IS 2620

UMLSec

Lecture 10
March 19, 2013

Objective

- Overview of UMLSec
  - How UML has been extended with security construct
  - Some security constructs in UMLSec
  - Validation of design
  - Acknowledgement: Courtesy of Jan Jurgens

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/flaws
- Correctness in conflict with cost.
- Thorough methods of system design not used if too expensive.

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*” (Lampson, Needham).

Previous approaches

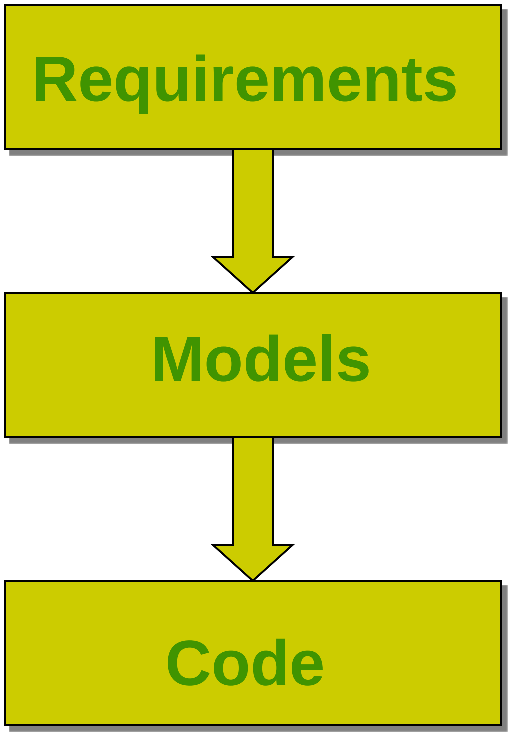
- “Penetrate-and-patch”: unsatisfactory.
  - insecure
    - damage until discovered
  - disruptive
    - distributing patches costs money, destroys confidence, annoys customers
- Traditional formal methods: expensive.
  - training people
  - constructing formal specifications.

Holistic view on Security

- Saltzer, Schroeder 1975:
  - “An expansive view of the problem is most appropriate to help ensure that no gaps appear in the strategy”
  - But “no complete method applicable to the construction of large general-purpose systems exists yet” (since 1975)

Model-based Security

- Goal:
  - Make the transition from human ideas to executed systems *easy*
  - Increase quality/assurance with bounded time-to-market and cost.



Relatively abstract

Goal: Secure by Design

Consider critical properties

- from very early stages
- within development context
- taking an expansive view
- seamlessly throughout the development lifecycle.

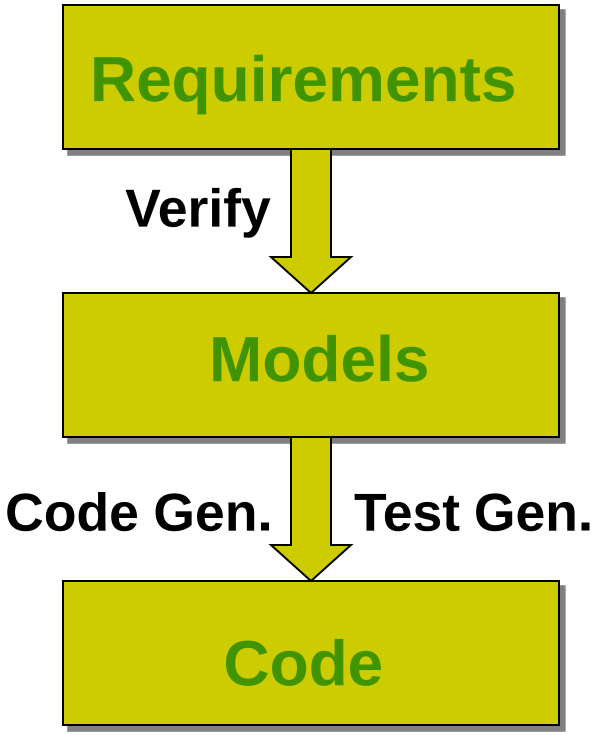
High Assurance/Secure design by model analysis.

High Assurance/Secure implementation by test generation.

Model-based Security Engineering

Combined strategy:

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



Secure by design

- Establish the system fulfills the security requirements
  - At the design level
  - By analyzing the model
- Make sure the code is secure
  - Generate test sequences from the model

Using UML

- UML
  - Provides opportunity for high-quality and cost- and time-efficient high-assurance systems development:
- De-facto standard in industrial modeling: large number of developers trained in UML.
- Relatively precisely defined
- Many tools (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, …).

Requirements on UML extension

Mandatory requirements:

- Provide basic security requirements such as secrecy/confidentiality 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

- 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, ...

UML Extension Goals

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

The High-assurance design UML profiles

- Recurring *critical security* requirements, failure/adversary scenarios, concepts offered as stereotypes with tags at 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.

UML Profile

UML - Review

Unified Modeling Language (UML):

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

Summary of UML Components

- Use case diagram
  - discuss requirements of the system
- Class diagram
  - data structure of the system
- Statechart diagram
  - dynamic component behavior
- 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 (as of 210)

[http://www.omg.org/spec/UML/2.3/]

UML Extension mechanisms

- Stereotype
  - specialize model element using «label».
  - Adds security relevant information to model elements
- Tagged value
  - attach {tag=value} pair to stereotyped element
- Constraint
  - refine semantics of stereotyped element.
- Profile:
  - gather above information.

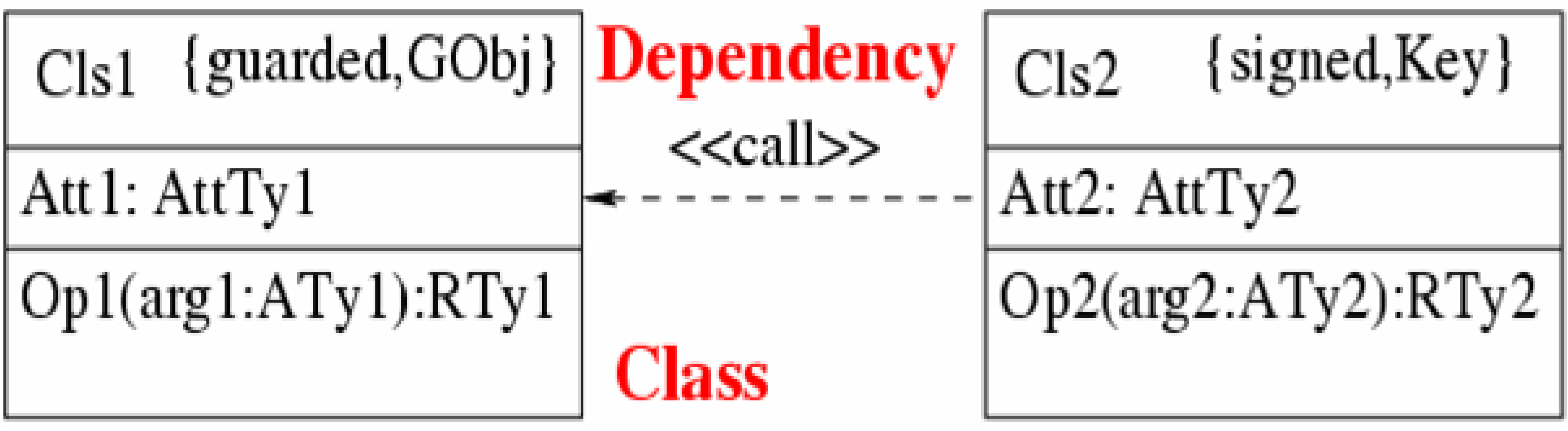
Stereotypes

- Central idea – stereotypes
- Add security relevant information to model elements of three kinds
  - Security assumptions on the physical level of the systems: e.g., «Internet»
  - Security requirements on the logical structure of the system, e.g.,
    - «secrecy» or
    - On specific data values, e.g., «critical»

Stereotypes

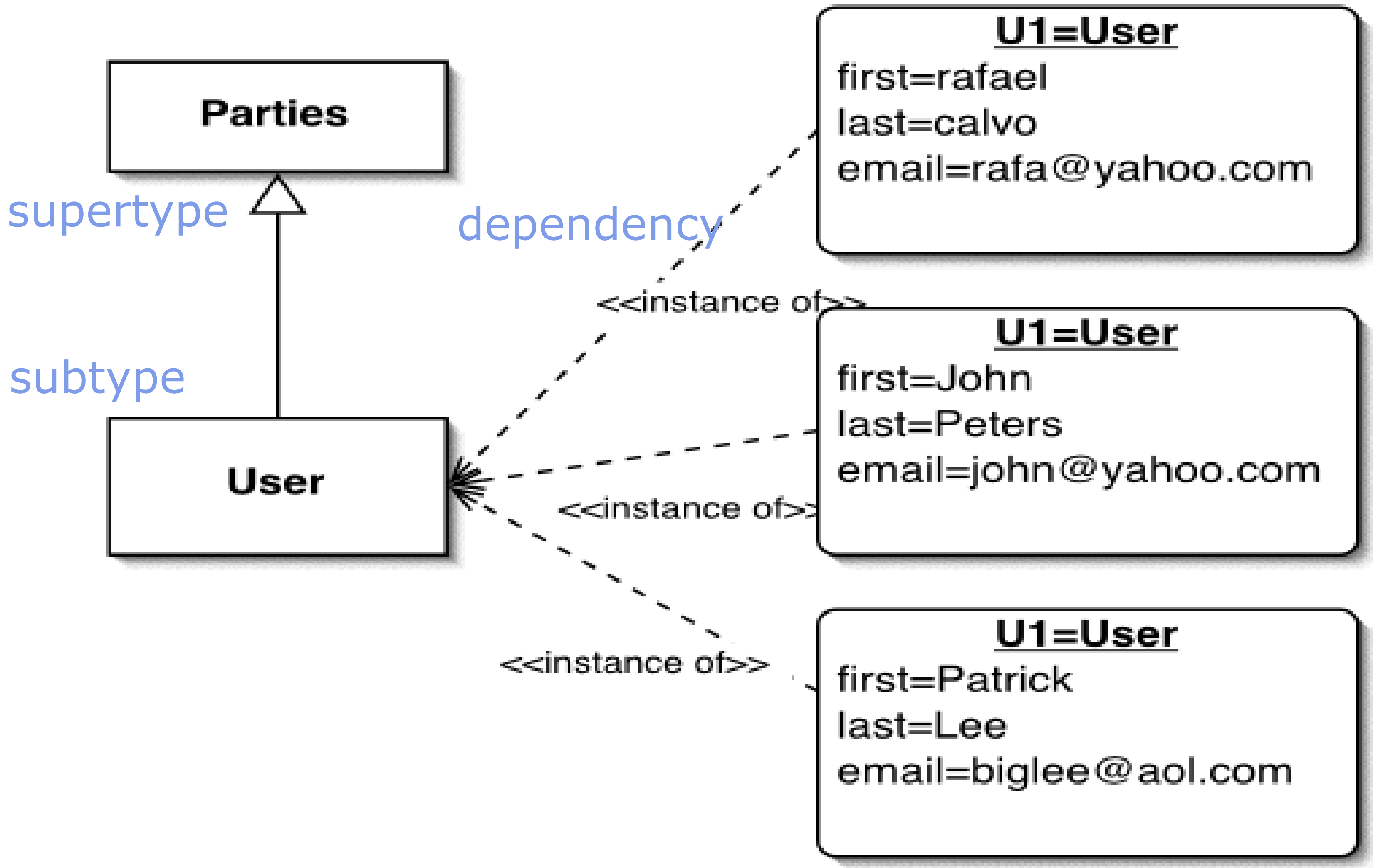
- Security policies that the system parts are supposed to obey; e.g.
  - «fair exchange», «secure links», «data security», «no down-flow»
- First two cases
  - Simply add some additional information to a model
- Third one
  - Constraints are associated that needs to be satisfied by the model

