$$X \rightarrow Y\text{:} \quad \{M, X\}_{K_Y}$$

$$Y \rightarrow X\text{:} \quad \{M\}_{K_X}$$

$X \rightarrow Y$: $\{M, X\}_{K_Y}$

$Y \rightarrow X$: $\{M\}_{K_X}$

The rule set, with $X, Y \in \{A, B, C\}, X \neq Y$:

$$\longrightarrow q_0(M)$$

$$q_0(x) \longrightarrow q_1(i_A(x))$$

$$q_1(x) \longrightarrow q(E_B(x))$$

$$q(E_Y(x)) \longrightarrow q_2^{XY}(x)$$

$$q_2^{XY}(i_X(x)) \longrightarrow q_3^{XY}(x)$$

$$q_3^{XY}(x) \longrightarrow q(E_X(x))$$

$$q(x) \longrightarrow q(i_X(x))$$

$$q(i_X(x)) \longrightarrow q(x)$$

$$q(x) \longrightarrow q(E_X(x))$$

$$q(E_C(x)) \longrightarrow q(x)$$

$X \to Y$: $\{M, X\}_{K_Y}$

$Y \to X$: $\{M\}_{K_X}$

The rule set, with $X, Y \in \{A, B, C\}, X \neq Y$:

$$\longrightarrow q_0(M)$$

$$q_0(x) \longrightarrow q_1(i_A(x))$$

$$q_1(x) \longrightarrow q(E_B(x))$$

$$q(E_Y(x)) \longrightarrow q_2^{XY}(x)$$

$$q_2^{XY}(i_X(x)) \longrightarrow q_3^{XY}(x)$$

$$q_3^{XY}(x) \longrightarrow q(E_X(x))$$

$$q(x) \longrightarrow q(i_X(x))$$

$$q(i_X(x)) \longrightarrow q(x)$$

$$q(x) \longrightarrow q(E_X(x))$$

$$q(E_C(x)) \longrightarrow q(x)$$

Derived rules:

$$q_1(x) \quad \longrightarrow \quad q_2^{XB}(x)$$

$X \to Y$: $\{M, X\}_{K_Y}$

$Y \to X$: $\{M\}_{K_X}$

The rule set, with $X, Y \in \{A, B, C\}, X \neq Y$:

$$\longrightarrow q_0(M)$$

$$q_0(x) \longrightarrow q_1(i_A(x))$$

$$q_1(x) \longrightarrow q(E_B(x))$$

$$q(E_Y(x)) \longrightarrow q_2^{XY}(x)$$

$$q_2^{XY}(i_X(x)) \longrightarrow q_3^{XY}(x)$$

$$q_3^{XY}(x) \longrightarrow q(E_X(x))$$

$$q(x) \longrightarrow q(i_X(x))$$

$$q(i_X(x)) \longrightarrow q(x)$$

$$q(x) \longrightarrow q(E_X(x))$$

$$q(E_C(x)) \longrightarrow q(x)$$

Derived rules:

$$q_1(x) \quad \longrightarrow \quad q_2^{XB}(x)$$

$$q_0(x) \quad \longrightarrow \quad q_2^{XB}(i_A(x))$$

$X \to Y$: $\{M, X\}_{K_Y}$

$Y \to X$: $\{M\}_{K_X}$

The rule set, with $X, Y \in \{A, B, C\}, X \neq Y$:

$$\to q_0(M)$$

$$q_0(x) \to q_1(i_A(x))$$

$$q_1(x) \to q(E_B(x))$$

$$q(E_Y(x)) \to q_2^{XY}(x)$$

$$q_2^{XY}(i_X(x)) \to q_3^{XY}(x)$$

$$q_3^{XY}(x) \to q(E_X(x))$$

$$q(x) \to q(i_X(x))$$

$$q(i_X(x)) \to q(x)$$

$$q(x) \to q(E_X(x))$$

$$q(E_C(x)) \to q(x)$$

Derived rules:

$$q_1(x) \quad \to \quad q_2^{XB}(x)$$

$$q_0(x) \quad \to \quad q_2^{XB}(i_A(x))$$

$$q_0(x) \quad \to \quad q_3^{AB}(x)$$

$X \to Y$: $\{M, X\}_{K_Y}$

$Y \to X$: $\{M\}_{K_X}$

The rule set, with $X, Y \in \{A, B, C\}, X \neq Y$:

