Sono Sicuro che è Sicuro?
Modelli Formali per la Sicurezza

Problema

- I mecanismi e i servizi di sicurezza raramente garantiscono la capacità rispettivamente di individuare/prevenire attacchi e di migliorare la sicurezza dei sistemi
- Si tratta di apparati complessi, di cui è estremamente difficile / oneroso valutare il comportamento

Un Approccio alla Soluzione

- L'analisi delle caratteristiche di un sistema di sicurezza è spesso ardua
  - Molte delle proprietà sono indecidibili, anche se spesso semidecidibili
- La tesi di fondo è che l'analisi delle proprietà può essere grandemente semplificata, se svolta su un opportuno modello del sistema

Modellazione (1)

- Esistono diversi tipi di modellizzazioni
  - Informali
  - Semiformali
  - Formali

Modellazione (2)

- I modelli non devono e non possono rappresentare tutto il sistema
  - Separazione degli aspetti il più possibile ortogonali fra loro
- Per ogni aspetto di interesse si definisce un modello che:
  - lo rappresenti come concetto “chiave”
  - che astragga da altri aspetti meno importanti

Modellazione (3)

- Per risolvere un problema estremamente complesso
  - lo si divide in diversi livelli di astrazione, affrontati in sequenza, ad esempio: top-down, bottom-up, o combinati
- La scelta del modello è essenziale per poter affrontare problemi complessi e per la qualità della realizzazione

Modellazione (4)

- È spesso necessario adottare modellizzazioni che favoriscano la validazione dell’applicazione rispetto ai parametri di qualità desiderati

Svantaggio della formalizzazione

- Adottare metodi di sviluppo formali:
  - è difficile
  - è impegnativo
  - richiede molto tempo
  - richiede elevate competenze

Modelli Formali

- Petri Nets
- Abstract State Machine
- Process Algebras

PN: Generalità (1)

- Una Rete di Petri (Petri Net - PN) è
  - un modello astratto e formale
  - per la rappresentazione del comportamento dinamico di sistemi discreti
  - che esibiscono attività asincrone e concorrenti

PN: Generalità (2)

- Molto usate nella modellizzazione di
  - Sistemi concorrenti
  - Interazione tra sistemi diversi, compresa utente-computer
  - Protocolli di comunicazione
  - Workflow
  - Sistemi complessi
  - ...

PN: Notazione (1)

- Concettualmente una PN è costituita
  - da un insieme di elementi, detti posti, che rappresentano i possibili stati del sistema
  - da un insieme di transizioni, che rappresentano gli eventi che quando si verificano determinano cambiamenti di stato
  - da un insieme di elementi, detti token, la cui presenza/assenza/numerosità/tipo/... permette l'attivazione delle transizioni

PN: Notazione (2)

- Le PN sono rappresentate come grafi
  - i cui nodi sono
    - i posti, raffigurati con dei cerchi
    - le transizioni, raffigurate con dei rettangoli
  - i cui archi sono i link che collegano posti e transizioni
- I token sono rappresentati come pallini all'interno dei posti

PN: Notazione (3)

- Attenzione: a differenza di altri formalismi, le transizioni di stato NON sono raffigurate come i link tra gli stati
- Gli archi di input connettono posti con transizioni
- Gli archi di output connettono transizioni con posti

PN: Notazione (4)

- In un dato istante uno o più posti possono essere marcati
  - La marcatura (token) indica che determinati eventi relativi a quel posto si sono compiuti
  - Una marcatura è indicata da un pallino all'interno di un posto

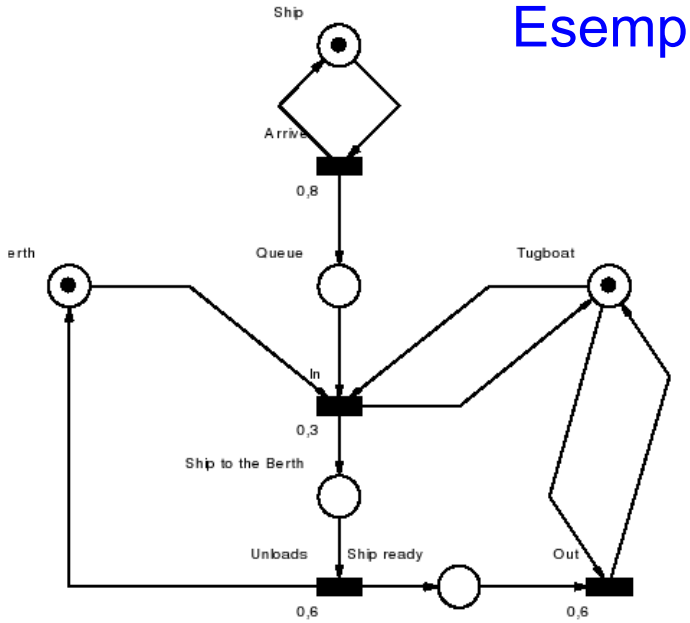
Esempio (1)

