# Model-based Security Engineering with UML

Jan Jürjens
Competence Center for IT Security
Software & Systems Engineering
TU Munich, Germany
juerjens@in.tum.de
http://www.jurjens.de/jan

## A Need for Security

Society and economies rely on computer networks for communication, finance, energy distribution, transportation...

Attacks threaten economical and physical integrity of people and organizations.

Interconnected systems can be attacked anonymously and from a safe distance.

Networked computers need to be secure.



## Problems

Many flaws found in designs of security-critical systems, sometimes years after publication or use.

Spectacular Example (1997):

NSA hacker team breaks into U.S. Department of Defense computers and the U.S.electric power grid system. Simulates power outages and 911 emergency telephone overloads in Washington, D.C..



## Causes I

- Designing secure systems correctly is difficult.
  Even experts may fail:
  - Needham-Schroeder protocol (1978)
  - attacks found 1981 (Denning, Sacco), 1995 (Lowe)
- Designers often lack background in security.
- Security as an afterthought.



## Causes II

„Blind“ use of mechanisms:

- Security often compromised by circumventing (rather than breaking) them.
- Assumptions on system context, physical environment.

„Those who think that their problem can be solved by simply applying cryptography don`t understand cryptography and don`t understand their problem“ (R. Needham).



## Causes III

„Penetrate-and-patch“ (aka „banana strategy):

- insecure
- disruptive

➔loose customer trust.

Goal: reduce number of flaws arising this way.

## Formal Methods

Lots of very successful research using logic-based methods to analyze systems for security flaws. Often based on specialized, „academic“ notations and concerned with crypto protocols or information flow:

LaPadula, Bell 73; Goguen, Meseguer 82; Millen, Clark, Freedman 87; Burrows, Abadi, Needham 89; Kemmerer 89; Gong, Needham, Yahalom 90; Meadows 91; McLean 94; Focardi, Gorrieri 94; Syverson, van Oorschot 94; Roscoe, Woodcock, Wulf 94; Lowe 96; Schneider 96; Abadi, Gordon 97; Mitchell, Mitchell, Stern 97; Paulson 98, …



## How used in Security Engineering ?

Saltzer, Schröder (1975): Security Design Principles.
Gasser (1988): Formal techniques, not integrated with system development.
Abrams, Jajodia, Podell (1995): Collection of unlinked approaches.
Abadi, Needham; Anderson, Needham (1996): Design rules for security protocols.
Anderson (2001): Use formal techniques for protocols.
Viega, McGraw (2002): How to avoid buffer-overflows.
Seemingly no encompassing, integrated formally based approach.



## Some Open Problems

Secure systems out of (in)secure mechanisms.
Security as pervasive property: vs. dependability, program analysis, formal methods, software engineering, programming languages, compilers, computer architectures, operating systems, reactive systems, …, …
Problem: no integration / coherence.
How to put all this stuff together in a water-tight way within security engineering approach ?
Necessary for security (attacks on boundaries between views / aspects / levels …).



## Towards Use in Practice

Increase security with bounded investment in time, costs (crucial for industry). Idea:

- Extract models from artefacts arising in industrial development and use of security-critical systems (UML models, source code, configuration data).
- Tool-supported theoretically sound efficient automated security analysis.

➔*Model-based Security Engineering*

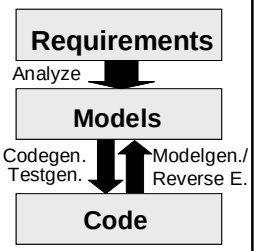


## Model-based Security Engineering

Combined strategy:

- Analyze models automatically against security requirements.
- Generate code (or tests) from models automatically.
- Generate models from code to get changes (or analyze legacy systems).

Goal: model-based = source-based.




Idea notation-independent. Here: use UML.



## Why UML ?

Seemingly de-facto standard in industrial modeling. Large number of developers trained in UML.
Relatively precisely defined (given the user community).
Many tools in development (also for code-generation, testing, reverse engineering, simulation, transformation).

