Security proofs in the symbolic model
bounded session verification, type systems

Karthikeyan Bhargavan

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

October 2013
Recap: Towards automated security proofs

- A decidability result (and algorithm) for attacker deduction
  - Deduction is PTIME complete
- A decidability result (and algorithm scheme) for protocol security for bounded number of sessions
  - Bounded protocol security is NP complete
- An undecidability result for unbounded number of sessions
  - Proof techniques that require user input or allow non-termination
Recap: Inference systems

- An inference rule is of the form

\[ M_1, \ldots, M_n \vdash M \]

- An inference system is a set of rules

- We rewrite the attacker’s deduction rules eliminating destructors:

\[
\begin{align*}
& x, y \vdash \langle x, y \rangle & x, y \vdash \text{hash}(x) \\
& x, y \vdash \text{aenc}(x)_y & x, y \vdash \text{senc}(x)_y \\
& x, y \vdash \text{sig}\{x\}_y & x, y \vdash \text{hmac}(x, y) \\
& \langle x, y \rangle \vdash x & \langle x, y \rangle \vdash y \\
& \text{aenc}(x)_y, \text{sk}(y) \vdash x & \text{senc}(x)_y, y \vdash x
\end{align*}
\]
Recap: Proofs

- Assume an inference system $I$ and a set of terms $T$
- A proof $\Pi$ of $T \vdash M$ in $I$ is a tree such that
  - the root is labeled $M$
  - every leaf is a term in $T$
  - every node has a result (label) $N$ and hypotheses $N_1, \ldots, N_n$, such that $N_1, \ldots, N_n \vdash N$ is an instance of an inference rule in $I$
- Example: A proof that $\langle \text{senc}(s)_{\langle a, b \rangle}, a \rangle, \text{senc}(b)_{a} \vdash s$ is

\[
\frac{
\frac{
\frac{
\langle \text{senc}(s)_{\langle a, b \rangle}, a \rangle
\text{senc}(s, \langle a, b \rangle)

\text{senc}(b, a)

}{a}

}{b}

}{\langle a, b \rangle}

}{s}
\]
Recap: Locality

- Inference system $I$ is *local* if whenever $T \vdash M$, there exists a proof $\Delta$ such that every conclusion in $\Delta$ is a subterm of a term in $T \cup \{M\}$.
  - Hence, in searching for a proof, one never need look outside the subterms of the known $T$ or the target $M$.
  - There is only a polynomial number of such subterms.

- *One-step deducibility*: test whether $T \vdash M$ in one inference step.
- Given locality and one-step deducibility, we obtain decidability.
- If one-step deducibility is in PTIME, so is intruder deduction.
Recap: Proving Locality

- **Exercise:** Prove that our attacker inference system is local

\[
\begin{align*}
x, y & \vdash \langle x, y \rangle & x, y & \vdash \text{hash}(x) \\
x, y & \vdash \text{aenc}(x)_y & x, y & \vdash \text{senc}(x)_y \\
x, y & \vdash \text{sig}\{x\}_y & x, y & \vdash \text{hmac}(x, y) \\
\langle x, y \rangle & \vdash x & \langle x, y \rangle & \vdash y \\
\text{aenc}(x)_y, \text{sk}(y) & \vdash x & \text{senc}(x)_y, y & \vdash x
\end{align*}
\]

- We only consider minimal proofs
- We show that any minimal proof that is a leaf or ends with a destructor uses only subterms of \( T \) (it does need to use \( M \))
- Proof is by induction on the height of the proof and involves case analysis on the last two rules
Summary: Intruder deduction

- For local inference systems, where one-step deducibility is decidable, the intruder deduction problem is decidable.

**Algorithm:** $T \vdash M$

- Reduce all terms in $T$ using only destructor rules to obtain a fixpoint set $F$ of subterms of $T$.
- Apply only constructor rules to $F$ to obtain a fixpoint set $G$ of subterms of $M$.
- If $M \in G$ return true, otherwise return false.

