

# **Automata-based analysis of recursive cryptographic protocols**

**Thomas Wilke**

**Joint work with Ralf Küsters**

**Christian-Albrechts-Universität zu Kiel**

**June 13, 2004**

# Un-/Decidability of security in the DY model

## Undecidable

- protocols with an **unbounded** number of **rewriting** receive-send actions (Amadio et al., Mitchell et al., ...)

## Decidable

- protocols with a **bounded** number of **rewriting** receive-send actions (Turuani, Rusinowitch)
- extension by **XOR** and **Diffie–Hellman exponentiation** (Chevalier, Küsters et al., Shmatikov et al.)

# Un-/Decidability of security in the DY model

## Undecidable

- protocols with an **unbounded** number of **rewriting** receive-send actions (Amadio et al., Mitchell et al., ...)

## Decidable

- protocols with a **bounded** number of **rewriting** receive-send actions (Turuani, Rusinowitch)
- extension by **XOR** and **Diffie–Hellman exponentiation** (Chevalier, Küsters et al., Shmatikov et al.)
- protocols with a **bounded** number of **recursive** receive-send actions

# Recursive Authentication Protocol

$$P_0 \xleftrightarrow{K_{0,1}} P_1 \xleftrightarrow{K_{1,2}} P_2 \xleftrightarrow{K_{2,3}} \dots \xleftrightarrow{K_{n-2,n-1}} P_{n-1} \xleftrightarrow{K_{n-1,n}} S = P_n$$

## Phase 1

$$\text{hash}(k, m) = \langle m, \text{keyed-hash}_k(m) \rangle$$

$$P_0 \rightarrow P_1: \text{hash}(K_{P_0}, \langle P_0, P_1, N_0, \text{init} \rangle)$$

$$P_1 \rightarrow P_2: \text{hash}(K_{P_1}, \langle P_1, P_2, N_1, \text{hash}(K_{P_0}, \langle P_0, P_1, N_0, \text{init} \rangle) \rangle)$$

$$P_2 \rightarrow S: \text{hash}(K_{P_2}, \langle P_2, S, N_2, \text{hash}(K_{P_1}, \langle P_1, P_2, N_1, \text{hash}(K_{P_0}, \langle P_0, P_1, N_0, \text{init} \rangle) \rangle) \rangle)$$

## Phase 2

$$S \rightarrow P_2: \text{enc}(K_{P_2}, \langle K_{2,3}, S, N_2 \rangle) \text{enc}(K_{P_2}, \langle K_{1,2}, P_1, N_2 \rangle)$$

$$\text{enc}(K_{P_1}, \langle K_{1,2}, P_2, N_1 \rangle) \text{enc}(K_{P_1}, \langle K_{0,1}, P_0, N_1 \rangle)$$

$$\text{enc}(K_{P_0}, \langle K_{0,1}, P_1, N_0 \rangle)$$

$$P_2 \rightarrow P_1: \text{enc}(K_{P_1}, \langle K_{1,2}, P_2, N_1 \rangle) \text{enc}(K_{P_1}, \langle K_{0,1}, P_0, N_1 \rangle) \text{enc}(K_{P_0}, \langle K_{0,1}, P_1, N_0 \rangle)$$

$$P_1 \rightarrow P_0: \text{enc}(K_{P_0}, \langle K_{0,1}, P_1, N_0 \rangle)$$

# Recursive Authentication Protocol

## Phase 1

$P_0 \rightarrow P_1: \text{hash}(K_{P_0}, \langle P_0, P_1, N_0, \text{init} \rangle) =: M_0$

$P_1 \rightarrow P_2: \text{hash}(K_{P_1}, \langle P_1, P_2, N_1, M_0 \rangle) =: M_1$

$P_2 \rightarrow P_3: \text{hash}(K_{P_2}, \langle P_2, P_3, N_2, M_1 \rangle) =: M_2$

...

