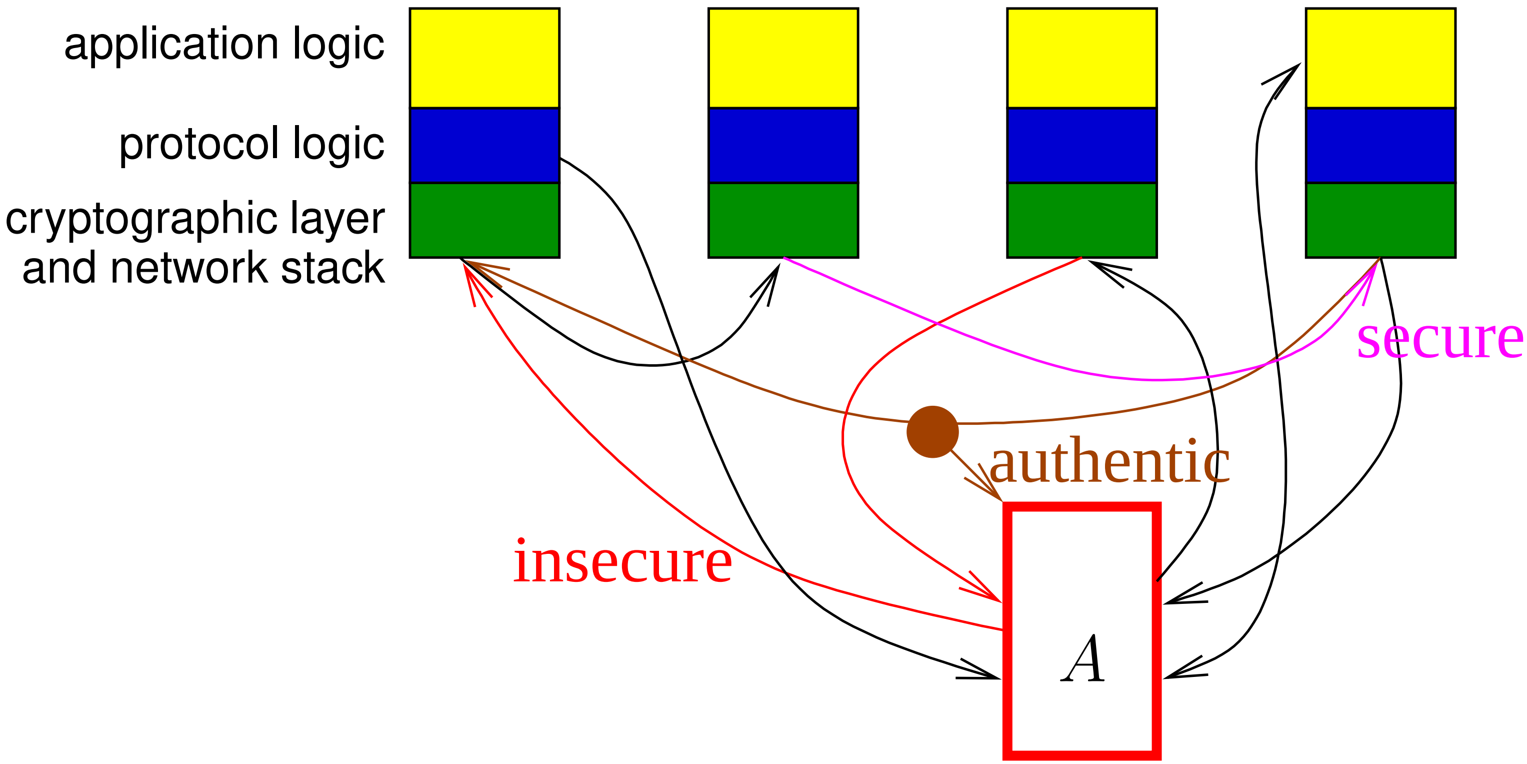
Computationally Sound Secrecy Proofs by Mechanized Flow Analysis

Michael Backes   Peeter Laud

Saarland University   Tartu University

A distributed system. . .

