Protocol Verification Techniques - State Enumeration

Design and Verification of Security Protocols and Security Ceremonies

Programa de Pós-Graduacão em Ciências da Computacão
Dr. Jean Everson Martina

August-November 2016

Programa de Pós-Graduação em Ciência da Computação
UFSC

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Before we start!

Before we start!

To be able to talk about Lowe's attack using FDR and CSP, we need first to get the grips with the theory behind communicating sequential processes (CSP).

Communicating Sequential Processes

- Communicating Sequential Processes (CSP) is a formal language for describing patterns of interaction in concurrent systems;

Communicating Sequential Processes

- Communicating Sequential Processes (CSP) is a formal language for describing patterns of interaction in concurrent systems;
- It is a member of the family of mathematical theories of concurrency known as process algebras;

Communicating Sequential Processes

- Communicating Sequential Processes (CSP) is a formal language for describing patterns of interaction in concurrent systems;
- It is a member of the family of mathematical theories of concurrency known as process algebras;
- CSP was first described in a 1978 paper by Tony Hoare;

Communicating Sequential Processes

- Communicating Sequential Processes (CSP) is a formal language for describing patterns of interaction in concurrent systems;
- It is a member of the family of mathematical theories of concurrency known as process algebras;
- CSP was first described in a 1978 paper by Tony Hoare;
- CSP has been practically applied in industry as a tool for specifying and verifying the concurrent aspects of a variety of different systems.

Communicating Sequential Processes

- The theory of CSP itself is also still the subject of active research, including work to increase its range of practical applicability;

Communicating Sequential Processes

- The theory of CSP itself is also still the subject of active research, including work to increase its range of practical applicability;
- CSP allows the description of systems in terms of component processes that operate independently, and interact with each other solely through message-passing communication;

Communicating Sequential Processes

- The theory of CSP itself is also still the subject of active research, including work to increase its range of practical applicability;
- CSP allows the description of systems in terms of component processes that operate independently, and interact with each other solely through message-passing communication;
- The relationships between different processes, and the way each process communicates with its environment, are described using various process algebraic operators;

Communicating Sequential Processes

- The theory of CSP itself is also still the subject of active research, including work to increase its range of practical applicability;
- CSP allows the description of systems in terms of component processes that operate independently, and interact with each other solely through message-passing communication;
- The relationships between different processes, and the way each process communicates with its environment, are described using various process algebraic operators;
- Using this algebraic approach, quite complex process descriptions can be easily constructed from a few primitive elements.

CSP Primitives

- CSP provides two classes of primitives in its process algebra:

CSP Primitives

- CSP provides two classes of primitives in its process algebra:
- Events:

CSP Primitives

- CSP provides two classes of primitives in its process algebra:
- Events:
  - Events represent communications or interactions;

CSP Primitives

- CSP provides two classes of primitives in its process algebra:
- Events:
  - Events represent communications or interactions;
  - Events are assumed to be indivisible and instantaneous;

CSP Primitives

- CSP provides two classes of primitives in its process algebra:
- Events:
  - Events represent communications or interactions;
  - Events are assumed to be indivisible and instantaneous;
  - They may be atomic names, compound names, or input/output events;

CSP Primitives

- CSP provides two classes of primitives in its process algebra:
- Events:
  - Events represent communications or interactions;
  - Events are assumed to be indivisible and instantaneous;
  - They may be atomic names, compound names, or input/output events;
- Primitive processes:

CSP Primitives

- CSP provides two classes of primitives in its process algebra:
- Events:
  - Events represent communications or interactions;
  - Events are assumed to be indivisible and instantaneous;
  - They may be atomic names, compound names, or input/output events;
- Primitive processes:
  - Primitive processes represent fundamental behaviours;

CSP Primitives

- CSP provides two classes of primitives in its process algebra:
- Events:
  - Events represent communications or interactions;
  - Events are assumed to be indivisible and instantaneous;
  - They may be atomic names, compound names, or input/output events;
- Primitive processes:
  - Primitive processes represent fundamental behaviours;
  - STOP (the process that communicates nothing, also called deadlock);

CSP Primitives

- CSP provides two classes of primitives in its process algebra:
- Events:
  - Events represent communications or interactions;
  - Events are assumed to be indivisible and instantaneous;
  - They may be atomic names, compound names, or input/output events;
- Primitive processes:
  - Primitive processes represent fundamental behaviours;
  - STOP (the process that communicates nothing, also called deadlock);
  - SKIP (which represents successful termination).

