Lecture 3

Solution: flaw in TLS variant.

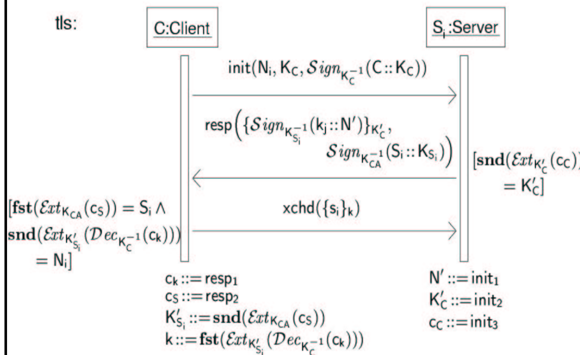
Industrial applications and flaws:

- smart-card based electronic purse scheme
- biometric authentication system

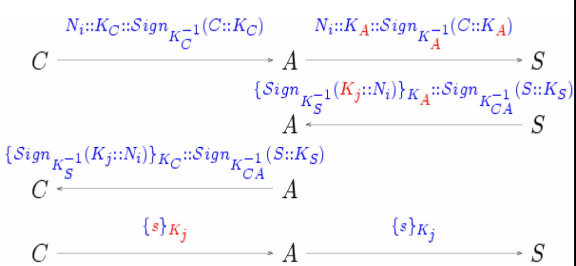
Lecture 4:

- General results and reasoning techniques.
- Tool demo.

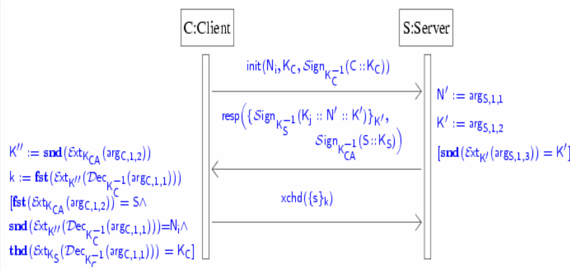
TLS Variant



Man-in-the-Middle Attack

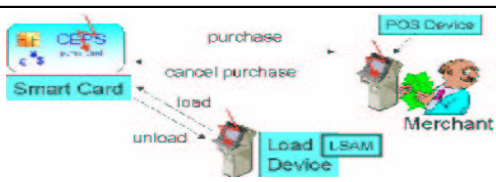


The Fix



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

Common Electronic Purse Specifications



Global electronic purse standard (90% of market).

Smart card contains account balance. Chip performs cryptographic operations securing the transactions.

More fraud protection than credit cards (transaction-bound authorisation).

FAIRPAY

Load Protocol

Unlinked, cash-based load transaction (on-line).

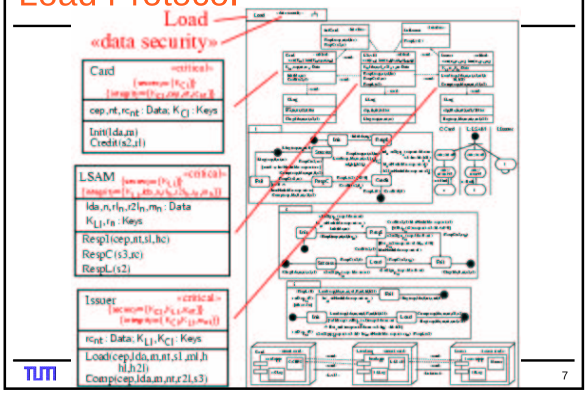
Load value onto card using cash at load device.

Load device contains Load Security Application Module (LSAM): secure data processing and storage.

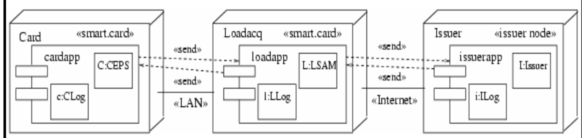
Card account balance adjusted; transaction data logged and sent to issuer for financial settlement.

Uses symmetric cryptography.

