Critical Systems Development with UML: Methods and Tools

Jan Jürjens

Software & Systems Engineering
TU Munich, Germany

juerjens@in.tum.de
http://www.jurjens.de/jan

Personal introduction + history

Me: Leading the Competence Center for IT-Security at Software & Systems Engineering, TU Munich

- Extensive collaboration with industry (BMW, HypoVereinsbank, T-Systems, Deutsche Bank, Siemens, Infineon, Allianz, …)
- PhD in Computer Science from Oxford Univ., Masters in Mathematics from Bremen Univ.
- Numerous publications incl. 1 book on the subject

This tutorial: part of series of 30 tutorials at international conferences. Continuously improved (please fill in feedback forms).

Critical Systems Development

High quality development of critical systems (dependable, security-critical, real-time,...) is difficult.

Many systems developed, deployed, used that do not satisfy their criticality requirements, sometimes with spectacular failures.

Quality vs. cost

Systems on which human life and commercial assets depend need careful development.

Systems operating under possible system failure or attack need to be free from weaknesses.

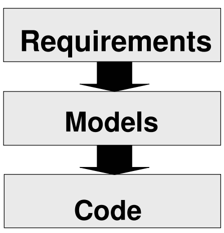
Correctness in conflict with cost.

Thorough methods of system design not used if too expensive.

Model-based Development

Goal: easen transition from human ideas to executed systems.

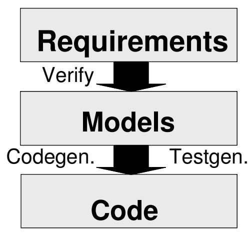
Increase quality with bounded time-to-market and cost.



Model-based Development

Combined strategy:

- Verify models against requirements
- Generate code from models where reasonable
- Write code and generate test-sequences otherwise.



Using UML

UML: unprecedented opportunity for high-quality and cost- and time-efficient critical systems development:

- De-facto standard in industrial modeling: large number of developers trained in UML.
- Relatively precisely defined (given the user community).
- Many tools (drawing specifications, simulation, …).

Challenges

- Adapt UML to critical system application domains.
- Correct use of UML in the application domains.
- Conflict between flexibility and unambiguity in the meaning of a notation.
- Improving tool-support for critical systems development with UML (analysis, …).

UML for CSD: Goals

Extensions for critical systems development.

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

The CSDUML profiles

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

Use associated constraints to evaluate specifications and indicate possible weaknesses.

Ensures that UML specification provides desired level of critical requirements.

Link to code via test-sequence generation.

This tutorial

Background knowledge on using UML for critical systems development.

- UML basics, including extension mechanisms.
- Extensions of UML (UMLsafe, UMLsec, ...)
- UML as a formal design technique.
- Model-based testing.
- Tools.
- Case studies.

Concentrate on safety and security.

Generalize to other application domains.

Before we start …

More material than useful to cover within the given time frame.

Make selection based on your background / interests:

- UML background (no, beginner, advanced)
- working background (industrial, academic)
- application domain interest (security, safety)

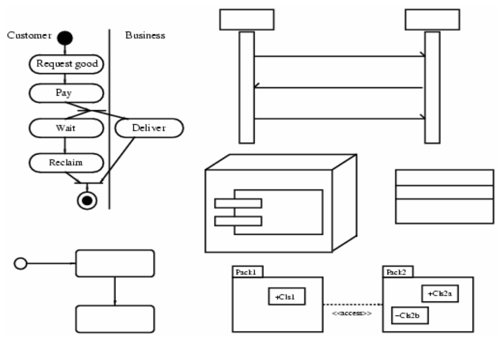
Roadmap

UML

Unified Modeling Language (UML):

- visual modelling for OO systems
- different views on a system
- high degree of abstraction possible
- de-facto industry standard (OMG)
- standard extension mechanisms

A glimpse at UML

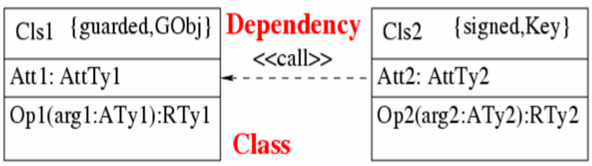


Used fragment of UML

Use case diagram: discuss requirements of the system
Class diagram: data structure of the system
Statechart diagram: dynamic component behaviour
Activity diagram: flow of control between components
Sequence diagram: interaction by message exchange
Deployment diagram: physical environment
Package/Subsystem: collect diagrams for system part

Current: UML 1.5 (released Mar 2003)

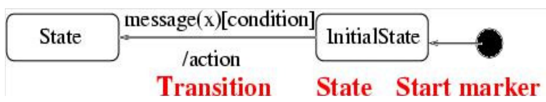
UML run-through: Class diagrams



Class structure of system.

Classes with attributes and operations/signals; relationships between classes.

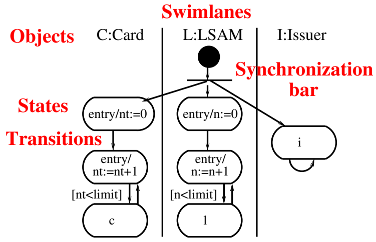
UML run-through: Statecharts



Dynamic behaviour of individual component.

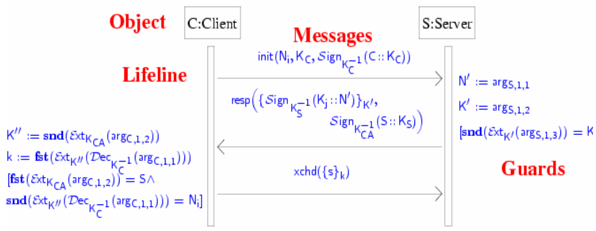
Input events cause state change and output actions.

UML run–through: Activity diagrams



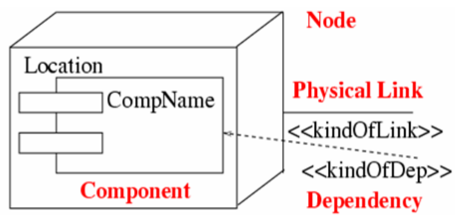
Specify the control flow between components within the system, at higher degree of abstraction than statecharts and sequence diagrams.

UML run-through: Sequence Diagrams



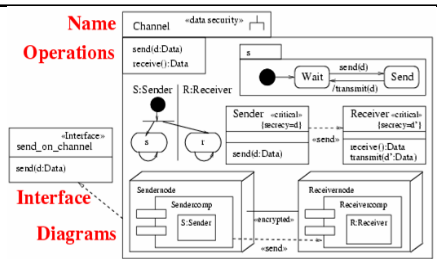
Describe interaction between objects or components via message exchange.

UML run-through: Deployment diagrams



Describe the physical layer on which the system is to be implemented.

UML run-through: Package



May be used to organize model elements into groups.

UML extension mechanisms

Stereotype: specialize model element using ≪label≫.

Tagged value: attach {tag=value} pair to stereotyped element.

Constraint: refine semantics of stereotyped element.

Profile: gather above information.

Roadmap

Prologue
UML
UMLsec
Security Analysis

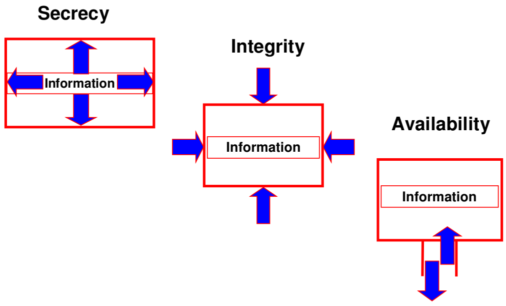
UMLsafe
Towards UML 2.0
Model-based Testing
Tools

Security: Problems

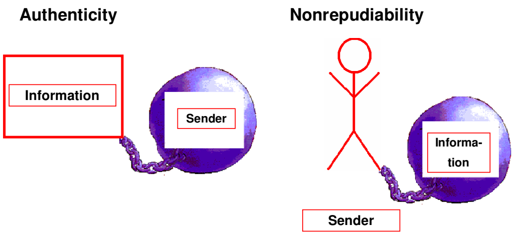
„Blind“ use of mechanisms:

- Security often compromised by circumventing (rather than breaking) them.
- Assumptions on system context, physical environment.
- „Trust us, we use SSL !“ doesn‘t work

Basic Security Requirements I



Basic Security Requirements II



Requirements on UML extension for security I

Mandatory requirements:

- Provide basic security requirements such as secrecy and integrity.
- 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).

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

UMLsec: general ideas

Activity diagram: secure control flow, coordination

Class diagram: exchange of data preserves security levels

Sequence diagram: security-critical interaction

Statechart diagram: security preserved within object

Deployment diagram: physical security requirements

Package: holistic view on security

UMLsec profile (excerpt)

Stereotype	Base class	Tags	Constraints	Description
Internet	link			Internet connection
secure links	subsystem		dependency security matched by links	enforces secure communication links
secrecy	dependency			assumes secrecy
secure dependency	subsystem		call, send respect data security	structural interaction data security
no down-flow	subsystem	high	prevents down-flow	information flow
data security	subsystem		provides secrecy, integrity	basic datasec requirements
fair exchange	package	start, stop	after start eventually reach stop	enforce fair exchange
guarded access	Subsystem		guarded objects acc. through guards.	access control using guard objects

≪Internet≫, ≪encrypted≫, …

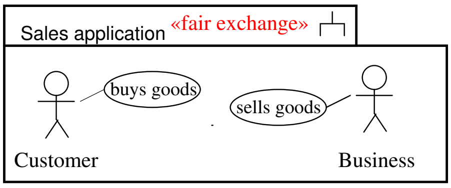
Kinds of communication links resp. system nodes.