UML run-through: Class diagrams

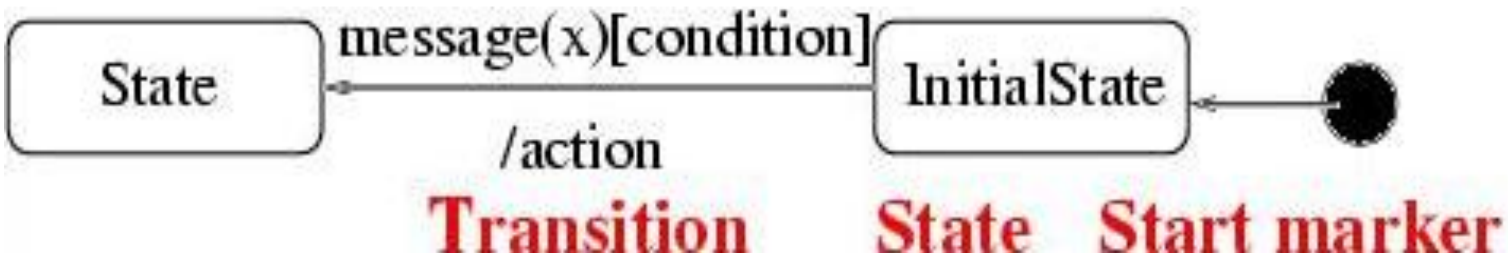


- Class structure of system.
- Classes with attributes and operations/signals;
  - relationships between classes.

UML run-through: Dependency



UML run-through: Statecharts

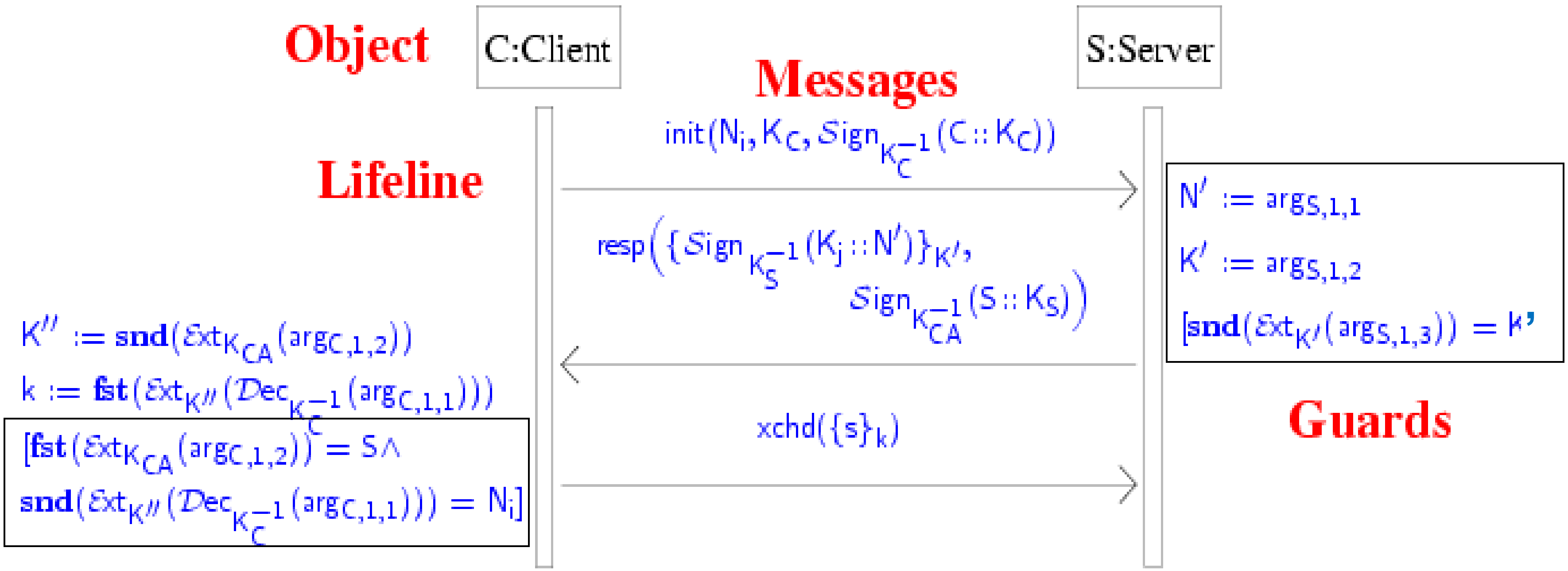


- Dynamic behavior of individual component.
- Input events cause state change and output actions.