$$\longrightarrow q_0(M)$$

$$q_0(x) \longrightarrow q_1(i_A(x))$$

$$q_1(x) \longrightarrow q(E_B(x))$$

$$q(E_Y(x)) \longrightarrow q_2^{XY}(x)$$

$$q_2^{XY}(i_X(x)) \longrightarrow q_3^{XY}(x)$$

$$q_3^{XY}(x) \longrightarrow q(E_X(x))$$

$$q(x) \longrightarrow q(i_X(x))$$

$$q(i_X(x)) \longrightarrow q(x)$$

$$q(x) \longrightarrow q(E_X(x))$$

$$q(E_C(x)) \longrightarrow q(x)$$

Derived rules:

$$q_1(x) \longrightarrow q_2^{XB}(x)$$

$$q_0(x) \longrightarrow q_2^{XB}(i_A(x))$$

$$q_0(x) \longrightarrow q_3^{AB}(x)$$

$$\longrightarrow q_3^{AB}(M)$$

$X \to Y$: $\{M, X\}_{K_Y}$

$Y \to X$: $\{M\}_{K_X}$

The rule set, with $X, Y \in \{A, B, C\}, X \neq Y$:

$$\to q_0(M)$$

$$q_0(x) \to q_1(i_A(x))$$

$$q_1(x) \to q(E_B(x))$$

$$q(E_Y(x)) \to q_2^{XY}(x)$$

$$q_2^{XY}(i_X(x)) \to q_3^{XY}(x)$$

$$q_3^{XY}(x) \to q(E_X(x))$$

$$q(x) \to q(i_X(x))$$

$$q(i_X(x)) \to q(x)$$

$$q(x) \to q(E_X(x))$$

$$q(E_C(x)) \to q(x)$$

Derived rules:

$$q_1(x) \quad \to \quad q_2^{XB}(x)$$

$$q_0(x) \quad \to \quad q_2^{XB}(i_A(x))$$

$$q_0(x) \quad \to \quad q_3^{AB}(x)$$

$$\to \quad q_3^{AB}(M)$$

$$q_3^{XY}(x) \quad \to \quad q_2^{X'X}(x)$$

$X \to Y$: $\{M, X\}_{K_Y}$

$Y \to X$: $\{M\}_{K_X}$

The rule set, with $X, Y \in \{A, B, C\}, X \neq Y$:

$$\to q_0(M)$$

$$q_0(x) \to q_1(i_A(x))$$

$$q_1(x) \to q(E_B(x))$$

$$q(E_Y(x)) \to q_2^{XY}(x)$$

$$q_2^{XY}(i_X(x)) \to q_3^{XY}(x)$$

$$q_3^{XY}(x) \to q(E_X(x))$$

$$q(x) \to q(i_X(x))$$

$$q(i_X(x)) \to q(x)$$

$$q(x) \to q(E_X(x))$$

$$q(E_C(x)) \to q(x)$$

Derived rules:

$$q_1(x) \to q_2^{XB}(x)$$

$$q_0(x) \to q_2^{XB}(i_A(x))$$

$$q_0(x) \to q_3^{AB}(x)$$

$$\to q_3^{AB}(M)$$

$$q_3^{XY}(x) \to q_2^{X'X}(x)$$

$$\to q_2^{X'A}(M)$$

$X \to Y$: $\{M, X\}_{K_Y}$

$Y \to X$: $\{M\}_{K_X}$

The rule set, with $X, Y \in \{A, B, C\}, X \neq Y$:

$$\to q_0(M)$$
$$q_0(x) \to q_1(i_A(x))$$
$$q_1(x) \to q(E_B(x))$$

$$q(E_Y(x)) \to q_2^{XY}(x)$$
$$q_2^{XY}(i_X(x)) \to q_3^{XY}(x)$$
$$q_3^{XY}(x) \to q(E_X(x))$$

$$q(x) \to q(i_X(x))$$
$$q(i_X(x)) \to q(x)$$
$$q(x) \to q(E_X(x))$$
$$q(E_C(x)) \to q(x)$$

Derived rules:

$$q_1(x) \to q_2^{XB}(x)$$
$$q_0(x) \to q_2^{XB}(i_A(x))$$
$$q_0(x) \to q_3^{AB}(x)$$
$$\to q_3^{AB}(M)$$

$$q_3^{XY}(x) \to q_2^{X'X}(x)$$
$$\to q_2^{X'A}(M)$$
$$q_3^{XC}(x) \to q(x)$$

$X \rightarrow Y$: $\{M, X\}_{K_Y}$

$Y \rightarrow X$: $\{M\}_{K_X}$

The rule set, with $X, Y \in \{A, B, C\}, X \neq Y$:

$$\longrightarrow q_0(M)$$
$$q_0(x) \longrightarrow q_1(i_A(x))$$
$$q_1(x) \longrightarrow q(E_B(x))$$

$$q(E_Y(x)) \longrightarrow q_2^{XY}(x)$$
$$q_2^{XY}(i_X(x)) \longrightarrow q_3^{XY}(x)$$
$$q_3^{XY}(x) \longrightarrow q(E_X(x))$$

$$q(x) \longrightarrow q(i_X(x))$$
$$q(i_X(x)) \longrightarrow q(x)$$
$$q(x) \longrightarrow q(E_X(x))$$
$$q(E_C(x)) \longrightarrow q(x)$$