- Un sistema portuale prevede
  - un molo, dove attraccano le navi
  - un rimorchiatore, che scorta l'ingresso/uscita delle navi
- Le navi arrivano all'esterno di un porto con una frequenza di 80 minuti
- Una nave è scortata dall'esterno al molo da un rimorchiatore
  - tempo richiesto: 30 min.
  - solo dopo l'attracco della nave al molo il rimorchiatore diventa disponibile per altro

Esempio (2)

- Le operazioni di scarico della nave richiedono 60 min
- Dopo aver completato lo scarico, se il rimorchiatore è disponibile scorta la nave all'esterno
  - tempo richiesto 60 min
  - il molo diventa disponibile per altre navi

Esempio (3)



ASM: Idee guida

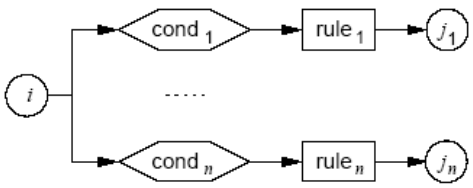
- ASM = FSM con stati generalizzati
  - Le ASM rappresentano la forma matematica di Macchine Astratte che estendono la nozione di Finite State Machine
- Ground Model (descrizioni formali)
- Raffinamenti

Finite State Machine (1)

- Un automa a stati finiti è definito da una 5-pla: FSM = <Q, Σ, δ, q0, F>, dove:
  - Q è l'insieme finito e non vuoto degli stati dell'automa, ciascuno caratterizzato da una particolare configurazione di valori delle variabili di stato
  - Σ è l'alfabeto di input
  - δ è la funzione transizione di stato δ :(Q X Σ)→Q (automi deterministici)
  - q0 è lo stato iniziale
  - F $\subseteq$Q è l'insieme di stati finali

Finite State Machine (2)

- Una FSM può essere definita da un programma della forma




```
if ctl_state = i then
  if cond_1 then
    rule_1
    ctl_state := j_1
      ...
  if cond_n then
    rule_n
    ctl_state := j_n
```


Finite State Machine (3)

- dove
  - ctl_state rappresenta lo stato, i cui valori appartengono a un insieme finito
  - i, j1, j2, …, jn, sono stati interni (i valori di ctl_state)
  - condk (k=1, 2, …, n) rappresentano le condizioni di input
  - rulek le azioni di output

Finite State Machine (4)

- Definizione in forma testuale (alternativa)

FSM(*i*, **if** *cond* **then** *rule*,*j*) =

if *ctl_state* = *i* **and** *cond* **then** {*rule*, *ctl_state*:=*j*}

Da FSM a ASM (1)

- Le ASM sono analoghe alle FSM
- Le differenze riguardano
  - la concezione degli stati:
    - nelle FSM esiste un unico stato di controllo (ctl_state), che può assumere valori in un insieme finito di un certo tipo
  - le condizioni di input e le azioni di output
    - alfabeto finito
  - la potenza computazionale
    - le ASM soddisfano la Tesi di Church-Turing

Da FSM a ASM (2)

- Nelle ASM invece gli stati sono associati a un insieme di valori di qualsiasi tipo, memorizzate in apposite locazioni
  - Una locazione rappresenta il concetto astratto di unità di memoria, indipendente dal particolare meccanismo di indirizzamento
  - L'astrazione al livello opportuno è ottenuta mediante parametrizzazione delle locazioni

Da FSM a ASM (3)

