Security proofs in the symbolic model
the applied pi calculus

Karthikeyan Bhargavan

INRIA
karthikeyan.bhargavan@inria.fr
http://prosecco.inria.fr/personal/karthik

September 2013
Recap: Cryptographic protocols

- **Cryptographic protocol**
  - A set of rules for the exchange of data between multiple principals that uses cryptography to achieve security goals against a threat model.

- **Principal**: a protocol participant, typically human or computer

- **Security Goal**: the confidentiality or integrity of a data item, or the authentication of a principal

- **Threat Model**: the capabilities of the attacker

- **Examples**
  - Communications protocols: TLS, IPsec, SSH, WPA
  - Tamper-proof hardware: Smartcard, Navigo, SIM card
  - Privacy preserving applications: BitCoin, Electronic Voting
Informal Notation

- **Principals**: A (alice), B (bob), C (charlie), ...
- **Messages**: $m, n, o, \ldots$
  - **Constructors**: $\langle m, n \rangle$ (pairing), $\{m\}_k$, sig${m}_k$, pk$(m)$
  - **Destructors**: proj$_1(m)$, proj$_2(m)$, dec$(m, k)$, verify$(m, s, k)$
    - proj$_1(\langle m, n \rangle) = m$, proj$_2(\langle m, n \rangle) = n$

A protocol is informally specified as a sequence of messages exchanged between principals:
1. $A \rightarrow B$: $m_1$
2. $B \rightarrow C$: $m_2$
3. $C \rightarrow A$: $m_3$
   ...

- Denotes the expected behaviour of a single run of the protocol
- The goal of the attacker is to disrupt this behaviour!
Alice (A) wishes to perform an online transaction with her bank (B):

\[ A \rightarrow B : \text{request} \]
\[ B \rightarrow A : \text{response} \]
Recap: Writing protocols, finding attacks

- Alice (A) wishes to perform an online transaction with her bank (B):
  \[ A \rightarrow B : \text{request} \]
  \[ B \rightarrow A : \text{response} \]

- Encryption for confidentiality
  \[ A \rightarrow B : \{ \text{request} \}_{pk(B)} \]
  \[ B \rightarrow A : \{ \text{response} \}_{pk(A)} \]

- Nonces to prevent replays
  \[ B \rightarrow A : \{ N \}_{pk(A)} \]
  \[ A \rightarrow B : \{ \langle N, \text{request} \rangle \}_{pk(B)}, \text{sig} \]
Recap: Writing protocols, finding attacks

- Alice (A) wishes to perform an online transaction with her bank (B):
  \[ A \rightarrow B : \text{request} \]
  \[ B \rightarrow A : \text{response} \]

- Encryption for confidentiality
  \[ A \rightarrow B : \{\text{request}\}_{pk(B)} \]
  \[ B \rightarrow A : \{\text{response}\}_{pk(A)} \]

- Signature for integrity and authenticity
  \[ A \rightarrow B : \{\text{request}\}_{pk(B)}, \text{sig}\{\{\text{request}\}_{pk(B)}\}_{sk(A)} \]
  \[ B \rightarrow A : \{\text{response}\}_{pk(A)}, \text{sig}\{\{\text{response}\}_{pk(A)}\}_{sk(B)} \]
Recap: Writing protocols, finding attacks

- Alice (A) wishes to perform an online transaction with her bank (B):
  
  \[ A \rightarrow B : \text{request} \]
  
  \[ B \rightarrow A : \text{response} \]

- Encryption for confidentiality
  
  \[ A \rightarrow B : \{ \text{request} \}_{pk(B)} \]
  
  \[ B \rightarrow A : \{ \text{response} \}_{pk(A)} \]

- Signature for integrity and authenticity
  
  \[ A \rightarrow B : \{ \text{request} \}_{pk(B)}, \text{sig}\{\{ \text{request} \}_{pk(B)}\}_{sk(A)} \]
  
  \[ B \rightarrow A : \{ \text{response} \}_{pk(A)}, \text{sig}\{\{ \text{response} \}_{pk(A)}\}_{sk(B)} \]