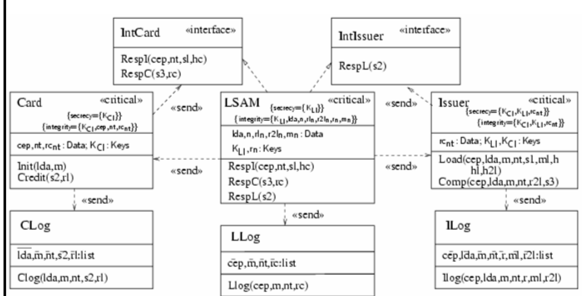
Load Protocol



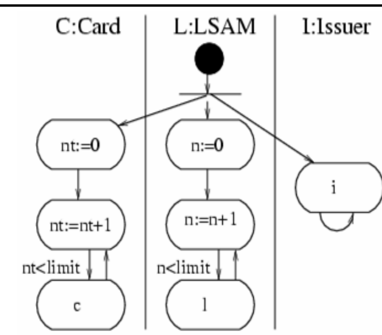
Load Protocol: Physical View



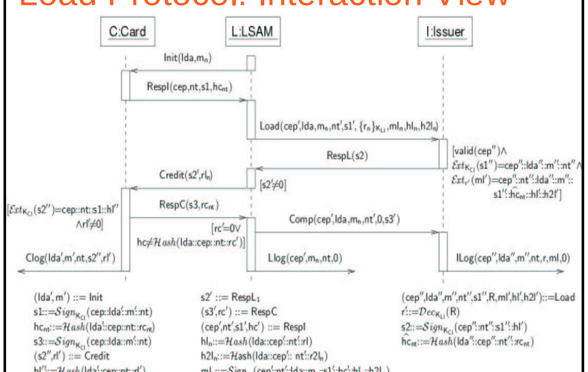
Load Protocol: Structural View



Load Protocol: Coordination View



Load Protocol: Interaction View



Security Threat Model

Card, LSAM, issuer security module assumed tamper-resistant.

Intercept communication links, replace components.

Possible attack motivations:

- **Cardholder**: charge without pay
- **Load acquirer**: keep cardholder's money
- **Card issuer**: demand money from load acquirer

May coincide or collude.

Audit Security

No direct communication between card and cardholder. Manipulate load device display.

Use post-transaction settlement scheme.

Relies on secure auditing.

Verify this here (only executions completed without exception).

Security Conditions (informal)

Cardholder security If card appears to have been loaded with *m* according to its logs, cardholder can prove to card Issuer that a load acquirer owes *m* to card issuer.

Load acquirer security Load acquirer has to pay *m* to card issuer only if load acquirer has received *m* from cardholder.

Card issuer security Sum of balances of cardholder and load acquirer remains unchanged by transaction.

Load Acquirer Security

Suppose card issuer *I* possesses $ml_n = Sign_{rn}(cep::nt::lda::m_n::s1::hc_{nt}::hl_n::h2l_n)$ and card *C* possesses rl_n, where $hl_n = Hash(lda::cep::nt::rl_n)$.

Then after execution either of following hold:

- $Llog(cep,lda,m_n,nt)$ has been sent to l:LLog (so load acquirer L has received and retains m_n in cash) or
- $Llog(cep, lda, 0, nt)$ has been sent to l : LLog (so L returns m_n to cardholder) and L has received rc_{nt} with $hc_{nt} = Hash(lda::cep::nt::rc_{nt})$ (negating ml_n).

"ml_n provides guarantee that load acquirer owes transaction amount to card issuer" (CEPS)

Flaw

L does not provide load acquirer security against adversaries of type insider.

Why ?

Flaw

?

Correction

?

Major Industrial Application

Verisoft project. Goal: Practical application of formal methods.

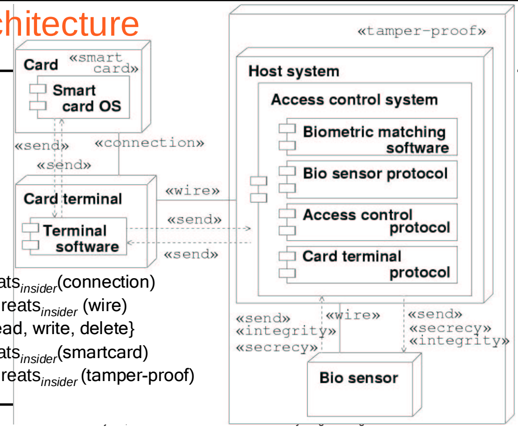
Planned for 8 years from 7/2003; 10 industrial + academic partners.

Integrated formal verification from application software down to operating system and prozessor on C-code level. Security-relevant:

- Biometric access control system
- Automotive emergeny application

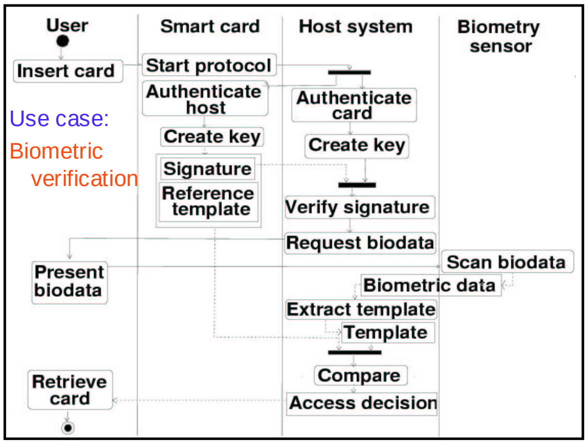
Apply UMLsec approach, tools.