$P_{n-1} \rightarrow S: \text{hash}(K_{P_{n-1}}, \langle P_{n-1}, S, N_{n-1}, M_{n-2} \rangle)$

## Phase 2

$S \rightarrow P_{n-1}: \langle R_{n-1}, L_{n-1}, R_{n-2}, L_{n-2}, \dots, R_0 \rangle$

where  $R_i = \text{enc}(K_{P_i}, \langle K_{i,i+1}, P_{i+1}, N_i \rangle)$

$L_i = \text{enc}(K_{P_i}, \langle K_{i-1,i}, P_{i-1}, N_i \rangle)$

$P_{n-1} \rightarrow P_{n-2}: \langle R_{n-2}, L_{n-2}, \dots, R_0 \rangle$

...

$P_1 \rightarrow P_0: R_0$

# Recursive definition of server

$P_2 \rightarrow S: \text{hash}(K_{P_2}, \langle P_2, S, N_2, \text{hash}(K_{P_1}, \langle P_1, P_2, N_1, \text{hash}(K_{P_0}, \langle P_0, P_1, N_0, \text{init} \rangle) \rangle) \rangle) \rangle)$

$S \rightarrow P_2: \text{enc}(K_{P_2}, \langle K_{2,3}, S, N_2 \rangle) \text{enc}(K_{P_2}, \langle K_{1,2}, P_1, N_2 \rangle)$

$\text{enc}(K_{P_1}, \langle K_{1,2}, P_2, N_1 \rangle) \text{enc}(K_{P_1}, \langle K_{0,1}, P_0, N_1 \rangle)$

$\text{enc}(K_{P_0}, \langle K_{0,1}, P_1, N_0 \rangle)$

$OUT(\text{hash}(K(X), \langle X, S, N, M \rangle)) =$

$HELP(\text{hash}(K(X), \langle X, S, N, M \rangle), \text{new}())$

$HELP(\text{hash}(K(X), \langle X, X', N, \text{init} \rangle), K) =$

$\text{enc}(K(X), \langle K, X', N \rangle)$

$HELP(\text{hash}(K(X), \langle X, X', N, \text{hash}(K(X''), \langle X'', X, N', M \rangle) \rangle), K) =$

$\langle \text{enc}(K(X), \langle K, X', N \rangle), \text{enc}(K(X), \langle K', X'', N \rangle),$

$HELP(\text{hash}(K(X''), \langle X'', X, N', M \rangle), K') \rangle$

where  $K' = \text{new}()$

# Related examples

## **IKE (Internet Key Exchange protocol)**

The responder chooses an item from an unbounded list of security associations.

## **Web service protocols**

Messages can have multiple (an unbounded number of) security tokens.

## **Main issue**

Open-endedness of message structure, which requires iterative or recursive actions.

# The message model

Terms built using:

- atoms from a finite set  $A$ , including principal names, symmetric and asymmetric keys,
- $enc_k(\cdot)$  for  $k \in A$  a key,
- $hash_k(\cdot)$  for  $k \in A$  a key,
- $\langle \cdot, \cdot \rangle$ ,
- **anonymous constants** from an infinite set  $C$ .



# The message model

Terms built using:

- atoms from a finite set  $A$ , including principal names, symmetric and asymmetric keys,
- $enc_k(\cdot)$  for  $k \in A$  a key,
- $hash_k(\cdot)$  for  $k \in A$  a key,
- $\langle \cdot, \cdot \rangle$ ,
- **anonymous constants** from an infinite set  $C$ .

**Notice** Atomic keys only!

# The action model: tree transducer

receive/send action =

sequence of recursive definitions of the form

$$F(t) = t'(F_0(t_0), F_1(t_1), \dots, F_{r-1}(t_{r-1}))$$

where

- all variables from the RHS occur on the LHS,
- $t$  is a **linear** term,
- all terms are **without** anonymous constants,
- each  $t_i$  is a **subterm** of  $t$ ,
- one function symbol marked which is initially called.

# Use of anonymous constants

Functions may have additional parameters for anonymous constants:

$$F(t; c_0, \dots, c_{r-1}) ,$$

where the formal parameter may only be

- a **variable** for an anonymous constant or
- the term *new()*.

## Examples

$$OUT(hash_{k_i}(\langle i, n, N, M \rangle)) = HELP(hash_{k_i}(\langle i, n, N, M \rangle); new())$$

$$HELP(hash_{k_i}(\langle i, j, N, init \rangle); K) = enc_i(\langle K, j, N \rangle)$$

# The server definition revisited

$0, \dots, n$  for principals,  $k_0, \dots, k_{n-1}$  for symmetric keys.

