Vorlesung

Methodische Grundlagen des Software-Engineering

im Sommersemester 2013

Prof. Dr. Jan Jürjens

TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

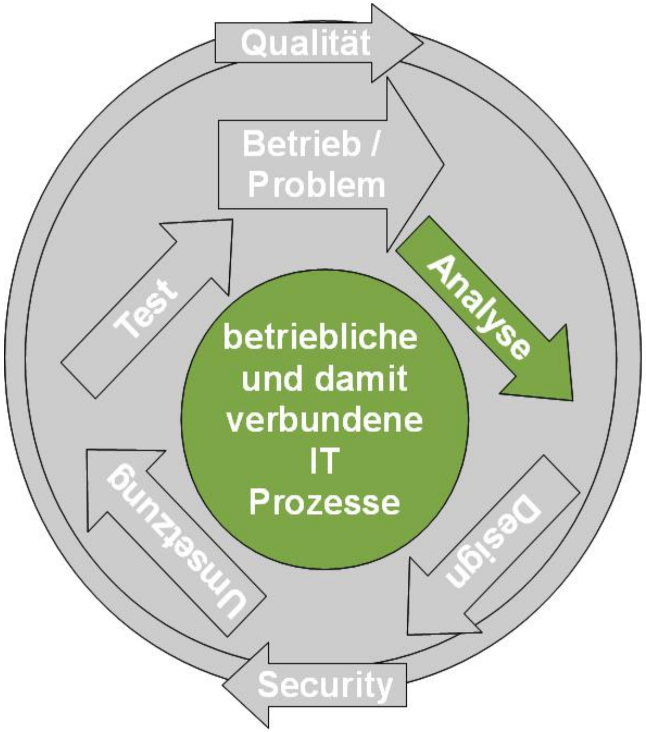
3.5 UML Model Analysis

v. 19.06.2013

3.5 UML Model Analysis

Einordnung
3.5 UML model analysis

- Geschäftsprozessmodellierung
- Process-Mining
- Modellbasierte Entwicklung sicherer Software
  - Model-Driven Architecture
  - Sicherheitsanforderungen
  - UMLsec
  - Design Principles
  - UML model analysis
  - Examples
    - TLS Variant
    - CEPS Purchase



Analyzing UML Models

To check security requirements in a UML model mechanically we need an analysable model, which means:

- The UMLsec profile is attached to it.
- The security-relevant information from the security-oriented stereotypes (i.a. adversary type).

This means, we need to formulate constraints on the UML models which model security requirements that can be rather subtle.

On the following slides we define and explain the properties of such a model which we need for formalizing the constraints in the UMLsec profile.

Notation (1/2)

We assume the usual definitions from elementary set theory and logic, which may be found for example in *"Handbook of logic in computer science"**, including the following definitions:

- $\mathbb{N}$ is the set of non-negative integers.
- $\mathbb{N}_n$ is the set of non-negative integers up to and including n, for any $n \in \mathbb{N}$.
- $P(X)$ is the set of subsets of a set X.

Given a sequence (or list) $l = (l_1, l_2, l_3, \ldots)$, we write:

- $head(l)$ for its head element l_1
- $tail(l)$ for its tail $(l_2, l_3, \ldots)$
- $[\ ]$ for the empty list (in particular for the empty string)

* S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors. Handbook of logic in computer science, volume 1-5, pages xii+827. The Clarendon Press, New York, 1992-2000.

Notation (2/2)

- A multi-set (or bag) is a set which may contain multiple copies of an element, with notation *{{ }}* instead of the usual brackets.
  - e.g. $\{\{1,1,1,1,1\}\}$ is the multi-set consisting of five copies of the element *1*.
- For a multi-set M and a set X:
  - $M \searrow X$ filters all elements out of M, which are elements of X.
- For two multi-sets M and N:
  - $M \cup N$ is their union.
  - $M \setminus N$ is the subtraction of N from M.
  - $M \subseteq N$ if $M \searrow N = M$
- For a multi-set M
  - $\lfloor M \rfloor$ is the set of elements in M.
  - $\#M$ is the number of elements in M.

Outline of Formal Semantics Messages

In UML, both objects and system components can communicate by exchanging messages from a given set *Events*.

- The arrival of such a message is called *event*.
- They consist of:
  - a message name from a given set *MsgNm*.

    (Message names may be prefixed with object or instance names from a given set *UMNames*)
  - possibly arguments to the message which are elements of a given set *Exp* of expressions (see this slide).

$$msg = messagename(exp_1, \dots, exp_n)$$

Outline of Formal Semantics Input/Output queues

Each object or component *O* may

- receive messages in an multi-set $inQu_O$ called input queue.
- releases messages to an multi-set $outQu_O$ called output queue.

We use multi-sets rather then sets, because several copies of the same message can be received concurrently.

