copyright: Mirosław Kutyłowski

Security and Cryptography 2019

Mirosław Kutyłowski

grading criteria:

- 0-50 points exam, 0-50 points from the lab, making total 0-100
- the points for the exam: 0-3 for each problem, finally the result rescaled (say if 6 questions in total making maximum 18 points, then the sum of received points rescaled by multiplying by $\frac{50}{18}$)
- one has to pass the exam with at least 40% of possible points
- exam requires problem solving, memorizing facts is not necessary (one can bring own notices, even a tablet with the flight modus)

skills to be learned: developing end-to-end security systems, flawless in the real sense!

presence: obligatory during the lectures (will be controlled), please report any justified absence

exam date: during the semester, after finishing each chapter a short test or take-home exam

place: Monday 10:15-12 L1, 110 , Tuesday 12:15-14 L1, 110

lab: Tuesday 10:15-12 L1, with dr Kubiak

I. ACCESS CONTROL

ACCESS CONTROL (AC)

- decision whether a subject (user, etc) is allowed to carry out a specific action (operation) on an object (resource)
- policy = set of rules of (axioms + derivation rules) for making the decision
- model: there are
  - Principals: users, processes, etc.
  - Operations performed by Principals: read, write, append, execute, delete, delegate...
  - Objects: the objects on which the principals perform operations: files, hardware execution (opening door lock)...
- goals:
  - availability (false rejection is a disaster in some cases)

  - access exactly when needed
  - simplicity (users may have problems to understand what is going on, complicated systems are prone to be faulty)

- practical problems:
  - diversity of application scenarios, no unique generalizations
  - administrator/system provider/product provider may be malicious
  - safety issues (access control to SCADA systems for industrial installations, ...)
  - constraints must respect future states
  - designing a system much harder: hardware, software design are much easier to test, in access control one cannot create a test environment
  - cultural differences (the users may react differently, this may lead to severe misunderstanding the situation by the AC designer)

Formal description

- goal: define secure and insecure states, secure state is such that the principals get access to objects as claimed by some predefined axiomatic properties,
- (there might be some gray zone where the state is neither secure nor insecure)
- AC defines rules define a dynamic system (changing access rights by principals are possible)
- question: given: a secure system with some access rights

  ***safety:* is it possible to change the state to insecure through a number of moves that are admissible according to access rights?**

Theorem – Undecidability

the problem whether from a given secure state we can reach an insecure state is **undecidable**

Proof idea

- a Turing machine can be regarded as an access control system:
  - a head may be treated as a principal
  - the tape represents objects (each entry is a data)
  - transition function of TM specifies what is allowed concerning data and security status
- ask if a Turing machine can reach some state+tape contents (which is declared as insecure)
- the problem is in general undecidable

□

Corollary

It is necessary to restrict the freedom to define Access Control systems to avoid undecidability issues. It is hard as many rewriting systems are equivalent to Turing machines (i.e. Post Correspondence Problem).

- restrictions must be strong in order to get low complexity of formal safety
- too strict restrictions make the system less flexible and

Main Models

- approaches:
  - discretionary (the owner of an object defines the access rules)
  - mandatory (the system takes care)
- main models:
  - Access Control Matrix or Access Matrix (AM)
  - Access Control List (ACL)
  - Role-Based Access Control (RBAC)
  - Attribute-Based Access Control (ABAC)
  - lattice models
- main decisions:
  - principals: **least priviledge** given to achieve what is needed
  - objects: options are A) explicit permissions needed or B) admitted unless prohibited
  - rule administration: this is the core part of the system

AM

- Objects, Subjects , Access function
- Access Function depicted as a Matrix: each entry defines allowed operations (e.g. read, write, delete, execute, ...)

application areas: database systems, operating systems, ... The most common case: the access rights of apps in a mobile device

problems of AM:

i. lack of scalability, no flexibility,

ii. enormous effort, unless an AC system is tiny

iii. central administration, single point of failure (attack the administrator to get unrestricted control over the system)

ACL

- object centric: for each object a list of admitted principals

- corresponds to SPKI - a good cryptographic framework
- there are file systems based on ACL, Red Hut Linux: default ACL in a directory
- relational database systems, SQL systems, network administration

advantages of ACL:

close to business models, simple, easy to implement in small and well distributed systems, where access given to "local principals"

problems of ACL:

i. management of principals is hard (blacklisting a certain principal means scanning all ACL's)
ii. poor scalability unless the number of prinicipals is small and certain "locality" properties
iii. safety problems: lack of coordination between ACL may lead to problems
iv. impossible as a global access control

so ACL is not much useful for large and complicated systems

alternative approach: *tickets* (like in Kerberos - to be discussed later)

RBAC

- organization:
  - *subjects* assigned to *roles*
  - *permissions* assigned to *roles*
- a textbook example: a hospital system with differents roles: doctor, nurse, patient, accounting staff, family members of a patient, insurance company controller, ...
  i. a patient may read own record but not write or erase it
  ii. a doctor can write/read a record (but not erase)
  iii. a nurse can write records specifying activities, but not diagnosis part (reserved for the doctor's role)
  iv. ...

