Formal Security Analysis

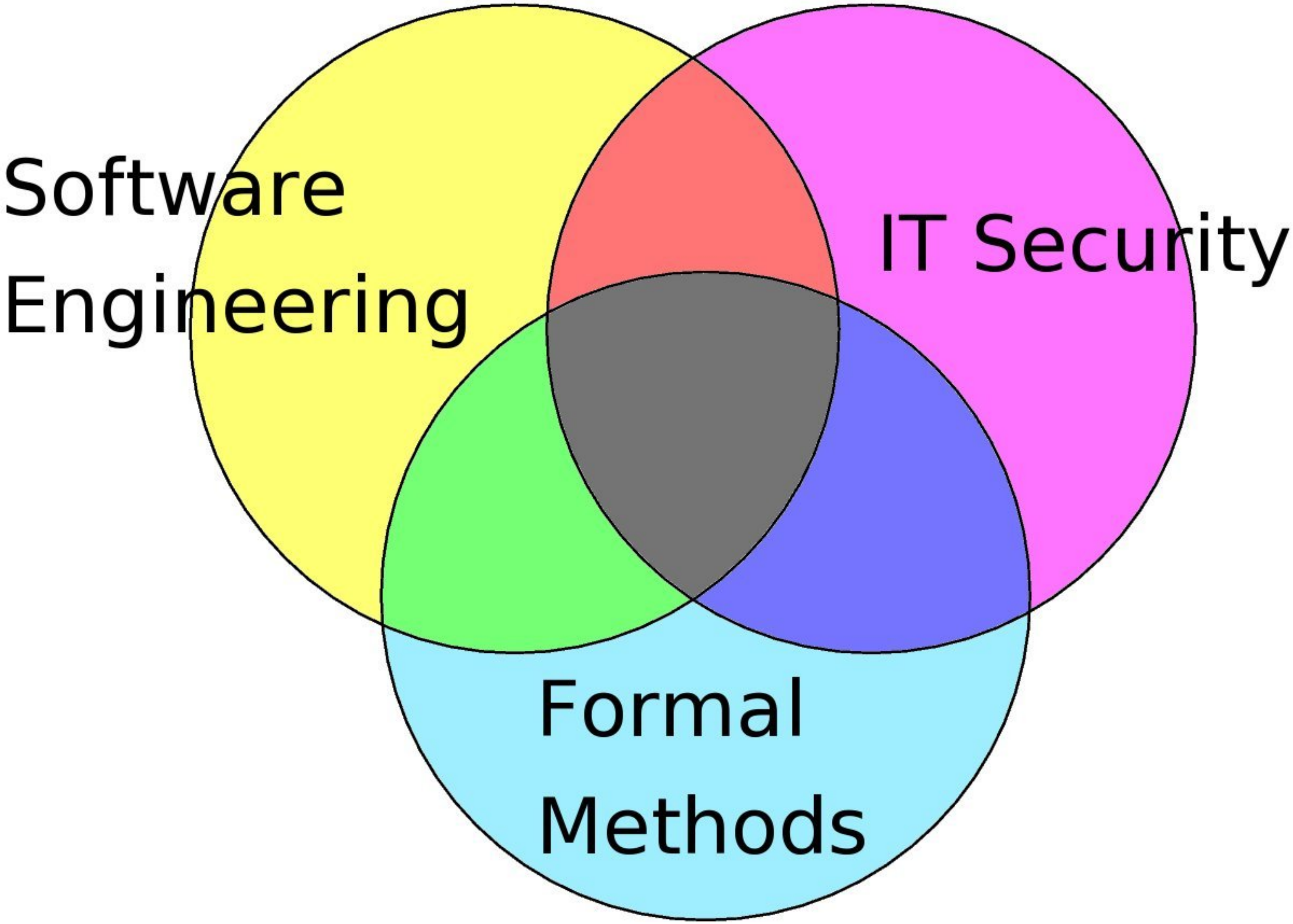
Dr. David von Oheimb

`David.von.Oheimb@siemens.com`

`ddvo.net`

Information & Communications Security
Siemens Corporate Technology
Munich, Germany

Classification



Contents

- Introduction
- Access Control

  — example: medical database
- Automata Models

  — example: Infineon SLE66
- Information Flow

  — example: Infineon SLE66
- Cryptoprotocol Analysis

  — example: Needham-Schroeder Protocol
- Evaluation & Certification

  — example: Infineon SLE88

Material

- Slides: `http://ddvo.net/teach/WS0506_FSA/`

- Books
  - ▶ Claudia Eckert: *IT-Sicherheit*. Oldenbourg, 3rd ed. 2004.
  - ▶ Matt Bishop: *Introduction to Computer Security*. Add.-Wes., 2004.
  - ▶ Dieter Gollmann: *Computer Security*. Wiley, 2000.
  - ▶ US Department of Defense. *DoD Trusted Computer System Evaluation Criteria (The Orange Book)*, DOD 5200.28.STD, 1985.

- Articles
  - ▶ Heiko Mantel, Werner Stephan, Markus Ullmann, and Roland Vogt: *Leitfaden für die Erstellung und Prüfung formaler Sicherheitsmodelle im Rahmen von ITSEC und Common Criteria.* Bundesamt für Sicherheit in der Informationstechnik (BSI), 2004

Papers

- D. Elliot Bell and Leonard J. La Padula: *Secure Computer Systems: Unified Exposition and Multics Interpretation.* MITRE Technical Report. 2997, 1976.
- VR. S. Sandhu, E.J. Coyne, H.L. Feinstein, C.E. Youman; *Role-Based Access Control Models.* IEEE Computer 29(2): 38-47, 1996
- David von Oheimb and Volkmar Lotz: *Formal Security Analysis with Interacting State Machines.* ESORICS 2002.
- David von Oheimb, Volkmar Lotz and Georg Walter: *Analyzing SLE 88 memory management security using Interacting State Machines.* International Journal of Information Security, 2005.
- John Rushby: *Noninterference, Transitivity, and Channel-Control Security Policies.* SRI International Technical Report CS-92-02, 1992.
- David von Oheimb: *Information flow control revisited: Noninfluence = Noninterference + Nonleakage.* ESORICS 2004.

Contents

- **Introduction**
- Access Control
- Automata Models
- Information Flow
- Cryptoprotocol Analysis
- Evaluation & Certification

Outline

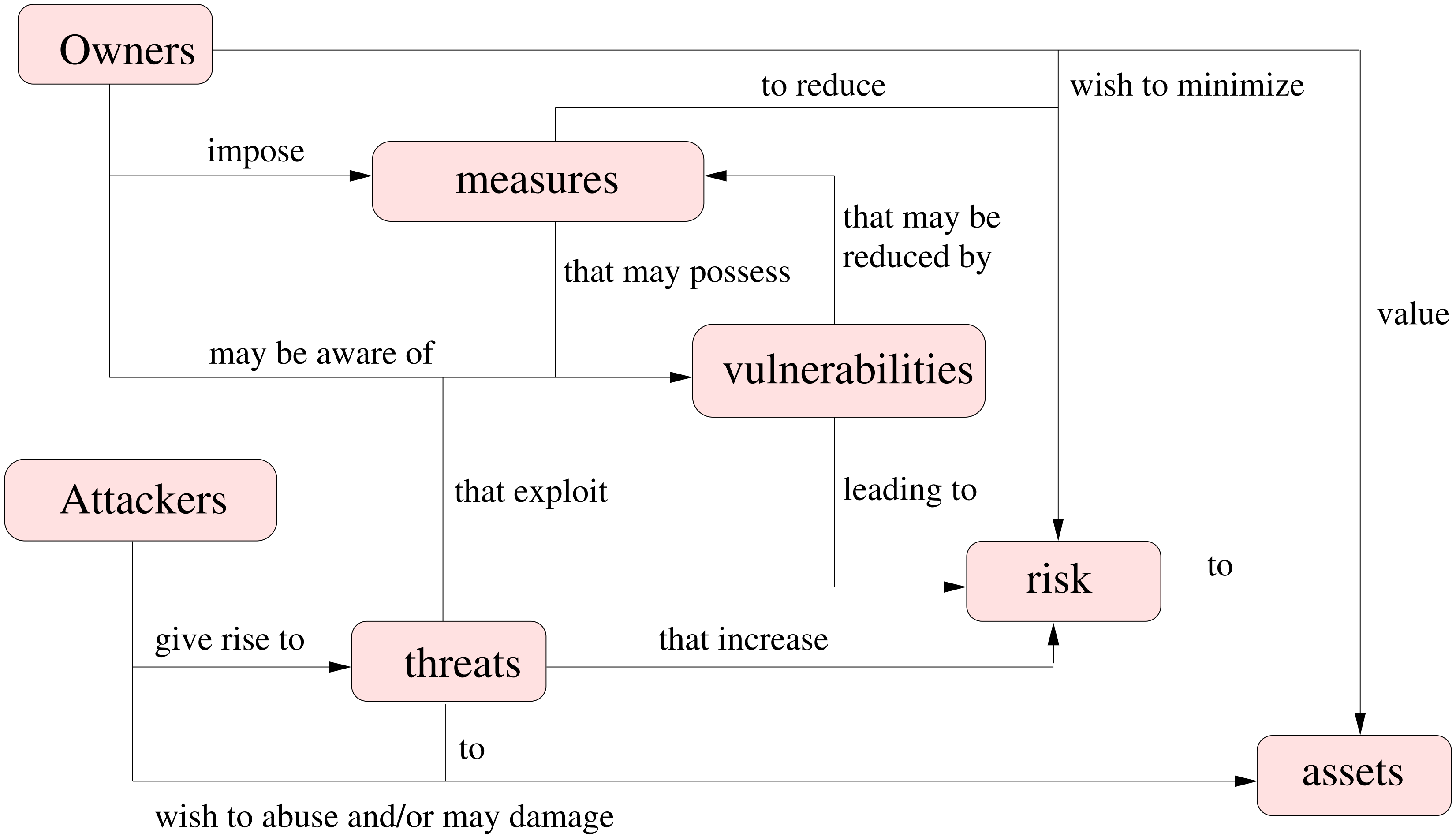
☞ **What is Information Security?**

- Goals, Threats, and Mechanisms
- Security Policies
- Security Models
- Conclusions on Security

Information Security

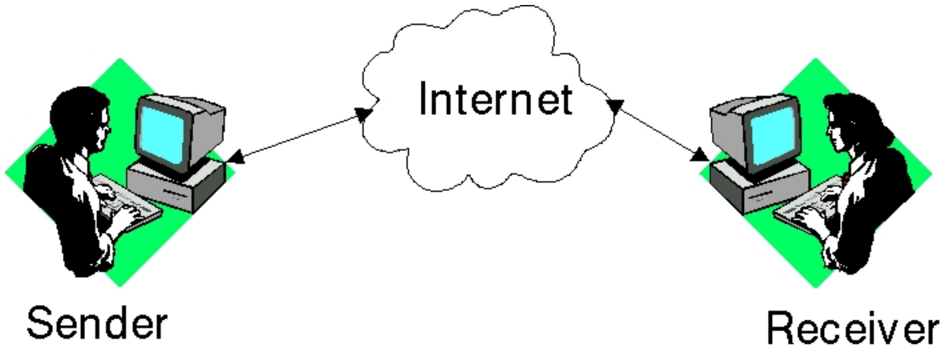
- IT/Computer security deals with the prevention, or at least detection, of unauthorized actions or possession by users of a computer system.
  - ▶ Authorization is central to definition.
  - ▶ Sensible only relative to a security policy, stating who may perform which actions.
- Information security is even more general. It deals with information independent of computer systems.
- Constitutes a basic right: protection of self (privacy, ...).
- Complements safety: prevent damage through errors or malfunction

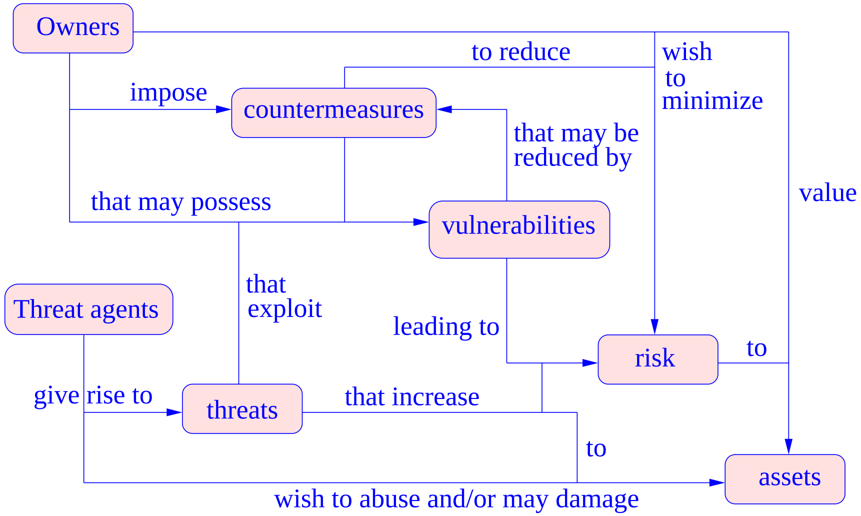
Security according to Common Criteria



- Classification depicts fundamental concepts and interrelationships.
- Policy (here implicit) defines authorized actions on assets, i.e., what constitutes legal use (or abuse/damage, respectively).

Example: email



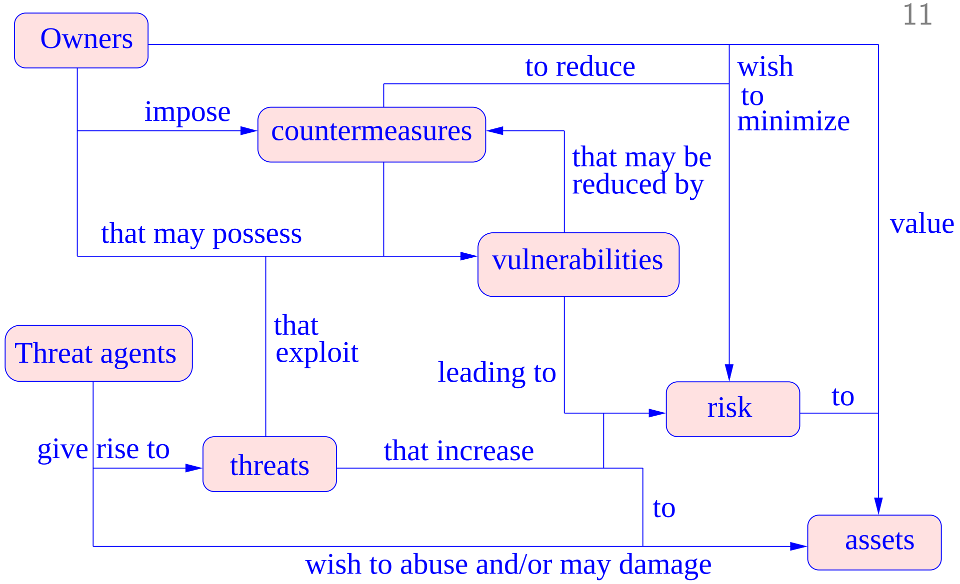


Assets: Mail

Threats:

- Who sent the mail?
- Has it been received?
- Was the mail read by others during transport?
- Was the mail modified during transport?

Example: e-voting



Assets: Data, e.g., individual votes, voter identity, results, etc.

Threats: (sample)

- How will the system ensure that only registered voters vote?
- How will it ensure that each voter can only vote once?
- How does the system ensure that votes are not later changed and are correctly tabulated?
- How are votes kept private and identities secret?
- System availability? Functional correctness?

E-voting — Swiss requirements

Elektronische Wahl- und Abstimmungssysteme und die elektronische Sammlung von Unterschriften müssen unter allen Umständen sicher funktionieren und vor möglichen Gefahren und Einwirkungen von außen geschützt sein. Sie müssen dabei ebenso viel Sicherheit bieten wie die gegenwärtig geltenden Systeme. Das bedeutet allerdings nicht hundertprozentige Sicherheit. Auch das geltende Abstimmungssystem kennt Schwachstellen.

Requirements in practice are difficult to formulate precisely.
This is part of the challenge in designing secure systems.

Security as a Software Engineering Problem

Situation: security loopholes in IT systems will be actively exploited
— in this sense even worse than safety problems!

Goal: achieve absence of attacks by absence of vulnerabilities
— and convince contractors/customers/users of this!

Problem: IT systems are very complex, security flaws hard to find.
Security cannot be added on, but must be co-designed with the system.

Remedy: address security in all development phases.
Reviews supported by formal security modeling/analysis.

During ...

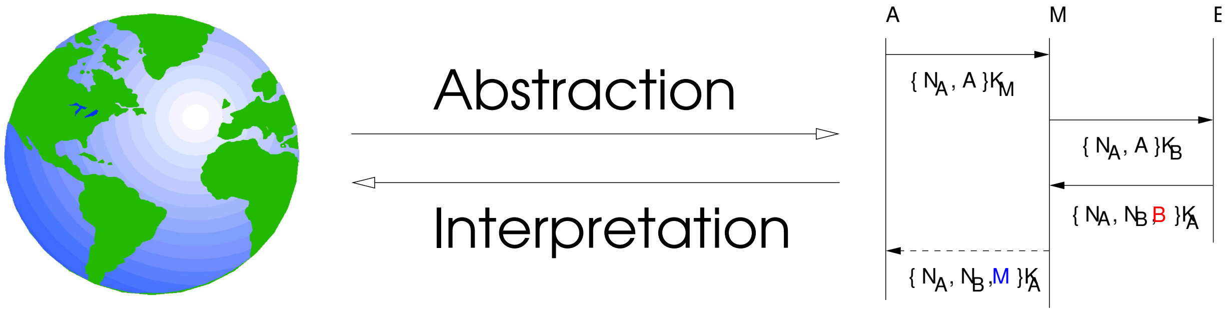
- **requirements analysis:** helps understanding the security issues
- **design, documentation:** helps improving the quality of specifications
- **implementation:** acts as correctness reference for testing/verification

What are Formal Methods?

- A language is formal if it has a well-defined syntax and semantics.
  Examples: Predicate logic, automata, $\lambda-$calculus, process algebra, ...
- A model is formal if it is specified with a formal language.
  Example:
  $$\forall x.\, bird(x) \rightarrow flies(x) \qquad bird(tweety)$$
- A proof is formal if it is done using a deductive system
  (i.e., a set of precise rules governing each proof step).
  Examples: Tableau calculus, axiomatic calculus, term rewriting, ...
- A formal proof is machine-assisted if
  it is performed, or at least checked, by an IT system.
  Examples: OFMC (model checker), Isabelle (theorem prover)

Why are Formal Security Models so helpful?

A formal security model is an abstract formal description of a system (and its environment) that focuses on the relevant security issues.



Its advantages/goals are:

- improves understanding of security by abstraction: simplification and concentration on the essentials
- prevents ambiguities, incompleteness, and inconsistencies and thus enhances quality of specifications
- provides basis for systematic testing or even formal verification and thus validates correctness of implementations

CC: Goals & General Approach

Common Criteria

...the standard for Information Security

Goal: Gaining confidence in the security of a system

- What are the goals to be achieved?
- Are the measures employed appropriate to achieve the goals?
- Are the measures implemented correctly?

Approach: assessment (evaluation) of system security by neutral experts

- Understanding how the system's security functionality works
- Gaining evidence that security functionality is correctly implemented
- Gaining evidence that the integrity of the system is kept

Result: Successful evaluation is awarded a certificate

Outline

- What is Information Security?
☞ **Goals, Threats, and Mechanisms**
- Security Policies
- Security Models
- Conclusions on Security

Security Goals

- Goals: CIA

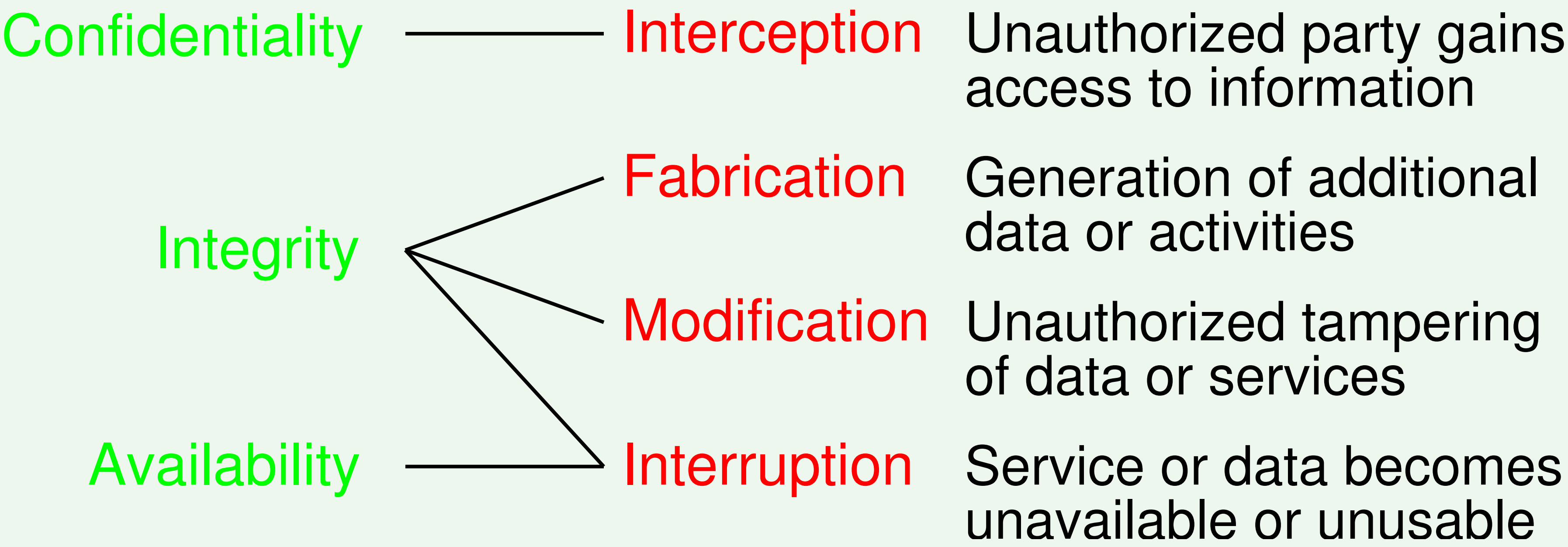
  Confidentiality: No unauthorized disclosure/leakage of information
  Integrity: No unauthorized modification of information
  Availability: No unauthorized impairment of functionality

  Note that CIA all require some form of **authorization**,
  which consists of some form of **authentication**and **access control**.

- Other goals

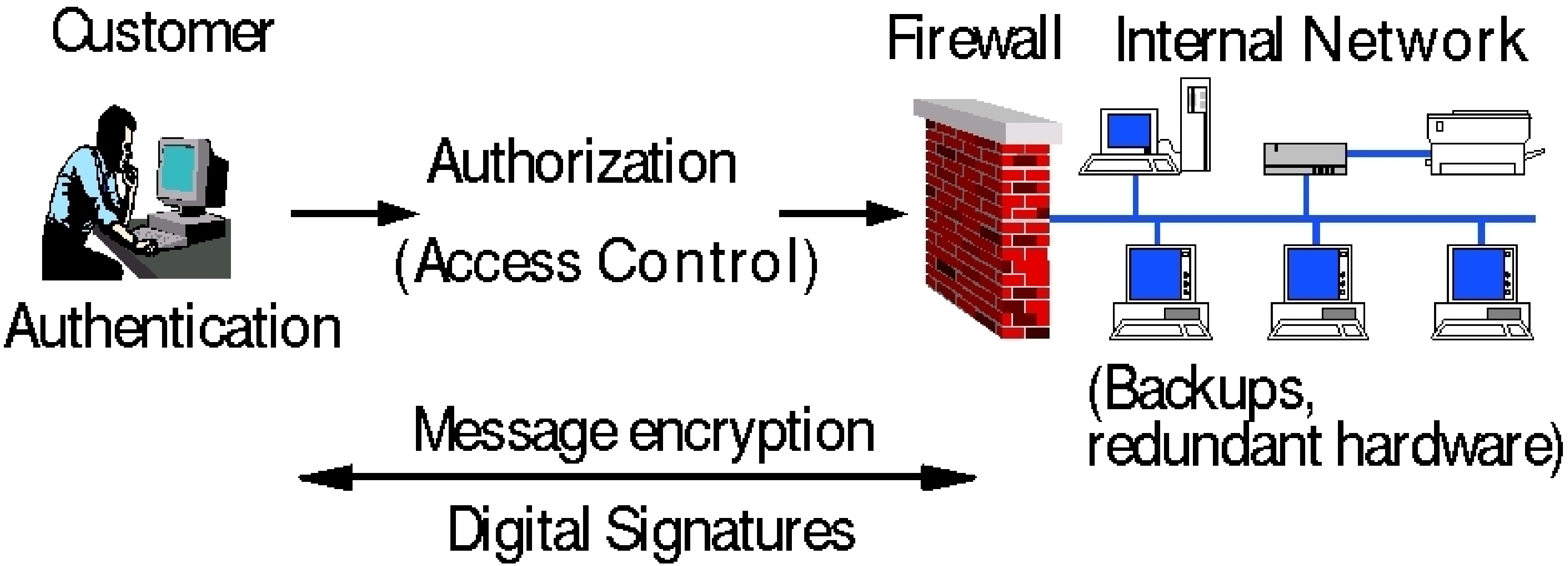
  Privacy: User data is only exposed in permitted ways.
  Nonrepudiation: One cannot deny responsibility for actions.
  Also called **accountability**
  Application specific requirements: E.g.,
  e-voting must suitably combine above!

Threats



Confidentiality	Interception	Unauthorized party gains access to information
Integrity	Fabrication	Generation of additional data or activities
	Modification	Unauthorized tampering of data or services
Availability	Interruption	Service or data becomes unavailable or unusable

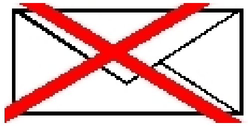
Security Mechanisms



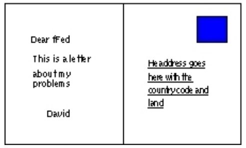
Let's consider how different mechanisms can be used to achieve goals in the face of threats, and what some of the challenges are.

Confidentiality/Privacy

Example Email is **not** a letter



but rather a post card!

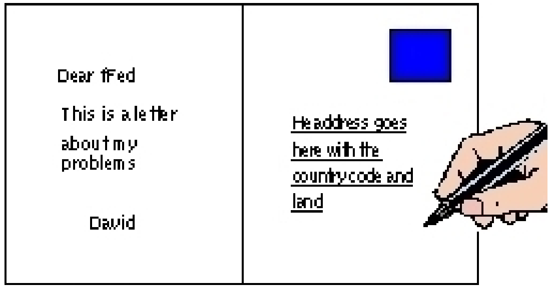


Threat Everyone can read it along the way!

Mechanism Network security, encryption, and access control

Challenges Key and policy management

Data integrity



Example Email (or forms, records, ...)

Threat Unallowed modification/falsification

Mechanism Digital signatures and/or access control

Challenges PKI and policy management

Availability

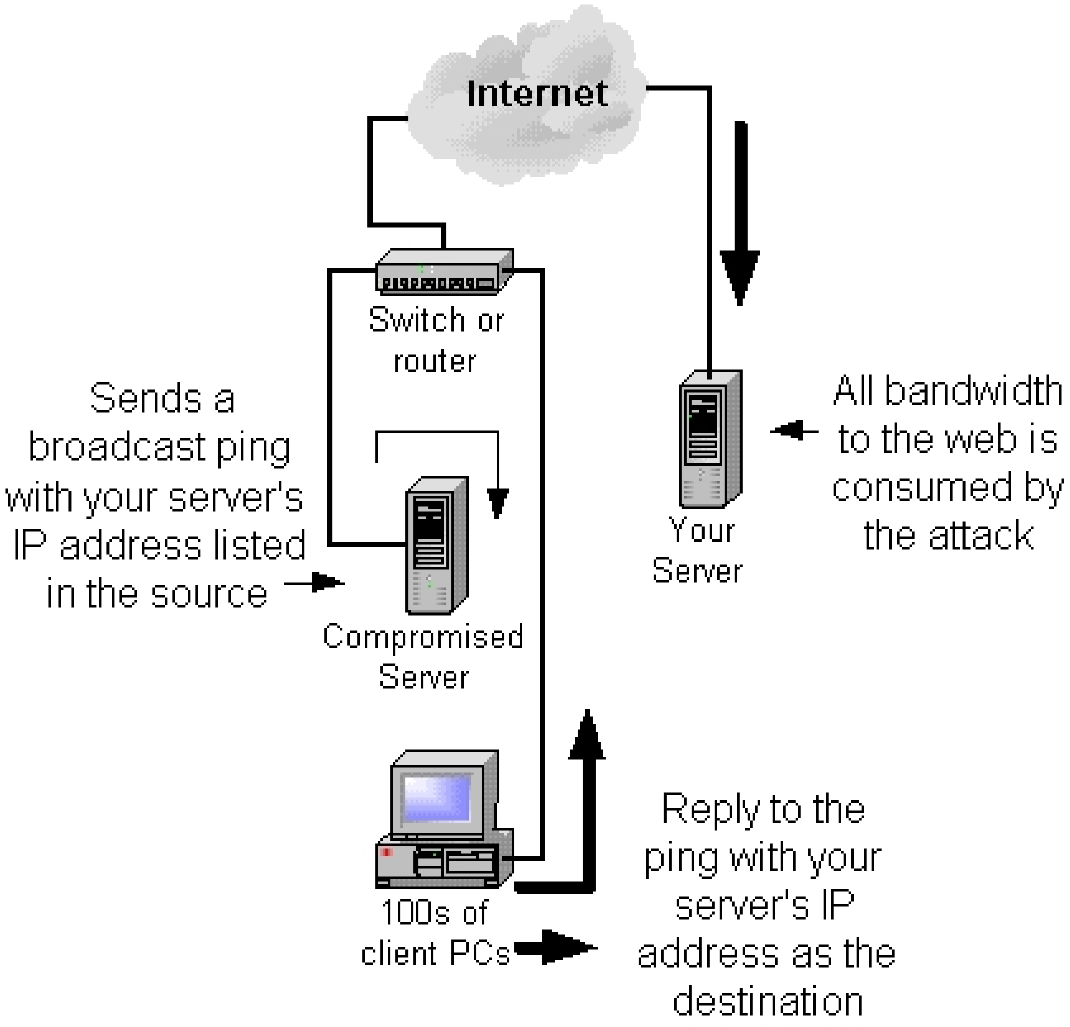
Example Communication with a server

Threats Denial of Service, break-ins, ...

Mechanism Fire-walls, virus-scanners, backups, redundant hardware, secure operating systems, etc.

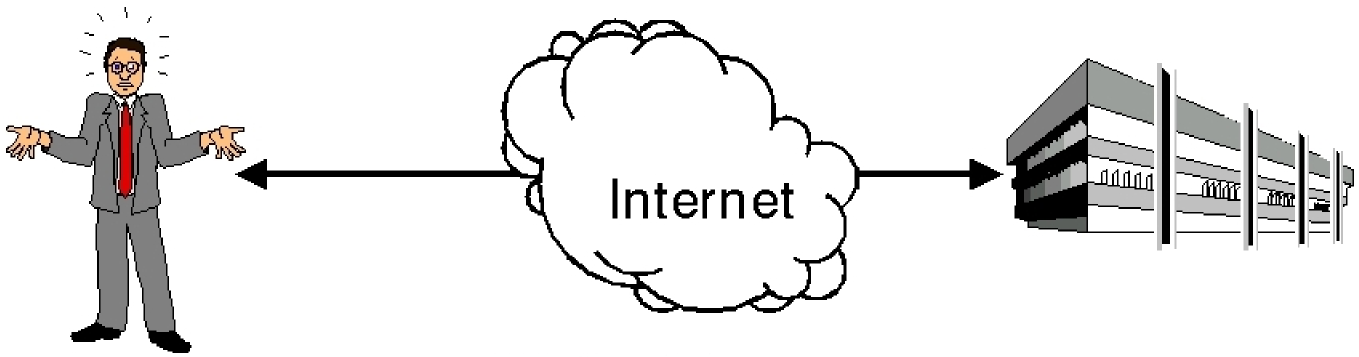
Challenges Difficult to cover all threats (and still have a usable system)

Also difficult to test/verify, because availability is a liveness property: “something good eventually happens”, while all others are safety properties: “something bad never happens”



Authentication: who is who?

Example



E-Mail addresses
are trivial to falsify

Threats Misuse of identity

Mechanisms

Credentials of requester: personal characteristics (biometric),
what one has (smartcard), or what one knows (password).

Processes, Data : cryptographic protocols, digital signatures, etc.

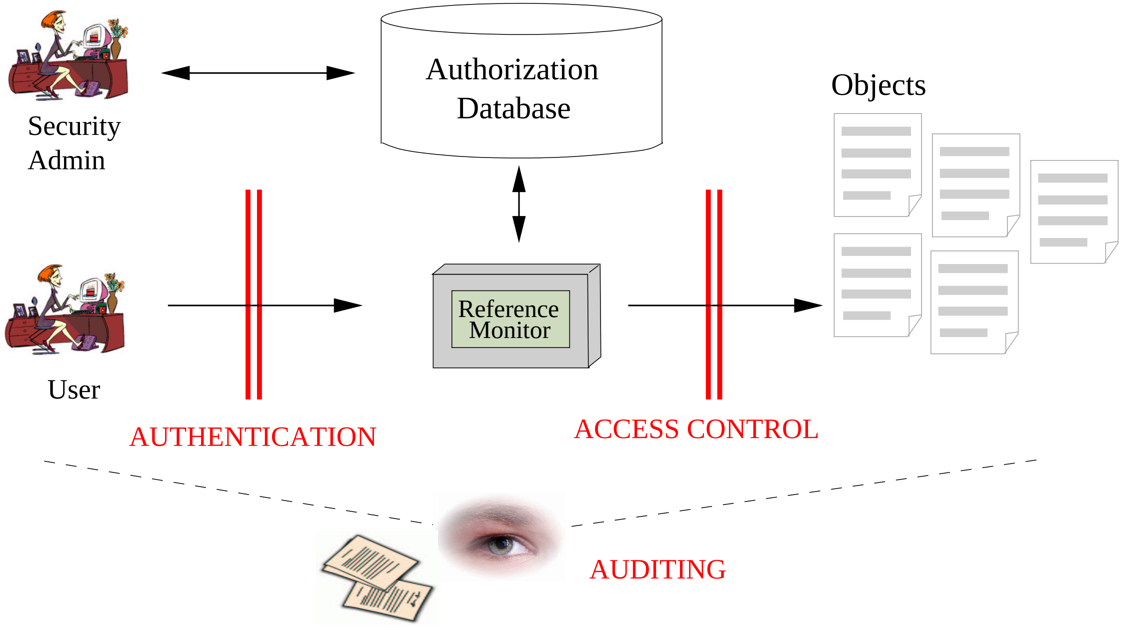
Challenges authentication hardware/mechanisms,
protocol design/analysis, PKIs

Access Control (AC): who has what permission?

Example Access to data, processes, networks, ...

Threats Unauthorized access of resources

Mechanisms Declarative and programmatic control mechanisms



Challenges Policy design, integration, and maintenance

Summary: Goals, Threats, and Mechanisms

- Standard breakdown. Important for analyzing system security relative to a policy.
- Designing adequate mechanisms is challenging.
- One must take a holistic approach to security engineering.
  - Security must be co-designed with the system, not added on.
  - One must be cognizant of the tradeoffs and costs involved.
- There are many open questions, both at the level of mechanisms and the design/integration process.

Outline

- What is Information Security?
- Goals, Threats, and Mechanisms

☞ **Security Policies**

- Security Models
- Conclusions on Security

An example: university computing

- IT security policy:

  A student has full access to information that he or she created. Students have no access to other students' information unless explicitly given. Students may access and execute a pre-defined selection of files and/or applications. ...

- Security policy describes access restrictions.

- Issues
  - How do we formalize such a policy?
  - What mechanisms would we use to enforce it?

Two more examples

- E-Banking

  A bank customer may list his account balances and recent transactions. He may transfer funds from his accounts provided his total overdraws are under 10,000. Transfers resulting in larger overdraws must be approved by his account manager. ...

  Above policy describes restrictions, where objects here include both data and processes.

- Privacy policies A clerk may only have access to personal data if this access is necessary to perform his/her current task, and only if the clerk is authorized to perform this task.
  In addition, the purpose of the task must correspond to the purposes for which the personal data was obtained or consent must be given by the data owners.

  Combines conditions with obligations on how data will be used.

Security Policies

- A security policy defines what is allowed.

  It defines those executions of a system (actions, data flow, etc.)
  that are acceptable,
  or complementarily, those that are not acceptable.

  - ► Typically defined at high level.
  - ► Typically defines a relationship between subjects and objects.
  - ► It is analogous to a set of laws.

Outline

- What is Information Security?
- Goals, Threats, and Mechanisms
- Security Policies

☞ **Security Models**

- Conclusions on Security

Security Models

- A security model is a +/- formal description of a system and a policy (or of a family of policies). Usually in terms of system state or sequences of states (traces).

  N.B.: model is overloaded in literature.
  E.g., formal policy, security mechanisms, semantic models, ...

- Security verification proves wrt. model that mechanisms enforce policy

- Models usually focus on specific characteristics of the reality/policies.

Protection state

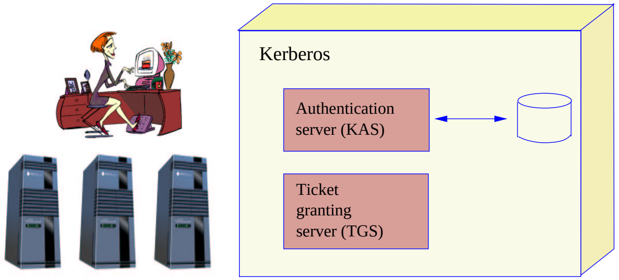
- A state of a system is the collection of all current values of memory locations, disk storage, processor registers, and other components.
- The substate addressing security is the system protection state.
- Examples of protection states

  File system: part of system state determining who is reading/writing files, access control information, etc.

  Network: e.g., packet header information (identifying protocols) and packet locations, internal firewall states, etc.

  Program: e.g., part of run-time image corresponding to program counter, call stack, memory management tables, etc.
- Abstraction: system execution described as transitions between protection states

Example 1: Kerberos



- Provides Single Sign-On mechanism in a distributed setting.
- Partitions authentication, authorization, and access control.

Security policy: expresses which users can access what servers in a realm (or cross-realm).

The policy is determined by the system administrator who registers users/servers in the database.

Protection state: Kerberos server state (e.g., policy tables), part of client state and application server state (e.g. state of protocol runs)

Example 2: security policy for proprietary data

Security policy for company X

All information on product Y is confidential: it may only be read or modified by a subgroup Z and the system administrators.

Mechanism implications

- All printouts must be securely stored or shredded.
- All computer copies must be protected (AC, cryptography, ...)
- As company X stores its backup tapes in a vault in the town bank, X must ensure that only authorized employees have access to these tapes. Hence the bank's control on access to the vault and the procedures used to transport tapes to/from the bank are considered as security mechanisms.

The security mechanisms are not only technical controls, but also procedural or operational controls.

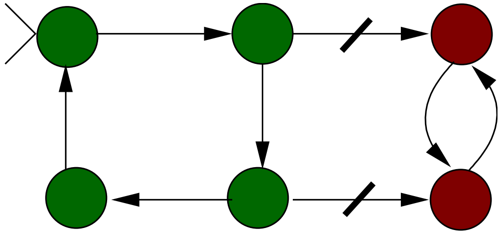
Protection state

Not just the IT state, but also existence and location of physical goods.

Protection state and security policy

- Let P be the system state space and $Q \subseteq P$ be the states in which system is authorized to reside in.
  - ▶ A state $s \in Q$ is called authorized (or secure),
  - ▶ any $s \in P \setminus Q$ is called unauthorized (or nonsecure).
- A security policy characterizes Q.

  Hence a security policy partitions the states of the system into authorized (or secure) states, and unauthorized (or nonsecure) states.
- A security mechanism prevents a system from entering $P \setminus Q$.



A secure system is a system that starts in an authorized state and cannot enter an unauthorized state.

Types of Security Models

We will consider

- Access Control models
- Automata-based models
- Information Flow models
- Cryptoprotocol models

Access Control models

- Discretionary vs. mandatory AC models.