Sending a message from an object or subsystem instance *S* to an object or subsystem instance *R*:

- *S* places the message *R.msg* into its multi-set $outQu_S$.
- A scheduler distributes the message from output queues to the intended input queues, while removing the messages head. In particular, *R.msg* is removed from $outQu_S$ and *msg* added to $inQu_R$.
- *R* removes *msg* from its input queue and processes its content.

Outline of Formal Semantics
Further details

- In the case of operation calls, we also need to keep track of the sender to allow sending return signals.
- This way of modelling communications allows for a very flexible treatment.
  - e.g. we can modify the behaviour of the scheduler to take account of knowledge on the underlying communication layer.
- This allows us to consider security issues or other aspects, such as ordering or delay of messages.

Outline of Formal Semantics
Single objects

- At the level of single objects, behaviour is modelled using statecharts or sequence diagrams.
- The internal activities contained as states of these statecharts can, e.g., be defined using statecharts or sequence diagrams.
- Using subsystems, one can then define the behaviour of a system component *C* by including an activity diagram that coordinates the respective activities of the various components and objects.

Outline of Formal Semantics UML machine (1/3)

- For each object or component *C* of a given system, our semantics defines a so-called UML machine *[[C]]*, which
  - is a state machine.
  - communicates with its environment using messages.
- Specifically, the behavioural semantics *[[D]]* of a statechart diagram *D* models the run-to-completion semantics of UML statecharts.
- Any sequence diagram *S* gives us the behaviour *[[S.C]]* of each contained component *C*.
- Subsystems group together diagrams describing different parts of a system: A system component *C* given by a subsystem *S* may contain subcomponents $C_1,\ldots,C_n$.
- These subcomponents may communicate through the communication links in the corresponding deployment diagram.

Outline of Formal Semantics UML machine (2/3)

To get the behavioural interpretation *[[S]]* of a UML subsystem specification *S*, it will be defined as follows:

1. It takes a multi-set of input events *(incoming messages).*
2. The events are distributed from the input multi-set and the link queues connecting the subcomponents and given as arguments to the functions defining the behaviour of the intended recipients in *S*.
3. The output messages from these functions are distributed to the link queues of the links connecting the sender of a message to the receiver, or given as the output from *[[S]]* when the receiver is not part of *S*.

[[S]] is a UML machine.

Outline of Formal Semantics UML machine (3/3)

- An execution of a UML subsystem *S* is then a sequence of states and the associated multi-sets of input and output messages of *[[S]]*.
- UML specifications may be non-deterministic, e.g. because several transitions in a statechart diagram may be able to fire at a given point in time.
- A subsystem *T* is a *black-box refinement* of *S* if every observable input/output behaviour of *T* is also an input/output behaviour of *S*.
- *T* is a *delayed black-box refinement* of *S* if every observable input/output behaviour of *T* differs from an input/output behaviour of *S* only in that delays may be introduced.

Security Analysis

Following Dolev, Yao (1983): To analyze system, verify against attacker model from threat scenarios in deployment diagrams who

- may participate in some protocol runs,
- knows some data in advance,
- may intercept messages on some links,
- may inject produced messages in some links,
- may access certain nodes.

Adversaries

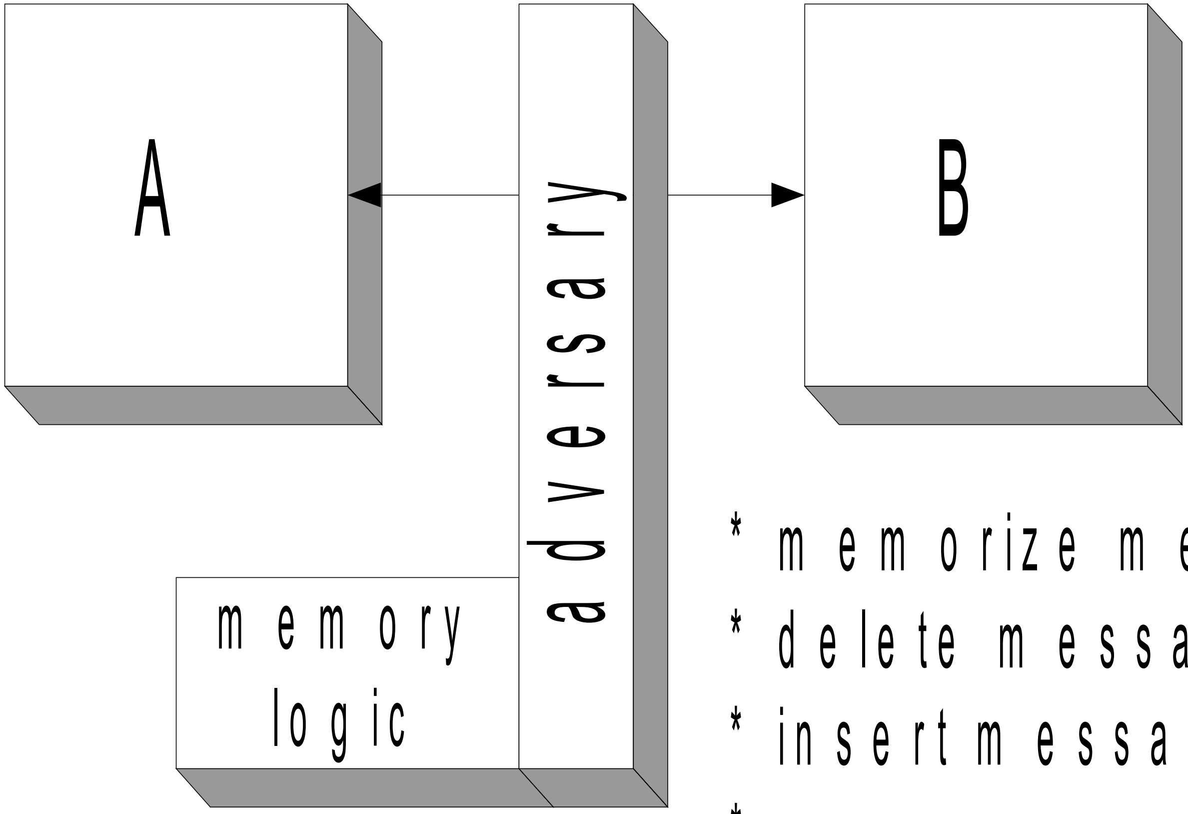
Model classes of adversaries.