**Exercise:** Show that $\langle \text{senc}(s)_{\langle a, b \rangle}, a \rangle, \text{senc}(b)_a \vdash s$

**Exercise:** Show that $n, t, si, \text{hmac}(\text{utf8}(si), \text{hash}(\langle p, \langle n, t \rangle \rangle)) \not\vdash p$
Deducibility constraints

- A way of encoding bounded session protocols as deduction conditions

\[ T_1 \models M_1 \land T_2 \models M_2 \land \cdots \land T_n \models M_n \]

- Informally, protocol is a game between its principals and an attacker:
  \( T_i \) is the set of messages sent to the attacker at step \( i \) of protocol,
  \( M_i \) is a message that the attacker must produce to move to step \( i + 1 \)
Deducibility constraints

- A way of encoding bounded session protocols as deduction conditions:

\[ T_1 \vdash M_1 \land T_2 \vdash M_2 \land \cdots \land T_n \vdash M_n \]

- Informally, protocol is a game between its principals and an attacker: 
  \( T_i \) is the set of messages sent to the attacker at step \( i \) of protocol, 
  \( M_i \) is a message that the attacker must produce to move to step \( i+1 \)

- **Monotonicity**: for each \( i \), \( T_i \subseteq T_{i+1} \)
  The attacker’s knowledge can only increase
Deducibility constraints

- A way of encoding bounded session protocols as deduction conditions

\[
T_1 \vdash M_1 \land T_2 \vdash M_2 \land \cdots \land T_n \vdash M_n
\]

- Informally, protocol is a game between its principals and an attacker: 
  \( T_i \) is the set of messages sent to the attacker at step \( i \) of protocol, 
  \( M_i \) is a message that the attacker must produce to move to step \( i + 1 \)

- **Monotonicity**: for each \( i \), \( T_i \subseteq T_{i+1} \)
The attacker’s knowledge can only increase

- **Origination**: for each \( i \), \( \text{fv}(T_{i+1}) \subseteq \text{fv}(M_1, \ldots, M_i) \)
  Variables in \( M_i \) represent values chosen by attacker
  The protocol cannot add new variables, but it can create new names, 
  and construct concrete messages from attacker variables.
Assume an inference system $I$, and a protocol encoded as a deducibility constraint system $C$:

$$T_1 \vdash M_1 \land T_2 \vdash M_2 \land \cdots \land T_n \vdash M_n$$

A substitution $\sigma$ (representing the attackers choice of messages) is a solution of $C$ in $I$, if there exists a proof of $T_i \sigma \vdash M_i \sigma$ for each $i$.

Informally, the attacker must produce a sequence of ground terms $M_i$ using only the data he has received up to that point.
Solving deducibility constraints

- Assume an inference system $I$, and a protocol encoded as a deducibility constraint system $C$:

$$T_1 ? M_1 \land T_2 ? M_2 \land \cdots \land T_n ? M_n$$

- A substitution $\sigma$ (representing the attackers choice of messages) is a solution of $C$ in $I$, if there exists a proof of $T_i \sigma \vdash M_i \sigma$ for each $i$.

- Informally, the attacker must produce a sequence of ground terms $M_i$ using only the data he has received up to that point.

- **Functional adequacy**: If $C$ has a solution, then the protocol can proceed to completion.

- **Secrecy**: If $C \land T_n ? s$ has a solution then $s$ is not secret.
Consider the secure RPC protocol from Lecture 1:

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

Assume that \(A\) is willing to make a single RPC to \(B\)
Assume that \(B\) is willing to accept RPCs both from \(A\) and from the attacker \(X\)
Encode this protocol as a deduction constraint system
Prove functional adequacy
Prove the non-secrecy of response
Can you prove the secrecy of request?
We develop a set of simplification rules that transform a deducibility constraint system into a simpler (but equivalent) system.

We apply a sequence of simplifications until we obtain a system where deciding solvability is trivial.

\[
\begin{align*}
\text{R}_1 & \quad C \land T \models u \leadsto C & \text{if } T \cup \{x \mid (T' \models x) \in C, T' \subsetneq T\} \not\models u \\
\text{R}_2 & \quad C \land T \models u \leadsto_\sigma C \land T \models u_\sigma & \text{if } t \in \text{st}(T), \sigma = \text{mgu}(t, u), t \neq u \\
& & \text{t, u not variables} \\
\text{R}_3 & \quad C \land T \models u \leadsto_\sigma C \land T \models u_\sigma & \text{if } t_1, t_2 \in \text{st}(T), \sigma = \text{mgu}(t_1, t_2), \text{ and } t_1 \neq t_2 \\
\text{R}_4 & \quad C \land T \models u \leadsto \bot & \text{if } \text{fv}(T \cup \{u\}) = \emptyset \text{ and } T \not\models u \\
\text{R}_f & \quad C \land T \models f(u, v) \leadsto C \land T \models u \land T \models v & \text{for } f \in \{\langle, \rangle, \text{senc}, \text{aenc}\}
\end{align*}
\]
Simplifying deducibility constraints

- Simplification $C \rightsquigarrow_{\sigma} C'$
  If we can find a solution $\theta$ for $C'$, then $\sigma \theta$ is a solution for $C$

- $C \land T \vdash M \rightsquigarrow C$
  if $T \cup \{x | (T' \vdash x) \in C, T' \subset T\} \vdash M$