- Più precisamente: gli stati delle ASM sono strutture matematiche in cui i dati sono oggetti astratti, e cioè elementi di insiemi a cui è possibile applicare operazioni e predicati
- Conseguenza
  - Le transizioni di stato delle FSM corrispondono alle transizioni di stato delle ASM, ma in più con aggiornamenti dei valori contenuti nelle locazioni
    - Assegnamenti della forma loc(x1, x2, …, xn) := val

Da FSM a ASM (4)

- La differenza tra il concetto di stato di FSM e ASM porta a macchine i cui stati possono essere domini di qualsiasi oggetto
- Analogamente all'estensione degli stati FSM (non strutturati) in stati ASM (strutturati), le condizioni di input di FSM sono estese ad arbitrarie espressioni sugli stati nelle ASM
  - Queste espressioni sono sentinelle (guard), in quanto determinano l'istruzione che deve essere eseguita.

Definizione di ASM

- Insieme di istruzioni (regole ASM) della forma

  if cond then Updates

dove:

  - Updates è un insieme di aggiornamenti di funzioni f(t1, t2, …, tn) := t

Nota

- Si parla di "regole ASM" per evidenziare la distinzione tra
  - il modello di esecuzione parallela per le ASM
  - e il modello di esecuzione a singola istruzione della programmazione tradizionale

Concetto di funzione

- Il termine funzione deve essere inteso in senso matematico, non informatico
  - Per esprimere dal punto di vista matematico il concetto di funzione, possiamo pensarla come una tabella contenente valori
  - Quando si parla di location si può pensare all'indicizzazione di una cella della tabella

Esempio ASM per Kerberos

- Il sistema è modellato mediante una DASM, costituita da quattro ASM:
  - MessagePassing
  - EncryptionDecryption
  - MultipleClients
  - MultipleEndServers

MessagePassing (1)

- È il modello che formalizza il formato dei messaggi scambiati tra gli agenti
  - In prima istanza consideriamo un modello semplificato, contenente un solo Client (C) e un solo EndServer (ES)
- La DASM relativa comprende una ASM per ciascuno dei 4 agenti del modello
  - Il KAS; il Client C; il TGS; l'EndServer ES
- Quindi, l'universo AGENT={KAS, C, TGS, ES}

MessagePassing (2)

- L'Universo MESSAGE contiene messaggi di due tipi, composti (concatenazione di dati) o crittografati (secondo una key), indicati rispettivamente con {X, X'} e $\{X, X'\}_{key}$
  - Per conoscere il tipo di un messaggio si usa la funzione type: MESSAGGE → {cleartext, encrypted}

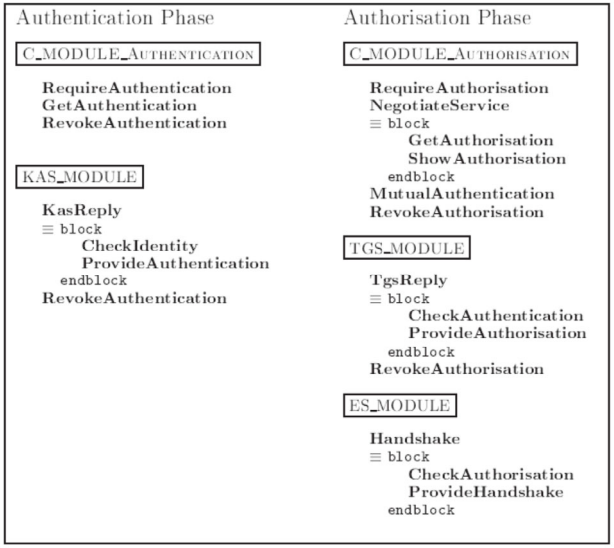
MessagePassing (3)

- Le chiavi, i ticket e le autenticazioni sono modellati da oggetti atomici
  - appartenenti rispettivamente agli universi KEY, TICKET e AUTH
- La funzione K: AGENT → KEY restituisce la chiave privata dell'agente, conosciuta solo dall'agente stesso e registrata nel db delle chiavi

MessagePassing (4)

- Le chiavi, i ticket e le autenticazioni sono spediti all'interno dei messaggi, da cui vengono estratti mediante le funzioni:
  - ExtractKey: MESSAGE → KEY
  - ExtractTicket: MESSAGE → TICKET
  - ExtractAuth: MESSAGE → AUTH
- Altre funzioni
  - sender, receiver: MESSAGE→ AGENT
  - mode: AGENT → {ReadyToSend, ReadyToReceive, ReadyToStart}