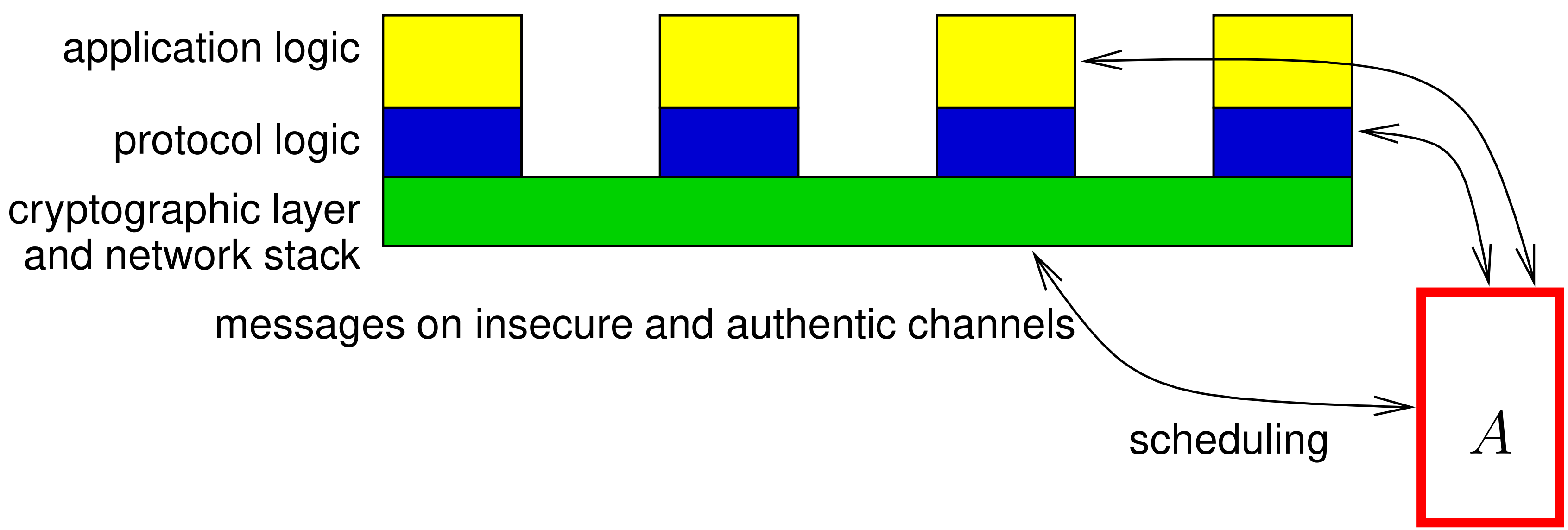
. . . can be modeled as



The simulatable cryptographic library

- May serve as the cryptographic layer / network stack.
- 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.

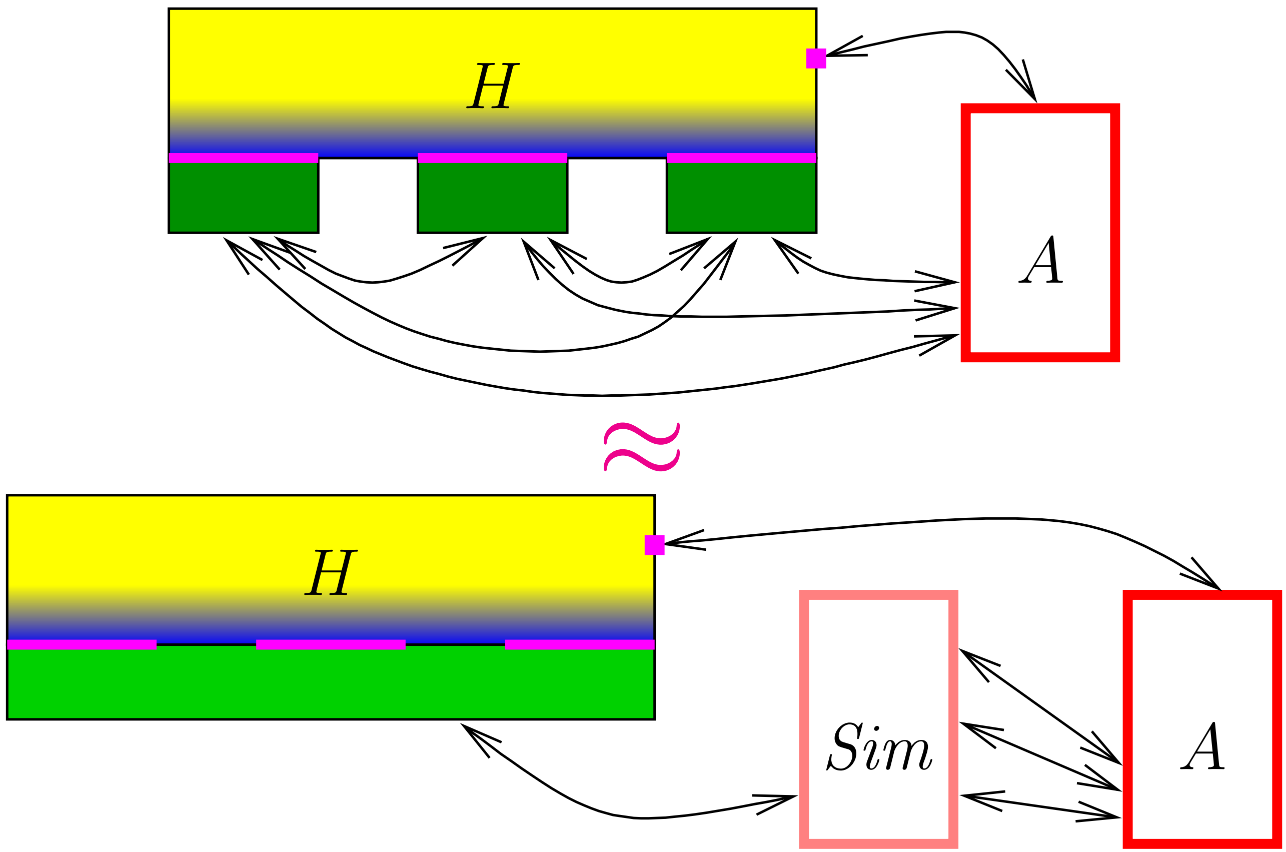
Main part — a database of terms recording their structure and parties that have access to them.