## UMLsec: Goals

Extension for secure systems development.

- evaluate UML specifications for weaknesses in design
- encapsulate established rules of prudent secure engineering as checklist
- make available to developers not specialized in secure systems
- consider security requirements from early design phases, in system context
- make certification cost-effective



## UMLsec: How

Recurring security requirements, adversary scenarios, concepts offered as stereotypes with tags on component-level.

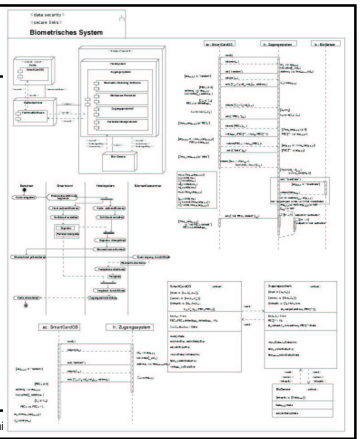
Use associated constraints to verify specifications using automated theorem provers and indicate possible weaknesses.

Ensures that UML specification provides desired level of security requirements.

Link to code via round-trip engineering etc.



Example: Biometric authentication system in industrial development. Secure ?






## This tutorial

Background knowledge on using UML for model-based security engineering.

- UMLsec extension
- Tools.
- Industrial applications (biometry, security protocols, electronic purses, …).
- Attacks against them.

Research-oriented (not user-oriented).



## Requirements on UML extension for security I

Provide basic security requirements such as secrecy, integrity, authenticity.

Allow considering different threat scenarios depending on adversary strengths.

Allow including important security concepts (e.g. *tamper-resistant hardware*).

Allow incorporating security mechanisms (e.g. access control).



## Requirements on UML extension for security II

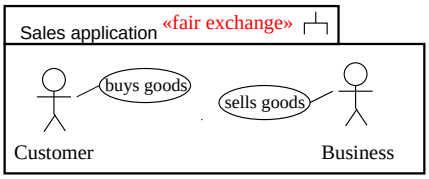
Provide security primitives (e.g. (a)symmetric encryption).

Allow considering underlying physical security.

Allow addressing security management (e.g. secure workflow).

Also: Include domain-specific security knowledge (Java, smart cards, CORBA, ...).

## Requirements with use case diagrams




Capture security requirements in use case diagrams.

Constraint: need to appear in corresponding activity diagram.



## ≪fair exchange≫

Ensures generic fair exchange condition.

Constraint: after a {start} state in activity diagram is reached, eventually reach {stop} state.

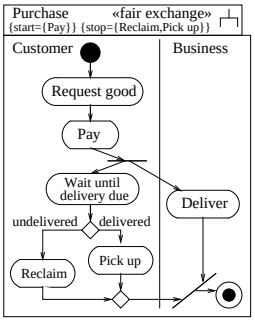
(Cannot be ensured for systems that an attacker can stop completely.)



## Example ≪fair exchange≫

Customer buys a good from a business.

Fair exchange means: after payment, customer is eventually either delivered good or able to reclaim payment.






## ≪Internet≫, ≪encrypted≫, …

Kinds of communication links resp. system nodes.

For adversary type A, stereotype s, have set $Threats_A(s) \in \{delete, read, insert, access\}$ of actions that adversaries are capable of.

Default attacker:

| Stereotype | $Threats_{default}()$ |
|---|---|
| Internet | {delete, read, insert} |
| encrypted | {delete} |
| LAN | ∅ |
| smart card | ∅ |



## ≪secure links≫

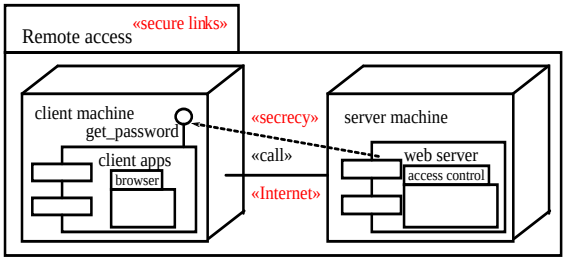
Ensures that physical layer meets security requirements on communication.