MessagePassing – Specifica



EncryptionDecryption

- Gli oggetti astratti del modello precedente sono raffinati dalle procedure di crittografia-decrittografia usate per costruire le credenziali del client
- Le funzioni sono:
  - encrypt: DATA X KEY → CRYPTDATA
  - decrypt: CRYPTDATA X KEY → DATA

  tali che

  decrypt(encrypt(t,k), k')=t se k=k',
  =undef altrimenti

Correttezza (1)

- Condizioni globali: ad ogni livello di specifica devono valere le seguenti
  - G1: defined(K(C)) & defined(K(TGS)) & defined(K(ES))
  - G2: NOT authenticated(C) ➔ NOT authorised (C)
- Condizioni Iniziali
  - Specifica degli stati

Correttezza (2)

- Condizioni di lavoro:
  - W1: monotonia del clock
  - W2: correttezza della password
  - W3: limite superiore al tempo di reazione del server
  - W4: validità delle credenziali del client

Correttezza (3)

- Esecuzione regolare: È regolare ogni esecuzione tale che
  - Le condizioni globali G1 e G2 sono soddisfatte
  - Le condizioni iniziali soddisfano la specifica degli stati
  - Le condizioni di lavoro W1, W2, W3, W4 sono soddisfatte

Communicating Sequential Processes

- CSP è una notazione per descrivere sistemi in cui agenti operano in parallelo e comunicano scambiandosi messaggi
- Ottimo strumento per studiare la concorrenza
  - In questo corso CSP sarà presentato come strumento per la modellizzazione
- Poco adatto per comunicare con parti interessate che NON siano informatici

Scopo

- Notazione per la descrizione delle interazioni tra agenti
- Particolarmente utile nel caso di sistemi in cui alcune componenti svolgono attività computazionali influenzate dalle attività computazionali svolte da altre componenti
  - Adatto per modellizzare componenti di sistemi distrubuiti

Applicazioni

- Modellizzazione di
  - interazione utente-sistema
  - protocolli (ad es. sicurezza)
  - sistemi di controllo safety-critical
- Specifica di sistemi

Interazione = Comunicazione

- Presupposto: l'interazione tra due componenti di un sistema, o di un sistema con l'esterno avviene mediante un'opportuna comunicazione
- Nell'ambito di CSP la comunicazione prende la forma di eventi o azioni visibili
  - All'interno di un processo vengono anche svolte attività interne, invisibili, che hanno effetti indiretti sul mondo esterno

Nozioni di base (1)

- CSP fornisce una modalità
  - per descrivere gli stati attraversati dal sistema
  - quali azioni sono eseguite
- Sia
  - Σ l'insieme di tutti gli eventi visibili di un processo
  - τ l'azione svolta internamente
    - per il momento è sufficiente l'astrazione secondo cui il processi svolge un'unica azione

Nozioni di base (2)

- Nel seguito useremo indifferentemente le espressioni
  - processo / programma
  - comunicazione / azione visibile

Processi equivalenti

- Chiamiamo equivalenti due diversi processi ciascuno dei quali produce un proprio pattern di azioni visibili e tali pattern non possono essere distinti l'uno dall'altro
- Di conseguenza, la caratteristica di uno specifico processo è data dalle sue forme di comunicazione

Processo Stop

- Il processo più semplice è Stop, che non svolge alcuna azione
  - nessuna azione visibile
  - nessuna azione interna
- E' il più semplice processo equivalente ad ogni processo che non comunica con altri
- E' una modalità per esprimere il deadlock

Prefixing

- Dato un processo P e una azione a, allora il processo a -> P (a then P) è il programma
  - che svolge l'azione a
  - e poi si comporta come P
- Se in e out sono due azioni in Σ, allora il processo in -> out -> Stop svolge le azioni in successione e poi si ferma
  - In realtà l'ambiente del processo potrebbe scegliere di non accettare un'azione

Forma più compatta

- È possibile rappresentare in forma più compatta il processo Alt precedente secondo la seguente notazione

μP.to -> fro -> P

Scelta di azioni (1)

- Il comportamento di un processo può essere conseguenza delle azioni svolte
- Sia A un sottoinsieme di Σ, allora il processo