Derived rules:

$$q_1(x) \quad \rightarrow \quad q_2^{XB}(x)$$
$$q_0(x) \quad \rightarrow \quad q_2^{XB}(i_A(x))$$
$$q_0(x) \quad \rightarrow \quad q_3^{AB}(x)$$
$$\rightarrow \quad q_3^{AB}(M)$$

$$q_3^{XY}(x) \quad \rightarrow \quad q_2^{X'X}(x)$$
$$\rightarrow \quad q_2^{X'A}(M)$$
$$q_3^{XC}(x) \quad \rightarrow \quad q(x)$$
$$q(x) \quad \rightarrow \quad q(x)$$

No more rules!

Secure!

The rules we have been dealing with define what we call a pushdown system: a set of rules for popping and pushing symbols on a stack while changing states.

This also provides us with a common method for solving the intruder deduction problem and the security problem for ping pong protocols.

The rules we have been dealing with define what we call a pushdown system: a set of rules for popping and pushing symbols on a stack while changing states.

This also provides us with a common method for solving the intruder deduction problem and the security problem for ping pong protocols.

$\Rightarrow$ the stack needs to deal with trees instead of strings.

This is because we can have messages of the form $\langle m_1, m_2 \rangle$ where both m_1 and m_2 are now arbitrary messages, not just constants.

We need push rules of the form $q_1(x), q_2(y) \rightarrow q(\langle x, y \rangle)$.

And pop rules of the form $q_1(\langle x, y \rangle) \rightarrow q_2(x)$.

Rephrasing the intruder deduction problem in this framework

$$q(x), q(y) \rightarrow q(\langle x, y \rangle) \quad \text{(push)}$$

$$q(\langle x, y \rangle) \rightarrow q(x) \quad \text{(pop)}$$

$$q(\langle x, y \rangle) \rightarrow q(y) \quad \text{(pop)}$$

$$q(x), q(y) \rightarrow q(\{x\}_y) \quad \text{(push)}$$

$$q(\{x\}_y), q(y) \rightarrow q(x) \quad \text{(pop)}$$

$$q(\{x\}_k), q(k^{-1}) \rightarrow q(x) \quad \text{(pop)}$$

$$q(x_1), \ldots, q(x_n) \rightarrow q(h(x_1, \ldots, x_n)) \quad \text{(push)}$$

$$q(h(x_1, \ldots, x_n)) \rightarrow q(h(x_i)) \quad \text{(pop)}$$

For initial intruder knowledge of terms like $\{a\}_k$ we have rules

$q_1(a) \quad q_2(k) \quad q_1(x), q_2(y) \rightarrow q(\{x\}_y)$.

Upto details we have the following kinds of rules

$$q_1(x_1), q_2(x_2) \longrightarrow q(f(x_1, x_2))$$

$$q(f(x_1, x_2)), q_2(x_2) \longrightarrow p(x_1)$$

$$q_1(x) \longrightarrow q_2(x)$$

$$\ldots \longrightarrow \ldots$$

Upto details we have the following kinds of rules

$$q_1(x_1), q_2(x_2) \rightarrow q(f(x_1, x_2))$$
$$q(f(x_1, x_2)), q_2(x_2) \rightarrow p(x_1)$$
$$q_1(x) \rightarrow q_2(x)$$
$$\ldots \rightarrow \ldots$$

These are also crucial for handling more complex classes of protocols.