We can delete a constraint $T_{i+1} \vdash M_{i+1}$, if we prove that the attacker could have generated $M_{i+1}$
using $T_{i+1}$ plus any variables that it chose in previous steps ($T_i \vdash x$).
Simplifying deducibility constraints

- **Simplification** \( C \rightsquigarrow \sigma C' \)
  
  If we can find a solution \( \theta \) for \( C' \), then \( \sigma \theta \) is a solution for \( C \)

- \( C \land T \vdash M \rightsquigarrow C \)
  
  if \( T \cup \{ x \mid (T' \vdash x) \in C, T' \subset T \} \vdash M \)

  We can delete a constraint \( T_{i+1} \vdash M_{i+1} \)
  
  if we prove that the attacker could have generated \( M_{i+1} \)
  
  using \( T_{i+1} \) plus any variables that it chose in previous steps \( (T_i \vdash x) \).

- \( C \land T \vdash M \rightsquigarrow \sigma C \sigma \land T \sigma \vdash M \sigma \)
  
  if \( N \in \text{st}(T), \sigma = \text{mgu}(M, N), M \neq N, M, N \) not variables
  
  If \( M \) matches a subterm of \( T \), then we unify \( M \) and \( N \).
  
  If the result \( C' \) has a solution, so does the original system \( C \)

  There is no point in unifying equal terms or variables.
C \land T \vdash M \sim_{\sigma} T \sigma \vdash M\sigma

if N, N' \in \text{st}(T), \sigma = \text{mgu}(M, N), M \neq N

If two subterms N, N' of T match, we unify the two.
If the result C' has a solution, so does the original system C
Simplifying deducibility constraints

- \( C \land T \vdash M \leadsto_{\sigma} C\sigma \land T\sigma \vdash M\sigma \) if \( N, N' \in \text{st}(T) \), \( \sigma = \text{mgu}(M, N) \), \( M \neq N \)
  
  If two subterms \( N, N' \) of \( T \) match, we unify the two. If the result \( C' \) has a solution, so does the original system \( C \)

- \( C \land T \vdash M \leadsto \bot \) if \( \text{fv}(T \cup \{M\}) = \emptyset \) and \( T \not\vdash M \)
  
  The system is unsolvable if there is a constraint \( T \vdash M \) that is unprovable and does not contain any (attacker-controlled) variables.
Simplifying deducibility constraints

- $C \land T \vdash M \rightsquigarrow_\sigma C\sigma \land T\sigma \vdash M\sigma$
  
  if $N, N' \in \text{st}(T)$, $\sigma = \text{mgu}(M, N)$, $M \neq N$

  If two subterms $N, N'$ of $T$ match, we unify the two.

  If the result $C'$ has a solution, so does the original system $C$

- $C \land T \vdash M \rightsquigarrow \bot$
  
  if $\text{fv}(T \cup \{M\}) = \emptyset$ and $T \not\models M$

  The system is unsolvable if there is a constraint $T \vdash M$ that is unprovable and does not contain any (attacker-controlled) variables.

- $C \land T \vdash f(M, N) \rightsquigarrow C \land T \vdash M \land T \vdash N$ if $f \in \{aenc, senc\}$

  Constraints on constructed terms can be simplified to their components. (The attacker can apply public constructors.)
Solving deducibility constraints

- The deducibility constraint system $\bot$ is unsolvable
- The deducibility constraint system $C$ is solvable if each of its constraints are of the form $T \vdash x$ for some variable $x$
- Exercise: Show that a solved deducibility constraint system has a solution $\sigma$
  \[
  T_1 \vdash M_1 \land T_2 \vdash M_2 \land \cdots \land T_n \vdash M_n
  \]

- Recall that $T_1 \neq \emptyset$ and $T_1$ has no variables
Correctness of simplification

- $C \rightsquigarrow^{*}_\sigma C'$ is the transitive closure of $\rightsquigarrow_\sigma$ that composes substitutions

- *Preservation*: Suppose $C \rightsquigarrow_\sigma C'$. If $C$ is a deducibility constraint system, so is $C'$
  - Proof: Show that $\rightsquigarrow_\sigma$ preserves monotonicity and origination
Correctness of simplification

- $C \sim^{\sigma}_* C'$ is the transitive closure of $\sim^{\sigma}$ that composes substitutions.

Preservation: Suppose $C \sim^{\sigma} C'$. If $C$ is a deducibility constraint system, so is $C'$
  - Proof: Show that $\sim^{\sigma}$ preserves monotonicity and origination.