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.

Adversary Model



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

Adversary types revisited

<<Internet>>, <<encrypted>>, <<LAN>>, <<smart card>>: *Stereotypes* for kinds of communication links resp. system nodes.

Default attacker is able to read, delete, insert and access messages on an Internet link. On an encrypted Internet link, such as a virtual private network, the attacker might still be able to delete messages, without knowing their encrypted content, by bringing down a network server.

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

Default attacker:

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

Cryptography: Assumptions

Keys are symbols, crypto-algorithms are abstract operations

- Can only decrypt with right keys.
- Can only compose with available messages.
- Cannot perform statistical attacks.
- Cannot guess an encrypted value without knowing the decryption key.
- Symmetric encryption provides data integrity (e.g. using Message Authentication Codes (MACs)).

Cryptography: Keys

- *Keys* is a set with a partial injective map:

  $( )^{-1} : Keys \rightarrow Keys$

- Keys are independent (No equations like $K = K' + 1$ for two different keys $K, K' \in Keys$).
- Keys which are public can be used for encryption and verifying signatures.
- Keys which are assumed to be secret are used for decryption and signing.
- Every key is either an encryption or decryption key (asymmetric), or both if k is satisfying $k^{-1} = k$ (symmetric).
- The numbers of symmetric and asymmetric keys are both infinite.

Cryptography: Keys, Var, Data

- *Var* is a infinite set of variables.
- *Data* is a infinite set of data values.
- *Keys*, *Var* and *Data* are mutually disjoint.
- *Data* contains the names $UMNames \cup MsgNm$.
- *Data* may also contain nonces and other secrets.

Cryptography: Quotient of a term algebra

We recall that a term algebra generated by a set of elements and operations is the set of terms formed by applying the operations to the elements.

A quotient of a term algebra under a given set of equations is derived from the term algebra by imposing these equations, and those that can by derived from them, on the terms.

Then the algebra of cryptographic expressions *Exp* is the quotient of the term algebra generated from the set:

$$Var \cup Keys \cup Data$$

Cryptographic Expressions (1/3)

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

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

Cryptographic Expressions (2/3)

Equations:

- $\forall E,K.Dec_K^{-1}(\{E\}_K) = E$
- $\forall E,K.Ext_K(Sign_K^{-1}(E)) = E$
- $\forall E_1,E_2.head(E_1::E_2) = E_1$
- $\forall E_1,E_2.tail(E_1::E_2) = E_2$
- $\forall E_1,E_2,E_3.E_1::E_2::E_3 = E_1::(E_2::E_3)$

Cryptographic Expressions (3/3)