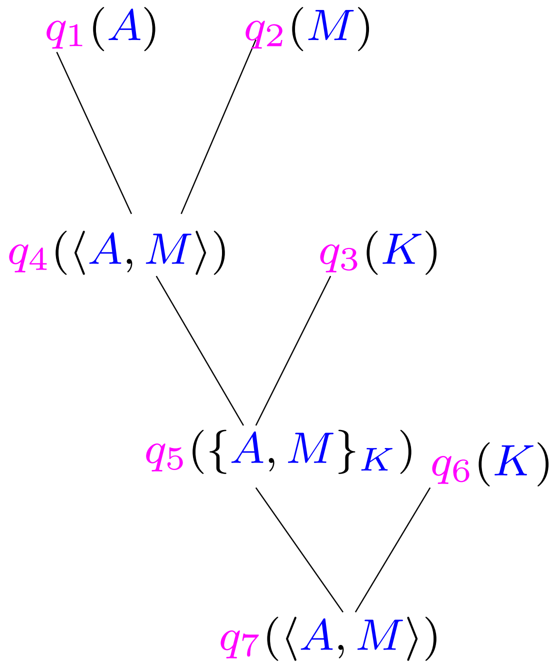
These can be automatically handled by several existing tools today.

E.g. the HX tool (Thomas Gawlitza):

`http://www2.in.tum.de/~gawlitza/hx`

Example: $q_1(A)$ $q_2(M)$ $q_3(K)$ $q_1(x), q_2(y) \to q_4(\langle x, y \rangle)$

$q_4(x), q_3(y) \to q_5(\{x\}_y)$ $q_6(K)$ $q_5(\{x\}_y), q_6(y) \to q_7(x)$



The rule applications form a branching pattern.

An example beyond ping-pong protocols.

$$X \to Y : \{c, M\}_{K_Y}$$
$$Y \to X : \{\langle d, M\rangle, e\}_{K_X}$$

The second step can be described using rules:

$$\to q_X(K_X) \qquad q(\{x\}_y), q_Y(y) \to q_4(x)$$
$$\to q_Y(K_Y) \qquad q_4(\langle x, y\rangle), q_1(x) \to q_5(y)$$
$$\to q_1(c) \qquad q_2(x), q_5(y) \to q_6(\langle x, y\rangle)$$
$$\to q_2(d) \qquad q_6(x), q_3(y) \to q_7(\langle x, y\rangle)$$
$$\to q_3(e) \qquad q_7(x), q_X(y) \to q(\{x\}_y)$$

Finally such rules are also obtained by approximations of complex protocols.

For a state q let $L(q)$ be the set of all messages m such that $q(m)$ is true according to the given set of rules. The previous rule

$$q(\{x\}_k), q(k^{-1}) \rightarrow q(x)$$

can be written as

$$\rightarrow q'(k)$$

$$k^{-1} \in L(q), q(\{x\}_y), q'(y) \rightarrow q(x)$$

Hence we need rules with side-conditions. The above is a membership condition. Other conditions are:

nonemptiness condition: $L(q) \neq \emptyset$

intersection-nonemptiness condition: $L(q_1) \cap L(q_2) \neq \emptyset$

Given only push rules, these conditions are easy to check!

Checking membership $m \in L(q)$ with push rules

We iteratively compute a set S of facts of the above form.

If $q'(c)$ is a rule and c occurs in m then the fact $c \in L(q')$ is added to S.

Checking membership $m \in L(q)$ with push rules

We iteratively compute a set S of facts of the above form.

If $q'(c)$ is a rule and c occurs in m then the fact $c \in L(q')$ is added to S.

If $q_1(x), q_2(y) \rightarrow q_3(\langle x, y \rangle)$ is a rule,
if facts $m_1 \in L(q_1)$ and $m_2 \in L(q_2)$ are in S, and
if $\langle m_1, m_2 \rangle$ occurs in m, then the fact $\langle m_1, m_2 \rangle \in L(q_3)$ is added to S.

Checking membership $m \in L(q)$ with push rules

We iteratively compute a set S of facts of the above form.

If $q'(c)$ is a rule and c occurs in m then the fact $c \in L(q')$ is added to S.

If $q_1(x), q_2(y) \rightarrow q_3(\langle x, y \rangle)$ is a rule,
if facts $m_1 \in L(q_1)$ and $m_2 \in L(q_2)$ are in S, and
if $\langle m_1, m_2 \rangle$ occurs in m, then the fact $\langle m_1, m_2 \rangle \in L(q_3)$ is added to S.

If $q_1(x), q_2(y) \rightarrow q_3(\{x\}_y)$ is a rule,
if facts $m_1 \in L(q_1)$ and $m_2 \in L(q_2)$ are in S, and
if $\{m_1\}_{m_2}$ occurs in m, then the fact $\{m_1\}_{m_2} \in L(q_3)$ is added to S.

Checking membership $m \in L(q)$ with push rules

We iteratively compute a set S of facts of the above form.

If $q'(c)$ is a rule and c occurs in m then the fact $c \in L(q')$ is added to S.