CSP Algebraic Operators

$$
\begin{array}{lcl}
Proc & ::= & STOP \\
 & | & SKIP
\end{array}
$$

CSP Algebraic Operators

$$
\begin{array}{lcll}
Proc & ::= & STOP & \\
 & | & SKIP & \\
 & | & e \rightarrow Proc & \text{(prefixing)}
\end{array}
$$

CSP Algebraic Operators

$$
\begin{array}{lcll}
Proc & ::= & STOP & \\
 & | & SKIP & \\
 & | & e \rightarrow Proc & \text{(prefixing)} \\
 & | & Proc \; \Box \; Proc & \text{(external choice)}
\end{array}
$$

CSP Algebraic Operators

$$
\begin{array}{lcll}
Proc & ::= & STOP & \\
 & | & SKIP & \\
 & | & e \rightarrow Proc & \text{(prefixing)} \\
 & | & Proc \,\Box\, Proc & \text{(external choice)} \\
 & | & Proc \sqcap Proc & \text{(nondeterministic choice)}
\end{array}
$$

CSP Algebraic Operators

$$
\begin{array}{rcll}
Proc & ::= & STOP & \\
 & | & SKIP & \\
 & | & e \rightarrow Proc & \text{(prefixing)} \\
 & | & Proc \,\Box\, Proc & \text{(external choice)} \\
 & | & Proc \sqcap Proc & \text{(nondeterministic choice)} \\
 & | & Proc \,|||\, Proc & \text{(interleaving)}
\end{array}
$$

CSP Algebraic Operators

$$
\begin{array}{lcll}
\mathit{Proc} & ::= & \mathit{STOP} & \\
 & \mid & \mathit{SKIP} & \\
 & \mid & e \rightarrow \mathit{Proc} & \text{(prefixing)} \\
 & \mid & \mathit{Proc} \,\Box\, \mathit{Proc} & \text{(external choice)} \\
 & \mid & \mathit{Proc} \sqcap \mathit{Proc} & \text{(nondeterministic choice)} \\
 & \mid & \mathit{Proc} \,|||\, \mathit{Proc} & \text{(interleaving)} \\
 & \mid & \mathit{Proc} \,|[\{X\}]|\, \mathit{Proc} & \text{(interface parallel)}
\end{array}
$$

CSP Algebraic Operators

$$
\begin{array}{lcll}
Proc & ::= & STOP & \\
 & | & SKIP & \\
 & | & e \rightarrow Proc & \text{(prefixing)} \\
 & | & Proc \,\Box\, Proc & \text{(external choice)} \\
 & | & Proc \sqcap Proc & \text{(nondeterministic choice)} \\
 & | & Proc \,|||\, Proc & \text{(interleaving)} \\
 & | & Proc \,|[\{X\}]|\, Proc & \text{(interface parallel)} \\
 & | & Proc \setminus X & \text{(hiding)}
\end{array}
$$

CSP Algebraic Operators

$$
\begin{array}{lcll}
Proc & ::= & STOP & \\
 & | & SKIP & \\
 & | & e \rightarrow Proc & \text{(prefixing)} \\
 & | & Proc \Box Proc & \text{(external choice)} \\
 & | & Proc \sqcap Proc & \text{(nondeterministic choice)} \\
 & | & Proc \mid\mid\mid Proc & \text{(interleaving)} \\
 & | & Proc \mid[\{X\}]\mid Proc & \text{(interface parallel)} \\
 & | & Proc \setminus X & \text{(hiding)} \\
 & | & Proc ; Proc & \text{(sequential composition)}
\end{array}
$$