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

Default attacker:

Stereotype	$\mathrm{Threats}_{default}()$
Internet	{delete, read, insert}
encrypted	{delete}
LAN	∅
smart card	∅

Requirements with use case diagrams



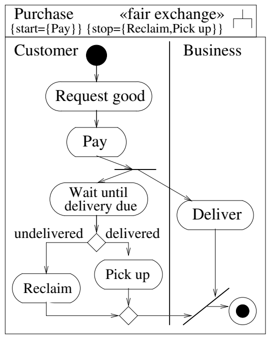
Capture security requirements in use case diagrams.

Constraint: need to appear in corresponding activity diagram.

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.



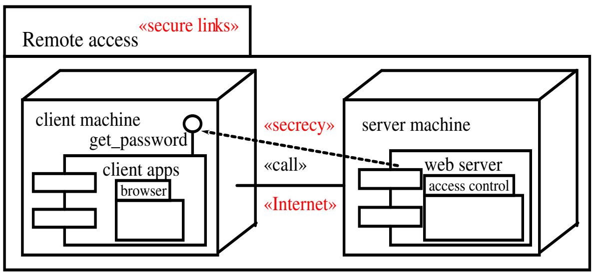
≪fair exchange≫

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 ≪secure links≫



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

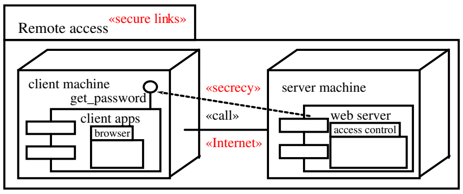
≪secure links≫

Ensures that physical layer meets security requirements on communication.

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

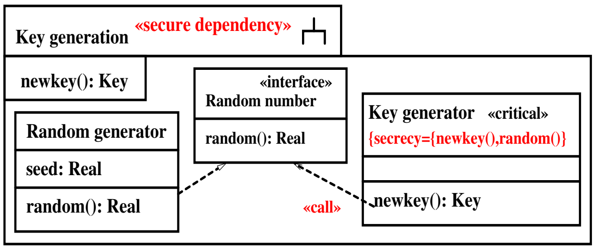
- if *s* = ≪secrecy≫: have read ∉ $Threats_A(t)$.
- if *s* = ≪integrity≫: have insert ∉ $Threats_A(t)$.

Example ≪secure links≫



Given default adversary type, constraint for stereotype ≪secure links≫ violated: According to the $Threats_{default}$(Internet) scenario, ≪Internet≫ link does not provide secrecy against default adversary.

Example ≪secure dependency≫



≪secure dependency≫ provided ?

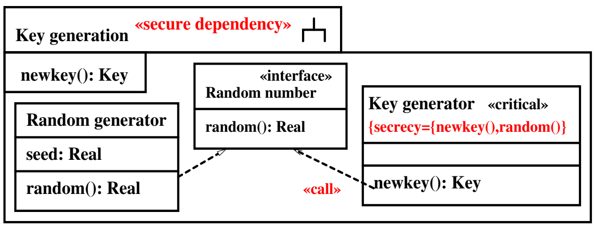
≪secure dependency≫

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}):

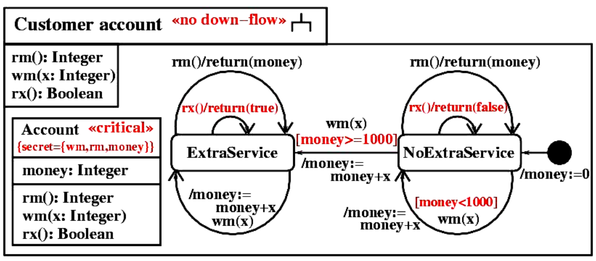
- Msg in *D* is {secrecy} in *C* if and only if also in *D*.
- If msg in *D* is {secrecy} in *C*, dependency stereotyped ≪secrecy≫.

Example ≪secure dependency≫



Violates ≪secure dependency≫: Random generator and ≪call≫ dependency do not give security level for random() to key generator.

Example ≪no down-flow≫



≪no down–flow≫ provided ?

≪no down–flow≫

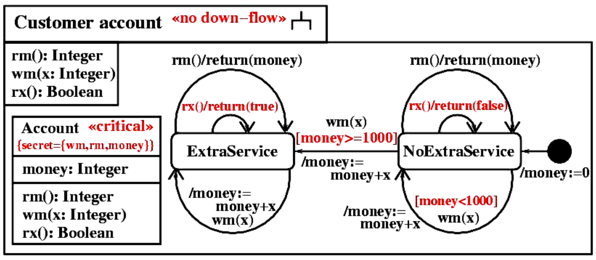
Enforce secure information flow.

Constraint:

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

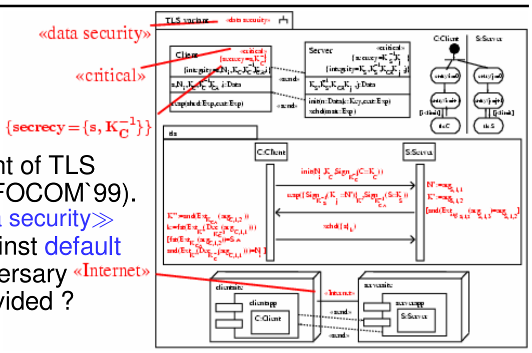
Formalize by referring to formal behavioural semantics.

Example ≪no down-flow≫



≪no down–flow≫ violated: partial information on input of secret wm() returned by non-secret rx().

Example ≪data security≫



Variant of TLS (INFOCOM`99). ≪data security≫ against default adversary provided ?

≪data security≫

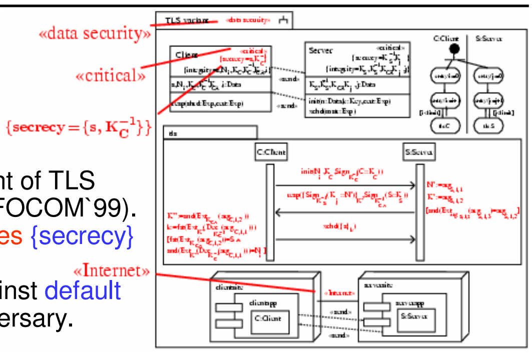
Security requirements of data marked ≪critical≫ enforced against threat scenario from deployment diagram.

Constraints:

Secrecy of {secrecy} data preserved.

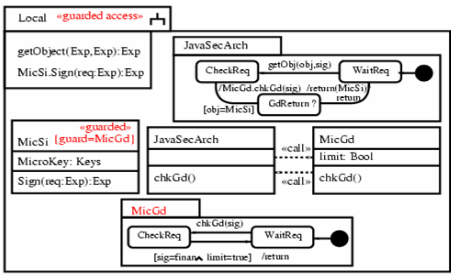
Integrity of {integrity} data preserved.

Example ≪data security≫



Variant of TLS (INFOCOM`99). Violates {secrecy} of s against default adversary.

Example ≪guarded access≫



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

≪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.

Does UMLsec meet requirements?

Security requirements: ≪secrecy≫,…

Threat scenarios: Use $Threats_{adv}(ster)$.

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.

Roadmap

Prologue
UML
UMLsec
Security Analysis

UMLsafe
Towards UML 2.0
Model-based Testing
Tools

Tool-support: Concepts

Meaning of diagrams stated informally in (OMG 2003).

Ambiguities problem for

- tool support
- establishing behavioral properties (safety, security)

Need precise semantics for used part of UML, especially to ensure security requirements.

Formal semantics for UML: How

Diagrams in context (using subsystems).

Model actions and internal activities explicitly.

Message exchange between objects or components (incl. event dispatching).

For UMLsec/safe: include adversary/failure model arising from threat scenario in deployment diagram.

Use Abstract State Machines (pseudo-code).

Tool-supported analysis

Choose drawing tool for UML specifications

Analyze specifications via XMI (XML Metadata Interchange)

skip compar.

Tool-supported analysis

Commercial modelling tools: so far mainly syntactic checks and code-generation.

Goal: more sophisticated analysis; connection to verification tools.

Several possibilities:

- General purpose language with integrated XML parser (Perl, …)
- Special purpose XML parsing language (XSLT, …)
- Data Binding (Castor; XMI: e.g. MDR)

Data-binding with MDR

MDR: MetaData Repository, Netbeans library (www.netbeans.org)

Extracts data from XMI file into Java Objects, following UML 1.4 meta-model.

Access data via methods.

Advantage: No need to worry about XML.

Framework for CSDUML tools: viki

Implements functionality
- MDR wrapper
- File handling
- Properties management
- Tool management

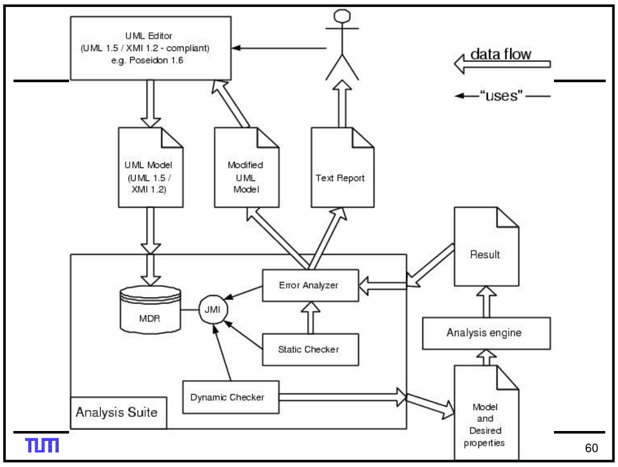
Exposes interfaces
- IVikiFramework
- IMdrWrapper
- IAppSettings

viki Tool

- Works in GUI and/or Text mode
- Implements interfaces
  - IVikiToolCommandLine
    - Text output only
  - IVikiToolGui
    - Output to JPanel + menu, buttons, etc
- Exposes set of commands
  - Automatically imported by the framework

Implementing tools

Exposes a set of commands.
Has its internal state (preserved between command calls).
Every single command is not interactive (read user input only at the beginning).
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 criticality 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.