Constraint: for each dependency *d* with stereotype $s \in \{\ll secrecy \gg, \ll integrity \gg\}$ between components on nodes $n \neq m$, have a communication link *l* between *n* and *m* with stereotype *t* such that

- if $s = \ll secrecy \gg$: have $read \notin Threats_A(t)$.
- if $s = \ll integrity \gg$: have $insert \notin Threats_A(t)$.

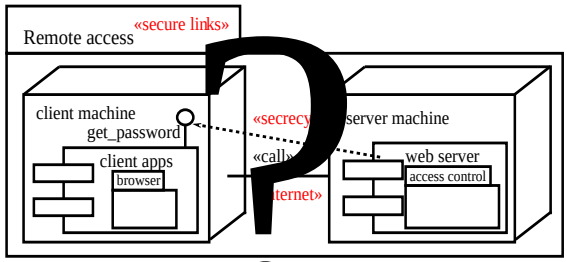


## Example ≪secure links≫




Given default adversary type, is ≪secure links≫ provided ?

## Example ≪secure links≫




Given default adversa pe, is ≪secure links≫ provided ?



## ≪secure dependency≫

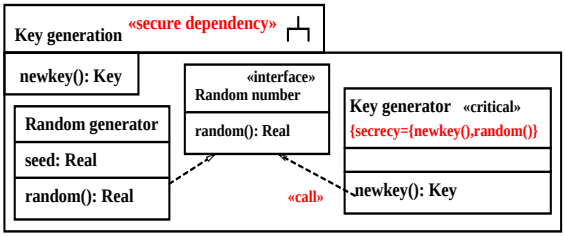
Ensure that ≪call≫ and ≪send≫ dependencies between components respect security requirements on communicated data given by tags {secrecy}, {integrity}.

Constraint: for ≪call≫ or ≪send≫ dependency from *C* to *D* (and similarly for {integrity}):

- Msg in *D* is {secrecy} in *C* if and only if also in *D*.
- If msg in *D* is {secrecy} in *C*, dependency stereotyped ≪secrecy≫.



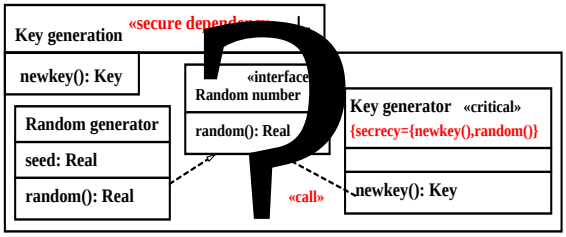
## Example ≪secure dependency≫




≪secure dependency≫ provided ?



## Example ≪secure dependency≫




≪secure dependenc provided ?



## ≪no down–flow≫

Enforce secure information flow.

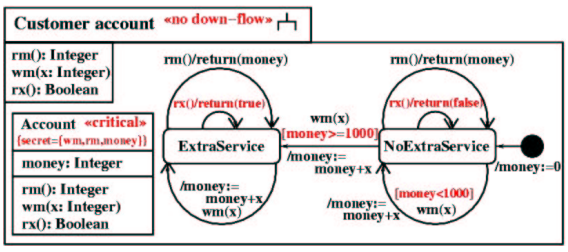
Constraint:

Value of any data specified in {secrecy} may influence only the values of data also specified in {secrecy}.

Formalize by referring to formal behavioural semantics.

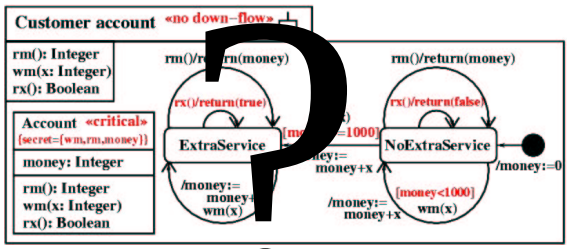


## Example ≪no down-flow≫




≪no down–flow≫ provided ?