?x : A -> P(x)

(scegli un x in A e P prosegue di conseguenza)

permette all'ambiente di scegliere una tra tutte le possibili azioni visibili in A

Scelta di azioni (2)

- Un caso particolare di scelta è

RUN_A = ?x:A-> RUN_A

  - lo stato RUN_A risultante da ogni scelta è sempre lo stesso
  - Si tratta di un processo a comportamento costante

Parametrizzazione

- Scegliendo un a ∈ A allora il processo prosegue comportandosi come P(a)
  - rappresenta la parametrizzazione del processo rispetto all'azione a
- a si comporta come un parametro per P(a)
- Strumento per la parametrizzazione del comportamento del processo
- Può essere usato per decidere come comportarsi a seguito dei possibili casi scelti

Scelta tra due processi (1)

- L'operatore □ permette di scegliere di continuare l'esecuzione secondo quanto previsto da due processi distinti

  (?x : B->P(x)) □ (?x : C->P(x))

- Se A = B ∪ C allora la seguente indica che i due processi si comportano in modo equivalente

?x : A->P(x) = (?x : B->P(x)) □ (?x : C->P(x))

Scelta tra due processi (2)

- Se B = Ø allora, dal momento che non viene fornita alcuna possibilità all'ambiente, si ottiene ?x : B->P(x) = Stop
- Quindi, essendo A = A ∪ Ø, si ottiene

  (?x : A->P(x)) = (?x : A->P(x)) □ Stop

- In generale, P □ Stop è equivalente a Stop

Scelta tra due processi (3)

- Il processo risultante da

  (?x : B->P(x)) □ (?x : C->P(x))

- non genera mai ambiguità
- Infatti, se B e C
  - sono disgiunti allora la loro combinazione fornisce una scelta senza ambiguità
  - non sono disgiunti, il comportamento prosegue ancora senza ambiguità perché è lo stesso in entrambi i casi

Scelta non deterministica

$P \sqcap Q$

Si comporta come P oppure come Q, indipendentemente dalla volontà dell'utente

- La scelta è determinata dalle azioni interne
  - Si può supporre che esista più di una azione interna $\tau_1, \tau_2, \ldots, \tau_n$

Confronto tra operatori di scelta

- L'operatore □ rappresenta una scelta compiuta dall'ambiente che interagisce con il processo in run time
- L'operatore ⊓ rappresenta una scelta compiuta dall'implementazione del processo

Operatori di I/O (1)

- Gli operatori di scelta permettono di caratterizzare le operazioni di input e output
  - l'input è espresso da ?
  - l'output è espresso da !

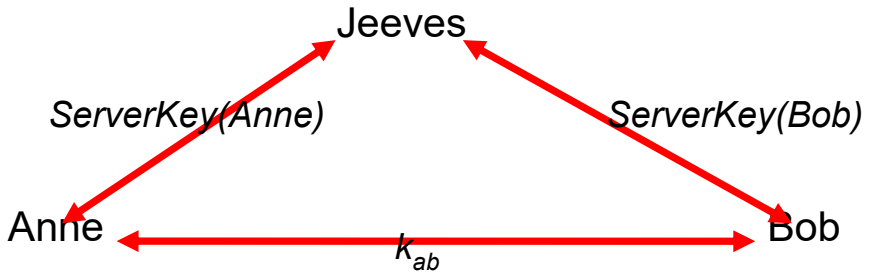
Operatori di I/O (2)

- L'operatore ? permette di indicare che l'ambiente fornisce elementi di input sotto forma di eventi che intervengono nell'esecuzione del processo
  - in?x ->P indica che l'evento x, che assume valori nell'insieme definito da in, è posto in prefixing rispetto a P
  - in rappresenta il channel rispetto a cui avviene l'input

Operatori di I/O (3)

- L'operatore ! permette di indicare che il processo produce elementi di output sotto forma di eventi che saranno poi consumati dall'ambiente
  - P->out!x -> Q indica che l'evento x, che assume valori nell'insieme definito da out, è un evento prodotto da P, ed è posto in prefixing rispetto a Q
  - out rappresenta il channel rispetto a cui avviene l'output

Schema di Riferimento Classico per Canale Sicuro (1)



Schema di Riferimento Classico per Canale Sicuro (2)