CSP Algebraic Operators

$$
\begin{array}{rcll}
\mathit{Proc} & ::= & \mathit{STOP} & \\
& | & \mathit{SKIP} & \\
& | & e \rightarrow \mathit{Proc} & \text{(prefixing)} \\
& | & \mathit{Proc} \,\Box\, \mathit{Proc} & \text{(external choice)} \\
& | & \mathit{Proc} \sqcap \mathit{Proc} & \text{(nondeterministic choice)} \\
& | & \mathit{Proc} \,|||\, \mathit{Proc} & \text{(interleaving)} \\
& | & \mathit{Proc} \,|[\{X\}]|\, \mathit{Proc} & \text{(interface parallel)} \\
& | & \mathit{Proc} \setminus X & \text{(hiding)} \\
& | & \mathit{Proc}; \mathit{Proc} & \text{(sequential composition)} \\
& | & \text{if } b \text{ then } \mathit{Proc} \text{ else } \mathit{Proc} & \text{(boolean conditional)}
\end{array}
$$

CSP Algebraic Operators

$$
\begin{array}{lcll}
\mathit{Proc} & ::= & \mathit{STOP} & \\
 & | & \mathit{SKIP} & \\
 & | & e \rightarrow \mathit{Proc} & \text{(prefixing)} \\
 & | & \mathit{Proc} \,\Box\, \mathit{Proc} & \text{(external choice)} \\
 & | & \mathit{Proc} \sqcap \mathit{Proc} & \text{(nondeterministic choice)} \\
 & | & \mathit{Proc} \,|||\, \mathit{Proc} & \text{(interleaving)} \\
 & | & \mathit{Proc} \,|[\{X\}]|\, \mathit{Proc} & \text{(interface parallel)} \\
 & | & \mathit{Proc} \setminus X & \text{(hiding)} \\
 & | & \mathit{Proc}; \mathit{Proc} & \text{(sequential composition)} \\
 & | & \text{if } b \text{ then } \mathit{Proc} \text{ else } \mathit{Proc} & \text{(boolean conditional)} \\
 & | & \mathit{Proc} \triangleright \mathit{Proc} & \text{(timeout)}
\end{array}
$$

CSP Algebraic Operators

$$
\begin{array}{lcll}
\mathit{Proc} & ::= & \mathit{STOP} & \\
 & | & \mathit{SKIP} & \\
 & | & e \rightarrow \mathit{Proc} & \text{(prefixing)} \\
 & | & \mathit{Proc} \,\Box\, \mathit{Proc} & \text{(external choice)} \\
 & | & \mathit{Proc} \sqcap \mathit{Proc} & \text{(nondeterministic choice)} \\
 & | & \mathit{Proc} \,|||\, \mathit{Proc} & \text{(interleaving)} \\
 & | & \mathit{Proc} \,|[\{X\}]|\, \mathit{Proc} & \text{(interface parallel)} \\
 & | & \mathit{Proc} \setminus X & \text{(hiding)} \\
 & | & \mathit{Proc}; \mathit{Proc} & \text{(sequential composition)} \\
 & | & \text{if } b \text{ then } \mathit{Proc} \text{ else } \mathit{Proc} & \text{(boolean conditional)} \\
 & | & \mathit{Proc} \rhd \mathit{Proc} & \text{(timeout)} \\
 & | & \mathit{Proc} \bigtriangleup \mathit{Proc} & \text{(interrupt)}
\end{array}
$$

CSP Example - Vending Machine

- One of the archetypal CSP examples is an abstract representation of a chocolate vending machine and its interactions with a person wishing to buy some chocolate;

CSP Example - Vending Machine

- One of the archetypal CSP examples is an abstract representation of a chocolate vending machine and its interactions with a person wishing to buy some chocolate;
- This vending machine might be able to carry out two different events, “coin” and “choc”;

CSP Example - Vending Machine

- One of the archetypal CSP examples is an abstract representation of a chocolate vending machine and its interactions with a person wishing to buy some chocolate;
- This vending machine might be able to carry out two different events, “coin” and “choc”;
- They which represent the insertion of payment and the delivery of a chocolate respectively.