## Example ≪no down-flow≫




≪no down–flow≫ pr ed ?



## ≪data security≫

Security requirements of data marked ≪critical≫ enforced against threat scenario from deployment diagram.

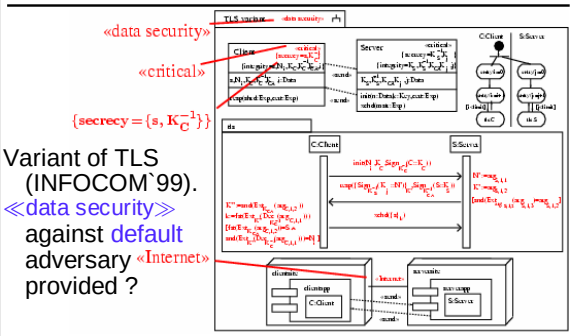
Constraints:

Secrecy of {secrecy} data preserved.

Integrity of {integrity} data preserved.



## Example ≪data security≫

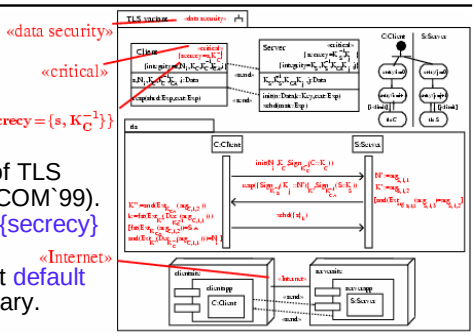



Variant of TLS (INFOCOM`99).

≪data security≫ against default adversary «Internet» provided ?



## Example ≪data security≫




Variant of TLS (INFOCOM`99).

Violates {secrecy} of s against default adversary.



## ≪guarded access≫

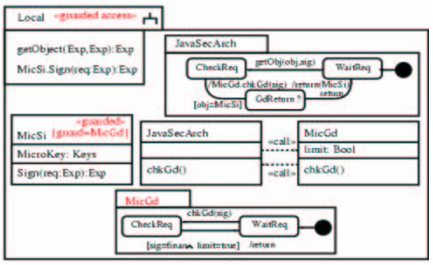
Ensures that in Java, ≪guarded≫ classes only accessed through {guard} classes.

Constraints:

- References of ≪guarded≫ objects remain secret.
- Each ≪guarded≫ class has {guard} class.



## Example ≪guarded access≫




Provides ≪guarded access≫: Access to MicSi protected by MicGd.

## Does UMLsec meet requirements?

**Security requirements**: ≪secrecy≫,…

**Threat scenarios:** Use $\text{Threats}_{adv}$(ster).

**Security concepts:** For example ≪smart card≫.

**Security mechanisms:** E.g. ≪guarded access≫.

**Security primitives:** Encryption built in.

**Physical security:** Given in deployment diagrams.

**Security management:** Use activity diagrams.

**Technology specific:** Java, CORBA security.



## UMLsec as Integrating Formal Framework

Have formalizations of major security requirements in one integrated notation.

Want to relate / combine requirements; get modularity / composability, hierarchical decomposition, refinement, … :

For example:

- If system satisfies ≪secure links≫ and subsystems satisfy ≪data security≫ then system satisfies ≪data security≫.



## Summary Lecture 1

Defined UMLsec extension.

Goal: express security requirements within an industrially used specification notation in a way which allows automated verification.

Aims:

- model-based security engineering, integrated with source-code, configuration data
- formal framework to relate different security aspects.

Coming up in Lecture 2: formal security analysis



## Security Analysis

Specify system parts as processes following Dolev, Yao 1982: In addition to expected participants, model attacker who:

- may participate in some protocol runs,
- knows some data in advance,
- may intercept messages on the public network,
- injects messages that it can produce into the public network.



## Security Analysis

Model classes of adversaries.

May attack different parts of the system according to threat scenarios.

Example: insider attacker may intercept communication links in LAN.

To evaluate security of specification, verify against adversary model.