Roadmap

Security Analysis

Specify protocol participants 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, simulate jointly with 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.

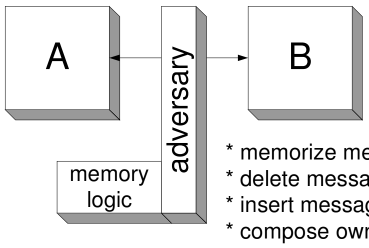
Expressions

Exp: term algebra generated by Var ∪ Keys ∪ Data and

- *_ :: _* (concatenation) and empty expression $\mathcal{E}$,
- *{ _ } _* (encryption)
- *Dec ( )* (decryption)
- *Sign ( )* (signing)
- *Ext_( )* (extracting from signature)
- *Hash(_)* (hashing)

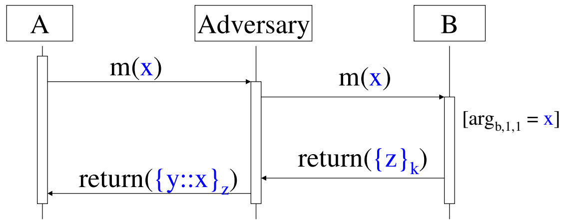
by factoring out the equations $Dec_{K^{-1}}(\{E\}_k) = E$ and $Ext_K(Sign_{K^{-1}}(E)) = E$ (for $K \in$ Keys).

Abstract adversary



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

Adversary: Simulation



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

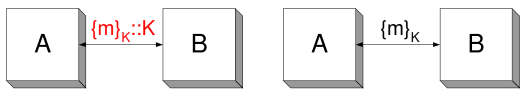
- $\forall e, k. Dec_{k^{-1}}(\{e\}_k) = e$

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 $M \in K_A^n$.

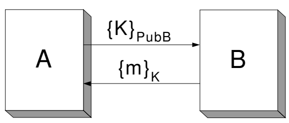
Example: secrecy



Against attacker who can read messages:

- Security of $\{m\}_K$::K not preserved
- Security of $\{m\}_K$ preserved

Example: secrecy



- Security of *m* is not preserved against an attacker who can delete and insert messages
- Security of *m* is preserved against an attacker who can listen, 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 Prolog, Setheo).

First-order logic: basic rules

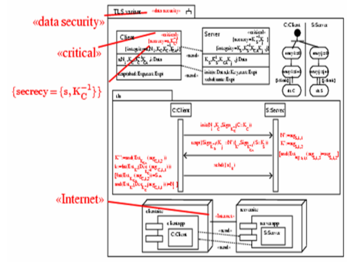
For initial adversary knowledge (K^0): Define *knows(E)* for any *E* initially known to the adversary (protocol-specific).

For evolving knowledge (K^n) define

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

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

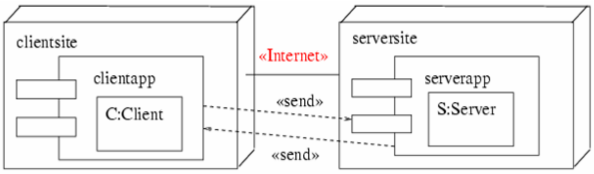
Example: Proposed Variant of TLS (SSL)



Apostolopoulos, Peris, Saha; IEEE Infocom 1999

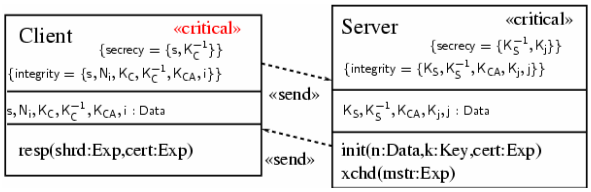
Goal: send secret *s* protected by session key K_j.

TLS Variant: Physical view



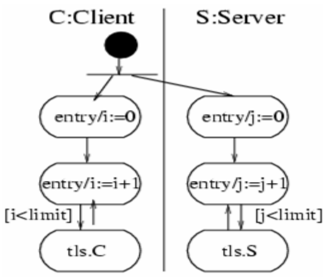
Deployment diagram.

TLS Variant: Structural view



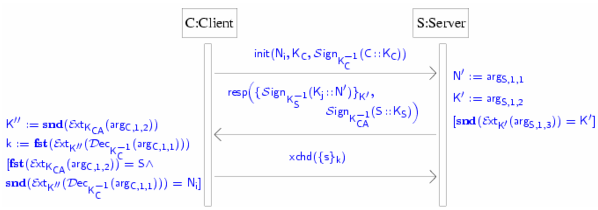
Class diagram

TLS Variant: Coordination view



Activity diagram.

TLS Variant: Interaction view



Sequence diagram.

Security protocols into 1st order logic

Sequence diagram: Each line of form

$[cond(arg_i,\ldots,arg_j)] \rightarrow exp(arg_i,\ldots,arg_j)$

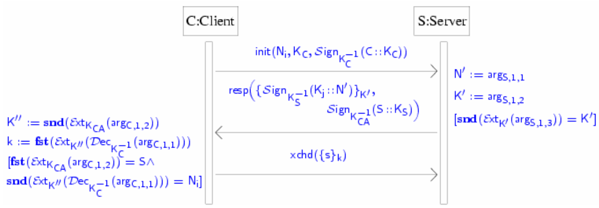
(where $arg_1,\ldots$ are all messages exchanged during one protocol run) is translated to:

$\forall exp_i.\ (knows(exp_1) \wedge \ldots \wedge knows(exp_n) \wedge cond(exp_1,\ldots,exp_n) \Rightarrow knows(exp(exp_1,\ldots,exp_n)))$

Adversary knowledge set approximated from above: abstract from senders, receivers, message order, …

➔ Find all attacks, may have false positives.

TLS Variant: Translation



knows(N_i) $\wedge$ …

$\wedge\ \forall$ exp… . (knows($exp_{S,1,3}$) $\wedge$ knows($exp_{S,1,2}$)

$\wedge$ snd($Ext_{exp_{S,1,2}}$($exp_{S,1,3}$)) = $exp_{S,1,2}$

➔ knows("arguments of resp method")

$\wedge$ …

Surprise

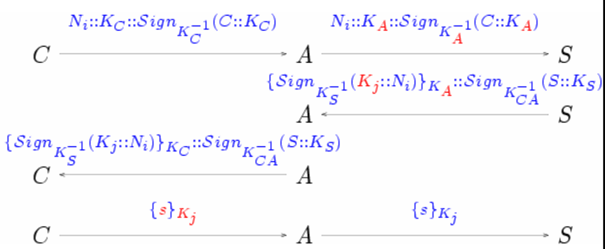
Add $knows(K_A) \wedge knows(K_A^{-1})$ (general previous knowledge of own keys).

Then can derive $knows(s)$ (!).

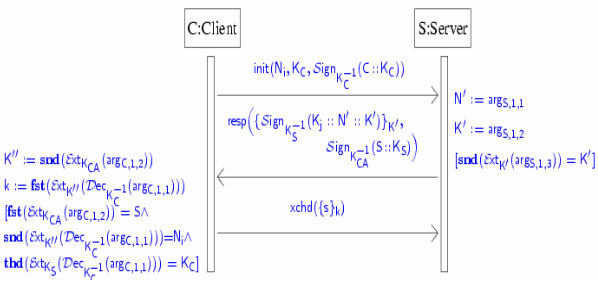
That is: $C||S$ does not preserve secrecy of s against adversaries whose initial knowledge contains K_A, K_A^{-1}.

Man-in-the-middle attack.

The attack



The fix



Security proof

Theorem. $C||S$ preserves the secrecy of s against adversaries with "reasonable" previous knowledge.

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.

Secure channel: approach

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

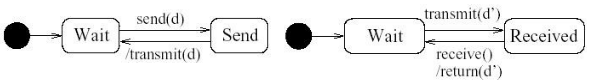
Give conditions under which it is secure to substitute channel abstractions by concrete protocols.

Secure channel abstraction

„Ideal“ of a secure channel:

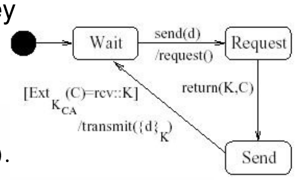
$$S = send(d).\overline{transmit}(s).S$$
$$R = transmit(d).\overline{receive}(d).R$$

Take $S \otimes R$ as secure channel abstraction. Trivially secure in absence of adversaries.



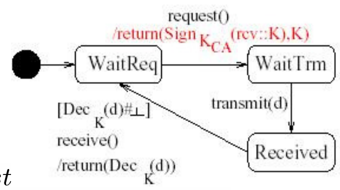
Concrete secure channel

Simple security protocol: encrypt under exchanged session key



$$S' := \text{iter}_i(send(d).\overline{request}.return(C).\ \text{if head}(\mathcal{E}xt_{K_R^{-1}}(\mathcal{D}ec_{K_S^{-1}}(C))) \in \text{Keys} \wedge \text{tail}(\mathcal{E}xt_{K_R}(\mathcal{D}ec_{K_S^{-1}}(C))) = j\ \text{then}\ \overline{transmit}(\{d :: i\}_K))$$

Concrete secure channel II



$$R' := \text{iter}_j(request\ \overline{return}(\{Sign_{K_R^{-1}}(K_j :: j)\}_{K_S}.\ transmit(E).receive.\ \text{if tail}(\mathcal{D}ec_{K_j}(E)) = j\ \text{then}\ \overline{return}(\text{head}(\mathcal{D}ec_{K_j}(E)))$$

Faithful representation ?