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

Possible operations also similar to the Dolev-Yao model.

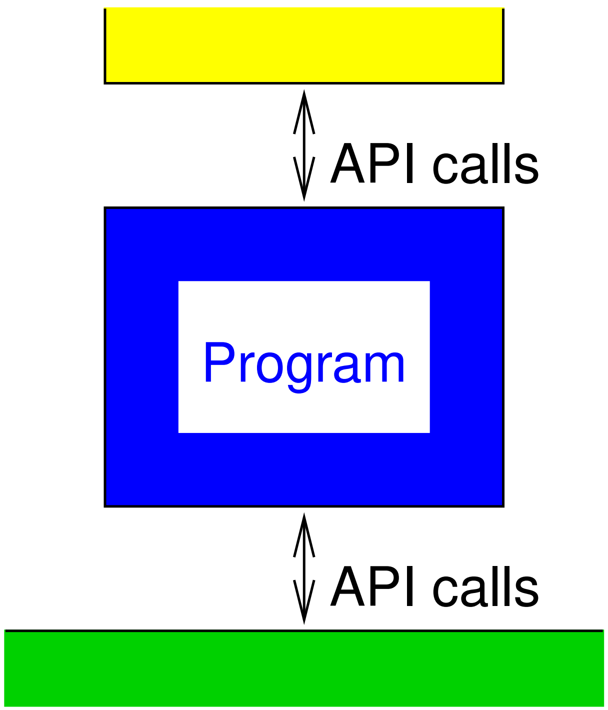
Simulatability

$\exists\, Sim$, 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.

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.

Simulation also requires

- No encryption cycles
- A symmetric key used by a participant does not become known to the adversary

Language for threads

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

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

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

Expressions

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

Processes

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

$$P_{\mathrm{inact}} ::= T_1 | \cdots | T_n$$

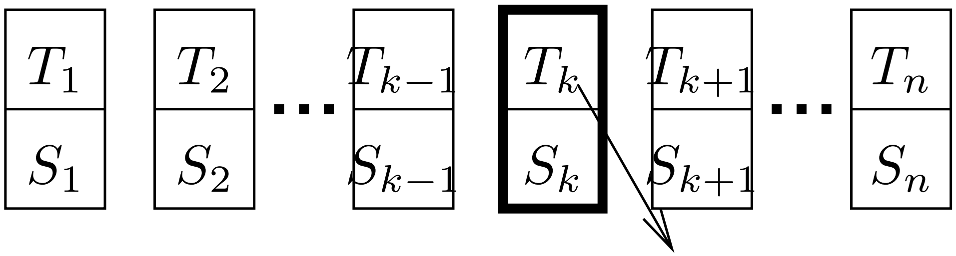
$$
\begin{array}{lcl}
P_{\mathrm{act}} & ::= & let\ x := e\ in\ P\ else\ P' \\
& \mid & if\ e = e\ then\ P\ else\ P' \\
& \mid & send_c\ e\ to\ e.P_{\mathrm{inact}}
\end{array}
$$