- Protocollo di sicurezza basato su un algoritmo simmetrico di crittografia
- I due agenti Anne (a) e Bob (b) devono comunicare attracerso un canale sicuro con l'aiuto di un server fidato (Jeeves)
- In generale, tutti i gli agenti registrati condividono con Jeeves chiavi di crittografia private e a lunga vita

Schema di Riferimento Classico per Canale Sicuro (3)

- Queste chiavi permettono ad ogni agente di comunicare in modo sicuro con Jeeves
- Non permettono la comunicazione tra Anne e Bob

Possibili comunicazioni sicure(1)

- Una soluzione che permetta a due agenti di comunicare tra di loro è data da:
  - Anne manda a Jeeves il msg destinato a Bob, codificato con il codice ServerKey(Anne)
  - Jeeves decritta il msg da Anne, lo ricodifica secondo ServerKey(Bob)
  - Jeeves invia il ms a Bob
- La comunicazione è sicura ma è troppo onerosa: Jeeves è un collo di bottiglia

Possibili comunicazioni sicure(2)

- Soluzione: Jeeves fornisce chiavi a lunga vita per ogni possibile coppia di agenti comunicanti
  - se il numero N di agenti è elevato servono N^2 chiavi
  - in genere la dimensione della rete potrebbe essere molto dinamica

Possibili comunicazioni sicure(3)

- Soluzione: richiedere l'intervento di Jeeves solo quando necessario per fornire una chiave che permetta la comunicazione sicura tra Anne e Bob

Un Protocollo Classico (1)

Msg1 *a->J* : $a.b.n_a$

Msg2 *J->a* : $\{n_a.b.k_{ab}.\{k_{ab}.a\}_{ServerKey(b)}\}_{ServerKey(a)}$

Msg3 *a->b* : $\{k_{ab}.a\}_{ServerKey(b)}$

Msg4 *b->a* : $\{n_b\}_{k_{ab}}$

Msg5 *a->b* : $\{n_b\text{-}1\}_{k_{ab}}$

Un Protocollo Classico (2)

- Nella forma

  MsgN x -> y : data

  – N indica lo step del protocollo
  – x l'agente mittente del msg
  – y il ricevente
  – data il contenuto in chiaro
  – $\{data\}_k$ il contenuto criptato con la chiave k
- Il contenuto dei messaggi è costituito dalla concatenazione di varie parti
  – Il simbolo . indica la concatenazione

Protocollo: step1

- Nel msg1 a comunica a J di voler comunicare con b (a.b)
  – n_a è un nonce, cioè un messaggio fittizio, creato da a
  – È un messaggio "in chiaro"
- Nel msg2 J
  – Crea la chiave di codifica k_{ab} che dovrà essere usata per le comunicazioni tra a e b
  – fornisce ad a la chiave

Protocollo: step2

- J crea la chiave di codifica k_{ab} che si dovrà usare per le comunicazioni tra a e b
- Il messaggio spedito da J è criptato secondo la chiave stabilita per le comunicazione tra a e J ($\{\}_{ServerKey(a)}$) e contiene:
  – Il nonce inviato da a allo step 1 e il nome di b, allo scopo di dare conferma alla richiesta del msg1
  – La chiave k_{ab}
  – Un msg ($\{k_{ab}.a\}_{ServerKey(b)}$) che a NON è in grado di capire in quanto criptato secondo la chiave stabilita per le comunicazione tra b e J, ma che a dovrà girare a b

Protocollo: step3

- a gira a b la parte del messaggio ricevuto da J, che non è stata decriptata ($\{k_{ab}.a\}_{ServerKey(b)}$)
- Tale messaggio ha lo scopo di segnalare a b che la comunicazione susseguente con a sarà sicura e autenticata da J
- b riceve il messaggio e decriptandolo con la chiave ServerKey(b) è in grado di conoscere la chiave k_{ab}

Protocollo: step4

- b crea un nonce e lo spedisce ad a codificato con la chiave k_{ab} allo scopo di indicare la disponibilità ad iniziare la comunicazione

Protocollo: step5

- a
  - estrae dal messaggio ricevuto da b il nonce
  - lo modifica secondo un criterio standard (tipicamente sottraendo 1)
  - invia a b il valore del nonce modificato
- Quando riceve il messaggio verifica che il valore del nonce ricevuto sia effettivamente quanto si aspetta e in tal caso la comunicazione criptata secondo k_{ab} può avere inizio

