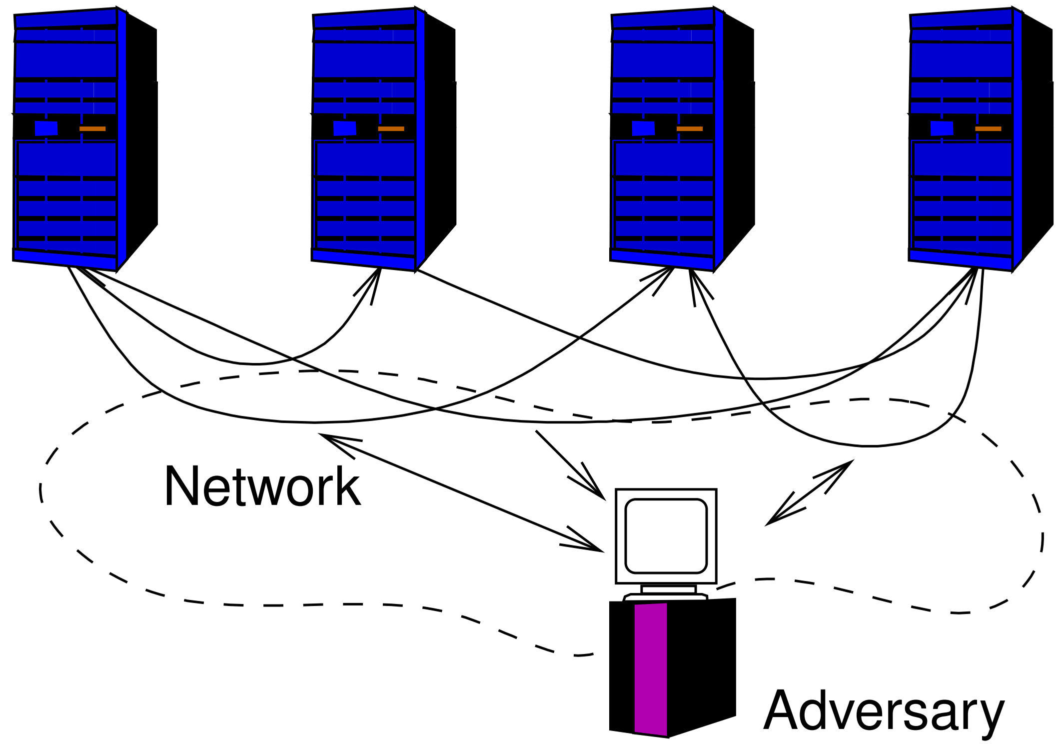
Secrecy Types for a Simulatable Cryptographic Library

Peeter Laud

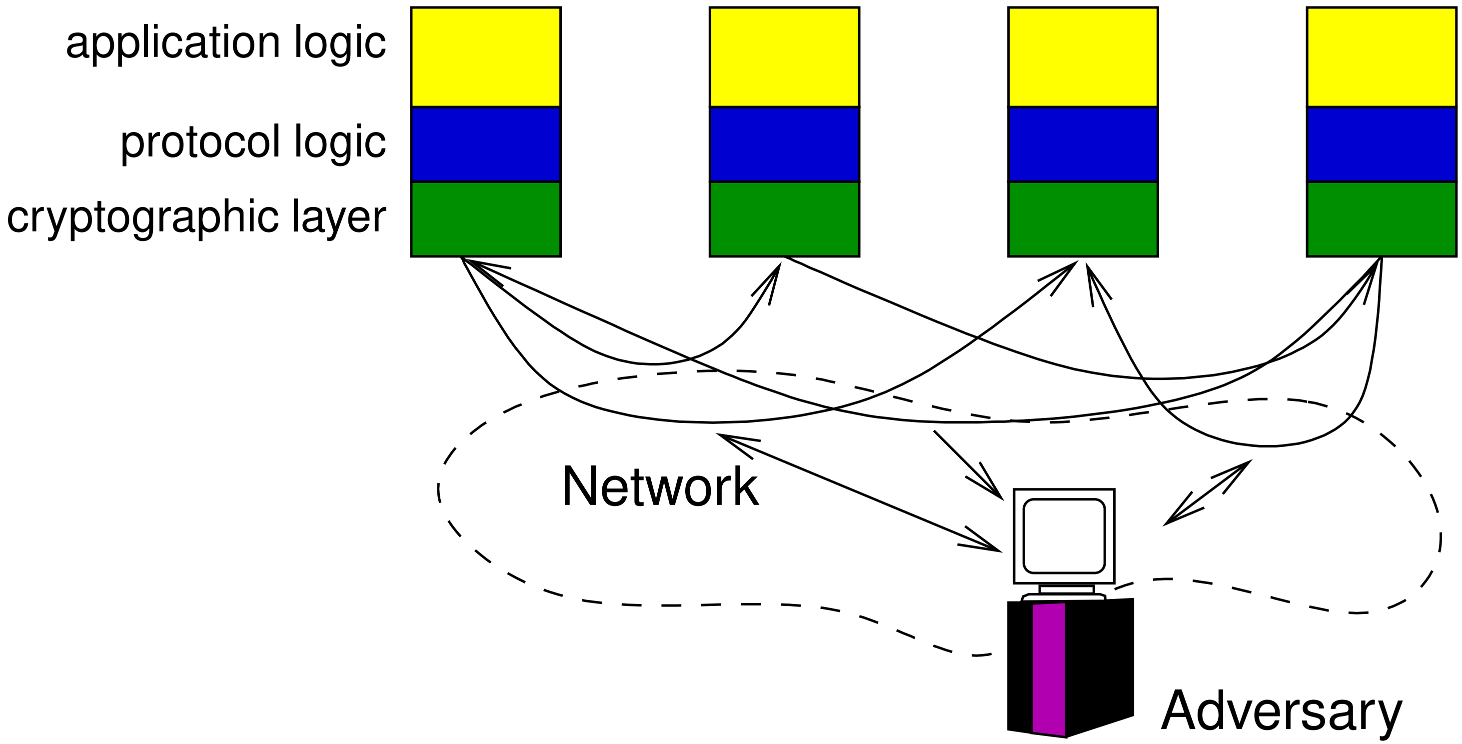
Tartu University & Cybernetica AS

A distributed system



Several sites, channels between some of them, channels may be secure, authentic or insecure.