Termination: There is no infinite chain $C \sim^{\sigma_1} C_1 \sim^{\sigma_2} \cdots \sim^{\sigma_n} C_n$
  So, simplification always terminates, either in a solved system or in $\perp$. 

Karthikeyan Bhargavan (INRIA)
Security proofs in the symbolic model
October 2013 15 / 38
Correctness of simplification

- $C \leadsto^*_\sigma C'$ is the transitive closure of $\leadsto_\sigma$ that composes substitutions
- **Preservation**: Suppose $C \leadsto_\sigma C'$. If $C$ is a deducibility constraint system, so is $C'$
  - Proof: Show that $\leadsto_\sigma$ preserves monotonicity and origination
- **Termination**: There is no infinite chain $C \leadsto_{\sigma_1} C_1 \leadsto_{\sigma_2} \cdots \leadsto_{\sigma_n} C_n$ So, simplification always terminates, either in a solved system or in $\bot$.
- **Correctness**: If $C \leadsto^*_\sigma C'$ and $\theta$ solves $C'$ then $\sigma \theta$ solves $C$
  So, a solution of a simplified system yields a solution of the original.
Correctness of simplification

- $C \rightsquigarrow^*_\sigma C'$ is the transitive closure of $\rightsquigarrow_\sigma$ that composes substitutions.
- **Preservation**: Suppose $C \rightsquigarrow_\sigma C'$. If $C$ is a deducibility constraint system, so is $C'$
  - Proof: Show that $\rightsquigarrow_\sigma$ preserves monotonicity and origination.
- **Termination**: There is no infinite chain $C \rightsquigarrow_\sigma_1 C_1 \rightsquigarrow_\sigma_2 \cdots \rightsquigarrow_\sigma_n C_n$
  So, simplification always terminates, either in a solved system or in $\bot$.
- **Correctness**: If $C \rightsquigarrow^*_\sigma C'$ and $\theta$ solves $C'$ then $\sigma \theta$ solves $C$
  So, a solution of a simplified system yields a solution of the original.
- **Completeness**: If $\theta$ solves $C$, then there exists a solved $C'$ with solution $\theta'$, such that $C \rightsquigarrow^*_\sigma C'$ and $\theta = \sigma \theta'$
  So, every solution can be found through simplification, but we may have to explore a lot of simplification paths.
Non-deterministic algorithm:
- Choose any simplification rule, choose a substitution $\sigma$, apply the rule $C \sim_{\sigma} C'$
- Repeat until either $C'$ is $\perp$ or solved

This algorithm requires each step to explore all possible substitutions (exponential branching + exponential growth of constraint system)

More generally, the security problem for deducibility constraint systems is NP-hard (co-NP-complete).

Deciding security properties for cryptographic protocols. Application to key cycles, Hubert Comon-Lundh, Véronique Cortier, Eugen Zalinescu
http://arxiv.org/abs/0708.3564
Processes with unbounded sessions

- Processes with replication:
  \[ P, Q, R ::= \text{Processes} \]
  
  \[ \ldots \]
  
  \[ P \parallel Q \text{ parallel composition} \]
  
  \[ !P \text{ replication} \]

- Semantics through structural congruence (\(\equiv\)):
  \[ !P \equiv P \parallel !P \]

- Secrecy: An extended process \(A\) keeps term \(M\) secret if there is no frame \(\phi\) and extended process \(B\), such that
  \[ A \overset{I}{\rightarrow} \text{new } \bar{a}.\phi \parallel B \text{ and new } \bar{a}.\phi \vdash M \]
Secrecy for our process calculus is undecidable

... even if the attacker is passive, only fresh names are used as keys (not $\langle a, b \rangle$), and all messages are of bounded size

Proof by reduction to Post Correspondence Problem (PCP)
Post Correspondence Problem (PCP)

- **Problem**: Given two finite lists \( v_0, \ldots, v_n \) and \( w_0, \ldots, w_n \), where each \( w_i \in \{a, b\}^* \), is there a sequence of indices \( i_1, \ldots, i_k \) such that \( v_{i_1} \cdots v_{i_k} = w_{i_1} \cdots w_{i_k} \)?

- **Example**: \( v_0 = a, v_1 = ab, v_2 = bba, w_0 = baa, w_1 = aa, w_2 = bb \)
  - Find a solution sequence of indices (with possible repeating indices)
  - Is there a solution if you remove \( v_0 \) and \( w_0 \)?