e[g]/a

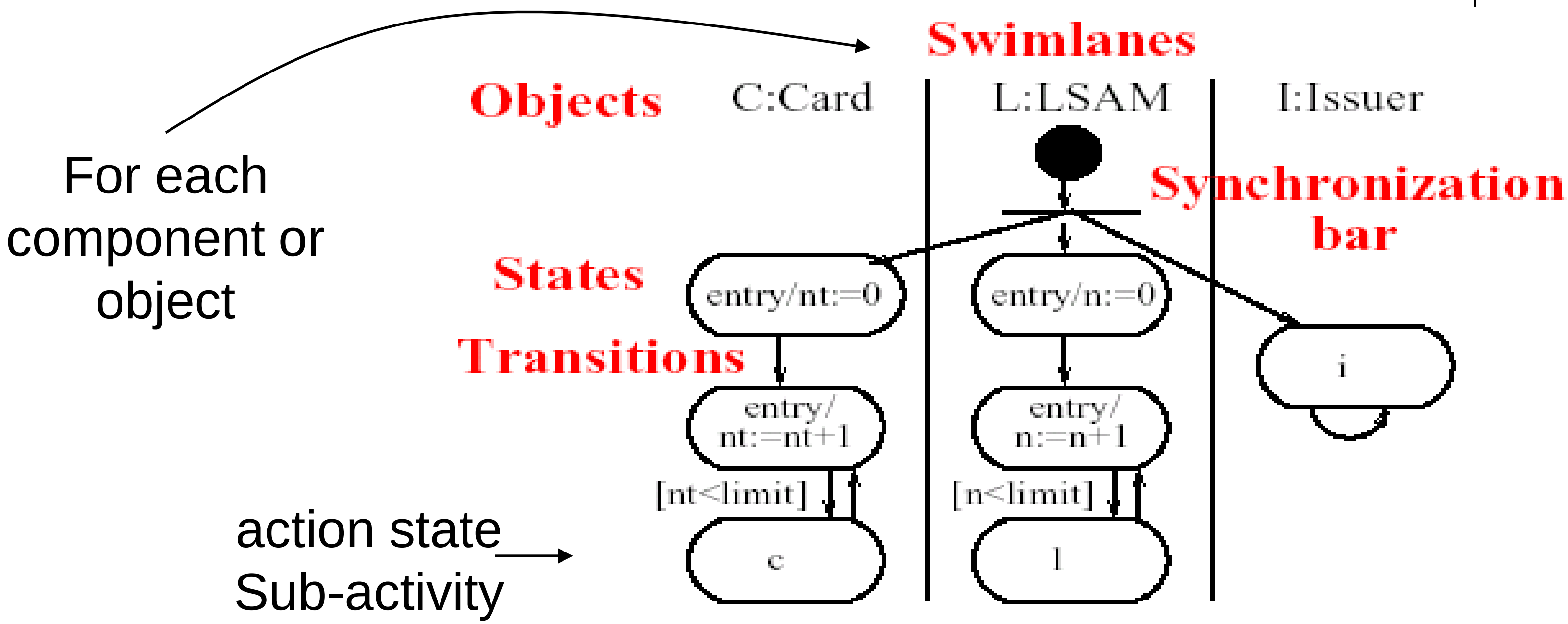
event[guard]/action

UML run-through: Sequence Diagrams



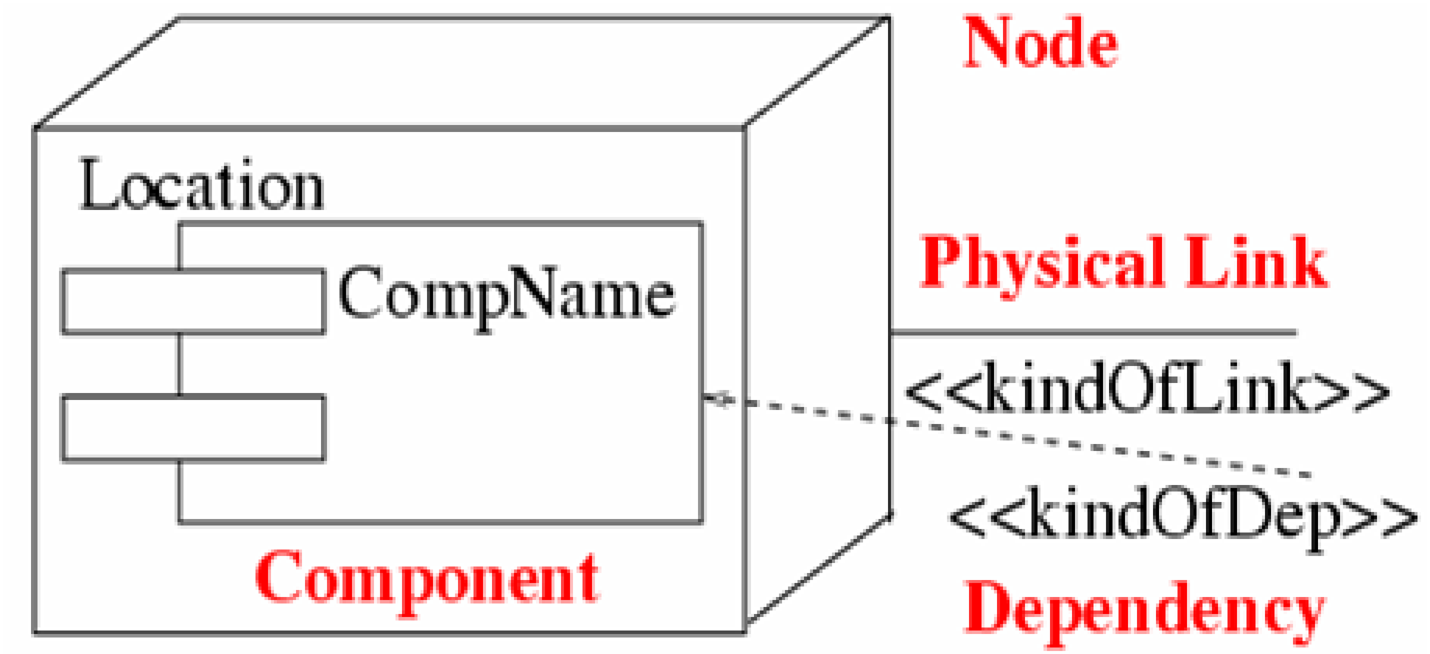
- Describe interaction between objects or components via message exchange.

UML run–through: Activity diagrams



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

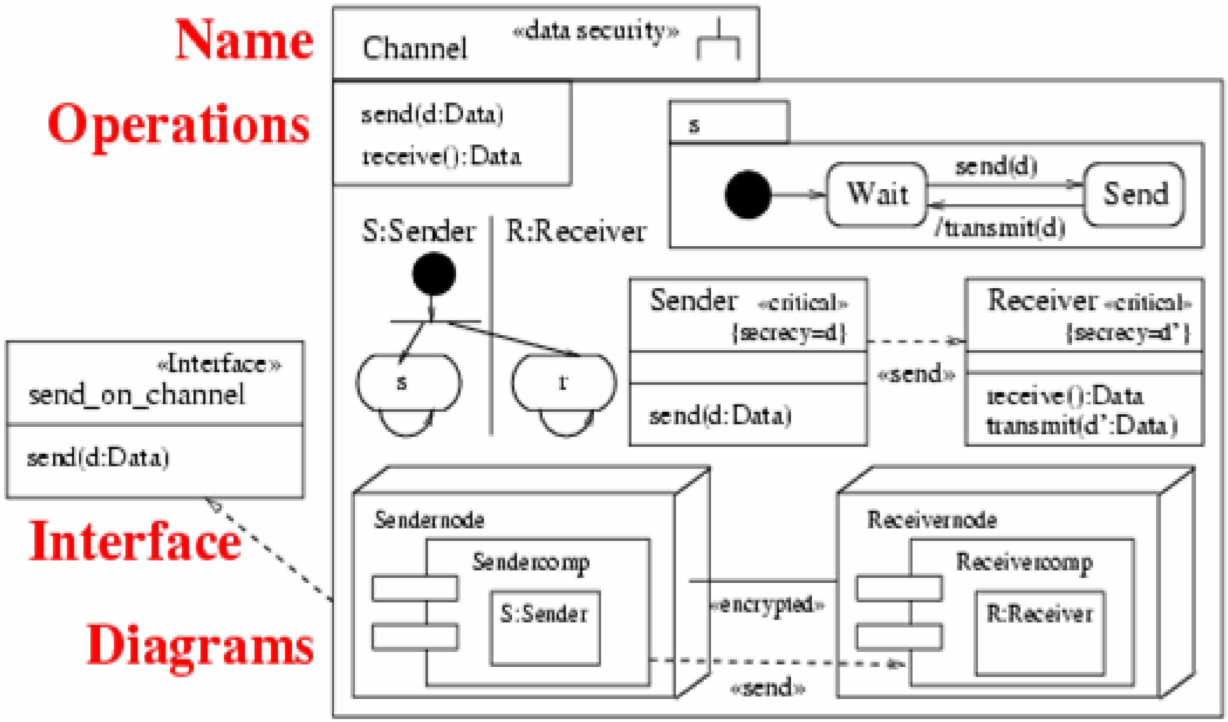
UML Deployment diagrams



Logical (connections)

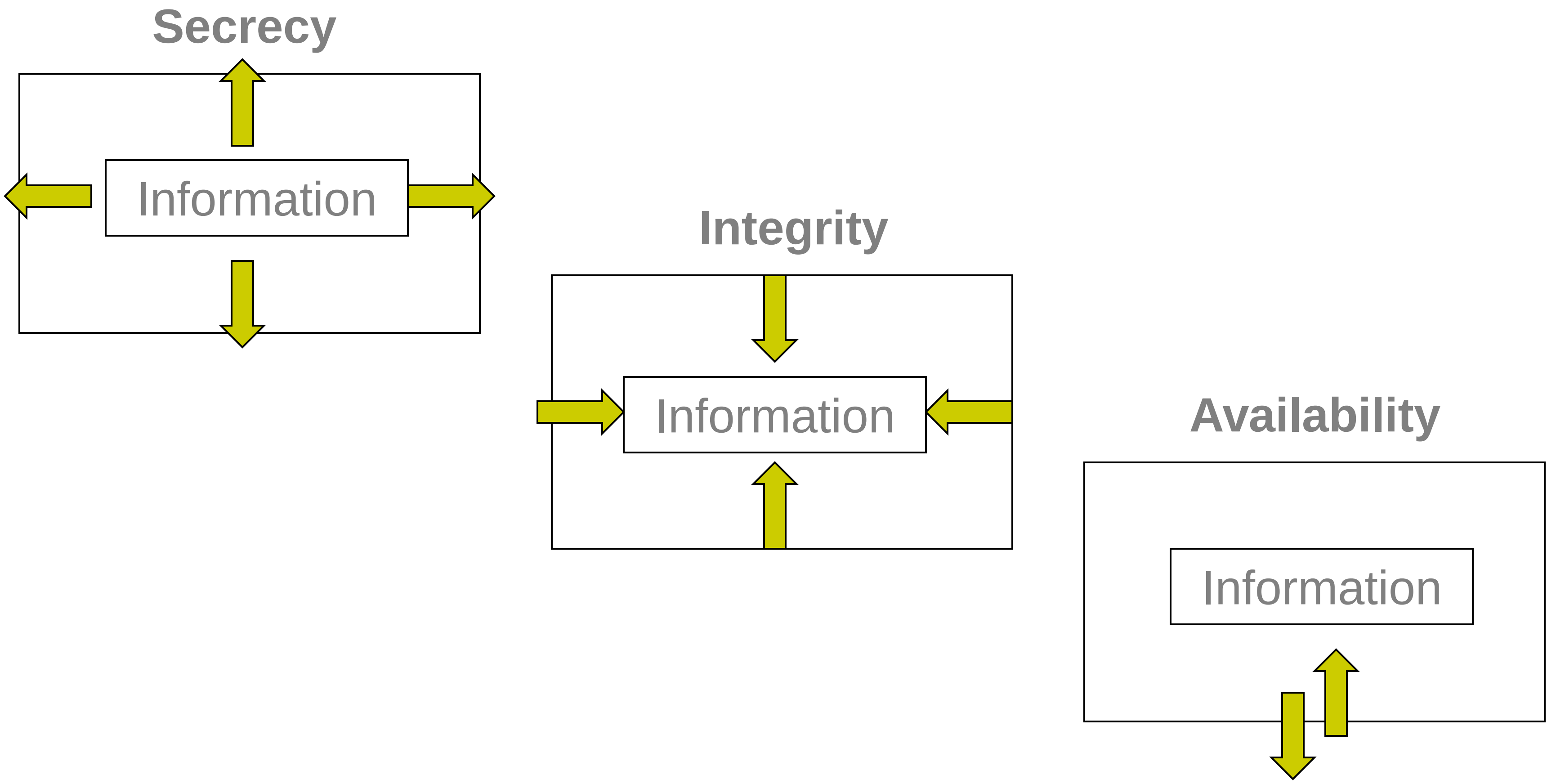
- Describe the physical layer on which the system is to be implemented.