For  $i, j, k < n$ ,

$OUT(hash_{k_i}(\langle i, n, N, M \rangle)) =$

$HELP(hash_{k_i}(\langle i, n, N, M \rangle); new())$

$HELP(hash_{k_i}(\langle i, j, N, init \rangle); K) =$

$enc_i(\langle K, j, N \rangle)$

$HELP(hash_{k_i}(\langle i, j, N, hash_{k_k}(\langle k, i, N', M \rangle) \rangle); K) =$

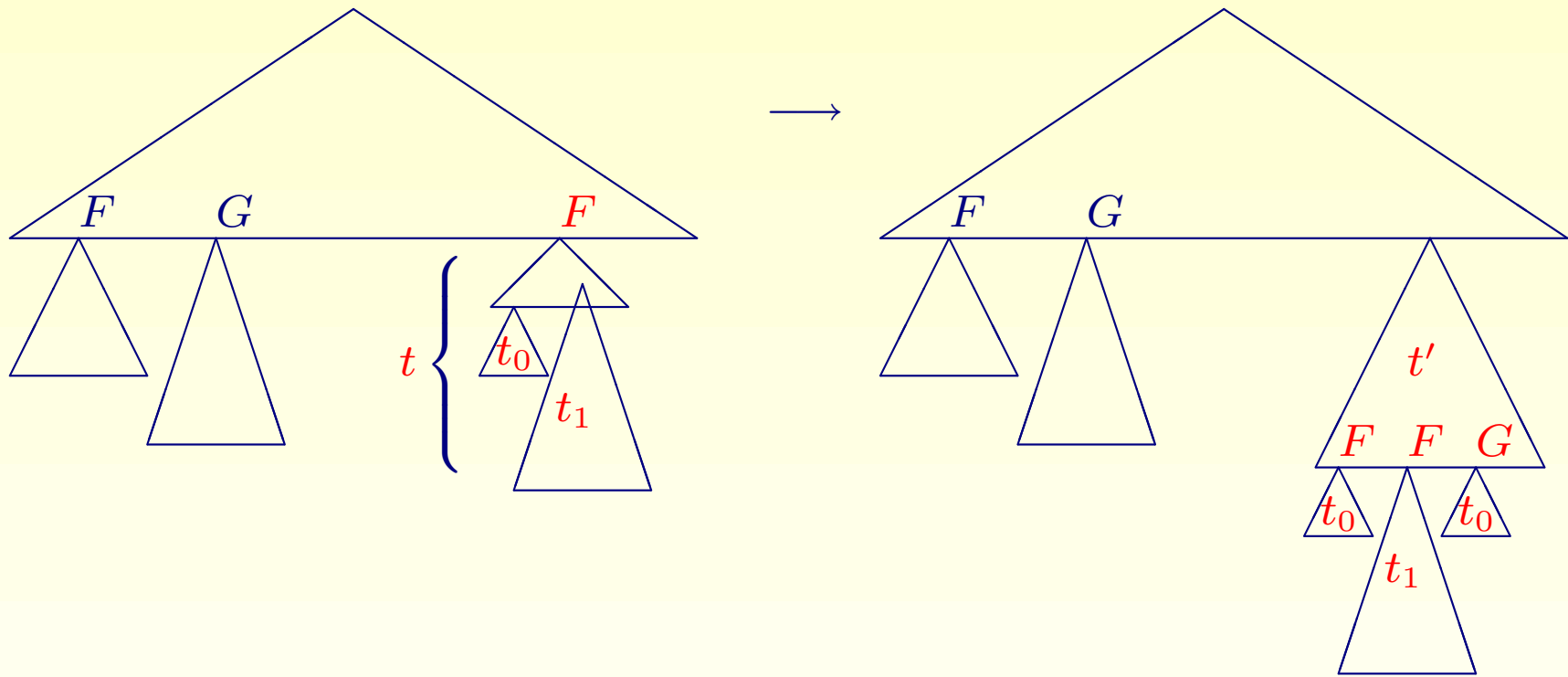
$\langle enc_{k_i}(\langle K, j, N \rangle), HELP2(\langle N, hash_{k_k}(\langle k, i, N', M \rangle) \rangle); new()) \rangle$