- Various types of models:
  - Models can capture policies
    for confidentiality (Bell-LaPadula)
    or for integrity (Biba, Clark-Wilson).
  - Some models apply to static policies (Bell-LaPadula),
    others consider dynamic changes of access rights (Chinese Wall).
  - Security models can be informal (Clark-Wilson),
    semi-formal, or formal (Bell-LaPadula, Harrison-Ruzzo-Ullman).

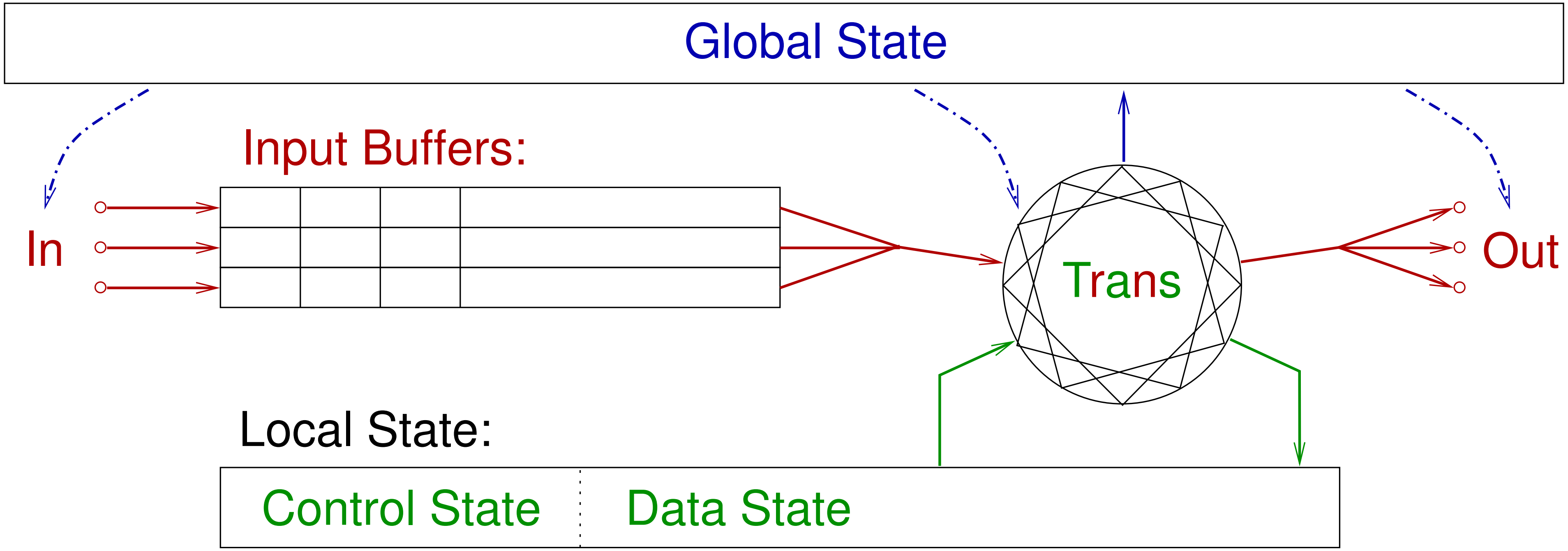
- Modern extension: *role-based access control (RBAC)*

Interacting State Machines (ISMs)

Automata with (nondeterministic) state transitions +

buffered i/o simultaneously on multiple connections

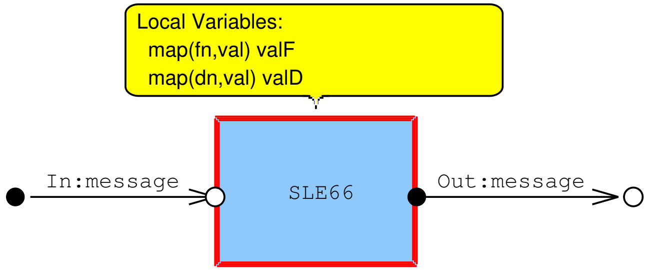
ISM system may depend on global state



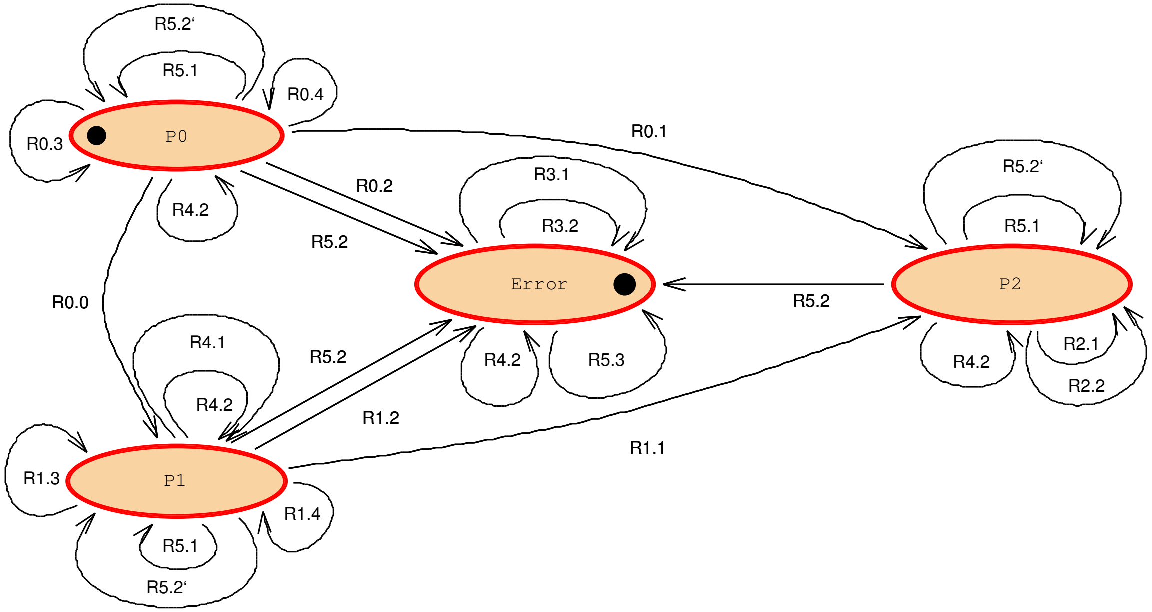
Applicable to a large variety of reactive systems

LKW Model of Infineon SLE 66 Smart Card

System Structure Diagram:

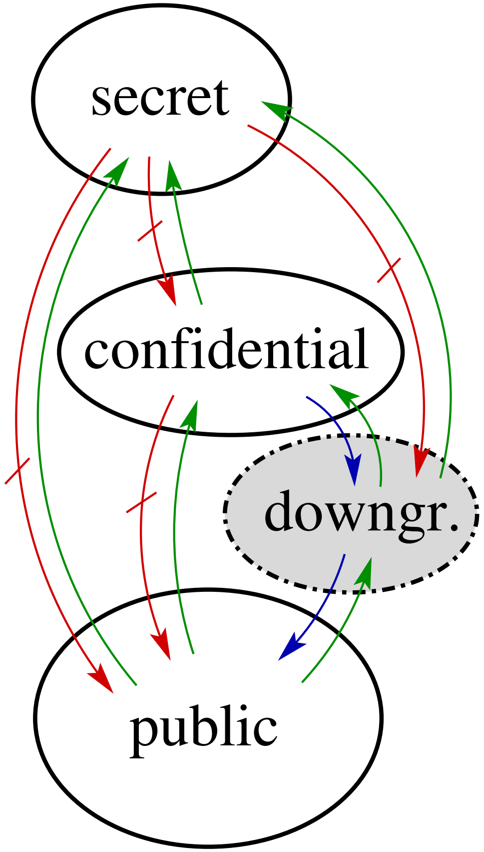


State Transition Diagram (abstracted):



First higher-level (EAL5) certification for a smart card processor!

Information Flow models



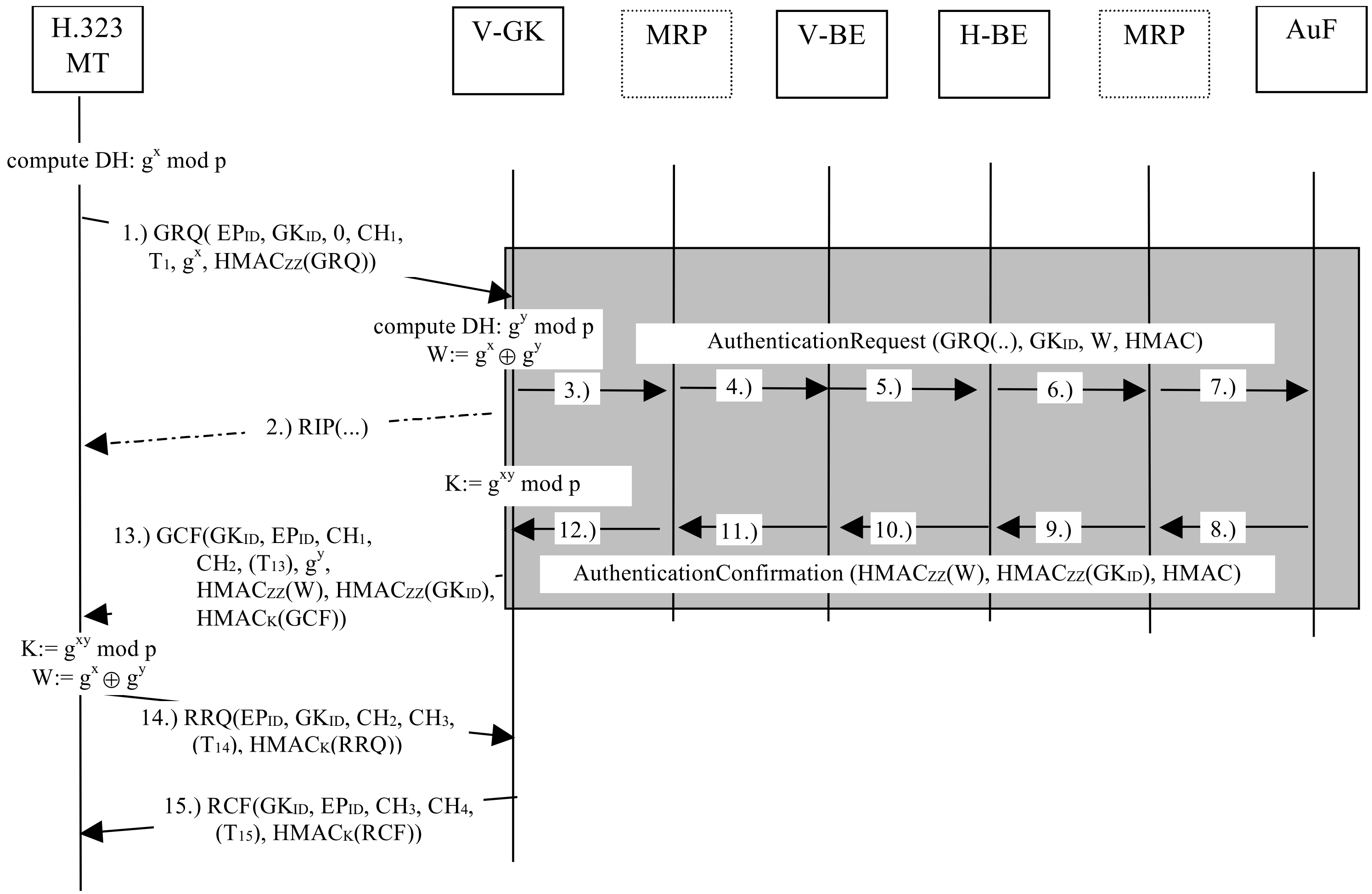
- Identify domains holding information
- Specify allowed flow between domains
- Check the observations that can be made about state and/or actions
- Consider also indirect and partial flow
- Classical model: Noninterference (Goguen & Meseguer)
- Many variants: Non-deducability, Restrictiveness, Nonleakage, ...

Cryptoprotocol models

- Describe message traffic between processes or principals

- Take cryptographic operations as perfect primitives
- Are specified with by domain-specific languages (e.g. HLPSL)
- Describe secrecy, authentication, . . . goals
- Are typically verified automatically using model-checkers

H.530: Authentication for Mobile Roaming



H.323 MT

V-GK

MRP

V-BE

H-BE

MRP

AuF

compute DH: g^x mod p

1.) GRQ(EP_{ID}, GK_{ID}, 0, CH_1, T_1, g^x, $HMAC_{ZZ}$(GRQ))

compute DH: g^y mod p
W:= $g^x \oplus g^y$

AuthenticationRequest (GRQ(..), GK_{ID}, W, HMAC)

3.) 4.) 5.) 6.) 7.)

2.) RIP(...)

K:= g^{xy} mod p

12.) 11.) 10.) 9.) 8.)

13.) GCF(GK_{ID}, EP_{ID}, CH_1, CH_2, (T_{13}), g^y, $HMAC_{ZZ}$(W), $HMAC_{ZZ}$(GK_{ID}), $HMAC_K$(GCF))

AuthenticationConfirmation ($HMAC_{ZZ}$(W), $HMAC_{ZZ}$(GK_{ID}), HMAC)

K:= g^{xy} mod p
W:= $g^x \oplus g^y$

14.) RRQ(EP_{ID}, GK_{ID}, CH_2, CH_3, (T_{14}), $HMAC_K$(RRQ))

15.) RCF(GK_{ID}, EP_{ID}, CH_3, CH_4, (T_{15}), $HMAC_K$(RCF))

Two vulnerabilities found and corrected. Solution patented.

Modeling considerations

Choice of Formalism: dependent on ...

- application domain, modeler's experience, tool availability, ...
- formalism should be simple, expressive, flexible, mature

Formality Level: should be adequate:

- the more formal, the more precise,
- but requires deeper mastering of formal methods

Abstraction Level: should be ...

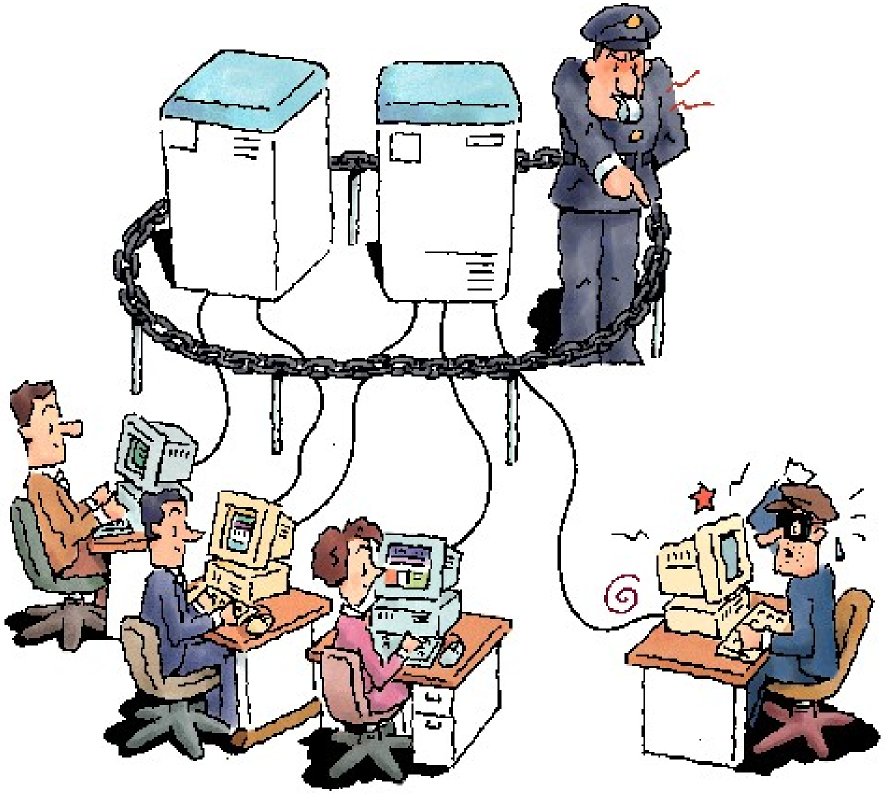
- high enough to achieve clarity
- low enough not to loose important detail

refinement allows for both high-level and detailed description

Outline

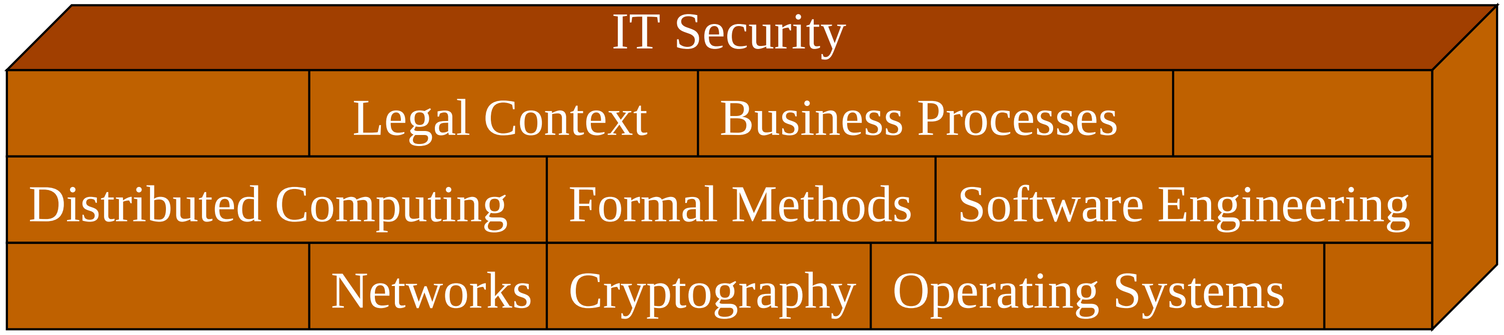
- What is Information Security?
- Goals, Threats, and Mechanisms
- Security Policies
- Security Models

☞ **Conclusions on Security**



Conclusions

- Security is an enabling technology.
- Security is a cross-section topic.



IT Security

Legal Context
Business Processes

Distributed Computing
Formal Methods
Software Engineering

Networks
Cryptography
Operating Systems

- Security is difficult.

… and therein lies the challenge, excitement, and reward!

Contents

- Introduction
- **Access Control**
- Information Flow
- Cryptoprotocol Analysis
- Evaluation & Certification

Outline

☞ **Access Control (AC)**

- Discretionary Access Control (DAC)
- Access Control Matrix Model
- Mandatory Access Control (MAC)
- Bell-LaPadula Model and Variants
- Role-Based Access Control (RBAC)

Access Control

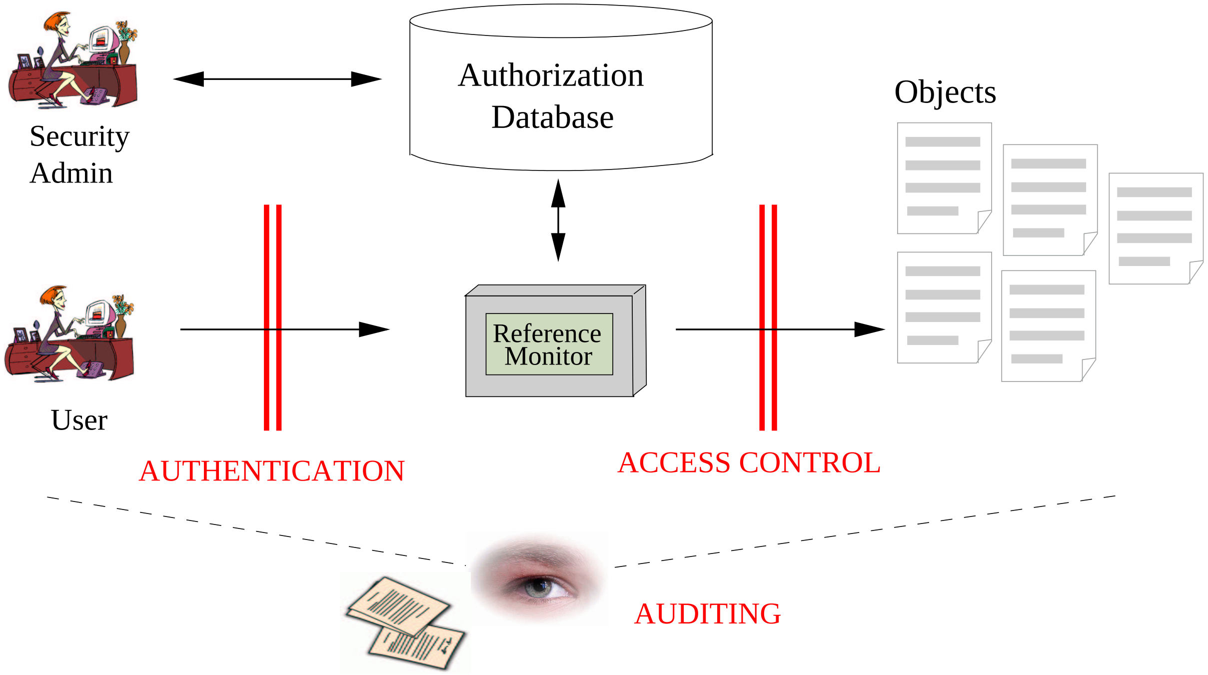
Many security policies (and mechanisms) focus on access control.

Access Control:

Protection of system resources against unauthorized access;
a process by which use of system resources is regulated
according to a security policy that determines authorized access.

certain subjects (entities, e.g. users, programs, processes)
have permissions (e.g. `rwx`)
on objects (e.g. data, programs, devices)
according to AC policies.

AC: Authorization and Auditing



Authentication establishes/verifies identity of requester.

Authorization decides whether legitimate (authenticated) requester is allowed to perform the requested action.

Auditing gathers data to discover violations or diagnose their cause.

AC Policies vs. AC Mechanisms

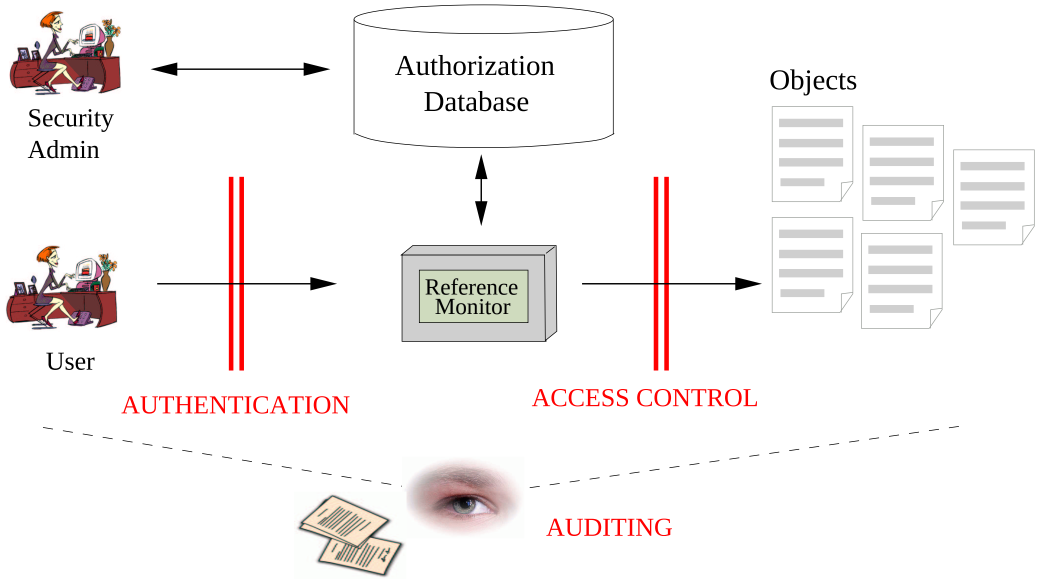
- Policy: specifies (un-)authorized accesses of a system and how access decisions are determined.
  - Discretionary AC.
  - Mandatory AC.
  - Role-based AC.

- Mechanism (structure): implements or enforces a policy.
  - Access matrix.
  - AC list (ACL).
  - Capability list.

This distinction allows for abstraction and independence.

Access Control — Typical Mechanisms

- System knows who the user is, i.e. authentication is done.
- Access requests pass through a gatekeeper (“reference monitor”).



OS must be designed that way: MMU, file system, firewall, . . .
OS-level AC provides basis for application-specific mechanisms.

- We will now look at several different access control models.

Outline

- Access Control (AC)

☞ **Discretionary Access Control (DAC)**

- Access Control Matrix Model

- Mandatory Access Control (MAC)

- Bell-LaPadula Model and Variants

- Role-Based Access Control (RBAC)

Discretionary Access Control (DAC)

- Premise: users are owners of resources and are responsible for controling their access.
- The owner of information or a resource is able to change its permission at his or her discretion. Owners can usually also transfer ownership of information to other users.
- Flexible, but open to mistakes, negligence, or abuse.
  - ► Requires that all system users understand and respect security policy and understand AC mechanisms.
  - ► Abuse, e.g. Trojan horses may that trick users into transferring rights.
- Dissemination of information is not controlled:

  a user who is able to read data can pass it to other users who are not authorized to read it without cognizance of the owner.

Types of DAC policies

- Closed DAC policies: authorization must be explicitly specified, since the default decision of reference monitor is denial.
- Open DAC policies: specify denials instead of permissions (default decision is access).
- Combination of positive and negative authorizations possible (but quite complex).

  Example: Deny in Windows XP

A DAC example: Unix

- Unix provides a mechanism suitable for a restricted class of DAC policies.
  - Controls access per object using permission scheme *owner/group/other*.
  - Permission bits assigned to objects by their owners.

```
-rw-r--r--   1 luca  softech  56643 Dec 8  17:19 file1.tex
drwxrwxrwt  26 root  root      4096 Dec 9  22:27 /tmp/
-rwsr-xr-x   1 root  shadow   80036 Oct 3  11:08 /usr/bin/passwd*
```

- Not all policies can be directly mapped onto this mechanism.

  How would we express that a patient can read his medical records at a hospital? Who owns the records? In which group is the patient?

- Supports limited delegation of rights using suid ("set user identification") [or sgid].
  - Executor takes on owner's user [or group] identity during execution.
  - Example: normal users "upgraded" to root privileges to change their passwords in the password file.
  - Open to abuse and the cause of many security holes.

Outline

- Access Control (AC)

☞ **Access Control Matrix Model**

- Discretionary Access Control (DAC)

- Mandatory Access Control (MAC)

- Bell-LaPadula Model and Variants

- Role-Based Access Control (RBAC)

Access Control Matrix Model

- Simple framework for describing a protection system by describing the permissions of subjects on objects.

  Subjects: users, processes, agents, groups, ...
  Objects: files, memory banks, other processes, ...
  Permissions (or rights): read, write, execute, print, ...

- Policy is a finite relation $\mathcal{P} \subseteq \text{Subjects} \times \text{Objects} \times \text{Permissions}$

Objects

	File 1	File 2	File 3	File 4	Account 1	Account 2
Alice	Own R W		W X		Inquiry Credit	
Bob	R	Own R W	W	R	Inquiry Debit	Inquiry Credit
Charlie	R W	R		Own R X		Inquiry Debit

Subjects

▻ Permission

given as a matrix.

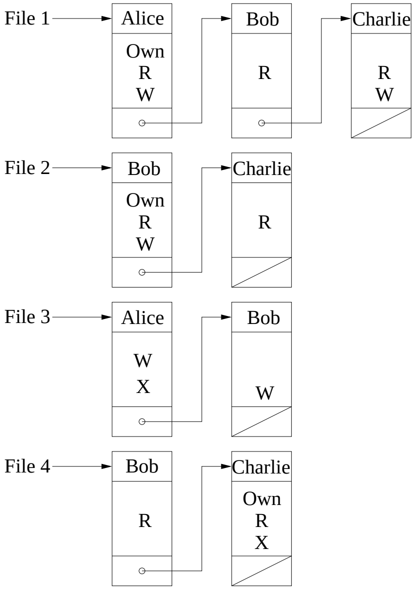
Access Matrix: Data Structures

- Matrices define access rights.
- Different possible realizations as mechanism.

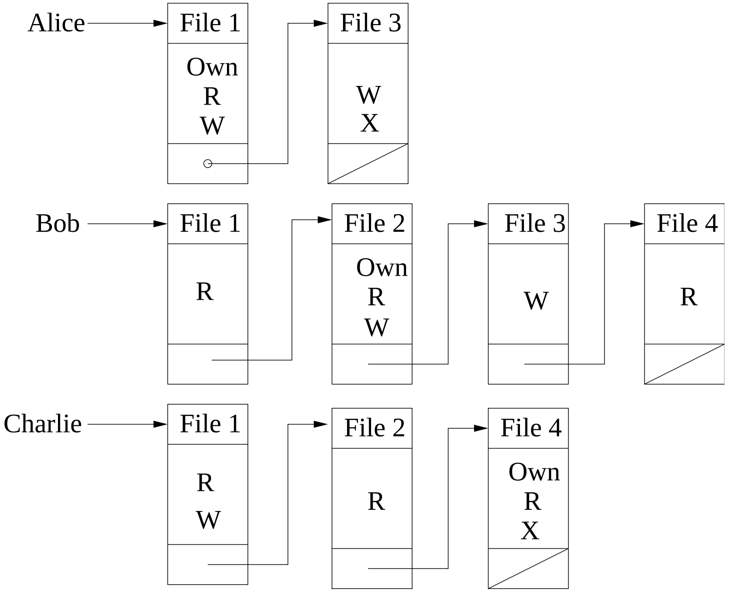
Access Matrix

	File 1	File 2	File 3	File 4
Alice	Own R W		W X	
Bob	R	Own R W	W	R
Charlie	R W	R		Own R X

AC List (ACL)



Capabilities List

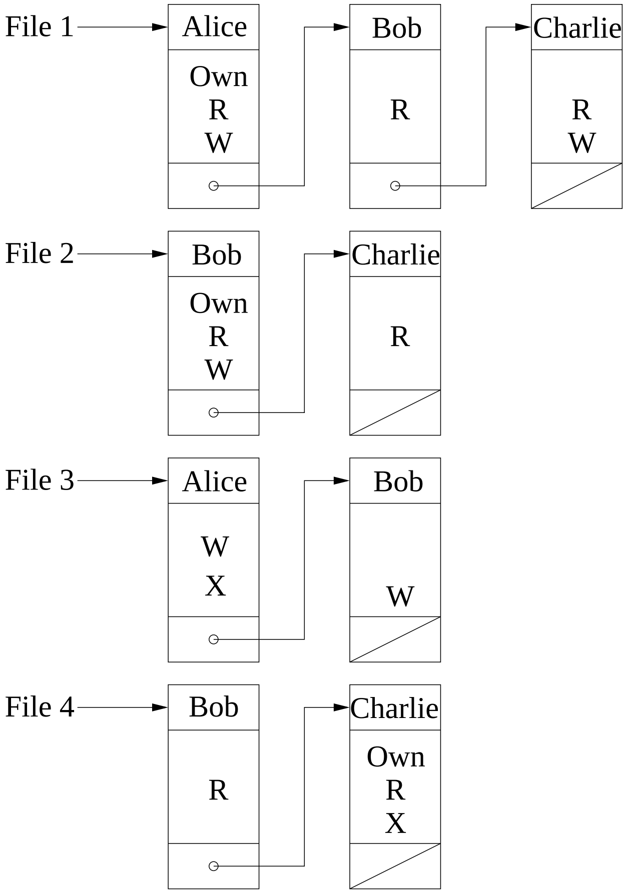


Represent as 2-dimensional objects or set of 1-dimensional objects.

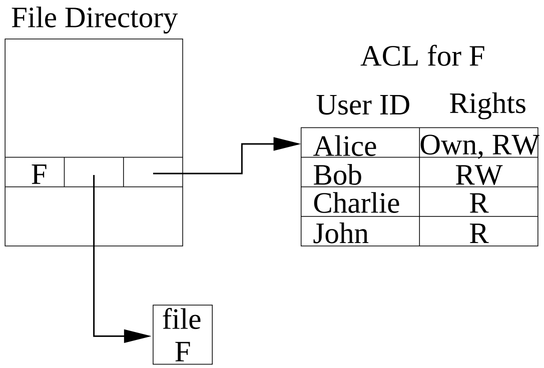
Access-Control (Authorization) List

- ACL: use lists to express view of each object o:

  each entry in the list gives the name of a subject s and the rights r in $M(s, o)$ of the access-matrix.



- Standard example: AC for files.



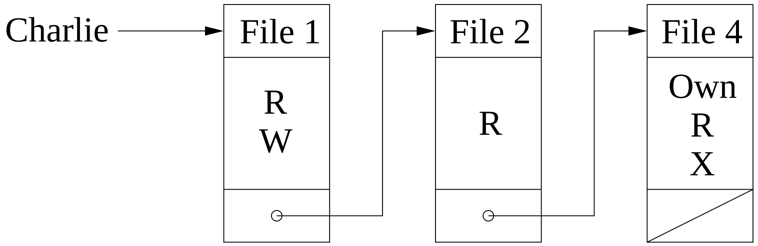
ACL for F

User ID	Rights
Alice	Own, RW
Bob	RW
Charlie	R
John	R

Owner has the sole authority to grant, revoke or decrease access rights to F to other users.

Exception in UNIX: superuser ("root") always has full access and can change all access rights.

Capability List



- Subject view of AC matrix.
- Less common than ACLs.
  - Not so compatible with object oriented view of the world.
  - Difficult to get an overview of who has permissions on an object.
  - Difficult to revoke a capability for a set of users. E.g., chmod o-rwx *
- Application in distributed setting (e.g., mobile agent, Kerberos).

  Users are endowed with credentials (e.g., from a credential server) that they present to network objects.

AC Matrix Model — Formal Definitions (I)

- A state (or: configuration) is a triple $X = (S, O, M)$:

  $S \subseteq$ **Subjects:** Set of subjects.
  $O \subseteq$ **Objects:** Set of objects.
  M : **Subjects** $\times$ **Objects** $\rightarrow \wp($**Permissions**$)$**:** a matrix defining the protection state, i.e. the permissions for each $(s, o) \in S \times O$

  $$\text{where } M(s, o) := \{p \in \text{Permissions} \mid (s, o, p) \in \mathcal{P}\}$$

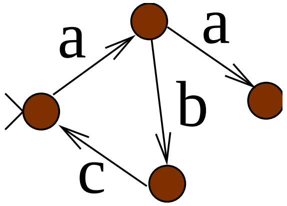
- State transitions described by commands (members of a set Com) like
  - **enter** permission p **into** $M(s, o)$
  - **create** subject s
  - **destroy** object o

  These transform one state into another by changing its parts.

AC Matrix Model — Formal Definitions (II)

- Write $X \rightsquigarrow_c X'$ to denote a state transition associated with c, where $c \in$ Com is a command.
- A starting state $X_0 = (S_0, O_0, M_0)$ and the transition relation $\rightsquigarrow$

  determine a state-transition system.



- So a model describes a set of (possible) system traces, namely (finite) sequences of transitions

$$X_0 \rightsquigarrow_{c_1} X_1 \rightsquigarrow_{c_2} X_2 \ldots \rightsquigarrow_{c_n} X_n$$

where all $X_i \in$ State and all $c_i \in$ Com.

Access Matrix — Policy Example (I)

	File 1	File 2	File 3	File 4
Alice	Own R W		W X	
Bob	R	Own R W	W	R
Charlie	R W	R		Own R X

Policy: A subject has read access to a file only if the permission R was initially present or has been explicitly granted by the file's owner.

Formalization:

For any $s_1, s_2 \in$ Subjects and $o_1 \in$ Objects, $\mathit{confer\_read}(s_2, s_1, o_1) \in$ Com is a command whose effect on the state is

$(S, O, M) \leadsto_{\mathit{confer\_read}(s_2, s_1, o_1)} (S, O, M')$ where

$$(\forall s, o.\ M'(s, o) = (\text{if } (s, o) = (s_1, o_1) \text{ then } M(s, o) \cup \{R\} \text{ else } M(s, o))).$$

Let $X_0 \leadsto_{c_1} X_1 \ldots \leadsto_{c_n} X_n$ be a system trace. State X_n is authorized iff $\forall s', o'.\ R \in M_n(s', o') \rightarrow (R \in M_0(s', o') \vee$

$$(\exists k < n, s.\ X_k \leadsto_{\mathit{confer\_read}(s, s', o')} X_{k+1} \ \wedge\ \mathit{Own} \in M_k(s, o'))).$$

Security Objective: the system is secure, i.e. all reachable states are authorized, i.e. for all traces $X_0 \leadsto_{c_1} X_1 \ldots \leadsto_{c_n} X_n$ the X_n is authorized.

Access Matrix — Policy Example (II)

	File 1	File 2	File 3	File 4
Alice	Own R W		W X	
Bob	R	Own R W	W	R
Charlie	R W	R		Own R X

Solution: For each transition that gives new read access to an object, access control checks that this is done only by the owner of the object using $confer\_read$. Formally:

Let $X = (S, O, M)$ and $X' = (S', O', M')$ be two states and c a command.
The transition $X \leadsto_c X'$ is locally acceptable iff $(R \notin M(s', o') \land R \in M'(s', o')) \longrightarrow$ $(\exists s.\ c = confer\_read(s, s', o') \ \land\ Own \in M(s, o'))$.

Theorem: If access control makes sure that only locally acceptable transitions take place, then all reachable states are authorized, i.e. the system is secure. Formally:

For any trace $X_0 \leadsto_{c_1} X_1 \ldots \leadsto_{c_n} X_n$, if $X_i \leadsto_{c_{i+1}} X_{i+1}$ is locally acceptable for all i, then X_n is authorized (for all n).

Proof: Assume that all transitions $X_i \leadsto_{c_{i+1}} X_{i+1}$ are locally acceptable.
Show by induction on n that X_n is authorized.

Base case: X_0 is trivially authorized.

Access Matrix — Policy Example (III)

	File 1	File 2	File 3	File 4
Alice	Own R W		W X	
Bob	R	Own R W	W	R
Charlie	R W	R		Own R X

Induction step: Take any trace $X_0 \leadsto_{c_1} X_1 \ldots \leadsto_{c_{n+1}} X_{n+1}$. We can assume that X_n is authorized and have to show that X_{n+1} is authorized.
Choosing any s' and o' such that $R \in M_{n+1}(s', o')$,
it remains to show $R \in M_0(s', o') \;\vee\; (\exists k < n+1.\; Q(k))$
where $Q(k) := (\exists s.\; X_k \leadsto_{\mathit{confer\_read}(s,s',o')} X_{k+1} \;\wedge\; \mathit{Own} \in M_k(s, o'))$.
We consider two cases.

1. $R \in M_n(s', o')$, i.e. R did not change.
   From the ind. hypothesis, we conclude $R \in M_0(s', o') \vee (\exists k < n.\; Q(k))$.
   Now $R \in M_0(s', o') \;\vee\; (\exists k < n+1.\; Q(k))$ follows immediately.
2. $R \notin M_n(s', o')$, i.e. R is newly set in $M_{n+1}(s', o')$.
   Since the transition $X_n \leadsto_{c_{n+1}} X_{n+1}$ is locally acceptable, we can infer
   $\exists s.\; c_{n+1} = \mathit{confer\_read}(s, s', o') \;\wedge\; \mathit{Own} \in M_n(s, o'))$.
   Thus we have $Q(n)$ and therefore $R \in M_0(s', o') \;\vee\; (\exists k < n+1.\; Q(k))$.