UML Package



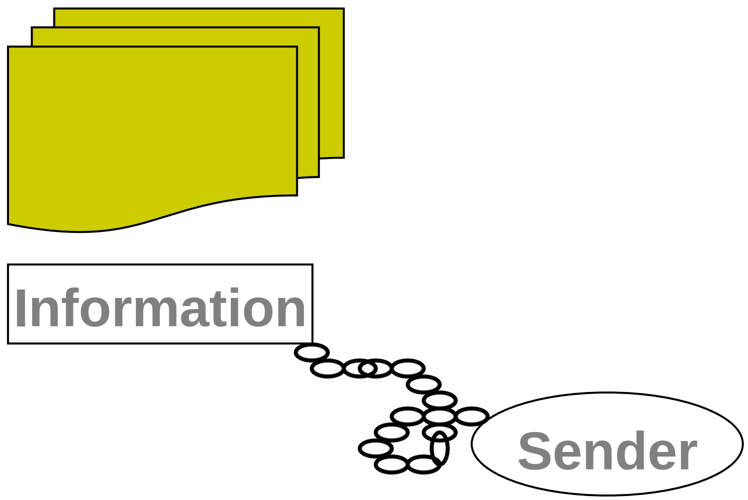
- May be used to organize model elements into groups within a physical system

Basic Security Requirements

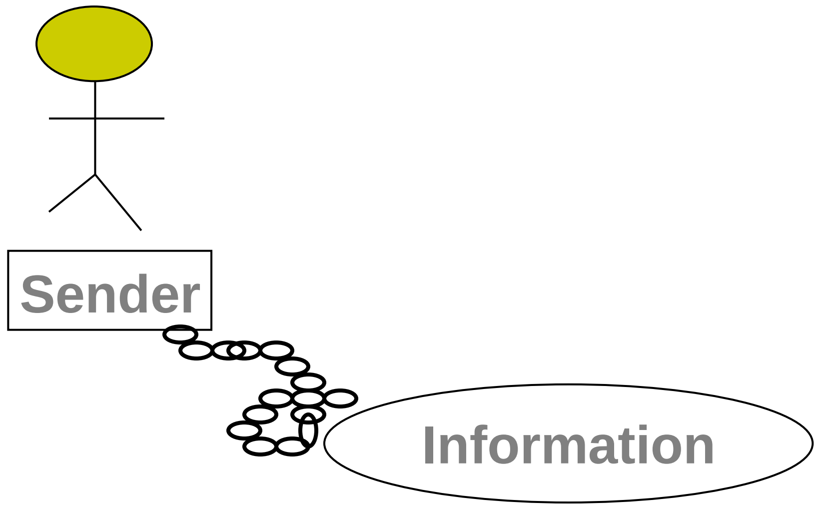


Basic Security Requirements II

Authenticity



Nonrepudiability



UMLsec profile

Stereotype	Base Class	Tags	Constraints	Description
fair exchange	subsystem	start, stop, adversary	after start eventually reach stop	enforce fair exchange
provable	subsystem	action, cert, adversary	action is non-deniable	non-repudiation requirement
rbac	subsystem	protected, role, right	only permitted activities executed	enforces role-based access control
Internet	link			Internet connection
encrypted	link			encrypted connection
LAN	link, node			LAN connection
wire	link			wire
smart card	node			smart card node
POS device	node			POS device
issuer node	node			issuer node
secrecy	dependency			assumes secrecy
integrity	dependency			assumes integrity
high	dependency			high sensitivity
critical	object, subsystem	secrecy, integrity, authenticity, high, fresh		critical object
secure links	subsystem	adversary	dependency security matched by links	enforces secure communication links
secure dependency	subsystem		«call», «send» respect data security	structural interaction data security
data security	subsystem	adversary, integ., auth.	provides secrecy, integrity, authenticity, freshness	basic data security requirements
no down-flow	subsystem		prevents down-flow	information flow condition
no up-flow	subsystem		prevents up-flow	information flow condition
guarded access	subsystem		guarded objects accessed through guards	access control using guard objects
guarded	object	guard		guarded object

Fig. 4.1. UMLsec stereotypes

UMLsec profile

Tag	Stereotype	Type	Multip.	Description
start	fair exchange	state	*	start states
stop	fair exchange	state	*	stop states
adversary	fair exchange	adversary model	1	adversary type
action	provable	state	*	provable action
cert	provable	expression	*	certificate
adversary	provable	adversary model	*	adversary type
protected	rbac	state	*	protected resources
role	rbac	(actor, role)	*	assign role to actor
right	rbac	(role, right)	*	assign right to role
secrecy	critical	data	*	secrecy of data
integrity	critical	(variable, expression)	*	integrity of data
authenticity	critical	(data, origin)	*	authenticity of data
high	critical	message	*	high-level message
fresh	critical	data	*	fresh data
adversary	secure links	adversary model	1	adversary type
adversary	data security	adversary model	1	adversary type
integrity	data security	(variable, expression)	*	integrity of data
authenticity	data security	(data, origin)	*	authenticity of data
guard	guarded	object name	1	guard object

Fig. 4.2. UMLsec tags

<<Internet>> , <<encrypted>> , …

- Kinds of communication links (resp. system nodes)
- For adversary type A, stereotype s, have
  - $Threats_A(s) \subseteq$ {delete, read, insert, access} of actions that adversaries are capable of.

Directly access a physical node

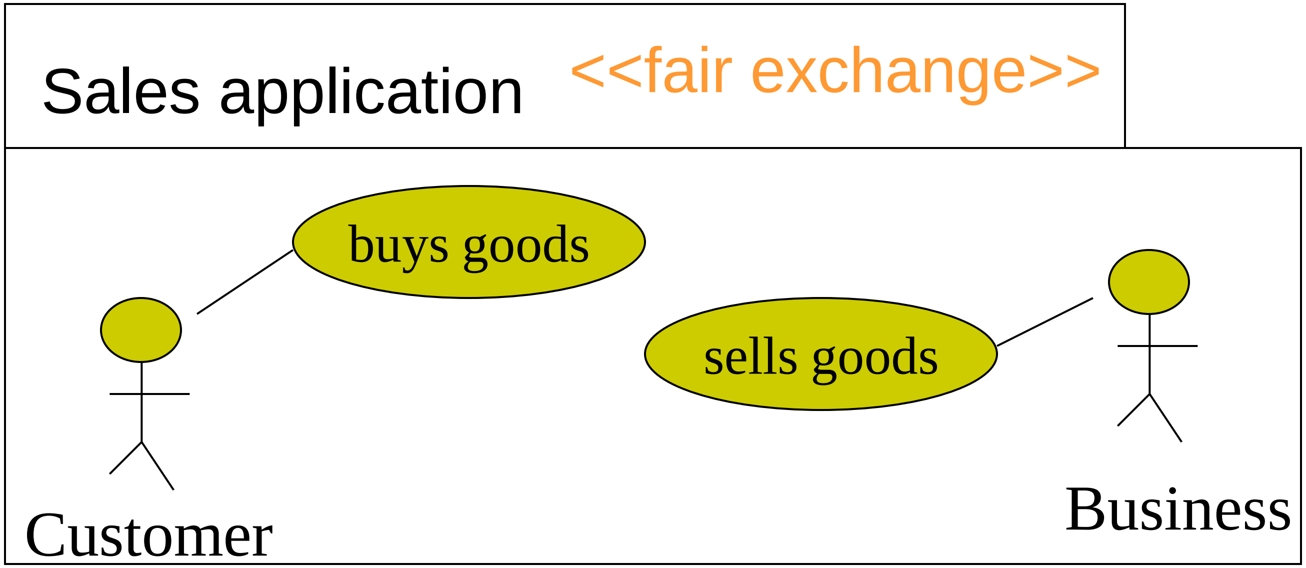
For links

Default attacker

Stereotype	$Threats_{default}()$
•Internet	{delete, read, insert}
•encrypted	{delete}
•LAN	$\emptyset$
•smart card	$\emptyset$

Insider attacker?

Requirements with use case diagrams



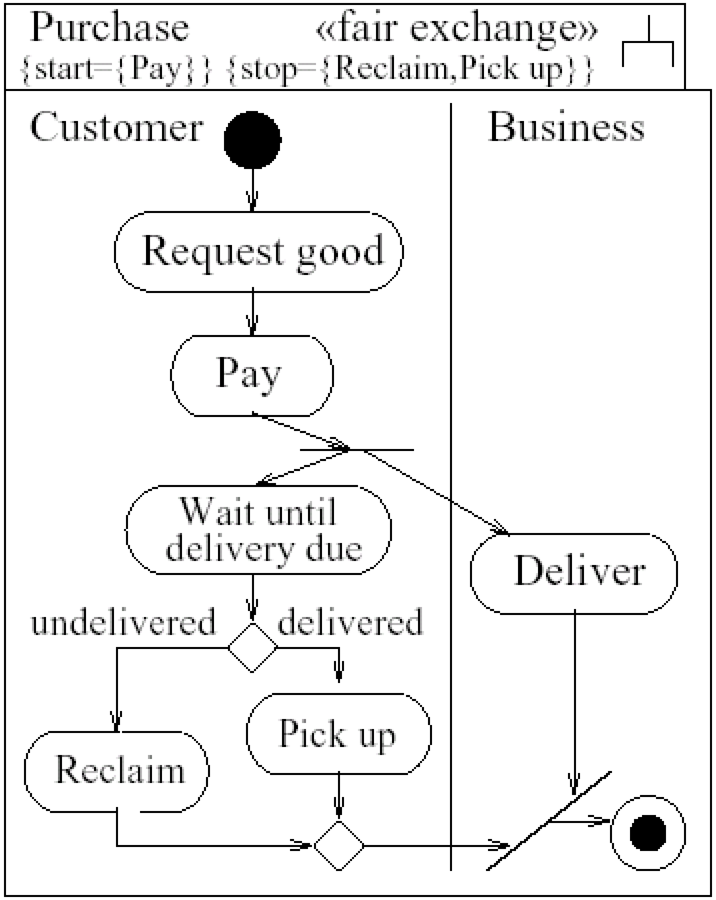
- Capture security requirements in use case diagrams.
- Constraint:
  - need to appear in corresponding activity diagram.

«fair exchange»

- Ensures generic fair exchange condition
  - Avoid cheating
- 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.

«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.