$HELP2(\langle N, hash_{k_k}(\langle k, i, N', M \rangle) \rangle); K' =$

$\langle enc_i(\langle K', k, N \rangle), HELP(hash_{k_k}(\langle k, i, N', M \rangle); K') \rangle$

# Tree transducer: the picture

$$F(t) = t'(F(t_0), F(t_1), G(t_0))$$



# The intruder model

## Dolev–Yao intruder!

- controls entire network
- has initial knowledge  $u$
- can derive information according to the following rules:
  - $u \in \text{der}(u)$ ,
  - if  $\langle t, t' \rangle \in \text{der}(u)$ , then  $t, t' \in \text{der}(u)$ ,
  - if  $k \in K_s$  and  $k, \text{enc}_k(t) \in \text{der}(u)$ , then  $t \in \text{der}(u)$ ,
  - if  $k \in K_a$  and  $k^{-1}, \text{enc}_k(t) \in \text{der}(u)$ , then  $t \in \text{der}(u)$ ,
  - if  $\text{hash}_k(t) \in \text{der}(u)$ , then  $t \in \text{der}(u)$ ,
  - if  $t, t' \in \text{der}(u)$ , then  $\langle t, t' \rangle \in \text{der}(u)$ ,
  - if  $k, t \in \text{der}(u)$ , then  $\text{hash}_k(t), \text{enc}_k(t) \in \text{der}(u)$ .

# The protocol model

principal =

finite sequence of receive-send actions

protocol = finite set of principals

protocol run =

interleaving of the receive-send actions of the principals  
with intruder inbetween

protocol is **insecure** (the intruder is successful) iff

in some run the intruder can derive an atom or an  
anonymous constant output in the last step of the  
protocol

## Example

$$OUT(enc_{K_0}(\langle K_{0,1}, P_1, N \rangle)) = K_{0,1}$$

# Main result

**Theorem** It is decidable whether a protocol is secure.



# Main result

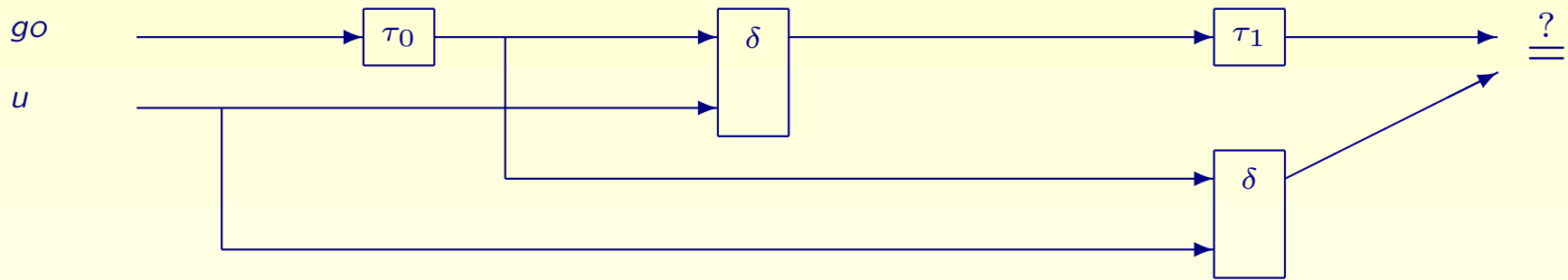
**Theorem** It is decidable whether a protocol is secure.

**Remark** No elementary upper bound for computational complexity known.

# First steps in the proof

$\tau_0, \tau_1$  receive-send actions,  $u$  initial knowledge of intruder.