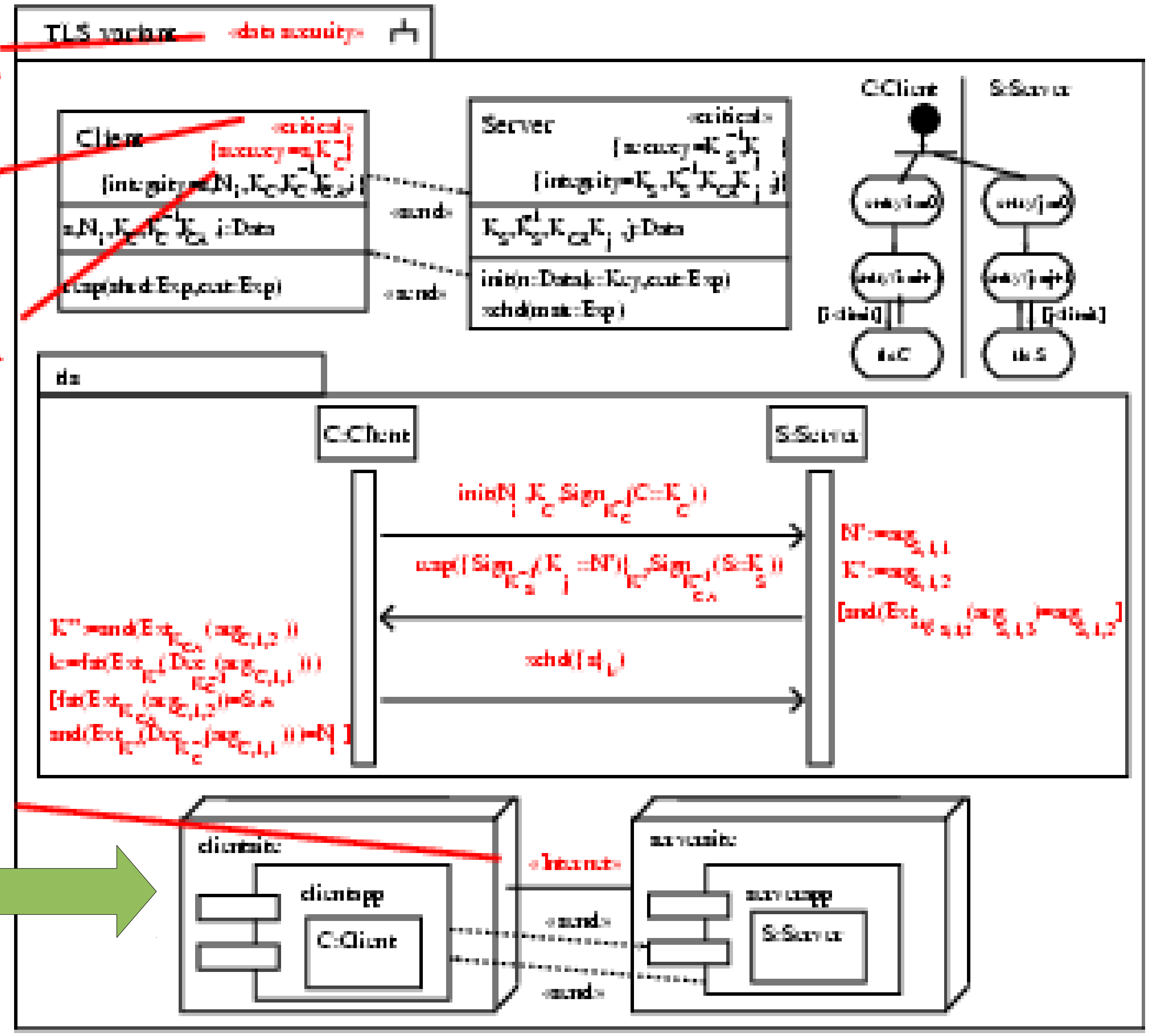
- For each $E \in Exp$, we use the following abbreviations:
  - $fst(E) =^{def} head(E)$
  - $snd(E) =^{def} head(tail(E))$
  - $thd(E) =^{def} head(tail(tail(E)))$
- We can include further crypto-specific primitives and laws (XOR, …).
- We use this abstract model of cryptographic algorithms which abstracts away the details on the level of bit sequence, in order to keep the mechanical analysis feasible.
- Based on this formalization of cryptographic operations, important conditions on security-critical data (freshness, secrecy, integrity, authenticity) can then be formulated at the level of UML diagrams in a mathematically precise way.

Example: Variant of TLS

«data security»

«critical»