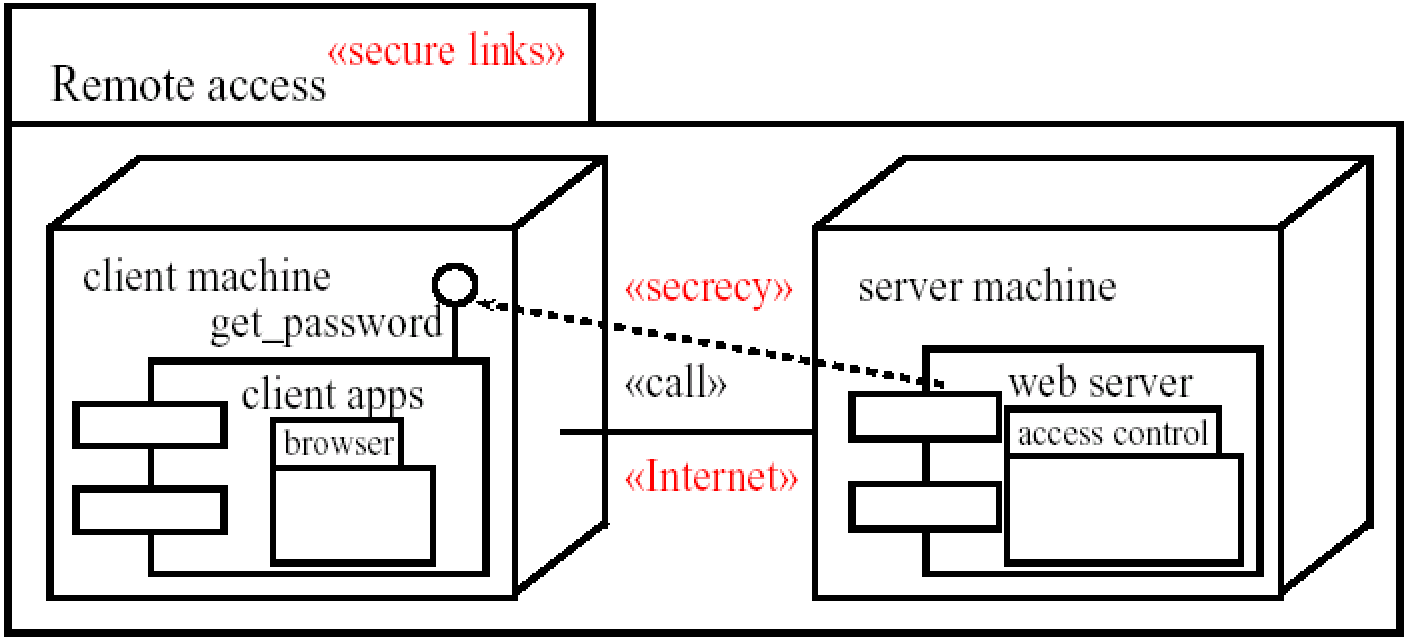


“Pay” may be «provable»

<<secure links>> Example

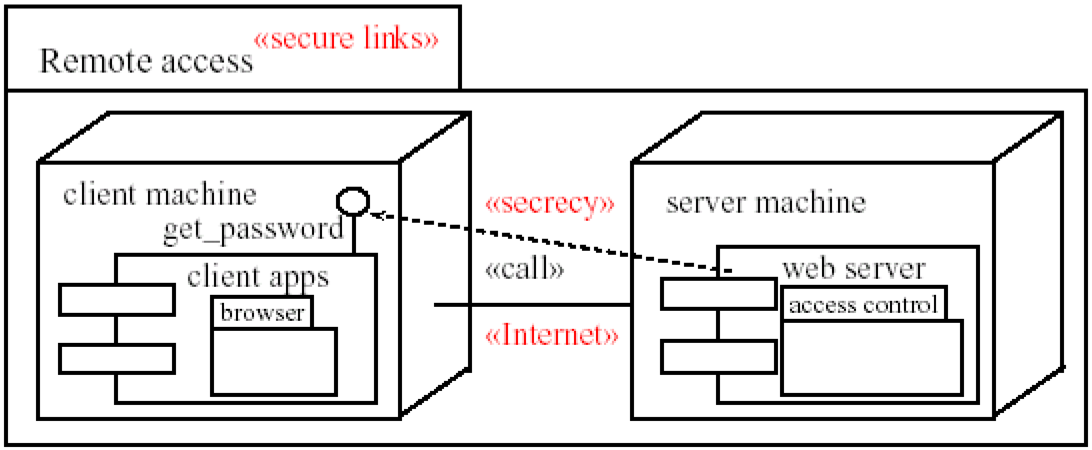
- Ensures that physical layer meets security requirements on communication.
- Constraint:
  - for each dependency *d* with stereotype *s* in { <<secrecy>> , <<integrity>>, <<high>>} between components on nodes *n, m*, have a communication link *l* between *n* and *m* *such that*
    - if *s* = <<high>> : have $Threats_A$ *(l) is empty.*
    - if *s* = <<secrecy>> : have read $\notin$ $Threats_A$ *(l).*
    - if *s* = <<integrity>> : have insert $\notin$ $Threats_A$ *(l).*

<<secure links>> Example



- Given default adversary type, is <<secure links>> provided ?

<<secure links>> Example

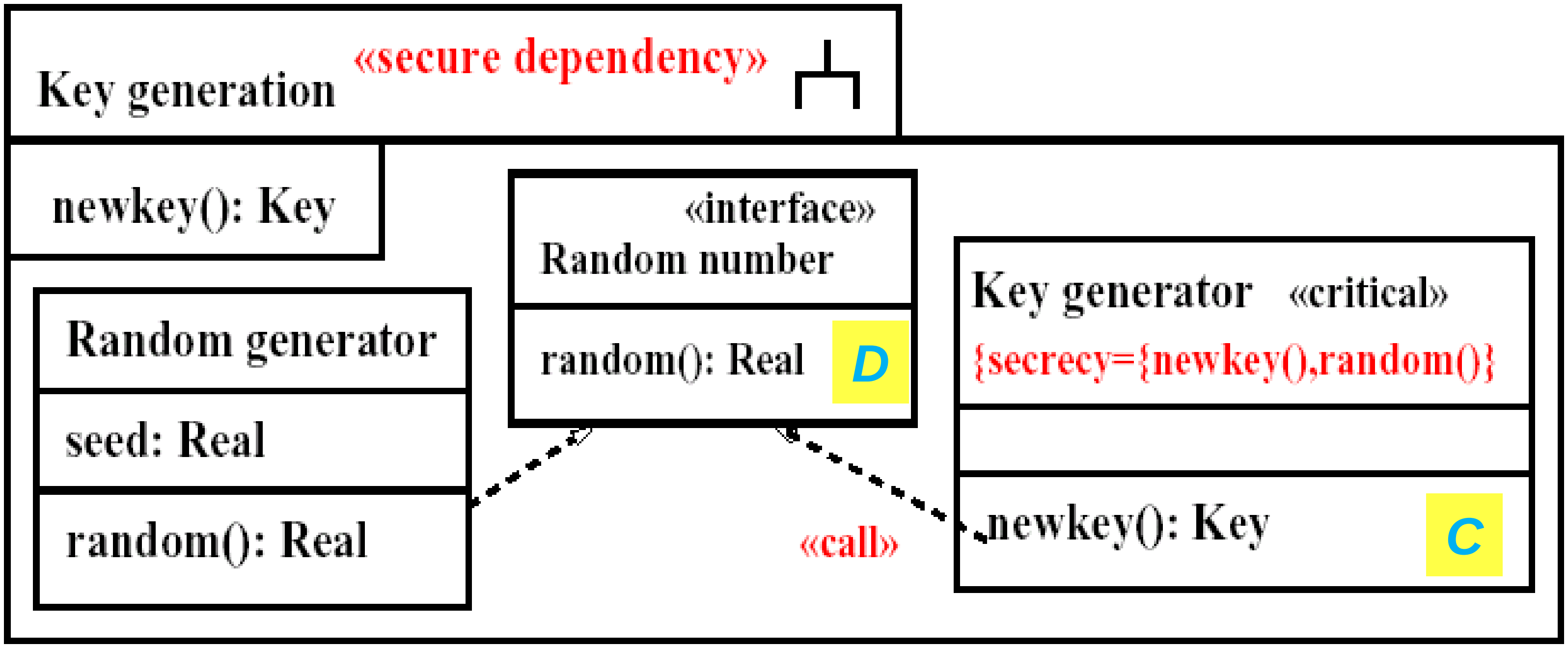


- Given default adversary type, constraint for stereotype <<secure links>> violated:
  - According to the $\text{Threats}_{default}$(Internet) scenario
    - (read $\in \text{Threats}_{default}$(Internet)),
  - <<Internet>> link does not provide secrecy against default adversary.