If $q_1(x), q_2(y) \rightarrow q_3(\langle x, y \rangle)$ is a rule,
if facts $m_1 \in L(q_1)$ and $m_2 \in L(q_2)$ are in S, and
if $\langle m_1, m_2 \rangle$ occurs in m, then the fact $\langle m_1, m_2 \rangle \in L(q_3)$ is added to S.

If $q_1(x), q_2(y) \rightarrow q_3(\{x\}_y)$ is a rule,
if facts $m_1 \in L(q_1)$ and $m_2 \in L(q_2)$ are in S, and
if $\{m_1\}_{m_2}$ occurs in m, then the fact $\{m_1\}_{m_2} \in L(q_3)$ is added to S.

Similarly for hash functions.

If the fact $m \in L(q)$ is added in this way then the membership test succeeds, else it fails. This requires polynomial time

Checking nonemptiness $L(q) \neq \emptyset$ with push rules

We iteratively compute a set S of facts of the above form.

If $q'(c)$ is a rule then the fact $L(q') \neq \emptyset$ is added to S.

Checking nonemptiness $L(q) \neq \emptyset$ with push rules

We iteratively compute a set S of facts of the above form.

If $q'(c)$ is a rule then the fact $L(q') \neq \emptyset$ is added to S.

If $q_1(x), q_2(y) \rightarrow q_3(\langle x, y \rangle)$ is a rule,
if facts $L(q_1) \neq \emptyset$ and $L(q_2) \neq \emptyset$ are in S,
then the fact $L(q_3) \neq \emptyset$ is added to S.

Checking nonemptiness $L(q) \neq \emptyset$ with push rules

We iteratively compute a set S of facts of the above form.

If $q'(c)$ is a rule then the fact $L(q') \neq \emptyset$ is added to S.

If $q_1(x), q_2(y) \rightarrow q_3(\langle x, y\rangle)$ is a rule,
if facts $L(q_1) \neq \emptyset$ and $L(q_2) \neq \emptyset$ are in S,
then the fact $L(q_3) \neq \emptyset$ is added to S.

Similarly for encryption and hash functions.

If the fact $L(q) \neq \emptyset$ is added in this way then the non-emptiness test succeeds, else it fails. This requires polynomial time

Checking intersection-nonemptiness $L(q_1) \cap L(q_2) \neq \emptyset$ with push rules

We iteratively compute a set S of facts of the above form.

If $q'(c)$ and $q''(c)$ are rules then the fact $L(q') \cap L(q'') \neq \emptyset$ is added to S.

Checking intersection-nonemptiness $L(q_1) \cap L(q_2) \neq \emptyset$ with push rules

We iteratively compute a set S of facts of the above form.

If $q'(c)$ and $q''(c)$ are rules then the fact $L(q') \cap L(q'') \neq \emptyset$ is added to S.

If $p(x), q(y) \to r(\langle x, y \rangle)$ and $p'(x), q'(y) \to r'(\langle x, y \rangle)$ are rules,
if facts $L(p) \cap L(p') \neq \emptyset$ and $L(q) \cap L(q') \neq \emptyset$ are in S,
then the fact $L(r) \cap L(r') \neq \emptyset$ is added to S.

Checking intersection-nonemptiness $L(q_1) \cap L(q_2) \neq \emptyset$ with push rules

We iteratively compute a set S of facts of the above form.

If $q'(c)$ and $q''(c)$ are rules then the fact $L(q') \cap L(q'') \neq \emptyset$ is added to S.

If $p(x), q(y) \to r(\langle x, y \rangle)$ and $p'(x), q'(y) \to r'(\langle x, y \rangle)$ are rules,
if facts $L(p) \cap L(p') \neq \emptyset$ and $L(q) \cap L(q') \neq \emptyset$ are in S,
then the fact $L(r) \cap L(r') \neq \emptyset$ is added to S.

Similarly for encryption and hash functions.

If the fact $L(q_1) \cap L(q_2) \neq \emptyset$ is added in this way then the intersection-nonemptiness test succeeds, else it fails.
This requires polynomial time

We now generalize the operations which add new rules.

Given rules $q(c)$ and $q(x) \longrightarrow p(x)$ we add the rule $q(c)$.

Given rules $q_1(x), q_2(y) \longrightarrow q(\langle x, y \rangle)$ and $q(x) \longrightarrow p(x)$ we add the rule $q_1(x), q_2(y) \longrightarrow p(\langle x, y \rangle)$.

Similarly for encryption and hash functions.