$\{\text{secrecy} = \{s, K_C^{-1}\}\}$



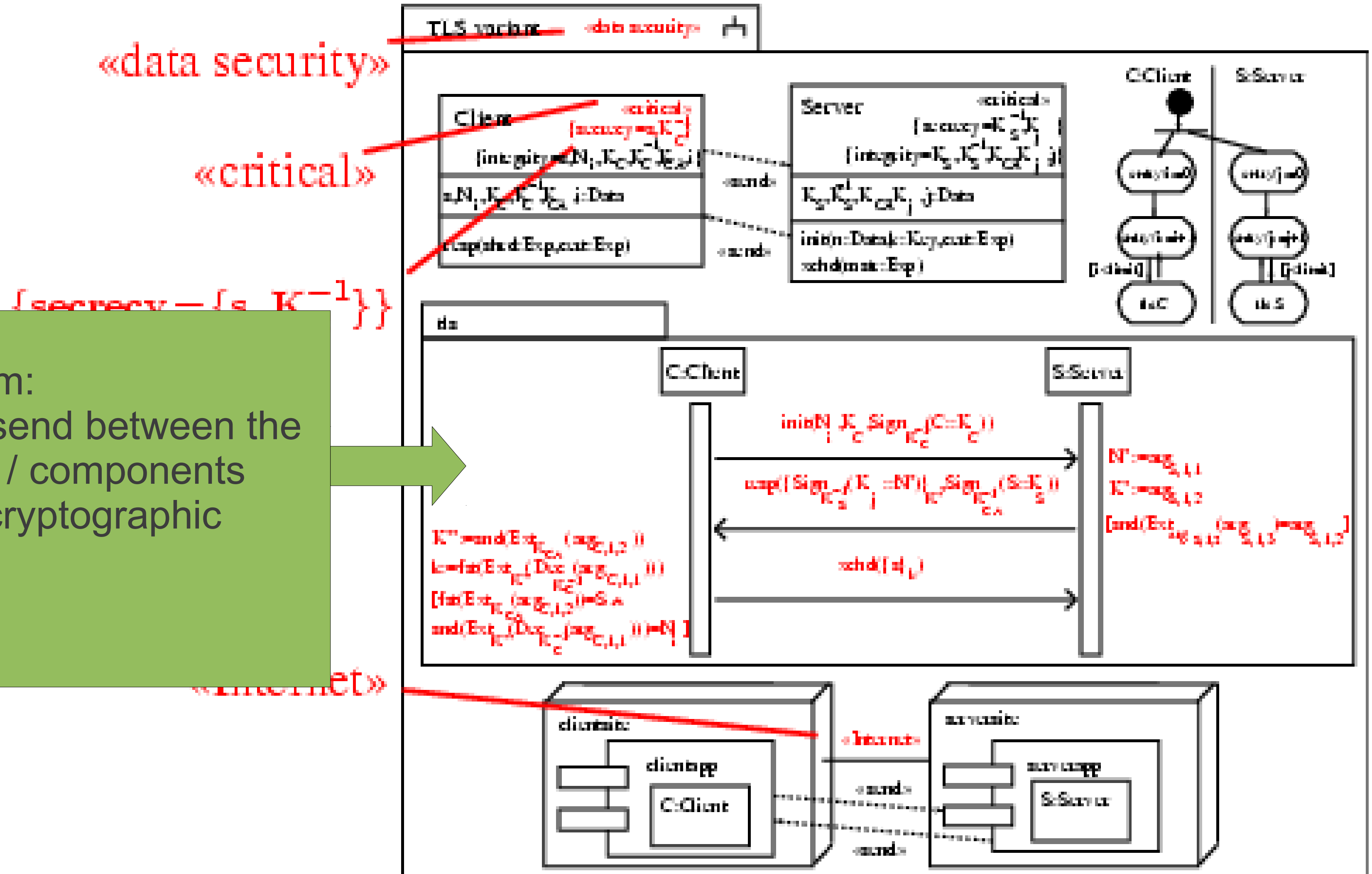
Deployment diagram:
- Information about communication links
- Here: Communication via internet
- The default attacker can perform these actions:

  $\text{Threats}_{\text{default}}(\text{Internet}) \in \{\text{delete}, \text{read}, \text{insert}, \text{access}\}$

Variant of TLS (INFOCOM`99)

Example: Variant of TLS

Sequence diagram:
- The messages send between the different objects / components
- Containing the cryptographic expressions
- e.g. xchd($\{s_i\}_k$)