<<secure dependency>>

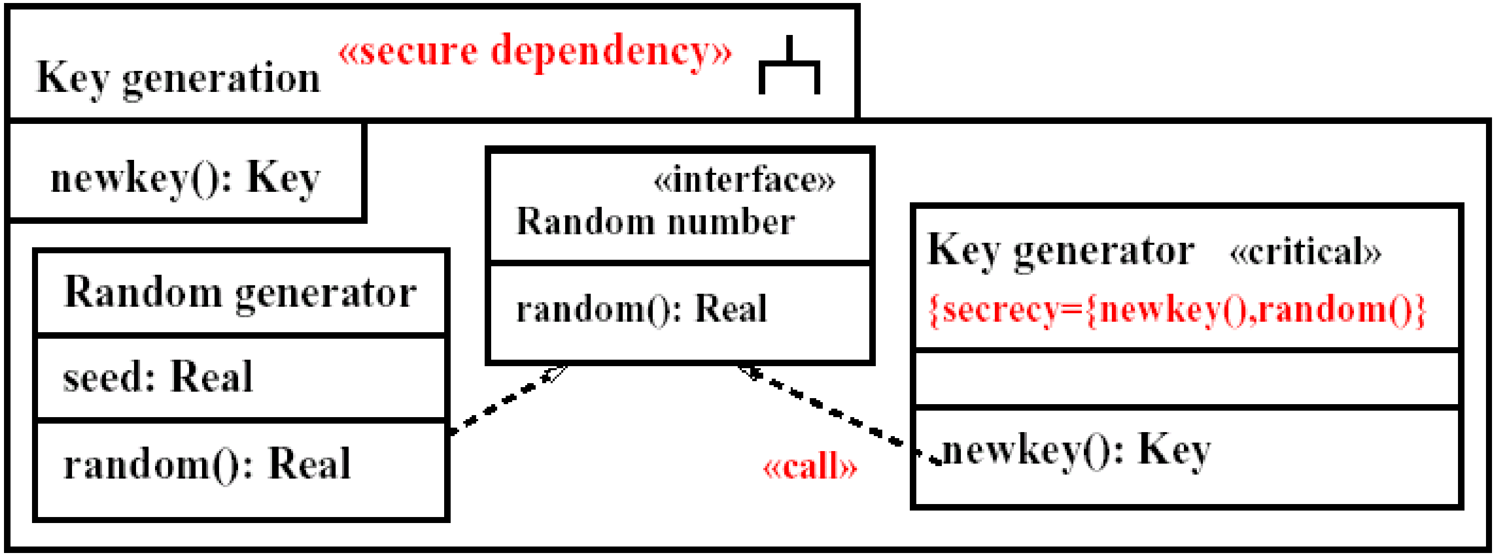
- Ensure that <<call>> and <<send>> dependencies between components respect security requirements on communicated data given by tags {secrecy}, {integrity} and {high}.
- Constraint:
  - for <<call>> or <<send>> dependency from *C* to *D* (for {secrecy}):
    - Msg in *D* is {secrecy} in *C* if and only if also in *D*.
    - If msg in *D* is {secrecy} in *C*, dependency is stereotyped <<secrecy>>.

Example <<secure dependency>>



<<secure dependency>> provided ?

Example
<<secure dependency>>



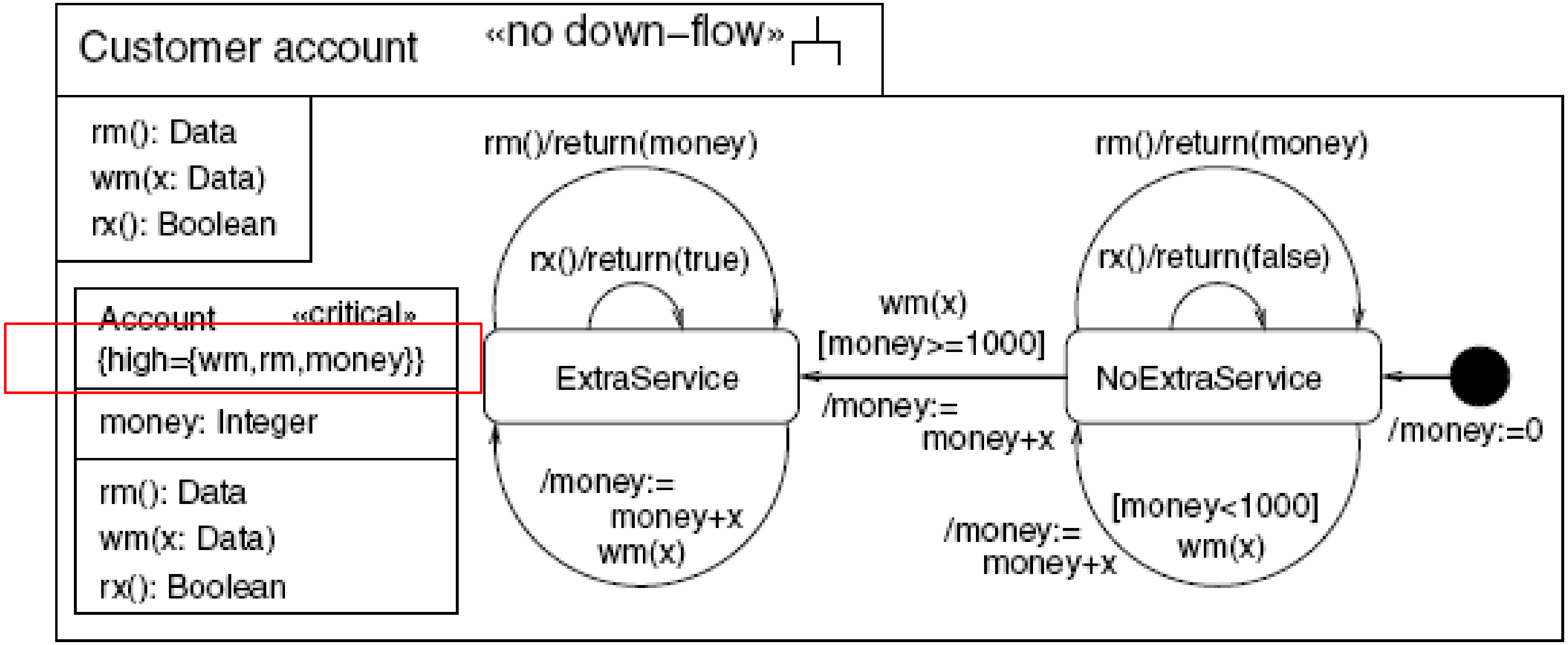
Violates <<secure dependency>> : Random generator and <<call>> dependency do not give security level for random() to key generator.

<<no down–flow>>

- Enforce secure information flow.
- Constraint:
  - Value of any data specified in {high} may influence only the values of data also specified in {high}.

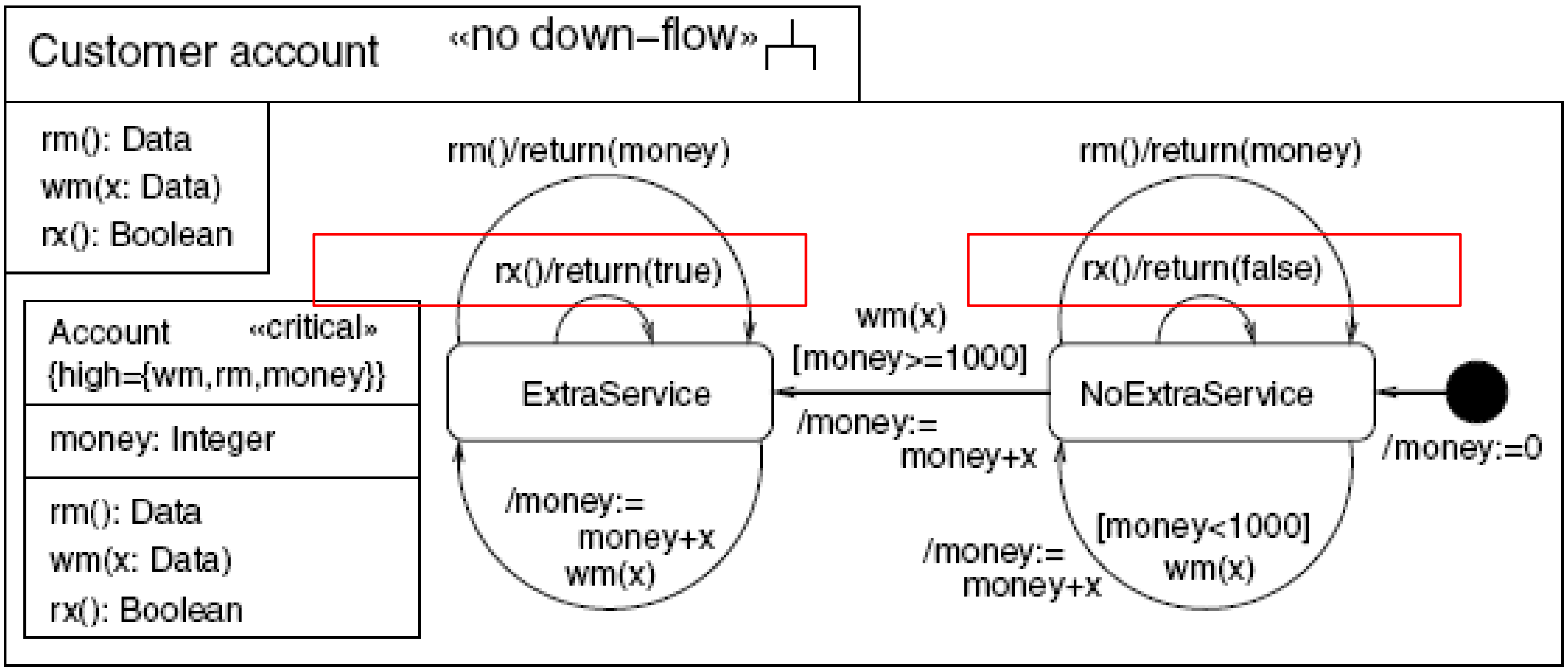
Formalize by referring to formal behavioral semantics.

Example
<<no down-flow>>



<<no down–flow>> provided ?

Example
<<no down-flow>>



- <<no down–flow>> violated: partial information on input of high wm() returned by non-high rx().

