

Security protocols: why?

SKIP FINO A SLIDE 21

- There are many cryptographic primitives
- How can we use them to secure applications?
- It is a difficult task, even if cryptography is “perfect”

Security protocols

- A security protocol is a distributed algorithm. It consists of a set of rules (conventions) that determine the **exchange of messages** between two or more principals.
- Since principals communicate over channels an **untrusted** network, security protocols ensure that communication is not abused

Security protocols (cont.)

Security protocols are three-line programs that people still manage to get wrong.

Roger M. Needham

We focus in particular on **authentication protocols**

Problem: how to establish a virtual trusted channel:

- negotiate parameters of channel
- ensure that channel is still trusted

Security protocols (cont.)

Main characters are

- generic principals (aka agents, parties...): A (Alice), B (Bob)
- a trusted server: S (Sam)
- the **Dolev-Yao** attacker: C (Charlie) or M (Mallory - Malicious) or E (Eve - Evil) (it can eavesdrop, replay, encrypt, decrypt, generate and inject messages.)

A protocol involves a sequence of message exchanges of the form:

$$A \rightarrow B : Msg$$

meaning that a principal A sends the message Msg to principal B
 $C(A)$ stands for C acting as A

Security protocols (cont.)

- Is **cryptography** enough?
- Attacks can be successful even without attacking the crypto-keys.

Wolf-in-the-middle attack: the wolf does not have the key

1. Little Red Riding Hood \rightarrow Grandma

The wolf does not need the key:
grandma opens the door



1' **Wolf**(Little Red Riding Hood) \rightarrow Grandma

2. Grandma \rightarrow Little Red Riding Hood

2'. **Wolf**(Grandma) \rightarrow Little Red Riding Hood

Little Red Riding Hood believes grandma
opens the door



Bank transfer protocol

Example (Naive version)

Alice \rightarrow *Bob@Bank* : Transfer 100 euros to account X
Bob@Bank \rightarrow *Alice* : Tranfer just carried out

- How does Bob know that he is really speaking with Alice?
- How does Bob know Alice just said it?
- Bad guys like **Charlie** can be around

Bank transfer protocol (cont.)

Example (Easy attack)

$Alice \rightarrow \text{Charlie}(Bob@Bank) : \text{Transfer 100 euros to account X}$
 $\text{Charlie}(Bob@Bank) \rightarrow Bob@Bank : \text{Transfer 200 euros to account X}$
 $Bob@Bank \rightarrow Alice : \text{Transfer just carried out}$

- **Charlie** can intercept and alter the message of Alice
- $C(B)$ stands for C pretending to be B .

Bank transfer protocol (cont.)

Example

Cryptographic protection

$Alice \rightarrow Charlie(Bob@Bank) : \{\text{Transfer 100 euros to account } X\}_K$
...

- The attacker can intercept the message of Alice, but cannot alter it

Bank transfer protocol (cont.)

Example (Cryptography may not be not enough: reflection attack)

$Alice \rightarrow Charlie(Bob@Bank) : \{\text{Transfer 100 euros to account } X\}_K$
 $Charlie(Bob@Bank) \rightarrow Alice : \{\text{Transfer 100 euros to account } X\}_K$

- The attacker can intercept the message of Alice, and just use it again
- Alice does not realize the received message is the one she generated

Bank transfer protocol (cont.)

Example (Possible fix)

$Alice \rightarrow Bob@Bank : \{I \text{ am } Alice, \text{ Transfer 100 euros to account } X\}_K$
 $Bob@Bank \rightarrow Alice : \text{Transfer just carried out}$

Bank transfer protocol (cont.)

Example (Replay attack)

Alice \rightarrow *Bob@Bank* : $\{I \text{ am Alice, Transfer 100 euros to account } X\}_K$

Charlie \rightarrow *Bob@Bank* : $\{I \text{ am Alice, Transfer 100 euros to account } X\}_K$

- The attacker can intercept the message of Alice, and just use it again
- Bob does not realize the received message is an old one: he has no way to verify the freshness of the message

Bank transfer protocol (cont.)