Is $S' \otimes R'$ equivalent to $S \otimes R$ in presence of adversary ? No: delay possible. But:

Theorem. $S' \otimes R'$ equivalent to $S \otimes R$ in presence of adversary with „reasonable“ previous knowledge.

Theorem. $S' \otimes R'$ preserves secrecy of d against such adversaries.

Demo

Java Security

Originally (JDK 1.0): sandbox.

Too simplistic and restrictive.

JDK 1.2/1.3: more fine-grained security control, signing, sealing, guarding objects, . . .)

BUT: complex, thus use is error-prone.

Java Security policies

Permission entries consist of:

- protection domains (i. e. URL's and keys)
- target resource (e.g. files on local machine)
- corresponding permissions (e.g. read, write, execute)

Signed and Sealed Objects

Need to protect integrity of objects used as authentication tokens or transported across JVMs.

A SignedObject contains an object and its signature.

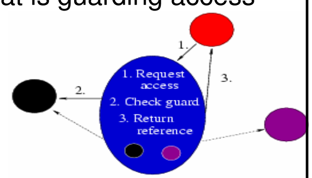
Similarly, need confidentiality.

A SealedObject is an encrypted object.

Guarded Objects

java.security.GuardedObject protects access to other objects.

- access controlled by getObject method
- invokes checkGuard method on the java.security.Guard that is guarding access
- If allowed: return reference. Otherwise: SecurityException



Problem: Complexity

- Granting of permission depends on execution context.
- Access control decisions may rely on multiple threads.
- A thread may involve several protection domains.
- Have method doPrivileged() overriding execution context.
- Guarded objects defer access control to run-time.
- Authentication in presence of adversaries can be subtle.
- Indirect granting of access with capabilities (keys).

→ Difficult to see which objects are granted permission.

⇒use UMLsec

Design Process

(1) Formulate access control requirements for sensitive objects.

(2) Give guard objects with appropriate access control checks.

(3) Check that guard objects protect objects sufficiently.

(4) Check that access control is consistent with functionality.

(5) Check mobile objects are sufficiently protected.

Reasoning

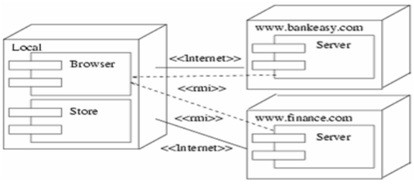
Theorem.

Suppose access to resource according to Guard object specifications granted only to objects signed with *K*.

Suppose all components keep secrecy of *K*.

Then only objects signed with *K* are granted access.

Example: Financial Application



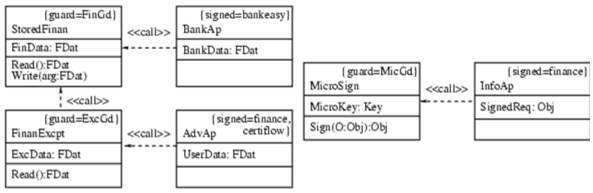
Internet bank, Bankeasy, and financial advisor, Finance, offer services to local user. Applets need certain Privileges (step1).

- Applets from and signed by bank read and write financial data between 1 pm and 2 pm.
- Applets from and signed by Finance use micropayment key five times a week.

Financial Application: Class diagram

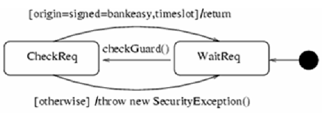
Sign and seal objects sent over Internet for Integrity and confidentiality.

GuardedObjects control access.

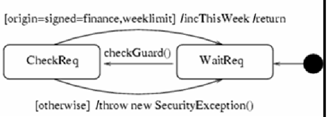


Financial Application: Guard objects (step 2)

timeslot true between 1pm and 2pm.



weeklimit true until access granted five times; inc ThisWeek increments counter.



Financial Application: Validation

Guard objects give sufficient protection (step 3).

Proposition. UML specification for guard objects only grants permissions implied by access permission requirements.

Access control consistent with functionality (step 4). Includes:

Proposition. Suppose applet in current execution context originates from and signed by Finance. Use of micropayment key requested (and less than five times before). Then permission granted.

Mobile objects sufficiently protected (step 5), since objects sent over Internet are signed and sealed.

CORBA access control

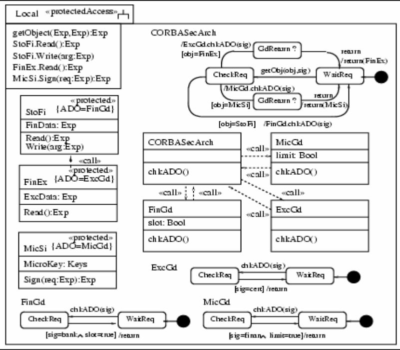
Object invocation access policy controls access of a client to a certain object via a certain method.

Realized by ORB and Security Service.

Use access decision functions to decide whether access permitted. Depends on

- called operation,
- privileges of the principals in whose account the client acts,
- control attributes of the target object.

Example: CORBA access control with UMLsec



Further Applications

- Analysis of multi-layer security protocol for web application of German bank
- Analysis of SAP access control configurations for German bank
- Biometric authentication protocol for German Telekom
- Automotive telematic application for German car manufacturer
- …

Rules of prudent security engineering

Saltzer, Schroeder (1975):

Design principles for security-critical systems.

Check how to enforce these with UMLsec.

Economy of mechanism

Keep the design as simple and small as possible.

Often systems made complicated to make them (look) secure.

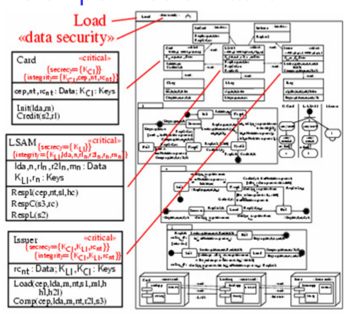
Method for reassurance may reduce this temptation.

Payoffs from formal evaluation may increase incentive for following the rule.

Fail-safe defaults

Base access decisions on permission rather than exclusion.

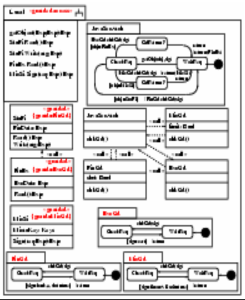
Example: secure log-keeping for audit control in Common Electronic Purse Specifications (CEPS).



Complete mediation

Every access to every object must be checked for authority.

E.g. in Java: use guarded objects. Use UMLsec to ensure proper use of guards.
More feasibly, mediation wrt. a set of sensitive objects.



Open design

The design should not be secret.

Method of reassurance may help to develop systems whose security does not rely on the secrecy of its design.

Separation of privilege

A protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key.

Example: signature of two or more principals required for privilege. Formulate requirements with activity diagrams.

Verify behavioural specifications wrt. them.

Least privilege

Every program and every user of the system should operate using the least set of privileges necessary to complete the job.

Least privilege: every proper diminishing of privileges gives system not satisfying functionality requirements.

Can make precise and check this.

Least common mechanism

Minimize the amount of mechanism common to more than one user and depended on by all users.

Object-orientation:

- data encapsulation
- data sharing well-defined (keep at necessary minimum).

Psychological acceptability

Human interface must be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly.

Wrt. development process: ease of use in development of secure systems.

User side: e.g. performance evaluation (acceptability of performance impact of security).

Discussion

No absolute rules, but warnings.

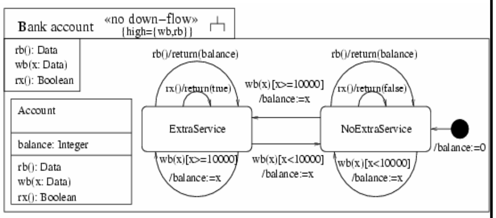
Violation of rules symptom of potential trouble; review design to be sure that trouble accounted for or unimportant.

Design principles reduce number and seriousness of flaws.

Security Patterns

Security patterns: use UML to encapsulate knowledge of prudent security engineering.

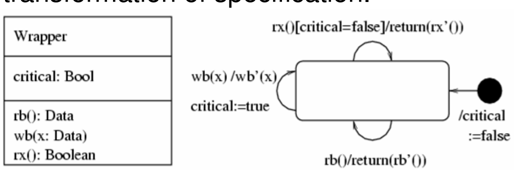
Example:



Does not preserve security of account balance.

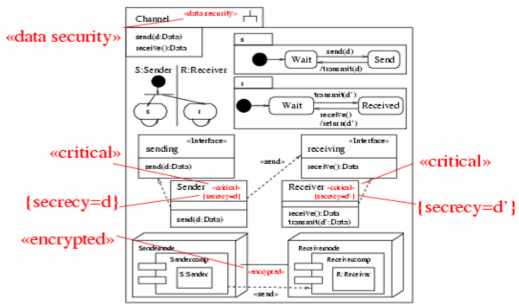
Solution: Wrapper Pattern

Technically, pattern application is transformation of specification.



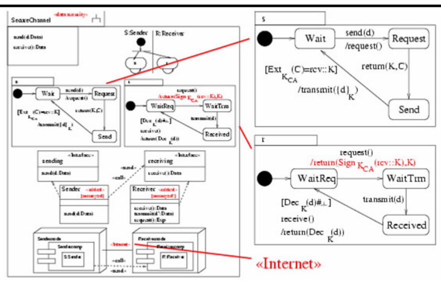
Use wrapper pattern to ensure that no low read after high write.
Can check this is secure (once and for all).

Secure channel pattern: problem



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

Secure channel pattern: (simple) solution



Exchange certificate and send encrypted data over Internet.