- PCP is undecidable [Post, 1946]
Undecidability of unbounded sessions security

- We can code the PCP problem as a secrecy goal in the applied pi calculus.
- Assume public free names $a, b, c$, and a secret symmetric key $k$.
- We code each string $v_i$ as a list, where each element is either $a$ or $b$. We use concat to concatenate lists.
Undecidability of unbounded sessions security

- We can code the PCP problem as a secrecy goal in the applied pi calculus.
- Assume public free names $a, b, c$, and a secret symmetric key $k$.
- We code each string $v_i$ as a list, where each element is either $a$ or $b$. We use concat to concatenate lists.
- Process $A$ sends the attacker each individual string encrypted under $k$:
  \[ P_A = \text{out}(c, \text{senc}(\langle v_0, w_0 \rangle, k)). \]
  \[
  \ldots \]
  \[ \text{out}(c, \text{senc}(\langle v_n, w_b \rangle, k)).0 \]
Undecidability of unbounded sessions security

- We can code the PCP problem as a secrecy goal in the applied pi calculus.
- Assume public free names $a$, $b$, $c$, and a secret symmetric key $k$.
- We code each string $v_i$ as a list, where each element is either $a$ or $b$. We use \texttt{concat} to concatenate lists.
- Process A sends the attacker each individual string encrypted under $k$:

  $$P_A = \texttt{out}(c, \texttt{senc}(<v_0, w_0>, k)).$$
  $$\ldots$$
  $$\texttt{out}(c, \texttt{senc}(<v_n, w_b>, k)).0$$

- Process B lets the attacker concatenate the two sets of strings in any sequence of his choice. (Replication crucial to get arbitrary sequences.)

  $$P_B = \texttt{in}(c, \texttt{senc}(<x, y>, k)).$$
  $$\texttt{out}(c, \texttt{senc}(<\texttt{concat}(x, v_0), \texttt{concat}(y, w_0)>, k)).$$
  $$\ldots$$
  $$\texttt{out}(c, \texttt{senc}(<\texttt{concat}(x, v_b), \texttt{concat}(y, w_n)>, k)).0$$
Assume public free names $a$, $b$, $c$, and a secret symmetric key $k$

$P_A = \textbf{out}(c, \text{senc}(\langle v_0, w_0 \rangle, k))$.

\ldots

$\textbf{out}(c, \text{senc}(\langle v_n, w_b \rangle, k)).0$

$P_B = \textbf{in}(c, \text{senc}(\langle x, y \rangle, k))$.

$\textbf{out}(c, \text{senc}(\langle \text{concat}(x, v_0), \text{concat}(y, w_0) \rangle, k))$.

\ldots

$\textbf{out}(c, \text{senc}(\langle \text{concat}(x, v_b), \text{concat}(y, w_n) \rangle, k)).0$
Assume public free names $a, b, c$, and a secret symmetric key $k$.

\[ P_A = \text{out}(c, \text{senc}((v_0, w_0), k)). \]

\[ \ldots \]

\[ \text{out}(c, \text{senc}((v_n, w_b), k)).0 \]

\[ P_B = \text{in}(c, \text{senc}((x, y), k)). \]

\[ \text{out}(c, \text{senc}((\text{concat}(x, v_0), \text{concat}(y, w_0)), k)). \]

\[ \ldots \]

\[ \text{out}(c, \text{senc}((\text{concat}(x, v_b), \text{concat}(y, w_n)), k)).0 \]

Process C tests if a solution to the PCP problem has been found. If yes, it leaks $k$ to the attacker

\[ P_C = \text{in}(c, \text{senc}(x, x)k). \]

\[ \text{out}(c, k).0 \]

So, the attacker gets the secret $k$ iff the PCP problem has a solution.

Corollary: secrecy in the applied pi calculus is undecidable.
Researchers use several strategies to build automated symbolic analysis tools for cryptographic protocols.

They identify decidable (bounded) subfragments of protocol specification languages (e.g. see FDR and AVISPA).

They require user interaction in the form of type annotations or proof guidance (e.g. see Isabelle/HOL and Cryptyc).

They allow non-termination but guarantee correctness of any results (e.g. see ProVerif).

We will study ProVerif later in this course.

We now look at security type systems and we will return to them in the final part of the course.
Type system for spi calculus: Cryptyc

*Authenticity by Typing for Security Protocols*, AD Gordon and ASA Jeffrey, TCS 2003

Formal type system corresponding to ProVerif

*Analyzing security protocols with secrecy types and logic of programs*, M Abadi and B Blanchet, JACM 2005

We will briefly study a security type system for applied pi and see how to prove secrecy properties using this technique

*Main Security Invariant*: A well-typed process will never send a secret value over a public channel
Types for applied pi: Environments and Signatures

- Environments record types for names and variables:

\[ E ::= E, a : T | E, x : T | \emptyset \]