- Nonces to prevent replays
  
  \[ B \rightarrow A : \{ N \}_{pk(A)} \]
  
  \[ A \rightarrow B : \{ \langle N, \text{request} \rangle \}_{pk(B)}, \text{sig}\{\{ \langle N, \text{request} \rangle \}_{pk(B)}\}_{sk(A)} \]
  
  \[ B \rightarrow A : \{ \langle N, \text{response} \rangle \}_{pk(A)}, \text{sig}\{\{ \langle N, \text{response} \rangle \}_{pk(A)}\}_{sk(B)} \]
Recap: From attacks to proofs

- Our informal notation is adequate for finding and explaining attacks
  - replay, man-in-the-middle, guessing attacks, ...
  - that compromise confidentiality and authenticity

- To precisely state and prove security theorems about cryptographic protocols, we need to move to a more formal setting.
  - Precisely state what actions each principal must do
  - Formalize security goals and threat model
  - Prove that these goals are met in all executions
Recap: A small process calculus

- Simplified version of the applied pi calculus [Abadi, Fournet, 2000]
- **Names**: $a, b, c, \ldots$ (used for keys, nonces, channels)
- **Messages**: $M, N, \ldots$
  - **Constructors**: $\langle m, n \rangle$ (pairing), $\{m\}_k$, $\text{sig}\{m\}_k$, $\text{pk}(m)$
  - **Destructors**: $\text{proj}_1(m)$, $\text{proj}_2(m)$, $\text{dec}(m, k)$, $\text{verify}(m, s, k)$
- **Processes**: $P, Q, R, \ldots$
  - $P, Q, R ::=$
    - 0: null process
    - $\text{new } a . P$: fresh name generation
    - $\text{in}(c, x).P$: message input (continue as $P$)
    - $\text{out}(c, M).P$: message output (continue as $P$)
    - $\text{let } x = g(M_1, \ldots, M_n) \text{ in } P \text{ else } Q$: destructor application
    - $\text{if } M = N \text{ then } P \text{ else } Q$: conditional
    - $P | Q$: parallel composition
    - $!P$: replication
Today: The Applied Pi Calculus

- **Introduction and syntax:** communication, concurrency, crypto
- **Term semantics:** equational theories, term reduction systems
- **Process semantics:** structural congruence, internal reduction
- **Attacker knowledge:** frames, deduction, labeled reduction
- **Security goals:** syntactic secrecy, authenticity
- **Proof technique:** invariants
Books:
- *Communicating and Mobile Systems: The Pi Calculus*, R. Milner
- *The Pi-Calculus: A Theory of Mobile Processes*, D. Sangiorgi

Papers:
- *Mobile values, new names, and secure communication*, M. Abadi and C. Fournet (POPL’01)
- *Applied pi calculus*, M.D. Ryan, B. Smyth (Tutorial, 2011)
Process Calculi

- **Process calculi** have been proposed as models for distributed systems
  - CCS [Milner, '80], CSP [Hoare, '85], ...
  - pi calculus, join calculus, ambient calculus, ...

- **Concurrency**: $P || Q$
  - *Interleaving semantics*: The actions in $P$ and $Q$ can happen in any order

- **Communication**: $\text{out}(c, M).P \parallel \text{in}(c, x).Q$
  - Both synchronous and asynchronous variants
  - *Synchronous*: instant communication, both processes evolve
    
    \[ \text{out}(c, M).P \parallel \text{in}(c, x).Q \longrightarrow P \parallel Q\{M/x\} \]
  
  - *Asynchronous*: output first, input may happen later
    
    \[ \text{out}(c, M).P \longrightarrow P \parallel \text{out } c(M) \]
    
    \[ \text{out}(c, M) \parallel \text{in}(c, x).Q \longrightarrow Q\{M/x\} \]
Proposed by Robin Milner in 1990s
Dynamic creation of channels, capabilities
  - useful to model mobile code, replicated servers, ... 
  - recently used to model security protocols, memory models, ...