$\square$

Access Matrix — Policy Example with Isabelle

Isabelle: generic interactive theorem proving system

HOL: higher-order logic, mixture of predicate logic and λ-calculus

ProofGeneral: XEmacs mode for Isabelle etc., used in live demo now

```
theory AC_matrix = Main:

typedecl Subject
typedecl Object

datatype Permission = Own | R | other_Permissions

types Protection_State = "Subject × Object ⇒ Permission set"
                 State = "Subject set × Object set × Protection_State"

datatype Com = confer_read "Subject × Subject × Object"
             | other_Coms
```

Access matrix — policy example with Isabelle: traces

For simplicity, only one trace, of unbounded length

consts *X* :: "nat ⇒ State"
  C :: "nat ⇒ Com" — 0-th command unused

syntax
"X_" :: "nat ⇒ State" ("X_")
"S_" :: "nat ⇒ Subject" ("S_")
"O_" :: "nat ⇒ Object" ("O_")
"M_" :: "nat ⇒ Protection_State" ("M_")
"C_" :: "nat ⇒ Com" ("C_")

translations
"X_n" ⇌ "X n"
"S_n" ⇌ "fst X_n"
"O_n" ⇌ "fst (snd X_n)"
"M_n" ⇌ "snd (snd X_n)"
"C_n" ⇌ "C n"

consts transition :: "State ⇒ Com ⇒ State ⇒ bool" ("(_ $\rightsquigarrow$_. _)")
constdefs is_trace :: "bool"
  "is_trace ≡ ∀n. $X_n \rightsquigarrow C_{(n+1)}.\ X_{(n+1)}$"

Access matrix — policy example with Isabelle: misc

axioms transition_confer_read: — unused
"(S,O_,M) $\leadsto$confer_read(s2,s1,o1).
(S,O_,(λ(s',o'). if (s',o') = (s1,o1) then M(s',o') ∪ {R} else M(s',o')))"

constdefs authorized :: "nat ⇒ bool"
"authorized n ≡ ∀ s' o'.
R ∈ M_n (s',o') ⟶
R ∈ M_0 (s',o') ∨ (∃ k<n. ∃ s. $C_{(k+1)}$ = confer_read(s,s',o') ∧ Own ∈ M_k (s,o'))"

constdefs locally_acceptable :: "nat ⇒ bool"
"locally_acceptable i ≡ ∀ s' o'.
(R ∉ M_i (s',o') ∧ R ∈ $M_{(i+1)}$ (s',o')) ⟶
(∃ s. $C_{(i+1)}$ = confer_read(s,s',o') ∧ Own ∈ M_i (s,o'))"

Access matrix — policy example: Isabelle proof script

“Classcial” tactic style, “proof assembly language”

```
theorem system_safe: "⟦is_trace; ∀i. locally_acceptable i⟧ ⟹ ∀n. authorized n"
apply (rule allI)
apply (rule nat.induct)
apply  (unfold authorized_def)
apply  (fast)
apply (rule allI, rule allI, rule impI)
apply (case_tac "R ∈ M_na (s', o')")
apply  (drule spec, drule spec, erule (1) impE, erule disjE)
apply   (erule disjI1)
apply  (rule disjI2)
apply  (erule exE, erule conjE)
apply  (rule_tac x = k in exI)
apply  (blast intro: less_SucI)
apply (simp add: locally_acceptable_def)
apply (drule spec, drule spec, drule spec, erule impE, erule (1) conjI)
apply (rule disjI2)
apply (rule_tac x = na in exI)
apply (blast)
done
```

Access matrix — policy example: Isabelle ISAR proof

Mostly automatic proof

```
theorem system_safe: "⟦is_trace; ∀ i. locally_acceptable i⟧ ⟹ authorized n"
apply (rule nat.induct)
apply (simp_all add: authorized_def locally_acceptable_def)
apply (blast intro: less_SucI)+
done
```

Intelligible Semi-Automatic Reasoning

```
theorem system_safe: "⟦is_trace; ∀ i. locally_acceptable i⟧ ⟹ ∀ n. authorized n"
proof
  fix n
  assume local_accept: "∀ i. locally_acceptable i"
  show "authorized n"
  proof (induct n, simp_all only: Suc_plus1)
    show "authorized 0" by (unfold authorized_def, fast)
  next
    fix n
    assume ind_hyp: "authorized n"
    show "authorized (n+1)"
    proof (unfold authorized_def, rule, rule, rule)
```

```
      fix s' o'
      assume assumpt: "R ∈ $M_{(n+1)}$ (s', o')"
      let ?Q = "λk. ∃ s. $C_{(k+1)}$ = confer_read (s, s', o') ∧ Own ∈ $M_k$ (s, o')"
      show "R ∈ $M_0$ (s', o') ∨ (∃ k<n+1. ?Q(k))"
      proof cases
        assume "R ∈ $M_n$ (s', o')"
        with ind_hyp have "R ∈ $M_0$ (s', o') ∨ (∃ k<n. ?Q(k))"
          by (unfold authorized_def, fast)
        then show ?thesis by (simp, blast intro: less_SucI)
      next
        assume "R ∉ $M_n$ (s', o')"
        with local_accept assumpt
        have "∃ s. $C_{(n+1)}$ = confer_read (s, s', o') ∧ Own ∈ $M_n$ (s, o')"
          by (simp add: locally_acceptable_def)
        hence "?Q(n)".
        thus "R ∈ $M_0$ (s', o') ∨ (∃ k<n+1. ?Q(k))" by (simp, fast)
      qed
    qed
  qed
qed

end
```

Outline

- Access Control (AC)
- Discretionary Access Control (DAC)
- Access Control Matrix Model
- ☞ **Mandatory Access Control (MAC)**
- Bell-LaPadula Model and Variants
- Role-Based Access Control (RBAC)

Mandatory Access Control (MAC)

- System wide access restrictions to objects.

  Mandatory because subjects may not transfer their access rights.

- AC decisions controlled by comparing

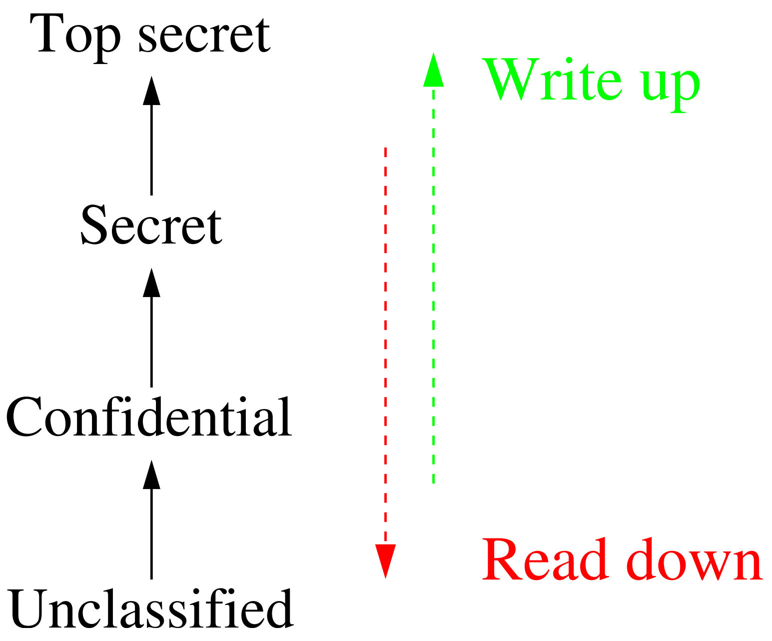
  security labels indicating sensitivity/criticality of objects, with **formal authorization,** i.e. security clearances, of subjects.

- Example from military: users and objects assigned a clearance level like *confidential*, *secret*, *top secret*, etc. Users can only read [write] objects of equal or lower [higher] levels.

- More rigid than DAC, but also more secure.

- Concrete examples (like Bell-LaPadula) later.

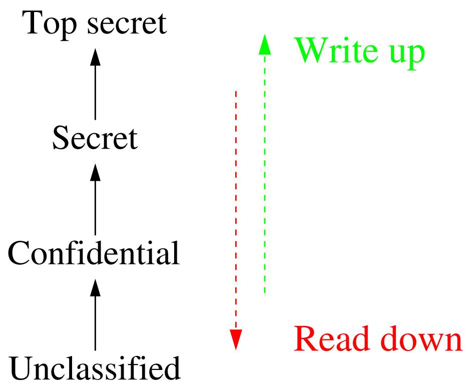
MAC: Linear Ordering



Two principles are required to hold for confidentiality:

- Read down: a subject's clearance must dominate (i.e. $\geq$) the security level of the object being read.
- Write up: a subject's clearance must be dominated by (i.e. $\leq$) the security level of the object being written.

MAC: Linear Ordering (cont.)



Top secret

Secret

Confidential

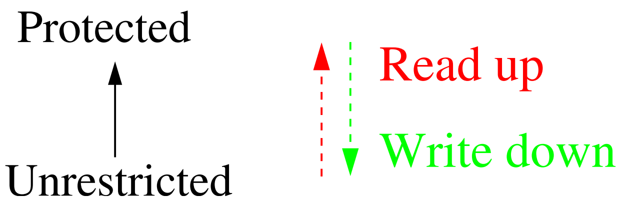
Unclassified

Write up

Read down

- Problems:
  - It allows to send email “up”, but is often restricted only to same level (i.e. $=$) to avoid “blind overwriting”.
  - It does not allow a subject to write “lower” data; to that end a subject should be enabled to dynamically decrease its level.

- Can be applied similarly for integrity: read up and write down:



Protected

Unrestricted

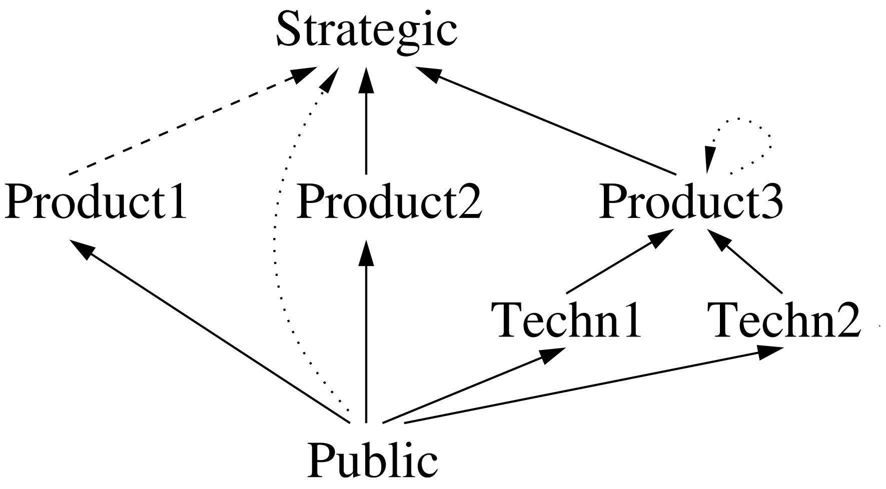
Read up

Write down

MAC: Ordering generalized

Def: a partial ordering $(L, \sqsubseteq)$ on a set L is a binary relation on L (i.e. a subset of $L \times L$) that is reflexive, antisymmetric, and transitive.

Example: Hasse diagram of company secrets



Questions:

- Given 2 objects at different security levels, what is the minimal level a subject must have to be allowed to read both objects?
- Given 2 subjects at different security levels, what is the maximal level an object can have so that it still can be read by both subjects?

MAC: The Lattice of Security Levels

Def: a lattice $(L, \sqsubseteq)$ is a partial ordering $(L, \sqsubseteq)$ on a set (of security levels) L, so that for every two elements $a, b \in L$ there exists a least upper bound $u \in L$ and a greatest lower bound $l \in L$, i.e.

$$a \sqsubseteq u \text{ and } b \sqsubseteq u \text{ and } \forall u' \in L.\ (a \sqsubseteq u' \land b \sqsubseteq u') \to u \sqsubseteq u'$$
$$l \sqsubseteq a \text{ and } l \sqsubseteq b \text{ and } \forall l' \in L.\ (l' \sqsubseteq a \land l' \sqsubseteq b) \to l' \sqsubseteq l$$

We write $lub(\{a, b\})$ or $a \sqcup b$ for u
and $glb(\{a, b\})$ or $a \sqcap b$ for l.

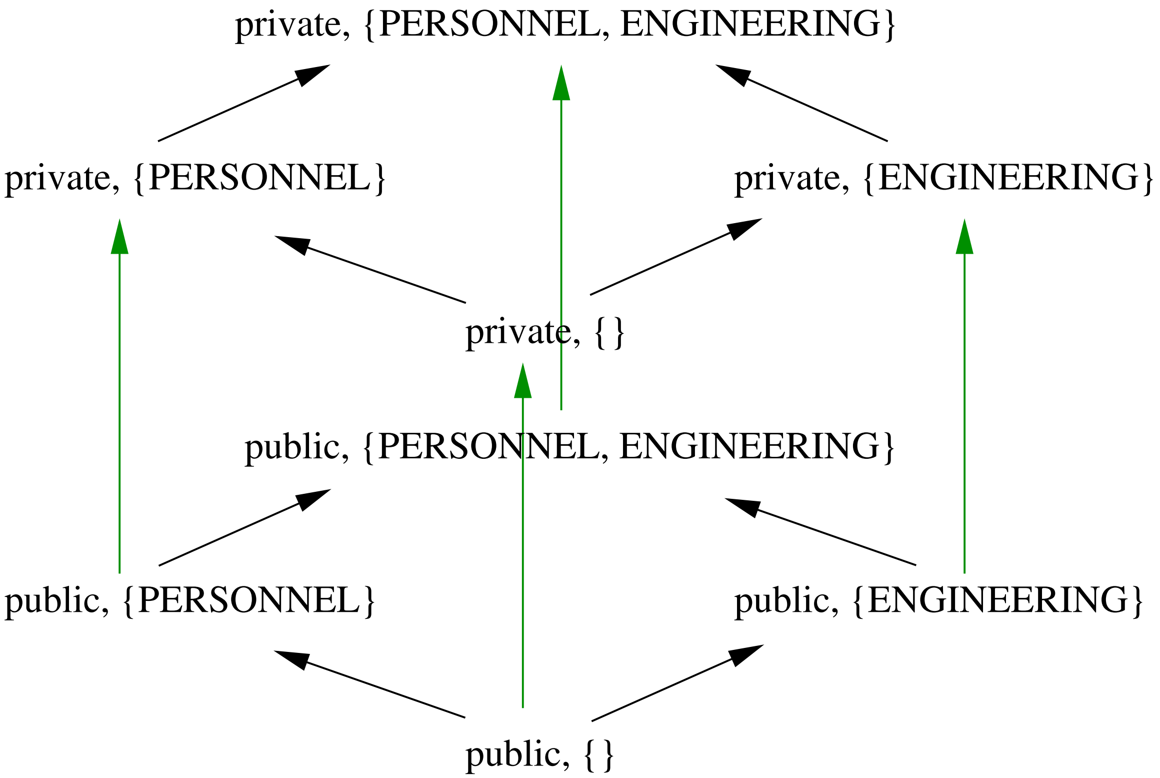
Examples:

- the linear ordering on the naturals: $(\mathbb{N}, \leq)$
- the subset ordering on powersets: $(\wp(S), \subseteq)$

Example (from DoD's Orange Book)

- A set H of classifications with a hierarchical (linear) ordering $\leq$.
- A set C of categories, e.g. project names, company divisions, etc.
- A security label is a pair (h, c) with $h \in H$ and $c \subseteq C$.
- Partial order of labels: $(h_1, c_1) \sqsubseteq (h_2, c_2)$ iff $h_1 \leq h_2$ and $c_1 \subseteq c_2$.

For hierarchical levels *public* and *private*, and categories PERSONNEL and ENGINEERING, we have the lattice:



Note that public,{PERSONNEL} $\not\sqsubseteq$ private,{ENGINEERING}.

Outline

- Access Control (AC)
- Discretionary Access Control (DAC)
- Access Control Matrix Model
- Mandatory Access Control (MAC)

☞ **Bell-LaPadula Model and Variants**

- Role-Based Access Control (RBAC)

The Bell-LaPadula (BLP) Model (1975)

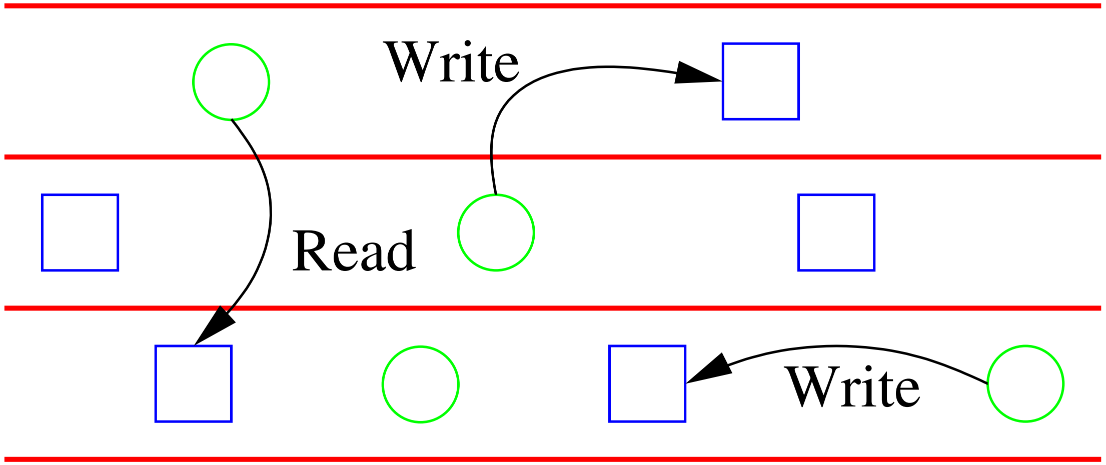
- Models confidentiality aspects of multi-user systems, e.g. in operating systems or database management systems.

- Probably most famous and influential security model:
  - Developed as part of U.S. government funded research at the MITRE corporation on security models and the prevention of disclosure threats in multi-user operating systems.
  - Basis of several standards, including DoD's Trusted Computer System Evaluation Criteria (TCSEC or "Orange Book").
  - It also raised some controversy (on suitable definition of security model).

The Bell-LaPadula (BLP) Model (cont.)

- BLP models confidentiality by combining aspects of DAC and MAC:
  - Access permissions are defined both through an AC matrix and through security levels.
  - Multi-level security (MLS): mandatory policies prevent information flowing downwards from a high security level to a low one.
  - BLP is a static model: security levels (labels) never change.

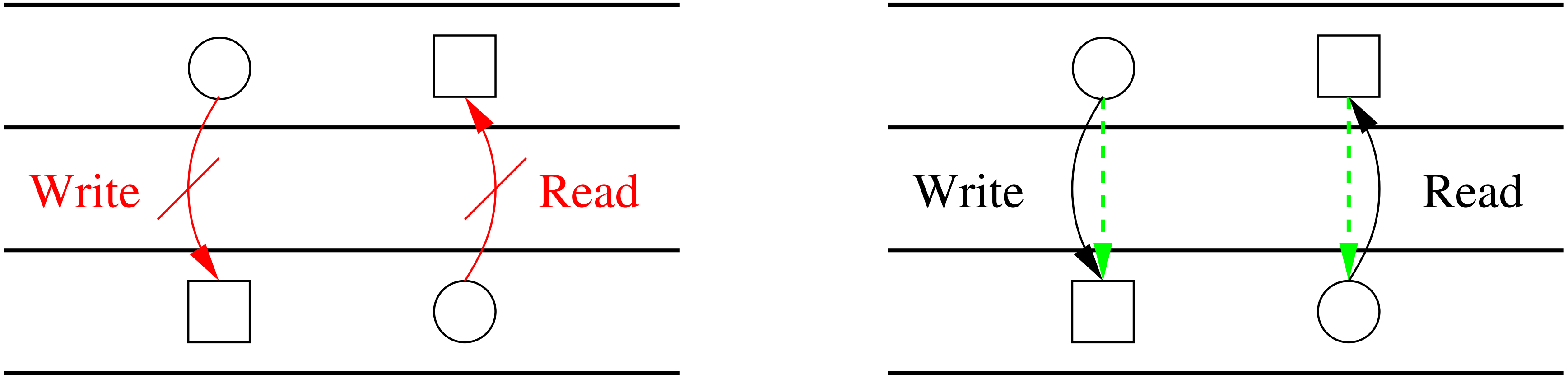
BLP: Level Diagrams



- Horizontal lines: boundaries between levels (with partial order $\leq$).
- Circles: subjects.
- Squares: objects.
- Directed arcs from subjects to objects: operations (e.g. read, write, execute).

BLP: Level Diagrams (cont.)

- Level diagrams also for disallowed operations and information flow:



- But level diagrams do not adequately represent the lattice properties of the security labels under the *dominates* ($\leq$) relation, e.g. when a subject s tries to access an object o with an unrelated security label (i.e. where neither $\mathit{label}(s) \leq \mathit{label}(o)$ nor $\mathit{label}(s) \geq \mathit{label}(o)$).

- Formal BLP model for “real” security policies.

BLP Formalization: Basic Sets

Basic sets:

- a set of subjects S and a set of objects O,
- a set of security levels L with partial ordering $\leq$,
- a set of access operations $A = \{execute, read, append, write\}$.

Four access rights defined in terms of two basic access modes:

	execute	read	append (blind write)	write
observe (look at contents of object)		×		×
alter (change contents of object)			×	×

- ► Append ("alter without observing"), e.g. for audit files.
- ► Execute does not require observing or altering, e.g. for cryptographic engines.

BLP: Formalization: State Set

The state set $\mathcal{B} \times \mathcal{M} \times \mathcal{F}$ captures all current permissions and all current instances of subjects accessing objects:

- $\mathcal{B} = \mathcal{P}(S \times O \times A)$ is the set of current accesses.

  $b \in \mathcal{B}$ is a collection of tuples (s, o, a), indicating that subject s currently performs operation a on object o.

- $\mathcal{M} = \{M(s, o) \mid s \in S \text{ and } o \in O\}$ is the set of access matrices.

- $\mathcal{F} \subset L^S \times L^S \times L^O$ is the set of security level assignments (f_S, f_C, f_O) where
  - $f_S : S \to L$ gives the maximal security level (also called clearance) each subject can have,
  - $f_C : S \to L$ gives the current security level of each subject, which must be $f_C \leq f_S$ (i.e. f_S must dominate f_C),
  - $f_O : O \to L$ gives the classification of each object.

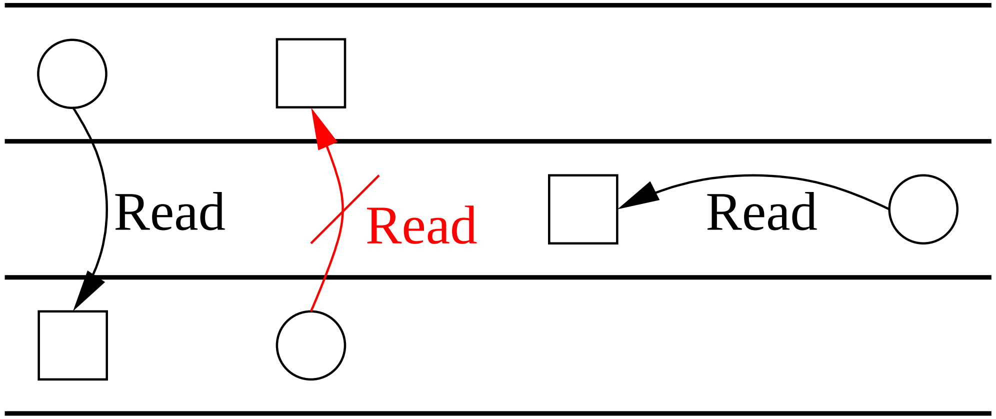
BLP: Security Properties for a State (b, M, f)

BLP defines different security properties for a state (b, M, f), e.g.

- Simple security property (ss-property):

> For each element $(s, o, a) \in b$ where the access operation a is $read$, the security level of the subject s dominates the classification of the object o, i.e. $f_O(o) \leq f_S(s)$.

Also known as no-read-up (NRU):

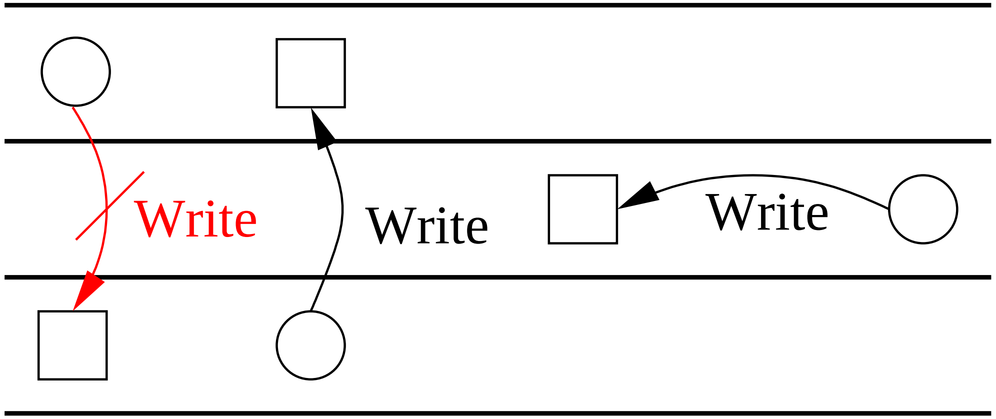


e.g.: an 'unclassified' s should not be able to read a 'confidential' o.

BLP: Security Properties for a State (b, M, f) (cont.)

- $*$-property (star-property):

For each $(s, o, a) \in b$ where a is *append* or *write*, the current level of subject s is dominated by the classification of object o, i.e. $f_C(s) \leq f_O(o)$. (This is a no-write-down security policy.) Moreover, if there is $(s, o, a) \in b$ where $a \in \{append, write\}$, then it must be $f_O(o') \leq f_O(o)$ for all o' with $(s, o', a') \in b$ where a' is *read* or *write*.



e.g.: a 'confidential' s should not be able to write an 'unclassified' o.

BLP: Security Properties (cont.)

- NWD prevents a high-level subject from sending messages to a low-level one. Possible solutions:
  - Temporarily downgrade the level of s (by means of current security level f_C).
  - Identify a set of subjects (called trusted subjects), which are permitted to violate the $*$-property.

- Rationale of ss-property and $*$-property: no information leakage.
  - No-read-up and no-write-down prevent untrusted subjects from simultaneously having read access to information at one level and write access to information at a lower level.
  - Read-down and write-up are fine: a ‘confidential’ s should be able to read an ‘unclassified’ o_1 in order to write a ‘confidential’ o_2.

BLP: Security Properties for a State (b, M, f) (cont.)

- Other security properties can be formalized in BLP, e.g. discretionary security property (ds-property).

> For each element of $(s, o, a) \in b$ it must be that $a \in M(s, o)$.

Access control based on named users and named objects, where subjects holding an access permission may pass that permission on to other subjects.

BLP: Security States

- Definition: A state is called secure if all three security properties are satisfied.
- Definition: A transition from state $v_1 = (b_1, M_1, f_1)$ to state $v_2 = (b_2, M_2, f_2)$ is said to be secure, if both v_1 and v_2 are secure.
- For example, a transition preserves the ss-property if and only if:

1. Each $(s, o, a) \in (b_2 \setminus b_1)$ satisfies the ss-property with respect to f_2.
2. If $(s, o, a) \in b_1$ does not satisfy the ss-property with respect to f_2, then $(s, o, a) \notin b_2$.

Preservation of the $*$-property and of the ds-property is defined similarly.

BLP: Basic Security Theorem

- A state is called secure if all three security properties are satisfied.

> Basic security theorem: if all state transitions in a system are secure and if the initial state of the system is secure, then every subsequent state will also be secure, no matter which inputs occur.

- Proof: by induction over the length of input sequences.
  - ► The theorem can be applied in general for state machine models.
  - ► The proof would build on the fact that each state transition preserves security but would not refer to the specific BLP security properties.

BLP: Features and Limitations

- BLP is well-suited for modeling confidentiality in operating systems or database management systems.

- It does suffer from a number of limitations, though:
  - It does not precisely describe transitions.
  - It does not specify how to change access rights or how to create and delete subjects and objects, i.e. it is a static model (this tranquility raised much controversy).
  - It contains covert channels, i.e. information-flows that are not controlled by security mechanisms.

The tranquility problem can be addressed by employing the Harrison-Ruzzo-Ullman model.

BLP: System Z and the Tranquility Property (I)

- System Z has only one state transition, which
  - downgrades all subjects and objects to the lowest security level,
  - enters all access rights in all positions of the AC matrix M.

- System Z satisfies BLP's notion of security.

> According to the basic security theorem of BLP, the state reached by this transition is secure, but is it really?

  - The case against BLP (McLean): BLP needs to be improved, as a system that can be brought into a state where everyone is allowed to read everything is not secure.
  - The case for BLP (Bell): this is not a problem of BLP but rather of correctly capturing the security requirements.
    If the user requirements call for such a transition, then it should be allowed in the model, else it should not be implemented.

BLP: System Z and the Tranquility Property (II)

- At the root of this disagreement is a state transition that changes security levels (and access rights).

- Most read and write requests on an actual system are not atomic, but are comprised of sequences of operations that may be interrupted by some other activity of the system.

  For example: a “print file” request may involve a sequence of system calls and kernel routines to locate the file, open it for reading, and then initiate a printing process.

- BLP is however a static model: NRU and NWD rules implicitly require that the security labels of the subjects and objects involved in some desired access not be changed while the access is still being processed in such a manner to produce a violation of a defined security policy.

  Else, a “secret” subject can request read access to a “secret” object and, while the request is being processed, lower its level to “unclassified” so that read access to a “secret” object is ultimately granted to an “unclassified” subject.

BLP: System Z and the Tranquility Property (III)

- BLP is a static model:

> Strong tranquility property: the security levels of subjects and objects never change during system operation.

> Weak tranquility property: the security levels never change in such a way as to violate a defined security policy.

For example, it can be required that the level of an object never be changed while it is being used by some subject.

- This limitation is lifted in dynamic models that are based on BLP.

BLP: Covert Channels

Sometimes it is not sufficient to hide the contents of objects, but also their existence must be hidden.

- In BLP, the AC mechanism itself can be used to construct a covert channel, where information flows from a high security level to a low one (which could constitute an "attack'").
  - ▶ A low-level subject creates an object `dummy.obj` at its own level.
  - ▶ Its high-level accomplice (e.g. a Trojan horse) either upgrades the security level of `dummy.obj` to high or leaves it unchanged.
  - ▶ Later, the low-level subject tries to read `dummy.obj`.
    Success or failure of this request disclose the action of the high-level subject.
    Hence, one bit of information has flown from high to low.
- That is: telling a subject that a certain operation is not permitted already constitutes information-flow.
- Problem can be solved; e.g. in database security an object may have different values at different security levels (polyinstantiation).

The Harrison-Ruzzo-Ullman Model (1976)

- BLP model does not state policies for changing access rights or for the creation and deletion of subjects and objects.
- The *Harrison-Ruzzo-Ullman model* defines authorization systems that address these issues.
  - State (S, O, M), for subjects S, objects O, matrix M.
  - State transitions described by commands of the form

```
command $c(x_1, \ldots, x_k)$
    if $r_1$ in $M(x_{s_1}, x_{o_1})$ and ... $r_m$ in $M(x_{s_m}, x_{o_m})$
    then $op_1$; ... $op_n$
end
```

for rights r_i, integers s_i and o_i, primitive operations op_i

enter r **into** $M(s, o)$	**delete** r **from** $M(s, o)$
create subject s'	**destroy** subject s'
create object o'	**destroy** object o'

The Harrison-Ruzzo-Ullman Model (cont.)

Example instances of

```
command $c(x_1, \ldots, x_k)$
  if $r_1$ in $M(x_{s_1}, x_{o_1})$ and $\ldots$ $r_m$ in $M(x_{s_m}, x_{o_m})$
  then $op_1$; $\ldots$ $op_n$
end
```

are

```
command $create.file(s, o)$
  create $o$
  enter $\mathrm{Own}$ into $M(s, o)$
  enter $\mathrm{R}$ into $M(s, o)$
  enter $\mathrm{W}$ into $M(s, o)$
end
```

```
command $confer.write(s1, s2, o)$
  if $\mathrm{Own} \in M(s1, o)$
  then enter $\mathrm{W}$ into $M(s2, o)$
end
```

The Harrison-Ruzzo-Ullman Model (cont.)

Six primitive operations causing a transition from state (S, O, M) to state (S', O', M').
Assumption: all subjects are objects, i.e. $S \subseteq O$.

Operation	Conditions	New State
enter r **into** $M(s,o)$	$s \in S$ $o \in O$	$S' = S$ $O' = O$ $M'(s,o) = M(s,o) \cup \{r\}$ $M'(s_1,o_1) = M(s_1,o_1)$ for $(s_1,o_1) \neq (s,o)$
delete r **from** $M(s,o)$	$s \in S$ $o \in O$	$S' = S$ $O' = O$ $M'(s,o) = M(s,o) \setminus \{r\}$ $M'(s_1,o_1) = M(s_1,o_1)$ for $(s_1,o_1) \neq (s,o)$

Operation	Conditions	New State
create subject s'	$s' \notin O$	$S' = S \cup \{s'\}$ $O' = O \cup \{s'\}$ $M'(s,o) = M(s,o)$ for $s \in S$, $o \in O$ $M'(s',o) = \emptyset$ for $o \in O'$ $M'(s,s') = \emptyset$ for $s \in S'$
destroy subject s'	$s' \in S$	$S' = S \setminus \{s'\}$ $O' = O \setminus \{s'\}$ $M'(s,o) = M(s,o)$ for $s \in S'$, $o \in O'$
create object o'	$o' \notin O$	$S' = S$ $O' = O \cup \{o'\}$ $M'(s,o) = M(s,o)$ for $s \in S$, $o \in O$ $M'(s,o') = \emptyset$ for $s \in S'$
destroy object o'	$o' \in O$ $o' \notin S$	$S' = S$ $O' = O \setminus \{o'\}$ $M'(s,o) = M(s,o)$ for $s \in S'$, $o \in O'$

The Harrison-Ruzzo-Ullman Model (cont.)

Def: A state $Q = (S, O, M)$ yields a state $Q' = (S', O', M')$ under

```
**command** $c(x_1, \dots, x_k)$
    **if** $r_1$ in $M(x_{s_1}, x_{o_1})$ and ... $r_m$ in $M(x_{s_m}, x_{o_m})$
    **then** $op_1$; ... $op_n$
**end**
```

with arguments $a_1, \dots, a_k$, written $Q \rightsquigarrow_{c(a_1,\dots,a_k)} Q'$, provided

- $Q' = Q$ if one of the conditions of c is not satisfied.
- $Q' = Q_n$ otherwise, where there exist states $Q_0, Q_1, \dots, Q_n$ such that $Q_0 = Q$ and $Q_n = Q'$ and for each i, with $0 \leq i \leq n$,

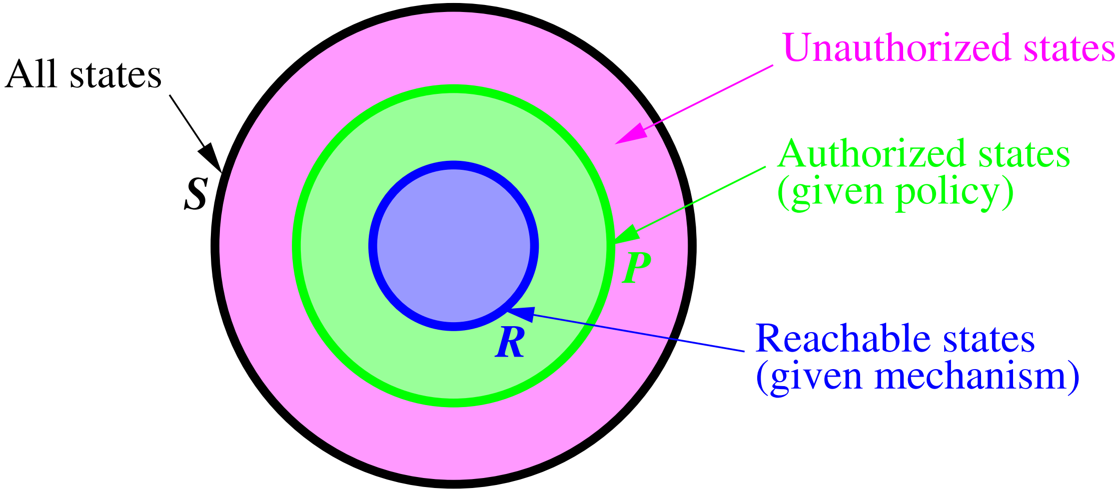
$$Q_i \rightsquigarrow_{op_{i+1}[a_j/x_j]} Q_{i+1}$$

where $op_{i+1}[a_j/x_j]$ denotes the primitive operation op_{i+1}, substituting $a_1, \dots, a_k$ for the variables $x_1, \dots, x_k$.

The Harrison-Ruzzo-Ullman Model (cont.)

Secure: $R \subseteq P$

Precise: $R = P$



- A configuration of the access matrix describes what subjects can do, not necessarily what they are authorized to do.
- A protection (or security) policy partitions the set of all possible states into authorized and unauthorized states.
- Whether a state is authorized depends on the previous state and on the command causing the state transition, e.g.

  no subject can acquire write access to a file
  unless that right has been explicitly granted by the file's owner.

The Chinese Wall Model (Brewer and Nash, 1989)

- A commercially inspired confidentiality model (whereas most commercial models focus on integrity).
- Models access rules in a consultancy business where analysts have to make sure that no conflicts of interest arise when they are dealing with different companies.
- Informally, conflicts arise
  - because clients are direct competitors in the same market, or
  - because of the ownership of companies.

Rule: There must be no information-flow that causes a conflict of interest.

The Chinese Wall Model (cont.)

An adaptation of Bell-LaPadula, with three levels of abstraction:

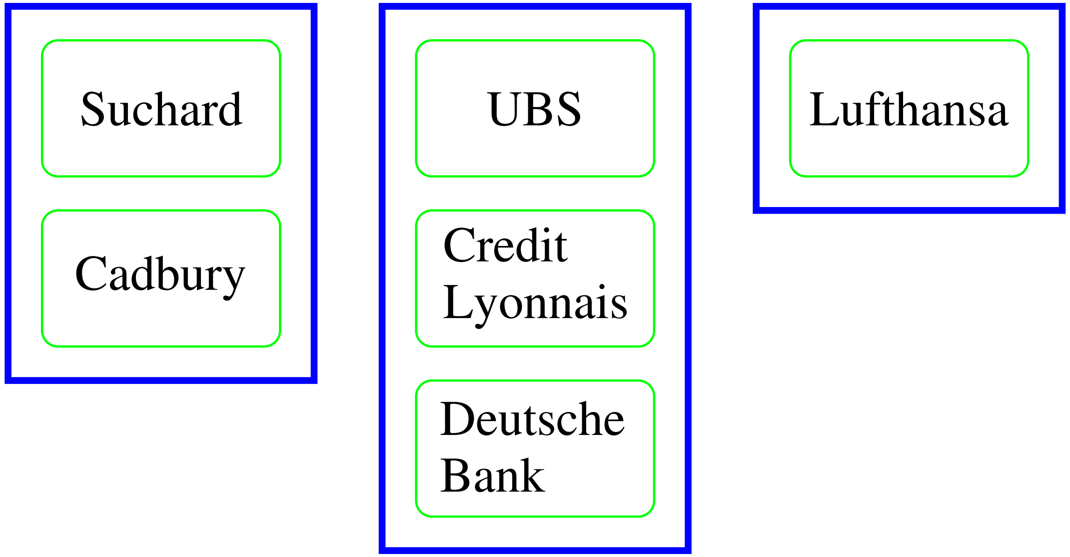
1. Companies, subjects and objects:
   - A set C of companies, and a set S of subjects (the analysts).
   - A set O of objects, which are items of information (e.g. files) concerning a single company.
2. All objects concerning the same company are collected in a company dataset. $cd : O \rightarrow C$ gives the company dataset of each object.
3. Conflict of interest classes indicate which companies are in competition.

   $cic : O \rightarrow \mathcal{P}(C)$ gives the conflict of interest class for each object o, i.e. the set of companies that should not learn the contents of o.

The security label of an object o is the pair $(cic(o), cd(o))$.

The Chinese Wall Model (cont.)

- Each object belongs to a unique company dataset.
- Each company dataset is contained in a unique conflict class.
- A conflict class may contain one or more company datasets.
- For example, chocolate, banks and airlines:



- Six company datasets: one for each company.
- Three conflict classes: {Suchard, Cadbury}, {UBS, Credit Lyonnais, Deutsche Bank}, {Lufthansa}.

The Chinese Wall Model (cont.)

- Conflicts arise not only from objects currently addressed but also from objects that have been accessed in the past.

  A Boolean $S \times O$ matrix N records the subjects' actions:

  $$N(s, o) = \text{true, if subject } s \text{ has had access to object } o$$

  A secure initial state: set $N(s, o) =$ false for all $s \in S$ and $o \in O$.

- Access permissions change dynamically and must be reexamined at every state transition: as a subject accesses some objects, other objects that would previously have been accessible are now denied.

The Chinese Wall Model (cont.)