Example (Fixed protocol)

Bob@Bank \rightarrow *Alice* : N_B

Alice \rightarrow *Bob@Bank* : {I am Alice, Transfer 100 euros to account X, N_B } $_{\kappa}$

...

- Use a nonce, a randomly generated number used only once.
- Principals do not see each other: their trust is based on the presence of expected and known terms in received messages.
- Bob can verify that the message is an answer to his request

This protocol is secure. It guarantees the secrecy and authenticity of the message

Security protocols (cont.)

- Is **cryptography** enough?
- Attacks can be successful even without attacking the crypto-keys.
- Cryptography translates a communication security problem into a **key management problem**; therefore, it can enhance security but it is not a substitute for security

An authentication protocol

- Security protocol: list of messages exchanged by principals include recurring terms, like *rhymes* in poem verses

Example (Needham-Schroeder Public key protocol)

1. $A \rightarrow B : \{N_A, A\}_{pk(K_B)}$
2. $B \rightarrow A : \{N_A, N_B\}_{pk(K_A)}$
3. $A \rightarrow B : \{N_B\}_{pk(K_B)}$

- Security goal: mutual authentication of A and B ,
- Principals do not see each other: their trust is based on the presence of expected and known terms in received messages.
- Exchange of messages: **information** + **confirmation**



Man-in-the-middle attack

- Attackers can forge messages that are accepted by legitimate parties, breaking the intended rhymes and leading to attacks.

Example (Lowe Attack)

1. $A \rightarrow M : \{N_A, A\}_{pk(K_M)}$
1'. $M_A \rightarrow B : \{N_A, A\}_{pk(K_B)}$
2'. $B \rightarrow M_A : \{N_A, N_B\}_{pk(K_A)}$
2. $M \rightarrow A : \{N_A, N_B\}_{pk(K_A)}$
3. $A \rightarrow M : \{N_B\}_{pk(K_M)}$
3'. $M_A \rightarrow B : \{N_B\}_{pk(K_B)}$

where M_X stands for M pretending to be X .

B believes to talk with A , while it is talking with M : B cannot authenticate A .

Fixed protocol

- **Confirmation** part must be enriched: msg 2) does not say who is the sender

Example

1. $A \rightarrow B : \{N_A, A\}_{pk(K_B)}$
2. $B \rightarrow A : \{N_A, N_B, B\}_{pk(K_A)}$
3. $A \rightarrow B : \{N_B\}_{pk(K_B)}$

- the attack is no longer possible

...

2'. $B \rightarrow M_A : \{N_A, N_B, B\}_{pk(K_A)}$

2. $M \rightarrow A : \{N_A, N_B, B\}_{pk(K_A)}$

because A is waiting for $\{N_A, N_B, M\}_{pk(K_A)}$



Bibliography

- John Clark and Jeremy Jacob: A survey of authentication protocol literature, 1997.
http://www.cs.york.ac.uk/~jac/PublishedPapers/reviewV1_1997.pdf
- Paul Syverson. A Taxonomy of Replay Attacks. CSFW 1994.

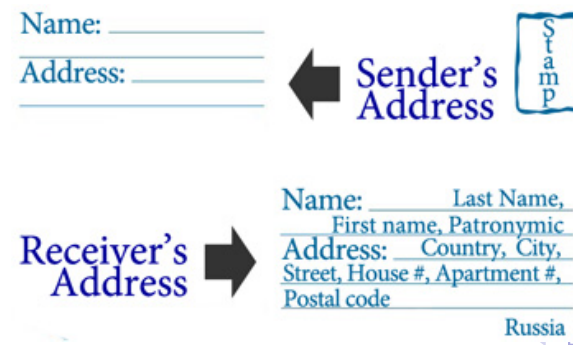
Formal techniques can help

Protocols can be specified by using **process calculi**

- **Process calculi** are mathematically rigorous languages with well defined semantics, for formally modelling concurrent and distributed systems
- Protocol agents are modelled as models of processes
- Protocols evolution is described in terms of transitions systems, graphs whose states are process calculi terms
- BUT
- **Transition systems** are usually **huge** and their exploration can be computationally demanding
- Static techniques (CFA, TS, AI) provide **approximate answers** just looking at the system description.