**Names:** $a, b, c$

**Fresh name generation:** `new a.P`
  - create a fresh (secret) communication channel
  - create a new memory location (channel)
  - create a fresh random nonce, key, ...

**Replication:** `!P`
  - create as many copies of $P$ as necessary
  - $P \parallel P \parallel \cdots \parallel P$
Adding Crypto: Applied Pi Calculus

- **Spi calculus** [Abadi, Gordon ’99] adds cryptography to pi calculus
  - specific primitives are hard-coded (symmetric and asymmetric encryption)
- **Applied pi calculus** [Abadi, Fournet ’00] generalizes spi calculus
  - an algebra of terms (constructors, destructors)
  - equational theory to encode arbitrary cryptographic primitives
  - can also encode complex message formats
- **Messages**: $m, n, o, \ldots$
**Spi calculus** [Abadi, Gordon '99] adds cryptography to pi calculus
- specific primitives are hard-coded (symmetric and asymmetric encryption)

**Applied pi calculus** [Abadi, Fournet '00] generalizes spi calculus
- an algebra of terms (constructors, destructors)
- equational theory to encode arbitrary cryptographic primitives
- can also encode complex message formats

**Messages**: $m, n, o, \ldots$
- **Constructors**: $\langle m, n \rangle$ (pairing), $\{ m \}_k$, $\text{sig}\{ m \}_k$, $\text{pk}(m)$
- **Destructors**: $\text{proj}_1(m)$, $\text{proj}_2(m)$, $\text{dec}(m, k)$, $\text{verify}(m, s, k)$
  - $\text{proj}_1(\langle m, n \rangle) = m$, $\text{proj}_2(\langle m, n \rangle) = n$
Term Syntax

- $\mathcal{N}$: an infinite (countable) set of names $a, b, c, \ldots$
- $\mathcal{X}$: an infinite (countable) set of variables $x, y, z, \ldots$
- $\mathcal{F}$: a finite signature of function symbols $f, g, h, \ldots$
  - Includes constructors and destructors $\mathcal{F} = \mathcal{F}_C \cup \mathcal{F}_D$
- Terms represent messages that may be sent between processes $M, N, O, \ldots ::= \text{Terms}$
  - $a$ name
  - $x$ variable
  - $f(M_1, \ldots, M_n)$ function application
- $\mathcal{T}(\Sigma)$: terms constructed from the symbols in $\Sigma$
  - $\Sigma$ contains names, variables, and functions
Process Syntax

Processes: \( P, Q, R, \ldots \)

- \( P, Q, R ::= \) Processes
- 0
- new \( a.P \)
- in\((c, x).P\)
- out\((c, M).P\)
- if \( M = N \) then \( P \) else \( Q \)
- \( P \parallel Q \)
- !\( P \)

 Processes
null process
fresh name generation
message input (continue as \( P \))
message output (continue as \( P \))
conditional
parallel composition
replication
A set of rules that define equality on terms $M = N$

Example: encoding symmetric encryption

- Functions: $F = \{senc, sdec\}$
- Equations: $\forall M, N. sdec(senc(M, N), N) = M$

$\equiv$ is the smallest equivalence on terms that includes these equations

- $senc(a, b) = senc(a, b)$ (reflexivity)
- $senc(a, k) \neq senc(c, k)$
- $sdec(senc(a, k), k) = a$ (equation)
- $sdec(senc(a, k), k') \neq a$
- $senc(sdec(senc(a, k), k), k') = senc(a, k')$ (transitivity)
Term Semantics: Term Rewriting System

