **A** sceglie a caso due valori iniziali r ed s con $r, s \in Z_p$.
   dove, r è il contatore dei voti di **X** ed s è il contatore di voti di **Y**.
   A invia a **B** i valori r ed s.
3. Adesso **B** esprime il suo voto, se sceglierà **X** verrà incrementato r, altrimenti verrà incrementato s.
   Supponendo che **B** abbia scelto **X**, egli invierà a **C** la coppia di valori $(r+1, s)$, (ricordiamo che le operazioni sono effettuate mod p).
4. L'utente **C**, esprimerà il suo voto incrementando uno dei due contatori ed invierà la coppia di valori così modificata a **D**.
5. **D** esprimerà il suo voto ed invia la coppia ad **E**.
6. A questo punto **E** esprimerà il suo voto ed invierà ad **A** la coppia di valori (r', s').
7. L'utente **A** ricevuta la coppia di valori (r', s') esprimerà il suo voto, sia ad esempio la preferenza per il candidato **Y**, e poi sottrarrà ai valori, $(r', s'+1)$, i valori casuali (r, s) che aveva scelto nel passo 1, ottenendo quindi il risultato della votazione.

Puntualizziamo che le comunicazioni tra gli utenti sono cifrate e quindi ogni utente non vede ciò che è stato mandato agli altri utenti.
Si possono verificare le due seguenti situazioni:

1. Utenti **onesti** (cioè seguono il protocollo), ma **curiosi** (vogliono scoprire il voto degli altri).
2. Utenti **non onesti**.

Nel primo caso supponiamo che gli utenti non possono collaborare tra di loro e che la persona curiosa sia l'utente **C**. **C** non potrà sapere niente circa il voto degli altri in quanto i voti che vede **C** sono mascherati dai valori iniziali r ed s scelti a caso quindi non ha informazioni sul voto espresso dagli utenti precedenti. Quindi il protocollo va bene se la persona curiosa è unica.
Se invece le persone curiose sono più di una allora è possibile ottenere qualche informazione, infatti se **C** ed **A** si coalizzassero insieme, riescono a scoprire il voto di **B**.
Nel secondo caso, può accadere che ci sia una persona non onesta ad esempio **A** il quale potrebbe fornire un valore diverso dal reale risultato dell'elezione. Questo problema può essere risolto obbligando l'utente in questione a fornire una dimostrazione sulla sua correttezza, ad esempio utilizzando una prova di tipo **a conoscenza zero**. Vediamo come fare:

B riceve $C_B(r)$ e $C_B(s)$ ovvero riceve i valori r ed s cifrati con la sua chiave pubblica.
C riceve i valori $C_C(r+1)$ e $C_C(s)$.
B può dimostrare la cosa seguente:
C cifra una coppia di valori $C_C(\alpha)$ e $C_C(\beta)$ dove α e β possono assumere solo i seguenti valori:

1. $\alpha = r,\ \beta = s+1.$

2. $\alpha = r+1,\ \beta = s.$

quindi dimostra che la coppia $C_C(\alpha),\ C_C(\beta)$, è stata computata in maniera corretta.
Questa affermazione è **verificabile**, infatti, se è nota la chiave privata di **B**, si valutano i valori r ed s e data la chiave privata di **C**, si verifica se è r che è stato incrementato, oppure è s che è stato incrementato. Quindi si può verificare se **B** ha agito correttamente o meno.
Dire che tale proposizione è **verificabile** significa dire che il problema è in **NP**. Abbiamo visto che un problema verificabile (che si trova in **NP**), è trasformabile in una istanza del problema della **3-colorabilità** che è **NP-Completo**, e quindi ammette un sistema di prova **a conoscenza zero**. Ognuno a turno può provare agli altri che sta agendo in maniera corretta usando tale sistema.
Il protocollo soddisfa le seguenti proprietà:

1. Gli utenti non possono imbrogliare.

2. Due o più persone curiose riescono a scoprire i voti espressi dagli altri votanti.

Esistono variazioni di questo protocollo che tollerano un numero di persone curiose molto più grande ad esempio la metà degli utenti (però sono troppo complessi).

Riferimenti

[1] Bruce Schneier, *Applied Cryptography, Protocols, Algorithms and Source Code in C*, John Wiley & Sons, Inc. 1995 (Seconda edizione).

[2] J. Kilian, *Uses of Randomness in Algorithms and Protocols*, MIT Press, 1990.

[3] W. Stallings, *Network and Internetwork Security*, Englewood Cliffs, N.J.: Prentice-Hall, 1995.

[4] Silvio Micali, *Fair Public-Key Cryptosistems, Advances in Cryptology CRYPTO '92 Proceedings*, Berlin: Springer-Verlag, 1993.

[5] A.C.-C. Yao, *Protocols for Secure Computations*, Proceedings of the 23rd IEEE Symposium on the Foundations of Computer Science, 1982.

[6] A. Salomaa, *Public-Key Cryptography*, Springer-Verlag, 1990.

[7] Ernest F. Brickell, Dorothi E. Denning, Stephen T. Kent, David P. Maher, Walter Tuchman, *The Skipjack Algorithm: Interim Report*, 1993.
Disponibile al seguente indirizzo:
http://www.austinlinks.com/Crypto/skipjack-review.html.

[8] http://csrc.nist.gov/encryption/skipjack-kea.htm

[9] http://www.cs.technion.ac.il/~biham/Reports/Skipjack