Threads

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

Program: $T_1 \mid \cdots \mid T_n$

Handling message to a thread

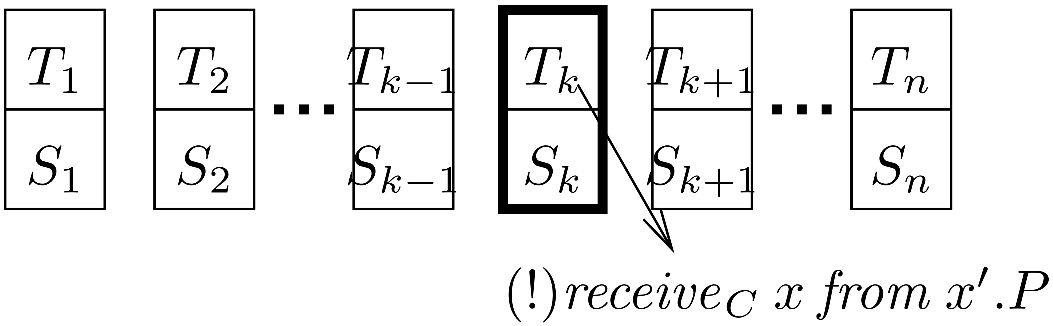


$(!)\mathit{receive}_c\ x\ \mathit{from}\ x'.P$

Message	M
Sender	Y
Abstr. channel	C

Compare c and C. If $c = C$ then…

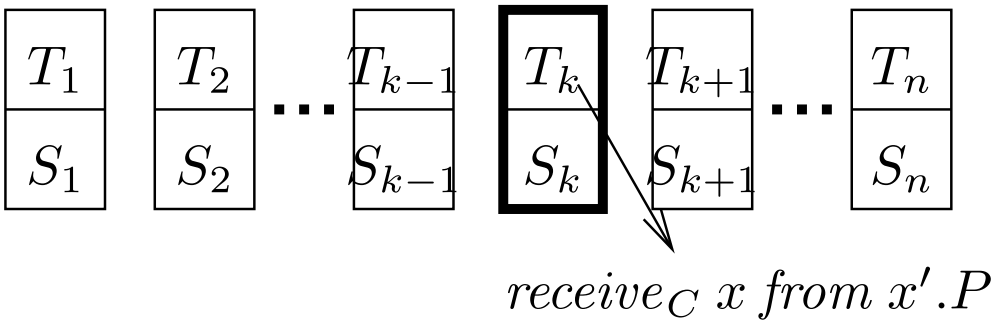
Starting the execution of a thread



Message	M
Sender	Y
Abstr. channel	C

Execute: $\boxed{P \mid S_k[x \mapsto M, x' \mapsto Y]} \longrightarrow \cdots$

Normal end of execution

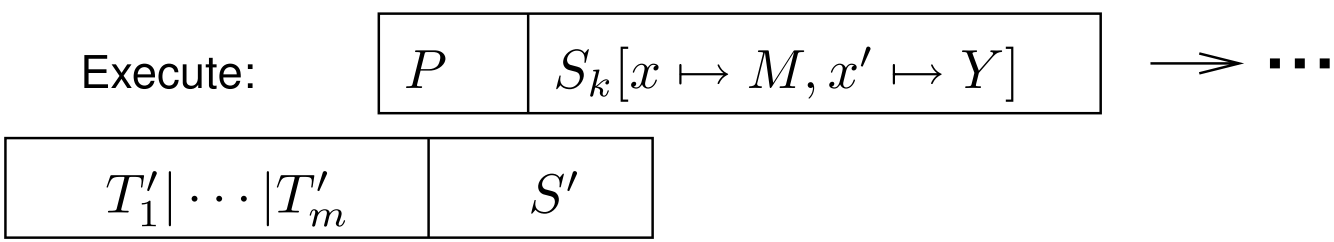


