Formal verification of security protocols
I. Write out the following protocol in the applied-pi calculus.

\[ A \rightarrow B : \text{enc}((A, \text{enc}(m, \text{pk}(B))), \text{pk}(B)) \]

\[ B \rightarrow A : \text{enc}(m, \text{pk}(A)) \]

II. Write the equational theory for XOR with identity 0. XOR is associative and commutative.

III. When can one get multiple reduction sequences from a single configuration? List out all possible scenarios.

BONUS: Is there a bound on the number of such sequences? If yes, what is it?
RECAP

- Saw how to use ProVerif for automated verification
- Wrote a simple protocol and properties in ProVerif
- ProVerif tries to prove a property; if it cannot be proved, tries to produce an attack trace
- Saw that ProVerif can often produce bizarre error traces, and needs some help to produce a “reasonable” error trace
PROVERIF: SYNTAX

- Crypto operations specified using equations or rewrite rules
- Declare types, constructor functions, and reduction rules for destructors before starting a protocol description
- Declare the desired property using the `query` keyword
- Enriched terms: allow one to include `new, if then else` etc in `term` syntax
ENRICHED TERM SYNTAX

\[ M, N ::= \]
\[ a, b, c, k, m, n, s \]
\[ x, y, z \]
\[ (M_1, \ldots, M_j) \]
\[ h(M_1, \ldots, M_j) \]
\[ i \]
\[ M + i \]
\[ i + M \]
\[ M - i \]
\[ M > N \]
\[ M < N \]
\[ M \geq N \]
\[ M \leq N \]
\[ M = N \]
\[ M \neq N \]
\[ M \land M \]
\[ M \lor \]
\[ \text{not}(M) \]
\[ \text{new } a : t; M \]
\[ \text{if } M \text{ then } N \text{ else } N' \]
\[ \text{let } T = M \text{ in } N \text{ else } N' \]
\[ \text{event } e(M_1, \ldots, M_n); M \]

enriched terms
names
variables
tuple
c constructor/destructor application
natural number \(i \in \mathbb{N}\)
addition \(i \in \mathbb{N}\)
addition \(i \in \mathbb{N}\)
subtraction \(i \in \mathbb{N}\)
greater
smaller
greater or equal
smaller or equal
term equality
term disequality
conjunction
disjunction
negation
name restriction
conditional
term evaluation
event
ProVerif has a type `nat` to represent natural numbers.

One can add and subtract a number `i` from a term `t` (`t+i`, `i+t`, `t-i`).

One can also test for order on terms (`>`, `<`, `>=`, `<=`, `=`, `<>`).

Constructors cannot have type `nat`, but destructors can:

```plaintext
type key.
fun enc5(nat, key) : bitstring.
fun dec5(bitstring, key) : nat.
reduc forall x:nat, y:key; dec5(enc5(x+5, y), y) = y.
```
Suppose I add multiplication over objects of some new type \texttt{prd}.

Express that multiplication is commutative? Use \texttt{equation}.

Equations need to be convergent (terminating and confluent rewriting) and linear (variables occur at most once in LHS and at most once in RHS).

\begin{verbatim}
  type prd.

  fun mult(prd, prd) : prd.

  equation forall x:prd, y:prd; mult(x, y) = mult(y, x).
\end{verbatim}
PROVERIF: STORAGE

- Can model persistent storage using **table**
- Can populate (**insert**) and access (**get**) entries, but not delete
- Tables are not accessible to the attacker

```
table d(t1,...,tn).
insert d(M1,...,Mn); P.
get d(u1,...,un) in P else Q.
get d(u1,...,un) suchthat M in P else Q.
```
PROVERIF: PROPERTIES

- Secrecy specified as intruder knowledge: \texttt{query attacker(t)}
- Properties can involve events
- Correspondence queries specified as implications over events: \texttt{query x:t, y:t; event(x, y) \implies event(y, x)}
- What about an event \texttt{Place-order(t)} and an event \texttt{Cust-pays(t)}? 
PROVERIF: PROPERTIES

- Secrecy specified as intruder knowledge: `query attacker(t)`
- Can check for any reachability property this way
- Is term $M$ sent on channel $c$? `query mess(c, M)`
- Is $(t_1, \ldots, t_n)$ present in table $d$? `query table(d(t_1, \ldots, t_n))`
- Does $ev$ occur? `query x_1:t_1, \ldots, x_n:t_n; event (ev(x_1, \ldots, x_n))`
Correspondence queries specified as implications over events: \texttt{query} \\
\texttt{x:t; ev1(x) \implies ev2(x)}

- If ev1 has happened, then ev2 has happened (for the same x)

- What about an event \texttt{Place-order(x)} and an event \texttt{Cust-pays(x)}?

- \texttt{inj-event} allows us to specify a one-to-one correspondence

- \texttt{query x:t; inj-event(order(x)) \implies inj-event(paid(x))}
But what about temporal order?

If an order has been placed, the payment was done before.

ProVerif has type time; can mark events with timestamp.

query x:t, i:time, i’:time;
  inj-event(order(x))@i =>
  (inj-event(paid(x))@i’ && i’ < i)