CSP Example - Vending Machine

- A machine which demands payment before offering a chocolate can be written as:

CSP Example - Vending Machine

- A machine which demands payment before offering a chocolate can be written as:
  - $VendingMachine = coin \rightarrow choc \rightarrow STOP$

CSP Example - Vending Machine

- A machine which demands payment before offering a chocolate can be written as:
  - $VendingMachine = coin \rightarrow choc \rightarrow STOP$
- A person who might choose to use a coin or card to make payments could be modelled as:

CSP Example - Vending Machine

- A machine which demands payment before offering a chocolate can be written as:
  - $VendingMachine = coin \rightarrow choc \rightarrow STOP$
- A person who might choose to use a coin or card to make payments could be modelled as:
  - $Person = (coin \rightarrow STOP) \Box (card \rightarrow STOP))$

CSP Example - Vending Machine

- A machine which demands payment before offering a chocolate can be written as:
  - $VendingMachine = coin \rightarrow choc \rightarrow STOP$
- A person who might choose to use a coin or card to make payments could be modelled as:
  - $Person = (coin \rightarrow STOP)\square(card \rightarrow STOP))$
- These two processes can be put in parallel, so that they can interact with each other;

CSP Example - Vending Machine

- A machine which demands payment before offering a chocolate can be written as:
  - $VendingMachine = coin \rightarrow choc \rightarrow STOP$
- A person who might choose to use a coin or card to make payments could be modelled as:
  - $Person = (coin \rightarrow STOP) \Box (card \rightarrow STOP))$
- These two processes can be put in parallel, so that they can interact with each other;
- The behaviour of the composite process depends on the events that the two processes must synchronise on:

CSP Example - Vending Machine

- A machine which demands payment before offering a chocolate can be written as:
  - $VendingMachine = coin \rightarrow choc \rightarrow STOP$
- A person who might choose to use a coin or card to make payments could be modelled as:
  - $Person = (coin \rightarrow STOP) \Box (card \rightarrow STOP))$
- These two processes can be put in parallel, so that they can interact with each other;
- The behaviour of the composite process depends on the events that the two processes must synchronise on:
  - $VendingMachine |[\{coin, card\}]| Person \equiv$
    $coin \rightarrow choc \rightarrow STOP$

CSP Example - Vending Machine

- If synchronization was only required on “coin”, we would obtain:

CSP Example - Vending Machine

- If synchronization was only required on "coin", we would obtain:
  - $\mathit{VendingMachine} \,|[\{\mathit{coin}\}]|\, \mathit{Person} \equiv$
    $(\mathit{coin} \rightarrow \mathit{choc} \rightarrow \mathit{STOP}) \,\square\, (\mathit{card} \rightarrow \mathit{STOP})$

CSP Example - Vending Machine

- If synchronization was only required on "coin", we would obtain:
  - $VendingMachine \,|[\{coin\}]|\, Person \equiv (coin \rightarrow choc \rightarrow STOP) \,\square\, (card \rightarrow STOP)$
- If we abstract this latter composite process by hiding the "coin" and "card" events:

CSP Example - Vending Machine

- If synchronization was only required on "coin", we would obtain:
  - $VendingMachine \,|[\{coin\}]|\, Person \equiv$ $(coin \rightarrow choc \rightarrow STOP) \,\Box\, (card \rightarrow STOP)$
- If we abstract this latter composite process by hiding the "coin" and "card" events:
  - $((coin \rightarrow choc \rightarrow STOP) \,\Box\, (card \rightarrow STOP))$ $\setminus \{coin, card\}$