- Well-formed environments do not have name or variable clashes

\[ E \vdash \diamond \]
Types for applied pi: Environments and Signatures

- Environments record types for names and variables:

\[ E ::= E, a : T | E, x : T | \emptyset \]

- Well-formed environments do not have name or variable clashes

\[ E \vdash \diamond \]

- To every constructor and destructor we give a type signature:

\[ g : (T_1, \ldots, T_n) \rightarrow T \]

- In general, each function may have multiple types, but given a set of argument types, there must be at most one result type

\[ \langle -, - \rangle : (T, T') \rightarrow T \times T' \text{ for all } T, T' \]
Types for Terms: $E \vdash M : T$

- **Term syntax:**
  - $M, N, O, \ldots ::=$ Terms
  - $a$ name
  - $x$ variable
  - $f(M_1, \ldots, M_n)$ function application

- The judgement $E \vdash M : T$ means that $M$ has type $T$ in $E$
Types for Terms: $E \vdash M : T$

- **Term syntax:**
  
  $M, N, O, \ldots ::= \text{Terms}$
  
  $a$ name
  
  $x$ variable
  
  $f(M_1, \ldots, M_n)$ function application

- The judgement $E \vdash M : T$ means that $M$ has type $T$ in $E$

- $E \vdash a : T$ if $a : T \in E$
Types for Terms: $E \vdash M : T$

- **Term syntax:**
  - $M, N, O, \ldots ::= \text{Terms}$
  - $a$ : name
  - $x$ : variable
  - $f(M_1, \ldots, M_n)$ : function application

- The judgement $E \vdash M : T$ means that $M$ has type $T$ in $E$
- $E \vdash a : T$ if $a : T \in E$
- $E \vdash x : T$ if $x : T \in E$
Types for Terms: $E \vdash M : T$

- Term syntax:
  - $M, N, O, \ldots ::= \text{Terms}$
  - $a$ name
  - $x$ variable
  - $f(M_1, \ldots, M_n)$ function application

- The judgement $E \vdash M : T$ means that $M$ has type $T$ in $E$
  - $E \vdash a : T$ if $a : T \in E$
  - $E \vdash x : T$ if $x : T \in E$
  - $E \vdash f(M_1, \ldots, M_n) : T$
    - if for each $i$, $E \vdash M_i : T_i$
    - and $f$ has a type $(T_1, \ldots, T_n) \rightarrow T$
Type syntax:

- $T, U, V, \ldots ::= \text{Types}$
- Public
- Secret
- $T_1 \times T_2$
- $\text{PK}(T)$
- $\text{SK}(T)$
- $\text{EK}(T)$
- $\text{MK}(T)$
- $\text{Chan}(T)$

- public data
- secret data
- pair
- public key for encrypting terms of type $T$
- private key for decrypting terms of type $T$
- symmetric key for encrypting terms of type $T$
- symmetric key for MACing terms of type $T$
- channel for sending terms of type $T$
Types for Terms: Type Signatures

- Types for pairing and symmetric encryption:

\[ \langle - , - \rangle : (T, T') \rightarrow T \times T' \text{ for all } T, T' \]

\[ \text{senc} : (T, \text{EK}(T)) \rightarrow \text{Public} \]

\[ \text{sdec} : (\text{Public}, \text{EK}(T)) \rightarrow T \]

- Rules for assigning types to function:
  - For every argument type tuple, there must be a unique result type
  - Public functions must be usable by the attacker (with his own keys)
    So every function must also have a public type:

\[ \text{senc} : (\text{Public}, \text{Public}) \rightarrow \text{Public} \]

\[ \text{sdec} : (\text{Public}, \text{Public}) \rightarrow \text{Public} \]

- Types must be consistent with the equational theory:
  - If \( E \vdash M : T \) and \( M = N \) then \( E \vdash N : T \)
  - Exercise: Check type consistency with \( \text{sdec}(\text{senc}(M, N), N) = M \)
Exercise: Define types for hmac(\(-, -\)) and verify(\(-, -, -\)), assuming the following (slightly modified) equation for verification:

\[
\text{verify}(M, \text{hmac}(M, K), K) = M
\]

Exercise: Define types for pk(\(-\)), aenc(\(-, -\)), and adec(\(-, -\)), assuming the usual equations.
The judgement $E \vdash P$ means that $P$ is well typed in $E$

Output: $E \vdash \text{out}(M, N).P$
if $E \vdash M : \text{Chan}(T)$ and $E \vdash N : T$ and $E \vdash P$
Types for processes

- The judgement $E \vdash P$ means that $P$ is well typed in $E$
- **Output**: $E \vdash \text{out}(M, N).P$
  if $E \vdash M : \text{Chan}(T)$ and $E \vdash N : T$ and $E \vdash P$
- **Input**: $E \vdash \text{in}(M, x).P$
  if $E \vdash M : \text{Chan}(T)$ and $E, x : T \vdash P$
  If the channel has more than one payload type $T$, then $P$ must be well typed against all such $T$
The judgement $E \vdash P$ means that $P$ is well typed in $E$.