- A convenient way of expressing some equational theories
- \( \mathcal{R} \): a set of rules of the form: \( l \rightarrow r \)
  - where \( l \in \mathcal{T}(\mathcal{F} \cup \mathcal{X}) \) (can use any function or variable, but no names)
  - and \( r \in \mathcal{T}(\mathcal{F} \cup \mathcal{X}) \) (can use any function and any variable in \( l \))
- \( S \rightarrow_{\mathcal{R}} T \): term \( S \) rewrites to \( T \) if
  - \( S = S_t[M] \): \( S \) has a subterm \( M \)
  - \( M = l\sigma \): \( M \) matches the left of some rewriting rule
  - \( T = S_t[r\sigma] \)
  - Note: \( S \in \mathcal{T}(\mathcal{F} \cup \mathcal{X} \cup \mathcal{N}) \): \( S \) can contain any symbol
  - Assume: for each \( x \in \text{dom}(\sigma) \), \( x\sigma \) uses only constructor functions, no destructors

- Example: encoding symmetric encryption
  - Rewrite rule: \( \text{sdec}(\text{senc}(x, y), y) \rightarrow x \)
  - Instance: \( \text{sdec}(\text{senc}(a, k), k) \rightarrow a \) (\( M = S, \sigma = \{a/x, k/y\} \))
  - Instance: \( \text{senc}(\text{sdec}(\text{senc}(a, k), k), k') \rightarrow \text{senc}(a, k') \) (\( M \equiv ?, \sigma \equiv ? \))
Term Semantics: Term Rewriting System

- $\rightarrow^*_R$: the reflexive, transitive closure of $\rightarrow_R$
- $=_R$: the symmetric, reflexive, transitive closure of $\rightarrow_R$
- **Convergence**: A term rewriting system $R$ is convergent if it is:
  - **Terminating**: there is no infinite chain $T_1 \rightarrow_R T_2 \rightarrow_R \cdots$
  - **Confluent**: if $S =_R T$, then there exists $U$ such that $S \rightarrow^*_R U$ and $T \rightarrow^*_R U$
Term Semantics: Term Rewriting System

- $\rightarrow^*_R$: the reflexive, transitive closure of $\rightarrow_R$
- $=_R$: the symmetric, reflexive, transitive closure of $\rightarrow_R$
- **Convergence**: A term rewriting system $\mathcal{R}$ is convergent if it is:
  - **Terminating**: there is no infinite chain $T_1 \rightarrow_R T_2 \rightarrow_R \cdots$
  - **Confluent**: if $S =_R T$, then there exists $U$ such that $S \rightarrow^*_R U$ and $T \rightarrow^*_R U$
- $M \downarrow_R$: the $\mathcal{R}$-reduced form of $M$
  - $S = M \downarrow_R$ if $M \rightarrow^*_R S$ and $S \not\rightarrow_R$
  - $M \downarrow_R$ is defined (is unique) only if $\rightarrow_R$ is convergent
Term Semantics: Term Rewriting System

- $\rightarrow^*_{\mathcal{R}}$: the reflexive, transitive closure of $\rightarrow_{\mathcal{R}}$
- $=_{\mathcal{R}}$: the symmetric, reflexive, transitive closure of $\rightarrow_{\mathcal{R}}$
- **Convergence**: A term rewriting system $\mathcal{R}$ is convergent if it is:
  - **Terminating**: there is no infinite chain $T_1 \rightarrow_{\mathcal{R}} T_2 \rightarrow_{\mathcal{R}} \cdots$
  - **Confluent**: if $S =_{\mathcal{R}} T$, then there exists $U$ such that $S \rightarrow^*_{\mathcal{R}} U$ and $T \rightarrow^*_{\mathcal{R}} U$
- $M \downarrow_{\mathcal{R}}$: the $\mathcal{R}$-reduced form of $M$
  - $S = M \downarrow_{\mathcal{R}}$ if $M \rightarrow^*_{\mathcal{R}} S$ and $S \not\rightarrow_{\mathcal{R}}$
  - $M \downarrow_{\mathcal{R}}$ is defined (is unique) only if $\rightarrow_{\mathcal{R}}$ is convergent
- For convergent $\mathcal{R}$, $M =_{\mathcal{R}} N$ if and only if $M \downarrow_{\mathcal{R}} = N \downarrow_{\mathcal{R}}$
Process Semantics: Structural Congruence