CSP Example - Vending Machine

- If synchronization was only required on “coin”, we would obtain:
  - $VendingMachine \,|[\{coin\}]|\, Person \equiv$ $(coin \rightarrow choc \rightarrow STOP) \,\square\, (card \rightarrow STOP)$
- If we abstract this latter composite process by hiding the “coin” and “card” events:
  - $((coin \rightarrow choc \rightarrow STOP) \,\square\, (card \rightarrow STOP))$ $\setminus \{coin, card\}$
- We get the non-deterministic process:

CSP Example - Vending Machine

- If synchronization was only required on "coin", we would obtain:
  - $\mathit{VendingMachine} \,|[\{\mathit{coin}\}]|\, \mathit{Person} \equiv (\mathit{coin} \rightarrow \mathit{choc} \rightarrow \mathit{STOP}) \,\square\, (\mathit{card} \rightarrow \mathit{STOP})$
- If we abstract this latter composite process by hiding the "coin" and "card" events:
  - $((\mathit{coin} \rightarrow \mathit{choc} \rightarrow \mathit{STOP}) \,\square\, (\mathit{card} \rightarrow \mathit{STOP})) \setminus \{\mathit{coin}, \mathit{card}\}$
- We get the non-deterministic process:
  - $(\mathit{choc} \rightarrow \mathit{STOP}) \sqcap \mathit{STOP}$

CSP Example - Vending Machine

- If synchronization was only required on "coin", we would obtain:
  - $VendingMachine \,|[\{coin\}]|\, Person \equiv$ $(coin \rightarrow choc \rightarrow STOP) \,\square\, (card \rightarrow STOP)$
- If we abstract this latter composite process by hiding the "coin" and "card" events:
  - $((coin \rightarrow choc \rightarrow STOP) \,\square\, (card \rightarrow STOP))$ $\setminus \{coin, card\}$
- We get the non-deterministic process:
  - $(choc \rightarrow STOP) \sqcap STOP$
- This process which either offers a "choc" event and then stops, or just stops;

CSP Example - Vending Machine

- If synchronization was only required on "coin", we would obtain:
  - $\mathit{VendingMachine} \,|[\{\mathit{coin}\}]|\, \mathit{Person} \equiv$ $(\mathit{coin} \to \mathit{choc} \to \mathit{STOP}) \,\Box\, (\mathit{card} \to \mathit{STOP})$
- If we abstract this latter composite process by hiding the "coin" and "card" events:
  - $((\mathit{coin} \to \mathit{choc} \to \mathit{STOP}) \,\Box\, (\mathit{card} \to \mathit{STOP}))$ $\setminus \{\mathit{coin}, \mathit{card}\}$
- We get the non-deterministic process:
  - $(\mathit{choc} \to \mathit{STOP}) \sqcap \mathit{STOP}$
- This process which either offers a "choc" event and then stops, or just stops;
- Externally non-determinism has been introduced.

Needham-Schroeder Public Key Protocol

1. A $\rightarrow$ B: $\{|N_a, A|\}_{K_b}$

Needham-Schroeder Public Key Protocol

1. A $\rightarrow$ B: $\{|N_a, A|\}_{K_b}$
2. B $\rightarrow$ A: $\{|N_a, N_b|\}_{K_a}$

Needham-Schroeder Public Key Protocol

1. A $\rightarrow$ B: $\{|N_a, A|\}_{K_b}$
2. B $\rightarrow$ A: $\{|N_a, N_b|\}_{K_a}$
3. A $\rightarrow$ B: $\{|N_b|\}_{K_b}$

NSPKP Goals

- The goal of the protocol is to establish mutual authentication between two parties A and B in the presence of adversary;

NSPKP Goals

- The goal of the protocol is to establish mutual authentication between two parties A and B in the presence of adversary;
- A and B obtain a secret shared key though direct communication using public key cryptography;

NSPKP Goals

- The goal of the protocol is to establish mutual authentication between two parties A and B in the presence of adversary;
- A and B obtain a secret shared key though direct communication using public key cryptography;
- This adversary can intercept messages, delay messages, read and copy messages and generate messages;

NSPKP Goals

- The goal of the protocol is to establish mutual authentication between two parties A and B in the presence of adversary;
- A and B obtain a secret shared key though direct communication using public key cryptography;
- This adversary can intercept messages, delay messages, read and copy messages and generate messages;
- This adversary can not learn the privates keys of principals.