Setting the scene

- **Control Flow Analysis** (CFA) have been applied to cryptographic protocols to detect possible flaws
- CFA soundly over-approximates the behaviour of protocols described in the process algebra **LYSA**; in particular, it tracks **message flow** for protocol and attacker
- The CFA addresses **message authenticity**, in the presence of a Dolev-Yao attacker: it verifies that a message encrypted by principal A (**origin**) and intended for B (**destination**) does indeed come from A and reaches B only



The Nature of Approximation

Block 1

If no violation of the message authentication property is detected, then no violation will ever arise at run time

Block 2

The existence of violations at static time does not necessarily imply their existence at run time

- BUT, the existence of static violations is a **warning bell** and should be further investigated



Approach

- Define a protocol (using a process calculus)
- Define a Dolev-Yao style attacker
- Track message flow for protocol and attacker using **control flow analysis**
- If messages end up in a wrong place then there may be a problem
 - Attacker can alter message flow arbitrarily
 - Focus on encryption and decryption

ora l'approccio adesso è definire un protocollo usando un'algebra di processo che vedremo, definire l'attaccante che è il Dolev-Yao, quindi questo abbiamo già definito e poi vedere come posso tracciare il flusso dei messaggi tenendo conto anche della presenza dell'attaccante usando un'analisi di tipo controflow, come abbiamo visto fino a ora. In particolare vedremo che la proprietà che osserviamo è quella di tracciare i messaggi nel loro percorso, quindi io vedrò in particolare non messaggi in chiaro, ma messaggi crittografati andrò a vedere se i messaggi cifrati in un processo e che devono essere decifrati in un'altro, seguono il percorso previsto dal protocollo, quindi il focus dell'attenzione è proprio su ciò che viene scambiato non in chiaro che può essere sempre alterabile dal nemico, dall'attaccante, ma sui messaggi cifrati che dovrebbero essere, se non altro per questioni di chiave, essere letti da giusto, da chi li deve leggere.

Making Narrations Precise

- Typically, protocols are described by narrations and a textual description **but details may be imprecise**
- We make systematic translations of the protocol narrations into a process calculus called LYSa
- LYSa is inspired by the Spi-calculus but processes:
 - communicate through a global network
 - match values on input and decryption
 - use symmetric key cryptography

Il tipo di definizione dei protocolli, ora i protocolli avete visto che spessissimo vengono dati nella notazione informale, alic sbob, in cui sostanzialmente si dice qual è l'ordine dei messaggi e quale qual è il sender e receiver di ciascun messaggio in ogni step del protocollo e qual è il contenuto. Non si fa cenno se non nelle descrizioni informali che seguono alle azioni che invece dovrebbero fare in ricezione i riceventi e cosa succede nei casi in cui le cose vadano storte. E' stato visto negli anni e nei decenni che spessissimo questa mancanza di precisione, di formalità, di essere espliciti nel descrivere i protocolli spesso è stata motivo di attacchi, proprio perché le ipotesi implicite hanno lasciato aperto qualche fessura per attaccare, quindi vedremo che utilizzeremo una modalità per rendere più precisa la narrazione dei protocolli e questo ci servirà per tradurli sostanzialmente e per modellarli con una nuova analisi di processo che viene chiamata Lysa e che è ispirata allo spicalculus ma se ne distanzia per qualche piccola caratteristica. Allora intanto si abolisce il concetto di canale perché si suppone che la comunicazione avvenga attraverso una rete globale e che quindi non ci sia più bisogno di specificare il canale perché sostanzialmente tutto ciò che passa sui canali e per natura sua in chiaro naturalmente quando passo i messaggi cifrati li proteggerò anche se passano in chiaro e poi l'altra caratteristica, ora qui in questo caso usiamo la chiave simmetrica, ma in realtà si può estendere anche all'uso di cifrari asimmetrici, l'altra caratteristica proprio sintattica, è che per un motivo che poi vi sarà chiaro più avanti e in questa algebra di processi l'input e la decryption incorporano il pattern matching quindi si fa in un solo colpo, input e pattern matching oppure decryption e pattern matching ora vi faccio vedere come. così