  But: even in this example we get into troubles: how to meet the requirements of GDPR: how to formalize "patient's own medical record"? Impossible with roles only.
- RBAC1:
  - hierarchy of roles (a directed acyclic graph)
  - a role gets all permissions from its lower roles in the hierarchy,

  advantages: description might be much smaller, logic of the system follows a strict military fashion

  disadvantages: expressive power the same as for RBAC

- RBAC2: RBAC0+constraints for RBAC matrices:
  - mutual exclusion: a user can get at most one role in a set, permission can be granted to only one role in a set, ...
  - cardinality: the maximum number of subjects with a given role might be set
  - prerequisitives: necessary to have role A before getting role B
- RBAC3=RBAC1 consolidated with RBAC2
- advantages of RBAC:
  - simple management of users
  - suitable for large systems with a simple "military" logic
- problems of RBAC:
  - no fine tuning of access rights
  - problems with dynamic changes
  - no useful in open systems

LATTICE models

- many security levels organized in a directed acyclic graph,
- c**onfidentiality policy Bell-LaPadula:**
  - information gathering like in army
  - levels
  - read allowed only if the subject's security level dominates the object's security level (read-down)
  - write allowed only if the subject's security level is dominated by the object's security level (write-up)
  - tranquility property: changing the security level of an object concurrent to its use is not possible
- **Integrity Policy Biba**
  - dual to BLP, like assigning orders in an army
  - write-down, read-up

- **Ring model**
  - a version of Biba model for operating system, focus on integrity
  - reading up is allowed, writing down is allowed. But within a specified ring (a, b) (between layers a and b)

Denning's Lattice Model

- a general model for information flow
- a partial order of different levels, not always a linear order
- lub operation: least upper bound of two nodes
- takes into account implicit and explicit data flows
- static and dynamic binding to levels may be considered

Explicit and implicit data flows

- explicit: e.g. via an assignment $b := a$
- implicit: via conditional: if $a = 0$ then $b := c$

  information on a is leaked via a change/no change in b
- programs considered composed of assignments via composing sequences and conditionals

Security rules

- explicit flows according to the rules
- for a sequence each single element must be secure
- a conditional is secure if
  - the flows in the body are secure
  - the implicit flow by conditional is secure

Binding to levels

- static: binding to levels is fixed (Biba, BLP, ...)
  - security checked during runtime
  - Data Mark Machine: mark data in the program, mark of a depends on its location
- dynamic:
  - huge problems with implicit data flows – the fact what is at a given level leaks data

- High Water Mark technique: after executing $b := a$ the level of b is raised to the least upper bound of a and b (in case of BLP, for Biba use Low Water Mark)

Decentralized Label Model

- any entity has a list of labels
- operation allowed if the lists of subject and object intersect
- semantics:
  - constructions such as "if appropriate" for exceptions

Chinese Wall (the name is misleading)

- focused on conflicts of interest
- Conflict of Interest Classes (CIC) – from each class only one dataset allowed
- In each CIC some number of datasets, datasets belong to companies
- each object in some dataset
- an extra CIC of one dataset of sanitized objects (data freely available)
- allowed access:
  - S can read O only if
    - S has already read an object from the same dataset as O
    - S has not read any object from the CIC containing O
  - writing more complicated: userA has access to dataset CompanyA in CIC x_1, userB has access to dataset CompanyB in CIC x_1, both have access to dataset BankA in CIC x_2. Unconstrained writing in objects from BankA would enable leakage from CompanyA to CompanyB
  - S can write into O only if
    - the same condition as for read ("simple security rule"), and
    - "no object can be read which is in a different company dataset to the one for which write access is requested" - consequence is that no writing if a subject has read datasets of two companies
- an axiomatized approach enabling formal verification is possible

AC for group of principals

- examples:
  - exporting a secret key from HSM and enabling reconstruction iff k shares are used