Lowe's Specification using CSP

- $MSG1 \equiv \{Msg1.a.b.Encrypt.k.n_a.a' |$
  $a \in Initiator, a' \in Initiator, b \in Responder,$
  $k \in Key, n_a \in Nonce\},$

Lowe's Specification using CSP

- $MSG1 \equiv \{Msg1.a.b.Encrypt.k.n_a.a' |$
  $a \in Initiator, a' \in Initiator, b \in Responder,$
  $k \in Key, n_a \in Nonce\},$
- $MSG2 \equiv \{Msg2.b.a.Encrypt.k.n_a.n_b |$
  $a \in Initiator, b \in Responder,$
  $k \in Key, n_a \in Nonce, n_b \in Nonce\},$

Lowe's Specification using CSP

- $MSG1 \equiv \{Msg1.a.b.Encrypt.k.n_a.a' |$
  $a \in Initiator, a' \in Initiator, b \in Responder,$
  $k \in Key, n_a \in Nonce\},$
- $MSG2 \equiv \{Msg2.b.a.Encrypt.k.n_a.n_b |$
  $a \in Initiator, b \in Responder,$
  $k \in Key, n_a \in Nonce, n_b \in Nonce\},$
- $MSG3 \equiv \{Msg3.a.b.Encrypt.k.n_b |$
  $a \in Initiator, b \in Responder, k \in Key, n_b \in Nonce\},$

Lowe's Specification using CSP

- $MSG1 \equiv \{Msg1.a.b.Encrypt.k.n_a.a' \mid a \in Initiator, a' \in Initiator, b \in Responder, k \in Key, n_a \in Nonce\}$,
- $MSG2 \equiv \{Msg2.b.a.Encrypt.k.n_a.n_b \mid a \in Initiator, b \in Responder, k \in Key, n_a \in Nonce, n_b \in Nonce\}$,
- $MSG3 \equiv \{Msg3.a.b.Encrypt.k.n_b \mid a \in Initiator, b \in Responder, k \in Key, n_b \in Nonce\}$,
- $MSG \equiv MSG1 \cup MSG2 \cup MSG3$.

Lowe's Specification using CSP

- Standard communications in the system will be modelled by the channel comm;

Lowe's Specification using CSP

- Standard communications in the system will be modelled by the channel comm;
- We also want to model the fact that the intruder can fake or intercept messages, and so we introduce extra channels fake and intercept;

Lowe's Specification using CSP

- Standard communications in the system will be modelled by the channel comm;
- We also want to model the fact that the intruder can fake or intercept messages, and so we introduce extra channels fake and intercept;
- $\mathit{channel} \quad \mathit{comm}, \mathit{fake}, \mathit{intercept} : \mathit{MSG}$.

Lowe's Specification using CSP

- Standard communications in the system will be modelled by the channel comm;
- We also want to model the fact that the intruder can fake or intercept messages, and so we introduce extra channels fake and intercept;
- $\mathit{channel} \quad \mathit{comm}, \mathit{fake}, \mathit{intercept} : \mathit{MSG}.$
- This will ensure that the receiver of a faked message is not aware that it is a fake, and that the sender of an intercepted message is not aware that it is intercepted.