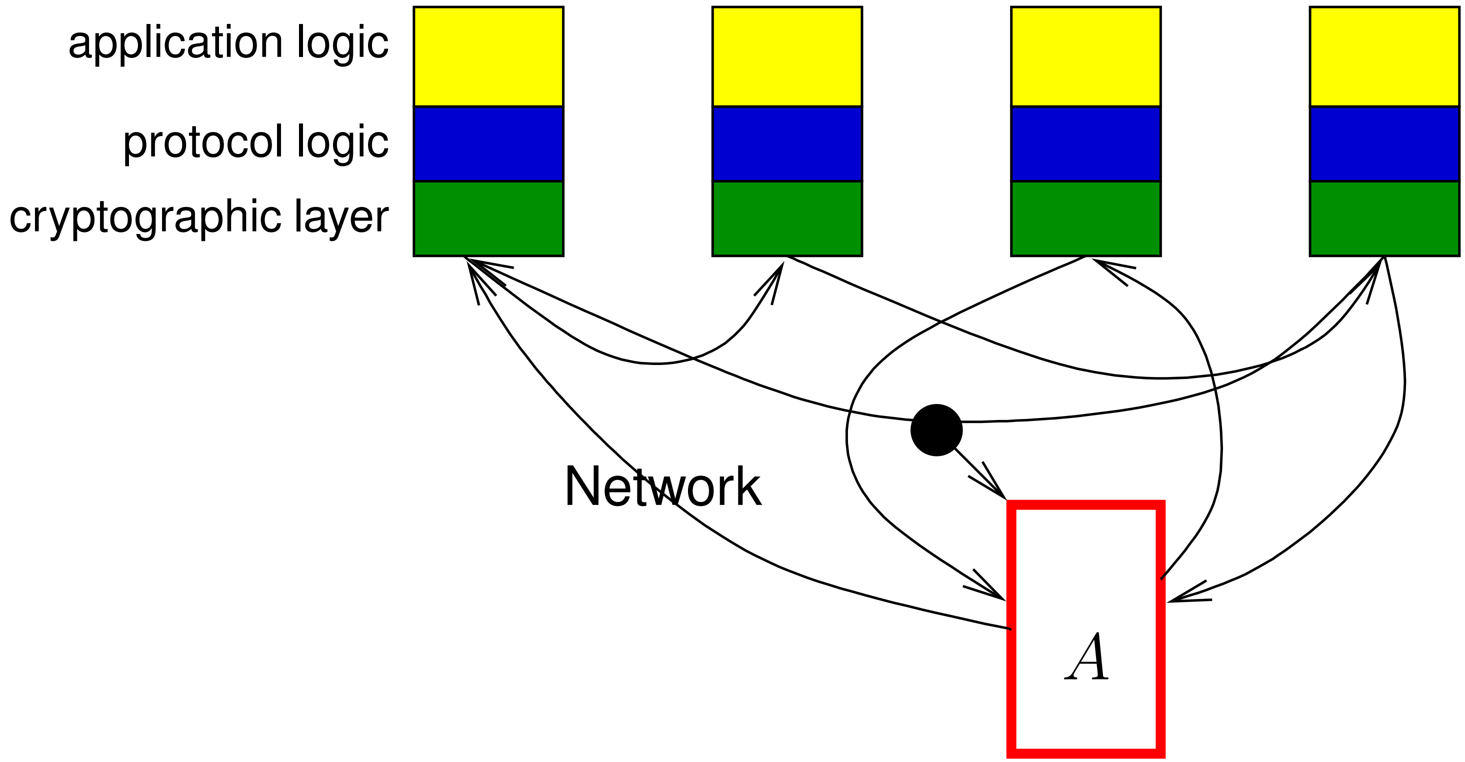
A distributed system



The instructions of a site can typically be partitioned to the following three layers.

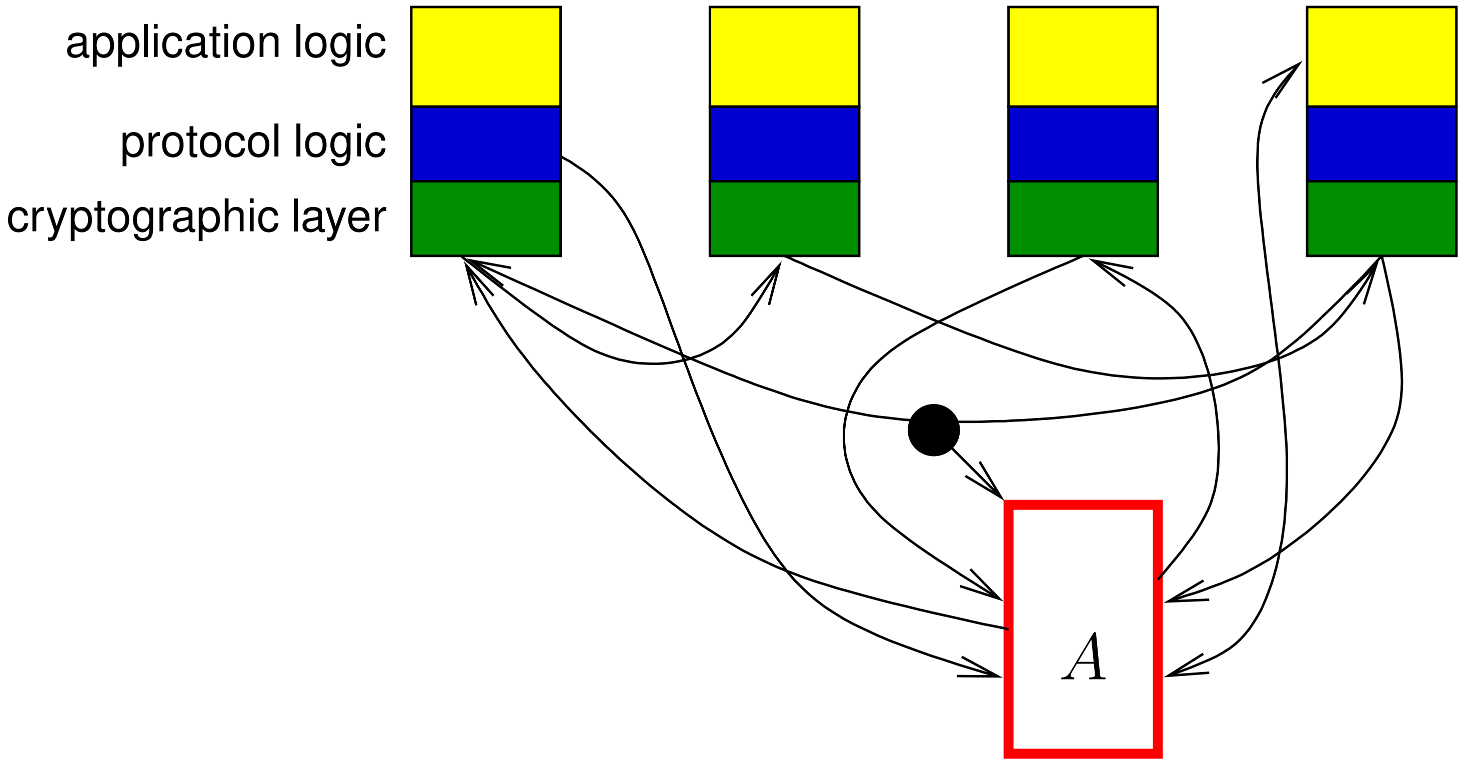
Site — a composition of three (or more) interacting Turing machines.

A distributed system



All three types of channels can be modeled with the help of secure channels.