Layered Security Protocols

- Protocol on higher layer uses services of protocol on lower layer.
- Big question: security properties additive ?
- Desirable: secure channel abstraction.

client authenticity

confidentiality, integrity, server authenticity

=

confidentiality, … + client authenticity ?

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.

SSL Protocol

Provided security services:

- Secure data transmission.
  - Integrity of data.
  - Confidentiality of data.
- Authentication of the server against the client.

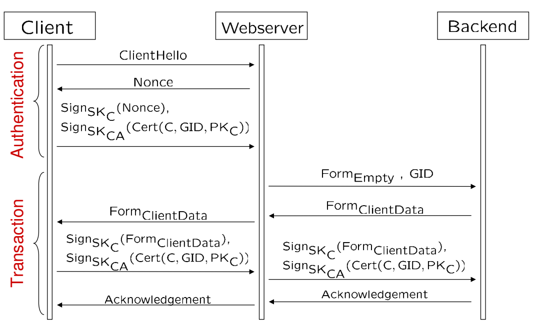
Verify using model-checker.

Authentication protocol

Provided security service:

- Authentication of the client against the bank's server.
- Was not provided by SSL because the underlying software did not support this feature.

Authentication protocol



Layered Security Analysis

- 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.

Verification of the Auth. protocol 1

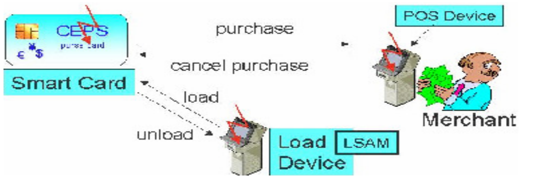
- Authentication:
  – It's not possible for the adversary to authenticate under a wrong identity against the web server (verification: 2 hours 40 minutes).
- Transaction:
  – It's not possible for the adversary to get the confidential client's data (verification: 2 hours 50 minutes).

Insight

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

Generalize to classes of protocols and security requirements.

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).

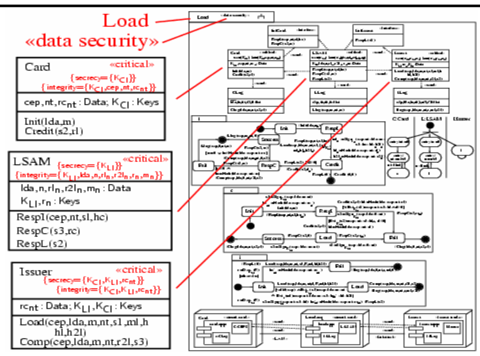
Load value onto card using cash at load device.

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

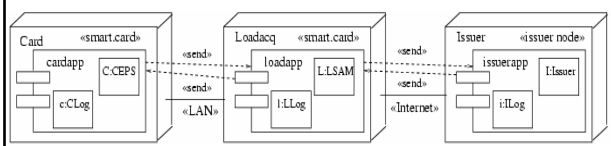
Card account balance adjusted; transaction data logged and sent to issuer for financial settlement.

Uses symmetric cryptography.

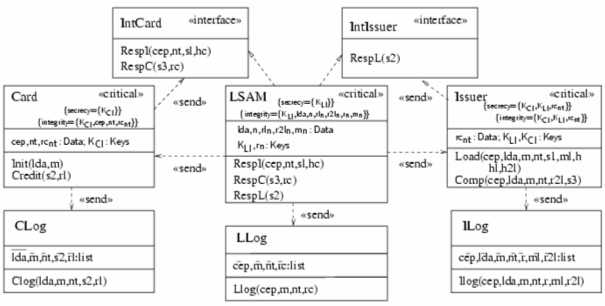
Load protocol



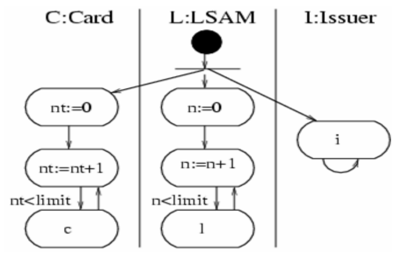
Load protocol: Physical view



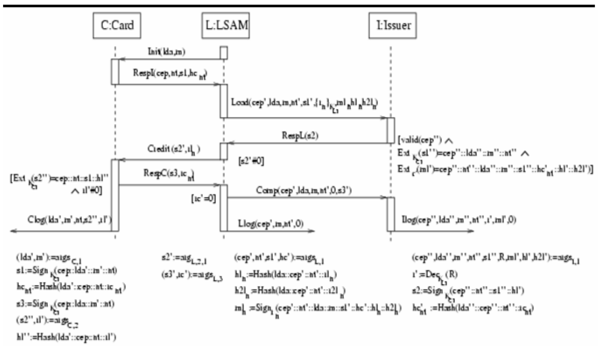
Load protocol: Structural view



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 $ml_n = Sign_{r_n}(cep::nt::lda::m_n::s1::hc_{nt}::hl_n::h2l_n)$ 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

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

Modification: use asymmetric key in ml_n, include signature certifying hc_{nt}.

Verify this version wrt. above conditions.

Further applications

- Analysis of SAP access control configurations
- Biometric authentication system of German telecommunication company
- Automobile emergency application of German car company
- Electronic signature architecture of German insurance company

Roadmap

Prologue
UML
UMLsec
Security Analysis

UMLsafe
Towards UML 2.0
Model-based Testing
Tools

Safety: Some Terminology

- Reliability: probability of a failure-free functioning of a software component for a specified period in a specified environment
- Safety: software execution without contributing to hazards
- Failures: perceived deviation of output values from expected values
- Faults: possible cause of failures in hardware, code or other artefacts

Safety

Safety-critical systems: five failure condition categories: catastrophic, hazardous, major, minor, no effect.

Corresponding safety levels A - E (DO-178B standards in avionics).

Safety goals: via the maximum allowed failure rate. For high degree of safety, testing not sufficient (1 failure per 100,000 years).

Fault-tolerance

Redundancy model determines which level of redundancy provided.

Goal: no hazards in presence of single-point failures.

In the following treatment:

- focus on safety-critical systems which in particular have to be reliable
- focus on fault-tolerance aspects of safety

Embedded Systems

In particular, embedded software increasingly used in safety-critical systems (flexibility):

- Automotive
- Avionics
- Aeronautics
- Robotics, Telemedicine
- …

Our treatment of safety-critical systems in particular applies to embedded systems.

Faults vs. Failures

Faults: existing deficiencies of a given system (e.g. hardware faults).

Failures: resulting deficient behaviour of the system.

For example, a faulty communication line may result in a communication failure.

Failures may be considered relative to system requirements (e.g., in real-time system, inacceptable communication delay can be considered a „failure").

From UMLsec to UMLsafe

Safety = „Security against stupid adversaries"

Security = „Safety for paranoids"

Adversaries in security correspond to failures in safety.

Replace adversary model in UMLsec by failure model to get UMLsafe.

Failure semantics modelling

For redundancy model *R*, stereotype *s*∈{≪crash/performance≫, ≪value≫}, have set $\text{Failures}_R(s) \subseteq \{\text{delay}(t), \text{loss}(p), \text{corrupt}(q)\}$, with interpretation:

- *t*: expected maximum time delay,
- *p*: probability that value not delivered within *t*,
- *q*: probability that value delivered in time corrupted

(in each case incorporating redundancy).
Or use ≪risk≫ stereotype with {failure} tag.

Example

Suppose redundancy model *R* uses controller with redundancy *3* and the fastest result. Then could take:

- delay(*t*): *t* delay of fastest controller,
- loss(*p*): *p* probability that fastest result not delivered within *t*,
- corrupt(*q*): *q* probability that fastest result is corrupted

(each wrt. the given failure semantics).

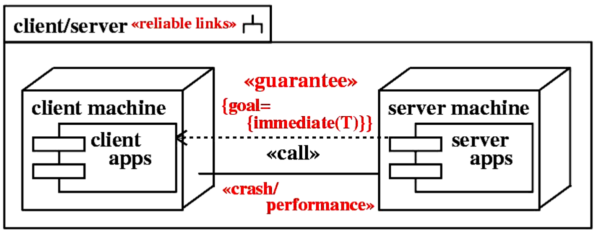
≪guarantee≫

Describe guarantees required from communication dependencies resp. system components.

Tags: {goal} with value subset of {immediate(t), eventual(p), correct(q)}, where

- t: expected maximum time delay,
- p: probability that value is delivered within t,
- q: probability that value delivered in time not corrupted.

Example ≪reliable links≫



Given redundancy model none, when is ≪reliable links≫ fulfilled ?

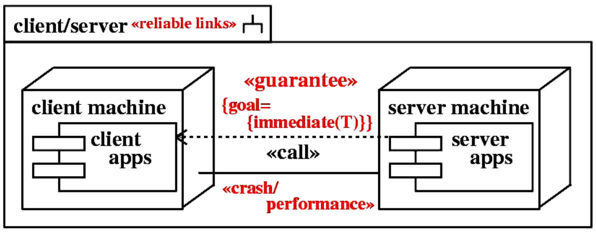
≪reliable links≫

Physical layer should meet reliability requirements on communication given redundancy model R.

Constraint: For dependency d stereotyped ≪guarantee≫ and each corresponding communication link l with stereotype s:

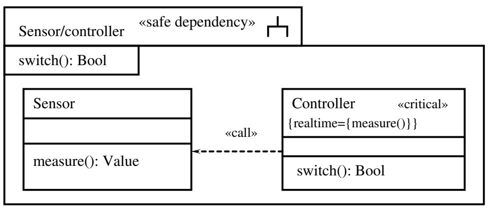
- if {goal} has immediate(t) as value then delay(t') $\in$ Failures$_R(s)$ implies $t' \le t$,
- if {goal} has eventual(p) as value then loss(p') $\in$ Failures$_R(s)$ implies $p' \le 1-p$, and
- if {goal} has correct(q) as value then corruption(q') $\in$ Failures$_R(s)$ implies $q' \le 1-q$.