Lowe's Specification using CSP

- We introduce two extra channels, defining the external interface of the protocol;

Lowe's Specification using CSP

- We introduce two extra channels, defining the external interface of the protocol;
- We represent a request from a user for initiator "a" to connect with responder "b" by the event $user.a.b$;

Lowe's Specification using CSP

- We introduce two extra channels, defining the external interface of the protocol;
- We represent a request from a user for initiator "a" to connect with responder "b" by the event *user.a.b*;
- We represent the resulting session by the event *session.a.b*;

Lowe's Specification using CSP

- We introduce two extra channels, defining the external interface of the protocol;
- We represent a request from a user for initiator "a" to connect with responder "b" by the event *user.a.b*;
- We represent the resulting session by the event *session.a.b*;

Lowe's Specification using CSP

- We also add channels to represent the state of the agents:

Lowe's Specification using CSP

- We also add channels to represent the state of the agents:
  - We represent the initiator "a" thinking it is taking part in a run of the protocol with "b" by the event $I\_running.a.b$;

Lowe's Specification using CSP

- We also add channels to represent the state of the agents:
  - We represent the initiator "a" thinking it is taking part in a run of the protocol with "b" by the event $I\_running.a.b$;
  - We represent the responder "b" thinking it is taking part in a run of the protocol with "a" by the event $R\_running.a.b$;

Lowe's Specification using CSP

- We also add channels to represent the state of the agents:
    - We represent the initiator "a" thinking it is taking part in a run of the protocol with "b" by the event $I\_running.a.b$;
    - We represent the responder "b" thinking it is taking part in a run of the protocol with "a" by the event $R\_running.a.b$;
    - We represent the initiator committing to the session by the $eventI\_commit.a.b$;

Lowe's Specification using CSP

- We also add channels to represent the state of the agents:
  - We represent the initiator "a" thinking it is taking part in a run of the protocol with "b" by the event $I\_running.a.b$;
  - We represent the responder "b" thinking it is taking part in a run of the protocol with "a" by the event $R\_running.a.b$;
  - We represent the initiator committing to the session by the $eventI\_commit.a.b$;
  - We represent the responder committing to the session by $I\_commit.a.b$;

Lowe's Specification using CSP

- We also add channels to represent the state of the agents:
  - We represent the initiator "a" thinking it is taking part in a run of the protocol with "b" by the event $I\_running.a.b$;
  - We represent the responder "b" thinking it is taking part in a run of the protocol with "a" by the event $R\_running.a.b$;
  - We represent the initiator committing to the session by the $eventI\_commit.a.b$;
  - We represent the responder committing to the session by $I\_commit.a.b$;
- We declare these channels by:
- $channel \quad user, session, I\_running, B\_running, I\_commit, B\_.commit : Initiator.Responder.$

Lowe's Specification using CSP

- We will represent a responder with identity "a", who has a single nonce n_a, by the CSP process $INITIATOR(a, na)$.;

Lowe's Specification using CSP

- We will represent a responder with identity "a", who has a single nonce n_a, by the CSP process $INITIATOR(a, na)$.;
- If we want to consider a responder with more than one nonce, then we can compose several such processes, either sequentially or interleaved;

Lowe's Specification using CSP

- We will represent a responder with identity "a", who has a single nonce n_a, by the CSP process $INITIATOR(a, na)$.;
- If we want to consider a responder with more than one nonce, then we can compose several such processes, either sequentially or interleaved;
- Ignoring, for the moment, the possibility of intruder action, the process can be defined by:
  - $INITIATOR(a, n_a) \equiv user.a?b \rightarrow I\_.running.a.b \rightarrow$
    $comm!Msg1.a.b.Encrypt.key(b).n_a, a \rightarrow$
    $comm.Msg2.b.a.Encrypt.key(a)?n'_a.n_b \rightarrow$
    $if \quad n_a = n'_a$
    $\quad then \quad comm!Msg3.a.b.Encrypt.key(b).n_b \rightarrow$
    $\quad I\_commit.a.b \rightarrow session.a.b \rightarrow SKIP$
    $else \quad STOP.$