A distributed system



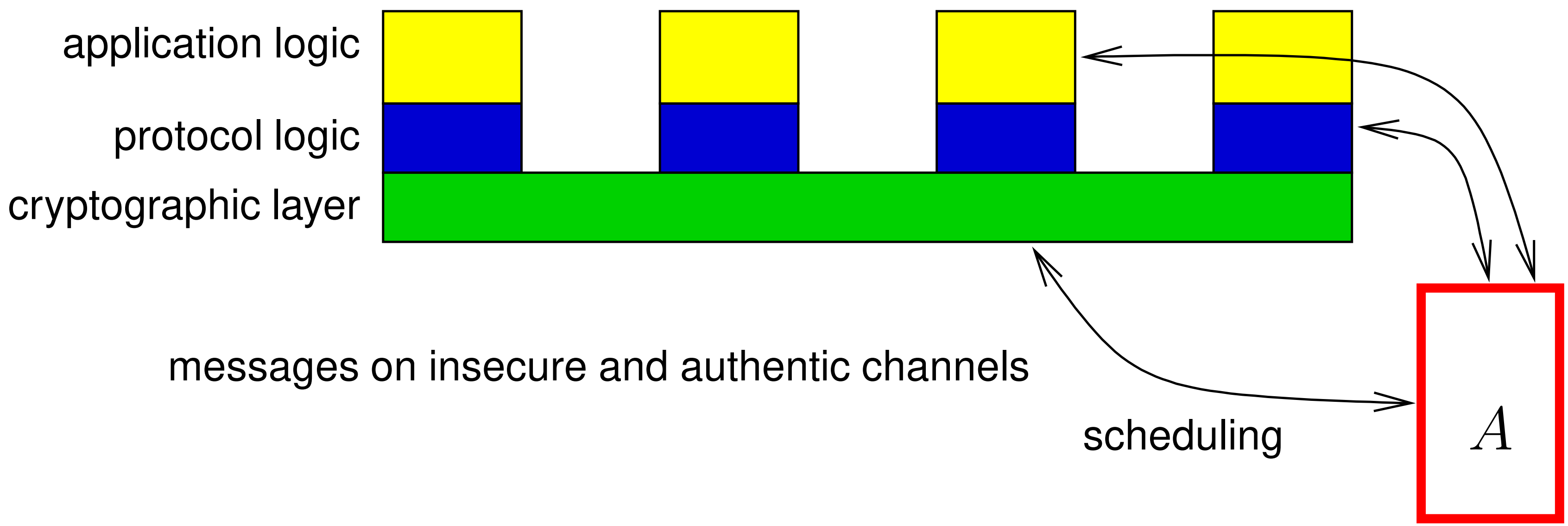
The upper layers may also influence and be influenced by the adversary.

Example: I/O, timing.

The simulatable cryptographic library

- May serve as the cryptographic layer.
- Takes API calls from the layer above to
  - generate new encryption/decryption keys, encrypt and decrypt;
    - both symmetrically and asymmetrically
  - generate new signature keys, sign and verify;
  - take and return (unstructured) data; construct and destruct tuples;
  - send messages to other parties.
- Receives messages from other parties and forwards them to the layer above.
- The overlying layer accesses all messages through handles.

The abstract cryptographic library



A monolithic library — consists of a single machine.

Cannot be directly implemented.

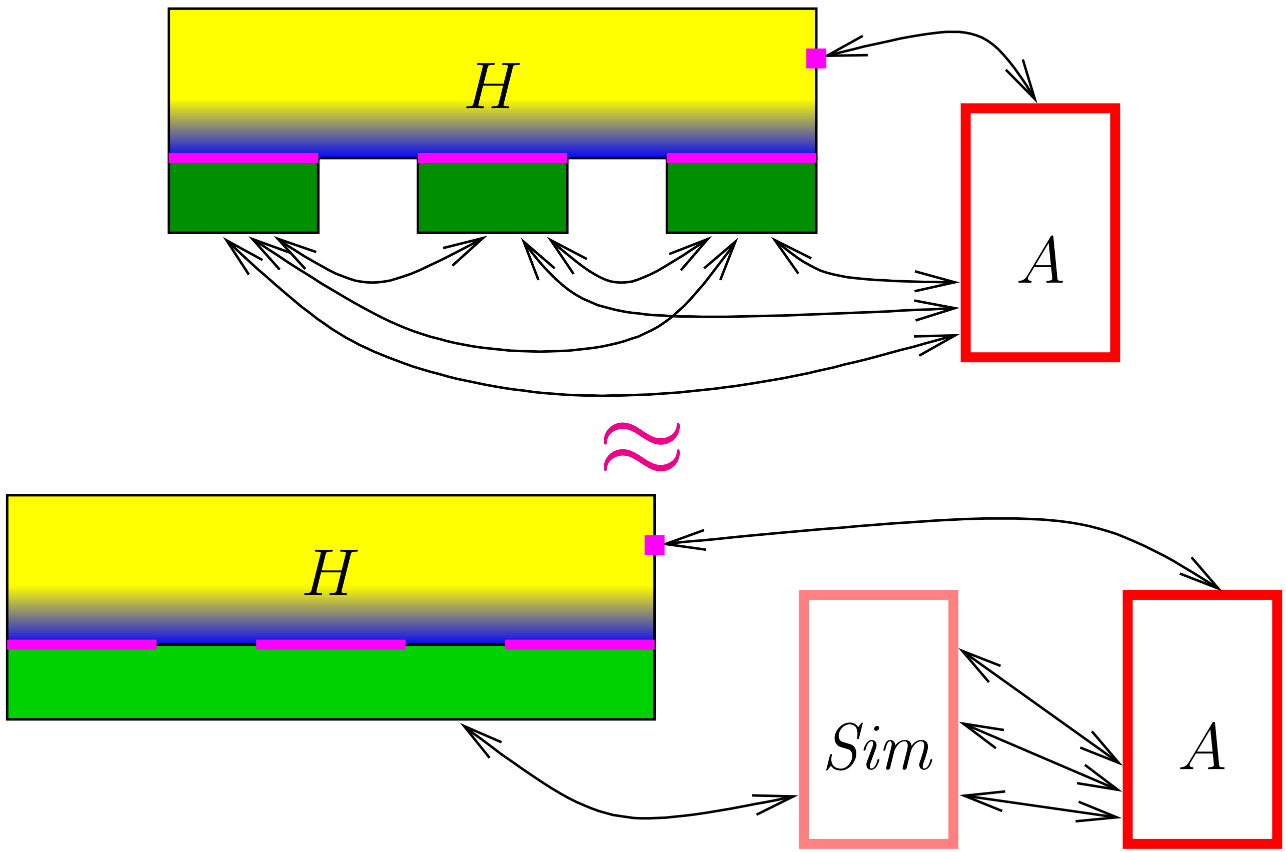
Main part — a database of terms recording their structure and parties that have access to them.

Terms in the database $\approx$ terms in the Dolev-Yao model.

Possible operations also similar to the Dolev-Yao model.