LySa Syntax

Expressions

$E ::= n$ name ($n \in \mathcal{N}$)
 x variable ($x \in \mathcal{X}$)
 ~~$\{E_1, \dots, E_k\}_{E_0}$~~ encryption

Processes

$P ::= \langle E_1, \dots, E_k \rangle . P$ output
 $(E_1, \dots, E_j; x_{j+1}, \dots, x_k) . P$ input (with matching)
 decrypt E as $\{E_1, \dots, E_j; x_{j+1}, \dots, x_k\}_{E_0}$ in P decryption (with matching)
 $P_1 \mid P_2$ parallel composition
 $(\nu n)P$ introduce new name n
 $!P$ replication
 0 terminated process

LYSA features: decryptions on the fly

- No channels: communication is on a single global network.
- Decryptions are embedded inside inputs and performed on the fly when receiving the corresponding outputs. The output

$$\langle A, N_A, \{B, K_{AB}\}_{K_{AS}} \rangle$$

matches the input, including the embedded encryption

$$(A, x_N, \{B, x_K\}_{K_{AS}})$$

and the variables x_N and x_K are bound to N_A and to K_{AB}

quindi abbiamo detto che non ci sono canali e che le decryption sono incorporate dentro gli input e fatte al volo quando si ricevono i corrispondenti output quindi, in particolare, potete immaginare che chi fa l' output potrebbe fare un'operazione di questo genere, quindi manda il nome in chiaro, il nonce in chiaro, e invece criptato manda la chiave di sessione e il il processo con il quale vuole parlare e questa comunicazione può corrispondere a questo input in cui, come vedete chi fa l'input si aspetta che la prima componente sia una A, la seconda non la conosce, quindi serve il nonce, quindi lo lega e poi riceve una encryption che lui si aspetta essere fatta da due componenti e criptata con la chiave long term tra A e S e all'interno si aspetta che ci sia la nuova chiave che non conosce, accompagnata dal nome del processo con cui dovrà comunicare. Ora la caratteristica invece che rispetta la proprietà di questa analisi è come vi dicevo, la possibilità di tracciare i messaggi di tipo cifrato,

LYSA features: message authentication

To track message origin and destination, encryption terms and pattern terms are labelled:

$$\{E_1, \dots, E_k\}_{E_0}^{\ell} [\text{dest } \mathcal{L}] \quad \text{and as} \quad \{M_1, \dots, M_k\}_{E'_0}^{\ell'} [\text{orig } \mathcal{L}']$$

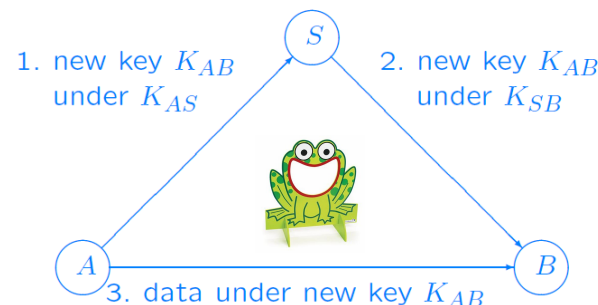
quindi quello che vorrò fare è aggiungere e questa è una aggiunta che faccio alla sintassi, delle etichette che mi dicono dove è stata creata qualcosa e a chi è destinata. Quindi in particolare ogni encryption la etichetterò con un'etichetta azzurra in alto ℓ , che mi dirà è stata creata dal processo etichettato come ℓ e pensatela come una dichiarazione, l'ha creata ℓ ed è destinata a tutti i processi la cui etichetta è dentro \mathcal{L} , in corrispondenza avrò la decryption che potrà essere sempre a programma idealmente associata a chi la fa ℓ' , e avere associato le etichette dei processi da cui quella encryption dovrebbe provenire quindi sostanzialmente io sto cominciando a costruire un'analisi con cui cercherò di vedere come tracciare encryption e decryption proprio dal punto di vista di chi le fa per chi. La mia osservazione sarà quella di capire se un encryption che il protocollo mi dice la deve fare l'initiator e deve arrivare al responder del protocollo effettivamente l'esecuzione anche in presenza di un nemico, abbia proprio questo comportamento, che quindi l'encryption che fa a per b riporti qua l'etichetta $\ell = a$ e tra i destinatari deve esserci b, viceversa, questo si incrocia con il controllo che faccio a valle, in cui chi fa la decryption in questo caso b, quindi ℓ' deve controllare che le l'encryption provenga da a.