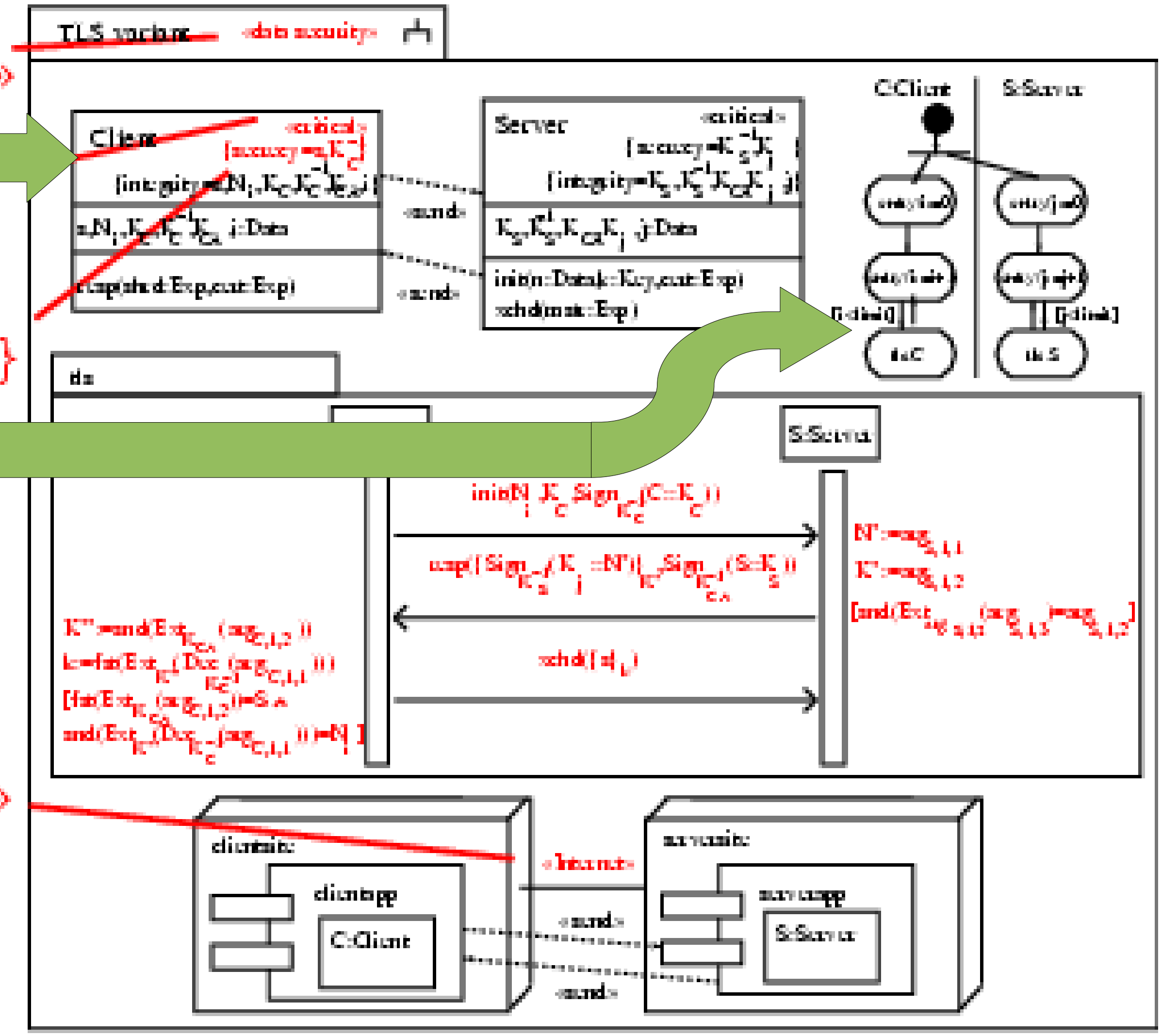


Variant of TLS (INFOCOM`99)

Example: Variant of TLS

Class diagram:
- Information about data variables and operations

Activity diagram:
- Coordinates the activities of the different objects / components



Variant of TLS (INFOCOM`99)

Security Analysis

The modular UML semantics allows a rather natural modelling of potential adversary behaviour.

- We can model specific types of adversaries that can attack different parts of the system in a specified way.
  - e.g. an attacker of type *insider* may be able to intercept the communication links in a company-wide local area network.
- We model the actual behaviour of the adversary by defining a class of UML Machines that can access the communication links of the system in a specific way.
- To evaluate the security of the system with respect to the given type of adversary, we consider the joint execution of the system with any UML Machine in the class.
- This way of reasoning allows an intuitive formulation of many security properties.

Concrete threats (1/2)

- The idea is that *$Threats_A(s)$* specifies the threat scenario associated with an adversary type *A* against a component or link stereotyped *s*.
  - On the one hand, the threat scenario determines, which data the adversary can obtain by accessing components.
  - On the other hand, it determines, which actions the adversary is permitted by the threat scenario to apply to the concerned links.
- From the abstract threats we derive the more basic concrete threats used for modelling and analysing the possible adversary behaviour.
- To analyse a UML subsystem specification *S* against a adversary of type *A*, we need to define the set *$threats^S_A(x)$* of concrete threats.

Concrete threats (2/2)

$threats^S_A(x)$ is the smallest set satisfying the following conditions:

- x is a link or a node in a deployment diagram.
- If each node n containing* x carries a stereotype s_n with $access \in Threats_A(s_n)$ then:
  - For every stereotype s attached to x, we have $Threats_A(s) \subseteq threats^S_A(x)$
  - If x is a link connected to a node that carries a stereotype t with $access \in Threats_A(t)$ then $\{delete, read, insert\} \subseteq threats^S_A(x)$

(nodes and subsystems may be nested one in another)*

Adversary machine (1/5)

- Now we can model the actual behaviour of an adversary of type A as a "type A adversary machine".
- This is a UML machine* with the following data: (* see this slide)
  - A set of states $State$ with a control state $control \in State$.
  - A set of current adversary knowledge $K_A \subseteq Exp$.
  - For each possible control state $c \in State$ and set of knowledge $K \subseteq Exp$:
    - A set $Delete_{c,K}$ which may contain the name of any link l, with $delete \in threats^S_A(l)$.
    - A set $Insert_{c,K}$ which may contain any pair (l,E) where l is the name of a link with $insert \in threats_A^S(l)$, and $E \in K$.
    - A set $newState_{c,k} \subseteq State$ of states.