Simulatability

$\exists\, Sim$, such that for all A and almost all H:



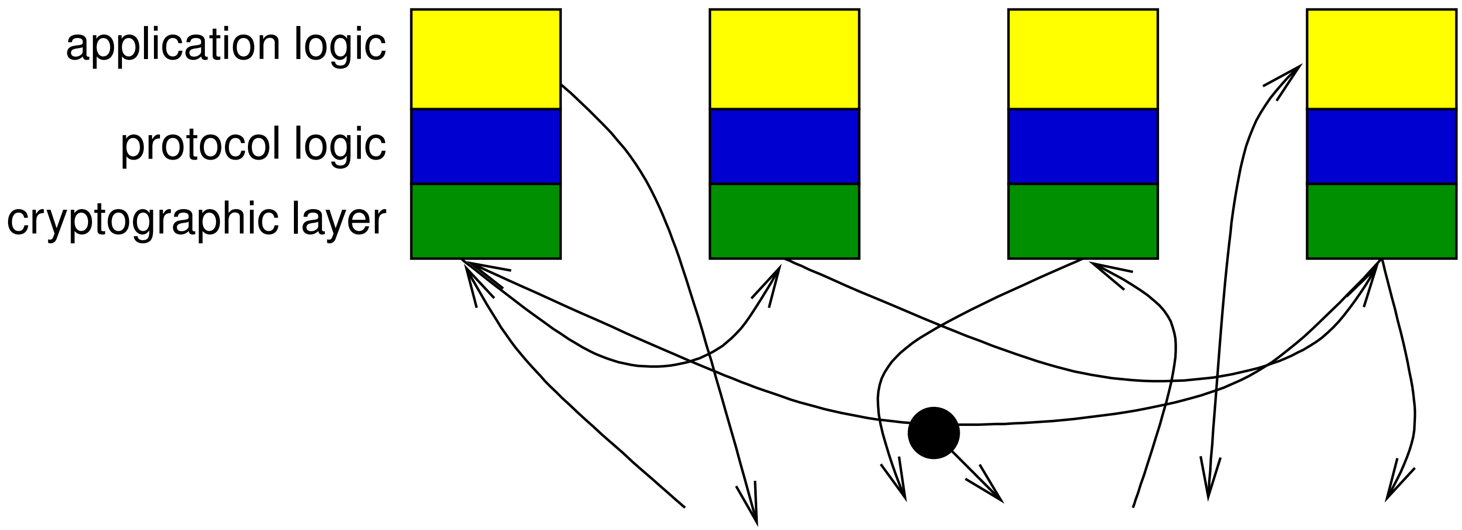
- The views of the user H must be indistinguishable.
- Conditions on H nontrivial, but not too restrictive.

Simulatability means...

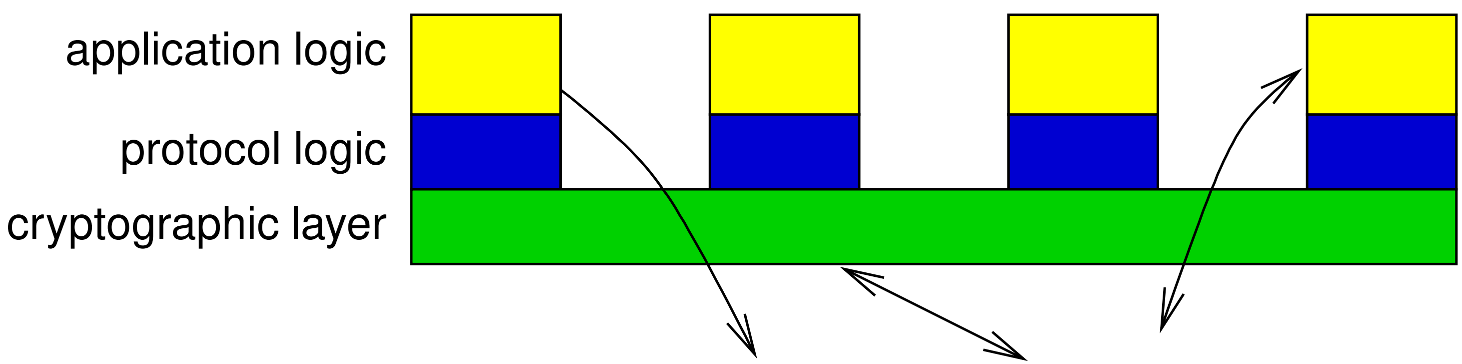
- We say that the real library is at least as secure as the ideal library.
- Meaning of the definition: anything that may happen to the user of the concrete library may also happen to the user of the abstract library.
  - this “anything” includes all bad things.
- Vice versa: if nothing bad can happen to the user of the abstract library then nothing bad can happen to the user of the concrete library.

In our case…

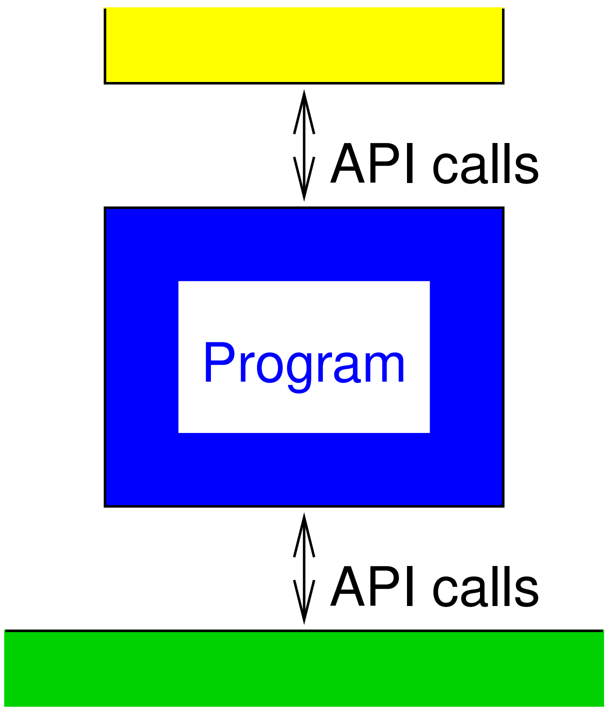
instead of analysing



we may analyse



A protocol participant



Theorem. (B & Pf, S&P '05)
A protocol participant keeps a data item M received from above secret if

- M is passed downwards only as unstructured data.
- M will not become known to the adversary.
- M does not affect the control flow of the Program.

Our contribution

- We have devised a type system for Programs.
  - …and also devised a suitable language for Programs.
    - Similar to spi-calculus
    - …but with completely deterministic semantics.
- If the Programs of all participants type then the secrecy of all data items received from above is preserved.
- This is the first security type system that has been applied to the systems containing the simulatable cryptographic library.
- Even more, it's the first instance of a formalism typically used by the automated protocol analysers that has been applied to this library.

Program

- $\text{Thread}_1|\text{Thread}_2|\cdots|\text{Thread}_n$
- Each activation of a program processes one received message.
  - Messages can be received from and sent to
    - the application layer;
    - other participants (through the crypto. layer).
- A received message is passed to the first thread.
- A thread may reject the message. It is then passed to the next thread.
- A thread may be replicated. Then a new thread is spawned upon receiving a message.
  - . . . and destroyed when the message is rejected.

Thread

- Upon an activation, the thread may decompose known messages and compose new messages.
- The $\mathbf{send}$-command is the only command visible outside the thread.
- The $\mathbf{send}$-command ends the execution of a thread.
  - Artifact of the cryptographic library.
- Other means to end the execution are the rejection of the message and doing nothing.
- When a thread has stopped execution, it has become zero or more threads.
  - but if a message was rejected, the state of the thread before execution is restored.

Language for threads

Variables $x \in \mathbf{Var}$. Constants / values $n, v \in \mathbb{Z}$.

Abstract channels $c \in \mathbf{Chan}$.