Encoding a Protocol in LySA

Example

A key exchange inspired protocol by the Wide Mouthed Frog

1. $A \rightarrow S : A, B, \{K_{AB}\}_{K_{AS}}$
2. $S \rightarrow B : A, \{K_{AB}\}_{K_{BS}}$
3. $A \rightarrow B : \{m_1, \dots, m_k\}_{K_{AB}}$



Questa è la famosa rana dalla bocca larga supponiamo di vederla in questa forma, per cui A spedisce ad S in chiaro "sono A voglio parlare con B, questa è la chiave di sessione, te la mando con la chiave long term che condividiamo" S raccoglie il messaggio e praticamente triangola con il processo B per cui gli dice in chiaro, A vuole parlare, ti mando con la chiave long term la nuova chiave di sessione che mi ha mandato A e infine A spedisce una tupla qualsiasi codificata con la nuova chiave di sessione K_{AB} .

Encoding a protocol in LYSa

Example (Wide Mouthed Frog)

1. $A \rightarrow S : A, B, \{K_{AB}\}_{K_{AS}}$
2. $S \rightarrow B : A, \{K_{AB}\}_{K_{BS}}$
3. $A \rightarrow B : \{m_1, \dots, m_k\}_{K_{AB}}$

$$(\nu K_{AB}) \langle A, B, \{K_{AB}\}_{K_{AS}} \rangle.$$

Come uso la nuova algebra di processo per rappresentare? Un po' come uno si aspetta e abbiamo già visto nello spi calculus, quindi in particolare ogni volta che c'è una chiave la vedrò sotto forma di un nome introdotto con una restrizione, non abbiamo bisogno del canale C_{AB} e quindi ho direttamente la tupla di output che chiaramente deve contenere le parti che la narrazione informale mi dice cioè A, B e la codifica della chiave di sessione con K_{AS} . Ora però per ovviare al fatto che non ho più un canale che mi dice è il canale che unisce A a S , la mia modellazione deve includere anche, e sono in chiaro, ma servono solo per ricordarsi la direzione del messaggio, anche due campi che mi dicono chi è il sender e il receiver di questo messaggio in particolare, quindi questa è la parte che uno si può aspettare più questa piccola precisazione del sender e del receiver, ma la modellazione fa vedere cosa succede naturalmente anche in ricezione,

Encoding a protocol in L_YS_A

Example (Wide Mouthed Frog)

1. $A \rightarrow S : A, B, \{K_{AB}\}_{K_{AS}}$
2. $S \rightarrow B : A, \{K_{AB}\}_{K_{BS}}$
3. $A \rightarrow B : \{m_1, \dots, m_k\}_{K_{AB}}$

$$(\nu K_{AB}) \langle A, S, A, B, \{K_{AB}\}_{K_{AS}} \rangle.$$

|

Encoding a protocol in LYSa

Example (Wide Mouthed Frog)