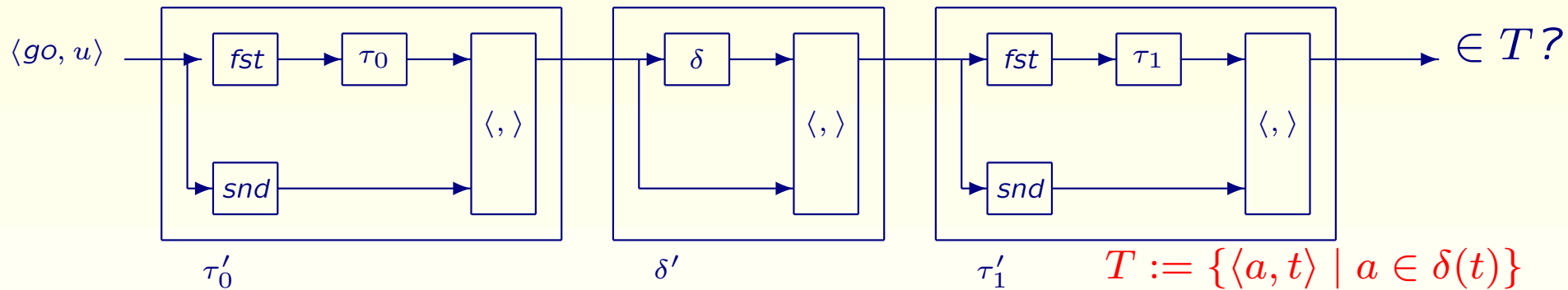
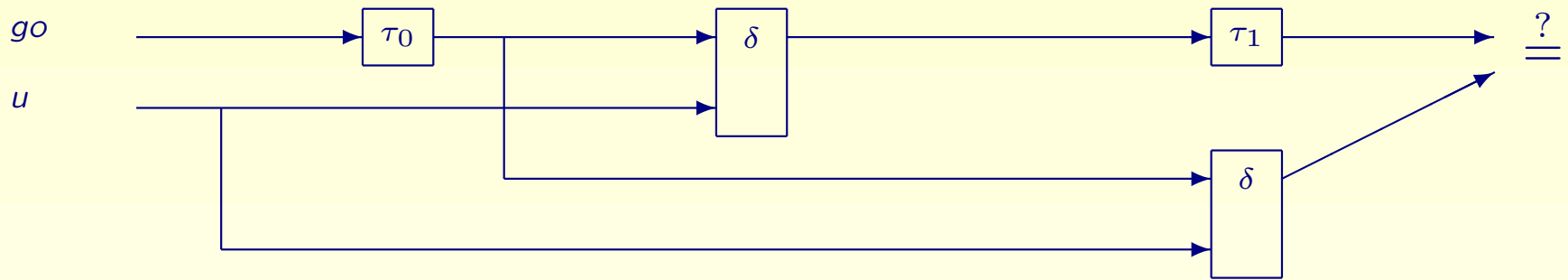
$\delta$  non-deterministic function (relation) for intruder (output can be derived from input).



# First steps in the proof

$\tau_0, \tau_1$  receive-send actions,  $u$  initial knowledge of intruder.