<<data security>>

- Behavior of Subsystem with this tag respects
  - Security requirements of data marked <<critical>> enforced against A from deployment diagram.
- Constraints:
  - Secrecy {secrecy} of data preserved against A
  - Integrity {integrity} of (v, E) preserved against A
  - Authenticity {integrity} of (a, o) pres
  - Freshness {fresh} : data in **Data** U fresh

Default (E is not mentioned): A should not be able to make the variable v take on a value previously known only to him

Assumption: A does not know data being protected

Notation

- $\_ :: \_$ (concatenation)
- $\mathbf{head}(\_)$ and $\mathbf{tail}(\_)$ (head and tail of a concatenation)
- $\{\_\}_\_$ (encryption)
- $\mathcal{D}ec_\_(\_)$ (decryption)
- $\mathcal{S}ign_\_(\_)$ (signing)
- $\mathcal{E}xt_\_(\_)$ (extracting from signature)
- $\mathcal{H}ash(\_)$ (hashing)

by factoring out the equations:

- $\mathcal{D}ec_{K^{-1}}(\{E\}_K) = E$ (for all $E \in \mathbf{Exp}$ and $K \in \mathbf{Keys}$)
- $\mathcal{E}xt_K(\mathcal{S}ign_{K^{-1}}(E)) = E$ (for all $E \in \mathbf{Exp}$ and $K \in \mathbf{Keys}$)
- and the usual laws regarding concatenation, **head**(), and **tail**():
  - $(E_1 :: E_2) :: E_3 = E_1 :: (E_2 :: E_3)$ (for all $E_1, E_2, E_3 \in \mathbf{Exp}$)
  - $\mathbf{head}(E_1 :: E_2) = E_1$ (for all expressions $E_1, E_2 \in \mathbf{Exp}$) and
  - $\mathbf{tail}(E_1 :: E_2) = E_2$ (for all expressions $E_1, E_2 \in \mathbf{Exp}$ such that there exist no E, E' with $E_1 = E :: E'$). For all other cases, **head**() and **tail**() are undefined.

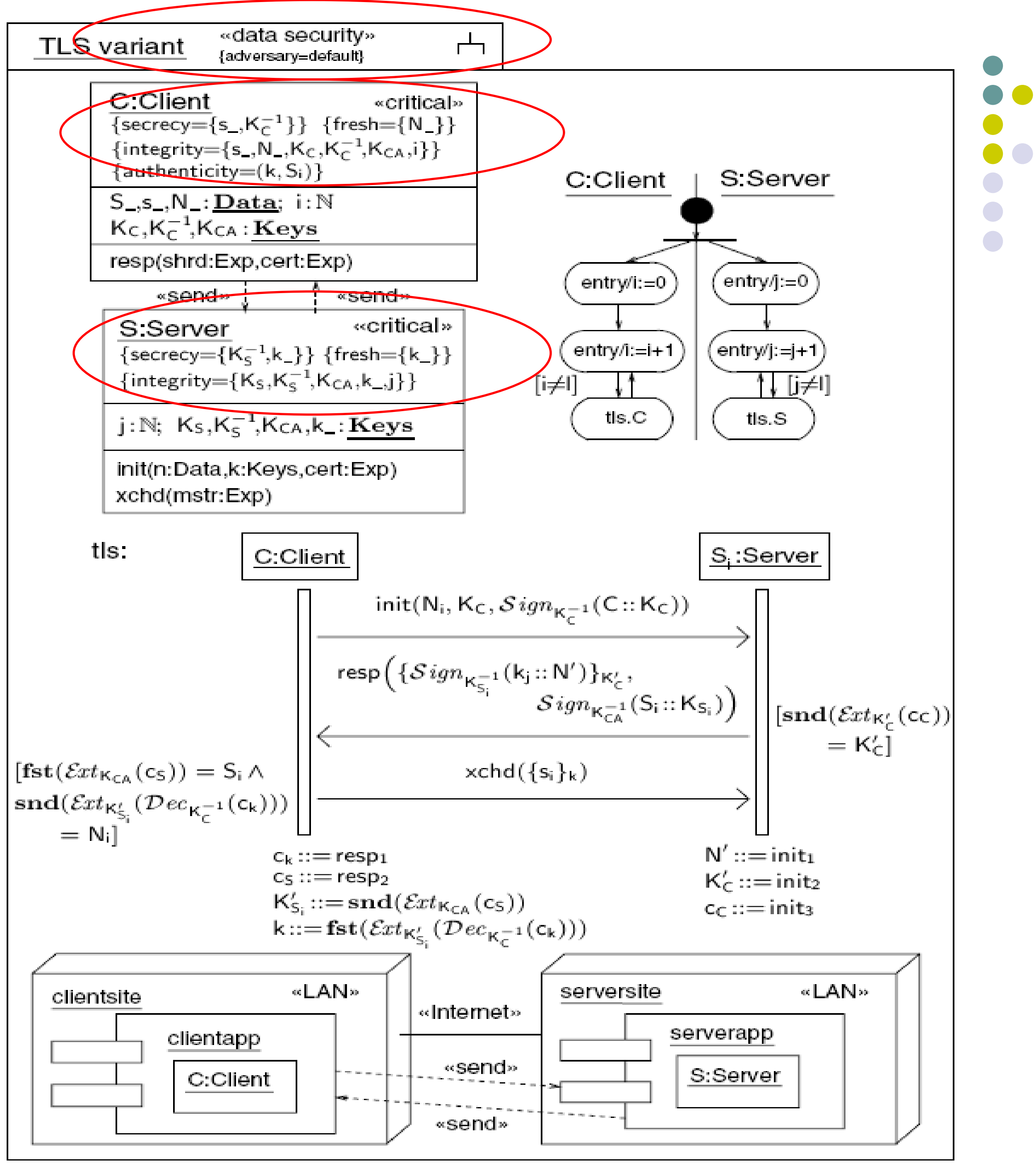
For each $E \in \mathbf{Exp}$, we use the following abbreviations:

- $\mathbf{fst}(E) \stackrel{\text{def}}{=} \mathbf{head}(E)$
- $\mathbf{snd}(E) \stackrel{\text{def}}{=} \mathbf{head}(\mathbf{tail}(E))$
- $\mathbf{thd}(E) \stackrel{\text{def}}{=} \mathbf{head}(\mathbf{tail}(\mathbf{tail}(E)))$.

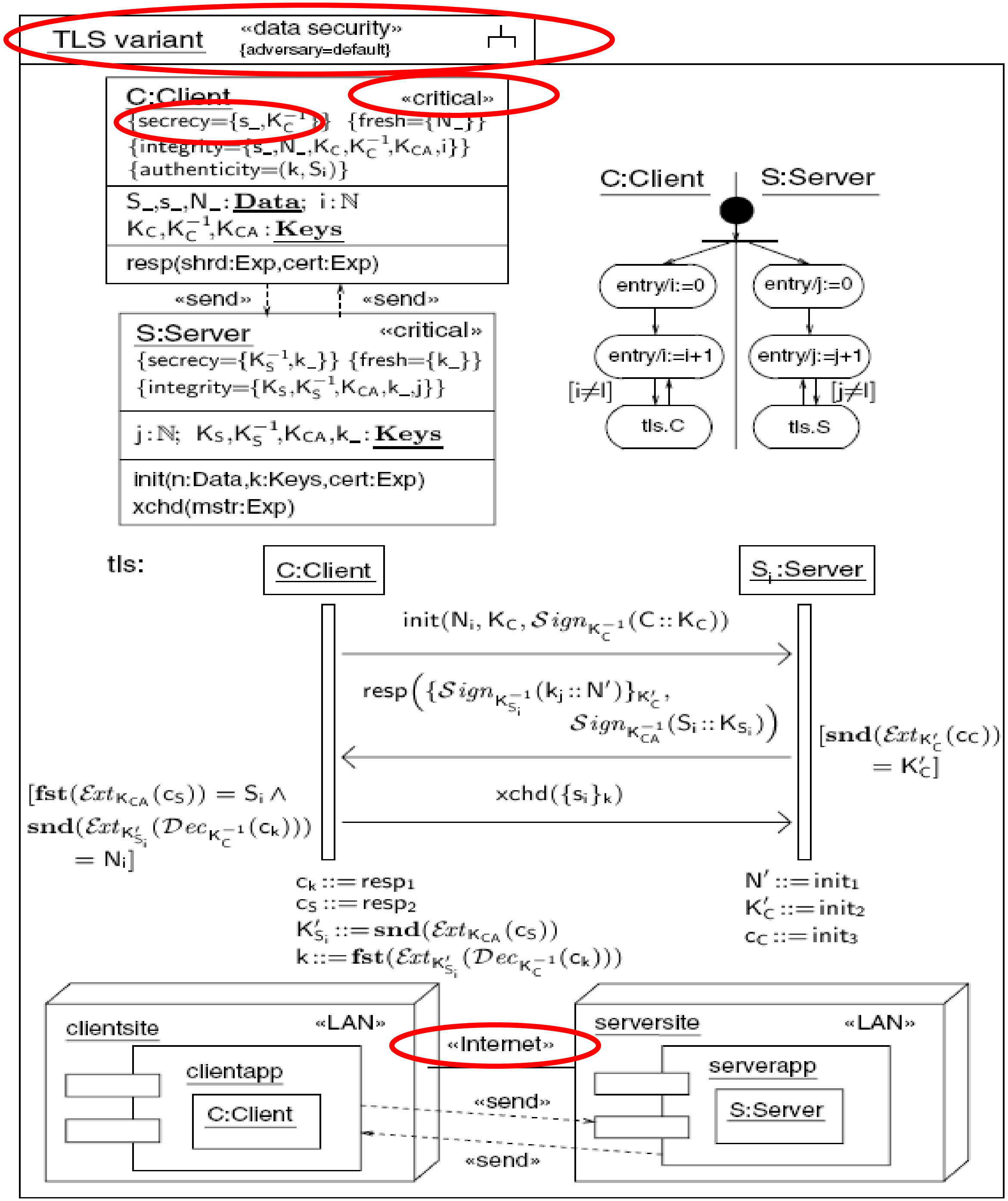
TLS goals: Secure channel between client and server

-Secrecy and Server Authenticity

Variant of TLS (INFOCOM`99): <<data security>> against default adversary provided ?



Violates {secrecy} of si against default adversary.