Example ≪reliable links≫



Given redundancy model none, ≪reliable links≫ fulfilled iff T ≥ expected delay according to Failures$_{none}$(≪crash/performance≫).

Example ≪reliable dependency≫



Assuming immediate(t) $\in$ goals(realtime), ≪reliable dependency≫ provided ?

≪reliable dependency≫

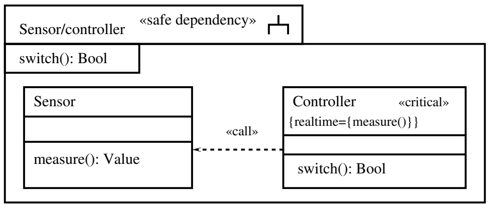
Communication dependencies should respect safety requirements on ≪critical≫ data.

For each safety level {l} for ≪critical≫ data, have goals(l)$\subseteq${immediate(t), eventual(p), correct(q)}.

Constraint: for each dependency d from C to D stereotyped ≪guarantee≫:

- Goals on data in D same as those in C.
- Goals on data in C that also appears in D met by guarantees of d.

Example ≪reliable dependency≫



Assuming immediate(t) ∈ goals(realtime), violates ≪reliable dependency≫, since Sensor and dependency do not provide realtime goal immediate(t) for measure() required by Controller.

Execution semantics

Behavioral interpretation of a UML subsystem:

(1) Takes input events.

(2) Events distributed from input and link queues between subcomponents to intended recipients where they are processed.

(3) Output distributed to link or output queues.

(4) Failure model applied as follows.

Failure models

lq^{l}_{n}: messages on link l delayed further n time units.

p^{h}_{n}: probability of failure at n^{th} iteration in history h.

For link l stereotyped s where loss(p)∈ Failures$_R$(s),

- history may give lq^{l}_{0}:=∅; then append p to $(p^{h}_{n})_{n \in \mathcal{N}}$,
- or no change, then append 1-p.

For link l stereotyped s where corruption(q)∈ Failures$_R$(s),

- history may give lq^{l}_{0}:={■}; then append q,
- or no change; append 1-q.

For link l stereotyped s with delay(t)∈ Failures$_R$(s), and $lq^{l}_{0} \neq \emptyset$, history may give lq^{l}_{n}:=lq^{l}_{0} for $n \leq t$; append $1/t$.

Then for each n, lq^{l}_{n}:=lq^{l}_{n+1}.

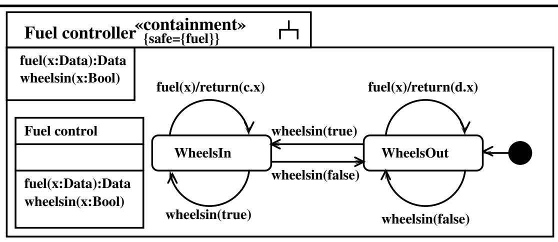
≪safe behaviour≫

Ensures that system behavior in presence of failure model provides required safety {goals}:

For any execution trace h, any transmission of a value along a communication dependency stereotyped ≪guarantee≫, the following constraints should hold, given the safety goal:

- eventual(p): With probability at least p, …
- immediate(t): … every value is delivered after at most t time steps.
- correct(q): Probability that a delivered value is corrupted during transmission is at most 1-q.

Example ≪containment≫



Containment satisfied ?

≪containment≫

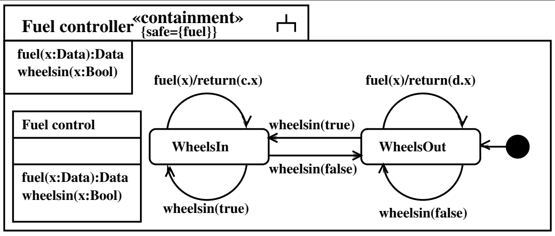
Prevent indirect corruption of data.

Constraint:

Value of any data element d may only be influenced by data whose requirements attached to ≪critical≫ imply those of d.

Make precise by referring to execution semantics (view of history associated with safety level).

Example ≪containment≫



Violates containment because a {safe} value depends on un{safe} value.

Can check this mechanically.

Other checks

Have other consistency checks such as

- Is the software‘s response to out-of-range values specified for every input ?
- If input arrives when it shouldn't, is a response specified ?

…and other safety checks from the literature.

IEC 61508 (1)

IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems

Strategy: derive safety requirements from a hazard and risk analysis and to design the system to meet those safety requirements, taking all possible causes of failure into account.

IEC 61508 (2)

- Concept: An understanding of the system and its environment is developed.
- Overall scope definition: The boundaries of the system and its environment are determined, and the scope of the hazard and risk analysis is specified.
- Hazard and risk analysis: Hazards and hazardous events of the system, the event sequences leading to the hazardous events, and the risks associated with the hazardous events are determined.
- Overall safety requirements: The specification for the overall safety requirements is developed in order to achieve the required functional safety.
- Safety requirements allocation: The safety functions contained in the overall safety requirements specification are allocated to the safety-related system, and a safety integrity level is allocated to each safety function.

IEC 61508 (3)

- Overall operation and maintenance planning: A plan is developed for operating and maintaining the system, and the required functional safety is ensured to be maintained during operation and maintenance.
- Overall safety validation planning: A plan for the overall safety validation of the system is developed.
- Overall installation and commissioning planning: Plans, ensuring that the required functional safety is achieved, are developed for the installation and commissioning of the system.
- Safety-related systems, E/E/PES: The E/E/PES safety-related system is created conforming to the safety requirements specification.
- Safety-related systems, other technology: Other technology safety-related systems are created to meet the requirements specified for such systems (outside scope of the standard).

IEC 61508 (4)

- External risk reduction facilities: External risk reduction facilities are created to meet the requirements specified for such facilities (outside scope of the standard).
- Overall installation and commissioning: The E/E/PES safety-related system is installed and commissioned.
- Overall safety validation: The E/E/PES safety-related system is validated to meet the overall safety requirements specification.
- Overall operation, maintenance and repair: The system is operated, maintained and repaired in order to ensure that the required functional safety is maintained.
- Overall modification and retrofit: The functional safety of the system is ensured to be appropriate both during and after modification and retrofit.

IEC 61508 (5)

- Decommissioning or disposal: The functional safety of the system is ensured to be appropriate during and after decommissioning or disposing of the system.

Roadmap

Some new concepts in UML 2.0

UML extended with concepts from UML RT (Selic, Rumbaugh 1998).

Focus on software architecture.

New: capsules, ports, connectors.

Capsules, ports, connectors

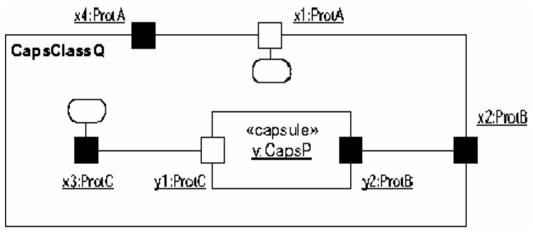
Capsules: architectural objects interacting through signal-based boundary objects (ports).

Port: object implementing interface of capsule. Associated with a protocol defining flow of information.

Connector: abstract signal-based communication channels between ports.

Functionality of capsule realized by associated state machine.

Example



From Selic, Rumbaugh 1998.

Roadmap

Tool-support: Test-generation

Two complementary strategies:

- Conformance testing
- Testing for criticality requirements

Conformance testing

Classical approach in model-based test-generation (much literature).

Can be superfluous when using code-generation [except to check your code-generator, but probably once and for all]

Works independently of criticality requirements.

Conformance testing: Problems

- Complete test-coverage usually infeasible. Need to somehow select test-cases.
- Can only test code against what is contained in the behavioral model. Usually, model is more abstract than code. So may have „blind spots“ in the code.

For both reasons, may miss critical test-cases.

Criticality testing

Shortcoming of classical model-based test-generation (conformance testing) motivates „criticality testing“ (e.g., papers by Jürjens, Wimmel at PSI’01, ASE’01, ICFEM’02).

Goal: model-based test-generation adequate for (security-, safety-) critical systems.

Criticality testing: Strategies

Strategies:

- Ensure test-case selection from behavioral models does not miss critical cases: Select according to information on criticality („internal“ criticality testing).
- Test code against possible environment interaction generated from external parts of the model (e.g. deployment diagram with information on physical environment).

Internal Criticality Testing

Need behavioral semantics of used specification language (precise enough to be understood by a tool).

Here: semantics for simplified fragment of UML in „pseudo-code“ (ASMs).

Select test-cases according to criticality annotations in the class diagrams.

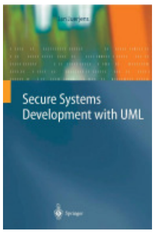
Test-cases: critical selections of intended behavior of the system.

External Criticality Testing

Generate test-sequences representing the environment behaviour from the criticality information in the deployment diagrams.

Some resources

Book: Jan Jürjens, Secure Systems Development with UML, Springer-Verlag, 2004
Tutorials: Sept.: SAFECOMP (Potsdam), ASE (Linz).
Summer School Lecture: FOSAD (Bertinoro, Italy, Sept.)
Workshop: CSDUML@UML04



More information (papers, slides, tool etc.): http://www4.in.tum.de/~juerjens/csdumltut (user Participant, password Iwasthere)

Finally

We are always interested in industrial challenges for our tools, methods, and ideas to solve practical problems.
More info: http://www4.in.tum.de/~secse

Contact me here or via Internet.