Expressions

$$
\begin{array}{rcl|l|l}
e & ::= & n & \mathit{symkey}() & \mathit{asymkeypair}() \\
& | & x & \mathit{symenc}(e,e) & \mathit{asymenc}(e,e) \\
& | & (e,\ldots,e) & \mathit{symdec}(e,e) & \mathit{asymdec}(e,e) \\
& | & \pi_i(e) & \mathit{nonce}() & \mathit{pubkey}(e) \\
& | & \mathit{store}(e) & \mathit{retrieve}(e) &
\end{array}
$$

Processes

$$P ::= P_{\mathrm{act}} \mid P_{\mathrm{inact}}$$

$$P_{\mathrm{inact}} ::= (T_1|\cdots|T_n) \mid \mathbf{Reject}$$

$$
\begin{array}{rcl}
P_{\mathrm{act}} & ::= & \mathit{let}\ x := e\ \mathit{in}\ P\ \mathit{else}\ P' \\
& | & \mathit{if}\ e = e\ \mathit{then}\ P\ \mathit{else}\ P' \\
& | & \mathit{send}_c\ e\ \mathit{to}\ e.P_{\mathrm{inact}}
\end{array}
$$

Threads

$$
\begin{array}{rcl}
T & ::= & \mathit{receive}_c\ x\ \mathit{from}\ x'.P \\
& | & !\mathit{receive}_c\ x\ \mathit{from}\ x'.P
\end{array}
$$

Abstract channels

- Used by the type system to tell apart different messages on the same channels between participants.
- A security rating (insecure, authentic, or secure) is assigned to each abstract channel.
- The protocol participants append to each outgoing message the name of the abstract channel.
  - The message uses the channel with the correct security rating.
  - If c in $receive_c$ does not match then abstract channel recorded in the incoming message, then this message is immediately rejected by that thread.

Language for threads

Variables $x \in \mathbf{Var}$. Constants / values $n, v \in \mathbb{Z}$.

Abstract channels $c \in \mathbf{Chan}$.

Expressions

$$
\begin{array}{rclclcl}
e & ::= & n & \mid & symkey() & \mid & asymkeypair() \\
 & \mid & x & \mid & symenc(e,e) & \mid & asymenc(e,e) \\
 & \mid & (e,\ldots,e) & \mid & symdec(e,e) & \mid & asymdec(e,e) \\
 & \mid & \pi_i(e) & \mid & nonce() & \mid & pubkey(e) \\
 & \mid & store(e) & \mid & retrieve(e) & &
\end{array}
$$

Processes

$$P ::= P_{\mathrm{act}} \mid P_{\mathrm{inact}}$$

$$P_{\mathrm{inact}} ::= (T_1 | \cdots | T_n) \mid \mathbf{Reject}$$

$$
\begin{array}{rcl}
P_{\mathrm{act}} & ::= & let\ x := e\ in\ P\ else\ P' \\
 & \mid & if\ e = e\ then\ P\ else\ P' \\
 & \mid & send_c\ e\ to\ e.P_{\mathrm{inact}}
\end{array}
$$

Threads

$$
\begin{array}{rcl}
T & ::= & receive_c\ x\ from\ x'.P \\
 & \mid & !receive_c\ x\ from\ x'.P
\end{array}
$$

Language for threads

Variables $x \in \mathbf{Var}$. Constants / values $n, v \in \mathbb{Z}$.

Abstract channels $c \in \mathbf{Chan}$.

Expressions

$$
\begin{array}{lll}
e & ::= & n \\
 & | & x \\
 & | & (e, \ldots, \\
 & | & \pi_i(e) \\
 & | & store(e)
\end{array}
\qquad
\begin{array}{l}
asymkeypair() \\
asymenc(e, e) \\
asymdec(e, e) \\
pubkey(e)
\end{array}
$$

When receiving v from participant p, becomes

$$P[M/x, p/x']$$

Processes

$$P ::= P_{\mathrm{act}} \mid P_{\mathrm{inact}}$$

$$P_{\mathrm{inact}} ::= (T_1 | \cdots | T_n) \mid \mathbf{Reject}$$

$$
\begin{array}{lll}
P_{\mathrm{act}} & ::= & let\ x := e\ in\ P\ else\ P' \\
 & | & if\ e = e\ then\ P\ else\ P' \\
 & | & send_c\ e\ to\ e.P_{\mathrm{inact}}
\end{array}
$$

Threads

$$
\begin{array}{lll}
T & ::= & receive_c\ x\ from\ x'.P \\
 & | & !receive_c\ x\ from\ x'.P
\end{array}
$$

Language for threads

Variables $x \in \mathbf{Var}$.

Constants / values $n, v \in \mathbb{Z}$.

Abstract channels $c \in \mathbf{Chan}$.

Expressions

$$
\begin{array}{lll}
e & ::= & n \\
 & | & x \\
 & | & (e, \ldots, \\
 & | & \pi_i(e) \\
 & | & store(e
\end{array}
$$

$\mathit{eypair}()$

$\mathit{nc}(e, e)$

$\mathit{lec}(e, e)$

$\mathit{y}(e)$

When receiving v from participant p, becomes

$$
\begin{array}{l}
P[M/x, p/x'] \,| \\
\qquad !\mathit{receive}_c\ x\ \mathit{from}\ x'.P
\end{array}
$$

Processes

$$P ::= P_{\mathrm{act}} \mid P_{\mathrm{inact}}$$

$$P_{\mathrm{inact}} ::= (T_1 | \cdots | T_n) \mid \mathbf{Reject}$$

$$
\begin{array}{lll}
P_{\mathrm{act}} & ::= & \mathit{let}\ x := e\ \mathit{in}\ P\ \mathit{else}\ P' \\
 & | & \mathit{if}\ e = e\ \mathit{then}\ P\ \mathit{else}\ P' \\
 & | & \mathit{send}_c\ e\ \mathit{to}\ e.P_{\mathrm{inact}}
\end{array}
$$

Threads

$$
\begin{array}{lll}
T & ::= & \mathit{receive}_c\ x\ \mathit{from}\ x'.P \\
 & | & !\mathit{receive}_c\ x\ \mathit{from}\ x'.P
\end{array}
$$

Language for threads

Variables $x \in \mathbf{Var}$. Constants / values $n, v \in \mathbb{Z}$.

Abstract channels $c \in \mathbf{Chan}$.

Expressions

$$
\begin{array}{rclclcl}
e & ::= & n & \mid & \mathit{symkey}() & \mid & \mathit{asymkeypair}() \\
& \mid & x & \mid & \mathit{symenc}(e,e) & \mid & \mathit{asymenc}(e,e) \\
& \mid & (e,\ldots,e) & \mid & \mathit{symdec}(e,e) & \mid & \mathit{asymdec}(e,e) \\
& \mid & \pi_i(e) & \mid & \mathit{nonce}() & \mid & \mathit{pubkey}(e) \\
& \mid & \mathit{store}(e) & \mid & \mathit{retrieve}(e) & &
\end{array}
$$

Processes

$$P ::= P_{\mathrm{act}} \mid P_{\mathrm{inact}}$$

$$P_{\mathrm{inact}} ::= (T_1 | \cdots | T_n) \mid \mathbf{Reject}$$

$$
\begin{array}{rcl}
P_{\mathrm{act}} & ::= & \mathit{let}\ x := e\ \mathit{in}\ P\ \mathit{else}\ P' \\
& \mid & \mathit{if}\ e = e\ \mathit{then}\ P\ \mathit{else}\ P' \\
& \mid & \mathit{send}_c\ e\ \mathit{to}\ e.P_{\mathrm{inact}}
\end{array}
$$

Threads

$$
\begin{array}{rcl}
T & ::= & \mathit{receive}_c\ x\ \mathit{from}\ x'.P \\
& \mid & !\mathit{receive}_c\ x\ \mathit{from}\ x'.P
\end{array}
$$

Language for threads

Variables $x \in \mathbf{Var}$.