If $q_1(x), q_2(y) \longrightarrow q(\langle x, y\rangle)$ and $q(\langle x, y\rangle) \longrightarrow p(x)$ are rules then we add the rule $L(q_2) \neq \emptyset, q_1(x) \longrightarrow p(x)$

If $q_1(x), q_2(y) \to q(\langle x, y \rangle)$ and $q(\langle x, y \rangle) \to p(x)$ are rules then
we add the rule $L(q_2) \neq \emptyset, q_1(x) \to p(x)$

If $q_1(x), q_2(y) \to q(\{x\}_y)$ and $q(\{x\}_y), q_3(y) \to p(x)$ are rules then
we add the rule $L(q_2) \cap L(q_3) \neq \emptyset, q_1(x) \to p(x)$

...

If $q_1(x), q_2(y) \rightarrow q(\langle x, y \rangle)$ and $q(\langle x, y \rangle) \rightarrow p(x)$ are rules then we add the rule $L(q_2) \neq \emptyset, q_1(x) \rightarrow p(x)$

If $q_1(x), q_2(y) \rightarrow q(\{x\}_y)$ and $q(\{x\}_y), q_3(y) \rightarrow p(x)$ are rules then we add the rule $L(q_2) \cap L(q_3) \neq \emptyset, q_1(x) \rightarrow p(x)$

...

If C, r is a rule and the condition C succeeds using only the current push rules, then we add the rule r.

If $q_1(x), q_2(y) \to q(\langle x, y\rangle)$ and $q(\langle x, y\rangle) \to p(x)$ are rules then we add the rule $L(q_2) \neq \emptyset, q_1(x) \to p(x)$

If $q_1(x), q_2(y) \to q(\{x\}_y)$ and $q(\{x\}_y), q_3(y) \to p(x)$ are rules then we add the rule $L(q_2) \cap L(q_3) \neq \emptyset, q_1(x) \to p(x)$

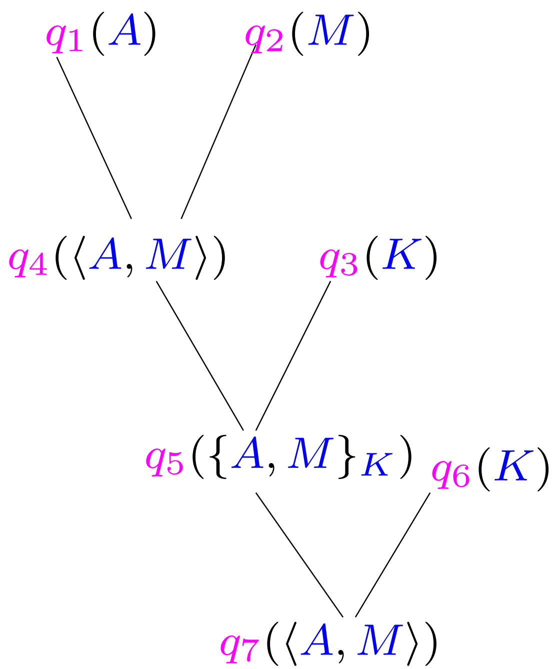
. . .

If C, r is a rule and the condition C succeeds using only the current push rules, then we add the rule r.

We do this repeatedly and finally remove all rules other than push rules.

Then questions like secrecy can be considered as membership tests.
Overall time complexity is polynomial.

Our old example: $q_1(A)$ $q_2(M)$ $q_3(K)$ $q_1(x), q_2(y) \to q_4(\langle x, y\rangle)$
$q_4(x), q_3(y) \to q_5(\{x\}_y)$ $q_6(K)$ $q_5(\{x\}_y), q_6(y) \to q_7(x)$



We add rules: $L(q_3) \cap L(q_6) \neq \emptyset, q_4(x) \to q_7(x)$ $q_4(x) \to q_7(x)$
$q_1(x), q_2(y) \to q_7(\langle x, y\rangle)$

To summarize, we have obtained a general polynomial time algorithm for

- the intruder deduction problem
- the secrecy problem for ping-pong protocols
- the (approximate) secrecy problem for more complex classes of protocols.

Justification for using only three agents

Each session in our ping-pong protocols involves a pair of agents (X, Y).

We may have (interleaved) sessions between several (non-disjoint) pairs $(X_1, Y_1), (X_2, Y_2), (X_3, Y_3) \ldots$.

If such an execution leads to leak of secret, then we show that their is a leak of secret with just two honest agents and one dishonest agent.

Idea: projection!

Such results are crucial for developing automated techniques for analyzing protocols.

In general we may assume, for every pair (X, Y) of distinct agents, an infinite number of secret values $M^i_{X,Y}$, to be used in different sessions between X and Y.