A simple policy to prevent conflict of interest:

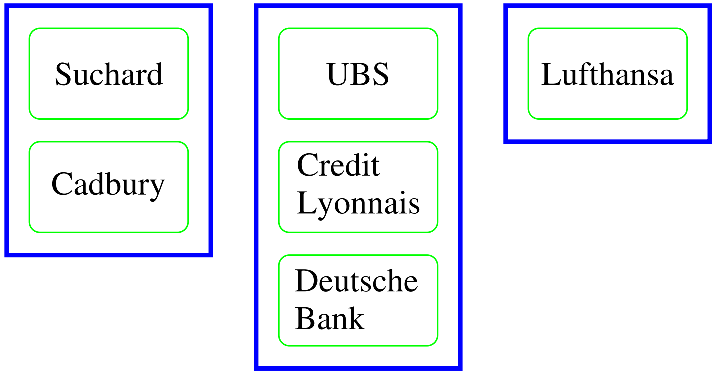
A subject s can access any information as long as it has never accessed information from a different company in the same conflict class.

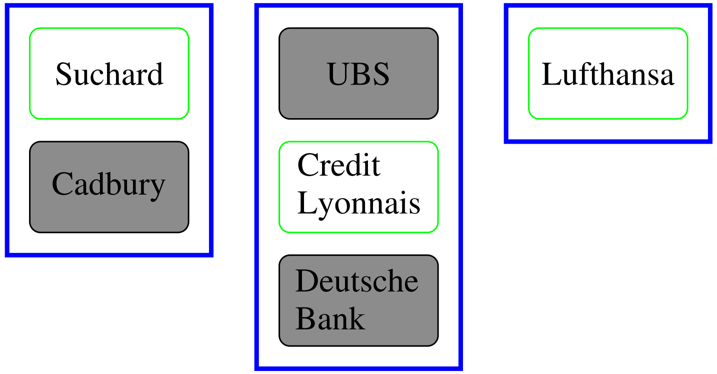
That is, access is granted if and only if requested object o belongs to

- either a company dataset already held by s (o is in the same company dataset as an object that has been previously accessed),
- or an entirely different conflict of interest class (i.e. the class has never before been accessed).

ss-property: s is permitted access to o only if for all o' with $N(s, o') =$ true, it holds $cd(o) = cd(o')$ or $cd(o) \notin cic(o')$.

The Chinese Wall Model (cont.)





- Initially (figure on the left), each object can be accessed.
- If s reads from a file on Suchard, then a subsequent access request
  - to any bank or to Lufthansa would be granted,
  - to Cadbury files would be denied.
- A subsequent access
  - to Lufthansa does not affect future accesses,
  - to a file on Credit Lyonnais blocks future accesses to UBS or Deutsche Bank.
- From that point on (figure on the right, with grey datasets blocked), only objects on Suchard, Lufthansa or Credit Lyonnais (or in a new conflict class) can be accessed.

The Chinese Wall Model (cont.)

ss-property: s is permitted access to o only if for all o' with $N(s, o') =$ true, it holds $cd(o) = cd(o')$ or $cd(o) \notin cic(o')$.

Indirect information-flow is still possible with this property, e.g.

- Two competitors, *Company1* and *Company2*, have their accounts with the same *Bank*.
- *Analyst1*, dealing with *Company1* and the *Bank*, updates the *Bank* portfolio with sensitive information about *Company1*.
- *Analyst2*, dealing with *Company2* and the *Bank*, now has access to information about a competitor's business.

The Chinese Wall Model (cont.)

- Information is sanitized if it has been purged of sensitive details and is not subject to access restrictions.

$$cic(o) = \emptyset \text{ for a sanitized object } o.$$

- Hence, grant write access to an object only if no other object can be read which is in a different company dataset and contains unsanitized information.

$*$-property: s is granted write access to o only if s has no read access to an object o' with $cd(o) \neq cd(o')$ and $cic(o') \neq \emptyset$.

Summarizing:

- BLP: access rights are (usually) assumed to be static.
- Chinese Wall: access rights are changed dynamically, and must thus be re-examined in every state transition.

The Biba Model (K.J. Biba, 1977)

- In BLP: no-read-up and no-write-down for confidentiality.
  - But: write-up and read-down can introduce integrity problems.
- Biba (also of MITRE) proposed a class of integrity models with the opposite rules:
  - Mandatory integrity model: no-read-down and no-write-up.
  - Relax no-read-down ("subject low watermark property"):
    Allow a subject to read down, but first lower its integrity level to that of the object being read.
  - Relax no-write-up ("object low watermark property"):
    Lower object level to that of subject doing the write.
- Biba and BLP can be combined (albeit not straightforwardly) to model both confidentiality and integrity.

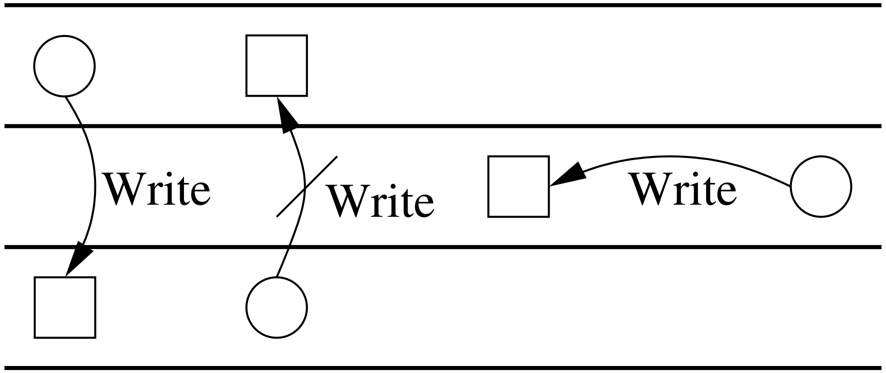
The Biba Model (cont.)

- Addresses integrity in terms of access by subjects to objects using a state machine model similar to that of BLP.
  - A lattice $(L, \leq)$ of security levels.
  - $f_S : S \rightarrow L$ and $f_O : O \rightarrow L$ assign integrity levels to subjects and objects.
  - Information may only flow downwards in the integrity lattice.
- Unlike BLP, there is no single high-level integrity policy but rather a variety of policies (some even mutually incompatible).
  - Static integrity levels.
  - Dynamic integrity levels.
  - Policies for invocation.

Biba: Static Integrity Levels

- Policies where integrity levels never change (mirroring BLP's tranquility).
- Two properties (dual of the mandatory BLP policies):

Simple integrity property: s can modify o if and only if $f_S(s) \geq f_O(o)$. (No-write-up.)



Integrity $*$-property: if s can read o_1, then s can have write access to some other object o_2 only if $f_O(o_2) \leq f_O(o_1)$.

Biba: Dynamic Integrity Levels

Low watermark properties (similar to Chinese Wall) automatically adjust the integrity level of an entity if it has come into contact with low-level information:

- Subject low watermark property: relax no-read-down.

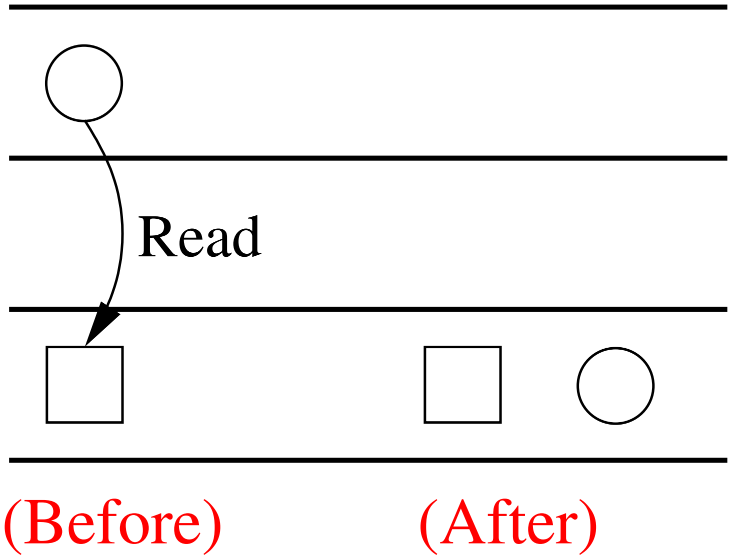
  Allow a subject to read down, but first lower its integrity level to that of the object being read.

- Object low watermark property: relax no-write-up.

  Lower object level to that of subject doing the write.

Biba: Dynamic Integrity Levels (cont.)

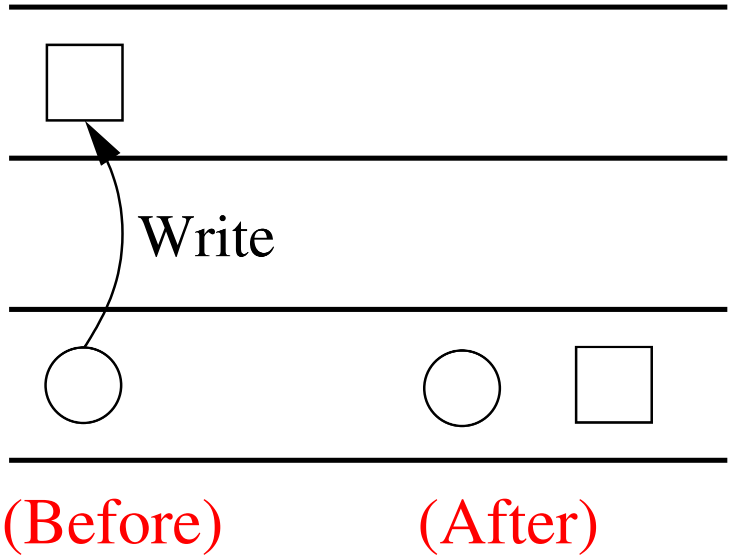
Subject low watermark property: s can read an o at any integrity level.
The new integrity level of s is $\mathit{inf}(f_S(s), f_O(o))$, where $f_S(s)$ and $f_O(o)$ are the integrity levels before the operation.



N.B.: the integrity level $\mathit{inf}(f_S(s), f_O(o))$ is the greatest lower bound of $f_S(s)$ and $f_O(o)$ in the underlying lattice of integrity levels.

Biba: Dynamic Integrity Levels (cont.)

Object low watermark property: s can modify an o at any integrity level.
The new integrity level of o is $\mathit{inf}(f_S(s), f_O(o))$, where $f_S(s)$ and $f_O(o)$ are the integrity levels before the operation.



Biba: Policies for Invocation

Biba can be extended with an operation invoke, so that a subject can invoke another subject, e.g. a software tool, to access an object.

- If we want to ensure that invocation does not bypass the mandatory integrity policies, we could add

  Invoke property: s_1 can invoke s_2 only if $f_S(s_2) \leq f_S(s_1)$.

  That is, subjects are only allowed to invoke tools at lower levels.

- Alternatively:

  Ring property: a subject s_1 can read objects at all integrity levels, but it can only modify objects o with $f_O(o) \leq f_S(s_1)$ and it can invoke a subject s_2 only if $f_S(s_1) \leq f_S(s_2)$.

These two properties are inconsistent, and one must look at the application to decide which is more appropriate.

Outline

- Access Control (AC)
- Discretionary Access Control (DAC)
- Access Control Matrix Model
- Mandatory Access Control (MAC)
- Bell-LaPadula Model and Variants

☞ **Role-Based Access Control (RBAC)**

Why RBAC?

- How do we formalize a policy when there are $10^3 - 10^6$ subjects and objects? AC matrices do not scale!
- Overcome using standard tricks: abstraction and hierarchy.

  Abstraction: Many subjects (or objects) have identical attributes, and policy is based on these attributes.

  Hierarchy: Often functional/organizational hierarchies that determine access rights.
- Approach to RBAC: decompose subject/object relationship by introducing a set of roles. Then assign subjects to roles and permissions to objects based on role. I.e.,

  $(s, o, p) \in \mathcal{P}$ iff s has role r and r has permission p on object o.
- This idea can be generalized by introducing a hierarchy on roles.

Role-Based Access Control (RBAC)

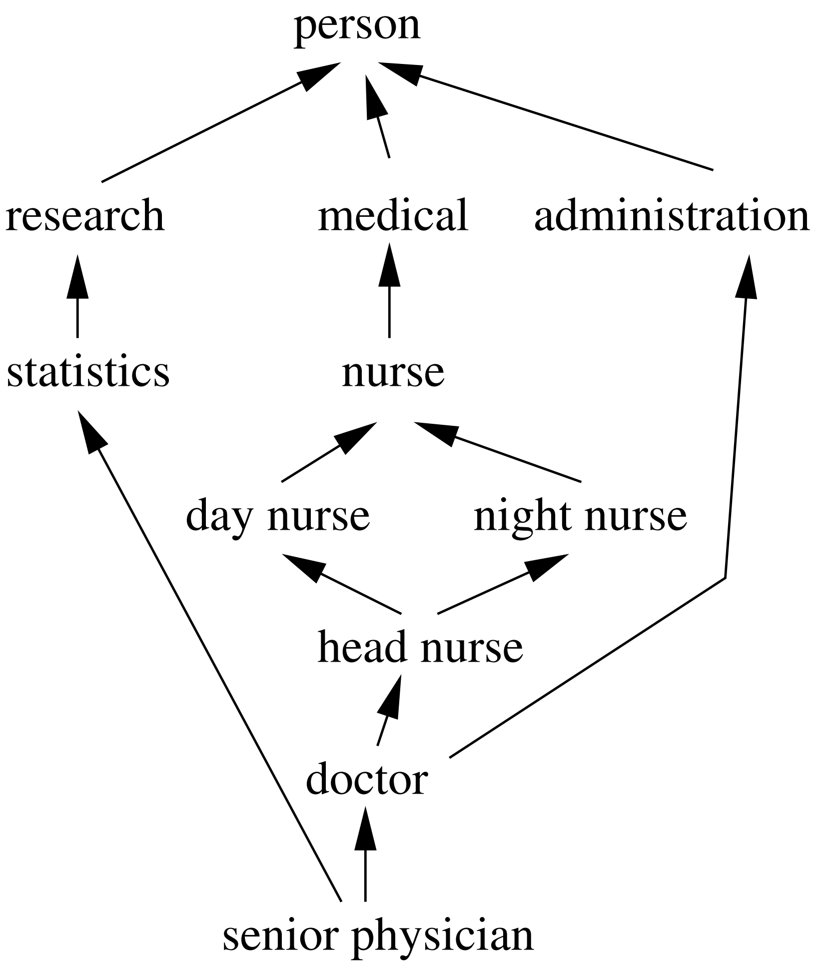
- Rights are associated with roles, and users are made members of appropriate roles.

  ⇒ Simpler management of rights:

  - Access decisions based on roles that users have as part of an organization (e.g. hospital).
  - Roles can have overlapping responsibilities and rights.
  - Roles can be updated without updating the rights of every user on individual basis.
  - Enterprise-specific security policies.

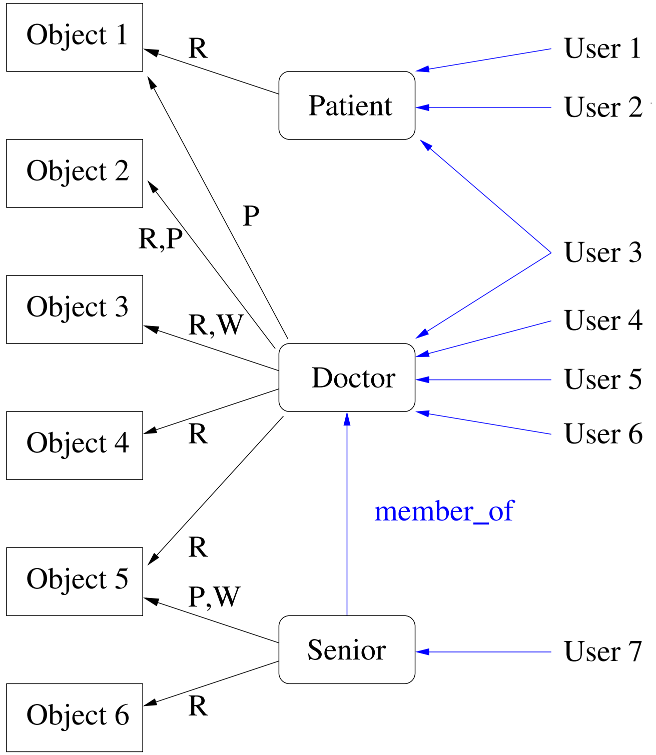
- Closely related to concept of user groups: a role brings together

  - a set of users on one side (as in groups) and
  - a set of rights.



Role-Based Access Control (RBAC) (cont.)

- Role hierarchies simplify policy expression.
- Example:
  - A member of role Senior has also all permissions defined by Doctor.
  - A Senior may delegate a task to a Doctor.
  - A member of roles Doctor or Patient can only access those resources allowed under his role(s).
- Needed by enforcement mechanism:
  - Rules for role assignment/authorization, and for permission assignment.
  - Also: rules for delegation.



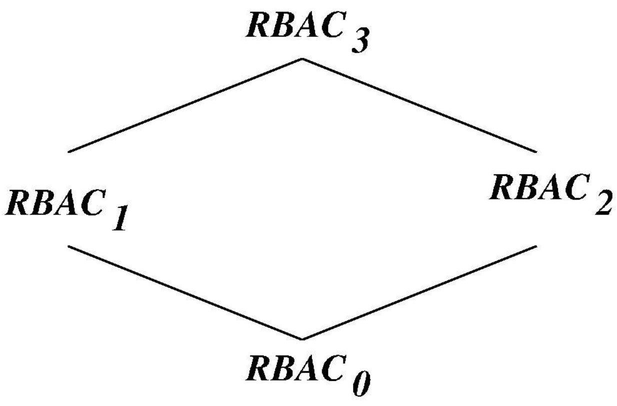
RBAC formalization: overview

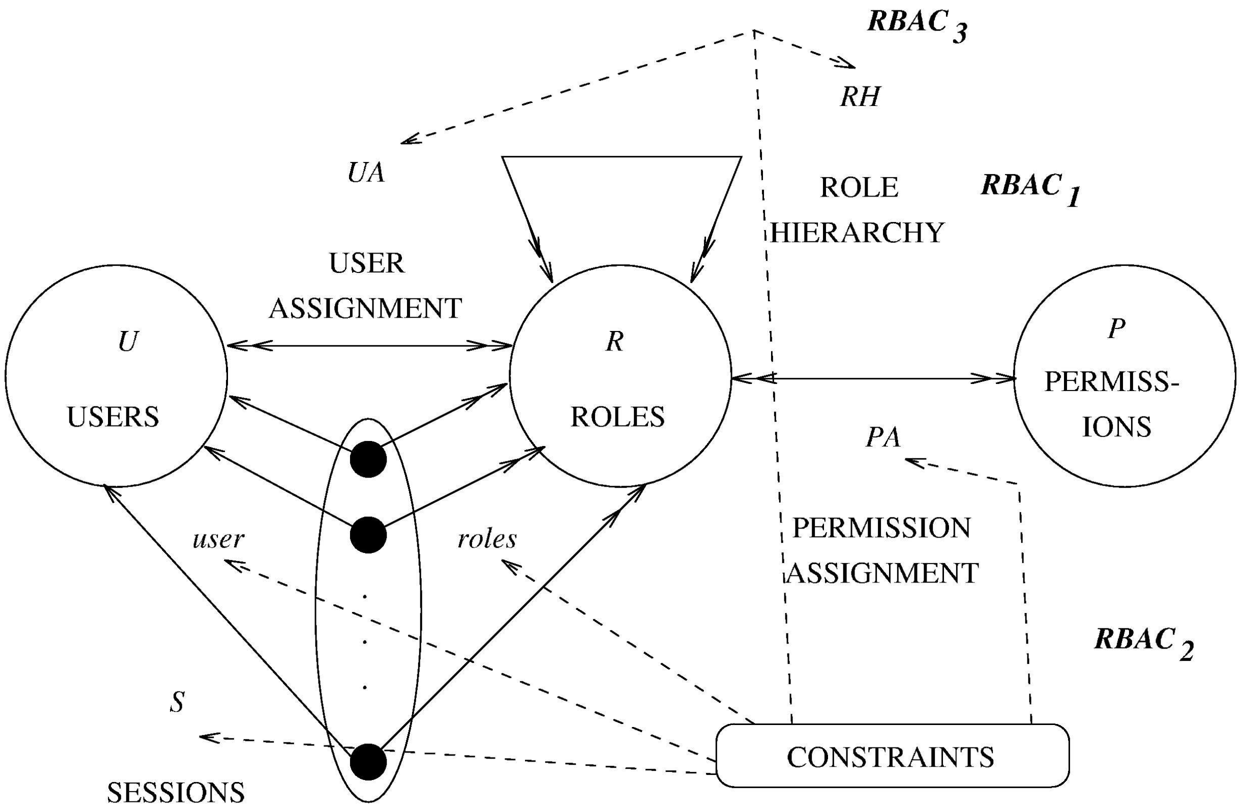
$RBAC_0$: plain

$RBAC_1$: with role hierarchy

$RBAC_2$: with constraints

$RBAC_3$: with both





RBAC formalization: $RBAC_0$ and $RBAC_1$

$RBAC_0$: plain

users: U

roles: R

permissions: P

user assignment: $UA \subseteq U \times R$

permission assignment: $PA \subseteq R \times P$

sessions: $AR \subseteq UA$ (active roles, note the restriction!)

access: $can\_exec = AR \circ PA \subseteq U \times P$,

i.e. $(u,p) \in can\_exec = \exists r.\ (u,r) \in AR \wedge (r,p) \in PA$

$RBAC_1$: with role hierarchy

role hierarchy: $RH \subseteq R \times R$, antisymmetric

sessions: $AR \subseteq UA \ \circ \ RH^*$ (redefined active roles)

where $X^* = I \cup X \cup X \circ X \cup \ldots$ is the reflexive-transitive closure

RBAC formalization: $RBAC_2$

$RBAC_2$: with constraints, for instance:

static separation of duty: $SSD \subseteq R \times R$

example: $(treasurer, auditor) \in SD$

constraint: $UA^{-1} \circ UA \subseteq \overline{SSD \cup SSD^{-1}}$, i.e.

$((u,r) \in UA \wedge (u,r') \in UA) \longrightarrow ((r,r') \notin SSD \wedge (r',r) \notin SSD)$

where $Z^{-1} = \{(y,x).\ (x,y) \in Z\}$ is inversion,

$\overline{Z} = \{(x,y).\ (x,y) \notin Z\}$ is complementation

dynamic separation of duty: $DSD \subseteq R \times R$

example: $(customer, customer\ consultant) \in DSD$

constraint: $AR^{-1} \circ AR \subseteq \overline{DSD \cup DSD^{-1}}$

cardinality constraints: e.g. $|\{u.\ (u, branch\ manager) \in UA\}| \leq 1$

prerequisite permissions: e.g.

$((clerk, r) \in RH \wedge (r, write) \in PA) \longrightarrow (r, read) \in PA$

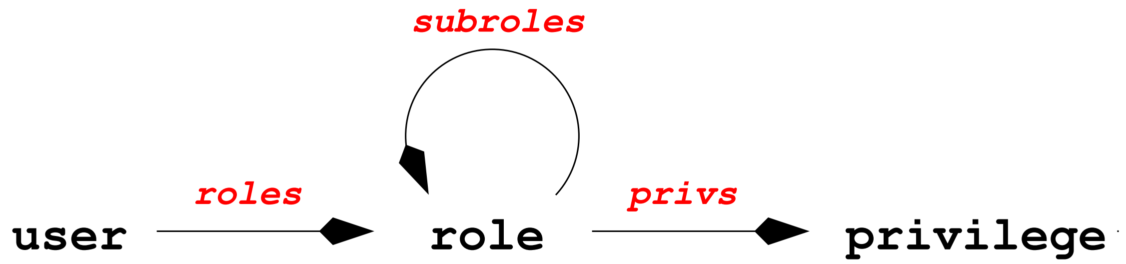
RBAC example: complex information system

Privileges:

$\mathit{roles} \subseteq \mathtt{user} \times \mathtt{role}$

$\mathit{subroles} \subseteq \mathtt{role} \times \mathtt{role}$

$\mathit{privs} \subseteq \mathtt{role} \times \mathtt{privilege}$



$$(u,p) \in \mathit{roles} \circ \mathit{subroles}^* \circ \mathit{privs}$$

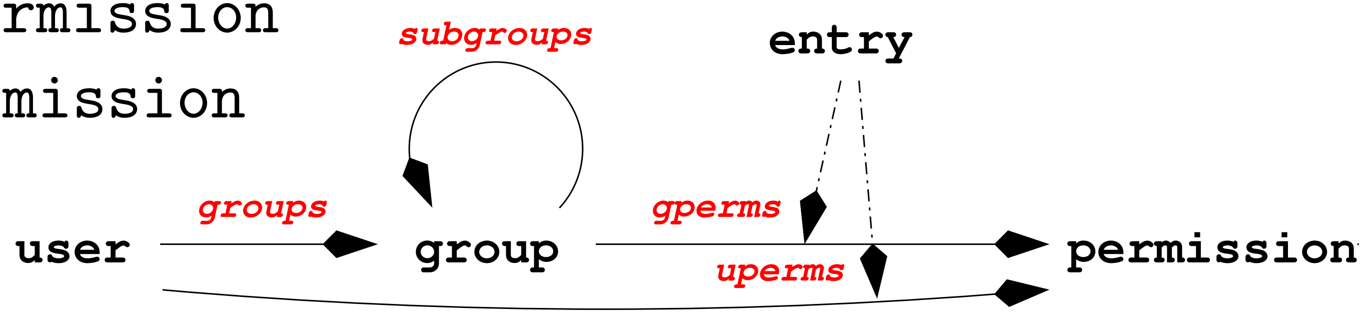
Permissions:

$\mathit{groups} \subseteq \mathtt{user} \times \mathtt{group}$

$\mathit{subgroups} \subseteq \mathtt{group} \times \mathtt{group}$

$\mathit{gperms} \subseteq \mathtt{group} \times \mathtt{permission}$

$\mathit{uperms} \subseteq \mathtt{user} \times \mathtt{permission}$



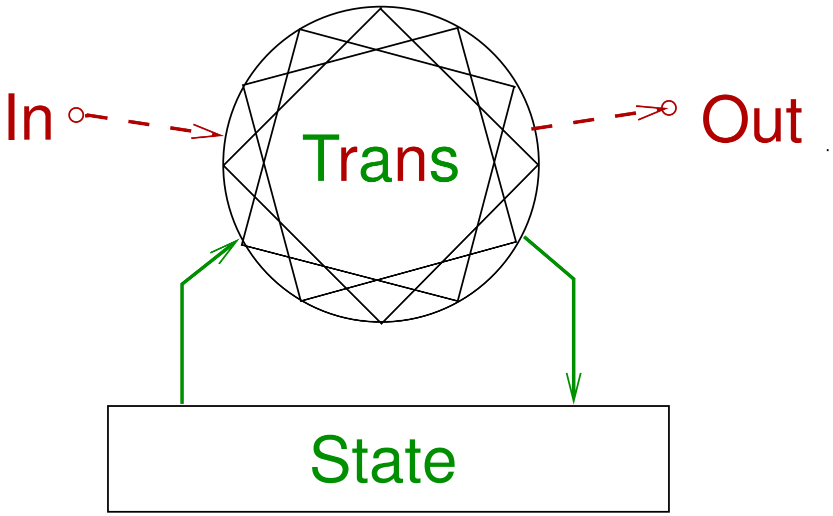
$$(u,p) \in (\mathit{groups} \circ \mathit{subgroups}^* \circ \mathit{gperms}(e)) \ \cup \ \mathit{uperms}(e)$$

Automata

☞ **Input/Output Automata (IOAs)**

- AutoFocus Automata
- Interacting State Machines (ISMs)

Input/Output Automata (IOAs)



- each reactive system component modeled as an automaton
- state machine with actions
- transitions may be nondeterministic
- input actions cannot be blocked
- other actions under control of automaton
- automata can be composed, forming new automata
- communication via synchronized actions
- strong metatheory: refinement, compositionality, . . .

IOAs: action signatures

Interface between an automaton and its environment:
action signature S, consisting of disjoint sets

$in(S)$: input actions

$out(S)$: output actions

$int(S)$: internal actions

Derived notions:

$acts(S) = in(S) \cup out(S) \cup int(S)$: all actions

$ext(S) = in(S) \cup out(S)$: external actions

$local(S) = out(S) \cup int(S)$: locally-controlled actions

IOAs: automata

An I/O automaton A consists of

$sig(A)$: action signature

$states(A)$: set of states

$start(A) \subseteq states(A)$: initial states (at least one)

$steps(A) \subseteq states(A) \times acts(A) \times states(A)$: transition relation
input enabled: $\forall \sigma.\ \forall a \in in(A).\ \exists \sigma'.\ (\sigma, a, \sigma') \in steps(A)$

$part(A) \subseteq \wp(local(A))$: countable partitioning
(used for expressing fairness, which is not an issue here)

IOAs: coffee machine CM

$in(S_1) = \{PUSH_1, PUSH_2\}$: buttons received

$out(S_1) = \{COFFEE, ESPRESSO, DOPPIO\}$

$int(S_1) = \{LOOSE\}$

$sig(CM) = S_1$

$states(CM) = \mathbb{N}$: variable 'button-pushed'

$start(A) = \{0\}$: initially, no button pushed

$$steps(A) = \{(x, PUSH_1, 1),\ (x, PUSH_2, 2),\ (x, LOOSE, 0),\\ (1, COFFEE, 0),\ (2, ESPRESSO, 0),\ (2, DOPPIO, 0)\\ \mid x \in states(CM)\}$$

IOAs: user $USER$

$in(S_2) = \{COFFEE, ESPRESSO, DOPPIO\}$

$out(S_2) = \{PUSH_1, PUSH_2\}$: buttons pushed

$int(S_2) = \emptyset$

$sig(USER) = S_2$

$states(USER) = \mathbb{B} \times \mathbb{B}$: variables '`waiting`', '`doppio`'

$start(A) = \{(F, F)\}$: not waiting and no doppio received

$$steps(A) = \{((F,T), PUSH_1, (T,T)),\ ((F,F), PUSH_2, (T,F)),\\ ((w,d), COFFEE, (F,d)),\ ((w,d), ESPRESSO, (F,d)),\\ ((w,d), DOPPIO, (F,T)) \mid w,d \in \mathbb{B}\}$$

IOAs: execution

execution fragment of A: a finite sequence σ_0, a_1, σ_1, . . . , a_n, σ_n or an infinite sequence σ_0, a_1, σ_1, . . . of states and actions of A such that $\forall i.\ (\sigma_i, a_{i+1}, \sigma_{i+1}) \in step(A)$

$execs(A)$: execution fragments beginning with some $\sigma_0 \in start(A)$

$finexecs(A) \subseteq execs(A)$: finite executions of A

$reachable(A)$: the final states σ_n of all finite executions of A

$sched(\alpha)$: the subsequence of *actions* in execution fragment α

$(fin)scheds(A)$: schedules of all (finite) executions of A

$beh(\alpha)$: the subsequence of *external* actions in execution fragment α

$(fin)behs(A)$: behaviors of all (finite) executions of A

Note: traces ($(fin)execs$, $(fin)scheds$, and $(fin)beh$) are prefix-closed.

IOAs: coffee machine executions

execution fragment of CM:

$$\alpha = [1, COFFEE, 0, PUSH_2, 2, LOOSE, 0, PUSH_1, 1]$$

$$execs(CM) = \{[0, PUSH_2, 2], [0, PUSH_1, \alpha, (PUSH_1, 1)^*], \ldots\}$$

$$finexecs(CM) = \{[0], [0, PUSH_2, 2], [0, PUSH_1, \alpha], \ldots\}$$

$$reachable(CM) = \{0, 1, 2\}$$

$$sched(\alpha) = [COFFEE, PUSH_2, LOOSE, PUSH_1]$$

$$(fin)scheds(CM) = \\ \{[], [PUSH_2], [LOOSE, PUSH_1, COFFEE], \ldots\}$$

$$beh(\alpha) = [COFFEE, PUSH_2, PUSH_1]$$

$$behs(CM) = \{[], [PUSH_2], [PUSH_1, COFFEE], \ldots\}$$

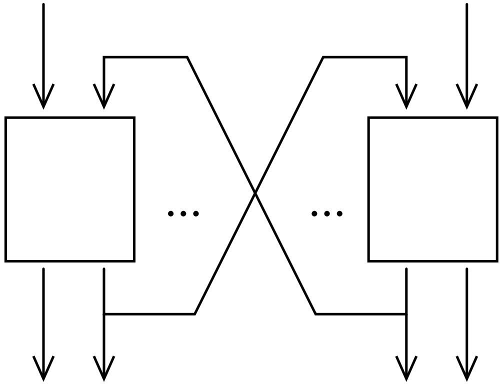
IOAs: composition of signatures

A countable collection $\{S_i\}_{i \in I}$ of action signatures is strongly compatible iff

- $out(S_i) \cap out(S_j) = \emptyset$ for all $i \neq j \in I$
- $int(S_i) \cap acts(S_j) = \emptyset$ for all $i \neq j \in I$
- no action is contained in infinitely many $acts(S_i)$ for all $i \in I$

The composition $\Pi_{i \in I} S_i$ of a countable collection of strongly compatible action signatures $\{S_i\}_{i \in I}$ is an action signature S with

$$\begin{aligned} in(S) &= \cup_{i \in I}\, in(S_i) - \cup_{i \in I}\, out(S_i) \\ out(S) &= \cup_{i \in I}\, out(S_i) - \emptyset \\ int(S) &= \cup_{i \in I}\, int(S_i) \end{aligned}$$

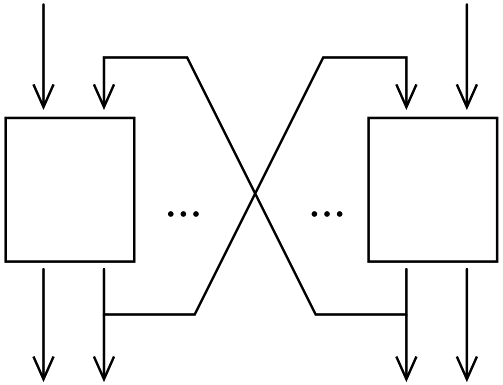


IOAs: composition of automata

The composition $\Pi_{i \in I} A_i$ of a countable collection of strongly compatible automata $\{A_i\}_{i \in I}$ is an automata A with

$$sig(A) = \Pi_{i \in I} sig(A_i)$$
$$states(A) = \Pi_{i \in I} states(A_i)$$
$$start(A) = \Pi_{i \in I} start(A_i)$$
$$steps(A) = \{(\sigma, a, \sigma') \mid \text{if } a \in acts(A_i) \text{ then } (\sigma[i], a, \sigma'[i]) \in steps(A_i) \ \text{ else } \sigma[i] = \sigma'[i],\ i \in I\}$$
$$part(A) = \cup_{i \in I} part(A_i)$$

A is input-enabled since all A_i are.



IOAs: coffee session

Compose CM and $USER$ as $CS = CM \times USER$.

$sig(CM)$ and $sig(USER)$ are strongly compatible because

- $out(CM) \cap out(USER) = \emptyset$
- $int(CM) \cap acts(USER) = \emptyset$ and $int(USER) \cap acts(CM) = \emptyset$
- no action is contained in infinitely many $\{acts(CM), acts(USER)\}$

The composition $CS = CM \times USER$ has the components

$sig(CS)$ **=** $sig(CM) \times sig(USER)$ having the components

$in(sig(CS))$ **=** $EA - EA = \emptyset$

$out(sig(CS))$ **=** EA where

$EA = \{PUSH_1, PUSH_2, COFFEE, ESPRESSO, DOPPIO\}$

$int(sig(CS))$ **=** $\{LOOSE\}$

$states(CS)$ **=** $\mathbb{N} \times \mathbb{B} \times \mathbb{B}$

$start(CS)$ **=** $(0, F, F)$, $steps(CS) = \ldots$

IOAs: execution and composition

The projection $\alpha|A_i$ of an execution fragment $\alpha = \sigma_0, a_1, \sigma_1, \ldots$ of a composition $\Pi_{i\in I}A_i$ is the sequence obtained from α by

- deleting those a_j, σ_j for which $a_j \notin acts(A_i)$
- replacing all remaining σ_j by their i-th component $\sigma_j[i]$

Proposition: Let $\{A_i\}_{i\in I}$ be a countable collection of strongly compatible automata and $A = \Pi_{i\in I}A_i$.
If $\alpha \in execs(A)$ then $\alpha|A_i \in execs(A_i)$ for every $i \in I$.
The same holds for $finexecs(A)$, $scheds(A)$, $finscheds(A)$, $behs(A)$, and $finbehs(A)$.

Examples: $\alpha = [(0,F,F), PUSH_2, (2,T,F), DOPPIO,$
$(0,F,T), PUSH_1, (1,T,T), LOOSE, (0,T,T)]$

$\alpha|CM = [0, PUSH_2, 2, DOPPIO, 0, PUSH_1, 1, LOOSE, 0]$

$beh(\alpha)|USER = [PUSH_2, DOPPIO, PUSH_1]$

IOAs: specification and refinement

A safety specification $\mathcal{P}$ is a prefix-closed set of action sequences.

An automaton A implements a specification $\mathcal{P}$ iff $finbehs(A) \subseteq \mathcal{P}$.

An automaton A implements an automaton A' with the same external signature iff $finbehs(A) \subseteq finbehs(A')$.

Examples: $\mathcal{P}_1 =$ sequences of actions from $\{PUSH_1, PUSH_2, COFFEE, ESPRESSO, DOPPIO\}$ where each $COFFEE$ is immediately preceded by $PUSH_1$.

Does CM implement $\mathcal{P}_1$? Yes. Coffee is given only promptly on request.

Does $USER$ implement $\mathcal{P}_1$? No. He may receive coffee anytime.

CM is implemented by CM' which is like CM but never gives a doppio.
Frustrating to the $USER$:
$behs(CM' \times USER) =$ all prefixes of $[(PUSH_2, ESPRESSO)^*]$

IOAs: compositionality

Let A be an automaton and $\mathcal{P}$ be a safety specification with actions from Φ where $\Phi \cap int(A) = \emptyset$. A preserves $\mathcal{P}$ iff
$\forall\beta.\ \beta a|A \in finbehs(A) \wedge a \in out(A) \wedge \beta|\Phi \in \mathcal{P} \longrightarrow \beta a|\Phi \in \mathcal{P}$.

Example: CM preserves $\mathcal{P}_1$ and $USER$ preserves $\mathcal{P}_1$.

Theorem 1: Let $\{A_i\}_{i\in I}$ be a countable collection of strongly compatible automata and $A = \Pi_{i\in I} A_i$ such that $in(A) = \emptyset$.
Let $\mathcal{P}$ be a safety specification over $ext(A)$.
If every A_i preserves $\mathcal{P}$, then A implements $\mathcal{P}$.

Example: CS implements $\mathcal{P}_1$.

Theorem 2: Let $\{A_i\}_{i\in I}$ and $\{B_i\}_{i\in I}$ be countable collections of strongly compatible automata.
If A_i implements B_i for all i, then $\Pi_{i\in I} A_i$ implements $\Pi_{i\in I} B_i$.

Example: $CM' \times USER$ implements CS.

IOAs: papers

- N. Lynch and M. Tuttle: An introduction to Input/Output Automata. CWI Quarterly 2(3):219-246, 1989.

- S. Garland and N. Lynch: The IOA Language and Toolset: Support for Designing, Analyzing, and Building Distributed Systems. MIT/LCS/TR-762, 1998.

- O. Müller: A Verification Environment for I/O Automata Based on Formalized Meta-Theory. PhD thesis, TU München, 1998.