Constants / values $n, v \in \mathbb{Z}$.

Abstract channels $c \in \mathbf{Chan}$.

Expressions

$$
\begin{array}{lcl}
e & ::= & n \quad | \quad \ldots ey() \quad | \quad \ldots mkeypair() \\
 & | & x \quad \ldots \quad menc(e,e) \\
 & | & (e, \ldots, \quad \ldots \quad mdec(e,e) \\
 & | & \pi_i(e) \quad | \quad \ldots(\quad | \quad \ldots key(e) \\
 & | & store(e) \quad | \quad retrieve(e)
\end{array}
$$

If $e \Downarrow v$, becomes $P[v/x]$

If $\nexists v : e \Downarrow v$, becomes P'

Processes

$$P ::= P_{\mathrm{act}} \mid P_{\mathrm{inact}}$$

$$P_{\mathrm{inact}} ::= (T_1 | \cdots | T_n) \mid \mathbf{Reject}$$

$$
\begin{array}{lcl}
P_{\mathrm{act}} & ::= & let\ x := e\ in\ P\ else\ P' \\
 & | & if\ e = e\ then\ P\ else\ P' \\
 & | & send_c\ e\ to\ e.P_{\mathrm{inact}}
\end{array}
$$

Threads

$$
\begin{array}{lcl}
T & ::= & receive_c\ x\ from\ x'.P \\
 & | & !receive_c\ x\ from\ x'.P
\end{array}
$$

Language for threads

Variables $x \in \mathbf{Var}$. Constants / values $n, v \in \mathbb{Z}$.

Abstract channels $c \in \mathbf{Chan}$.

Expressions

$$
\begin{array}{lclcl}
e & ::= & n & \mid & \mathit{symkey}() & \mid & \mathit{asymkeypair}() \\
& \mid & x & \mid & \mathit{symenc}(e,e) & \mid & \mathit{asymenc}(e,e) \\
& \mid & (e,\ldots,e) & \mid & \mathit{symdec}(e,e) & \mid & \mathit{asymdec}(e,e) \\
& \mid & \pi_i(e) & \mid & \mathit{nonce}() & \mid & \mathit{pubkey}(e) \\
& \mid & \mathit{store}(e) & \mid & \mathit{retrieve}(e) & &
\end{array}
$$

Processes

$$P ::= P_{\mathrm{act}} \mid P_{\mathrm{inact}}$$

$$P_{\mathrm{inact}} ::= (T_1|\cdots|T_n) \mid \mathbf{Reject}$$

$$
\begin{array}{lcl}
P_{\mathrm{act}} & ::= & \mathit{let}\ x := e\ \mathit{in}\ P\ \mathit{else}\ P' \\
& \mid & \mathit{if}\ e = e\ \mathit{then}\ P\ \mathit{else}\ P' \\
& \mid & \mathit{send}_c\ e\ \mathit{to}\ e.P_{\mathrm{inact}}
\end{array}
$$

Threads

$$
\begin{array}{lcl}
T & ::= & \mathit{receive}_c\ x\ \mathit{from}\ x'.P \\
& \mid & !\mathit{receive}_c\ x\ \mathit{from}\ x'.P
\end{array}
$$

Language for threads

Variables $x \in \mathbf{Var}$.

Constants / values $n, v \in \mathbb{Z}$.

Abstract channels $c \in \mathbf{Chan}$.

Expressions

$$
\begin{array}{lllll}
e & ::= & n & \mid \; symkey() & \mid \; asymkeypair() \\
& \mid & x & \mid \; symenc(e,e) & \mid \; asymenc(e,e) \\
& \mid & (e,\ldots,e) & \mid \; symdec(e,e) & \mid \; asymdec(e,e) \\
& \mid & \pi_i(e) & \mid \; nonce() & \mid \; pubkey(e) \\
& \mid & store(e) & \mid \; retrieve(e) &
\end{array}
$$

Processes

$$P ::= P_{\mathrm{act}} \mid P_{\mathrm{inact}}$$

$$P_{\mathrm{inact}} ::= (T_1 | \cdots | T_n) \mid \mathbf{Reject}$$

$$
\begin{array}{lll}
P_{\mathrm{act}} & ::= & let\ x := e\ in\ P\ else\ P' \\
& \mid & if\ e = e\ then\ P\ else\ P' \\
& \mid & send_c\ e\ to\ e.P_{\mathrm{inact}}
\end{array}
$$

Threads

$$
\begin{array}{lll}
T & ::= & receive_c\ x\ from\ x'.P \\
& \mid & !receive_c\ x\ from\ x'.P
\end{array}
$$

Language for threads

Variables $x \in \mathbf{Var}$. Constants / values $n, v \in \mathbb{Z}$.

Abstract channels $c \in \mathbf{Chan}$.

Expressions

$$
\begin{array}{rcl|l|l}
e & ::= & n & symkey() & asymkeypair() \\
 & | & x & symenc(e,e) & asymenc(e,e) \\
 & | & (e,\ldots,e) & symdec(e,e) & asymdec(e,e) \\
 & | & \pi_i(e) & nonce() & pubkey(e) \\
 & | & store(e) & retrieve(e) &
\end{array}
$$

Processes

$$P ::= P_{\mathrm{act}} \mid P_{\mathrm{inact}}$$

$$P_{\mathrm{inact}} ::= (T_1|\cdots|T_n) \mid \mathbf{Reject}$$

$$
\begin{array}{rcl}
P_{\mathrm{act}} & ::= & let\ x := e\ in\ P\ else\ P' \\
 & | & if\ e = e\ then\ P\ else\ P' \\
 & | & send_c\ e\ to\ e.P_{\mathrm{inact}}
\end{array}
$$

Threads

$$
\begin{array}{rcl}
T & ::= & receive_c\ x\ from\ x'.P \\
 & | & !receive_c\ x\ from\ x'.P
\end{array}
$$

Language for threads

Variables $x \in \mathbf{Var}$. Constants / values $n, v \in \mathbb{Z}$.

Abstract channels $c \in \mathbf{Chan}$.

Expressions

$$
\begin{array}{lllllll}
e & ::= & n & \mid & symkey() & \mid & asymkeypair() \\
 & \mid & x & \mid & symenc(e,e) & \mid & asymenc(e,e) \\
 & \mid & (e,\ldots,e) & \mid & symdec(e,e) & \mid & asymdec(e,e) \\
 & \mid & \pi_i(e) & \mid & nonce() & \mid & pubkey(e) \\
 & \mid & store(e) & \mid & retrieve(e) & &
\end{array}
$$

Processes

$$P ::= P_{\mathrm{act}} \mid P_{\mathrm{inact}}$$

$$P_{\mathrm{inact}} ::= (T_1|\cdots|T_n) \mid \mathbf{Reject}$$

$$
\begin{array}{lll}
P_{\mathrm{act}} & ::= & let\ x := e\ in\ P\ else\ P' \\
 & \mid & if\ e = e\ then\ P\ else\ P' \\
 & \mid & send_c\ e\ to\ e.P_{\mathrm{inact}}
\end{array}
$$

Threads

$$
\begin{array}{lll}
T & ::= & receive_c\ x\ from\ x'.P \\
 & \mid & !receive_c\ x\ from\ x'.P
\end{array}
$$

Language for threads

Variables $x \in \mathbf{Var}$. Constants / values $n, v \in \mathbb{Z}$.