1. $A \rightarrow S : A, B, \{K_{AB}\}_{K_{AS}}$
2. $S \rightarrow B : A, \{K_{AB}\}_{K_{BS}}$
3. $A \rightarrow B : \{m_1, \dots, m_k\}_{K_{AB}}$

$$(\nu K_{AB}) \langle A, S, A, B, \{K_{AB}\}_{K_{AS}} \rangle.$$

|

$$(A, S, A; x_B, x)$$

ora in ricezione come abbiamo visto chi sta di là, cioè il server, che è la trusted entity, andrà a fare l'input. Ora siccome abbiamo detto che l'input si fa con pattern matching, qual è l'informazione che il server deve sapere già? Naturalmente che sta leggendo dal canale che lo unisce ad A e quindi si aspetta che le due prime componenti siano proprio A e S, ma si aspetta anche che visto che il messaggio viene nel protocollo fatto così e viene da A, che anche la terza componente sia una A. Quindi sostanzialmente lui sta ricevendo da A un messaggio di richiesta di triangolazione per la chiave di sessione, quindi l'unica cosa che non sa e chi è il destinatario di questa chiave di sessione, quindi serve averlo legato sulla variabile x_B e soprattutto qual è l'encryption che dovrà decriptare per trovare la chiave da mandare.

Encoding a protocol in L_YS_A

Example (Wide Mouthed Frog)

1. $A \rightarrow S : A, B, \{K_{AB}\}_{K_{AS}}$
2. $S \rightarrow B : A, \{K_{AB}\}_{K_{BS}}$
3. $A \rightarrow B : \{m_1, \dots, m_k\}_{K_{AB}}$

$(\nu K_{AB}) \langle A, S, A, B, \{K_{AB}\}_{K_{AS}} \rangle.$

|

$(A, S, A; x_B, x). \text{decrypt } x \text{ as } \{; x^K\}_{K_{AS}} \text{ in } \dots$

Adding annotations

Example (Wide Mouthed Frog)

1. $A \rightarrow S : A, B, \{K_{AB}\}_{K_{AS}}$
2. $S \rightarrow B : A, \{K_{AB}\}_{K_{BS}}$
3. $A \rightarrow B : \{m_1, \dots, m_k\}_{K_{AB}}$

$(\nu K_{AB}) \langle A, S, A, B, \{K_{AB}\}_{K_{AS}}^A \rangle.$

|

$(A, S, A; x_B, x). \text{decrypt } x \text{ as } \{; x^K\}_{K_{AS}}^S \text{ in } \dots$

L'altro discorso da fare è quello legato alla proprietà, quindi abbiamo visto che io voglio controllare che encryption corrispondenti seguano l'idea del protocollo, l'idea del protocollo appunto, nel caso della rana dalla bocca larga è che l'encryption che viene fatta nel primo caso sia creata da A e abbia come destinatario S, nel secondo step l'encryption deve essere creata da S e letta e decifrata da B e infine, nell'ultimo step, abbiamo che l'encryption deve essere creata da A e decifrata da B, quindi se lo vedete sul lato dell'input S si aspetta qualcosa da decifrare, qualcosa che gli proviene da A, nel secondo B si aspetta qualcosa che deve essere stato criptato da S e infine, nel terzo step, B si aspetta un encryption creata da A. Quindi nella nostra modellazione dobbiamo includere anche le etichette, quindi includiamo le etichette intanto del processo che ha creato l'encryption e poi aggiungiamo non solo chi l'ha creata, ma a chi è destinata e viceversa, nella decryption solo chi decifra, ma anche il controllo che l'origine sia quella che uno si aspetta. Come mai tracciamo questa proprietà? La risposta è che abbiamo visto che tracciare questo movimento, cioè questi spostamenti tra l'origine e la destinazione di un encryption è proprio una delle cose che è utile perché spesso l'effetto collaterale di un attacco e specialmente di un man in the middle sostanzialmente va ad alterare proprio questo flusso, quindi vediamo spessissimo in molti casi che l'intervento attivo di un attaccante di tipo Dolev-Yao altera questo flusso, quindi può deviare il percorso di un encryption dalla sua origine alla destinazione o includendo lui stesso delle encryption o deviando su altri processi e quindi se volete violazioni di questo tipo in qualche maniera sono l'effetto collaterale di attacchi che sono più complessi di questo. A questo punto, quello che ci serve è capire come fare l'analisi di tipo cfa di questo tipo, di algebra di processo e in particolare anche di questa proprietà.