Thanks for your attention !

BREAK !

Note:

We are always interested in industrial challenges for our tools, methods, and ideas to solve practical problems.
More info: http://www4.in.tum.de/~secse

Contact me here or via Internet.

Roadmap

UML Drawing Tools

Wide range of existing tools.

Consider some, selected under following criteria (Shabalin 2002):

- Support for all (UMLsec/safe-) relevant diagram types.
- Support for custom UML extensions.
- Availability (test version, etc).
- Prevalence on the market.

Selected Tools

- Rational Rose. Developed by major participant in development of UML; market leader.
- Visio for Enterprise Architect. Part of Microsoft Developer Studio .NET.
- Together. Often referenced as one of the best UML tools.
- ArgoUML. Open Source Project, therefore interesting for academic community. Commercial variant Poseidon.

Comparison

Evaluated features:

Support for custom UML extensions.

- Model export; standards support; tool interoperability.
- Ability to enforce model rules, detect errors, etc.
- User interface quality.
- Possibility to use the tool for free for academic institutions.

Rational Rose (Rational Software Corporation)

One of the oldest on the market.

\+ Free academic license.

\+ Widely used in the industry.

\+ Export to different XMI versions.

\- Insufficient support for UML extensions (custom stereotypes yes; tags and constraints no).

\- Limited support for checking syntactic correctness.

\- Very inconvenient user interface. Bad layout control.

\- Lack of compatibility between versions and with other Rational products for UML modelling.

Together from TogetherSoft

Widely used in the development community. Very good round-trip engineering between the UML model and the code.

\+ Free academic license.

\+ Written in Java, therefore platform-independent.

\+ Nice, intuitive user interface.

\+ Export to different XMI versions; recommendations which for which tool.

\- Insufficient support for UML extensions (custom stereotypes yes; tags and constraints no).

Visio from Microsoft Corporation

Has recently been extended with UML editing support

\+ Good user interface

\+ Full support for UML extensions

\+ Very good correspondence to UML standard. Checks dynamically for syntactic correctness; suggestions for fixing errors

\- No free academic license

\- Proprietary, undocumented file format; very limited XMI export

\- No round-trip engineering support. No way back after code generation

Choice: ArgoUML / Poseidon

ArgoUML: Open Source Project. Commercial extension Poseidon (Gentleware), same internal data format

+ Open Source
+ Written in Java, therefore platform-independent
+ XMI default model format
+ Poseidon: solid mature product with good UML specification support

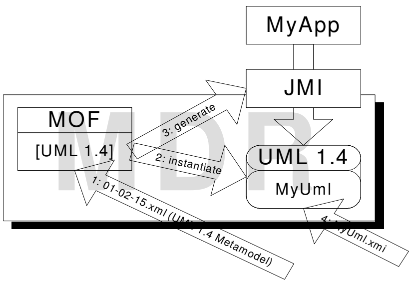
MDR Standards

- MOF (Meta Object Facility)
  Abstract format for describing metamodels
- XMI (XML Metadata Interchange)
  Defines XML format for a MOF metamodel
- JMI (Java Metadata Interface)
  Defines mapping from MOF to Java

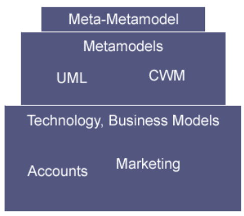
MDR Services

- Load and Store a MOF Metamodel (XMI format)
- Instantiate and Populate a Metamodel (XMI format)
- Generate a JMI (Java Metadata Interface) Definition for a Metamodel
- Access a Metamodel Instance

UML Processing



MOF Architecture



- Meta-Metamodel (M3)
  - defined by OMG
- Metamodels (M2)
  - user-defined
  - e.g. UML 1.5, MOF, CWM
  - can be created with uml2mof
- Business Model (M1)
  - instances of Metamodels
  - e.g. UML class diagram
- Information (M0)
  - instance of model
  - e.g. implementation of UML modelled classes in Java

MOF (Meta Object Facility)

skip details

OMG Standard for Metamodeling

Meta-Metamodel	MetaClass, MetaAssociation - MOF Model
Metamodel	Class, Attribute, Dependency - UML (as language), CWM
Model	Person, House, City - UML model
Data	(Bob Marley, 1975) (Bonn) - Running Program

JMI: MOF Interfaces

- IDL mapping for manipulating Metadata
  - API for manipulating information contained in an instance of a Metamodel
  - MOF is MOF compliant!
  - Metamodels can be manipulated by this IDL mapping
  - JMI is MOF to Java mapping
  - JMI has same functionality
- Reflective APIs
  - manipulation of complex information
  - can be used without generating the IDL mapping
  - MDR has implemented these interfaces

MDR Repository: Loading Models

- Metamodel is instance of another Metamodel
- Loading Model = Loading Metamodel
- Needed Objects:
  - MDRepository
  - MofPackage
  - XMISaxReaderImpl

- Java Code-Snippet:

```
MDRepository rep;
UmlPackage uml;
// Objekte erzeugen:
rep =

    MDRManager.getDefault().getDefaultRepository()
    ;
reader =
  (XMISaxReaderImpl)Lookup.getDefault().lookup(
          XmiReader.class);

// loading extent:
uml = (UmlPackage)rep.getExtent(„name");

// creating Extent:
uml = (UmlPackage)rep.createExtent(„name");

// loading XMI:
reader.read(„url", MofPackage);,
```


MDR Repository: Reading Data

- Requires open Repository and Package
- Requires JMI Interfaces
- Example: Loading UML Class:

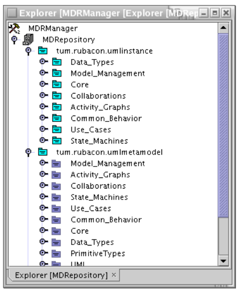
```
Iterator it =
  uml.getCore().getUmlClass(
  ).refAllOfClass().iterator
  ();

while (it.hasNext()) {
   UmlClass uc =
  (umlClass)it.next();

   // .. do anything with
  UmlClass ..
}
```


Netbeans MDR Explorer

- Part of Netbeans IDE
- Browse Repositories
- Create Instances
- Load XMI Data
- Generate JMI Interfaces
- Shows
  - Extents
  - Metamodels
  - Instances



Roadmap

Prologue
UML
UMLsec
Security Analysis

UMLsafe
Towards UML 2.0
Model-based Testing
Tools

Security Protocols

System distributed over untrusted networks.

„Adversary“ intercepts, modifies, deletes, inserts messages.

Cryptography provides security.

Cryptographic Protocol: Exchange of messages for distributing session keys, authenticating principals etc. using cryptographic algorithms

Security Protocols: Problems

Many protocols have vulnerabilities or subtleties for various reasons

- weak cryptography
- core message exchange
- interfaces, prologues, epilogues
- deployment
- implementation bugs

Using UML

Goal: transport results from formal methods to security practice

Enable developers (not trained in formal methods) to

- check correctness of hand-made security protocols
- deploy protocols correctly in system context
- allow to analyze larger system parts beyond protocols

Formal semantics for UML: Why

Meaning of diagrams stated imprecisely in (OMG 2001).

Ambiguities problem for

- tool support
- establishing behavioral properties (e.g. security)

Need precise semantics for used part of UML, especially to ensure security requirements.

Formal semantics for UML: How

Diagrams in context (using subsystems).

Model actions and internal activities explicitly.

Message exchange between objects or components (incl. event dispatching).

For UMLsec: include adversary arising from threat scenario in deployment diagram.

Use Abstract State Machines (pseudo-code).

Security Analysis

Specify protocol participants 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, simulate jointly with 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.

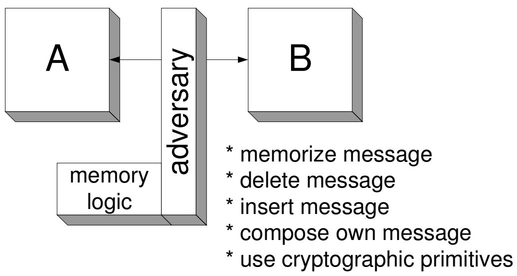
Expressions

Exp: term algebra generated by Var ∪ Keys ∪ Data and

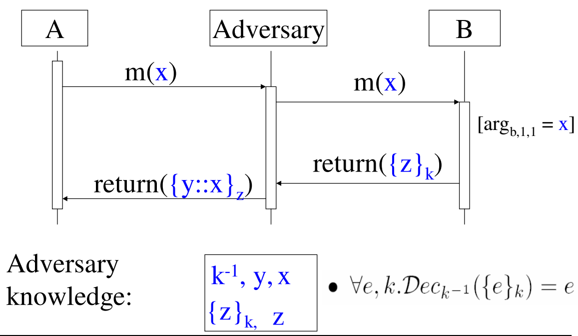
- _ :: _ (concatenation) and empty expression $\mathcal{E}$,
- { _ } _ (encryption)
- Dec ( ) (decryption)
- Sign ( ) (signing)
- Ext_( ) (extracting from signature)
- Hash(_) (hashing)

by factoring out the equations $Dec_{K^{-1}}(\{E\}_k) = E$ and $Ext_K(Sign_{K^{-1}}(E)) = E$ (for $K \in$ Keys).

Abstract adversary



Adversary: Simulation



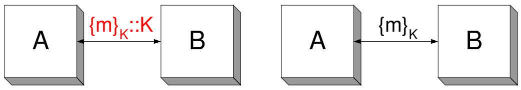
Abstract adversary

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 $M \in K_A^n$.