## Security Analysis II

Keys are symbols, crypto-algorithms are abstract operations.

- Can only decrypt with right keys.
- Can only compose with available messages.
- Cannot perform statistical attacks.

## Cryptographic Expressions

*Exp*: quotient of term algebra generated from sets *Data*, *Keys*, *Var* of symbols using

- *_::_* (concatenation), *head(_)*, *tail(_)*,
- $(\_)^{-1}$ (inverse keys)
- *{ _ }_* (encryption)
- *Dec_( )* (decryption)
- *Sign_( )* (signing)
- *Ext_( )* (extracting from signature)

under equations …



## Cryptographic Expressions

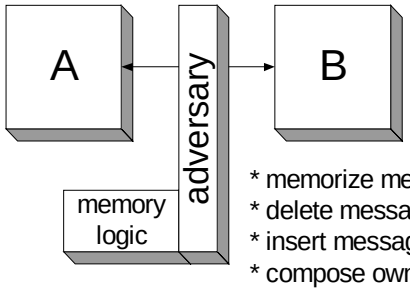
- $\forall E,K.Dec_K^{-1}(\{E\}_K)=E$
- $\forall E,K.Ext_K(Sign_K^{-1}(E))=E$
- $\forall E_1,E_2.head(E_1::E_2)=E_1$
- $\forall E_1,E_2.tail(E_1::E_2)=E_2$
- Associativity for *::*.

Write $E_1::E_2::E_3$ for $E_1::(E_2::E_3)$ and $fst(E_1::E_2)$ for $head(E_1::E_2)$ etc.

Can include further crypto-specific primitives and laws (XOR, …).



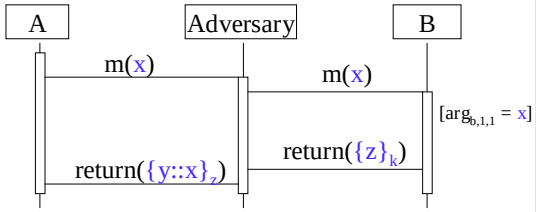
## Adversary Model




* memorize message
* delete message
* insert message
* compose own message
* use cryptographic primitives



## Adversary: Simulation




Adversary knowledge: $k^{-1}$, y, x, $\{z\}_k$, z

- $\forall e,k.\mathcal{D}ec_{k^{-1}}(\{e\}_k)=e$



## Abstract adversary

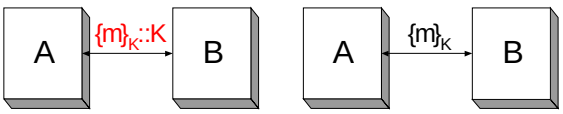
Specify set $K_A^0$ of initial knowledge of an adversary of type *A*. Let $K_A^{n+1}$ be the Exp-subalgebra generated by $K_A^n$ and the expressions received after *n+1*st iteration of the protocol.

Definition (Dolev, Yao 1982).

*S* keeps secrecy of *M* against attackers of type *A* if there is no *n* with $M \in K_A^n$ .



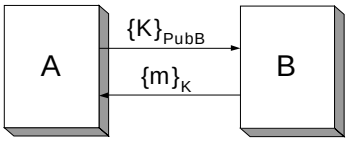
## Encryption vs. Secrecy




Against eavesdropper:

- Secrecy of *m, K* not preserved.
- Secrecy of *m, K* preserved.

## Hybrid Crypto vs. Secrecy




- Secrecy of *m* not preserved against an attacker who can delete and insert messages.
- Secrecy of *m* preserved against an attacker who can eavesdrop, but not alter the link.



## Security analysis in first-order logic

Idea: approximate set of possible data values flowing through system from above.

Predicate *knows(E)* meaning that the adversary may get to know *E* during the execution of the protocol.

For any secret *s*, check whether can derive *knows(s)* using automated theorem prover.



## First-order logic: basic rules

For initial adversary knowledge ($K^0$): Define *knows(E)* for any *E* initially known to the adversary (protocol-specific, e.g. $K_A$, $K_A^{-1}$).