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

$C \rightarrow A$: $N_i::K_C::Sign_{K_C^{-1}}(C::K_C)$

$A \rightarrow S$: $N_i::K_A::Sign_{K_A^{-1}}(C::K_A)$

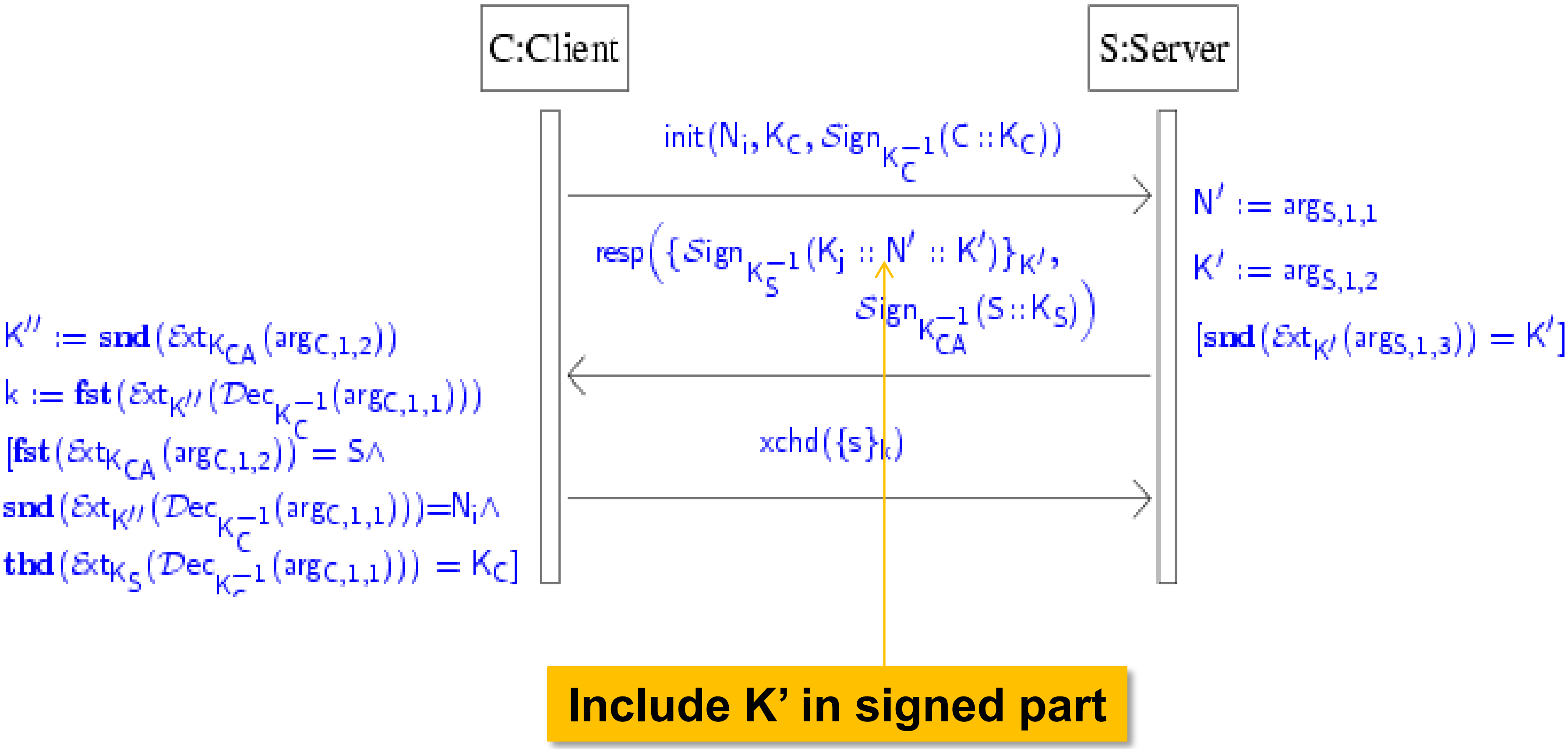
$S \rightarrow A$: $\{Sign_{K_S^{-1}}(K_j::N_i)\}_{K_A}::Sign_{K_{CA}^{-1}}(S::K_S)$

$A \rightarrow C$: $\{Sign_{K_S^{-1}}(K_j::N_i)\}_{K_C}::Sign_{K_{CA}^{-1}}(S::K_S)$

$C \rightarrow A$: $\{s\}_{K_j}$

$A \rightarrow S$: $\{s\}_{K_j}$

The fix



C:Client
S:Server
init(N_i, K_C, Sign_{K_C^{-1}}(C :: K_C))
N' := args_{S,1,1}
K' := args_{S,1,2}
resp({Sign_{K_S^{-1}}(K_j :: N' :: K')}_{K'}, Sign_{K_{CA}^{-1}}(S :: K_S))
[snd(Ext_{K'}(args_{S,1,3})) = K']
K'' := snd(Ext_{K_{CA}}(arg_{C,1,2}))
k := fst(Ext_{K''}(Dec_{K_C^{-1}}(arg_{C,1,1})))
[fst(Ext_{K_{CA}}(arg_{C,1,2})) = S ∧
snd(Ext_{K''}(Dec_{K_C^{-1}}(arg_{C,1,1})))=N_i ∧
thd(Ext_{K_S}(Dec_{K_C^{-1}}(arg_{C,1,1}))) = K_C]
xchd({s}_k)
Include K' in signed part

<<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.

Application

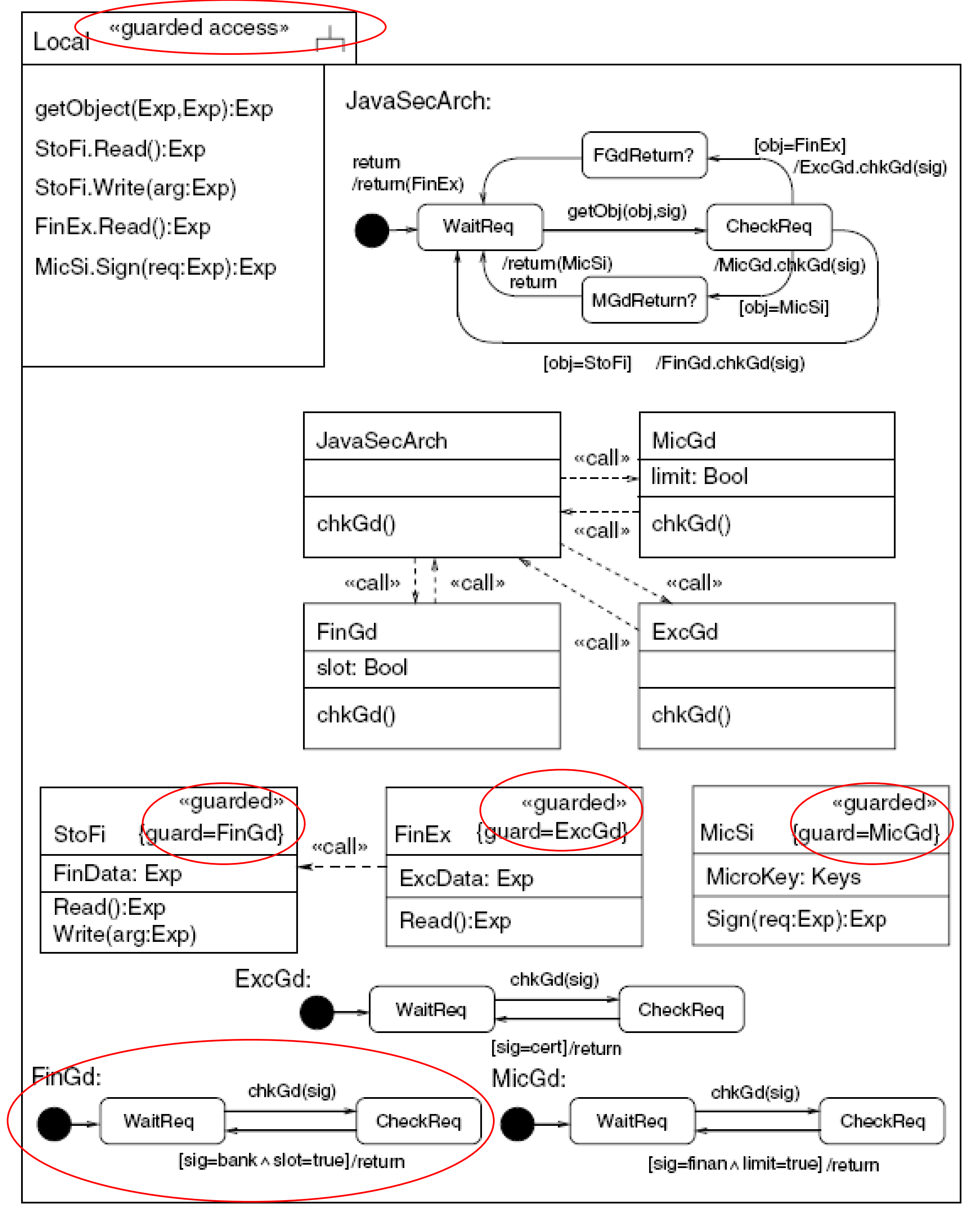
- Web-based financial application
  - Internet Bank: BankEasy
  - Financial advisor: Finance
- A local client needs to provide applets from these certain privileges
  - Access to local financial data: using GuardedObjects
  - Guarded objects: StoFi, FinEx, MicSi

Example: applets that are signed by the bank can read and write the financial data stored in local database, but only between 1 – 2PM

  - Enforced by FinGd guard object
    - Slot is fulfilled iff time is 1-2PM

- Provides <<guarded access>> : Access to MicSi protected by MicGd.

slot could be "between 1 and 2PM



Does UMLsec meet requirements?

- Security requirements: <<secrecy>> ,…
- Threat scenarios: Use Threats_{adv}(ster).
- Security concepts: e.g. <<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.

Design Principles

- How principles are enforced
  - Economy of mechanism
    - Guidance on employment of sec mechanisms to developers – use simple mechanism where appropriate
  - Fails-safe defaults
    - Check on relevant invariants – e.g., when interrupted
  - Complete mediation
    - E.g., guarded access
  - Open design
    - Approach does not use secrecy of design

Design Principles

- Separation of privilege
  - E.g. guarded objects that check for two signatures
- Least privilege
  - Basically meet the functional requirements as specified; includes an algorithm to determine least privilege given a functional specification
- Least Common Mechanism
  - Based on the object oriented approach
- Psychological acceptability
  - Emphasis on ease of development through a standard tool extension