Automata

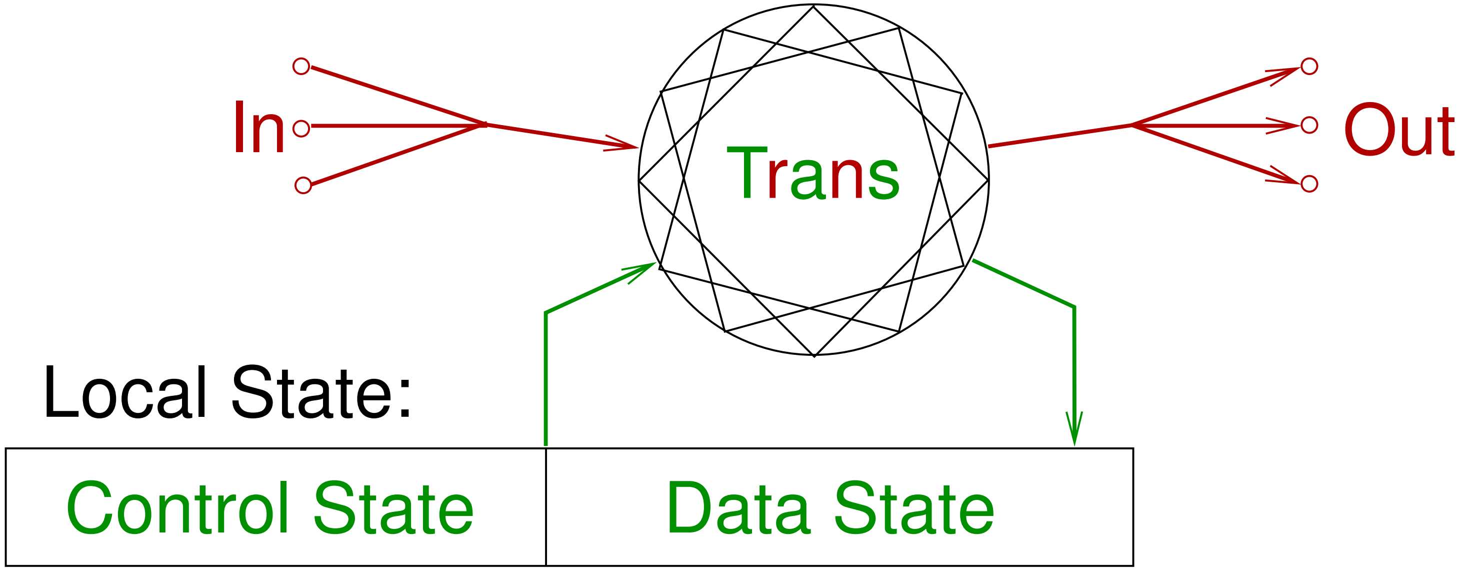
- Input/Output Automata (IOAs)

☞ **AutoFocus Automata**

- Interacting State Machines (ISMs)

AutoFocus Automata

Automata with (nondeterministic) state transitions +
clock-synchronous i/o simultaneously on multiple connections
Automata may be hierarchical



Functional language for types and expressions

AutoFocus Toolset

Graphical browser/editor with version control by TUM

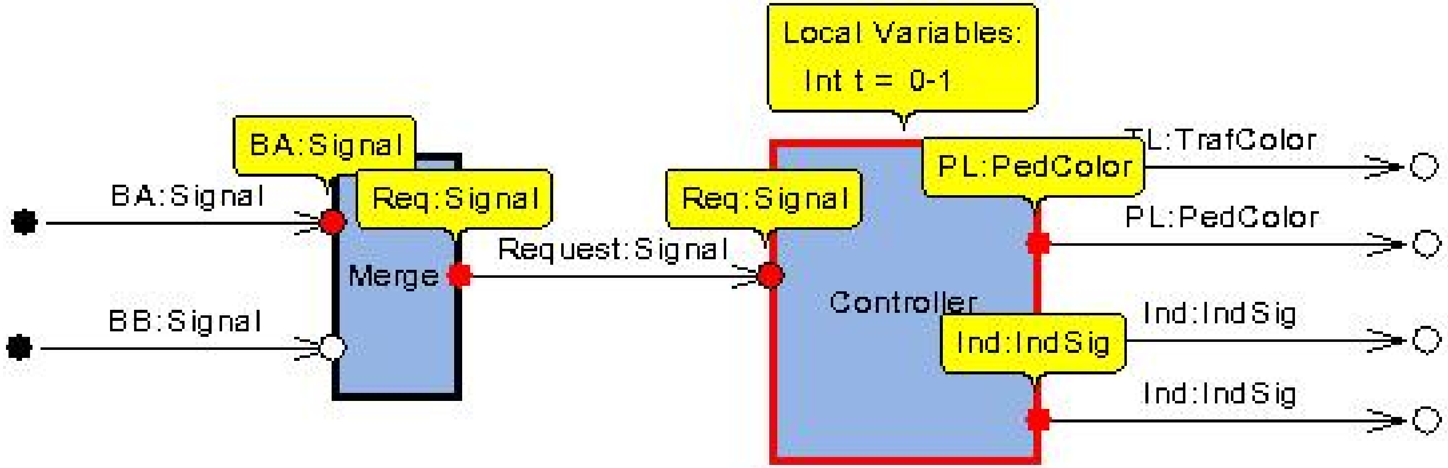
Modelchecking/testing/simulation tools by Validas

Code generators by Validas

- First Prize in competition at Formal Methods 1999
- Homepage: `autofocus.in.tum.de`

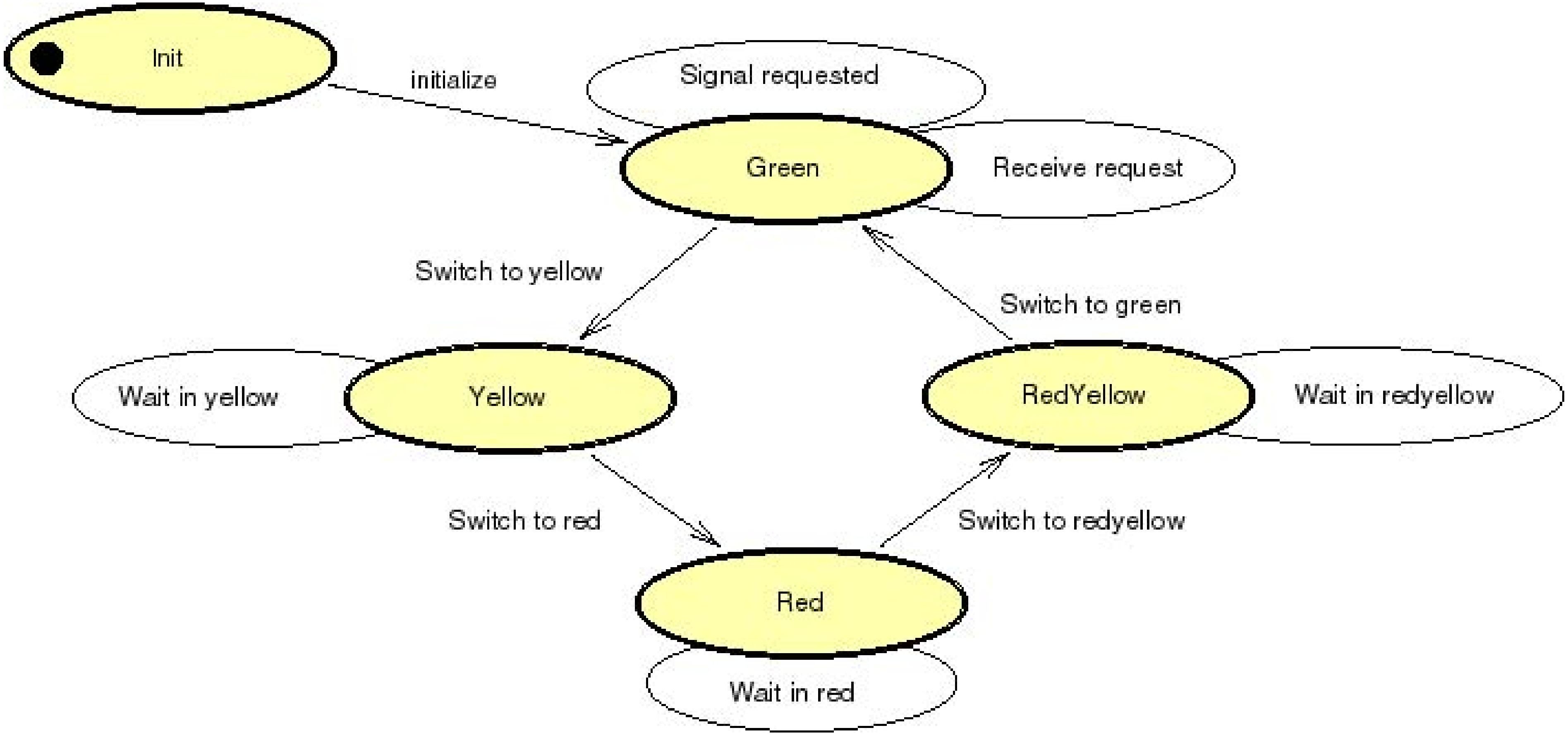
System Structure Diagrams (SSDs)

defining components with local variables, interfaces, and connections



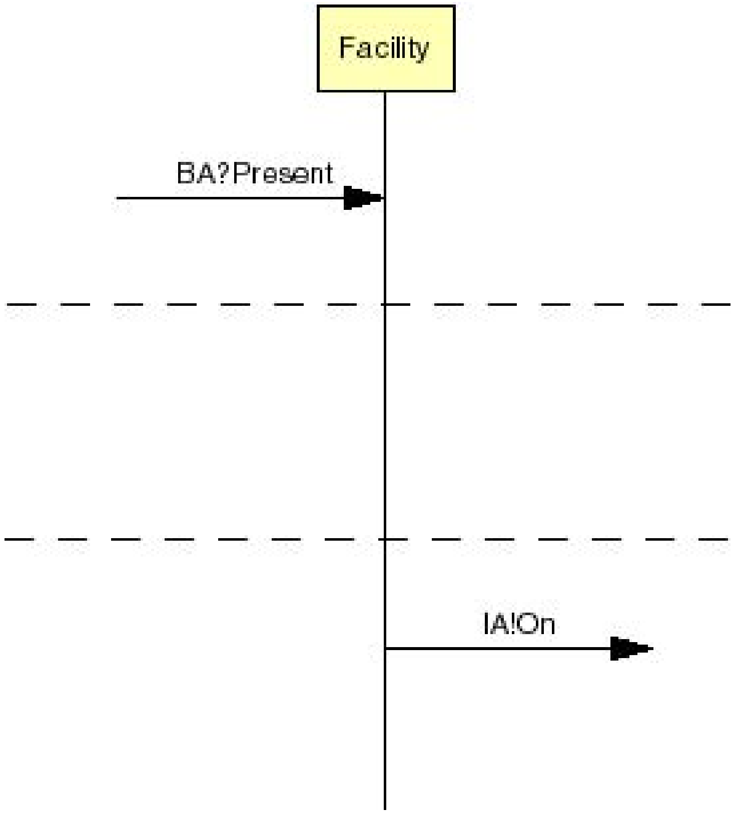
State Transition Diagrams (STDs)

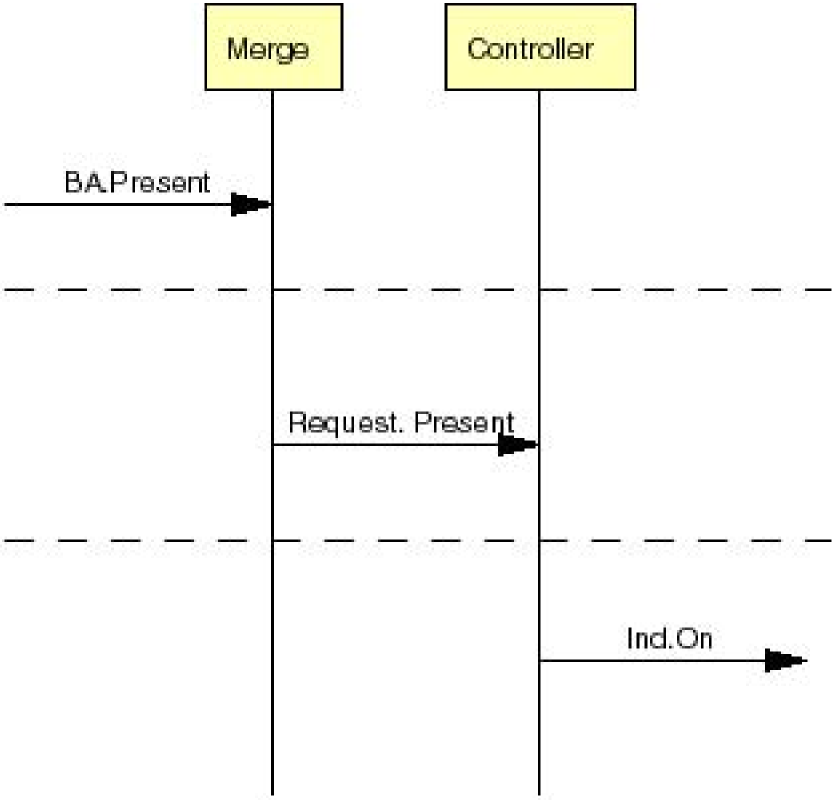
defining preconditions, input, output, and effects of transitions



Extended Event Traces (EETs)

describing the event order for exemplary executions and test cases





Automata

- Input/Output Automata (IOAs)
- AutoFocus Automata

☞ **Interacting State Machines (ISMs)**

Requirements

Expressiveness: state transitions, concurrency, asynchronous messages
$\rightsquigarrow$ applicable to a large variety of reactive systems

Ease of modeling: systems describable directly

Simplicity: minimum of expertise and time required

Flexibility: adaptation and extension

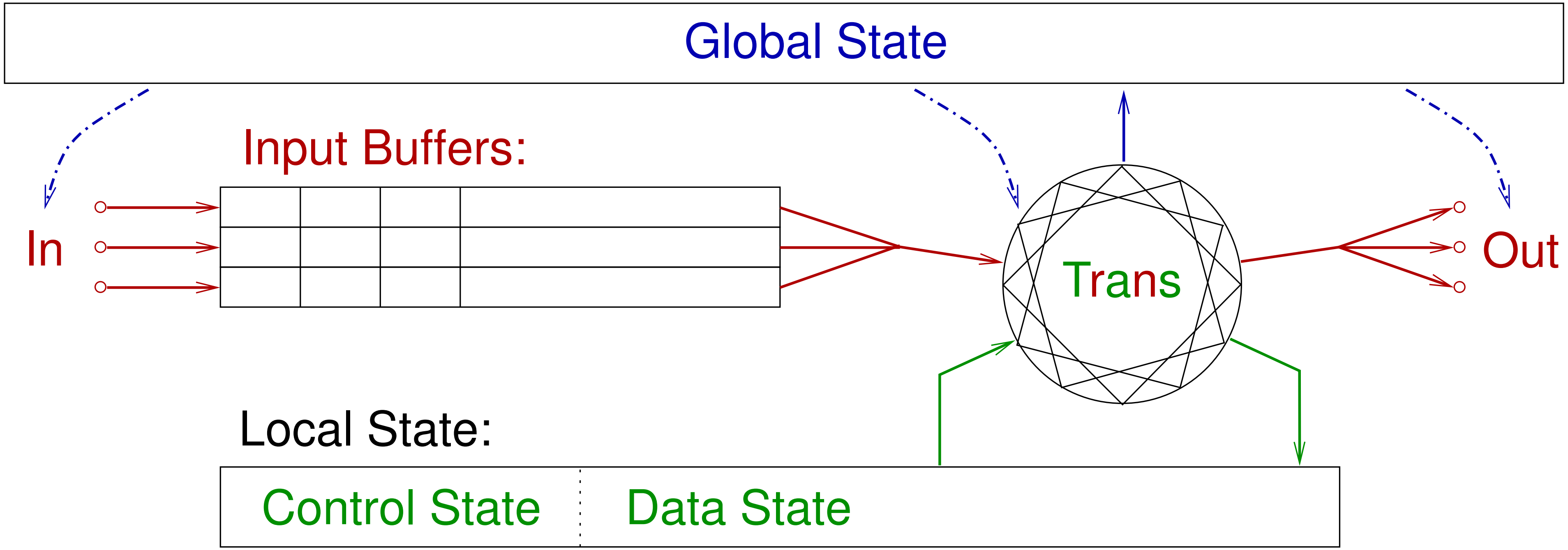
Strength of the semantics: refinement, compositionality, . . .

Graphical capabilities: overview and intuition

Tool support: mature and freely available (including sources)

Interacting State Machines (ISMs)

Automata with (nondeterministic) state transitions +
buffered i/o simultaneously on multiple connections
ISM system may depend on global state



Transitions defined in executable and/or axiomatic style
Finite executions only ($\rightsquigarrow$ no liveness properties)

ISM Framework

AutoFocus:

Syntactic perspective
Graphical documentation
Type and consistency checks

Isabelle/HOL:

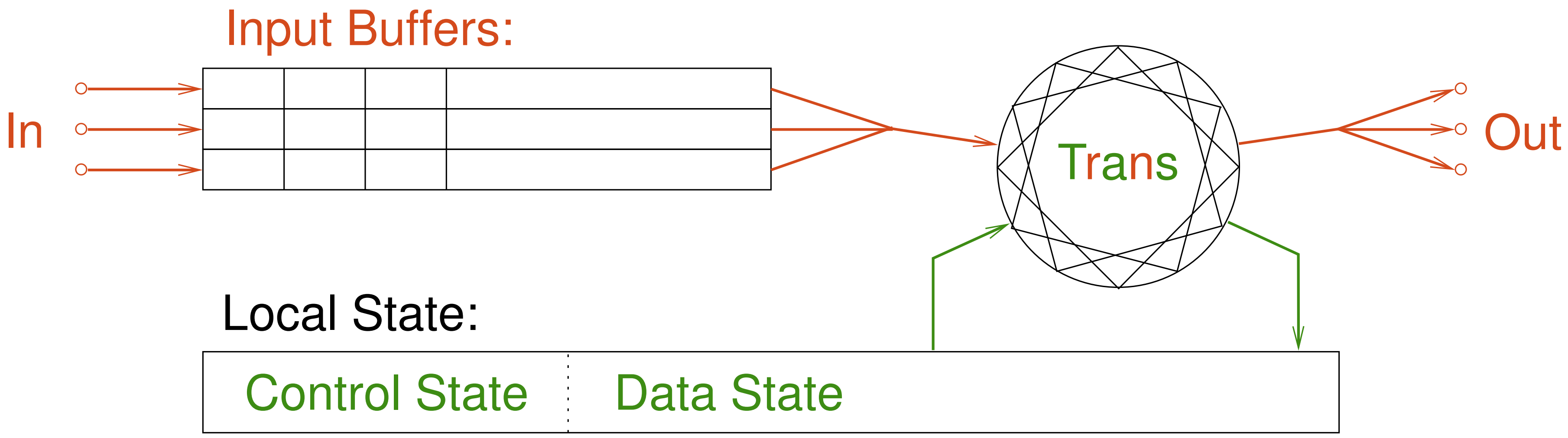
Semantic perspective
Textual documentation
Validation and correctness proofs

AutoFocus drawing $\longrightarrow$ Quest file $\xrightarrow{\mathrm{Conv}_1}$ Isabelle theory file

Within Isabelle: **ism** sections $\xrightarrow{\mathrm{Conv}_2}$ Standard HOL definitions

Elementary ISMs

$MSGs = \mathcal{P} \to \mathcal{M}^*$	family of messages $\mathcal{M}$, indexed by port names $\mathcal{P}$
$CONF(\Sigma) = MSGs \times \Sigma$	configuration with local state Σ
$TRANS(\Sigma) = \wp((MSGs \times \Sigma) \times (MSGs \times \Sigma))$	transitions
$ISM(\Sigma) = \wp(\mathcal{P}) \times \wp(\mathcal{P}) \times \Sigma \times TRANS(\Sigma)$	ISM type
$a = (In(a),\, Out(a),\, \sigma_0(a),\, Trans(a))$	ISM value



Input Buffers:

In

Trans

Out

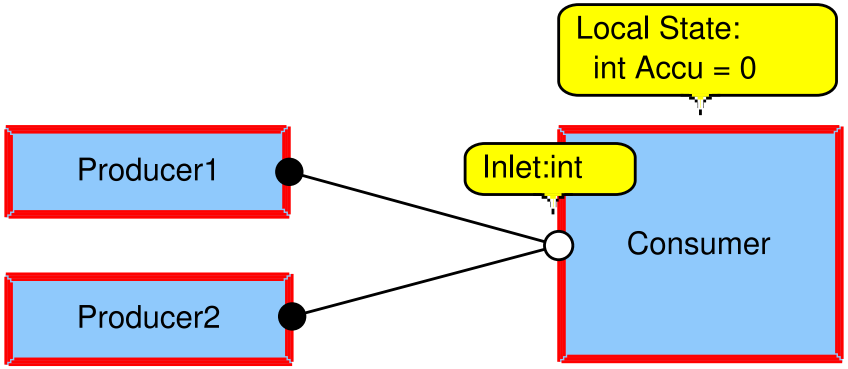
Local State:

Control State

Data State

Producer-Consumer Example

Two producers sending random integer values to a port named $Inlet$ of a consumer which sums them up in a local variable named $Accu$



$$\mathcal{P} = \{Inlet\}$$
$$\mathcal{M} = \mathbb{Z}$$
$$MSGs = \{Inlet\} \to \mathbb{Z}^*$$

$$Producer_i = (\emptyset,\ \{Inlet\},\ \bullet,\ \{((\textcurrency, \bullet), (\textcurrency(Inlet := \langle n \rangle), \bullet)) \mid n \in \mathbb{Z}\})$$
$$Consumer = (\{Inlet\},\ \emptyset,\ 0,$$
$$\{((\textcurrency(Inlet := \langle n \rangle), a), (\textcurrency, a + n)) \mid n, a \in \mathbb{Z}\})$$

where $\textcurrency = \lambda p.\ \langle\rangle$ and $m(X := s) = \lambda p.$ if $p = X$ then s else $m(p)$

Composite Runs

Let $A = (A_i)_{i \in I}$ be a family of ISMs. The set of *composite runs* $CRuns(A)$ of type $\wp((CONF(\Pi_{i\in I}\Sigma_i))^*)$ is inductively defined as

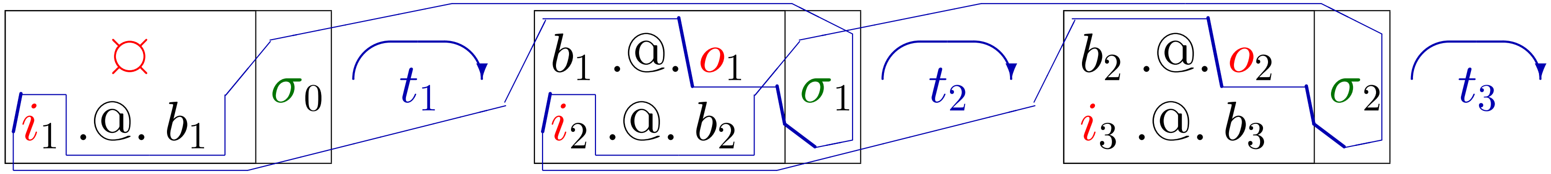
$$\frac{}{\langle(\text{¤}, \Pi_{i\in I}\,\sigma_0(A_i))\rangle \in CRuns(A)}$$

$$\frac{\begin{array}{c} j \in I \\ cs^\frown(i \;.@.\; b, S[j:=\sigma]) \in CRuns(A) \\ ((i,\sigma),(o,\sigma')) \in Trans(A_j) \end{array}}{cs \frown (i \;.@.\; b, S[j:=\sigma]) \frown (b \;.@.\; o, S[j:=\sigma']) \in CRuns(A)}$$

where .@. concatenates message families on a port by port basis:

$m \;.@.\; n = \lambda p.\; m(p) \,@\, n(p)$, *e.g.*

$(\text{¤}(Inlet := \langle 1, -3\rangle)) \;.@.\; (\text{¤}(Inlet := \langle 6\rangle)) \;=\; \text{¤}(Inlet := \langle 1, -3, 6\rangle)$

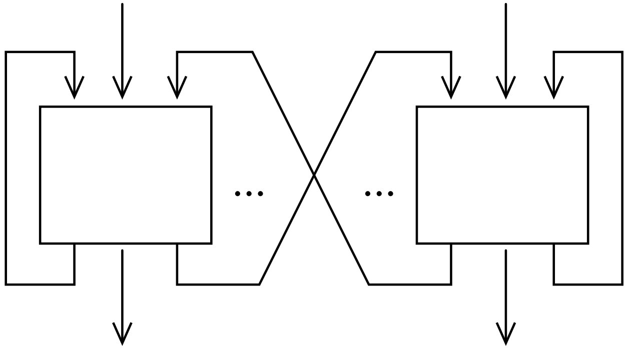


Parallel composition of ISMs

Let $A = (A_i)_{i\in I}$ be a family of ISMs. Their *parallel composition* $\|_{i\in I} A_i$ is an ISM of type $\mathit{ISM}(\mathit{CONF}(\Pi_{i\in I}\Sigma_i))$ is defined as

$$(\mathit{AllIn}(A)\backslash\mathit{AllOut}(A),\ \mathit{AllOut}(A)\backslash\mathit{AllIn}(A),\ (\between, S_0(A)),\ \mathit{PTrans}(A))$$

where



- $\mathit{AllIn}(A) = \bigcup_{i\in I} \mathit{In}(A_i)$
- $\mathit{AllOut}(A) = \bigcup_{i\in I} \mathit{Out}(A_i)$
- $S_0(A) = \Pi_{i\in I}\,\sigma_0(A_i)$ is the Cartesian product of all initial local states
- $\mathit{PTrans}(A)$ of type $\mathit{TRANS}(\mathit{CONF}(\Pi_{i\in I}\Sigma_i))$ is the parallel composition of their transition relations, defined as . . .

Parallel transition relation

$$\frac{\begin{gathered} j \in I \\ ((i, \sigma), (o, \sigma')) \in \mathit{Trans}(A_j) \end{gathered}}{\begin{gathered} ((i_{|\overline{\mathit{AllOut}(A)}}, (i_{|\mathit{AllOut}(A)}\ .@.\ b, S[j := \sigma\ ])), \\ (o_{|\overline{\mathit{AllIn}(A)}},\ (b\ .@.\ o_{|\mathit{AllIn}(A)}, S[j := \sigma']))) \in \mathit{PTrans}(A) \end{gathered}}$$

where

- $S[j := \sigma]$ is the replacement of the j-th component of the tuple S by σ
- $m_{|P}$ denotes the restriction $\lambda p.$ if $p \in P$ then $m(p)$ else $\langle\rangle$ of the message family m to the set of ports P
- $o_{|\overline{\mathit{AllIn}(A)}}$ denotes those parts of the output o provided to any outer ISM
- $o_{|\mathit{AllIn}(A)}$ denotes the internal output to peer ISMs or direct feedback, which is added to the current buffer contents b

Producer-Consumer Example: Composition & Run

$$I = \{1, 2, 3\},\ A_1 = Producer_1,\ A_2 = Producer_2,\ A_3 = Consumer$$

$$\begin{aligned}\Sigma &= \Pi_{i \in I} \Sigma_i = \mathbb{Z} \\ A &= \|_{i \in I} A_i \ = \ (\emptyset,\ \emptyset,\ (\text{¤}, 0),\ PCT) \text{ where}\end{aligned}$$

$$\begin{aligned} PCT = \{((\text{¤},\ (b, a)),\ (\text{¤},\ (b\ .@.\ \text{¤}(Inlet := \langle n \rangle), a))) \\ \mid n, a \in \mathbb{Z} \wedge b \in MSGs\} \\ \cup\ \{((\text{¤},\ (\text{¤}(Inlet := \langle n \rangle)\ .@.\ b, a)),\ (\text{¤},\ (b, a + n))) \\ \mid n, a \in \mathbb{Z} \wedge b \in MSGs\} \end{aligned}$$

A possible trace is

$$\begin{aligned}&\langle(\text{¤}, 0), \\ &(\text{¤}(Inlet := \langle 1 \rangle), 0)),\ (\text{¤}(Inlet := \langle 1, -3 \rangle), 0)), \\ &(\text{¤}(Inlet := \langle -3 \rangle), 1)),\ (\text{¤}, -2), \\ &(\text{¤}(Inlet := \langle 6 \rangle), -2)),\ (\text{¤}, 4)\rangle\end{aligned}$$

ISM definition in Isabelle/HOL

ism *name* **=**
  ports *pn_type*
    inputs *I_pns*
    outputs *O_pns*
  messages *msg_type*
  states *[state_type]*
  *[***control** *cs_type [***init** *cs_expr0]]*
  *[***data** *ds_type [***init** *ds_expr0] [***name** *ds_name]]*
 *[***transitions**
  *(tr_name [attrs]]***:** *[cs_expr (***->** *|* $\rightarrow$*) cs_expr']*
  *[***pre** *(bool_expr)*$^+$*]*
  *[***in** *(I_pn I_msgs)*$^+$*]*
  *[***out** *(O_pn O_msgs)*$^+$*]*
  *[***post** *((lvar_name* **:=** *expr)*$^+$ *| ds_expr')])*$^+$ *]*

Producer-Consumer Example: Isabelle definition

```
datatype Pn = Inlet

ism Producer =
  ports Pn
    inputs   "{}"
    outputs "{Inlet}"
  messages int
  states
    data unit
  transitions
    produce:
      out Inlet "[n]"
```

```
record C_data = Accu :: int

ism Consumer =
  ports Pn
    inputs   "{Inlet}"
    outputs "{}"
  messages int
  states
    data C_data name "s"
  transitions
    consume:
      in Inlet "[n]"
      post Accu := "Accu s + n"
```

LKW Model of the Infineon SLE66

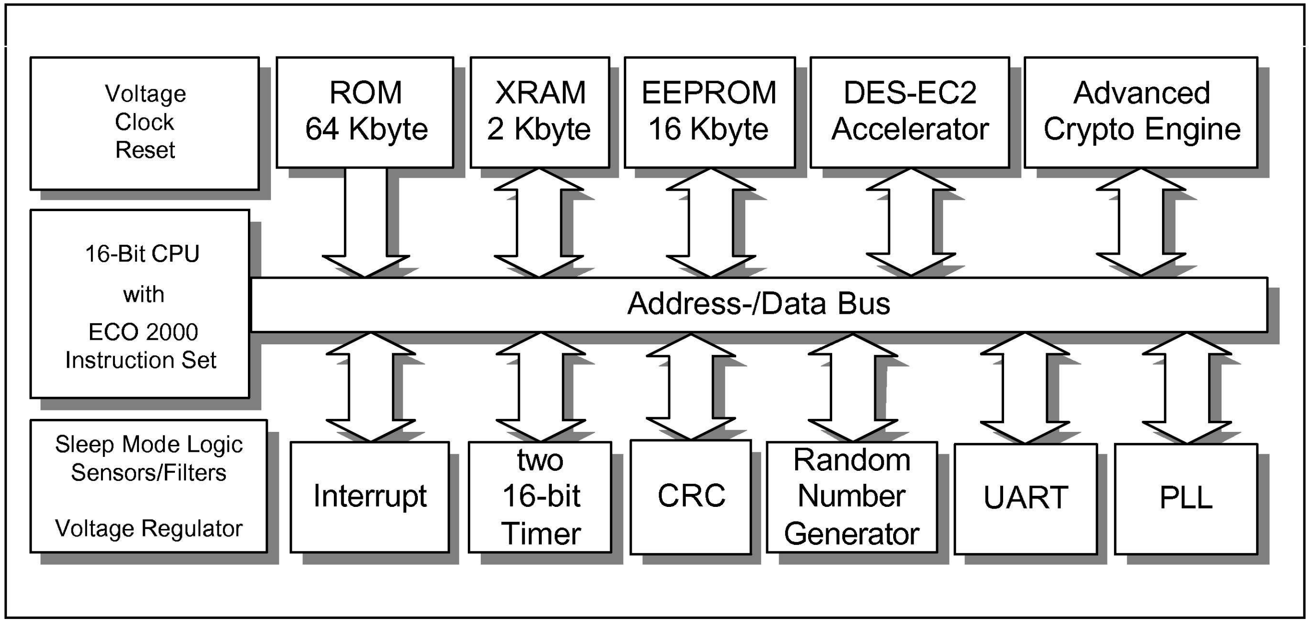
☞ **The SLE 66 family**

- LKW Model
- Security Properties

The SLE 66 family

SLE 66: family of smart card chips by Infineon Technologies

- General-purpose microprocessor with RAM, ROM, and EEPROM:



- Encryption unit, random number generator, sensors, . . .
- No MMU, no on-chip operation system functionality

⇝ Secure platform for customized BIOS and single application

SLE 66 Security Objectives

Applications: electronic passports, electronic payment systems, . . .

Security level: elementary, no assumptions about high-level functionality

Security objectives:

protect information stored in the different memory components:

- The data stored in any of the memory components
  shall be protected against unauthorized disclosure or modification.
- The security relevant functions implemented in firmware or hardware
  shall be protected against unauthorized disclosure or modification.
- Hardware test routines
  shall be protected against unauthorized execution.

SLE 66 Security Mechanisms

Objectives achieved by a set of security enforcing functions:

- System life-cycle divided in several phases.
  Entry to the phases controlled by test functions,
  checking various preconditions and authorization.
- Data stored in memory encrypted by hardware means.
  Several keys and key sources, including chip specific random number
- Sensors and active shields against physical tampering
- Provisions against differential power analysis (DPA)

LKW Model of the Infineon SLE66

- The SLE 66 family

☞ **LKW Model**

- Security Properties

Lotz-Kessler-Walter (LKW) Model

One of first formal models for security properties of hardware

Extrinsic value: Security certification on level ITSEC E4 / CC EAL5

Intrinsic value: Feedback for development and quality control

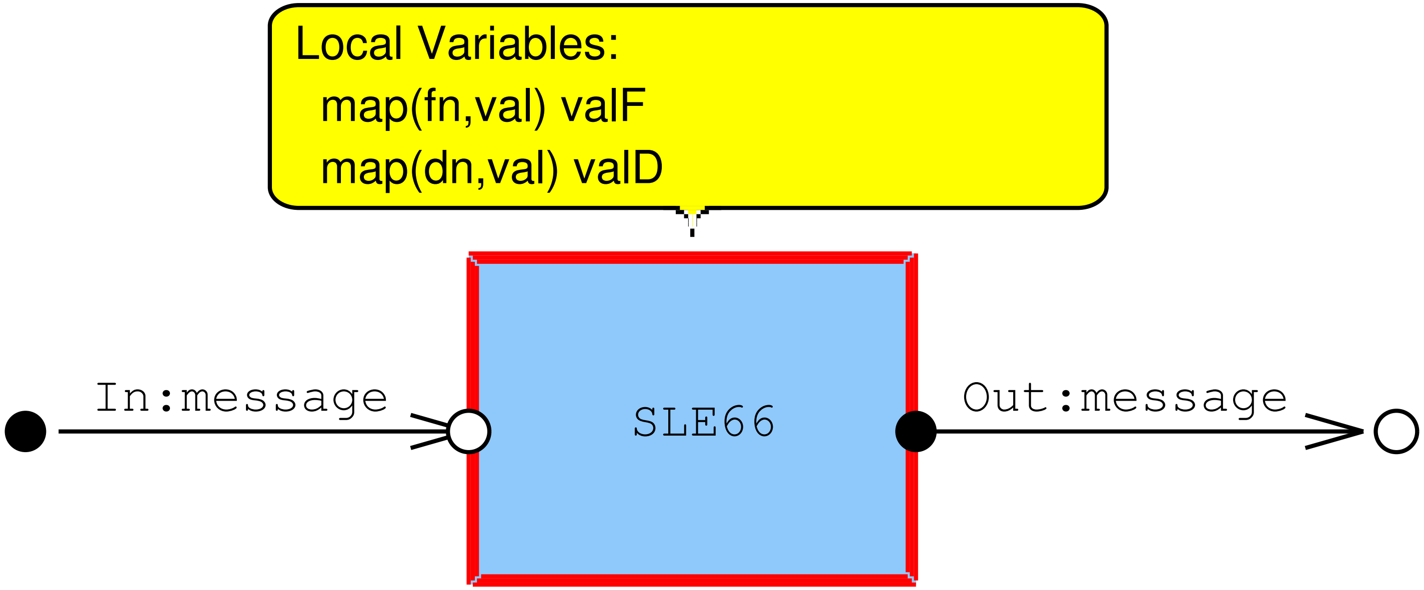
Abstract system model based on an ad-hoc automaton formalism
Formalization of security requirements, verification
Total effort: two months

Minor syntactical, typographical and semantical slips
Type errors, missing assumptions, incomplete proofs

$\Rightarrow$ ported to Isabelle/HOL + ISMs
Effort: two weeks

Added later: analysis of nonleakage

LKW Model: System Architecture



Local Variables:
map(fn,val) valF
map(dn,val) valD

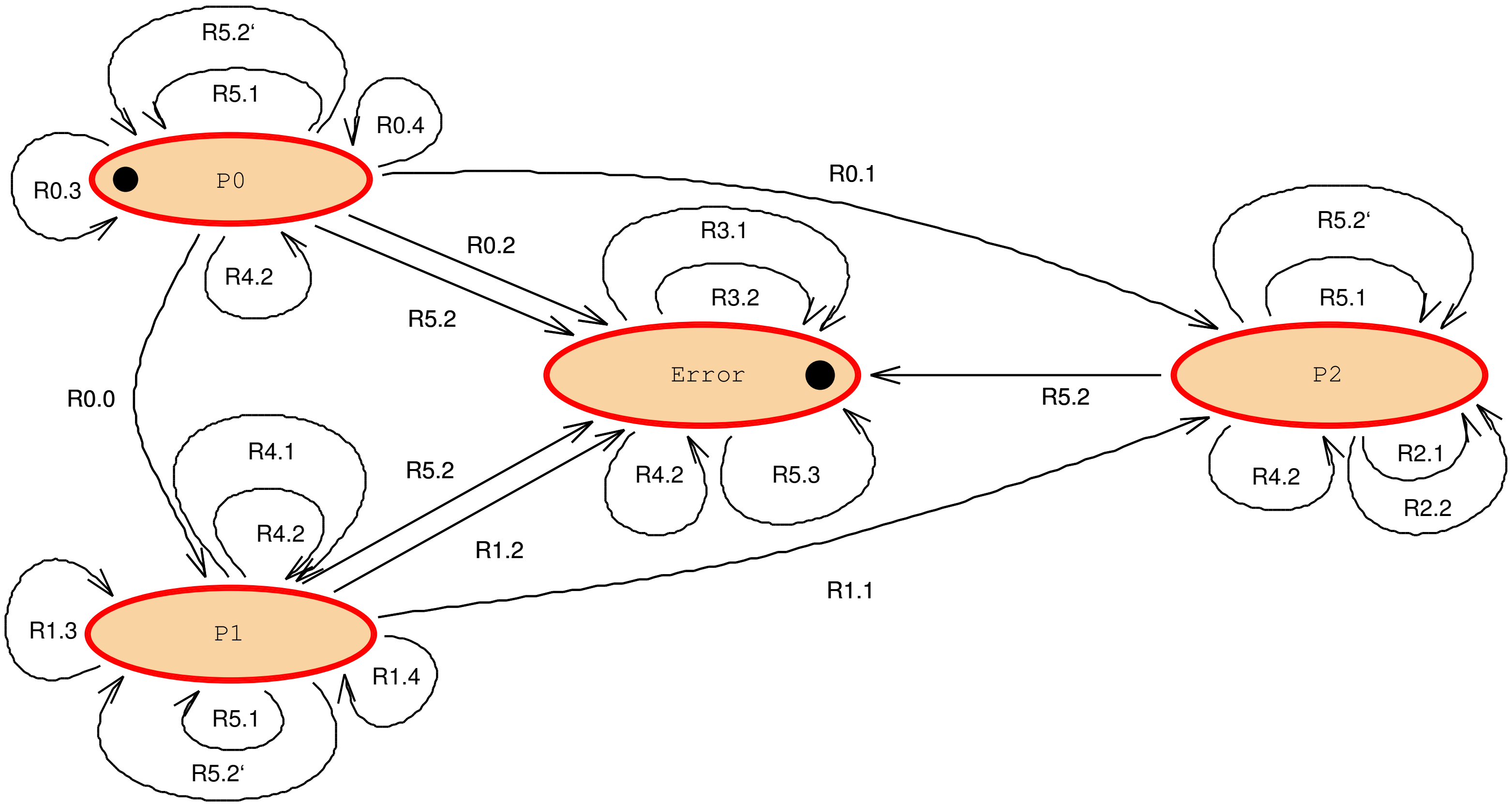
`In`: input port receiving commands

`Out`: output port emitting results/reaction

`valF` maps function names to function code, e.g. firmware

`valD` maps data object names to data values, e.g. personalization data

LKW Model: State Transitions (abstracted)



Phase *0*: chip construction

Phase *1*: upload of Smartcard Embedded Software and personalization

Phase *2*: deployment (normal usage)

Phase *Error*: locked mode from which there is no escape

LKW Model: Isabelle Theory

theory *SLE66 = ISM_package:*

- Build upon the general ISM theory
- Define various building blocks
- ISM section

- Underspecification often used for abstraction
- $\rightsquigarrow$ not all properties derivable from construction, but axioms needed

LKW Model: Names

```
typedecl fn                        — function name
typedecl dn                        — data object name
datatype on = F fn | D dn          — object name

consts
  f_SN   :: "fn"          — the name of the function giving the serial number

consts
  FTest0 :: "fn set"      — the names of test functions of phase 0
  FTest1 :: "fn set"      — the names of test functions of phase 1
  FTest  :: "fn set"      — the names of all test functions
defs
  FTest_def:  "FTest  ≡ FTest0 ∪ FTest1"

axioms
  FTest01_disjunct: "FTest0 ∩ FTest1 = {}"
  f_SN_not_FTest:   "f_SN ∉ FTest"
```

```
consts
  F_Sec  :: "fn set"  — the names of all security-relevant functions
  F_PSec :: "fn set"  — the subset of F_Sec relevant for the processor
  F_ASec :: "fn set"  — the names of _Sec relevant for applications
  F_NSec :: "fn set"  — the names of all non-security-relevant functions
defs
  F_ASec_def: "F_ASec ≡ F_Sec - F_PSec"
  F_NSec_def: "F_NSec ≡ -F_Sec"
axioms
  F_PSec_is_Sec:    "F_PSec ⊆ F_Sec"
  FTest_is_PSec:    "FTest ⊆ F_PSec"

consts
  D_Sec  :: "dn set"  — the names of all security-relevant data objects
  D_PSec :: "dn set"  — the subset of D_Sec relevant for the processor
  D_ASec :: "dn set"  — the names of D_Sec relevant for applications
  D_NSec :: "dn set"  — the names of all non-security-relevant data objects
defs
  D_ASec_def: "D_ASec ≡ D_Sec - D_PSec"
  D_NSec_def: "D_NSec ≡ -D_Sec"

consts Sec  ::  "on set"  — the names of all security-relevant objects
defs   Sec_def: "Sec ≡ {F fn |fn. fn ∈ F_Sec} ∪ {D dn |dn. dn ∈ D_Sec}"
```

LKW Model: State (1)

Control state of SLE 66 ISM: phase

```
datatype ph = P0 | P1 | P2 | Error
typedecl val — data and function values
consts SN :: val — serial number
```

Date state of SLE 66 ISM: two partial functions

```
record chip_data =
  valF :: "fn ⇀ val"
  valD :: "dn ⇀ val"
```

The overall state:

```
types SLE66_state = "ph × chip_data"
```

For simplification, date encryption left implicit

LKW Model: State (2)

Lookup:

constdefs

```
  val :: "chip_data ⇒ on ⇀ val"
 "val s on ≡ case on of F fn ⇒ valF s fn  | D dn ⇒ valD s dn"
```

Available functions:

constdefs

```
  fct :: "chip_data ⇒ fn set"
 "fct s ≡ dom (valF s)"
```

Functions results and their effect on the state:

consts

```
 "output"   :: "fn ⇒ chip_data ⇒ val"
 "change"   :: "fn ⇒ chip_data ⇒ chip_data"
                                  — change is unused for test functions
 "positive" :: "val ⇒ bool"       — check for positive test outcome
```

LKW Model: ISM definition (1)

Two port names:

```
datatype interface = In | Out
```

Subjects issuing commands:

```
typedecl sb
consts Pmf :: sb     — processor manufacturer
```

Commands as input, values as potential output:

```
datatype message =
  Exec sb fn | Load sb fn val | Spy on   — input
| Val val | Ok | No                      — output

consts subject :: "message ⇒ sb"
primrec
  "subject (Exec sb fn  ) = sb"
  "subject (Load sb fn v) = sb"
```

LKW Model: ISM definition (2)

```
ism SLE66 =

  ports interface
    inputs   "{In}"
    outputs "{Out}"

  messages message

  states
    control ph init "P0"
    data    chip_data name "s" — The data state variable is called s.
                                 — The initial data state is left unspecified.

  transitions
      . . .
```

LKW Model: Transitions, R0.0

R0.0 thru R0.4: function execution in initial phase 0.