- \( \equiv \): a structural congruence relation between processes
- \( P \parallel 0 \equiv P \)
- \( P \parallel Q \equiv Q \parallel P \)
- \( P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R \)
\[\equiv: \text{a structural congruence relation between processes}\]

\[P \parallel 0 \equiv P\]

\[P \parallel Q \equiv Q \parallel P\]

\[P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R\]

\[!P \equiv P \parallel !P\]
\[\equiv: \text{a structural congruence relation between processes}\]

- \[P \parallel 0 \equiv P\]
- \[P \parallel Q \equiv Q \parallel P\]
- \[P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R\]
- \[!P \equiv P \parallel !P\]
- \[\text{new } a.0 \equiv 0\]
- \[\text{new } a.\text{new } b.P \equiv \text{new } b.\text{new } a.P\]
- \[(\text{new } a.P) \parallel Q \equiv \text{new } a.(P \parallel Q) \text{ if } a \not\in \text{fn}(Q)\]
Process Semantics: Internal Reduction

\( \tau \rightarrow \): computation within a closed process

\textbf{out}(c, M).P \parallel \textbf{in}(c, x).Q \overset{\tau}{\rightarrow} P \parallel Q\{M/x\}

- More general form expects a pattern at the input and unifies it with \( M \)

\textbf{if} M = M \textbf{ then } P \textbf{ else } Q \overset{\tau}{\rightarrow} P

\textbf{if} M = N \textbf{ then } P \textbf{ else } Q \overset{\tau}{\rightarrow} Q, \text{ if } M \neq N

- relies on \( =_{\mathcal{R}} \) defined by the term rewriting semantics

\( P \overset{\tau}{\rightarrow} Q \text{ if } P \equiv P' \text{ and } Q \equiv Q' \text{ and } P' \overset{\tau}{\rightarrow} Q' \)
Exercises

- Write processes for the one-message protocol:

  \[ A \rightarrow B : \{ \text{request} \}_{\text{pk}(B)}, \text{sig}\{ \{ \text{request} \}_{\text{pk}(B)} \}_{\text{sk}(A)} \]

- Define the term rewriting system for public key encryption

- Describe the internal reduction sequence of these processes
**Frames:** \( \phi = \textbf{new} \bar{a}.\sigma \) where \( \sigma \) is a substitution of the form:

\[ \sigma = \{ M_1/x_1, M_2/x_2, \ldots, M_n/x_n \} \]

- \( \bar{a} \) are bound and may be renamed as necessary
- \( M_i \) may not contain destructor symbols

We use frames to represent the terms that may become known to the adversary

Example: \( \textbf{new} \ k.\{\text{senc}(a, k)/x_1, \text{senc}(b, k)/x_2\} \)

As a process evolves, we collect all the information given to the adversary in a frame

The adversary can use a frame to deduce additional terms
Assume that the functions in $\mathcal{F}$ are divided into private and public functions: $\mathcal{F} = \mathcal{F}_{\text{pub}} \cup \mathcal{F}_{\text{priv}}$

The attacker can use public function symbols and fresh names to deduce terms from a frame