**Output:** $E \vdash \text{out}(M, N).P$

if $E \vdash M : \text{Chan}(T)$ and $E \vdash N : T$ and $E \vdash P$

**Input:** $E \vdash \text{in}(M, x).P$

if $E \vdash M : \text{Chan}(T)$ and $E, x : T \vdash P$

If the channel has more than one payload type $T$, then $P$ must be well typed against all such $T$.

**New:** $E \vdash \text{new } a.P$

if $E, a : T \vdash P$ (we are allowed to choose any $T$)
Types for processes

- **Conditional**: $E \vdash \text{if } M = N \text{ then } P \text{ else } Q$
  - if $E \vdash M : T$ and $E \vdash N : T$ and $E \vdash P$ and $E \vdash Q$
  - or if $E \vdash M : T$ and $E \vdash N : T' \neq T$ and $E \vdash Q$

Other rules are straightforward.

- **Parallel**: $E \vdash P \parallel Q$
  - if $E \vdash P$ and $E \vdash Q$

- **Replication**: $E \vdash !P$
  - if $E \vdash P$

- **Zero**: $E \vdash 0$
  - if $E$
Types for processes

- **Conditional:** $E \vdash \text{if } M = N \text{ then } P \text{ else } Q$
  
  if $E \vdash M : T$ and $E \vdash N : T$ and $E \vdash P$ and $E \vdash Q$

  or if $E \vdash M : T$ and $E \vdash N : T' \neq T$ and $E \vdash Q$

- Other rules are straightforward.

- **Parallel:** $E \vdash P \parallel Q$ if $E \vdash P$ and $E \vdash Q$

- **Replication:** $E \vdash !P$ if $E \vdash P$

- **Zero:** $E \vdash 0$

- We extend these rules to extended processes: $E \vdash A$
Properties of the type system

- **Type Soundness**: A well typed process remains well typed in all runs

- **Substitution**: If $E \triangleright M : T$ and $E, x : T \triangleright A$ then $E \triangleright A \{M/x\}$

- **Subject Congruence**: If $E \triangleright A$ and $A \cong B$ then $E \triangleright B$

- **Subject Reduction**: If $E \triangleright A$ and $A \rightarrow B$ then $E \triangleright B$

- **Attacker typability**: Any type process $A$ that uses only names and variables of type Public or Chan (Public) can be typed. We can give type Public to all its variables and names. The constraint on public functions ensures that there is no type error. Hence, typing does not restrict the attacker model.

- **Secrecy**: If $s : Secret \triangleright A$, then new $s. A$ preserves the secrecy of $s$. For all new $a. A \not\rightarrow B$, we have $(B) \not\equiv s$. Follows from subject reduction and attacker typability.
Properties of the type system

- **Type Soundness**: A well typed process remains well typed in all runs
- **Substitution**: If $E \vdash M : T$ and $E, x : T \vdash A$ then $E \vdash A\{M/x\}$
Properties of the type system

- **Type Soundness**: A well-typed process remains well-typed in all runs.
  - **Substitution**: If $E \vdash M : T$ and $E, x : T \vdash A$ then $E \vdash A\{M/x\}$
  - **Subject Congruence**: If $E \vdash A$ and $A \equiv B$ then $E \vdash B$
Properties of the type system

- **Type Soundness**: A well typed process remains well typed in all runs
  - *Substitution*: If \( E \vdash M : T \) and \( E, x : T \vdash A \) then \( E \vdash A\{M/x\} \)
  - *Subject Congruence*: If \( E \vdash A \) and \( A \equiv B \) then \( E \vdash B \)
  - *Subject Reduction*: If \( E \vdash A \) and \( A \xrightarrow{I} B \) then \( E \vdash B \)
Properties of the type system

- **Type Soundness**: A well typed process remains well typed in all runs.
  - *Substitution*: If \( E \vdash M : T \) and \( E, x : T \vdash A \) then \( E \vdash A\{M/x\} \)
  - *Subject Congruence*: If \( E \vdash A \) and \( A \equiv B \) then \( E \vdash B \)
  - *Subject Reduction*: If \( E \vdash A \) and \( A \xrightarrow{l} B \) then \( E \vdash B \)