- Only the processor manufacturer is allowed to invoke functions.
- The selected function must be present.

R0.0: if function belongs to *FTest0* and the corresponding test succeeds, phase 1 is entered, and functions *FTest0* are disabled.

```
R00: P0 → P1
  pre "f ∈ fct s∩FTest0", "positive (output f s)"
  in  In  "[Exec Pmf f]"
  out Out "[Ok]"
  post valF := "valF s⌊(-FTest0)"
```

LKW Model: R0.1, R0.2

R0.1: *shortcut leaving out phase 1.*

If the function belongs to `FTest1` and the test succeeds, phase 2 is entered, and all test functions are disabled.

```
R01: P0 ⟶ P2
  pre "f ∈ fct s∩FTest1", "positive (output f s)"
  in  In  "[Exec Pmf f]"
  out Out "[Ok]"
  post valF := "valF s⌊(-FTest)"
```

R0.2: if test fails, the system enters the error state.

```
R02: P0 ⟶ Error
  pre "f ∈ fct s∩FTest0", "¬positive (output f s)"
  in  In  "[Exec Pmf f]"
  out Out "[No]"
```

LKW Model: R0.3, R0.4

R0.3: successful execution of all other function:
the function yields a value and may change the chip state

```
R03: P0 ⟶ P0
  pre "f ∈ fct s - FTest"
  in  In  "[Exec Pmf f]"
  out Out "[Val (output f s)]"
  post "change f s"
```

R0.4: in all remaining cases of function execution,
the chip responds with *No* and its state remains unchanged.

```
R04: P0 ⟶ P0
  pre "sb ≠ Pmf ∨ f ∉ fct s"
  in  In  "[Exec sb f]"
  out Out "[No]"
```

LKW Model: R1.1-R1.4: functions in upload phase 1

```
R11: P1 ⟶ P2
   pre "f ∈ fct s∩FTest1", "positive (output f s)"
   in  In  "[Exec Pmf f]"
   out Out "[Ok]"
   post valF := "valF s⌊(-FTest1)"
R12: P1 ⟶ Error
   pre "f ∈ fct s∩FTest1", "¬positive (output f s)"
   in  In  "[Exec Pmf f]"
   out Out "[No]"
R13: P1 ⟶ P1
   pre "f ∈ fct s - FTest1"
   in  In  "[Exec Pmf f]"
   out Out "[Val (output f s)]"
   post "change f s"
R14: P1 ⟶ P1
   pre "sb ≠ Pmf ∨ f ∉ fct s"
   in  In  "[Exec sb f]"
   out Out "[No]"
```

LKW Model: R2.1 and R2.2

R2.1 and R2.2: function execution in usage phase 2, analogously to R0.3 and R0.4.

```
R21: P2 → P2
  pre "f ∈ fct s"
  in  In  "[Exec sb f]"
  out Out "[Val (output f s)]"
  post "change f s"

R22: P2 → P2
  pre "f ∉ fct s"
  in  In  "[Exec sb f]"
  out Out "[No]"
```

LKW Model: R3.1 and R3.2

R3.1 and R3.2: function execution in the error phase:
the only function allowed to be executed is chip identification.

```
R31: Error ⟶ Error
  pre "f_SN ∈ fct s"
  in   In  "[Exec sb f_SN]"
  out Out "[Val SN]"

R32: Error ⟶ Error
  pre "f ∉ fct s∩{f_SN}"
  in   In  "[Exec sb f]"
  out Out "[No]"
```

LKW Model: R4.1 and R4.2

Effects of uploading new functionality.

- Must be done by the processor manufacturer
- Allowed only in phase 1
- Meanwhile, also security-critical application functions are loadable.

R4.1: the admissible situations

```
R41: P1 ⟶ P1
  pre "f ∈ F_NSec ∪ (F_ASec - fct s)"
  in  In  "[Load Pmf f v]"
  out Out "[Ok]"
  post valF := "valF s(f↦v)"
```

R4.2: all other cases

```
R42: ph ⟶ ph
  pre "f ∉ F_NSec ∪ (F_ASec - fct s) ∨ sb ≠ Pmf ∨ ph ≠ P1"
  in  In  "[Load sb f v]"
  out Out "[No]"
```

LKW Model: R5.1

R5.1 thru R5.3: the effects of attacks

Special "spy" input models any attempts to tamper with the chip and to read security-relevant objects via physical probing on side channels (by mechanical, electrical, optical, and/or chemical means), e.g. differential power analysis or inspection with microscope

Modeling physical attacks in more detail is not feasible: would require a model of physical hardware.

R5.1: the innocent case of reading non-security-relevant objects in any regular phase, which actually reveals the requested information.

```
R51: ph ⟶ ph
  pre "on ∉ Sec", "ph ≠ Error"
  in  In  "[Spy on]"
  out Out "case val s on of None ⇒ [] | Some v ⇒ [Val v]"
```

LKW Model: R5.2

R5.2: attempt to read security-relevant objects in a regular phase. The requested object may be revealed or not. If a secret is leaked, the chip has to detect this and enter the error phase.

"Destructive reading": attacks may reveal information even about security-relevant objects, but after the first of any such attacks, the processor hardware will be "destroyed", i.e. cannot be used regularly.

```
R52: ph  →  Error
   pre "on ∈ Sec", "v ∈ {[],[Val (the (val s on))]}", "ph ≠ Error"
   in   In   "[Spy on]"
   out  Out  "v"
   post "any"

R52':ph  →  ph
   pre "on ∈ Sec", "ph ≠ Error"
   in   In   "[Spy on]"
   out  Out  "[]"
```

LKW Model: R5.3

R5.3: in the error phase no (further) information is revealed.

$R53:\ Error \rightarrow Error$
in *In* "*[Spy on]*"
out *Out* "*[]*"
post "*any*"

R5.2 and R5.3 ⇒ the attacker may obtain (the representation of) at most one security-relevant object from the chip memory.

Such singleton leakage is harmless!

All data stored on the chip is encrypted. The value obtained may be

the encryption key itself: no further data item, in particular none encrypted with the key, can be obtained.

encrypted value: attacker cannot any more extract the respective key.

Both cases not helpful to the attacker.

LKW Model: Rule features

R52: *ph* $\longrightarrow$ *Error*

pre "*ph* $\neq$ *Error*", "*oname* $\in$ *Sec*",

"*v* $\in$ {[], [*Val* (*the* (*val* σ *oname*))]}"

in *In* "[*Spy oname*]"

out *Out* "*v*"

post "any"

Typical:

Both input and output

Underspecification

Nondeterminism (2 ×)

Generic transitions

LKW Model: ISM Runs

types

```
  SLE66_trans = "(unit, interface, message, SLE66_state) trans"
```

constdefs

```
  Trans :: "SLE66_trans set"                   — all possible transitions
 "Trans ≡ trans SLE66.ism"

  TRuns :: "(SLE66_trans list) set"            — all possible transition sequences
 "TRuns ≡ truns SLE66.ism"

  Runs :: "(SLE66_state list) set"             — all possible state sequences
 "Runs ≡ runs SLE66.ism"
```

LKW Model of the Infineon SLE66

- The SLE 66 family

- LKW Model

☞ **Security Properties**

LKW Model: Security Objectives

In (confidential) original security requirements specification by Infineon:

SO1. "The hardware must be protected against espionage of the security functionality."

SO2. "The hardware must be protected against unauthorised modification of the security functionality."

SO3. "The information stored in all memory devices must be protected against unauthorised access."

SO4. "The information stored in all memory devices must be protected against unauthorised modification."

SO5. "It must not be possible to execute the test routines of the STS test mode without authorisation."

Later, additional requirements were added:

SO[1+2]'. confidentiality+integrity of Smartcard Embedded Software.

LKW Model: Formalized Security Objective FSO1

FSO1: *in any sequence* `ts` *of transitions performed by the chip, if the chip outputs a value* `v` *representing the code of any security-relevant function during its hitherto life* `ts`*, then the next state is in the error phase, or the output was due to a function call by the processor manufacturer.*

```
theorem FSO1: "⟦ts ∈ TRuns; ((p,(ph,s)),c,(p',(ph',s'))) ∈ set ts;
                p' Out = [Val v]; v ∈ ValF_Sec (truns2runs ts)⟧ ⟹
                ph' = Error ∨ (∃fn. p In = [Exec Pmf fn])"
```

The set `ValF_Sec r` holds *the code of all security-relevant functions present anywhere in a run* `r`:

```
constdefs
  ValF_Sec :: "SLE66_state list  ⇒ val set"
 "ValF_Sec r ≡ ⋃{ran (valF s⌊F_Sec) | ph s. (ph,s) ∈ set r}"
```

LKW Model: Proof of FSO1 (1)

Proof of `FSO1` by

- unfolding some definitions, e.g. of the SLE 66 ISM
- applying properties of auxiliary concepts like *truns2runs*
- a case split on all possible transitions

Isabelle solves most of the cases automatically (with straightforward term rewriting and purely predicate-logical reasoning), except two:

R2.1 (normal function execution) is handled using `Axiom3`:
In phase 2, a function cannot reveal (by "guessing" or by accident) any members of ValF_Sec r

Axiom3: "$[\![r \in Runs;\ (P2,s) \in set\ r;\ f \in fct\ s]\!] \Longrightarrow output\ f\ s \notin ValF\_Sec\ r$"

LKW Model: Proof of FSO1 (2)

R5.1 (harmless *Spy* attack) relies on the lemma

```
"⟦r ∈ Runs; (ph, s) ∈ set r; n ∉ Sec; val s n = Some v⟧ ⟹ v ∉ ValF_Sec r"
```

which in turn relies on `Axiom4`:

If a function can be referenced in two (different) ways and one of them declares it to be security-relevant, the other does the same.

```
Axiom4: "⟦r ∈ Runs;
         (ph, s) ∈ set r; (ph', s') ∈ set r;
         val s n = Some v; val s' n' = Some v;
         n ∈ Sec⟧ ⟹ n' ∈ Sec"
```

When machine-checking the orginal pen-and-paper proofs, we noticed that Axiom4 was missing!

Such experience demonstrates importance of machine support when conducting formal analysis.

LKW Model: FSO21

Translation of SO2 splits into two parts: overwriting and deletion.

FSO21': *for any transition not ending in the error phase,*
if a security-relevant function `g` *is present in both the pre-state and the post-state, the code associated with it stays the same*:

theorem `FSO21'`: $"[\![((p,(ph,s)),c,(p',(ph',s')))\in Trans;\ ph' \neq Error;$
$g \in fct\ s \cap fct\ s' \cap F\_Sec]\!] \Longrightarrow valF\ s'\ g = valF\ s\ g"$

This is a generalization of the original `FSO21`, to reflect the extensions made to the *Load* operation in rule `R41`:

We do not compare the *initial* and current values of g
but the *previous* and current values of g

$\leadsto$ takes into account also functions added in the meantime

LKW Model: Proof of FSO21

Proof of `FSO21` by case distinction over all possible transitions.

Most cases are trivial except where function execution may change the stored objects (as described by `R03`, `R13`, and `R21`).
There, invariance of security-relevant functions g is needed, which follows easily from `Axiom1` and `Axiom2`:

Security-relevant functions do not modify security-relevant functions:
Axiom1: "f∈fct s∩F_Sec ⟹ valF (change f s)⌊F_Sec = valF s⌊F_Sec"

In comparison to the version of this axiom in the original model, the scope of functions `f` has been extended from *"initially available"* to *"security-relevant"*, reflecting the changes to rule `R41`.

Also non-security-relevant functions do not modify s.-r. functions:
Axiom2: "f∈fct s∩F_NSec ⟹ valF (change f s)⌊F_Sec = valF s⌊F_Sec"

LKW Model: FSO22

FSO22: similarly to `FSO21'`,

for any transition within the same phase that is not the error phase, the set of existing security-relevant functions is non-decreasing:

theorem `FSO22: "⟦((p,(ph,s)),c,(p',(ph',s')))∈Trans; ph' ≠ Error; ph = ph'⟧ ⟹ fct s∩F_Sec ⊆ fct s'∩F_Sec"`

Proof: analougous of `FSO21'`.

LKW Model: FSO3

FSO3: *when trying to get hold of a security-relevant data object* `on`*,*
if the attacker obtains a security-relevant value,
then the chip enters the error phase:

theorem *FSO3* :"⟦((p,(ph,s)),c,(p',(ph',s')))∈Trans; p In = [Spy on];
on ∈ Sec; p' Out ≠ []⟧ ⟹ ph' = Error"

Proof: by case distinction.

FSO13: *once the chip is in the error phase, it stays there*
and the only possible output is the serial number:

theorem *FSO13*: "⟦((p,(ph,s)),c,(p',(ph',s')))∈Trans; ph = Error;
p' Out = [Val v]⟧ ⟹ v = SN ∧ ph' = Error"

Proof: by case distinction.

LKW Model: FSO4

FSO4: *for any transition not ending in the error phase,*
if it changes the state,
this is done in a well-behaved way: `s'` *is derived from* `s` *. . .*

- *via the desired effect of executing an existing function, or*
- *there is a phase change where only test functions are affected, or*
- *only a single function* `f` *is affected by a* `Load` *operation:*

```
theorem FSO4:
"⟦((p,(ph,s)),c,(p',(ph',s'))) ∈ Trans; ph' ≠ Error⟧ ⟹
 s' = s ∨
 (∃sb f  . p In = [Exec sb f] ∧ f ∈ fct s ∧ s' = change f s) ∨
 (ph'≠ph ∧ valD s' = valD s ∧ valF s'⌊(-FTest) = valF
s⌊(-FTest))∨
 (∃sb f v. p In = [Load sb f v] ∧
              valD s' = valD s ∧ valF s'⌊(-{f}) = valF s⌊(-{f}))"
```

Proof: by case distinction.

LKW Model: FSO5

FSO5: *in any sequence of transitions performed by the chip, any attempt to execute a test function not issued by the processor manufacturer is refused:*

```
theorem FSO5: "⟦ts ∈ TRuns; ((p,(ph,s)),c,(p',(ph',s'))) ∈ set ts;
               p In = [Exec sb f]; f ∈ FTest⟧ ⟹
               sb = Pmf ∨ s' = s ∧ p' Out = [No]"
```

A second omission of the LKW model was:
In the proof of the security objective `FSO5`, an argumentation about the accessibility of certain functions was not given.

We fix this by introducing an auxiliary property and proving it to be an invariant of the ISM.

As usual, finding the appropriate invariant was the main challenge.

LKW Model: Proof of FSO5 with invariant

The invariant states that

- *in phase 1, the test functions from* `FTest0` *have been disabled*
- *in phase 2, all test functions have been disabled*

constdefs

```
  no_FTest_invariant :: "SLE66_state ⇒ bool"
 "no_FTest_invariant ≡ λ(ph,s). ∀f ∈ fct s.
   (ph = P1 ⟶ f ∉ FTest0) ∧ (ph = P2 ⟶ f ∉ FTest)"
```

When proving the invariant, 14 of the 19 cases are trivial.
The remaining ones require simple properties of the set `FTest`,
and two of them require additionally `Axiom1` and `Axiom2`.

The invariant implies

lemma `P2_no_FTest:`

```
  "⟦(P2,s) ∈ reach SLE66.ism; f ∈ fct s⟧ ⟹ f ∉ FTest"
```

Exploiting the lemma for the case of rule `R21`, we can prove `FSO5`.

LKW Model: Conclusion

Abstract specification: ISM + a few axioms, e.g.

Axiom1: "$f{\in}fct\ s{\cap}F\_Sec \Longrightarrow valF\ (change\ f\ s)\lfloor F\_Sec = valF\ s\lfloor F\_Sec$"

Security objectives: predicates on the system behavior, e.g.

theorem FS05: "$[\![ts \in TRuns;\ ((p,(ph,s)),c,(p',(ph',s'))) \in set\ ts;$
$p\ In = [Exec\ sb\ f];\ f \in FTest]\!] \Longrightarrow$
$sb = Pmf \lor s' = s \land p'\ Out = [No]$"

Experience:

- Detected omissions: one axiom, one invariant
- Isabelle proofs: just a few steps, 50% automatic
- New requirements cause only slight changes

Contents

- Introduction
- Access Control
- **Information Flow**
- Cryptoprotocol Analysis
- Evaluation & Certification

Outline

- Denning model

- Noninterference
  - Classical notion, unwinding
  - Access control interpretation
  - Nondeterminism

- Nonleakage and Noninfluence
  - Motivation, notion, variants
  - Noninfluence
  - SLE 66 case study

Explicit and Implicit Information Flow

- Access control models do not consider *covert channels*: information transfer via e.g. timing behavior, or existence of files
  - An action causes an *information flow* from an object x to an object y, if we may learn more about x by observing y.
  - If we already knew x, then no information can flow from x.
- We distinguish:
  - Explicit information flow: observing y after the assignment `y:=x` tells one the value of x.
  - Implicit information flow: for conditional `if x=0 then y:=1`, observing y after the statement may tell one something about x even if the assignment `y:=1` has not been executed.
- Information flow models cover implicit information flow

The Denning Model (1)

- A formal definition can be given in terms of *information theory*.

  For instance, information flow from x to y is defined by the decrease in the *equivocation (conditional entropy)* of x given the value of y.

- The *Denning model* considers systems with transitions of the form `if` $P(z_1, \ldots, z_n)$ `then` y `:=` $f(x_1, \ldots, x_m)$. Its components are
  - A lattice $(L, \leq)$ of security labels.
  - A set of labeled objects.
  - The security policy: a flow is illegal when it violates

> Rule: information flow from an object x with label $l(x)$ to an object y with label $l(y)$ is permitted only if $l(x) \leq l(y)$.

- A system is called secure if there is no illegal information flow.

The Denning Model (2)

- We can distinguish:
  - Static enforcement of information flow policies:
    a program is checked at compile-time using a type system
    → *Language-based security* by Sabelfeld, Myers et al.
  - Dynamic enforcement using run-time flow control mechanism:
    transitions can be secured by adding an extra precondition:

$$\texttt{if}\ P(z_1, \ldots, z_n)\ \boxed{\wedge\ sup(\{l(x_1), \ldots, l(x_m), l(z_1), \ldots, l(z_n)\}) \leq l(y)}$$
$$\texttt{then}\ y\ \texttt{:=}\ f(x_1, \ldots, x_m)$$

- The Denning information flow model covers indirect information flow, but

> Theorem: checking whether a given system is secure in the Denning information flow model is an undecidable problem.

Outline

- Denning model

☞ **Noninterference**

  - Classical notion, unwinding
  - Access control interpretation
  - Nondeterminism

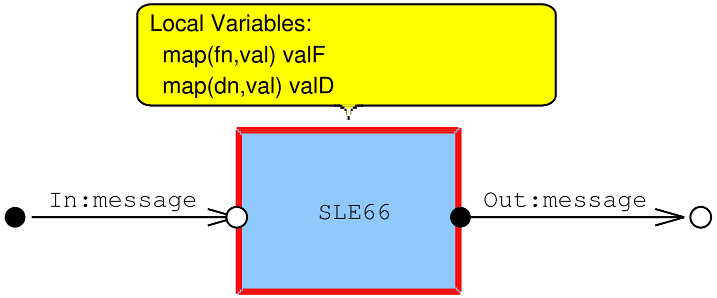
- Nonleakage and Noninfluence
  - Motivation, notion, variants
  - Noninfluence
  - SLE 66 case study

Noninterference

- information is classified using *domains* (security 'levels')
- users, variables, files, actions, processes, etc. are assigned to domains
- *policy*: relation (e.g. partial order) on domains, called *interference* $\leadsto$
- its complement is called *noninterference* relation $\not\leadsto$
- if $d \not\leadsto d'$, then 'actions' of d must not influence d', where 'action' often means: variation of contents
- confidentiality: observations about d impossible for d'
- integrity: changes to d' impossible for d

Motivation

Task: Security analysis for Infineon SLE66 smart card processor



Main concern: confidentiality of on-chip secrets

Initial solution: *representation* of secret values is not output

Problem: leakage of re-encoded and partial information

***Maximal* solution:** observable output *independent* of secrets

Approach: some sort of noninterference

Generic Notions

System model: — Moore automaton

$step : action \times state \rightarrow state$

$run : action^* \times state \rightarrow state$

— also nondeterministic variants

Security model:

$domain$ — secrecy level/area

$obs : domain \times state \rightarrow output$

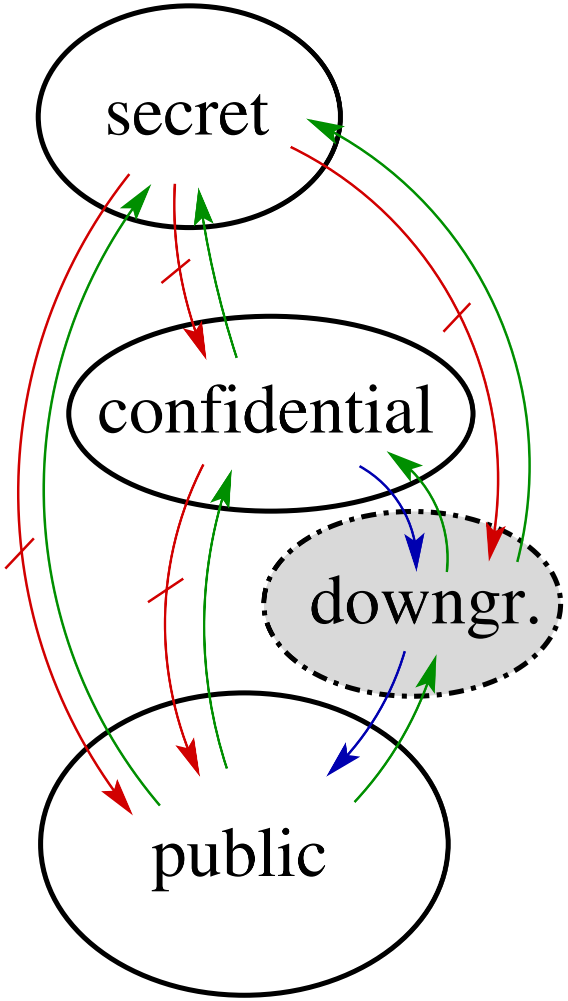
$dom : action \rightarrow domain$ — input domain

Policy or interference relation

$\rightsquigarrow : \wp(domain \times domain)$

— always reflexive, possibly intransitive

Noninterference relation: $\not\rightsquigarrow$



Noninterference [GM82/84,Rus92]

Aim: secrecy of the presence/absence of actions

$$\boxed{\begin{array}{l} \mathit{noninterference} \equiv \\ \forall \alpha\ u.\ \mathit{obs}(u, \mathit{run}(\alpha, s_0)) = \mathit{obs}(u, \mathit{run}(\mathit{ipurge}(u, \alpha), s_0)) \end{array}}$$

$\mathit{ipurge}(u, \alpha)$ =”remove from the sequence α all actions that may not influence u, directly or via the domains of subsequent actions within α”

Observational equivalence/relation

$\cdot \lhd \cdot \overset{\cdot}{\bumpeq} \cdot \lhd \cdot :\ \mathit{domain} \ \to \wp(\mathit{state}\ \times\ \mathit{action}^*\ \times\ \mathit{state}\ \times\ \mathit{action}^*)$

$s \lhd \alpha \overset{u}{\bumpeq} t \lhd \beta \equiv \mathit{obs}(u, \mathit{run}(\alpha, s)) = \mathit{obs}(u, \mathit{run}(\beta, t))$

$$\boxed{\mathit{noninterference} \equiv \forall \alpha\ u.\ s_0 \lhd \alpha \overset{u}{\bumpeq} s_0 \lhd \mathit{ipurge}(u, \alpha)}$$

ipurge & sources

$$ipurge : \ domain \ \times \ action^* \ \rightarrow \ action^*$$

$$ipurge(u, []) = []$$

$$ipurge(u, a \frown \alpha) = if \ dom(a) \in sources(a \frown \alpha, u)$$

$$then \ a \frown ipurge(u, \alpha) \ else \ ipurge(u, \alpha)$$

$sources(\alpha, u)$ = “all domains of actions in α that may influence u, directly or via the domains of subsequent actions within α”

e.g., $v \in sources(a_1 \frown a_2 \frown a_3 \frown a_4, u)$

if $v = dom(a_2) \rightsquigarrow dom(a_4) \rightsquigarrow u$ (even if $v \not\rightsquigarrow u$)

$$sources : \ action^* \ \times \ domain \ \rightarrow \ \wp(domain)$$

$$sources([], u) = \{u\}$$

$$sources(a \frown \alpha, u) = sources(\alpha, u) \cup$$

$$\{w. \ \exists v. \ dom(a) = w \wedge w \rightsquigarrow v \ \wedge \ v \in sources(\alpha, u)\}$$

Unwinding

Problem: noninterference is global property, to be shown for any α

Idea: induction on α shows preservation of

unwinding relation $\sim$ between states,
parameterized by domain: $domain \rightarrow \wp(state \times state)$

— some kind of equality on the sub-state belonging to the domain

— no need to be reflexive, symmetric, nor transitive [Man00/03]

— lifting to sets of domains: $s \overset{U}{\approx} t \equiv \forall u \in U.\ s \overset{u}{\sim} t$

Local properties: essentially $s \overset{u}{\sim} t \longrightarrow step(a, s) \overset{u}{\sim} step(a, t)$

(step consistency, step respect, local respect)

Proof Sketch

Theorem Goal: $obs(u, run(\alpha, s_0)) = obs(u, run(ipurge(u, \alpha), s_0))$

Main Lemma:

$$\forall s\ t.\ s \overset{sources(\alpha,u)}{\approx} t \longrightarrow run(\alpha, s) \overset{u}{\sim} run(ipurge(u, \alpha), t)$$

Proof of Theorem: specialize by $s = t = s_0$, use $s_0 \overset{sources(\alpha,u)}{\approx} s_0$,
and apply output consistency $\forall u\ s\ t.\ s \overset{u}{\sim} t \longrightarrow obs(u, s) = obs(u, t)$

Proof of Main Lemma: by induction $\alpha' \rightarrow a \frown \alpha'$

$s \overset{sources(a \frown \alpha',u)}{\approx} t$ implies

$if\ dom(a) \in sources(a \frown \alpha', u)$

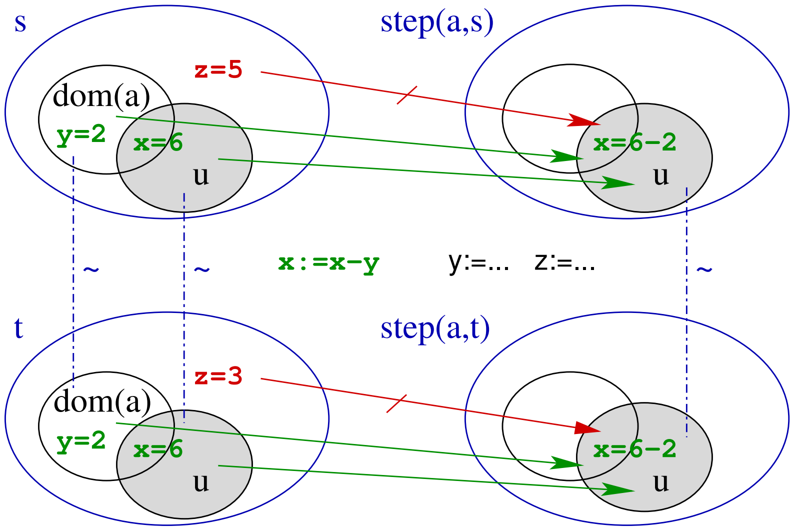
(step consistency + respect): $then\ step(a, s) \overset{sources(\alpha',u)}{\approx} step(a, t)$

(local respect): $else\ step(a, s) \overset{sources(\alpha',u)}{\approx} t$, then

ind. hypothesis implies $run(\alpha', step(a, s)) \overset{u}{\sim} run(ipurge(u, a \frown \alpha'), t)$

Step Consistency and Step Respect

$$\mathit{weakly\_step\_consistent} \equiv$$
$$\forall a\ u\ s\ t.\ \mathit{dom}(a) \rightsquigarrow u\ \wedge\ s \overset{\mathit{dom}(a)}{\sim} t\ \wedge\ s \overset{u}{\sim} t \longrightarrow \mathit{step}(a,s) \overset{u}{\sim} \mathit{step}(a,t)$$



$$\mathit{step\_respect} \equiv \forall a\ u\ s\ t.\ \mathit{dom}(a) \not\rightsquigarrow u\ \wedge\ s \overset{u}{\sim} t \longrightarrow \mathit{step}(a,s) \overset{u}{\sim} \mathit{step}(a,t)$$
$$\mathit{local\_respect\_left} \equiv \forall a\ u\ s\ t.\ \mathit{dom}(a) \not\rightsquigarrow u \wedge s \overset{u}{\sim} t \longrightarrow \mathit{step}(a,s) \overset{u}{\sim} t$$
$$\mathit{local\_respect\_right} \equiv \forall a\ u\ s\ t.\ \mathit{dom}(a) \not\rightsquigarrow u\ \wedge\ s \overset{u}{\sim} t \longrightarrow s \overset{u}{\sim} \mathit{step}(a,t)$$

Outline

- Denning model

- Noninterference
  - ▶ Classical notion, unwinding
  - ☞ **Access control interpretation**
  - ▶ Nondeterminism

- Nonleakage and Noninfluence
  - ▶ Motivation, notion, variants
  - ▶ Noninfluence
  - ▶ SLE 66 case study

Access Control Interpretation

More concrete system model with explicit read/write to variables

- State contents maps names to values

$$contents : state \times name \longrightarrow value$$

- Names of objects a domain is allowed to read or write:

$$observe : domain \longrightarrow \wp(name)$$
$$alter : domain \longrightarrow \wp(name)$$

- The canonical unwinding relation induced by *contents* and *observe*:

$$s \overset{u}{\sim} t \equiv \forall n \in observe(u).\ contents(s, n) = contents(t, n)$$

This happens to be an equivalence.

Reference Monitor Assumptions (1)

More concrete conditions implying step consistency and local respect

- $RMA_1 \equiv output\_consistent$, fulfilled immediately if the output function yields all values observable for the given domain: $output(u,s) \equiv \{(n, contents(s,n)) \mid n \in observe\ u\}$
- If action a changes the contents of variable n observable by domain u and if $dom(a)$ may influence u, the new value depends only on values observable by $dom(a)$ and u:

$$\begin{aligned} RMA_2 \equiv \forall a\ u\ s\ t\ n.\ s \overset{dom(a)}{\sim} t\ \land\ dom(a) \rightsquigarrow u\ \land\ s \overset{u}{\sim} t\ \land \\ n \in observe\ u\ \land\ (contents(step(a,s),n)\ \neq\ contents(s,n) \lor \\ contents(step(a,t),n)\ \neq\ contents(t,n)) \longrightarrow \\ contents(step(a,s),n) = contents(step(a,t),n) \end{aligned}$$

Note that RMA_2 is equivalent to $weakly\_step\_consistent$

Reference Monitor Assumptions (2)

- Any changes must be granted by *alter*:

  $RMA_3 \equiv \forall a\ s\ n.$
  $contents(step(a,s),n) \neq contents(s,n) \longrightarrow n \in alter(dom(a))$

  In conjunction with the condition

  $AC\_policy\_consistent \equiv \forall u\ v.\ alter(u) \cap observe(v) \neq \emptyset \longrightarrow u \leadsto v$,

  this implies local respect:

  $RMA_3\ \wedge\ AC\_policy\_consistent\ \longrightarrow\ local\_respect$

- Hence, enforcement of access control implies security:

  theorem $access\_control\_secure$:
  $RMA_1\ \wedge\ RMA_2\ \wedge\ RMA_3\ \wedge\ AC\_policy\_consistent\ \longrightarrow$
  $noninterference$

Nondeterminism

$$\begin{array}{lll} Step: & action & \rightarrow \wp(state \times state) \\ Run: & action^* & \rightarrow \wp(state \times state) \end{array}$$

new: non-unique outcome,
partiality/reachability

$$Noninterference \equiv \forall \alpha\ u\ \beta.\ ipurge(u,\alpha) = ipurge(u,\beta) \longrightarrow \forall s.\ (s_0, s) \in Run(\alpha) \longrightarrow \exists t.\ (s_0, t) \in Run(\beta) \land obs(u,s) = obs(u,t)$$

Complications for weak step consistency ⇒
stronger notions preserving simultaneous unwinding relation $\approx$:
uniform step consistency, step respect, and (right-hand) local respect

Requires in general more proof effort, yet not for two important cases:

- functional $Step(a)$
- two-level domain hierarchy $\{H, L\}$

Outline

- Denning model

- Noninterference
  - ▶ Classical notion, unwinding
  - ▶ Access control interpretation
  - ▶ Nondeterminism

☞ **Nonleakage and Noninfluence**
  - ▶ Motivation, notion, variants
  - ▶ Noninfluence
  - ▶ SLE 66 case study

Nonleakage and Noninfluence

Event-based systems:

- visibility of actions/events is primary,
- secret state is secondary (via side-effects)

⇒ Noninterference

State-oriented systems:

- secret state is primary,
- actions/events are secondary or irrelevant

⇒ Nonleakage

State-event-systems:

- visibility of actions/events is relevant
- also secrecy in state is essential

⇒ Noninfluence

Concept

Language-based security: no assignments of high-values to low-variables, enforced by type system

Semantically: take (x, y) as elements of the state space with high-level data (on left) and low-level data (on right).

Step function $S(x, y) = (S_H(x, y), S_L(x, y))$ does not leak information from high to low if $S_L(x_1, y) = S_L(x_2, y)$ (functional independence).

Observational equivalence $(x, y)\overset{L}{\sim}(x', y') \;:\longleftrightarrow\; y = y'$ allows re-formulation:

$$\boxed{\begin{array}{l} s \overset{L}{\sim} t \;\longrightarrow S(s) \overset{L}{\sim} S(t) \;\; (\text{preservation of } \overset{L}{\sim}) \\ \text{step consistency} + \text{respect} \end{array}}$$

Generalization to action sequences α and arbitrary policies $\rightsquigarrow$

Definition

$$nonleakage \equiv \forall \alpha\ s\ u\ t.\ s \overset{sources(\alpha,u)}{\approx} t \longrightarrow s \lhd \alpha \overset{u}{\simeq} t \lhd \alpha$$

"the outcome of u's observation is independent of those domains from which no (direct or indirect) information flow is allowed."