Architecture

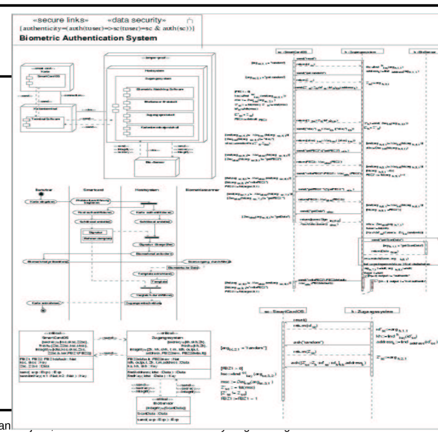


$Threats_{insider}$(connection)
= $Threats_{insider}$ (wire)
= {read, write, delete}
$Threats_{insider}$(smartcard)
= $Threats_{insider}$ (tamper-proof)
= Ø

Use case:
Biometric verification



Big Picture



Translation to First-order Logic II

Message order not enforced by smart card (!).

Connection from smart card
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)*

Crypto Protocol Part 2: Problem ?

?

Some Further UMLsec Applications

Java Security Architecture, Security Architecture Patterns (➔ Saturday)

Secure Design Principles by Saltzer, Schroeder

Telematic automobile emergency application of German car company

Electronic signature architecture of German insurance company

Electronic purse for Oktoberfest

Summary Lecture 3

Conclusions:

- Security really is difficult.
- There really are a lot of security flaws in industrially developed and used systems.
- Many of them can actually be detected on the specification level in a model-based approach.
- This can be done using automated tool support.

Lecture 4:

- General results and reasoning techniques.
- Tool demo.

Secure Channel Abstractions

So far, usually concentrated on specific properties of protocols in isolation.

Need to refine security properties so protocol is still secure in system context. Surprisingly problematic.

Motivates research towards providing secure channel abstractions to use security protocols securely in the system context.

Refinement Problem

Common formalizations of security properties not preserved by refinement (!).

Bad: re-verify after each refinement.

Code is refinement of spec !

Refinement Problem: Examples

if H=0 then (0 or 1) else (0 or 1)

Might view as secure. Might refine to:

if H=0 then 0 else 1

choose K_1 *or … or choose* K_n

Secure for large n, but not:

choose K_1

Refinement Problem: Causes

At least two kinds of non-determinism:

- under-specification
- unpredictability

Refinement: Get rid of under-specification.

Security: Keep unpredictability.

Some formalisms model both kinds by same non-determinsm operator.

➔ Problem.

Refinement Problem: Solution

Separate two kinds of non-determinism:

Usual non-determinism = under-specification (e.g. choice between firable transitions).

Security formalized so all resolutions must satisfy it. Preserved by refinement.

For unpredictability, use only dedicated operators (nonce generation, …). Not removed by refinement.

Modularity

Can also show formalizations of security properties are composable (rely/guarantee style).

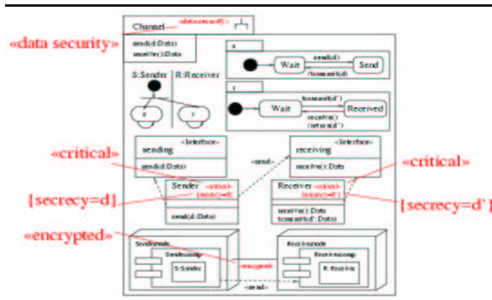
Have initial results for secure information flow.

Secure Channel: Approach

- Define a secure channel abstraction.
- Define concrete secure channel (protocol).
- Show simulates the abstraction.

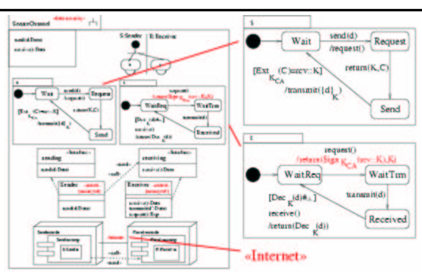
Give conditions under which it is secure to substitute channel abstractions by concrete protocols.

Secure Channel Pattern: Problem



To keep *d* secret, must be sent encrypted.

Secure Channel Pattern: (Toy) Solution



Exchange certificate and send encrypted data over Internet.

Secure Channel Abstraction

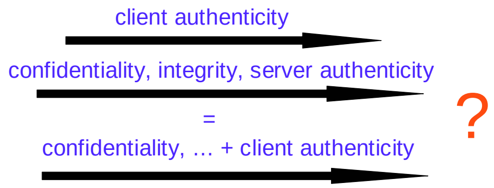
Show concrete channel is delayed equivalent to abstract channel, given a reasonable adversary model.

Use notion of delayed bisimulation.

Details: talk on Saturday (Vodca'04).

Layered Security Protocols

Protokol *on top* uses security *below*.



Security properties additive ?