Adversary machine (2/5)

The machine is executed iteratively.

- from a specified initial state $control := control^0$.
- with an initial adversary knowledge $K := K^0_A$.

Each iteration proceeds with the following steps:

- The contents of all link queues belonging to a link l with $read \in threats^S_A(l)$ are added to K.
- The content of any link queue belonging to a link $l \in Delete_{control,K}$ is mapped to $\emptyset$.
- The content of any link queue belonging to a link l is enlarged with all expressions E where $(l,E) \in Insert_{control,K}$.
- The next control state is chosen non-deterministically from the set $newState_{control,K}$.

Adversary machine (3/5)

- The set K^0_A of initial knowledge is defined to be the algebra of expressions generated by the sets K^a_A and K^p_A.
- K^a_A is the set of accessible knowledge:

  Contains all data values v given in the UML specification under consideration for which each node n containing v carries a stereotype s_n with $access \in Threats_A(s_n)$.
- K^p_A is the set of previous knowledge:

  Can be used to give the adversary access to additional data supposed to be known before start of the execution of the system, such as public keys.

Adversary machine (4/5)

- An adversary *A* is able to remove all values sent over the link *l*, represented by $delete_l \in threats^S_A(l)$.
- *A* is not able to selectively remove a value *e* with the known meaning from *l*.
- Example: The messages sent over the Internet within a virtual private network are encrypted. Thus, an adversary who is unable to break the encryption may be able to delete all messages indiscriminatly, but not a single message whose meaning would be known to the adversary.

Adversary machine (5/5)

Regarding to the slide UML machine (3/3):

- A subsystem *T* is a black-box refinement in presence of an adversary of type *A* of a subsystem *S* if every observable input/output behaviour of an execution of *T* in presence of an adversary of type *A* is also an input/output behaviour of an excution of *S* in the presence of an adversary of type *A*.
- T is a delayed black-box refinement in the presence of an adversary of type *A* of a subsystem *S*, except that delays may be introduced in *T*.

Behavioural interpretation: Security Evaluation

To evaluate the security of the system with respect to the given type of adversary, we then define the "execution of the subsystem S in the presence of an adversary of type A" as the UML Machine *$[[S]]_A$* by extending the definition of *[[S]]* on the slide UML machine (2/3).

1. A multi-set of input events received *(incoming messages).*
2. The events are distributed to the subcomponents.
3. The output messages from the subcomponents are distributed.
4. The most general type A adversary machine is applied to the link queues as detailed on the previous slides.

Important Security Properties

- Now we can specify the important security properties of
  - secrecy
  - integrity
  - authenticity
  - freshness

  by following the standard approach of Dolev, Yao (1983).
- As we remember from this slide, it defines security requirements in an intuitive way by incorporating the attacker model.
- We will also see how to define secure information flow requirements.

Secrecy (1/4)

Definition of secrecy:

- Idea: A system specification preserves the secrecy of a piece of data *d* if the system never sends out any information from which *d* could be derived by the adversary.
- *d* is leaked if there is an adversary of a given type that does not initially know *d* and an input sequence to the system such that after the execution of the system, the adversary knows *d* (as defined on previous slides).
- Otherwise, *d* is said to be kept secret.

Secrecy (2/4)

Formalization of secrecy:

- We say that a UML subsystem *S* preserves the secrecy of an expression *E* from adversaries of type *A* if *E* does not appear in the knowledge set *K* of *A* during any execution of $[[S]]_A$.
- *S* preserves the secrecy of a variable *v* from adversaries of type *A* if for every expression *E* which is a value of the variable *v* at any point, *S* preserves the secrecy of *E* from adversaries of type *A*.

Secrecy (3/4)

- Note that, by construction of the adversary knowledge (see this slide), this definition takes into account the fact that he adversary may break up expressions to access a secret subexpression.
- This definition is especially convenient to verify if one can give an upper bound for the set of knowledge *K*, which is often possible when the security-relevant part of the specification of the system S is given as a sequence of commands of the form:
  - *await event e*
  - *check condition g*
  - *output event e'*

Secrecy (4/4)

Examples:

- The system that sends the expression $\{m\}_K::K \in Exp$ over an unprotected Internet link does not preserve the secrecy of m or K against attackers eavesdropping on the Internet, but the system that sends $\{m\}_K$ and nothing else does, assuming that it preserves the secrecy of K against attackers eavesdropping on the Internet.
- A system S that receives a key K encrypted with the public key of S over a dedicated communication link and sends back $\{m\}_K$ over the link does not preserve the secrecy of m against attackers eavesdropping on and inserting messages on the link, but does so against attackers that cannot insert messages to a link.