$\phi \vdash M$: $M$ can be deduced from $\phi$

- $\text{new } \bar{a}.\{\bar{M}/\bar{x}\} \vdash M_i$
- $\text{new } \bar{a}.\{\bar{M}/\bar{x}\} \vdash b$, if $b \in \mathcal{N} \setminus \bar{a}$
- $\text{new } \bar{a}.\{\bar{M}/\bar{x}\} \vdash f(M_1, \ldots, M_n)$
  if $\phi \vdash M_i$ for each $i \in [1..n]$
- $\text{new } \bar{a}.\{\bar{M}/\bar{x}\} \vdash N$
  if $\text{new } \bar{a}.\{\bar{M}/\bar{x}\} \vdash N$ and $M = R N$
- Exercise: Show how to deduce $a$ from $\text{new } a, k.\{\text{senc}(a, k)/x_1, k/x_2\}$
Extended Processes

- Extend processes to record frames:
  \[ A, B, C ::= \]
  \[ P \] process
  \[ \text{new} \ a.A \] fresh name generation
  \[ A \parallel B \] parallel composition
  \[ \{M/x\} \] active substitution

  - Each active substitution assigns to a unique (free) variable
  - Bound variables and names are renamed to be unique

- \( \phi(A) \): the frame of \( A \), replace every plain process \( P \) in \( A \) by 0

  - Example: \( \phi(\text{new} \ a.(P \parallel \{M/x\} \parallel Q)) \equiv \text{new} \ a.\{M/x\} \)
Extended Process Semantics

- $\equiv$: we extend $\equiv$ to operate over extended processes
- $\rightarrow^\tau$: we extend internal reductions over extended processes
- $\rightarrow^l$: a new labelled semantics for interacting with the environment
  - $\text{in}(c, x). P \stackrel{\text{in}(c, M)}{\rightarrow} P\{M/x\}$
  - $\text{out}(c, M). P \stackrel{\text{out}(c, M\downarrow)}{\rightarrow} P \parallel \{M\downarrow/x\}$, where $x$ is a fresh variable
Extended Process Semantics

- $\equiv$: we extend $\equiv$ to operate over extended processes
- $\rightarrow^\tau$: we extend internal reductions over extended processes
- $\rightarrow^l$: a new labelled semantics for interacting with the environment
  
  - $\text{in}(c, x).P \xrightarrow{\text{in}(c, M)} P\{M/x\}$
  - $\text{out}(c, M).P \xrightarrow{\text{out}(c, M)} P \parallel \{M\downarrow /x\}$, where $x$ is a fresh variable
  - $A \rightarrow^l B$ if $A \equiv A'$ and $B \equiv B'$ and $A' \rightarrow B'$

Every output on a public channel goes to the attacker's frame
Every input on a public channel must be deducible from the frame
Extended Process Semantics

- $\equiv$: we extend $\equiv$ to operate over extended processes
- $\rightarrow^\tau$: we extend internal reductions over extended processes
- $\rightarrow^l$: a new labelled semantics for interacting with the environment

\begin{itemize}
  \item $\text{in}(c, x).P \xrightarrow{\text{in}(c, M)} P\{M/x\}$
  \item $\text{out}(c, M).P \xrightarrow{\text{out}(c, M)} P \parallel \{M \downarrow /x\}$, where $x$ is a fresh variable
  \item $A \xrightarrow{l} B$ if $A \equiv A'$ and $B \equiv B'$ and $A' \xrightarrow{l} B'$
  \item $C[A] \xrightarrow{l} C[B]$ if
    \begin{itemize}
      \item $A \xrightarrow{l} B$, and
      \item $C$ is an evaluation context (extended process with a hole)
      \item if $l = \text{in}(c, M)$ then $\phi(C[A]) \vdash M$
    \end{itemize}
\end{itemize}

Every output on a public channel goes to the attacker’s frame
Every input on a public channel must be deducible from the frame
Exercises

- Write extended processes for the one-message protocol:
  
  \[ A \rightarrow B : \{\text{request}\}_{pk(B)}, \text{sig}\{\{\text{request}\}_{pk(B)}\}_{sk(A)} \]

- Describe the labeled reduction sequence of these processes as it interacts with an unknown environment over the net channel.
Properties and Security Goals

- **Functional Adequacy**: Is there a valid reduction sequence that reaches the end of each process (no deadlock)
  - Note that the attacker can still block network messages, so there may well be valid reduction sequences that do not complete the protocol

- **Syntactic Secrecy**: Is there a reduction sequence that ends in a frame where the attacker can deduce a secret name $a$?
  - In our protocol, can the attacker deduce request or $sk(a)$?