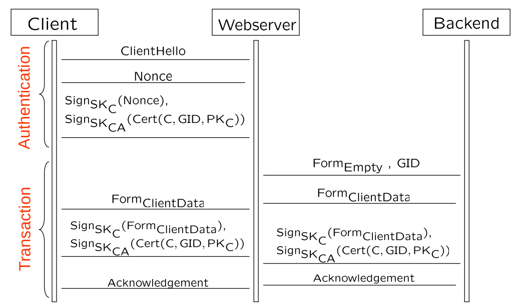
Here: Bank application

- Security analysis of web-based banking application, to be put to commercial use (clients fill out and sign digital order forms).
- In cooperation with major German bank.
- Layered security protocol
  – first layer: SSL protocol.
  – second layer: client authentication protocol
- Main security requirements:
  – personal data confidential.
  – orders not submitted in name of others.

The Application II

- Two layer architecture.
- When user logs on, an SSL-connection is established (first layer).
  – Provides secrecy, integrity, server authentication but no client authentication (this version).
- Custom-made protocol on top of SSL for client authentication.
- Session key generated by SSL used to encrypt messages on second layer.

Authentication protocol



Layered Security Protocol

- Adjust adversary model to account for SSL security properties.
- Justify that specialised adversary model wrt. top-level protocol is as powerful as generic adversary wrt. protocol composition.
- Verify top-level protocol wrt. specialised adversary.
- Implies verification of protocol composition.

Insight

Protocol layering indeed additive wrt. security properties in this particular case.

Generalize to classes of protocols and security requirements.

Beyond Specification Analysis

Model-based test generation.

Configuration analysis.

- Analyze permission data using Prolog (e.g. SAP R/3)
- Analyze firewall configurations using model-checkers

Source-code analysis (C).

Beyond Security

Apply to other non-functional requirements

- fault tolerance
- safety
- dependability
- real-time

Relate to security.

Tool-supported analysis

Draw UML models with editor.

Save UML models as XMI (XML dialect).

Connect to verification tools (automated theorem prover, model-checker …), e.g. using XMI Data Binding.

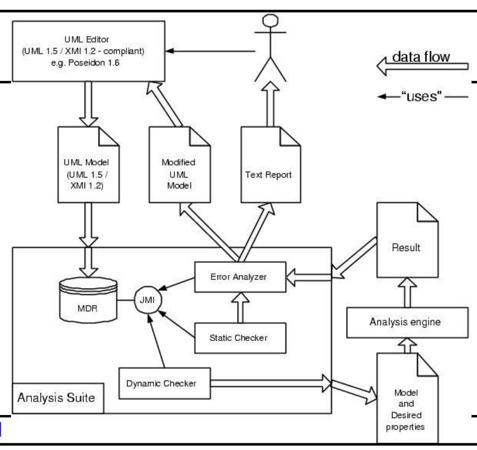
UMLsec Framework

Framework for analysis plug-ins to access UML models on conceptual level over various UI's.

Exposes a set of commands. Has internal state (preserved between command calls).

Framework and analysis tools accessible and available at http://www4.in.tum.de/~umlsec .

Upload UML model (as .xmi file) on website. Analyse model for included security requirements. Download report and UML model with highlighted weaknesses.



Connection with analysis tool

Industrial CASE tool with UML-like notation: AUTOFOCUS (http://autofocus. informatik.tu-muenchen.de)

- Simulation
- Validation (Consistency, Testing, Model Checking)
- Code Generation (e.g. Java, C, Ada)
- Connection to Matlab

Connect UML tool to underlying analysis engine.

Summary Lecture 4

Conclusions:

- Have general results and reasoning techniques.
- Have tool support for automatically checking UMLsec constraints.
- Can apply approach as well to models generated from configuration data, source code.
- Can apply to other non-functional requirements.

Conclusions

Model-based Security Engineering using UML:

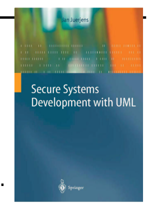
- formally based approach
- automated tool support
- industrially used notation
- integrated approach (source-code, configuration data)

Ongoing Work, Open Problems

Ongoing work on most of the above issues:

- Security properties: E.g. composability
- Crypto verification: crypto-specific equations
- Tools: E.g. Extensibility for self-defined stereotypes
- Source-code analysis: extract Dolev-Yao model
- Application domains: Mobility

Resources



Jan Jürjens, Secure Systems Development with UML, Springer 04 (get Oct.)

Tutorials: Sept.: SAFECOMP (Potsdam), ASE (Linz), NODe (Erfurt). Oct.: UML (Lisabon). Nov.: ISSRE (Bretagne).

Spring School: May 2005, Carlos IV Univ. Madrid

Workshops: CSDUML@UML04, WITS05@POPL05

More information (papers, slides, tool etc.): http://www.umlsec.org