Integrity (1/2)

Definition of integrity:

- If during the execution of the considered system, a system variable is assigned a value different from the ones it is supposed to be, then the adversary must have caused this variable to contain the value. The integrity of the variable is violated.
- A system preserves the integrity of a variable if there is no adversary such that at some point during the execution of the system in presence of the adversary, the variable has a value different from the ones it should have.

Integrity (2/2)

Formalization of integrity:

- Give a set $E \subseteq Exp$ of acceptable expressions:
  - A subsystem S preserves the integrity of an attribute a with respect to E from adversaries of type A with initial knowledge K^0 if during any execution of $[[S]]_A$, at any point the attribute a is undefined or evaluates to an element of E.
  - If $E = Exp \setminus K^0$, we simply say that S preserves the integrity of an attribute a from adversaries of type A with initial knowledge K^0.

Note to Secrecy & Integrity

! Note that this formalization of secrecy resp. integrity is relatively "*coarse / simple*" in that it may not prevent implicit information flow, but is comparatively easy to verify and seems to be sufficient in practice.*

* M. Abadi. Security protocols and their properties. In F. L. Bauer and R. Steinbrüggen, editors, Foundations of Secure Computation, pages 39-60, IOS Press, Amsterdam, 2000.

Authenticity (1/3)

Definition of authenticity:

- A message has its origin at a system part if during any execution of the system, the message appears at first at that part.
- To provide authenticity then means to secure the information on the message origin.

Authenticity (2/3)

Formalization of authenticity:

- Suppose we are given attributes *a* and *o* in a subsystem *S*, where *o* is supposed to store the origin of the message stored in *a*.
- We say that *S* provides (message) authenticity of the attribute *a* with respect to its origin *o* from adversaries type *A* with initial knowledge K^0 if during any execution of $[[S]]_A$, at any point the value of the attribute *a* appeared as a subexpression first within the execution in $outQu_o$, of all output queues and link queues in *S*.

Authenticity (3/3)

Link to integrity:*

- If the identity of the sender of a message is part of the message, integrity of the message implies the possibility to authenticate the sender.
- In this situation, data integrity implies message authenticity.

* A. Menezes, P. van Oorschot, and S. Vanstone. Handbook of Applied Cryptography, FL, 1996.
D. Gollmann. Facets of security. In C. Priami, editor, Global Computing. Programming Environments, Languages, Security, and Analysis of Systems, IST/FET International Workshop, (GC 2003)

Freshness (1/2)

Freshness of a value may mean two properties*:

- Unpredictability: An attacker cannot guess what its value was.
- Newness: The value has never appeared before during the execution of the system.

Freshness in the sense of unpredictability of *data* is captured by considering a type *A* of adversary that does not include *data* in its set of previous knowledge K^p_A.

(* following a written communication by Gavin Lowe)

Freshness (2/2)

Definition of freshness (in the sense of newness):

An atomic value *data* $\in$ *(Data* $\cup$ *Keys)* in a subsystem *S* is fresh within a subsystem instance or object *D* contained in *S* if the value *data* appears in the specification *S* only in diagram parts specifying *D*, which are called the scope of data in *S*.

Secure information flow (1/3)

- This alternative way of specifying secrecy- and integrity-like requirements, which gives protection also against partial flow of information, can be more difficult to deal with, especially when handling with encryption.
- For this definition, we need to assign to each piece of system data one of two security levels:
  - High: Highly sensitive or highly trusted.
  - Low: Less sensitive or less trusted.

LEHRSTUHL 14 SOFTWARE ENGINEERING

Secure information flow (2/3)

- Given a set of messages H and a sequence m of event multi-sets, we write:
  - m^H for the sequence of event multi-sets derived from those in m by deleting all events the message names of which are not in H.
  - m_H for the sequence of event multi-sets derived from those in m by deleting all events the message names of which are in H.

Secure information flow (3/3)

Definition of secure information flow:

- A prevents down-flow with respect to H if for any two sequences i, j of event multi-sets and any two output sequences $o \in [[S]]_A(i)$ and $p \in [[S]]_A(j)$,
  $i_H = j_H$ implies $o_H = p_H$.
- A prevents up-flow with respect to H if for any two sequences i, j of event multi-sets and any two output sequences $o \in [[S]]_A(i)$ and $p \in [[S]]_A(j)$,
  $i^H = j^H$ implies $o^H = p^H$.

Summary

- Formal Semantics
  - Notation
  - UML machine
- Cryptography
  - Quotient of a term algebra
  - Expressions
- Security Analysis
  - Adversary machine
- Important Security Properties
  - Secrecy
  - Integrity
  - Authenticity
  - Freshness
  - Secure information flow