A ping-pong protocol can be represented by a set of rules of the form

$$r_0[X, Y] \equiv q(\alpha_0[X, Y](M^i_{X,Y})$$

$$r_1[X, Y] \equiv q(\beta_1[X, Y](x) \rightarrow q(\alpha_1[X, Y](x))$$

$$\cdots$$

$$r_k[X, Y] \equiv q(\beta_k[X, Y](x)) \rightarrow q(\alpha_k[X, Y](x))$$

For the protocol

$$X \rightarrow Y: \quad \{M\}_{K_Y}, X$$

$$Y \rightarrow X: \quad \{M\}_{K_X}$$

we have $\alpha_0[X, Y] = i_X E_Y \quad \beta_1[X, Y] = i_X E_Y \quad \alpha_1[X, Y] = E_X$

The attacker learns new messages using execution sequences of the form

$w_0(M^i_{X,Y}), w_1(M^i_{X,Y}), w_2(M^i_{X,Y}), \ldots$

Different sequences starting with different initial secrets continue independently of each other.

By symmetry, if $M^i_{X,Y}$ is leaked for some i then it is leaked for all i. Hence for every pair of agents (X, Y) we consider just one secret $M_{X,Y}$.

Also by symmetry, if $M_{X,Y}$ is leaked for some pair (X, Y) of honest agents then it is leaked for all pairs of honest agents.

Hence we consider just one secret $M = M_{A,B}$ between honest agents A and B.

Now we just have the rules $r_0(A, B)$ and $r_1[X, Y]$, . . . ,$r_k[X, Y]$ for distinct X and Y.

Also as usual, we have rules for dishonest agents' abilities to do apply symbols i_X and E_X and remove symbols i_X and E_Z for all agents X and all dishonest agents Z.

The replacement of an agent U by an agent U' in an execution sequence is defined as follows (we never replace a dishonest agent by an honest agent).

Every rule $r_i[U, V]$ is replaced by rule $r_i[U', V]$.

Every rule $r_i[V, U]$ is replaced by rule $r_i[V, U']$.

Addition and removal of symbols i_U, E_U are replaced by addition and removal of symbols $i_{U'}, E_{U'}$.

Let w be the original message obtained from the execution sequence. After the replacement we obtain $w[U \mapsto U']$.

(The $\mapsto$ symbol denoted replacement).

We also need to check there are no sessions between identical agents.

We choose one dishonest agent C. All agents other than A and B are replaced by agent C.

This may create sessions between C and C. But these can be thought of as actions of dishonest agents like encryption, decryption, etc.

Hence if M was obtained in the original execution then it is also obtained by using only A, B and C.

Towards a precise description of general protocols

Goal: develop techniques for analyzing more complex protocols.

E.g. we would like to prove a bound on the number of agents required for analyzing general protocols.

Towards a precise description of general protocols

Goal: develop techniques for analyzing more complex protocols.

E.g. we would like to prove a bound on the number of agents required for analyzing general protocols.

$$
\begin{array}{ll}
1. & A \longrightarrow B : n_a \\
2. & B \longrightarrow A : n_a, n_b \\
3. & A \longrightarrow B : n_b
\end{array}
$$

We need to model

- states of agents: above, A and B each can be in three states.
- intruder's knowledge.

Each protocol may be played between various agents simultaneously with different values of nonces.

A protocol state is a collection of just agent states, together with the intruder's knowledge.

$\Rightarrow$ use multisets, or sets with multiplicities, or unordered lists.

We use rules on multisets to describe execution of protocol steps.

$\Rightarrow$ multiset rewriting (Durgin et. al)

To describe creation of fresh nonces, we use rules like

$$A \rightsquigarrow \exists x \cdot B(x)$$

Given a multiset M, A this rule allows us to obtain the multiset $M, B(c)$ where c is a completely fresh constant.

Encryption free Needham-Schroeder protocol

$$A \longrightarrow B : n_a$$
$$B \longrightarrow A : n_a, n_b$$
$$A \longrightarrow B : n_b$$

Predicates used:

$A_0()$	Alice is in state 0 (initial role state)
$A_1(nonce)$	Alice is in state 1, with a nonce
$A_2(nonce, nonce)$	Alice is in state 2, with two nonces

$B_0()$	Bob is in state 0 (initial role state)
$B_1(nonce, nonce)$	Bob is in state 1, with a nonce
$B_2(nonce, nonce)$	Bob is in state 2, with two nonces
$I(message)$	Intruder knows $message$