Example (Wide Mouthed Frog)

1. $A \rightarrow S : A, B, \{K_{AB}\}_{K_{AS}}$
2. $S \rightarrow B : A, \{K_{AB}\}_{K_{BS}}$
3. $A \rightarrow B : \{m_1, \dots, m_k\}_{K_{AB}}$

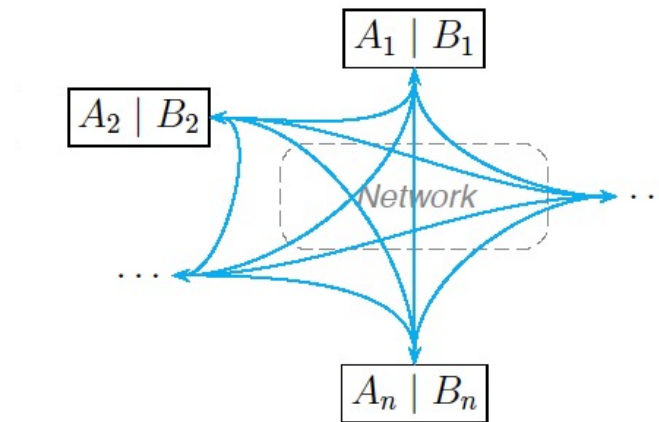
$(\nu K_{AB}) \langle A, S, A, B, \{K_{AB}\}_{K_{AS}}^A [\text{dest } S] \rangle.$

$(A, S, A; x_B, x). \text{decrypt } x \text{ as } \{; x^K\}_{K_{AS}}^S [\text{orig } A] \text{ in } \dots$

TORNARE INDIETRO A SLIDE 24

Protocol encoding: multiple instances

Note that protocols are played by several principals able to play both the **initiator role** A_i and the **responder** one B_i , plus a **server** S , if any



Protocol encoding: multiple instances (2)

There are **multiple instances** of agents playing the roles of initiators, responders and servers. They run in parallel.

dovrà aggiungere a valle degli input anche la decryption, nella decryption vedete che non c'è che una sola componente che non conosce, perché la nuova chiave di sessione che ha appena creato A. Sintatticamente le cose che devono essere riconosciute, quelle che non si conoscono, sono separate da un punto e virgola come vedete a sinistra del punto e virgola non c'è niente quindi non c'è nulla nel messaggio che lui deve riconoscere, quindi leggerà qualsiasi valore di chiave venga trovato all'interno dell' encryption, l'unica cosa che deve matchare naturalmente è la chiave, quindi lui riesce a decrittare solo se gli arriva in questo momento senza perturbazioni del nemico qualcosa criptato con la sua chiave long term. Ora naturalmente io vi sto facendo vedere che cosa succede laddove esistano solo in questo caso tre partecipanti a un protocollo, ora dovete immaginare che a cose normali ci sono molte istanze di agenti che giocano anche ruoli diversi dello stesso protocollo e magari anche più protocolli in parallelo, quindi naturalmente è un pochino più complicata di così la vera modellazione.

```

0.  $(\nu_{i=1}^n K_i^A)(\nu_{j=1}^n K_j^B)$ 
1.  $\prod_{i=1}^n \prod_{j \neq i}^n !(\nu K_{ij})$ 
    $\langle I_i, A, I_{-1}, S, I_i, A, \{I_j, B, K_{ij}\}_{K_i^A}^{A_i} [\text{dest } S] \rangle$ 
3.  $(\nu m_{1ij}) \cdots (\nu m_{kij})$ 
    $\langle I_i, A, I_j, B, \{m_{1ij}, \dots, m_{kij}\}_{K_{ij}^A}^{A_i} [\text{dest } B_j] \rangle$ 

2'.  $\prod_{j=1}^n ! (I_{-1}, S, I_j, B; y_j)$ 
2''.  $\prod_{i=1}^n \text{decrypt } y_j \text{ as } \{I_i, A; y_{ij}^K\}_{K_j^B}^{B_j} [\text{orig } S] \text{ in}$ 
3'.  $(I_i, A, I_j, B; z_{ij})$ 
3''.  $\text{decrypt } z_{ij} \text{ as } \{z_{ij}^{m_1}, \dots, z_{ij}^{m_k}\}_{y_{ij}^K}^{B_j} [\text{orig } A_i] \text{ in } 0$ 

1'.  $\prod_{i=0}^n ! (I_i, A, I_{-1}, S, I_i, A; x_i)$ 
1''.  $\prod_{j=0}^n \text{decrypt } x_i \text{ as } \{I_j, B; x_{ij}^K\}_{K_i^A}^{S_i} [\text{orig } A_i] \text{ in}$ 
2.  $\langle I_{-1}, S, I_j, B, \{I_i, A, x_{ij}^K\}_{K_j^B}^{S_i} [\text{dest } B_j] \rangle$ 
    
```