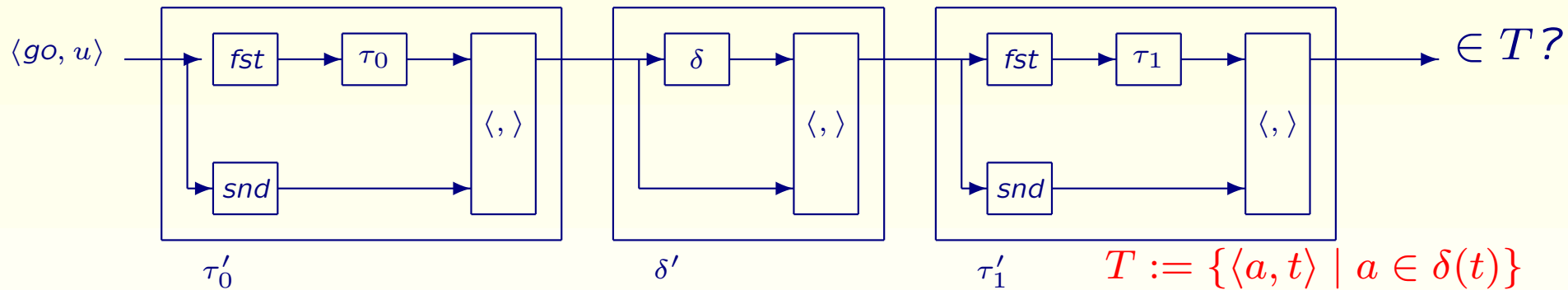
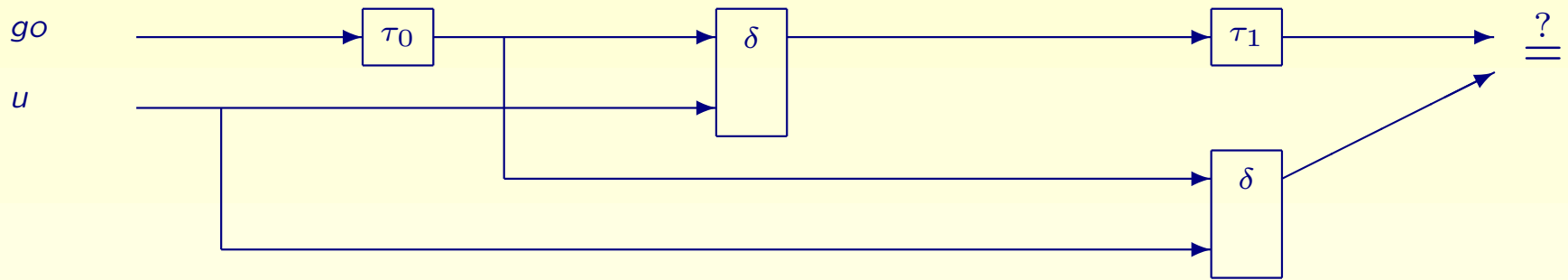
$\delta$  non-deterministic function (relation) for intruder (output can be derived from input).



# First steps in the proof

$\tau_0, \tau_1$  receive-send actions,  $u$  initial knowledge of intruder.

$\delta$  non-deterministic function (relation) for intruder (output can be derived from input).



## General condition

$$\langle go, u \rangle \in \tau'_0{}^{-1} (\dots (\tau'_{n-2}{}^{-1} (\delta'^{-1} (\tau'_{n-1}{}^{-1} (T)))) \dots).$$

# Main steps in the proof

**Show:** the  $\tau'_i$ 's and  $\delta'$  can be modelled as tree transducers.

**Show:**  $T$  can be recognized by a **finite-state tree automaton**, that is,  $T$  is *regular*.

**Show:** **inverse image** of regular set under a tree transducer is regular.

$\Rightarrow \tau_0'^{-1}(\dots(\tau_{n-2}'^{-1}(\delta'^{-1}(\tau_{n-1}'^{-1}(T))))\dots)$  is regular.

**Recall:** “word problem” for regular tree languages is decidable.

$\Rightarrow$  Success of the intruder is decidable.

# Main steps in the proof

**Show:** the  $\tau'_i$ 's and  $\delta'$  can be modelled as tree transducers.

**Show:**  $T$  can be recognized by a **finite-state tree automaton**, that is,  $T$  is *regular*.

**Show:** **inverse image** of regular set under a tree transducer is regular.

$\Rightarrow \tau_0'^{-1}(\dots(\tau_{n-2}'^{-1}(\delta'^{-1}(\tau_{n-1}'^{-1}(T))))\dots)$  is regular.

**Recall:** “word problem” for regular tree languages is decidable.

$\Rightarrow$  Success of the intruder is decidable.

## Notice

Very special tree transducers, very special tree automata.

# A normal form for derived messages

## $an(\cdot)$

- $u \in an(u)$ ,
- if  $\langle t, t' \rangle \in an(u)$ , then  $t, t' \in an(u)$ ,
- if  $k \in K_s$  and  $k, enc_k(t) \in an(u)$ , then  $t \in an(u)$ ,
- if  $k \in K_a$  and  $k^{-1}, enc_k(t) \in an(u)$ , then  $t \in an(u)$ ,

## $syn(\cdot)$

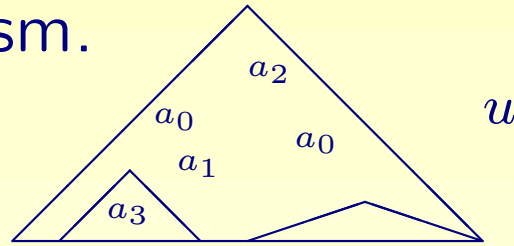
- $S \subseteq syn(S)$ ,
- if  $hash_k(t) \in syn(S)$ , then  $t \in syn(S)$ ,
- if  $t, t' \in syn(S)$ , then  $\langle t, t' \rangle \in syn(S)$ ,
- if  $k, t \in syn(S)$ , then  $hash_k(t), enc_k(t) \in syn(S)$ .