Define above equations.

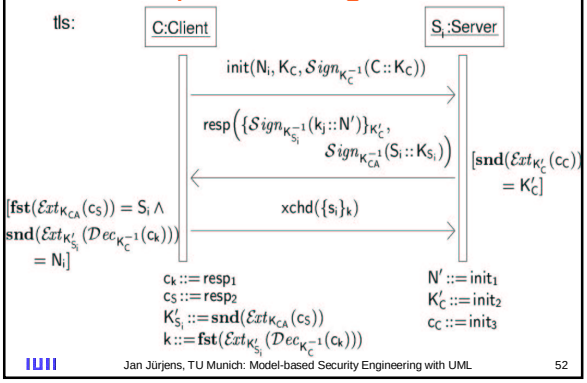
For evolving knowledge ($K^n$) define

$\forall E_1,E_2.(knows(E_1) \wedge knows(E_2) \Rightarrow$
$knows(E_1::E_2) \wedge knows(\{E_1\}_{E2}) \wedge$
$knows(Dec_{E2}(E_1)) \wedge knows(Sign_{E2}(E_1)) \wedge$
$knows(Ext_{E2}(E_1)))$

$\forall E.(knows(E) \Rightarrow$
$knows(head(E)) \wedge knows(tail(E)))$

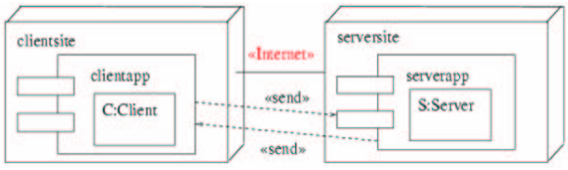


## Given Sequence Diagram …






## … and Physical Layer Model …




Deployment diagram.

Derived adversary model: read, delete, insert data.



## … Translate to 1st Order Logic

Connection (or statechart transition) *TR1=(in(msg_in),cond(msg_in),out(msg_out))*

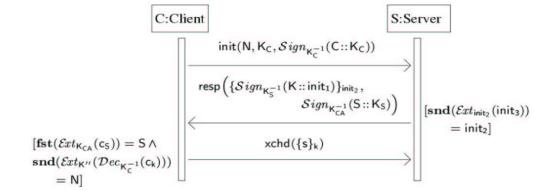
followed by *TR2* gives predicate *PRED(TR1)=*
$\forall msg\_in.\ [knows(msg\_in) \wedge cond(msg\_in)$
$\Rightarrow knows(msg\_out)$
$\wedge PRED(TR2)]$

(Assume: order enforced (!).)

Can include senders, receivers in messages.

Abstraction: find all attacks, may have false positives.

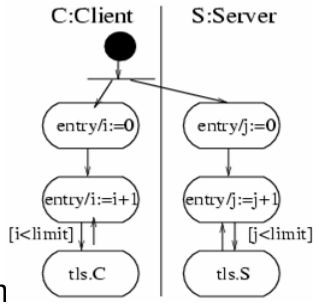
## Example: Translation to Logic




$knows(N) \wedge knows(K_C) \wedge knows(Sign_{KC\text{-}1}(C::K_C))$
$\wedge \forall init_1, init_2, init_3.[knows(init_1) \wedge knows(init_2) \wedge$
$knows(init_3) \wedge snd(Ext_{init2}(init_3)) = init_2$
$\Rightarrow knows(\{Sign_{KS\text{-}1}(\ldots)\}_{\ldots}) \wedge [\ldots] \wedge [\ldots \Rightarrow \ldots] \ldots]$



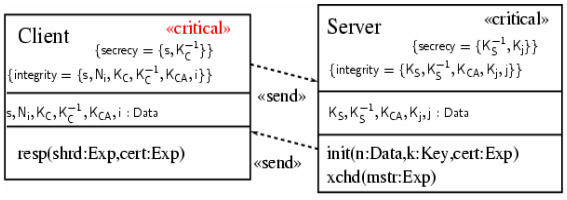
## Execute in System Context




Activity diagram



## Formulate Data Security Requirements




Class diagram.