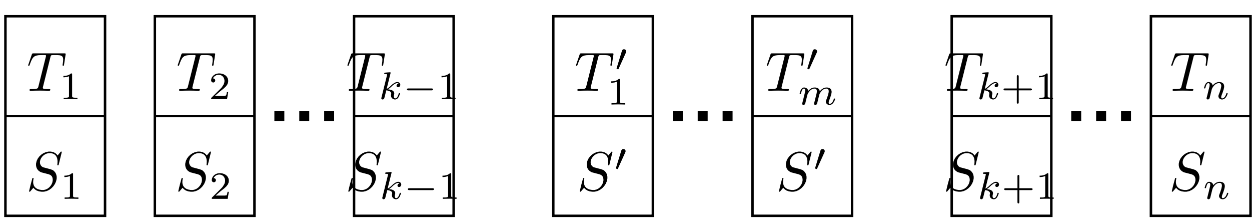
Message M

Sender Y

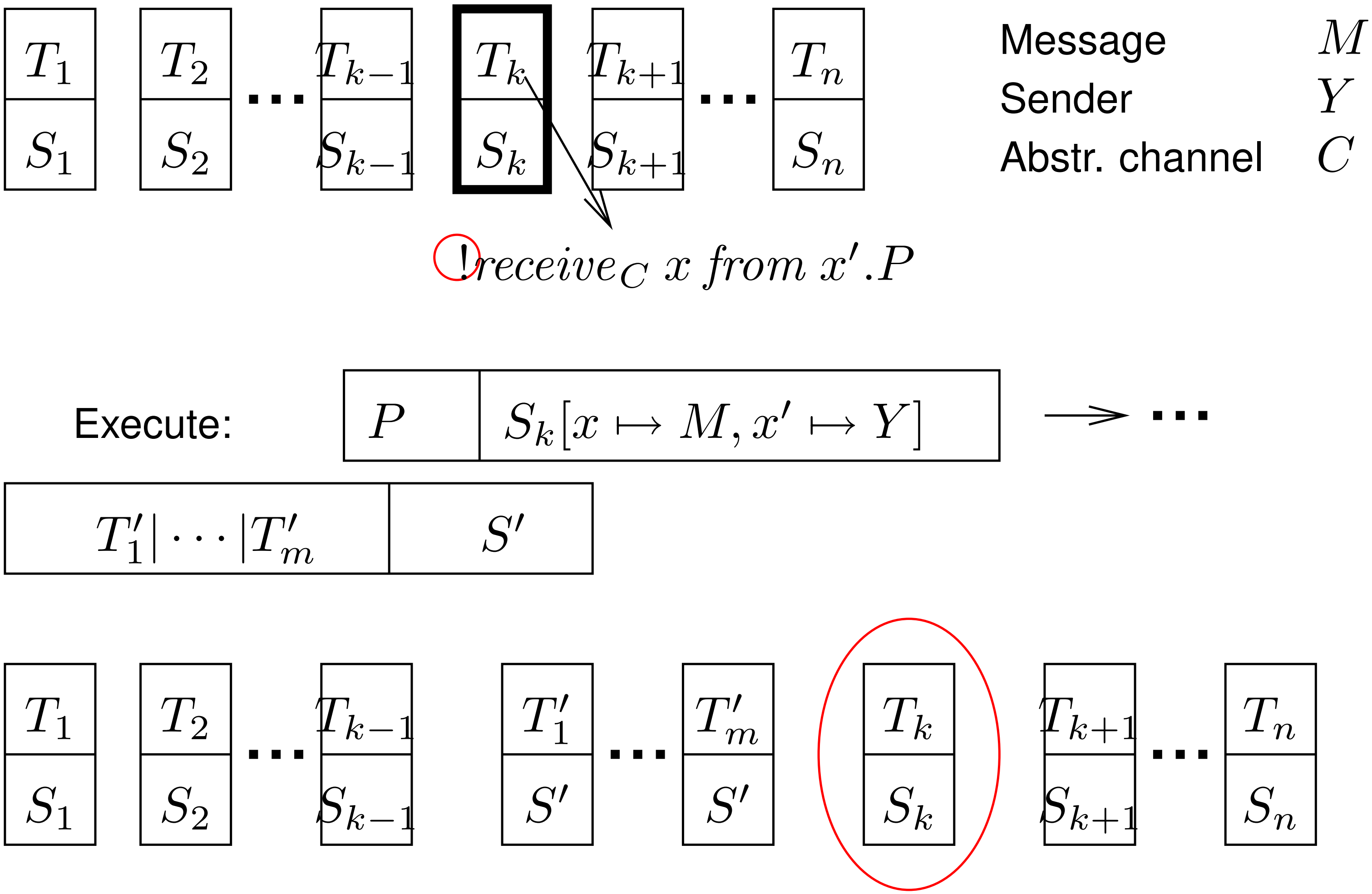
Abstr. channel C

Execute:





Normal end of execution



Abnormal end of execution

T_1	T_2	$\ldots$	T_{k-1}	$\boxed{T_k}$	T_{k+1}	$\ldots$	T_n
S_1	S_2	$\ldots$	S_{k-1}	$\boxed{S_k}$	S_{k+1}	$\ldots$	S_n

$(!)\mathit{receive}_C\ x\ \mathit{from}\ x'.P$

Message	M
Sender	Y
Abstr. channel	C

Execute:

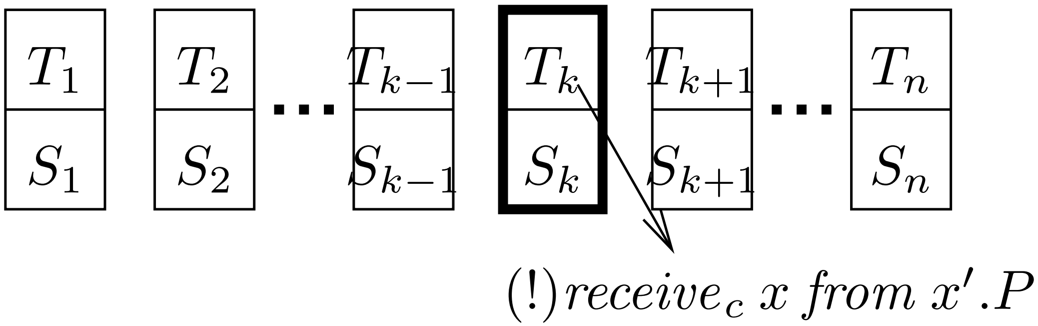
P	$S_k[x \mapsto M, x' \mapsto Y]$

$\longrightarrow \ldots$

Reject	S'