Lowe's Specification using CSP

- We now introduce the possibility of enemy action by applying a renaming to the above process;

Lowe's Specification using CSP

- We now introduce the possibility of enemy action by applying a renaming to the above process;
- Our renaming should ensure that message Is and message 3s sent by the initiator can be intercepted;

Lowe's Specification using CSP

- We now introduce the possibility of enemy action by applying a renaming to the above process;
- Our renaming should ensure that message Is and message 3s sent by the initiator can be intercepted;
- And message 2s can be faked;

Lowe's Specification using CSP

- We now introduce the possibility of enemy action by applying a renaming to the above process;
- Our renaming should ensure that message ls and message 3s sent by the initiator can be intercepted;
- And message 2s can be faked;
- We define an initiator with identity A and nonce Na by: $\mathit{INITIATOR}1 \equiv \mathit{INITIATOR}(A, N_a)$ $[[\mathit{comm}.\mathit{Msg}1 \leftarrow \mathit{comm}.\mathit{Msg}1, \mathit{comm}.\mathit{Msg}1 \leftarrow \mathit{intercept}.\mathit{Msg}1,$ $\mathit{comm}.\mathit{Msg}2 \leftarrow \mathit{comm}.\mathit{Msg}2, \mathit{comm}.\mathit{Msg}2 \leftarrow \mathit{fake}.\mathit{Msg}2,$ $\mathit{comm}.\mathit{Msg}3 \leftarrow \mathit{comm}.\mathit{Msg}3, \mathit{comm}.\mathit{Msg}3 \leftarrow \mathit{intercept}.\mathit{Msg}3]].$

Lowe's Intruder

- The intruder should be able to:

Lowe's Intruder

- The intruder should be able to:
  - Overhear and/or intercept any messages being passed in the system;

Lowe's Intruder

- The intruder should be able to:
  - Overhear and/or intercept any messages being passed in the system;
  - Decrypt messages that are encrypted with his own public key, so as to learn new nonces;

Lowe's Intruder

- The intruder should be able to:
  - Overhear and/or intercept any messages being passed in the system;
  - Decrypt messages that are encrypted with his own public key, so as to learn new nonces;
  - Introduce new messages into the system, using nonces he knows;

Lowe's Intruder

- The intruder should be able to:
  - Overhear and/or intercept any messages being passed in the system;
  - Decrypt messages that are encrypted with his own public key, so as to learn new nonces;
  - Introduce new messages into the system, using nonces he knows;
  - Replay any message he has seen (possibly changing plain-text parts), even if he does not understand the contents of the encrypted part;

Lowe's Intruder

- The intruder should be able to:
  - Overhear and/or intercept any messages being passed in the system;
  - Decrypt messages that are encrypted with his own public key, so as to learn new nonces;
  - Introduce new messages into the system, using nonces he knows;
  - Replay any message he has seen (possibly changing plain-text parts), even if he does not understand the contents of the encrypted part;

Lowe's Intruder

- Replay any message he has seen (possibly changing plain-text parts), even if he does not understand the contents of the encrypted part;

Lowe's Intruder

- Replay any message he has seen (possibly changing plain-text parts), even if he does not understand the contents of the encrypted part;
- We assume that the intruder is a user of the computer network;

Lowe's Intruder

- Replay any message he has seen (possibly changing plain-text parts), even if he does not understand the contents of the encrypted part;
- We assume that the intruder is a user of the computer network;
- We will define the most general (i.e. the most non-deterministic) intruder who can act as above;

Lowe's Intruder

- Replay any message he has seen (possibly changing plain-text parts), even if he does not understand the contents of the encrypted part;
- We assume that the intruder is a user of the computer network;
- We will define the most general (i.e. the most non-deterministic) intruder who can act as above;
- We consider an intruder with identity I, with public key K_i, who initially knows a nonce N_i.

To be continued....

On next lecture we will follow from here...

Questions????

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

UNIVERSIDADE FEDERAL
DE SANTA CATARINA