- **Authentication**: If one principal completes the protocol, can we guarantee that the other principal has a consistent state?
  - In our protocol, can $A$ and $B$ have different values for $A$, $B$, request at the end of the protocol?
Proving Security Goals

- To show functional adequacy or an attack on secrecy/authentication, all we need is to display one trace (easy.)
- To prove secrecy/authentication, we need to show that all reduction sequences preserve this property (hard)
- This is hard because the number of reduction sequences is not finite
  - In fact, the problem is undecidable (next lecture)
  - If however, we restrict processes to finite number of sessions and names, it is decidable but PSPACE-complete (next lecture)
- In any case, it can be hard to prove both by hand and by computer
  - This explains why we have so many attacks on well-studied protocols
- We will explore one symbolic proof technique today
  - We will see wo more in this course.
Finding Invariants

- A method to prove syntactic secrecy
- Find an invariant on the shape of the extended process as it evolves
  - For all $B$ such that $A \rightarrow^* B$, $B$ must have the shape $I$
  - Example: $B \equiv \text{new } \vec{a} \cdot \{\text{senc}(M, a_i)/x\} \ || \ P$
    for some $M, i, P$ such that $M, P$ do not mention $a_i$
- Show that the frame of this invariant establishes your secrecy goal
  - The attacker cannot deduce a secret from the frame of $I$
  - $\phi(I) = \text{new } \vec{a} \cdot \{\text{senc}(M, a_i)/x\}$
    The attacker cannot deduce the secrets $a_k$ or $M$ from this frame
Exercise

- Start from the extended processes for the one-message protocol:

\[ A \rightarrow B : \{ \text{request} \}_{\text{pk}(B)}, \text{sig}\{\{\text{request}\}_{\text{pk}(B)}\}_{\text{sk}(A)} \]

- The processes are of the form: \( S = \text{new } s_A, s_B, \text{request.}(P_A \parallel P_B) \)
where \( P_A \) can use \( s_A, \text{pk}(s_B) \), and request and \( P_B \) can use \( s_B, \text{pk}(s_a) \)
and both of them communicate over a free channel net.

- Assume that the attacker knows \( \text{net, pk}(s_a), \text{and pk}(s_b) \)

- Find an invariant \( I \) that captures all \( B \) such that \( S \xrightarrow{*} B \)

- Show that the attacker cannot deduce \text{request} from \( \phi(I) \)
  
  - Every deduced term that mentions \text{request} has a subterm of the form \( \text{senc(request, pk(s_b))} \)
  
  - Corollary: \( s_a, s_b, \text{and request} \) are kept secret
What can an attacker learn from a frame?

For syntactic secrecy, we ask whether he can derive a secret name $a$.

For weak secrecy, we ask whether he can distinguish between frames.

*Static Equivalence*

- $(M =_{R} N)\phi$: if $\phi = \text{new } \vec{a}.\sigma$ and $M\sigma \downarrow_{R} = N\sigma \downarrow_{R}$
- $\phi_{1} \sim \phi_{2}$: if $\text{dom}(\phi_{1}) = \text{dom}(\phi_{2})$ and for all $M$ that use public functions, $M\phi_{1}$ is a constructor terms iff $M\phi_{2}$ is a constructor term and for all $M, N$ that use public functions, $(M =_{R} N)\phi_{1}$ iff $(M =_{R} N)\phi_{2}$

Captures guessing attacks: Is

$\{\text{aenc}(a, pk(s))/x, a/y\} \sim \{\text{aenc}(b, pk(s)), a/y\}$?