Abstract channels $c \in \mathbf{Chan}$.

Expressions

$$
\begin{array}{rclclcl}
e & ::= & n & \mid & symkey() & \mid & asymkeypair() \\
& \mid & x & \mid & symenc(e,e) & \mid & asymenc(e,e) \\
& \mid & (e,\ldots,e) & \mid & symdec(e,e) & \mid & asymdec(e,e) \\
& \mid & \pi_i(e) & \mid & nonce() & \mid & pubkey(e) \\
& \mid & store(e) & \mid & retrieve(e) & &
\end{array}
$$

Processes

$$P ::= P_{\mathrm{act}} \mid P_{\mathrm{inact}}$$

$$P_{\mathrm{inact}} ::= (T_1 | \cdots | T_n) \mid \mathbf{Reject}$$

$$
\begin{array}{rcl}
P_{\mathrm{act}} & ::= & let\ x := e\ in\ P\ else\ P' \\
& \mid & if\ e = e\ then\ P\ else\ P' \\
& \mid & send_c\ e\ to\ e.P_{\mathrm{inact}}
\end{array}
$$

Threads

$$
\begin{array}{rcl}
T & ::= & receive_c\ x\ from\ x'.P \\
& \mid & !receive_c\ x\ from\ x'.P
\end{array}
$$

Language for threads

Variables $x \in \mathbf{Var}$. Constants / values $n, v \in \mathbb{Z}$.

Abstract channels $c \in \mathbf{Chan}$.

Expressions

$$
\begin{array}{rcllll}
e & ::= & n & \mid \; symkey() & \mid \; asymkeypair() \\
& \mid & x & \mid \; symenc(e,e) & \mid \; asymenc(e,e) \\
& \mid & (e,\ldots,e) & \mid \; symdec(e,e) & \mid \; asymdec(e,e) \\
& \mid & \pi_i(e) & \mid \; nonce() & \mid \; pubkey(e) \\
& \mid & store(e) & \mid \; retrieve(e) &
\end{array}
$$

Processes

$$P ::= P_{\mathrm{act}} \mid P_{\mathrm{inact}}$$

$$P_{\mathrm{inact}} ::= (T_1 | \cdots | T_n) \mid \mathbf{Reject}$$

$$
\begin{array}{rcl}
P_{\mathrm{act}} & ::= & let\ x := e\ in\ P\ else\ P' \\
& \mid & if\ e = e\ then\ P\ else\ P' \\
& \mid & send_c\ e\ to\ e.P_{\mathrm{inact}}
\end{array}
$$

Threads

$$
\begin{array}{rcl}
T & ::= & receive_c\ x\ from\ x'.P \\
& \mid & !receive_c\ x\ from\ x'.P
\end{array}
$$

Inspiration for the type system

- For cryptographic protocols, many type systems have been devised for
  - various security properties;
  - various language features;
  - various precision.
- These type systems are correct wrt. the term-rewriting-based semantics of cryptographic operations.
- The inner workings of the abstract cryptographic library are similar to the term-rewriting semantics.

Can some existing type system be modified, such that it will be correct wrt. the semantics defined by the abstract cryptographic library?

Can some existing type system be modified, such that it will be correct wrt. the semantics defined by the abstract cryptographic library?

Yes.

Can some existing type system be modified, such that it will be correct wrt. the semantics defined by the abstract cryptographic library?

Yes.

We use the type system of Abadi and Blanchet (FoSSaCS 2001).

Typings

- A typing Γ of a program or process or thread P assigns types to all abstract channels and free variables of P.
- The programs of all participants must type with respect to the same typing Γ.

Types

$$T ::= T^{(I)} \mid T^{(H)}$$

$$T^{(I)} ::= \underline{\mathrm{PubRD}} \mid \underline{\mathrm{SecRD}} \mid \underline{\mathrm{AllRD}}$$

$$T^{(H)} ::= T^{(A)} \mid \underline{\mathrm{DK}}(T^{(A)})$$

$$\begin{array}{rcll} T^{(A)} & ::= & \underline{\mathrm{Public}} & \mid \quad \underline{\mathrm{PubData}} \\ & \mid & \underline{\mathrm{SNonce}} & \mid \quad \underline{\mathrm{SecData}} \\ & \mid & \underline{\mathrm{EK}}(T^{(A)}) & \mid \quad \underline{\mathrm{SK}^i}(T^{(A)}) \\ & \mid & \underline{\mathrm{List}}(T_1^{(A)}, \ldots, T_n^{(A)}) & \mid \quad T_1^{(A)} + T_2^{(A)} \end{array}$$

Types

$$T ::= T^{(I)} \mid T^{(H)}$$

$$T^{(I)} ::= \underline{\mathrm{PubRD}} \mid \underline{\mathrm{SecRD}} \mid \underline{\mathrm{AllRD}}$$

$$T^{(H)} ::= T^{(A)} \mid \underline{\mathrm{DK}}(T^{(A)})$$

$$\begin{array}{rclcl} T^{(A)} & ::= & \underline{\mathrm{Public}} & \mid & \underline{\mathrm{PubData}} \\ & \mid & \underline{\mathrm{SNonce}} & \mid & \underline{\mathrm{SecData}} \\ & \mid & \underline{\mathrm{EK}}(T^{(A)}) & \mid & \underline{\mathrm{SK}}^{i}(T^{(A)}) \\ & \mid & \underline{\mathrm{List}}(T_1^{(A)}, \ldots, T_n^{(A)}) & \mid & T_1^{(A)} + T_2^{(A)} \end{array}$$

secrets received from the application layer

Types

$$T ::= T^{(I)} \mid T^{(H)}$$

$$T^{(I)} ::= \underline{\mathrm{PubRD}} \mid \underline{\mathrm{SecRD}} \mid \underline{\mathrm{AllRD}}$$

$$T^{(H)} ::= T^{(A)} \mid \underline{\mathrm{DK}}(T^{(A)})$$

$$\begin{array}{rcll} T^{(A)} & ::= & \underline{\mathrm{Public}} & \mid \ \underline{\mathrm{PubData}} \\ & \mid & \underline{\mathrm{SNonce}} & \mid \ \underline{\mathrm{SecData}} \\ & \mid & \underline{\mathrm{EK}}(T^{(A)}) & \mid \ \underline{\mathrm{SK}^i}(T^{(A)}) \\ & \mid & \underline{\mathrm{List}}(T_1^{(A)}, \ldots, T_n^{(A)}) & \mid \ T_1^{(A)} + T_2^{(A)} \end{array}$$

names of participants (as addresses)