- like Main Lemma, but no purging (visibility of actions irrelevant)
- unwinding relation $\sim$ is part of the notion:
  the secrets for u are those state components not constrained by $\sim$
- corresponding unwinding theorem: nonleakage implied by

  $$weakly\_step\_consistent\ \wedge\ step\_respect\ \wedge\ output\_consistent$$

Variants

If (domains of) actions are irrelevant:

$$\boxed{weak\_nonleakage \equiv \forall \alpha\ s\ u\ t.\ s \overset{chain(\alpha,u)}{\approx} t \longrightarrow s \lhd \alpha \overset{u}{\simeq} t \lhd \alpha}$$

where $chain : action^* \times domain \to \wp(domain)$

e.g., $v \in chain(a_1 \frown a_2 \frown a_3 \frown a_4, u)$ if $\exists v'.\ v \rightsquigarrow v' \rightsquigarrow u$

- implied by $output\_consistent \wedge weak\_step\_consistent\_respect$

Weak combination of step consistency and step respect:

$$\forall s\ u\ t.\ s \overset{\{w.\ w \rightsquigarrow u\}}{\approx} t \longrightarrow \forall a.\ step(a,s) \overset{u}{\sim} step(a,t)$$

If additionally the policy is transitive:

$$\boxed{trans\_weak\_nonleakage \equiv \forall s\ u\ t.\ s \overset{\{w.\ w \rightsquigarrow u\}}{\approx} t \longrightarrow \forall \alpha.\ s \lhd \alpha \overset{u}{\simeq} t \lhd \alpha}$$

- implied by $weak\_step\_consistent\_respect \wedge output\_consistent$

Noninfluence

combining noninterference and nonleakage:

$$\boxed{noninfluence \equiv \forall \alpha\ s\ u\ t.\ s \overset{sources(\alpha,u)}{\approx} t \longrightarrow s \lhd \alpha \overset{u}{\bumpeq} t \lhd ipurge(\alpha, u)}$$

- useful if both . . .
  - certain actions should be kept secret and
  - initially present secret data should not leak
- stronger than $noninterference$
- implied by
  $weakly\_step\_consistent \ \wedge \ local\_respect \ \wedge \ output\_consistent$
- appeared already as Main Lemma (Rushby's Lemma 5)

Outline

- Denning model

- Noninterference
  - ▶ Classical notion, unwinding
  - ▶ Access control interpretation
  - ▶ Nondeterminism

- Nonleakage and Noninfluence
  - ▶ Motivation, notion, variants
  - ▶ Noninfluence
  - ☞ **SLE 66 case study**

Infineon SLE66 Case Study: Unwinding

Security objective: secret functionality and data is not leaked

Applied notion: nondeterministic transitive weak Nonleakage

Unwinding: equality on: inputs, outputs, non-secret functions and data, phase, function availability

```
unwind :: "SLE66_state ⇒ on set ⇒ SLE66_state ⇒ bool"
unwind_def2: "(ph, s) ~A~ (ph',t) = (ph = ph' ∧ fct s = fct t ∧
                  (∀f∈fct s. output f s = output f t) ∧
                  (∀fn. F fn ∈ A ⟶ valF s fn = valF t fn) ∧
                  (∀dn. D dn ∈ A ⟶ valD s dn = valD t dn))"
```

Infineon SLE66 Case Study: Theorem

Main proof: $weak\_uni\_Step\_consistent\_respect$ for $U = \{$`-Sec`$\}$

Minor complication: invariants required (⇒ reachable states)

theorem $noleak\_Sec$: "$\bigwedge s\ t.\ [\![s \in reach\ ism;\ t \in reach\ ism;$
$((p,s),c,(p',s')) \in transs\ ;\ \ s\ \tilde{}{-}Sec\tilde{}\ t]\!] \Longrightarrow \exists t'.$
$((p,t),c,(p',t')) \in transs \wedge s'\ \tilde{}{-}Sec\tilde{}\ t'$"

Results:

- underspecified functions require nonleakage assumptions
- anticipated (non-critical) single data leakage confirmed
- availability of secret functions is leaked
  $\leadsto$ security objectives clarified: availability is public
- no other information leaked

Conclusion

- refinements and generalizations on Rushby's work
- introduction of new notions for data flow security:
  noninterference + nonleakage = noninfluence
- insights on unwinding and observation relations
- application in machine-assisted security analysis:
  - smart card processors (secrecy)
  - operating systems (process separation)

Contents

- Introduction
- Access Control
- Information Flow
- **Cryptoprotocol Analysis**
- Evaluation & Certification

Motivation then and now

Three can keep a secret, if two of them are dead.

— Benjamin Franklin

We interact and transact by directing flocks of digital packets towards each other through cyberspace, carrying love notes, digital cash, and secret corporate documents.
Our personal and economic lives rely on our ability to let such ethereal carrier pigeons mediate at a distance what we used to do with face-to-face meetings, paper documents, and a firm handshake.
How do we converse privately when every syllable is bounced off a satellite and smeared over an entire continent?
How should a bank know that it really *is* Bill Gates requesting from his laptop in Fiji a transfer of $10,000,000,000 to another bank?

Fortunately, the mathematics of cryptography can help.

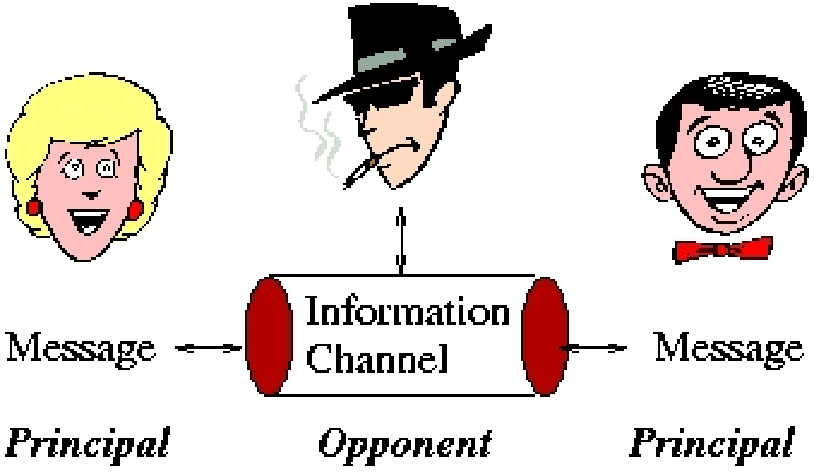
— Ron Rivest

Outline

☞ **Cryptographic Ingredients**

- Crypto Protocols
- Paulson’s Inductive Method
- Model Checking with the AVISPA Tool

What's it all about?



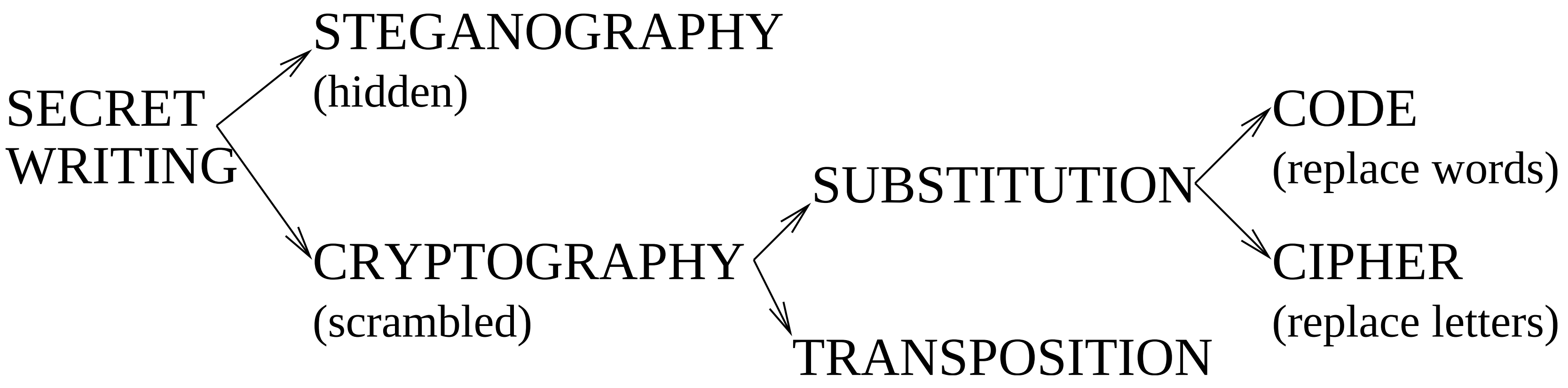
- How do we turn untrustworthy channels into trustworthy ones?

  Confidentiality: Transmited information remains secret.
  Integrity: Information not corrupted (or alterations detected).
  Authentication: Principals know who they are speaking to.

- Other goals desirable. E.g., anonymity or timeliness (freshness).

- Cryptography is the enabling technology.

Information hiding



SECRET WRITING

STEGANOGRAPHY (hidden)

CRYPTOGRAPHY (scrambled)

SUBSTITUTION

TRANSPOSITION

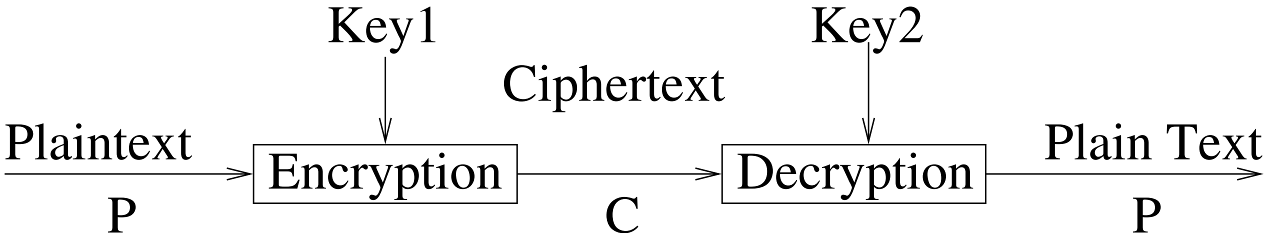
CODE (replace words)

CIPHER (replace letters)

- Cryptology: the study of secret writing.
- Steganography: the science of hiding messages in other messages.
- Cryptography: the science of secret writing.

  N.B. Terms like encrypt, encode, and encipher
  are often (loosely and wrongly) used interchangeably

General cryptographic schema



where $E_{key_1}(P) = C, \;\; D_{key_2}(C) = P$

- Security depends on secrecy of the key, not the algorithm.
- Encryption and decryption should be easy, if keys are known.
- *Symmetric* algorithms
  - Key1 $=$ Key2, or are easily derived from each other.
- *Asymmetric* or *public key* algorithms
  - Different keys, which cannot be derived from each other.
  - Public key can be published without compromising private key.

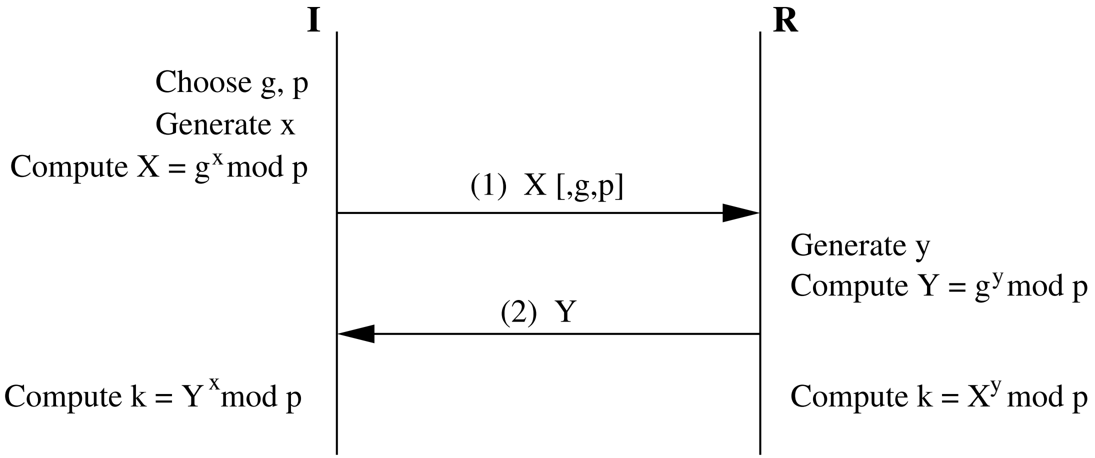
Communications using symmetric cryptography

1. Alice and Bob agree on a cryptosystem. (Can be performed in public.)
2. Alice and Bob agree on key.
3. Alice encrypts plaintext message using encryption algorithm and key.
4. Alice sends ciphertext to Bob.
5. Bob decrypts ciphertext using the same algorithm and key.

- Good cryptosystem: all security is inherent in knowledge of key and none is inherent in knowledge of algorithm.
- Benefits: offers confidentiality, integrity, and authentication.
- Main problems:
  - Keys must be distributed in secret.
  - A network of n users requires $\dfrac{n \times (n-1)}{2}$ keys.

The Diffie-Hellman Key-Exchange

- Initiator I and responder R exchange “half-keys” to arrive at mutual session key k.



I

R

Choose g, p
Generate x
Compute X = g^x mod p

(1) X [,g,p]

Generate y
Compute Y = g^y mod p

(2) Y

Compute k = Y^x mod p

Compute k = X^y mod p

- I and R agree on $g > 1$ (*generator*) and a large prime p. May be public.
- Generated keys are equal:

$$k_I = Y^x \text{ mod } p = (g^y)^x \text{ mod } p = (g^x)^y \text{ mod } p = X^y \text{ mod } p = k_R$$

- Security (i.e. secrecy of the generated keys) depends on the difficulty of computing the discrete logarithm of an exponentiated number modulo a large prime number.

Diffie-Hellman (cont.)

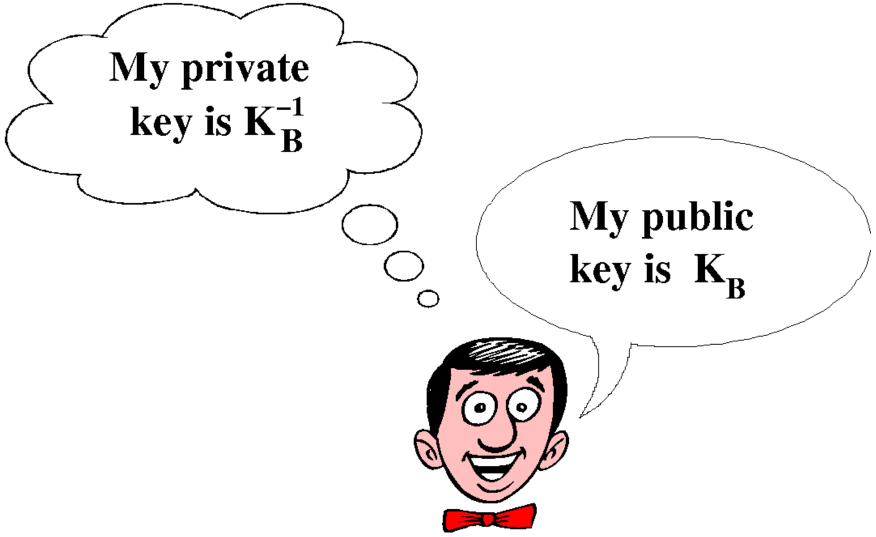
- Unknown if breaking DH as hard as computing discrete logarithms.
- Strength: creates a shared secret out of nothing!
- Strength: if the result is used as short-term session key, provides perfect forward secrecy!

  Even if an attacker acquires all long-term keys and knows all past (and future) messages encrpyted with the short-term key, he cannot recover the messge contents.

- Weakness: Keys are unauthenticated!
- Solution: sign the exponents. But this requires public/shared keys!

Communications using public-key cryptography

Bob: public key K_B and private key K_B^{-1}.



$$\{\{P\}_{K_B}\}_{K_B^{-1}} = P = \{\{P\}_{K_B^{-1}}\}_{K_B}$$

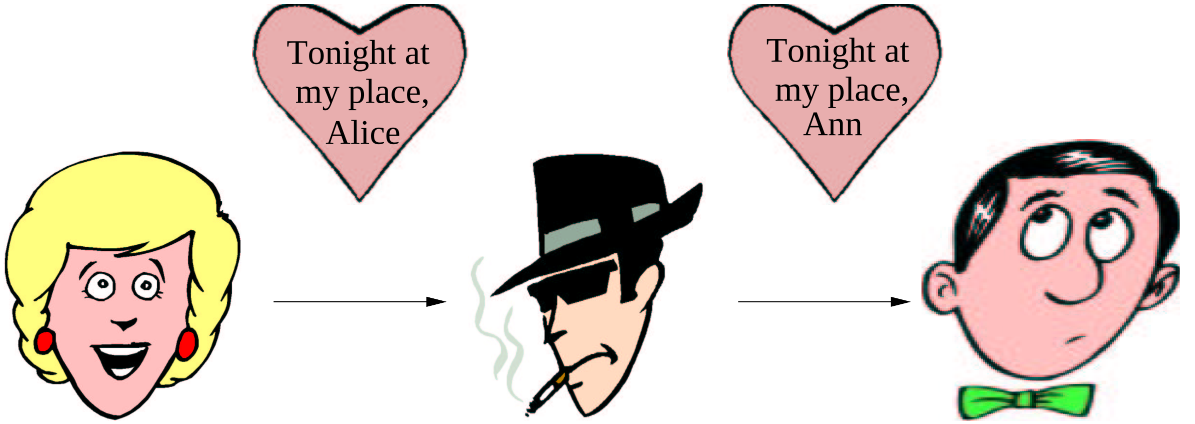
Obtain confidentiality of P by

1. Alice and Bob agree on a public-key cryptosystem.
   (Can be fixed for a network.)
2. Bob sends Alice his public key K_B.
   (Or: looked up from a database, attached to message, ...)
3. Alice encrypts message using Bob's public key K_B and sends it to Bob.
4. Bob decrypts message with his private key K_B^{-1}.

Communications using public-key cryptography (cont.)

- Good cryptosystem: all security is inherent in knowledge of key and none is inherent in knowledge of algorithm.
- It is computationally hard to deduce the private key K_B^{-1} from the public key K_B and hence decrypt (private key is sort of trap-door one-way function).
- Anyone can encrypt a message with K_B, which can then be decrypted only by owner of K_B^{-1}.
- Public-key algorithms are less efficient than symmetric ones.
- Eases key-management problem: only two keys per agent.
- Can be used to securely distribute session keys, which are then used with symmetric algorithms for further traffic ($\Rightarrow$ *hybrid cryptosystem*).
- Owner of private key K_B^{-1} can encrypt messages with it ($=$ *digital signature*), which can then be read by everybody using K_B.

The data origin problem



- Problem of proof of data origin.
- How do we know, or even prove to others, that a message originated from a particular person?
- Use a token (a “signature”) that can be applied only by the right sender and but can be checked by any receiver.

Digital signature implementation

- Public-key algorithms like RSA provide a realization of digital signatures: $\{\{P\}_{K_A}\}_{K_A^{-1}} = P = \{\{P\}_{K_A^{-1}}\}_{K_A}$ with private K_A^{-1}
- Forgery prevented by signing messages with fixed structure, e.g.,
  - ▶ Message names its sender
    1. Alice encrypts message using her private key K_A^{-1} and sends it.
    2. Bob decrypts message with Alice's public key K_A.
  - ▶ More efficient: cryptographic hash signed and sent with the message.
- Message can additionally be encrypted for confidentiality.
- Public key cryptography supports both
  - ▶ checking the origin and authenticity (also possible with shared key)
  - ▶ proving to others (non-repudiation)

    Is this possible using a shared key? No, receiver could forge signature

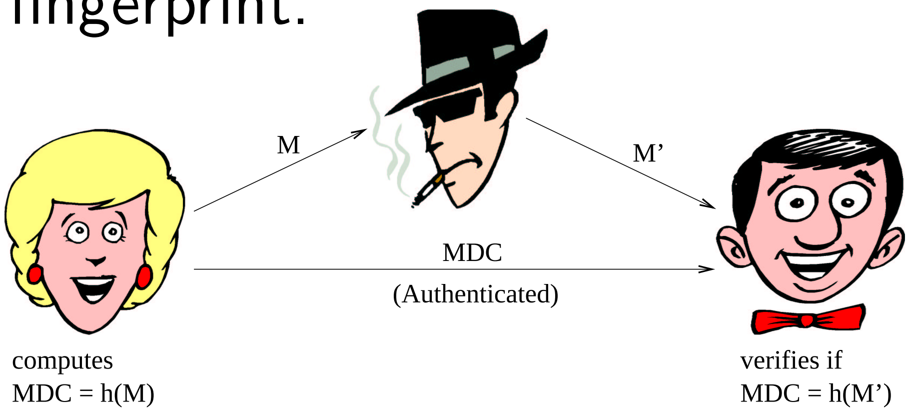
Hash Functions

- Hash functions serve as a secure *modification detection code (MDC)*.
- A *hash function* is a one-way function of all of the bits in a message so that any change in the bits results in a change in the hash code.
- Properties that a hash function H should satisfy are:

1. H can be applied to a block of data of any size.
2. H produces a fixed-length output.
3. $H(x)$ is relatively easy to compute for any input x.
4. For any given h, it is computationally infeasible to find x such that $h = H(x)$ (one-way property).
5. For any given x it is computationally infeasible to find $y \neq x$ such that $H(y) = H(x)$ (weak collision resistance, 2^{nd}-preimage resistance).
6. It is computationally infeasible to find a pair (x, y) such that $H(y) = H(x)$ (strong collision resistance).

Applications of Hash Functions

1. Message integrity: modification detection code (MDC) provides checkable fingerprint.



Requires 2nd-preimage resistance and authenticated MDC.
Typical implementation: *message authentication code (MAC)* using signed hashes. Additionally gives non-repudiation property.

2. Protect stored passwords:

- Instead of password x the value $h(x)$ is stored in the password file.
- When a user logs in giving a password x', the system applies the hash function h and compares $h(x')$ with the expected value $h(x)$.

Outline

- Cryptographic Ingredients

☞ **Crypto Protocols**

- Paulson's Inductive Method
- Model Checking with the AVISPA Tool

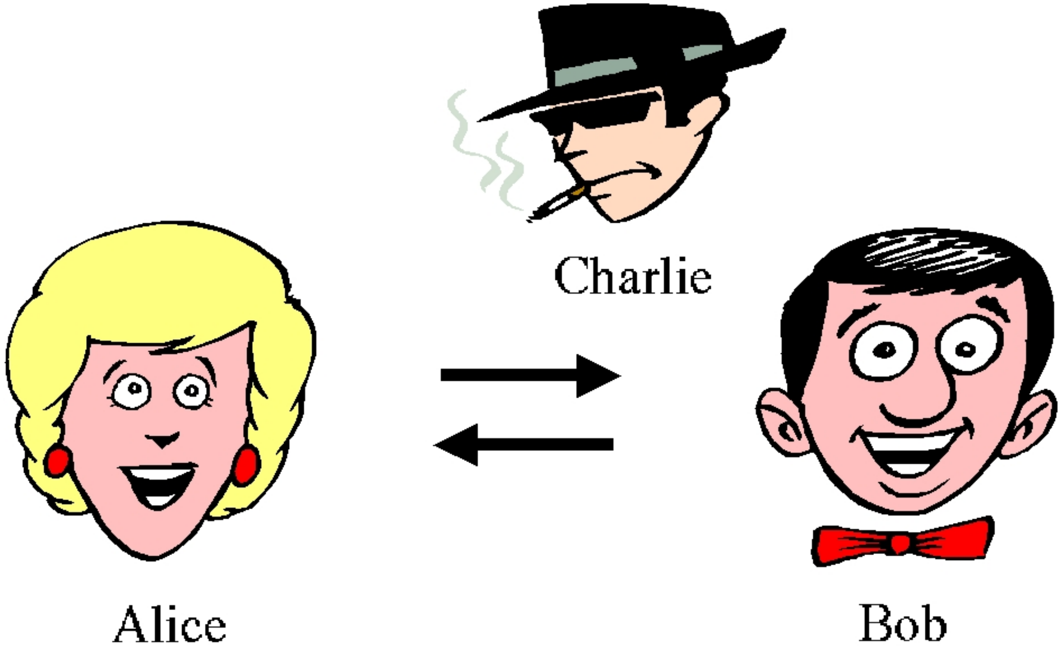
Motivation — bottom up

- How can cryptographic primitives be combined so that the result has properties that the individual building blocks lack?

- Examples:
  - Public keys may be distributed in the clear, but this requires message authentication.
  - Diffie-Hellman creates shared keys "out of nothing", but also requires message authentication.
  - Digital signatures guarantee message authentication, but not the timeliness of the message.

Motivation — top down

Example: Securing an e-banking application.



$A \rightarrow B$: “Send \$10.000 to account XYZ”
$B \rightarrow A$: “I’ll transfer it now”

How does B know the message originated from A?
How does B know A just said it?

Needham-Schroeder Public Key protocol (simplified)

Notation:

A, B	agent names (Alice, Bob)
Na	nonce ("number used only once") chosen by Alice
Ka	Alice's public key
$\{X\}_{Ka}$	message X encrypted using Ka anybody can encryt, but only Alice can recover X

Protocol:

1. $A \rightarrow B : \{Na.A\}_{Kb}$
2. $B \rightarrow A : \{Na.Nb\}_{Ka}$
3. $A \rightarrow B : \{Nb\}_{Kb}$

Goals:

Alice freshly authenticates Bob, and vice versa
(while the nonces are kept secret)

Why Are Security Protocols Often Wrong?

Simple algorithms built from simple primitives, but complicated by

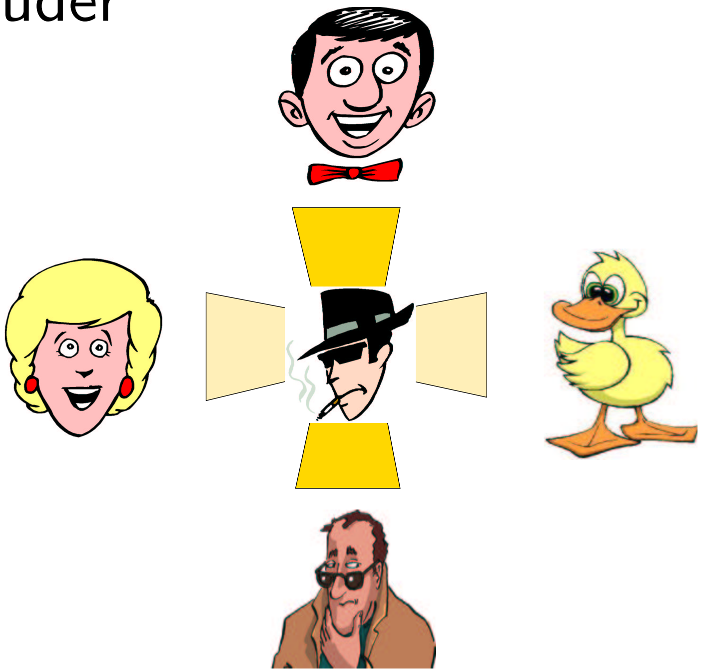
- vague specifications
- obscure concepts
- concurrency
- a hostile environment

Theses:

- A protocol without clear goals (and assumptions) is useless.
- A protocol without a proof of correctness is probably wrong.

Dolev-Yao Intruder Model

Intruder has full control over the network — he *is* the network:



- all messages sent by principals go to the intruder
- operations the intruder can do on messages:
  - forward, replay, suppress
  - decompose and analyze (if keys known)
  - modify, synthesize
  - send anywhere
- intruder cannot break cryptography
- intruder may play role(s) of (normal) principals
- intruder gains knowlege of compromised principals

Outline

- Cryptographic Ingredients
- Crypto Protocols

☞ **Paulson's Inductive Method**

- Model Checking with the AVISPA Tool

Paulson's Inductive Method

Events**:** `Says A B X:` A sends B message X $A \rightarrow B : X$

`Notes A X :` A stores/remembers X

Event *trace*: sequences of events

$$\begin{array}{l} A \rightarrow B : M_1 \\ C \rightarrow D : P_1 \\ B \rightarrow A : M_2 \\ D \rightarrow C : P_2 \\ \vdots \end{array}$$

Trace-based interleaving semantics: protocol denotes a trace set.

Interleavings of (partial) protocol runs and attacker messages.

Dolev-Yao model: the attacker controls the network.

Foundations for a formal model

- Inductive definitions are common in mathematics/informatics.
- Example: the set of binary trees $\mathcal{T}$ is the smallest set such that:

  1. $nil \in \mathcal{T}$
  2. If $t_1 \in \mathcal{T}$ and $t_2 \in \mathcal{T}$, then $node(t_1, t_2) \in \mathcal{T}$.

- Inductive definitions can be fully formalized in logic.
  - As set of Horn Clauses (as above) or as least fixedpoint of a monotone function over some universe.
  - Formalization possible in set-theory or higher-order logic.
  - Reasoning principle: (structural) induction over trees, rule induction.

Modeling: protocol as an inductively defined set

$$\begin{aligned} A \to B &: \ \{A.N_A\}_{K_B} \\ B \to A &: \ \{N_A.N_B\}_{K_A} \\ A \to B &: \ \{N_B\}_{K_B} \end{aligned}$$

Set P formalizes protocol steps.

0. $\langle\rangle \in P$

1. $t, A \to B : \{A.N_A\}_{K_B} \in P$ if $t \in P$ and $fresh_t(N_A)$

2. $t, B \to A : \{N_A.N_B\}_{K_A} \in P$ if $t \in P$, $fresh_t(N_B)$, and $A' \to B : \{A.N_A\}_{K_B} \in t$

3. $t, A \to B : \{N_B\}_{K_B} \in P$ if $t \in P$, $A \to B : \{A.N_A\}_{K_B} \in t$ and $B' \to A : \{N_A.N_B\}_{K_A} \in t$

4. $t, Spy \to B : X \in P$ if $t \in P$ and $X \in synthesize(analyze(knows(Spy, t)))$

Rules 0–3 formalize the protocol steps and rule 4 the attacker model.

Agents and Messages

- agent A,B, . . . = `Server` | `Friend` i | `Spy`

- message X,Y, . . .

=	`Agent` A	Agent name
$\mid$	`Number` N	Guessable number, timestamp, ...
$\mid$	`Nonce` N	Unguessable number
$\mid$	`Key` K	Crypto key (unguessable)
$\mid$	`Hash` X	Hashing
$\mid$	X.Y	Compound message
$\mid$	$\{X\}_K$	Encryption, public- or shared-key

- messages form free algebra (with injective constructors) $\leadsto$ messages have unique structure $\leadsto$ no type-flaw attacks

Defining Protocols

- $\mathtt{traces} : \wp(message^*)$
- defined inductively:

$$\frac{}{[] \in \mathtt{traces}}$$

$$\frac{evs \in \mathtt{traces} \qquad X \in \mathtt{synth}\ (\mathtt{analz}\ (\mathtt{knows}\ \mathtt{Spy}\ evs))}{\mathtt{Says}\ \mathtt{Spy}\ B\ X \frown evs \in \mathtt{traces}}$$

for every transition of agent A sending message Y (containing a fresh nonce N) to B, if condition P holds and A has received X and noted Z:

$$\frac{evs \in \mathtt{traces} \qquad P \qquad \mathtt{Says}\ C\ A\ X \in \mathtt{set}(evs) \qquad \mathtt{Nonce}\ N \notin \mathtt{used}\ evs \qquad \mathtt{Notes}\ A\ Z \in \mathtt{set}(evs)}{\mathtt{Says}\ A\ B\ (Y(N)) \frown evs \in \mathtt{traces}}$$

- Suppression/loss of messages implicit
- Agents can be engaged in multiple protocol runs

Freshness

- $\texttt{parts}\ \wp(message) \to \wp(message)$:
- components potentially recoverable from a set of messages
- defined inductively:

$$\frac{X \in H}{X \in \texttt{parts}\ H} \qquad \frac{X.Y \in \texttt{parts}\ H}{X \in \texttt{parts}\ H} \qquad \frac{X.Y \in \texttt{parts}\ H}{Y \in \texttt{parts}\ H} \qquad \frac{\{X\}_K \in \texttt{parts}\ H}{X \in \texttt{parts}\ H}$$

- example: $\texttt{parts}\ \{\texttt{Agent}\ A.\texttt{Nonce}\ Nb,\ \texttt{Key}\ K\} =$
  $\{\texttt{Agent}\ A.\texttt{Nonce}\ Nb,\ \texttt{Agent}\ A,\ \texttt{Nonce}\ Nb,\ \texttt{Key}\ K\}$
- $\texttt{used} :\ event^* \to \wp(message)$
- components contained in a trace of events:
- defined recursively:

$$\begin{array}{lcl}
\texttt{used}\ [] & = & \bigcup_A \texttt{parts}\ (\texttt{initState}\ A) \\
\texttt{used}\ (\texttt{Says}\ A\ B\ X \frown evs) & = & \texttt{parts}\ \{X\} \cup \texttt{used}\ evs \\
\texttt{used}\ (\texttt{Notes}\ A\ X \frown evs) & = & \texttt{parts}\ \{X\} \cup \texttt{used}\ evs
\end{array}$$

Agent Knowledge

- $\texttt{knows} : \; agent \to event^* \to \wp(message)$
- defined recursively:

$$\begin{array}{lcl}
\texttt{knows}\ C\ [] & = & \texttt{initState}\ C \\
\texttt{knows}\ C\ (\texttt{Says}\ A\ B\ X \frown evs) & = & \texttt{knows}\ C\ evs \cup \\
 & & (\texttt{if}\ C = A \lor C = \texttt{Spy}\ \texttt{then}\ \{X\}\ \texttt{else}\ \emptyset) \\
\texttt{knows}\ C\ (\texttt{Notes}\ A\ X \frown evs) & = & \texttt{knows}\ C\ evs \cup \\
 & & (\texttt{if}\ (C = A \land C \neq \texttt{Spy}) \lor \\
 & & (A \in \texttt{bad} \land C = \texttt{Spy})\ \texttt{then}\ \{X\}\ \texttt{else}\ \emptyset)
\end{array}$$

- abbreviation: $\texttt{spies} \equiv \texttt{knows}\ \texttt{Spy}$
- properties: e.g. $X \in \texttt{spies}\ evs \longrightarrow X \in \texttt{initState}\ \texttt{Spy} \lor$ $\exists A\ B.\ \texttt{Says}\ A\ B\ X \in \texttt{set}(evs) \lor (\texttt{Notes}\ A\ X \in \texttt{set}(evs) \land A \in \texttt{bad})$:

  The intruder has initial knowledge and learns all messages sent, as well as all messages noted by compromised ("bad") principals.

Analyzing Messages

- $\texttt{analz} : \ \wp(message) \rightarrow \wp(message)$:
- components actually derivable
- defined inductively:

$$\frac{X \in H}{X \in \texttt{analz}\ H} \qquad \frac{X.Y \in \texttt{analz}\ H}{X \in \texttt{analz}\ H} \qquad \frac{X.Y \in \texttt{analz}\ H}{Y \in \texttt{analz}\ H}$$

$$\frac{\{X\}_K \in \texttt{analz}\ H \qquad \texttt{Key}\ (\texttt{invKey}\ K) \in \texttt{analz}\ H}{X \in \texttt{analz}\ H}$$

- NB: no rule for `Hash`, because hashing is not invertible.
- properties: $\texttt{analz}\ G \cup \texttt{analz}\ H \subseteq \texttt{analz}\ (G \cup H)$, etc.

Synthesizing Messages

- $\texttt{synth} : \ \wp(message) \to \wp(message)$:
- messages constructable
- defined inductively:

$$\frac{X \in H}{X \in \texttt{synth}\ H} \qquad \frac{}{\texttt{Agent}\ A \in \texttt{synth}\ H} \qquad \frac{}{\texttt{Number}\ N \in \texttt{synth}\ H}$$

$$\frac{X \in \texttt{synth}\ H}{\texttt{Hash}\ X \in \texttt{synth}\ H} \qquad \frac{X \in \texttt{synth}\ H \qquad Y \in \texttt{synth}\ H}{X.Y \in \texttt{synth}\ H}$$

$$\frac{X \in \texttt{synth}\ H \qquad \texttt{Key}\ K \in H}{\{X\}_K \in \texttt{synth}\ H}$$

- properties: $\texttt{analz}\ (\texttt{synth}\ H) = \texttt{analz}\ H \cup \texttt{synth}\ H$, etc.