## Lemma [Paulson]

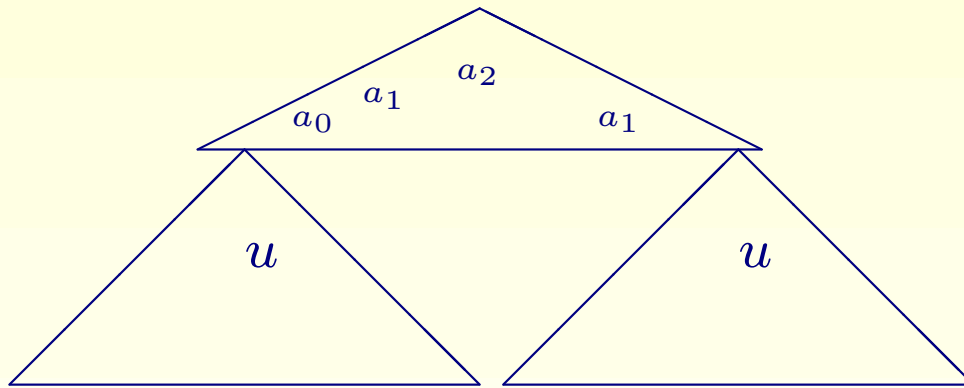
$$der(u) = syn(an(u))$$

# A transducer for the intruder

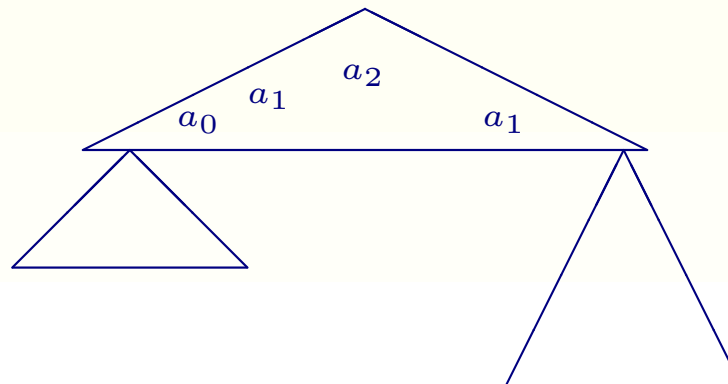
1. Determine  $an(u) \cap A$ : **regular constraints** and  $\epsilon$ -transitions and non-determinism.



2. Generate the top of the derived message:  $\epsilon$ -transitions.



3. Destruct the copies of the original message: “forgetting” transitions.





# Tree automata and regular constraints

Ordinary finite-state bottom-up tree automata, but ...

... only a **fixed number** of anonymous constants can be distinguished.

## Example

The set  $T = \{\langle a, t \rangle : a \in \delta(t)\}$  can be recognized.

# Tree automata and regular constraints

Ordinary finite-state bottom-up tree automata, but ...

... only a **fixed number** of anonymous constants can be distinguished.

## Example

The set  $T = \{\langle a, t \rangle : a \in \delta(t)\}$  can be recognized.

## Regular constraints

$$F(t : L; c_0, \dots, c_{n-1}) = \dots$$

## Example

Transducer for intruder: for each possible set of derivable keys a different rule.

# Undecidable extensions

- allowing non-linear LHS
- allowing complex keys
- adding Xor
- adding Diffie–Hellman exponentiation

# Summary

A Dolev-Yao protocol model for **recursive** protocols . . .

. . . but only a **bounded** number of sessions . . .

. . . and algebraic extensions undecidable.

Main tool: **finite-state automata**—model and decision procedure.

Potential area of application: group protocols, web services protocols, more?

# Current and future work

- determine exact complexity of security problem
- implement the decision procedure, study more examples
- study restrictions and refinements
- develop right automata “practice”
- take into account other security goals