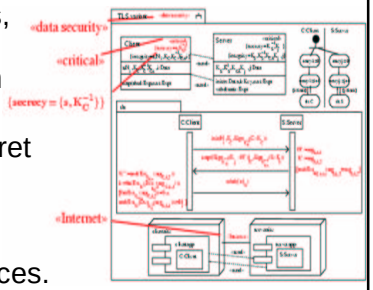
Gives conjecture: *knows(s)* derivable ?



## Example: Proposed Variant of TLS (SSL)

Apostolopoulos, Peris, Saha; IEEE Infocom 1999.

Goal: send secret protected by session key using fewer server resources.






## TLS Variant in TPTP notation I

```
input_formula(tls_abstract_protocol,axiom,(
![ArgS_11, ArgS_12, ArgS_13, ArgC_11, ArgC_12] : (
  ![DataC_KK, DataC_k, DataC_n] : (
    % Client -> Attacker (1. message)
    (      knows(n)
         & knows(k_c)
         & knows(sign(conc(c, k_c), inv(k_c) ) ) )
  & % Server -> Attacker (2. message)
    ( (   knows(ArgS_11)
         & knows(ArgS_12)
         & knows(ArgS_13)
         & ( ? [X] : equal( sign(conc(X, ArgS_12), inv(ArgS_12) ),
                            ArgS_13 ) ) )
    => (   knows(enc(sign(conc(kgen(ArgS_12), ArgS_11), inv(k_s) ),
                     ArgS_12 ) )
         & knows(sign(conc(s, k_s), inv(k_ca) ) ) ) ) )
```



## TLS Variant in TPTP notation II

```
  & % Client -> Attacker (3. message)
    ( (   knows(ArgC_11)
        & knows(ArgC_12)
        & equal(sign(conc(s, DataC_KK), inv(k_ca)), ArgC_12 )
        & equal(enc(sign(conc(DataC_k, DataC_n), inv(DataC_KK) ),
                    k_c), ArgC_11 )
        & ( ? [DataC_ks] : equal(sign(conc(s, DataC_ks), inv(k_ca) ),
                                 ArgC_12 ) )
        & equal(enc(sign(conc(DataC_k, n), inv(DataC_KK) ), k_c),
                ArgC_11 )
        & equal(enc(sign(conc(DataC_k, DataC_n), inv(DataC_KK) ), k_c)
                ArgC_11 )
      )
    => ( knows(symenc(secret, DataC_k)) ) )
) ) )).
```

## Surprise …

```
E-SETHEO csp03 single processor running on host ...
(c) 2003 Max-Planck-Institut fuer Informatik and
    Technische Universitaet Muenchen

tlsvariant-freshkey-check.tptp
...
time limit information: 300 total (entering statistics module).
problem analysis ...
testing if first-order ...
first-order problem
...
statistics: 19 0 7 46 3 6 2 0 1 2 14 8 0 2 28 6
...
schedule selection: problem is horn with equality (class he).
schedule:605 3 300 597
...
entering next strategy 605 with resource 3 seconds.
...
analyzing results ...
proof found
time limit information: 298 total / 297 strategy (leaving wrapper).
...
e-SETHEO done. exiting
```

Attack



## … which means:

Can derive *knows(s)* (!).

That is: Protocol does not preserve secrecy of *s* against adversaries.

➔ Completely insecure wrt stated goals.

But why ?

Could look at proof tree.

Or: use prolog-based attack generator.



## Man-in-the-Middle Attack

?



## The fix

?

e-Setheo: *knows(s)* not derivable. Thus secure.



## Summary Lecture 2

Automated formal security analysis for security requirements included in UMLsec models as stereotypes.

Running example: TLS variant with security flaw.

Exercise: find flaw, propose correction ☺.

Coming up tomorrow: Electronic purses, biometric authentication systems (including more flaws). General results and reasoning techniques.