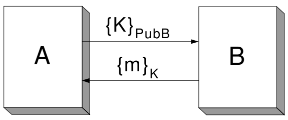
Example: secrecy



Against attacker who can read messages:

- Security of $\{m\}_K::K$ not preserved
- Security of $\{m\}_K$ preserved

Example: secrecy



- Security of *m* is not preserved against an attacker who can delete and insert messages
- Security of *m* is preserved against an attacker who can listen, 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 Prolog, Setheo).

First-order logic: basic rules

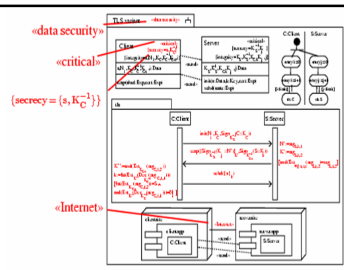
For initial adversary knowledge (K^0): Define *knows(E)* for any *E* initially known to the adversary (protocol-specific).

For evolving knowledge (K^n) define

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

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

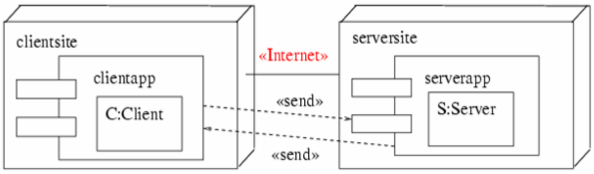
Example: Proposed Variant of TLS (SSL)



Apostolopoulos, Peris, Saha; IEEE Infocom 1999

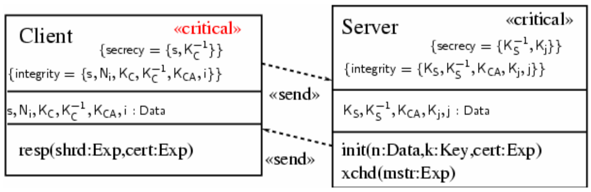
Goal: send secret *s* protected by session key K_j.

TLS Variant: Physical view



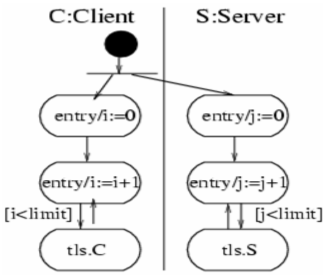
Deployment diagram.

TLS Variant: Structural view



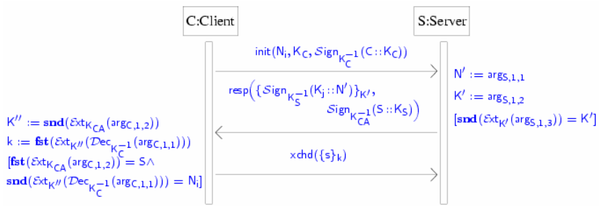
Class diagram

TLS Variant: Coordination view



Activity diagram.

TLS Variant: Interaction view



Sequence diagram.

Security protocols into 1st order logic

Sequence diagram: Each line of form

[cond(arg$_i$,…,arg$_j$)] → exp(arg$_i$,…,arg$_j$)

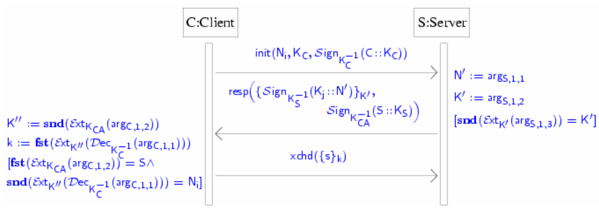
(where *arg$_1$,…* are all messages exchanged during one protocol run) is translated to:

∀ exp$_i$. (knows(exp$_1$) ∧ … ∧ knows(exp$_n$) ∧ cond(exp$_1$,…,exp$_n$) ⇒ knows(exp(exp$_1$,…,exp$_n$)))

Adversary knowledge set approximated from above: abstract from senders, receivers, message order, …

➔ Find all attacks, may have false positives.

TLS Variant: Translation



knows(N_i) ∧ …

∧ ∀ exp… . (knows($exp_{S,1,3}$) ∧ knows($exp_{S,1,2}$)

∧ snd($Ext_{exp_{S,1,2}}$($exp_{S,1,3}$)) = $exp_{S,1,2}$

➔ knows("arguments of resp method")

∧ …

Surprise

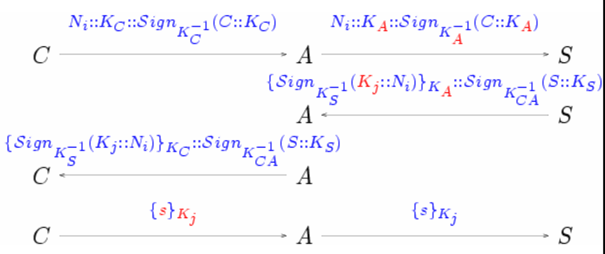
Add *knows*(K_A) ∧ *knows*(K_A^{-1}) (general previous knowledge of own keys).

Then can derive *knows*(*s*) (!).

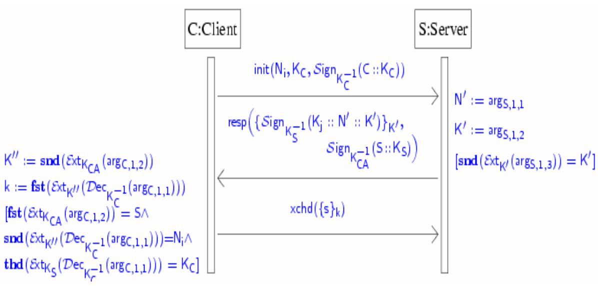
That is: $C||S$ does not preserve secrecy of *s* against adversaries whose initial knowledge contains K_A, K_A^{-1}.

Man-in-the-middle attack.

The attack



The fix



Security proof

Theorem. $C||S$ preserves the secrecy of *s* against adversaries whose initial knowledge $\mathcal{K}$ satisfies the following.

$$(\{C.s_I, K_C^{-1}, K_S^{-1}\} \cup \{S.k_j : j \geq J\}$$
$$\cup \{\{Sign_{K_S^{-1}}(X :: C.N_I :: K_C)\}_{K_C} : X \in \mathbf{Keys}\})$$
$$\cap \mathcal{K} = \emptyset$$

$$Sign_{K_C^{-1}}(C :: X) \in \mathcal{K} \Rightarrow X = K_C$$

$$Sign_{K_{CA}^{-1}}(S :: X) \in \mathcal{K} \Rightarrow X = K_S$$

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.

Secure channel: approach

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

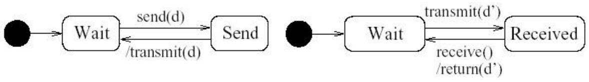
Give conditions under which it is secure to substitute channel abstractions by concrete protocols.

Secure channel abstraction

„Ideal" of a secure channel:

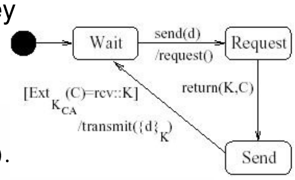
$$S = send(d).\overline{transmit}(s).S$$
$$R = transmit(d).\overline{receive}(d).R$$

Take $S \otimes^{\mathfrak{I}} R$ for $\mathfrak{I}$:={send,receive} as secure channel abstraction. Trivially secure in absence of adversaries.



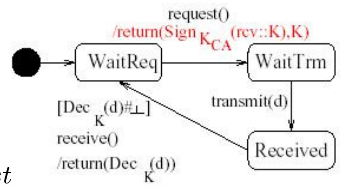
Concrete secure channel

Simple security protocol: encrypt under exchanged session key



$$S' := \text{iter}_i(send(d).$$
$$\overline{request}.return(C).$$
$$\text{if head}(\mathcal{E}xt_{K_R^{-1}}(\mathcal{D}ec_{K_S^{-1}}(C))) \in$$
$$\text{Keys} \wedge \text{tail}(\mathcal{E}xt_{K_R}(\mathcal{D}ec_{K_S^{-1}}(C))) = j$$
$$\text{then } \overline{transmit}(\{d :: i\}_K)$$

Concrete secure channel II



$$R' := \text{iter}_j(request$$
$$\overline{return}(\{Sign_{K_R^{-1}}(K_j :: j)\}_{K_S}.$$
$$transmit(E).receive.$$
$$\text{if tail}(\mathcal{D}ec_{K_j}(E)) = j$$
$$\text{then } \overline{return}(\text{head}(\mathcal{D}ec_{K_j}(E))))$$

Bisimulation

A binary relation R on processes is a bisimulation iff $(P\ R\ Q)$ implies that for all actions α,

- if $P \rightarrow^{\alpha} P'$ then exists $Q \rightarrow^{\alpha} Q'$ with $P'\ R\ Q'$ and
- if $Q \rightarrow^{\alpha} Q'$ then exists $P \rightarrow^{\alpha} P'$ with $P'\ R\ Q'$.

P, Q are bisimilar if there exists a bisimulation R with $P\ R\ Q$.

Faithful representation ?

Is $(R'||S')\otimes^{\mathfrak{I}}\mathcal{A}$ bisimilar to $S\otimes^{\mathfrak{I}}R$?
No: delay possible. But:

Theorem. Suppose $\mathcal{A}$ does not contain the messages *send, receive* nor any value in $\{K(S)^{-1},K(R)^{-1}\} \cup \{K_n,\{x::n\}_{K_n}: x \in Exp \wedge n \in \mathbb{N}\}$ nor $Sign_{K(R)}^{-1}(K'::n)$ unless $K'=K_n$. Then $(R'||S')\otimes^{\mathfrak{I}}\mathcal{A}$ is bisimilar to $(S\otimes^{\mathfrak{I}}R)\otimes\mathcal{A}_b$.

Theorem. $(R'||S')$ preserves secrecy of d against such $\mathcal{A}$.