Rules:

$$
\begin{aligned}
A_0() &\rightsquigarrow \exists x. A_1(x), I(x) \\
B_0(), I(x) &\rightsquigarrow \exists y. B_1(x,y), I(\langle x,y\rangle), I(x) \\
A_1(x), I(\langle x,y\rangle) &\rightsquigarrow A_2(x,y), I(y), I(\langle x,y\rangle) \\
B_1(x,y), I(y) &\rightsquigarrow B_2(x,y), I(y) \\
&\dots
\end{aligned}
$$

Rules:

$$A_0() \rightsquigarrow \exists x. A_1(x), I(x)$$

$$B_0(), I(x) \rightsquigarrow \exists y. B_1(x,y), I(\langle x,y\rangle), I(x)$$

$$A_1(x), I(\langle x,y\rangle) \rightsquigarrow A_2(x,y), I(y), I(\langle x,y\rangle)$$

$$B_1(x,y), I(y) \rightsquigarrow B_2(x,y), I(y)$$

$$\ldots$$

Example execution:

$B_0(), A_0()$

Rules:

$$\begin{aligned}
A_0() &\rightsquigarrow \exists x. A_1(x), I(x)\\
B_0(), I(x) &\rightsquigarrow \exists y. B_1(x,y), I(\langle x,y\rangle), I(x)\\
A_1(x), I(\langle x,y\rangle) &\rightsquigarrow A_2(x,y), I(y), I(\langle x,y\rangle)\\
B_1(x,y), I(y) &\rightsquigarrow B_2(x,y), I(y)\\
&\ldots
\end{aligned}$$

Example execution:

$B_0(), A_0()$

$\rightsquigarrow A_1(n_a), I(n_a), B_0()$

Rules:

$$\begin{aligned} A_0() &\rightsquigarrow \exists x. A_1(x), I(x) \\ B_0(), I(x) &\rightsquigarrow \exists y. B_1(x, y), I(\langle x, y \rangle), I(x) \\ A_1(x), I(\langle x, y \rangle) &\rightsquigarrow A_2(x, y), I(y), I(\langle x, y \rangle) \\ B_1(x, y), I(y) &\rightsquigarrow B_2(x, y), I(y) \\ &\dots \end{aligned}$$

Example execution:

$B_0(), A_0()$

$\rightsquigarrow A_1(n_a), I(n_a), B_0()$

$\rightsquigarrow B_1(n_a, n_b), I(\langle n_a, n_b \rangle), I(n_a), A_1(n_a)$

Rules:

$$
\begin{aligned}
A_0() &\rightsquigarrow \exists x. A_1(x), I(x) \\
B_0(), I(x) &\rightsquigarrow \exists y. B_1(x,y), I(\langle x, y\rangle), I(x) \\
A_1(x), I(\langle x, y\rangle) &\rightsquigarrow A_2(x,y), I(y), I(\langle x, y\rangle) \\
B_1(x,y), I(y) &\rightsquigarrow B_2(x,y), I(y) \\
&\ldots
\end{aligned}
$$

Example execution:

$$
\begin{aligned}
&B_0(), A_0() \\
&\rightsquigarrow A_1(n_a), I(n_a), B_0() \\
&\rightsquigarrow B_1(n_a, n_b), I(\langle n_a, n_b\rangle), I(n_a), A_1(n_a) \\
&\rightsquigarrow A_2(n_a, n_b), I(n_b), I(\langle n_a, n_b\rangle), I(n_a), B_1(n_a, n_b)
\end{aligned}
$$

Rules:

$$\begin{aligned}
A_0() &\rightsquigarrow \exists x. A_1(x), I(x) \\
B_0(), I(x) &\rightsquigarrow \exists y. B_1(x, y), I(\langle x, y \rangle), I(x) \\
A_1(x), I(\langle x, y \rangle) &\rightsquigarrow A_2(x, y), I(y), I(\langle x, y \rangle) \\
B_1(x, y), I(y) &\rightsquigarrow B_2(x, y), I(y) \\
&\ldots
\end{aligned}$$

Example execution:

$B_0(), A_0()$

$\rightsquigarrow A_1(n_a), I(n_a), B_0()$

$\rightsquigarrow B_1(n_a, n_b), I(\langle n_a, n_b \rangle), I(n_a), A_1(n_a)$

$\rightsquigarrow A_2(n_a, n_b), I(n_b), I(\langle n_a, n_b \rangle), I(n_a), B_1(n_a, n_b)$

$\rightsquigarrow B_2(n_a, n_b), I(n_b), I(\langle n_a, n_b \rangle), I(n_a), A_2(n_a, n_b)$