Types

$$T ::= T^{(I)} \mid T^{(H)}$$

$$T^{(I)} ::= \underline{\mathrm{PubRD}} \mid \underline{\mathrm{SecRD}} \mid \underline{\mathrm{AllRD}}$$

$$T^{(H)} ::= T^{(A)} \mid \underline{\mathrm{DK}}(T^{(A)})$$

$$\begin{array}{rclcl} T^{(A)} & ::= & \underline{\mathrm{Public}} & \mid & \underline{\mathrm{PubData}} \\ & \mid & \underline{\mathrm{SNonce}} & \mid & \underline{\mathrm{SecData}} \\ & \mid & \underline{\mathrm{EK}}(T^{(A)}) & \mid & \underline{\mathrm{SK}}^{i}(T^{(A)}) \\ & \mid & \underline{\mathrm{List}}(T_1^{(A)}, \ldots, T_n^{(A)}) & \mid & T_1^{(A)} + T_2^{(A)} \end{array}$$

$store()$-d $\underline{\mathrm{PubRD}}$

$store()$-d $\underline{\mathrm{SecRD}}$

Types

$$T ::= T^{(I)} \mid T^{(H)}$$

$$T^{(I)} ::= \underline{\mathrm{PubRD}} \mid \underline{\mathrm{SecRD}} \mid \underline{\mathrm{AllRD}}$$

$$T^{(H)} ::= T^{(A)} \mid \underline{\mathrm{DK}}(T^{(A)})$$

$$\begin{array}{rcll} T^{(A)} & ::= & \underline{\mathrm{Public}} & \mid \ \underline{\mathrm{PubData}} \\ & \mid & \underline{\mathrm{SNonce}} & \mid \ \underline{\mathrm{SecData}} \\ & \mid & \underline{\mathrm{EK}}(T^{(A)}) & \mid \ \underline{\mathrm{SK}}^{i}(T^{(A)}) \\ & \mid & \underline{\mathrm{List}}(T_1^{(A)}, \ldots, T_n^{(A)}) & \mid \ T_1^{(A)} + T_2^{(A)} \end{array}$$

Keys to encrypt data of type $T^{(A)}$.
The type of ciphertext is $\underline{\mathrm{Public}}$.

Types

$$T ::= T^{(I)} \mid T^{(H)}$$

$$T^{(I)} ::= \underline{\mathrm{PubRD}} \mid \underline{\mathrm{SecRD}} \mid \underline{\mathrm{AllRD}}$$

$$T^{(H)} ::= T^{(A)} \mid \underline{\mathrm{DK}}(T^{(A)})$$

$$\begin{array}{rcll} T^{(A)} & ::= & \underline{\mathrm{Public}} & \mid \ \underline{\mathrm{PubData}} \\ & \mid & \underline{\mathrm{SNonce}} & \mid \ \underline{\mathrm{SecData}} \\ & \mid & \underline{\mathrm{EK}}(T^{(A)}) & \mid \ \underline{\mathrm{SK}}^{i}(T^{(A)}) \\ & \mid & \underline{\mathrm{List}}(T_1^{(A)}, \ldots, T_n^{(A)}) & \mid \ T_1^{(A)} + T_2^{(A)} \end{array}$$

Keys to decrypt ciphertexts, if the plaintext has type $T^{(A)}$

Types

$$T ::= T^{(I)} \mid T^{(H)}$$

$$T^{(I)} ::= \underline{\mathrm{PubRD}} \mid \underline{\mathrm{SecRD}} \mid \underline{\mathrm{AllRD}}$$

$$T^{(H)} ::= T^{(A)} \mid \underline{\mathrm{DK}}(T^{(A)})$$

$$\begin{array}{lcll} T^{(A)} & ::= & \underline{\mathrm{Public}} & \mid \underline{\mathrm{PubData}} \\ & \mid & \underline{\mathrm{SNonce}} & \mid \underline{\mathrm{SecData}} \\ & \mid & \underline{\mathrm{EK}}(T^{(A)}) & \mid \underline{\mathrm{SK}}^{i}(T^{(A)}) \\ & \mid & \underline{\mathrm{List}}(T_1^{(A)}, \ldots, T_n^{(A)}) & \mid T_1^{(A)} + T_2^{(A)} \end{array}$$

The order of the symmetric key. It may only encrypt messages containing (visible) symmetric keys of smaller order.

Types

$$T ::= T^{(I)} \mid T^{(H)}$$

$$T^{(I)} ::= \underline{\mathrm{PubRD}} \mid \underline{\mathrm{SecRD}} \mid \underline{\mathrm{AllRD}}$$

$$T^{(H)} ::= T^{(A)} \mid \underline{\mathrm{DK}}(T^{(A)})$$

$$\begin{array}{rclcl} T^{(A)} & ::= & \underline{\mathrm{Public}} & \mid & \underline{\mathrm{PubData}} \\ & \mid & \underline{\mathrm{SNonce}} & \mid & \underline{\mathrm{SecData}} \\ & \mid & \underline{\mathrm{EK}}(T^{(A)}) & \mid & \underline{\mathrm{SK}^{i}}(T^{(A)}) \\ & \mid & \underline{\mathrm{List}}(T_1^{(A)}, \ldots, T_n^{(A)}) & \mid & T_1^{(A)} + T_2^{(A)} \end{array}$$

Ordering on types

- $\underline{\mathrm{AllRD}}$ is the least upper bound of $\underline{\mathrm{SecRD}}$ and $\underline{\mathrm{PubRD}}$;
- $T_1 + T_2$ is the least upper bound of T_1 and T_2;
- $\underline{\mathrm{List}}$ is covariant wrt. its arguments;
- $\underline{\mathrm{PubData}} \leq \underline{\mathrm{Public}}$. $\underline{\mathrm{EK}}(\ldots) \leq \underline{\mathrm{Public}}$.
- $\underline{\mathrm{List}}(\underline{\mathrm{Public}}, \ldots, \underline{\mathrm{Public}}) \leq \underline{\mathrm{Public}}$.

Conditions on Γ(channel)

- $\Gamma(c)$ is a type $T^{(H)}$.
- $\Gamma(c)$ is the type of messages exchanged over the abstract channel c.
- If c is an authentic channel then $\Gamma(c) \leq \underline{\mathrm{Public}}$.
- If c is an insecure channel then $\Gamma(c) = \underline{\mathrm{Public}}$.
- If c is a channel between the participant and the application layer then $\Gamma(c{\downarrow}) = \underline{\mathrm{SecRD}}$ and $\Gamma(c{\uparrow}) = \underline{\mathrm{AllRD}}$.

Conclusions

- It is possible to devise mechanisms for automated analysis of protocols, if the cryptographic operations are implemented by the simulatable cryptographic library.
  - The one that we presented should be treated just as a proof of concept.
- The resulting formalisms are no more complex than those targeted at the term-rewriting-based semantics of cryptographic protocols.
  - ...and we get the correctness wrt. to the computational semantics for free.
- If you intend to use the term-rewriting semantics as the formal foundation of your tool, then please consider using the simulatable cryptographic library instead.

Conclusions

- It is possible to devise mechanisms for automated analysis of protocols, if the cryptographic operations are implemented by the simulatable cryptographic library.
  - The one that we presented should be treated just as a proof of concept.
- The resulting formalisms are no more complex than those targeted at the term-rewriting-based semantics of cryptographic protocols.
  - . . . and we get the correctness wrt. to the computational semantics for free.
- If you intend to use the term-rewriting semantics as the formal foundation of your tool, then please consider using the simulatable cryptographic library instead.