- **Attacker typability**: Any untyped process \( A \) that uses only names and variables of type Public or Chan(Public) can be typed.
  - We can give type Public to all its variables and names.
  - The constraint on public functions ensures that there is no type error.
  - Hence, typing does not restrict the attacker model.
Properties of the type system

- **Type Soundness**: A well typed process remains well typed in all runs
  - **Substitution**: If \( E \vdash M : T \) and \( E, x : T \vdash A \) then \( E \vdash A\{M/x\} \)
  - **Subject Congruence**: If \( E \vdash A \) and \( A \equiv B \) then \( E \vdash B \)
  - **Subject Reduction**: If \( E \vdash A \) and \( A \xrightarrow{I} B \) then \( E \vdash B \)

- **Attacker typability**: Any untyped process \( A \) that uses only names and variables of type Public or Chan(Public) can be typed
  - We can give type Public to all its variables and names
  - The constraint on public functions ensures that there is no type error
  - Hence, typing does not restrict the attacker model

- **Secrecy**: If \( s : \text{Secret} \vdash A \), then \textbf{new} \( s.A \) preserves the secrecy of \( s \)
  - For all \textbf{new} \( a.A \xrightarrow{\bar{I}} B \), we have \( \phi(B) \not\vdash s \)
  - Follows from subject reduction and attacker typability
Exercise: Secure RPC revisited

- Write processes for the one-message protocol:
  \[ A \rightarrow B : \langle \text{senc(request, } k), \text{hmac(senc(request, } k), } k' \rangle \]

- Define an environment for the free names \((k, k', \text{request}, c)\)
- Show that the processes are well-typed in this environment
- As a corollary, obtain that request is kept secret
Towards Current Cryptographic Protocol Research
Verifying Cryptographic Hardware

Used by banks, Navigo, ... 
Crypto protocols in hardware 
Secret key cannot be extracted from device 
Needs to be resistant to side-channel attacks

Research in France: Graham Steel (INRIA, CryptoSense), Véronique Cortier (CNRS) 
Publications in CRYPTO’12, CCS’12
Verifying Cryptographic Hardware

Used by banks, Navigo, ... 
Crypto protocols in hardware
Secret key cannot be extracted from device
Needs to be resistant to side-channel attacks

Research in France: Graham Steel (INRIA, CryptoSense), Véronique Cortier (CNRS)
*Publications in CRYPTO’12, CCS’12*

---

Efficient Padding Oracle Attacks on Cryptographic Hardware*

Romain Bardou¹, Riccardo Focardi²*, Yusuke Kawamoto³**, Lorenzo Simionato²†, Graham Steel⁴***, and Joe-Kai Tsay⁵***

¹ INRIA SecSI, LSV, CNRS & ENS-Cachan, France  
² DAIS, Università Ca’ Foscari, Venezia, Italy
Verifying Protocol Implementations

In practice, few attacks on crypto protocols
Many attacks on protocol implementations
E.g. OpenSSL, SChannel (TLS) . . .
How to verify that a 5kLOC protocol implementation is correct?

Research in the Paris area: Cédric Fournet (Microsoft), Karthik Bhargavan (INRIA)
Publications in POPL ’10, Oakland’13
In practice, few attacks on crypto protocols. Many attacks on protocol implementations. E.g. OpenSSL, SChannel (TLS) . . .

How to verify that a 5kLOC protocol implementation is correct?

Research in the Paris area: Cédric Fournet (Microsoft), Karthik Bhargavan (INRIA)

*Publications in POPL’10, Oakland’13*
New cryptographic protocols for the Web
Single sign-on, Cloud-based storage, . . .

New models for the browser, cloud, etc.
Formal semantics and type systems for
JavaScript and PHP

Research in the Paris area: Karthik Bhargavan (INRIA)
*Publications in CSF’12, Usenix’13, POPL’14*
Verifying Web Application Security

New cryptographic protocols for the Web
Single sign-on, Cloud-based storage, . . .

New models for the browser, cloud, etc.
Formal semantics and type systems for
JavaScript and PHP

Research in the Paris area: Karthik
Bhargavan (INRIA)
*Publications in CSF’12, Usenix’13, POPL’14*
Internships at INRIA and INRIA-Microsoft Research Joint Center
- Programming language semantics, type systems
- Cryptographic protocols, security proofs
- Symbolic and computational models

Email karthikeyan.bhargavan@inria.fr

Follow http://prosecco.inria.fr/internships
Course Outline

- Done (K Bhargavan): Intro, Proofs in the symbolic model
- 4 weeks (D Pointcheval): Proofs in the computational model
- 4 weeks (B Blanchet): Tools and techniques for proof automation
- 4 weeks (K Bhargavan): Verifying protocol implementations