LYSA the analysis

Our *Control Flow analysis* computes an (over-)approximation to

- the messages on the networks: κ
- the values of the variables: ρ

For instance:

$$\begin{aligned} \langle A, S, A, B, \{K_{AB}\}_{K_A}^A [\text{dest } S] \rangle &\in \kappa \\ \{K_{AB}\}_{K_A}^A [\text{dest } S] &\in \rho(x) \\ K_{AB} &\in \rho(x^K) \end{aligned}$$

detto macroscopicamente, l'analisi che vedremo calcola di nuovo una sovrappassimazione dei messaggi che passano sulla rete, quindi c'è ancora una componente κ che ormai dovremmo riconoscere, calcola poi i valori che sono legati alle singole variabili, quindi per fare un esempio concreto io avrò che siccome questo è il primo messaggio del protocollo della rana, tutta la tupla apparterrà a κ , in particolare, siccome chi lo riceve deve legare la parte dell'encryption alla variabile x , allora il singolo pezzettino che riguarda l'encryption sarà legato alla x e infine nel processo di decryption la variabile da legare è stata x^K e quella sarà legata a K_{AB} .

LYSA: the analysis (cont.)


- the messages on the networks: κ
- the values of the variables: ρ
- the possible mismatches of origin and destination: ψ
E.g. if also $\langle A \rightarrow S, A, B, \{K\}_{K_A}^{A'} [\text{dest } S] \rangle \in \kappa$ then

$$(A', S) \in \psi$$

something encrypted at A' may be decrypted at S , where the expected origin was A

e più abbiamo detto, vogliamo tracciare questa proprietà di origine destinazione quando vado ad analizzare tutto vada anche a controllare che non ci sia mismatching tra origine e destinazione e quindi se questo succedesse e solo quando succede il mismatching vado a ricordarmelo in una nuova componente che finora non abbiamo visto, una componente di errore che mi dice attenzione la mia analisi con tutto il beneficio dell'inventario dato dal fatto che è una sovrapposizione, mi predice che è possibile che un encryption che era creato per un percorso origine destinazione di un certo tipo invece è erroneamente arrivata, quindi ha come origine A' e come destinazione S e questo invece non era previsto dal protocollo, quindi c'è qualcosa che non va potenzialmente.

Semantics

- Standard **reduction semantics** $P \rightarrow P'$
- The standard semantics ignores annotations
- We can also make a **reference monitor semantics** $P \rightarrow_{\text{RM}} P'$
- The reference monitor gets stuck when annotations are violated
- The reference monitor **aborts** the execution of P 

whenever $P \rightarrow^* Q \rightarrow Q'$

but $P \rightarrow_{\text{RM}}^* Q \not\rightarrow_{\text{RM}} Q'$