Protocollo Yahalom

Msg1 *a->b* : $a.n_a$

Msg2 *b->J* : $b.\{a.n_a.n_b\}_{ServerKey(b)}$

Msg3 *J->a* : $\{b.k_{ab}.n_a.n_b\}_{ServerKey(a)}.\{a.k_{ab}\}_{ServerKey(b)}$

Msg4 *a->b* : $\{a.k_{ab}\}_{ServerKey(b)}.\{n_b\}_{k_{ab}}$

Premessa (1)

- Due agenti coinvolti nella comunicazione (a e b) e uno di supporto (J)
- Tra i due agenti a e b
  - uno è l'initiator (poniamo a)
  - uno è il responder (b)
- Si vuole modellizzare con CSP i tre ruoli

Premessa (2)

- Ogni agente ha due domini di comunicazione, verso
  - l'altro agente
  - il proprio utente
- Assumiamo per semplicità che ogni processo sia provvisto dei due canali send e receive per tutte le comunicazioni con gli altri nodi
  - L'input assume la forma: receive.a.b.m
  - L'output assume la forma: send.a.b.m

Premessa (3)

- Tutti i messaggi del protocollo sono sottoposti a send/receive
- Le minacce alla sicurezza della comunicazione possono provenire da tutte le direzioni

Accettazione messaggi

- Necessità di vincolare un processo ad accettare solo i messaggi che hanno forma che il processo sia in grado di comprendere
  - scelta sui messaggi accettabili
  - se il messaggio ricevuto non è accettabile, deve essere eseguito un processo AbortRun

Initiator (1)

Initiator (a, n_a) =

env?b:Agent -> send.a.b.a.n_a ->

□

$k_{ab} \in$ Key
$n_b \in$ Nonce
$m \in T$

(receive.J.a.$\{b.k_{ab}.n_a.n_b\}_{ServerKey(a)}$.m ->
send.a.b.m.$\{n_b\}_{k_{ab}}$ -> Session(a,b,k_{ab},n_a,n_b))

dove T è l'insieme degli oggetti che il nodo può accettare

Initiator (2)

- Per i nostri scopi non è necessario dettagliare ulteriormente lo stato Session
  - Possiamo assumere che sia lo stato della sessione di comunicazione in cui avvengono gli scambi di informazione tra i comunicanti
- La chiave ServerKey(a) è la chiave che a condivide con J
- La comunicazione iniziale env?b:Agent è la richiesta che l'ambiente di a invia ad a per cominciare la comunicazione con b

Initiator (3)

- Il pacchetto che a riceve dal server e passa a b secondo il protocollo originale è

$$\{a.k_{ab}\}_{ServerKey(b)}$$

- Nella modellizzazione con CSP diventa semplicemente l'input m
  - a non svolge alcuna operazione su di esso, solo verifica che appartenga all'insieme delle azioni accettabili

Responder (1)

- Responder (b, n_b) =

$$\Box_{\substack{k_{ab} \in Key \\ n_a \in Nonce \\ a \in Agent}} \Big(\begin{array}{l} receive.a.b.a.n_a \rightarrow \\ send.b.J.b.\{a.n_a.n_b\}_{ServerKey(b)} \rightarrow \\ receive.a.b.\{a.k_{ab}\}_{ServerKey(b)}.\{n_b\}_{k_{ab}} \rightarrow \\ Session(b,a,k_{ab},n_a,n_b) \end{array} \Big)$$

dove Agent è l'insieme degli agenti con cui b può comunicare

Responder (2)

- Il protocollo è attivato dalla ricezione di un messaggio da parte di a
  - Non più da parte dell'ambiente

Server

- Serv (J, k_{ab}) =

$$\square_{\substack{n_a, n_b \in \text{Nonce} \\ a, b \in \text{Agent}}} \Big(\text{receive.b.J.b.}\{a.n_a.n_b\}_{\text{ServerKey(b)}} \text{ -> } \text{send.J.a.}\{b.k_{ab}.n_a.n_b\}_{\text{ServerKey(a)}} \; \{a.k_{ab}\}_{\text{ServerKey(b)}} \text{ -> Server (J,ks)} \Big)$$