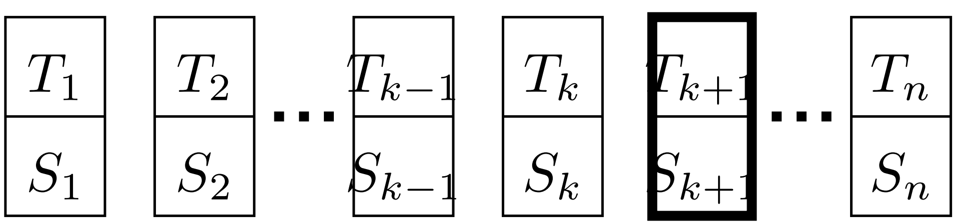
T_1	T_2	$\ldots$	T_{k-1}	T_k	$\boxed{T_{k+1}}$	$\ldots$	T_n
S_1	S_2	$\ldots$	S_{k-1}	S_k	$\boxed{S_{k+1}}$	$\ldots$	S_n

Handling message to a thread



Message	M
Sender	Y
Abstr. channel	C

Compare c and C. If $c \neq C$ then



Analysis: Labels for interesting points

Expressions

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

Processes

$$P ::= P_{\text{act}} \mid P_{\text{inact}} \mid \mathbf{Reject}$$

$$P_{\text{inact}} ::= T_1 | \cdots | T_n$$

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

Threads

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

Program: $T_1 \mid \cdots \mid T_n$

Goal of the analysis

- For each variable, collect the values (“terms”) that can be stored in that variable.
- May also depend on the program point.
- We’ll set up a system of constraints.
- It includes a variable S_ℓ for each program point.
- Possible values of S_ℓ: mappings
  - from variables defined at ℓ
  - to abstractions of sets of terms.

Abstract values

$$AV ::= AV_I \mid AV_H \mid \mathsf{seckey}(\ell) \qquad AV_I = \mathsf{X_P} \mid \mathsf{X_S}$$

$$\begin{array}{rcll}
AV_H & ::= & \mathsf{store}(AV_I) & \mid \ \mathsf{nonce}(\ell) \\
& \mid & \mathsf{symkey}(i,\ell) & \mid \ \mathsf{symkeyname}(\ell) \\
& \mid & \mathsf{AnyPubVal} & \mid \ \mathsf{pubkey}(\ell) \\
& \mid & (AV_H, \ldots, AV_H) & \mid \ \mathsf{pubenc}(AV_H, AV_H, \ell) \\
& \mid & \mathsf{symenc}(AV_H, AV_H, \ell) &
\end{array}$$

- Set of AV — abstraction of a set of terms.
- AnyPubVal corresponds to all terms that the adversary knows.
  - . . . or may know without further interaction.

Constraints for assignments

- Let ℓ_0 be the label directly above the current statement. Let $\mathbf{I} = \mathbf{S}_{\ell_0}$.
- $let^{\ell}\ x := C^{(\ell')}(x_1, \ldots, x_k)$ generates

$$\mathbf{S}_\ell \geq \mathbf{I}[x \mapsto \{C(v_1, \ldots, v_k(, \ell')) \,|\, v_i \in \mathbf{I}(x_i)\}]$$

- $let^{\ell}\ x := D(y)$, where D deconstructs C, generates

$$\mathbf{S}_\ell \geq \mathbf{I}[x \mapsto \{v \,|\, C(\ldots v \ldots) \in \mathbf{I}(y)\}]$$

- Projecting from AnyPubVal results in AnyPubVal.

Decrypting AnyPubVal?

- For each $asymkeypair^{\ell}()$ and $symkey^{\ell}(i)$ we have a constraint variable $\mathbf{E}_{\ell}$.
- It collects the set of terms that may be encrypted with the key created at ℓ.
- $let\ x := (a)symenc(k, y)$ also generates for each $\mathbf{E}_{\ell}$

$$\mathsf{key}(\ell) \in \mathbf{I}(k) \Rightarrow \mathbf{E}_{\ell} \supseteq \mathbf{I}(y)$$