Needham-Schroeder-Lowe Protocol

```
theory NS_Public = Public:
consts ns_public :: "event list set"
inductive ns_public intros

 Nil:  "[] ∈ ns_public"
 Fake: "⟦evsf ∈ ns_public;  X ∈ synth (analz (spies evsf))⟧
        ⟹ Says Spy B X  # evsf ∈ ns_public"

 NS1:  "⟦evs1 ∈ ns_public;  Nonce NA ∉ used evs1⟧
        ⟹ Says A B {Nonce NA. Agent A}(pubEK B) # evs1 ∈ ns_public"

 NS2:  "⟦evs2 ∈ ns_public;  Nonce NB ∉ used evs2;
         Says A' B {Nonce NA. Agent A}(pubEK B) ∈ set evs2⟧
    ⟹ Says B A {Nonce NA. Nonce NB. Agent B}(pubEK A) # evs2 ∈ ns_public"

 NS3:  "⟦evs3 ∈ ns_public;
         Says A  B {Nonce NA. Agent A}(pubEK B) ∈ set evs3;
         Says B' A {Nonce NA. Nonce NB. Agent B}(pubEK A) ∈ set evs3⟧
     ⟹ Says A B {Nonce NB}(pubEK B) # evs3 ∈ ns_public"

lemma "∃NB. ∃evs ∈ ns_public. Says A B {Nonce NB}(pubEK B) ∈ set evs"
```

Needham-Schroeder-Lowe: Properties for Alice

```
lemma Spy_analz_priEK :
      "⟦evs ∈ ns_public⟧ ⟹ (Key (priEK A) ∈ analz (spies evs)) = (A ∈ bad)"

lemma no_nonce_NS1_NS2: "⟦evs ∈ ns_public;
       {|Nonce NA. Agent A|}(pubEK B) ∈ parts (spies evs);
       {|NA'. Nonce NA. Agent D|}(pubEK C) ∈ parts (spies evs)⟧
       ⟹ Nonce NA ∈ analz (spies evs)"

lemma unique_NA:
      "⟦{|Nonce NA. Agent A|}(pubEK B) ∈ parts(spies evs);
       {|Nonce NA. Agent A'|}(pubEK B') ∈ parts(spies evs);
       Nonce NA ∉ analz (spies evs); evs ∈ ns_public⟧ ⟹ A=A' ∧ B=B'"

theorem Spy_not_see_NA:
      "⟦Says A B {|Nonce NA. Agent A|}(pubEK B) ∈ set evs;
        A ∉ bad;  B ∉ bad;  evs ∈ ns_public⟧ ⟹ Nonce NA ∉ analz (spies evs)"

theorem A_trusts_NS2:
      "⟦Says A  B {|Nonce NA. Agent A|}(pubEK B) ∈ set evs;
       Says B' A {|Nonce NA. Nonce NB. Agent B|}(pubEK A) ∈ set evs;
       A ∉ bad;  B ∉ bad;  evs ∈ ns_public⟧
      ⟹ Says B A {|Nonce NA. Nonce NB. Agent B|}(pubEK A) ∈ set evs"
```

Needham-Schroeder-Lowe: Properties for Bob

lemma B_trusts_NS1 : "$[\![evs \in ns\_public;\ Nonce\ NA \notin analz\ (spies\ evs);$
$\{Nonce\ NA.\ Agent\ A\}_{(pubEK\ B)} \in parts\ (spies\ evs)]\!] \Longrightarrow$
$Says\ A\ B\ \{Nonce\ NA.\ Agent\ A\}_{(pubEK\ B)} \in set\ evs$"

lemma $unique\_NB$:
"$[\![CrypT(pubEK\ A)\ \ \{\!|Nonce\ NA,\ Nonce\ NB,\ Agent\ B|\!\} \in parts\ (spies\ evs);$
$\{Nonce\ NA'.\ Nonce\ NB.\ Agent\ B'\}_{(pubEK\ A')} \in parts\ (spies\ evs);$
$Nonce\ NB \notin analz\ (spies\ evs);\ evs \in ns\_public]\!] \Longrightarrow A=A' \land NA=NA' \land B=B'$"

theorem $Spy\_not\_see\_NB$:
"$[\![Says\ B\ A\ \{Nonce\ NA.\ Nonce\ NB.\ Agent\ B\}_{(pubEK\ A)} \in set\ evs;$
$A \notin bad;\ \ B \notin bad;\ \ evs \in ns\_public]\!] \Longrightarrow Nonce\ NB \notin analz\ (spies\ evs)$"

theorem B_trusts_NS3: "$[\![A \notin bad;\ \ B \notin bad;\ \ evs \in ns\_public;$
$Says\ B\ \ A\ \{Nonce\ NA.\ Nonce\ NB.\ Agent\ B\}_{(pubEK\ A)} \in set\ evs;$
$Says\ A'\ B\ \{Nonce\ NB\}_{(pubEK\ B)} \in set\ evs;]\!]$
$\Longrightarrow Says\ A\ B\ \{Nonce\ NB\}_{(pubEK\ B)} \in set\ evs$"

theorem $B\_trusts\_protocol$: "$[\![A \notin bad;\ \ B \notin bad;\ \ evs \in ns\_public;$
$Says\ B\ A\ \{Nonce\ NA.\ Nonce\ NB.\ Agent\ B\}_{(pubEK\ A)} \in set\ evs;$
$\{Nonce\ NB\}_{(pubEK\ B)} \in parts\ (spies\ evs)]\!]$
$\Longrightarrow Says\ A\ B\ \{Nonce\ NA.\ Agent\ A\}_{(pubEK\ B)} \in set\ evs$"

Conclusions on Inductive Method

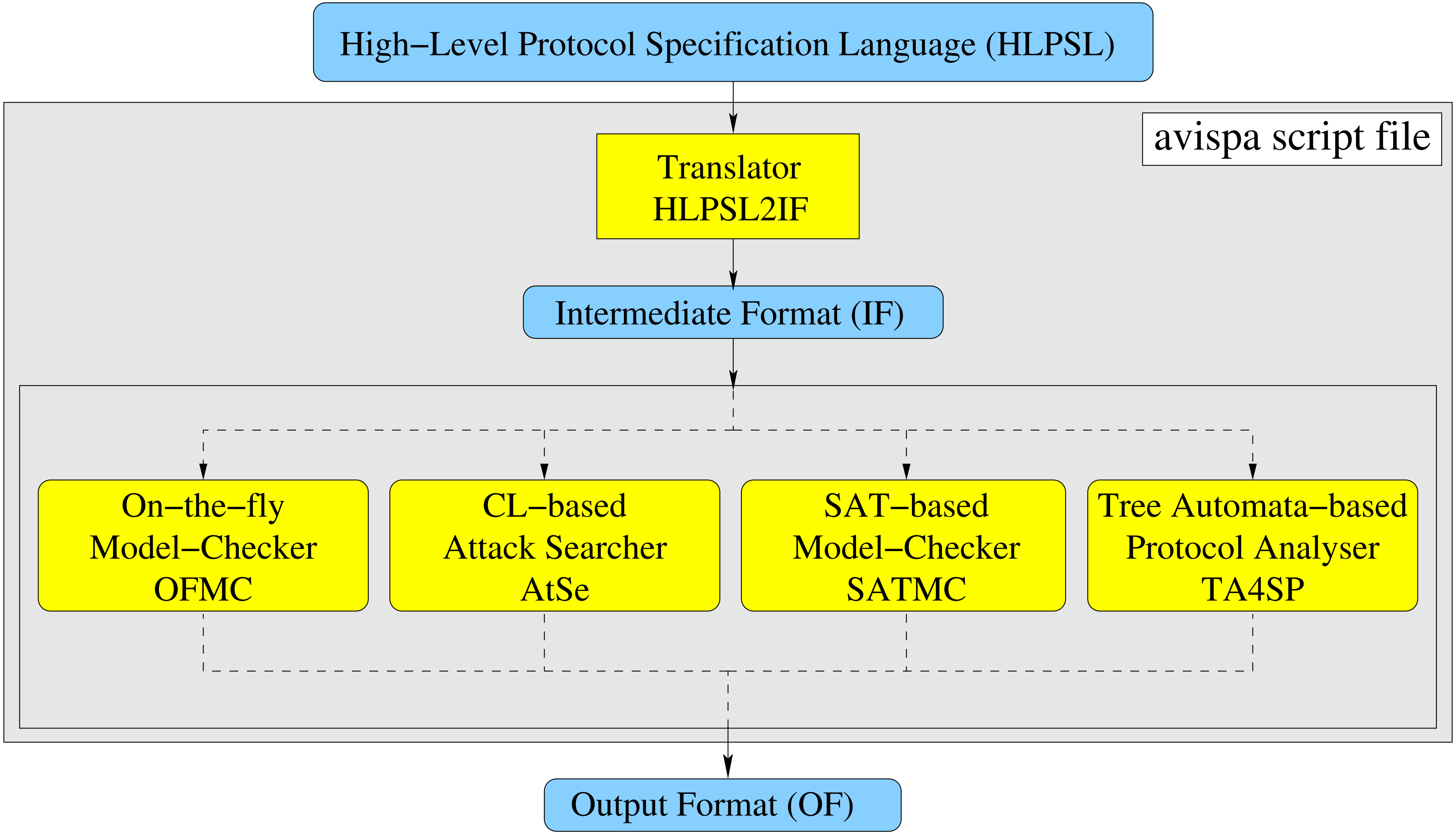
- operational protocol model (event traces)
- focuses on events, states not directly accessible
- rather simple foundations, rather easily understood
- mechanized using a theorem prover like Isabelle/HOL
- proofs are interactive, only semi-automatic
- conducting proofs gives insights in protocol features
- flaws come out in terms of unprovable goals.
- can handle complex protocols (like e.g. SET)
- analysis takes days or weeks

Outline

- Cryptographic Ingredients
- Crypto Protocols
- Paulson's Inductive Method

☞ **Model Checking with the AVISPA Tool**

AVISPA Tool



High–Level Protocol Specification Language (HLPSL)

avispa script file

Translator
HLPSL2IF

Intermediate Format (IF)

On–the–fly
Model–Checker
OFMC

CL–based
Attack Searcher
AtSe

SAT–based
Model–Checker
SATMC

Tree Automata–based
Protocol Analyser
TA4SP

Output Format (OF)

NSPK specified in HLPSL

```
%% PROTOCOL: NSPK: Needham-Schroeder Public-Key Protocol
%% VARIANT: original version (of 1978) without key server
%% PURPOSE: Two-party mutual autentication
%% MODELER: David von Oheimb, Siemens CT IC 3, January 2005
%% ALICE_BOB:
%% 1. A  - {Na.A}_Kb ----> B
%% 2. A <- {Na.Nb}_Ka ---  B
%% 3. A  - {Nb}_Kb ------> B
%% PROBLEMS: 3
%% ATTACKS: Man-in-the-middle attack,
%% where in the first session Alice talks with the intruder as desired
%% and in the second session Bob wants to talk with Alice but actually
%% talks to the intruder. Therefore, also the nonce Nb gets leaked.
%% 1.1  A  - {Na.A}_Ki --> i
%% 2.1                     i(A)  - {Na.A}_Kb  -> B
%% 2.2                     i(A) <- {Na.Nb}_Ka -  B
%% 1.2  A <- {Na.Nb}_Ka -  i
%% 1.3  A  - {Nb}Ki -----> i
%% 2.3                     i(A)  - {Nb}_Kb ----> B
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% HLPSL:
```

```
role alice (A, B: agent,
            Ka, Kb: public_key,
            SND, RCV: channel (dy))
played_by A def=

  local State : nat,
        Na, Nb: text

  init State := 0

  transition

    0.  State  = 0 /\ RCV(start) =|>
        State':= 2 /\ Na' := new() /\ SND({Na'.A}_Kb)
                   /\ secret(Na',na,{A,B})
                   /\ witness(A,B,bob_alice_na,Na')

    2.  State  = 2 /\ RCV({Na.Nb'}_Ka) =|>
        State':= 4 /\ SND({Nb'}_Kb)
                   /\ request(A,B,alice_bob_nb,Nb')
end role
```

```
role bob(A, B: agent,
         Ka, Kb: public_key,
         SND, RCV: channel (dy))
played_by B def=

  local State : nat,
        Na, Nb: text

  init State := 1

  transition

    1.  State  = 1 /\ RCV({Na'.A}_Kb) =|>
        State':= 3 /\ Nb' := new() /\ SND({Na'.Nb'}_Ka)
                   /\ secret(Nb',nb,{A,B})
                   /\ witness(B,A,alice_bob_nb,Nb')

    3.  State  = 3 /\ RCV({Nb}_Kb) =|>
        State':= 5 /\ request(B,A,bob_alice_na,Na)

end role
```

```
role session(A, B: agent, Ka, Kb: public_key) def=

  local SA, RA, SB, RB: channel (dy)

  composition

        alice(A,B,Ka,Kb,SA,RA)
     /\ bob  (A,B,Ka,Kb,SB,RB)

end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role environment() def=

    const a, b          : agent,
          ka, kb, ki    : public_key,
          na, nb,
          alice_bob_nb,
          bob_alice_na  : protocol_id
```

```
    intruder_knowledge = {a, b, ka, kb, ki, inv(ki)}

    composition

        session(a,b,ka,kb)
     /\ session(a,i,ka,ki)
     /\ session(i,b,ki,kb)

end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

goal

  secrecy_of na, nb
  authentication_on alice_bob_nb
  authentication_on bob_alice_na

end goal

environment()
```

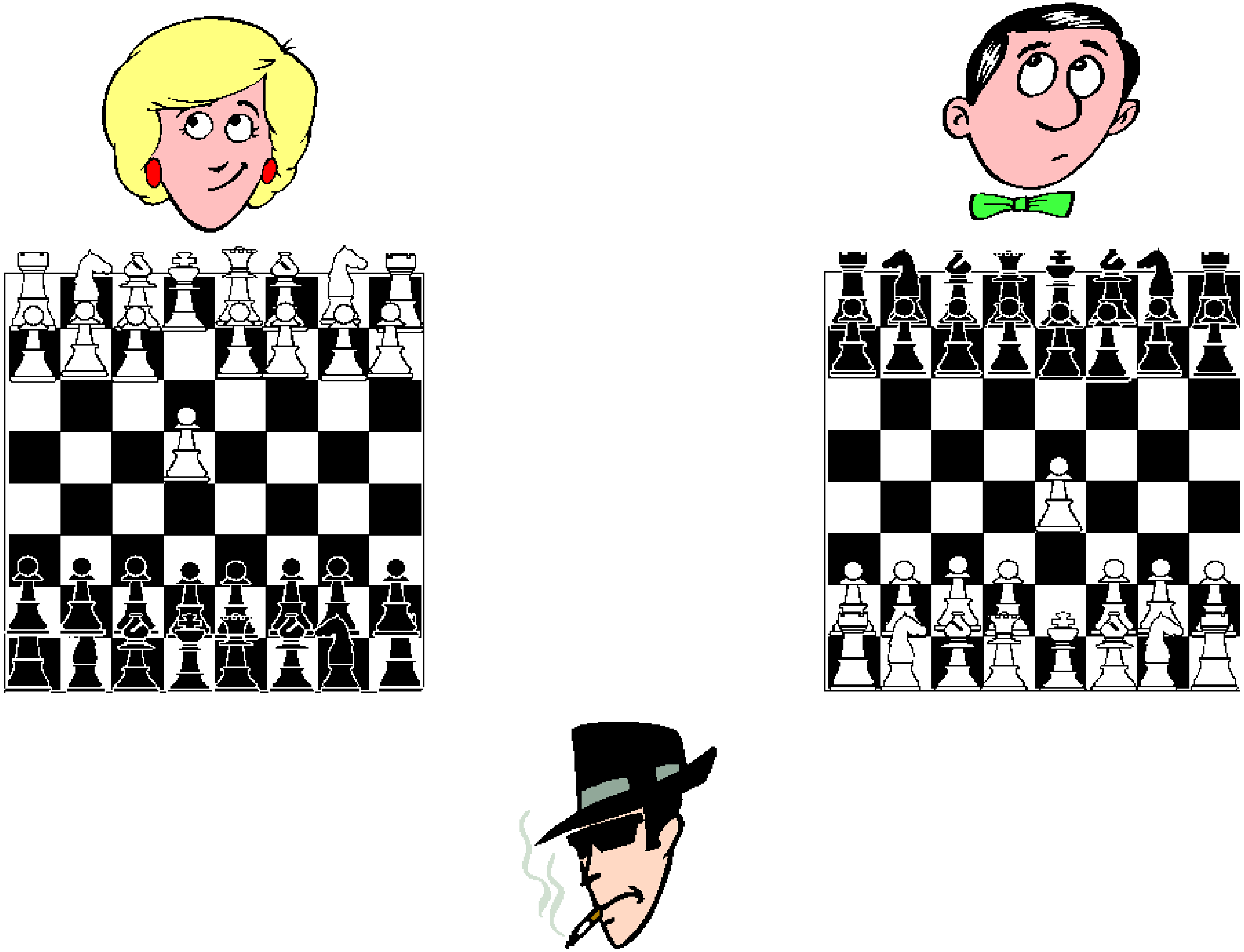
NSPK Variant with Key Server

If Alice/Bob does not know the public key of the peer, asks a key server.

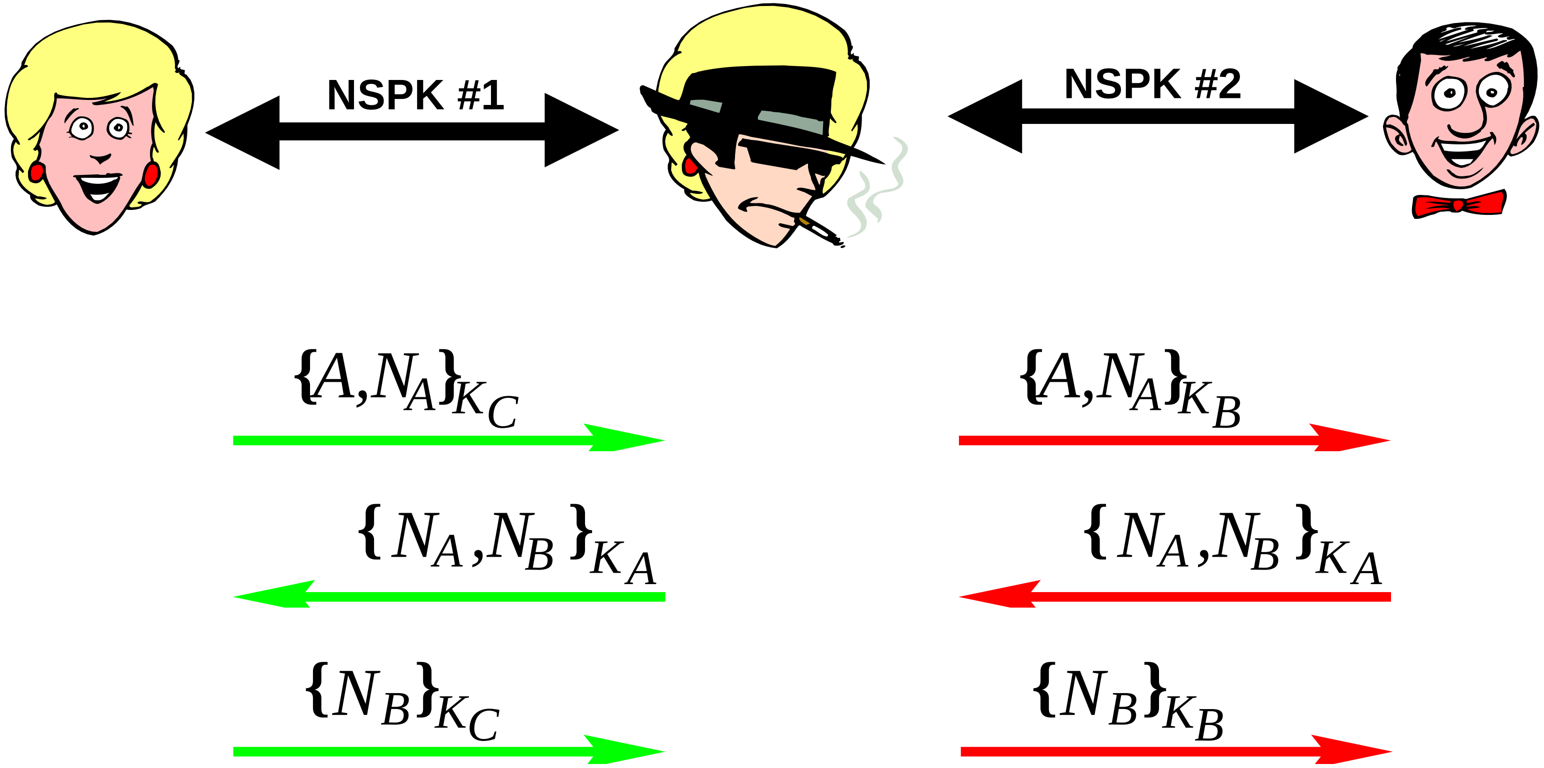
```
1a. A  -------------------- {A.B} ----------> S
1b. A <-------------------- {B.Kb}_inv(Ks) -  S
1c. A  - {Na.A}_Kb --> B
2a.                    B  - {B.A} ----------> S
2b.                    B <- {A.Ka}_inv(Ks) -  S
2c. A <- {Na.Nb}_Ka -  B
3 . A  - {Nb}_Kb    -> B

role alice (A, B: agent,
            Ka, Ks: public_key,
            KeyRing: (agent.public_key) set,
            SND, RCV: channel(dy))
played_by A def=

  local State : nat,
        Na, Nb: text,
        Kb: public_key
```

```
  init State := 0

  transition

   % Start, if alice must request bob's public key from key server
   ask.    State  = 0 /\ RCV(start) /\ not(in(B.Kb', KeyRing))
       =|> State':= 1 /\ SND(A.B)

   % Receipt of response from key server
   learn.  State  = 1 /\ RCV({B.Kb'}_inv(Ks))
       =|> State':= 0 /\ KeyRing':=cons(B.Kb', KeyRing)

   % Start/resume, provided alice already knows bob's public key
   knows.  State  = 0 /\ RCV(start) /\ in(B.Kb', KeyRing)
       =|> State':= 4 /\ Na':=new() /\ SND({Na'.A}_Kb')
                      /\ secret(Na',na,{A,B})
                      /\ witness(A,B,bob_alice_na,Na')

   cont.   State  = 4 /\ RCV({Na.Nb'}_Ka)
       =|> State':= 6 /\ SND({Nb'}_Kb)
                      /\ request(A,B,alice_bob_nb,Nb')

end role
```

Attack: Man in the Middle



Attack on Needham-Schroeder PK (details)



$\{A,N_A\}_{K_C}$ $\{A,N_A\}_{K_B}$

$\{N_A,N_B\}_{K_A}$ $\{N_A,N_B\}_{K_A}$

$\{N_B\}_{K_C}$ $\{N_B\}_{K_B}$

B believes he is speaking with A!

Examples of kinds of attack

- Replay (or freshness) attack: reuse (parts of) previous messages.
- Man-in-the-middle (or parallel sessions) attack: $A \leftrightarrow \mathcal{M} \leftrightarrow B$.
- Masquerading attack: pretend to be another principal, e.g.
  - $\mathcal{M}$ forges source address (e.g., present in network protocols), or
  - $\mathcal{M}$ convinces other principals that A's public key is $K_{\mathcal{M}}$.
- Type flaw attack: substitute a different type of message field.

  Example: use a name (or a key or ...) as a nonce.
- Reflection attack send transmitted information back to originator.

Attacks on NSPK found with OFMC

Invoking `avispa NSPK.hlpsl` yields two attacks:

```
% OFMC
% Version of 2005/06/14
SUMMARY
  UNSAFE
DETAILS
  ATTACK_FOUND
PROTOCOL
  NSPK.if
GOAL
  secrecy_of_nb
  authentication_on_bob_alice_na
BACKEND
  OFMC
COMMENTS
STATISTICS
  parseTime: 0.00s
  searchTime: 0.13s
  visitedNodes: 27 nodes
  depth: 3 plies
```

```
ATTACK TRACE
i -> (a,6): start
(a,6) -> i: {Na(1).a}_ki                 secret(Na(1).na,{a,i})
                                         witness(a.i.bob_alice_na.Na(1),i)

i -> (b,3): {Na(1).a}_kb
(b,3) -> i: {Na(1).Nb(2)}_ka             secret(Nb(2).nb,{a,b})
                                         witness(b.a.alice_bob_nb.Nb(2),i)

i -> (a,6): {Na(1).Nb(2)}_ka
(a,6) -> i: {Nb(2)}_ki                   request(a.i.alice_bob_nb.Nb(2),6)
i -> (i,17): Nb(2)                       iknows(Nb(2))
i -> (b,3): {Nb(2)}_kb                   request(b.a.bob_alice_na.Na(1),3)

% Reached State:
% secret(Nb(2).nb,{a,b})
% secret(Na(1).na,{a,i})
% request(b,a,bob_alice_na,Na(1),3)
% witness(a,i,bob_alice_na,Na(1))
% request(a,i,alice_bob_nb,Nb(2),6)
% witness(b,a,alice_bob_nb,Nb(2))
% state_alice(a,b,ka,kb,0,dummy_nonce,dummy_nonce,set_59,3)
% state_bob  (b,a,ka,kb,5,Na(1)      ,Nb(2)      ,set_67,3)
% state_alice(a,i,ka,ki,4,Na(1)      ,Nb(2)      ,set_71,6)
% state_bob  (b,i,ki,kb,1,dummy_nonce,dummy_nonce,set_75,10)
```

Attack States

Utilizing predicates `iknows`, `secret`, `witness`, and `(w)request`, which are *stable*, i.e. once they become true, they stay so.

- Violation of secrecy:

```
attack_state secrecy_of_x (X,AgentSet) :=
  secret(X,x,AgentSet) &
  iknows(X) & not(contains(i,AgentSet))
```

- Violation of weak authentication:

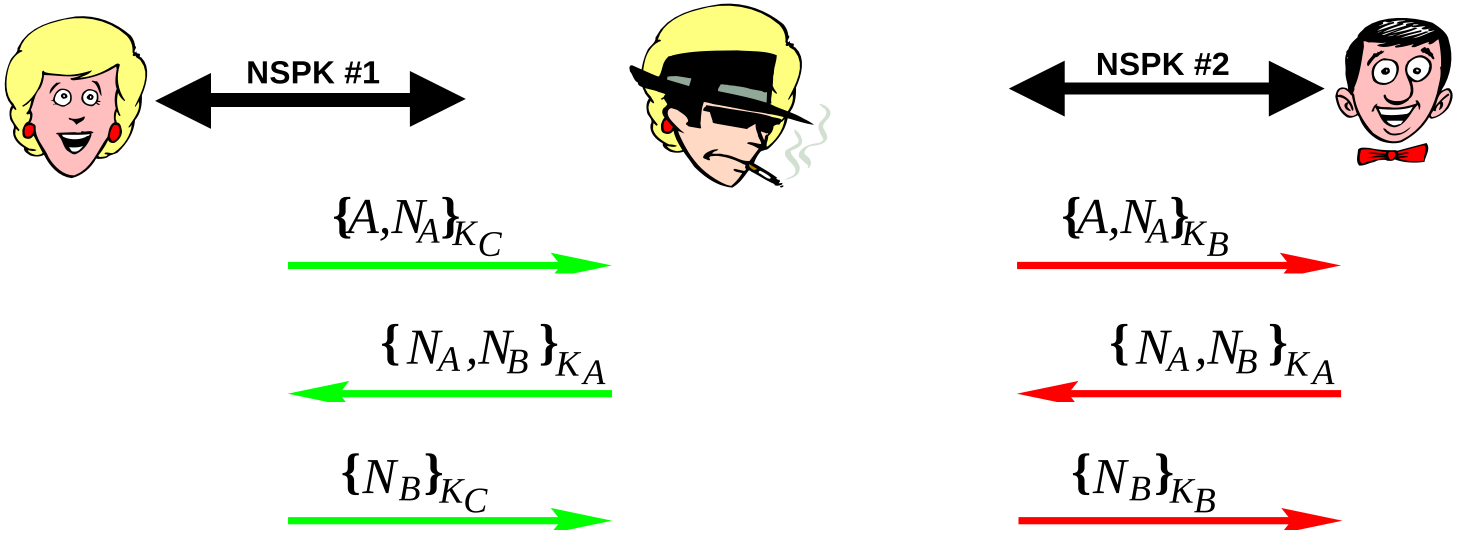
```
attack_state weak_authentication_on_a_b_n (A,B,N,SID) :=
  wrequest(A,B,a_b_n,N,SID) & not(B=i)
  not(witness(B,A,a_b_n,N))
```

- Violation of strong authentication: as before, or replay attack

```
attack_state replay_protection_on_a_b_n (A,B,N,SID1,SID2) :=
  request(A,B,a_b_n,N,SID1) & not(B=i)
  request(A,B,a_b_n,N,SID2) & not(SID1=SID2)
```

What was wrong with NSPK?

The attack:



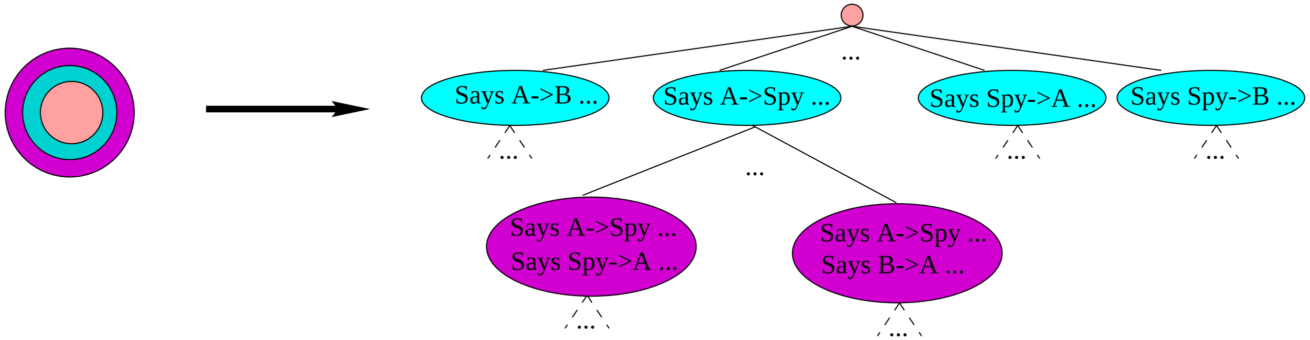
The problem: in step 2: $B \to A : \{N_A.N_B\}_{K_A}$ replayed.

Lowe's solution: B should give his name: $B \to A : \{N_A.N_B.B\}_{K_A}$

Question: Is the improved version now correct?

OFMC: Falsification using state enumeration

- Inductive definition corresponds to an infinite tree.

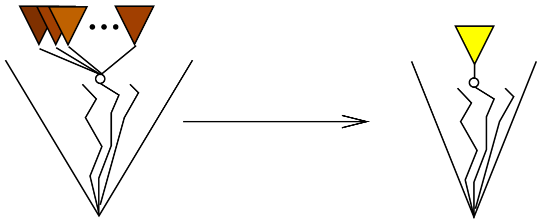


- Properties correspond to a subset of nodes, e.g., $Na \in \texttt{knows Spy}\ evs$.
- State enumeration can be used to find attack in the infinite tree.
- But naive search is hopeless! Challenges:

  Tree too wide: the spy is extraordinarily prolific!
  Too many interleavings: much “redundant” information.

Below we present three ideas for tackling these problems.

OFMC Idea 1: symbolic representations



- Spy very prolific. Generates all instances of

  $t,\ Spy \to B : X \ \in\ P$ if $t \ \in\ P$ and $X \ \in\ synthesize(analyze(knows(Spy,\ t)))$

- Alternative: allow messages to contain variables. Apply rules using unification.

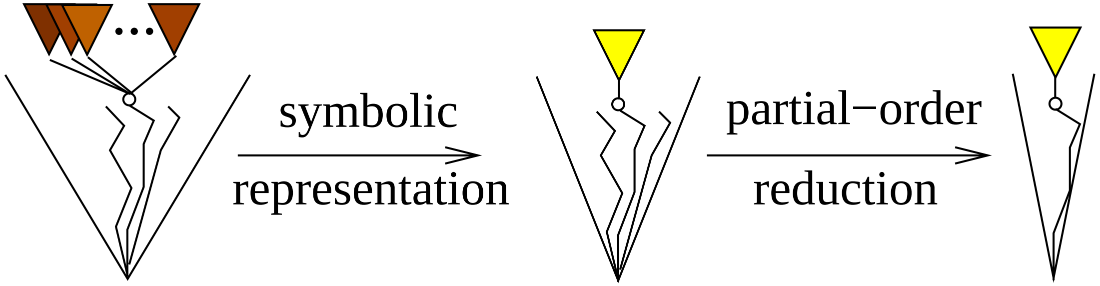
  $a \to Spy : \{a.N_a\}_{K_{Spy}}$
  $Spy \to b : \{X_2.X_3\}_{K_b}$ $\qquad X_1 = \{X_2.X_3\}_{K_b}$
  $b \to Spy : \{X_3.N_b\}_{K_{X_2}}$
  $Spy \to a : \{X_3.N_b\}_{K_{X_2}}$
  $a \to Spy : \{N_b\}_{K_{Spy}}$ $\qquad X_2 = a,\ X_3 = N_a$
  $Spy \to b : \{N_b\}_{K_b}$

  $A \to B : \quad \{A.N_A\}_{K_B}$
  $B \to A : \quad \{N_A.N_B\}_{K_A}$
  $A \to B : \quad \{N_B\}_{K_B}$

- For messages X from the Spy: $X \ \in\ synthesize(analyze(knows(Spy,\ t)))$.

  $\Longrightarrow$ Implement using narrowing with constraints.

OFMC Idea 2: partial order reduction



- Many messages are redundant. Example:

  The Spy isn't helped by repeating the same transmission.

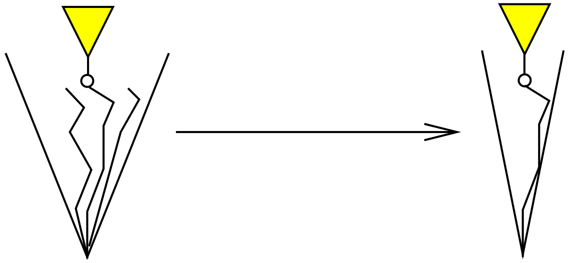
- Many orderings are redundant. Example:

  The Spy need only say X if the recipient immediately acts on it.

- Formally these define equivalence relations on traces that are respected by security properties.

$\Longrightarrow$ Restrict search to representatives!

OFMC Idea 3: lazy data structures



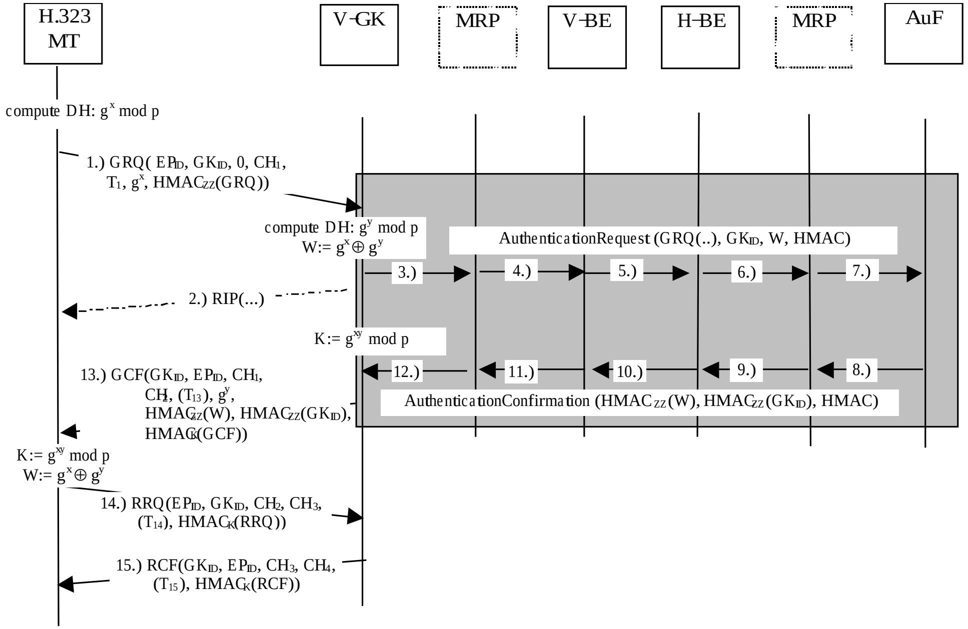
- Lazy evaluation as foundation for “on-the-fly” model checking.

  1. Apply narrowing with constraints to build infinite search tree.
  2. Use partial order reduction to build a reduced tree.
  3. Search the reduced tree by iterative deepening.

- Clean division of model, reduction techniques, and search.
  - Tasks are efficiently co-routined in a demand-driven fashion.
  - Modern compilers (e.g., for Haskell) produce fast binaries.

EU Project AVISPA: security sensitive protocols

- Goal: advance the state-of-the-art so that validation becomes standard practice.



- Apply to standardization of IETF, ITU, and W3C protocols.

Authentication: Kerberos, AAA, PANA, http-digest
Key agreement: IKEv2
Session control: SIP, H323
Mobility: mobile-IP, mobile QoS, mobile multi-media
End-to-End and Peer-to-Peer scenarios: SOAP, Geopriv

Conclusions on Model Checking

- operational protocol model (state transitions)
- focuses on messages and states
- simple foundations, easy to use
- mechanized, many model checkers available
- checking is (almost) automatic
- output gives no insights in protocol features
- flaws come out in terms of counterexmples: attack traces
- can handle industrial-scale protocols (like e.g. H.530)
- analysis takes hours or days

Contents

- Introduction
- Access Control
- Information Flow
- Cryptoprotocol Analysis
- **Evaluation & Certification**

Evaluation & Certification: Goals & General Approach

Goal: Gaining confidence in the security of a system

- What are the goals to be achieved?
- Are the measures employed appropriate to achieve the goals?
- Are the measures implemented correctly?

Approach: assessment (evaluation) of system security by neutral experts

- Understanding how the system's security functionality works
- Gaining evidence that security functionality is correctly implemented
- Gaining evidence that the integrity of the system is kept

Result: Successful evaluation is awarded a certificate

History of Evaluation Criteria

1985: **TCSEC** Trusted Computer System Evaluation Criteria (USA)
Particular security functionalities required

1989-93: German, UK, French, Canadian criteria

1991: **ITSEC** Information Technology Security Evaluation Criteria
Harmonisation of European criteria
ITSEC assurance levels provide basis for CC assurance levels

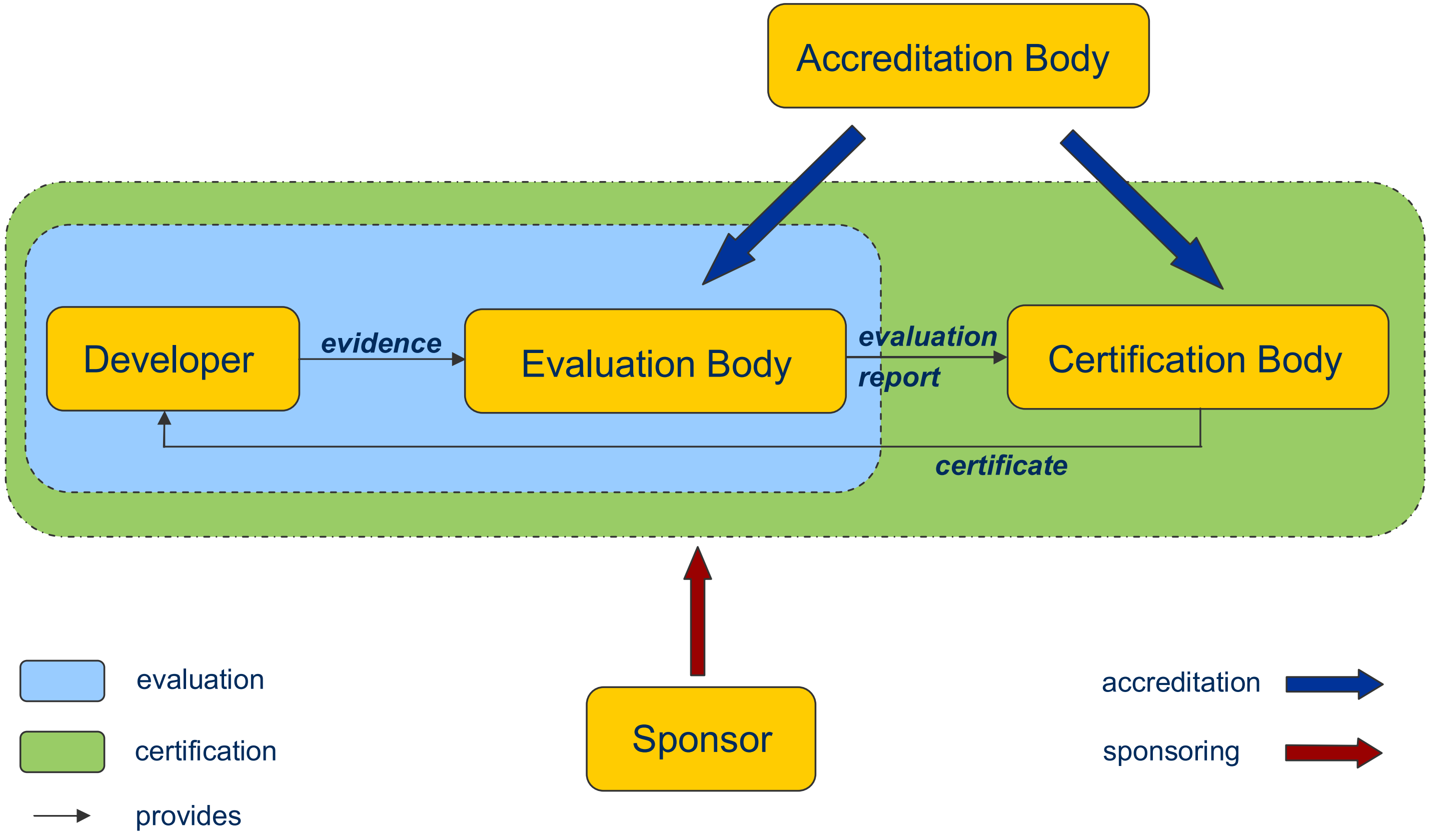
1993: Federal Criteria Draft (USA)
Attempt to update TCSEC and harmonise TCSEC+CTCPEC
Introducing Protection Profiles

1999: **CC** Common Criteria for IT Security Evaluation (ISO/IEC 15408)
Flexible approach (functional and assurance requirements components)

Common Criteria: Process Scheme

Common Criteria

...the standard for Information Security



CC: Security Target

- Definition of the Target of Evaluation (TOE)
  and separation from its environment

- Definition of the TOE's security threats, objectives and requirements

- Introduction of TOE Security Functions (TSF):
  measures intended to counter the threats

- Determination of Evaluation Assurance Level (EAL)

⇒ The Security Target is the document to which
all subsequent evaluation activities and results refer!

⇒ Interpretation of results is only reasonable if referring to the ST context

CC: Evaluation Assurance Levels

EAL1: functionally tested

EAL2: structurally tested

EAL3: methodically tested and checked

EAL4: methodically designed, tested, and reviewed,
including security policy model

EAL5: semiformally designed and tested
including formal security policy model

EAL6: semiformally verified design and tested

EAL7: formally verified design and tested

Increasing requirements on scope, depth and rigor

CC: EAL example: EAL5

In red: additional requirements compared to EAL4

- Complete source code is subject to analysis
- Formal security policy model
- Semiformal description techniques
- Modular design
- Documentation of developer’s tests up to low-level design
- Vulnerability analysis refers to moderate attack potential
- Covert channel analysis
- Comprehensive configuration management

CC: How to scale an Evaluation

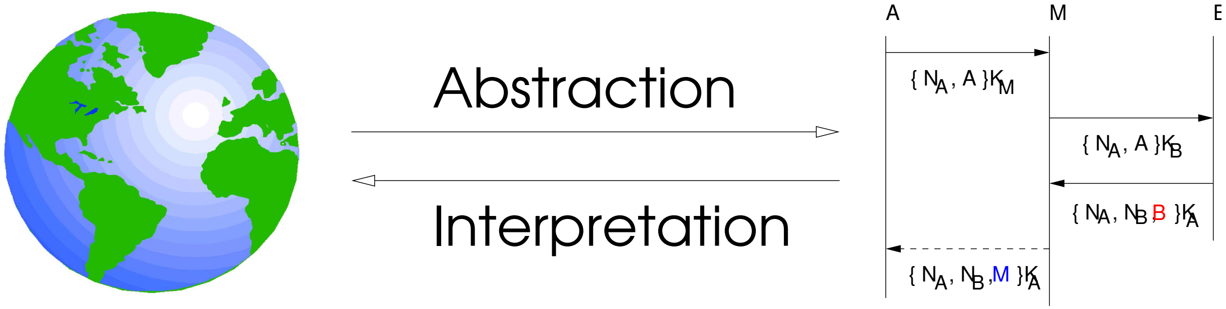
- Separation of TOE and TOE environment
- Detail level of TOE summary specification
- Definition of security objectives
- Definition of security functional requirements
- Strength-of-function claims
- EAL selection

Contents

- Introduction
- Access Control
- Information Flow
- Cryptoprotocol Analysis
- Evaluation & Certification

Conclusion

A formal security model is an abstract formal description of a system (and its environment) that focuses on the relevant security issues.



- improves understanding of security issues by
  - ▶ abstraction: concentration on the essentials helps to keep overview
  - ▶ systematic approach: generic patterns simplify the analysis
- prevents ambiguities, incompleteness, and inconsistencies and thus enhances quality of specifications
- provides basis for systematic testing or even formal verification and thus validates correctness of implementations

⇒ gives maximal confidence in the security of the system