- $let^{\ell'}\ y := symdec(k, x)$ also generates for each $\mathbf{E}_{\ell}$

$$\mathsf{symkey}(\ldots, \ell) \in \mathbf{I}(k) \wedge \mathsf{AnyPubVal} \in \mathbf{I}(x) \Rightarrow \mathbf{S}_{\ell'}(y) \supseteq \mathbf{E}_{\ell}$$

and also

$$\mathbf{I}(k) \dot{\cap} \{\mathsf{AnyPubVal}\} \neq \emptyset \Rightarrow \mathsf{AnyPubVal} \in \mathbf{S}_{\ell'}(y)$$

Liveness

- For all labels ℓ of *if*- and *let*-statements we have constraint variables $\mathbf{L}_{\ell,\mathsf{true}}$ and $\mathbf{L}_{\ell,\mathsf{false}}$.
- Possible values: false and true with $\mathsf{false} \leq \mathsf{true}$.
- $if^{\ell}\ x = x'\ then \ldots$ generates

$$\mathbf{L}_{\ell_0} \wedge \mathbf{I}(x) \mathbin{\dot{\cap}} \mathbf{I}(x') \neq \emptyset \Rightarrow \mathbf{L}_{\ell,\mathsf{true}}$$

- Some *let*-statements always fail, too.
- We almost always generate $\mathbf{L}_{\ell_0} \wedge \mathbf{L}_{\ell,\mathsf{false}}$.
- All constraints in previous slides also check whether $\mathbf{L}_{\ell,\mathsf{true}}$ is true.

Communication

- Each abstract channel name c has an associated security level: secure, authentic, insecure, from/to the user.
- We have a constraint variable C_c for each secure or authentic abstract channel c.
  - It collects the terms flowing over the channel c.
- We have a constraint variable P.
  - It collects the terms that the adversary may know.

Sending and receiving

- $send_c\ x\ over\ z$ generates $\mathbf{P} \supseteq \mathbf{I}(z)$ and also
  - $\mathbf{C}_c \supseteq \mathbf{I}(x)$ if c is secure or authentic;
  - $\mathbf{P} \supseteq \mathbf{I}(x)$ if c is authentic or insecure.
- $(!)receive_c^\ell\ x\ from\ z$ generates $\mathbf{S}_\ell \geq \mathbf{I}[z \mapsto \{\mathsf{X_P}\}]$ and also
  - $\mathbf{S}_\ell(x) \supseteq \mathbf{C}_c$ if c is secure or authentic;
  - $\mathbf{S}_\ell(x) \supseteq \mathbf{P}$ if c is insecure;
  - $\mathbf{S}_\ell(x) \supseteq \{\mathsf{X_S}\}$ if c is from the user.
- ... and liveness checks are there, too.

Adversary's computation

$$\mathsf{store}(AV) \in \mathbf{P} \Rightarrow AV \in \mathbf{P}$$

$$(AV_1, \ldots, AV_j) \in \mathbf{P} \Rightarrow AV_i \in \mathbf{P}$$

$$\mathsf{symenc}(AV_k, AV_t, \ell) \in \mathbf{P} \Rightarrow (\exists AV' \in \mathbf{P} : AV_k \cong_{\mathbf{P}} AV') \Rightarrow AV_t \in \mathbf{P}$$

$$\mathsf{pubenc}(\mathsf{AnyPubVal}, AV_t, \ell) \in \mathbf{P} \Rightarrow AV_t \in \mathbf{P}$$

$$\mathsf{pubenc}(AV_k, AV_t, \ell) \in \mathbf{P} \Rightarrow AV_k \in \mathbf{P}$$

$$\mathsf{symenc}(\mathsf{symkey}(i, \ell), AV_t, \ell') \in \mathbf{P} \Rightarrow \mathsf{symkeyname}(\ell) \in \mathbf{P}$$

$$\{\mathsf{X_P}, \mathsf{AnyPubVal}\} \subseteq \mathbf{P}$$

$AV \cong_{\mathbf{P}} AV'$ if the abstract terms AV and AV' may denote the same concrete term.

Public-key decryption

- In $x := \ asymdec(k, y)$ there are two possibilities:
  - x was created inside the protocol;
  - x was created by the adversary.
- We analyse those two cases separately.

Public-key decryption

- In $x := \ asymdec(k, y)$ there are two possibilities:
  - x was created inside the protocol;
  - x was created by the adversary.
- We analyse those two cases separately.
- Let ℓ be a label of some if or let.
- Let n be the number of public-key decryptions before ℓ (including let^ℓ itself).
- The constraint system includes variables S_ℓ^b where $b \in \{0,1\}^n$.
- They record the abstractions of variables in case that the result of the i-th public-key decryption was generated by b_i
  - 1 — the protocol participants; 0 — the adversary.

The labels b

- We also have constraint variables $\mathbf{E}_\ell^b$, $\mathbf{L}_{\ell,\ldots}^b$.
- And abstract values: $\mathsf{nonce}(\ell, b)$, $\mathsf{symkey}(i, \ell, b)$, $\mathsf{pubenc}(AV_k, AV_t, \ell, b)$, etc.
- But we still have constraint variables $\mathbf{C}_c$ and $\mathbf{P}$.
- $let^\ell\ x := \ asymdec(k, y)$ generates

$$\mathsf{AnyPubVal} \in \mathbf{I}^b(y) \wedge \mathsf{seckey}(\ell', b') \in \mathbf{I}^b(k) \Rightarrow \mathbf{S}_\ell^{b1}(x) \supseteq \mathbf{E}_{\ell'}^{b'}$$

$$\mathsf{AnyPubVal} \in \mathbf{I}^b(y) \Rightarrow \mathsf{AnyPubVal} \in \mathbf{S}_\ell^{b0}(x) \ .$$

- The size of the analysis blows up exponentially. But it is still small.

Implementation

- Generate the constraints, solve them, check that the secrecy conditions hold.
- Solving — we use an iterative solver by Fecht and Seidl.
- Solving may diverge in theory. But such protocols do not occur in practice.
- Secrecy conditions:
  - $\mathsf{X_S} \notin \mathbf{P}$;
  - if $\mathsf{X_S} \in \mathbf{S}^b_\ell(x)$ and x is used at ℓ then x is either stored as payload or returned to the user;
  - No encryption cycles occur in abstract values;
  - $\mathsf{seckey}(\ldots) \notin \mathbf{P}$.
- Speed: couple of seconds per protocol on a couple of years old PC.

Relationships between variables

- $\mathit{if}\ x = x'$ creates relations between the values of x and x'.
- $\mathit{let}\ x := E(x_1, \ldots, x_k)$ creates relations between the values of x and $x_1, \ldots, x_k$.
- We may record them as constraints.
- For $\mathit{if}\ x = x'$:

$$\mathbf{X}(x) \mathrel{\dot{\subseteq}} \mathbf{X}(x') \quad \text{and} \quad \mathbf{X}(x') \mathrel{\dot{\subseteq}} \mathbf{X}(x)$$

- Let $\mathcal{C}^b_\ell$ be the set of constraints after the program point ℓ.
- Let $\mathfrak{L}(\mathcal{C}, \mathbf{V})$ be the greatest mapping from variables to sets of abstract values that
  - is less than or equal to $\mathbf{V}$;
  - satisfies the constraints in $\mathcal{C}$.

Constraints expressing the relationships

- Let $\mathcal{C}_I$ be the incoming constraints of a if- or let-statement.
- The constraints for the $else$-branch are $\mathcal{C}_I$.
- The constraints for the default-branch are $\mathcal{C}_I$ and
  - $\mathbf{X}(x) \dot{\subseteq} \mathbf{X}(x')$ and $\mathbf{X}(x') \dot{\subseteq} \mathbf{X}(x)$ for $if\ x = x'$;
  - $\mathbf{X}(x) \dot{\subseteq} \{(AV_1, \ldots, AV_k) \,|\, AV_i \in \mathbf{X}(x_i))\}$ and $\mathbf{X}(x_i) \dot{\subseteq} \{AV_i \,|\, (AV_1, \ldots, AV_k) \in \mathbf{X}(x)\}$ for $let\ x := (x_1, \ldots, x_k)$;
  - $\mathbf{X}(x) \dot{\subseteq} \{(\mathsf{Anything}, \ldots, AV_i, \ldots, \mathsf{Anything}) \,|\, AV_i \in \mathbf{X}(x_i))\}$ and $\mathbf{X}(x_i) \dot{\subseteq} \{AV_i \,|\, (AV_1, \ldots, AV_k) \in \mathbf{X}(x)\} \cup \{\mathsf{AnyPubVal} \,|\, \mathsf{AnyPubVal} \in \mathbf{X}(x)\}$ for $let\ x_i := \pi_i^k(x)$;
  - etc.
- This defines the constraints $\mathcal{C}_\ell^b$ for all ℓ and b.

Using those constraints

- For all $\mathbf{S}_\ell^b$ let $\mathbf{R}_\ell^b$ be an auxiliary constraint variable.
- Add the constraints $\mathbf{R}_\ell^b \geq \mathfrak{L}(\mathcal{C}_\ell^b, \mathbf{S}_\ell^b)$.
- In previous slides, the variable $\mathbf{I}$ is one of the variables $\mathbf{R}_\ell^b$.
- Implementation: the same constraint solver is used to evaluate $\mathfrak{L}$.

Conclusions

- It is possible to devise mechanisms for automated analysis of protocols, if the cryptographic operations are implemented by the simulatable cryptographic library.
- 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.