- access granted when a smart card and a smartphone of a user are involved in identification

- in general:
  - access granted iff a certain number of subjects involved simultaneously
  - the simplest scenario: threshold scheme – at least k out of n in a group request an access
  - general case: *access structure:*
    - → a family $\mathcal{F}$ of subsets of a group which is an ideal: if $A \in \mathcal{F}$ and $A \subset B$, then $B \in \mathcal{F}$
    - → it might be difficult to provide a compact description of an ideal, however the practical cases should enable formulation as a simple Boolean formula using $\wedge$ and $\vee$ operators (and no negation $\neg$)
    - → given such a formula it is easy to create an access system based on secret sharing and authentication with the key reconstructed from secret sharing: $\wedge$ corresponds to secret sharing based on XOR, $\vee$ corresponds to Shamir's scheme 1 out of k based on polynomial of degree 1,

ABAC

recommended architecture:

- language: – standard: eXtensible Access Control Markup Language (XACML)
- policies:
  - rule → policy as a set of rules → policy sets composed of policies
  - policies might be distributed, PIP (policy information point) keeps track where to find policies
  - unique names within one PIP
  - the core part: policy evaluation engine provides automatic evaluation based on formal description, the engine might be very complicated
  - policy evaluation: process answering access queries, computation based on entities, attributes and current policies – ADF (access decision function):
    i. identify policies applicable to the request
    ii. the request evaluated against each rule from these policies
    iii. the results are combined within a policy
    iv. the results combined between policies
- subjects:
  - entities asking for access

- attributes:
  - e.g. name, ID, network domain, email address
  - numeric: age, and other multivalued
- resource:
  - actions on resources to be denied or permited (or undecided)
- data form: tree structures (or forest) typical for XML
- environment:
  - it provides a set of logical and mathematical operators on attributes of the subject, resource and environment.
- recommended architecture:
  - → Policy Enforcement Point (PEP) – handles different syntax from external systems (not everything in XACML, so translation needed)
  - → Policy Information Point (PIP) – responsible for gathering data on attributes, environment,
  - → Policy Administration Point (PAP) - provides to PDP everything concerning policies
  - → Policy Decision Point (PDP) - makes actual evaluation
- steps :
  1. policies written
  2. access request goes to PEP
  3. access request goes to context handler
  4. request for attributes value to PIP
  5. PIP gets attributes evaluation for environment, resources and subjects
  6. results returned to context handler
  7. data on resources from resources to context handler
  8. everything to PDP
  9. response from PDP with a decision
  10. response from context handler to PEP
  11. obligations from PEP to obligations service (not really within the standard what and how to do)

- major problems:

    i. creating policies is a complicated task, errors are likely

    ii. trade-off between expresiveness and complexity of evaluation

rule it consists of:

- **target:**

    a) if no target given in the rule, then the target from the policy applies

    b) target defines subject, resource and action: they should match the subject, resource and action from the request (if not then the rule is not applicable)

    c) there are shortcuts <AnySubject/>, <AnyResource/>, <AnyAction/> to match with any ...

    d) subjects for the target intepreted as subtree starting in a node, so anything below would match

    e) for resources: different options

        i. the contents of the identified node only,

        ii. the contents of the identified node and the contents of its immediate child nodes

        iii. the contents of the identified node and all its descendant nodes

    f) XML pointers may be used

- **effect:** "Permit" or "Deny" (applied if the rule is applicable)

- **condition:** Boolean expression to be satisfied (so in order to apply the rule we must have both matching with the target and condition)

- **obligation expression:** it is mandatory to be fulfilled by PEP (e.g. when giving access to an account in online banking to send SMS to the client)

- **advice expression:** may be ignored by PEP

rules need not to be consistent with each other. This is the job of policy to handle it.

* **policy:** components

- **target:** different approaches to how combine definitions of targets (must be in one of targets from rules or in all targets of the rules)

- **rule-combining algorithm identifier** determines how to combine the rules (e.g. all rules MUST have the effect Permit, or at least one Permit), some algorithms are predefined but one can define own ones

- **set of rules**
- **obligation expressions** and **advice expressions**: own for the policy

decisions of a policy:

- permit
- deny
- not applicable
- indeterminate

access granted only if "permit" and obligations fulfilled

attributes evaluation

a few flexible options:

- string matching algorithms
- XPath selection in an XML structure
- result might be *a bag* of attributes - all matchings according to the selection specified

target evaluation

- AnyOf : match if all options match, if one does not match then " no match"
- AllOf: it suffices to match one option, but in this option all components must be matched

rule evaluation:

target:	**condition:**	**rule value:**
match or no target	true	effect
match or no target	false	not applicable
match or no target	indeterminite	indeterminate
no match	*	no applicable
indeterminite	*	not applicable

Table 1.

policy evaluation:

target:	**rule value:**	**policy value:**
match	at least one rule with Effect	according to rule-combining algorithm
match	all rules with NotApplicable	not applicable
match	at least one rule Indeterminite	according to rule-combining algorithm
no match	*	no applicable
indeterminite	*	indeterminite

standard rule combining algorithms:

a) deny-overrides

b) permit-overrides

c) first-applicable

d) only-one-applicable

security issues:

i. replay attack – works unless some freshness safeguards

ii. message insertion – it may create a lot of harm, message authentication needed

iii. message deletion – the system should prevent permitting access if some message undelivered

iv. message modification - obviously one could change the decision (authentication required)

v. NotApplicable - dangerous since sometimes authomatically converted to Permit (web-servers...)

vi. negative rules - not always effective: in some cases evaluation leaves undeterminite, so there is no False and access is granted

vii. DoS - via loops in evaluating policies

example from OASIS specification:

see section 4 of

http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf

---

II. COMMON CRITERIA FRAMEWORK

http://www.commoncriteriaportal.org

Book: Using the Common Criteria for IT Security Evaluation, Debra S. Herrmann

Problem: somebody has to deploy a secure IT system, how to purchase it?

- problematic requirements according to BSI guide:

  i. **incomplete** – forgetting some threats is common

  ii. **not embedded:** not corresponding really to the environment where the product has to be deployed

  iii. **implicit:** customer has in mind but the developer might be unaware of them

  iv. **not testable**: ambiguous, source of legal disputes, ...

v. **too detailed:** unnecessary details make it harder to adjust the design

vi. **unspecified meaning:** e.g. "*protect privacy*"

vii. **inconsistent:** e.g. ignoring trade-offs

- *specification-based purchasing process* versus *selection-based purchasing process*
- the user is not capable of determining the properties of the product himself: too complicated, too specialized knowledge required, a single error makes the product useless
- specifications of concrete products might be useless for the customers – hard to understand and compare the products
- informal specifications and descriptions, no access to crucial data

Idea of Common Criteria Framework:

- standardize the process of
  - designing requirements (Protection Profile, PP) (customer)
  - designing products (Security Target ST), (developer)
  - evaluation of products (licensed labs checking conformance of implementation with the documentation) (certification body)
- international agreement of bodies from some countries (USA, France, UK, Germany, India, Turkey, Sweden, Spain, Australia, Canada, Malaysia, Netherlands, Korea, New Zeland, Italy, Turkey) but Israel only "consuming", no Poland, China, Singapore,
- idea: ease the process, reuse work, build up from standard components
- typically ST as a response for PP:
  - more detailed
  - maybe chooses some concrete options
  - maybe fulfills more requirements (more PP)
  - relation with PP should be testable

Value:

- CC certification does not mean a product is secure
- it only says that is has been developed according to PP
- assurance level concerns only the stated requirements , e.g. trivial requirements $\Rightarrow$ high EAL level (common mistake in public procurement: EAL level ... without specifying PP)
- but it is cleaning up the chaos of different assumptions, descriptions, ...

Example for PP: BAC (Basic Access Control)

- used to secure wireless communication between a reader and an e-Passport (of an old generation)
- encryption primitive

$$\mathrm{EM}(K, S) = \mathrm{Enc}(\mathrm{KB}_{\mathrm{Enc}}, S) \| \mathrm{MAC}(\mathrm{KB}_{\mathrm{Mac}}, \mathrm{Enc}(\mathrm{KB}_{\mathrm{Enc}}, S), S)$$

  where the key K is $(\mathrm{KB}_{\mathrm{Enc}}, \mathrm{KB}_{\mathrm{Mac}})$
- steps:
  1. The MRTD chip sends a nonce $r_{\mathrm{PI\mathbb{C}C}}$ to the terminal
  2. The terminal sends the encrypted challenge

$$e_{\mathrm{PCD}} = \mathrm{EM}(K, r_{\mathrm{PCD}}, r_{\mathrm{PI\mathbb{C}C}}, K_{\mathrm{PCD}})$$

     to the MRTD chip, where $r_{\mathrm{PI\mathbb{C}C}}$ is the MRTD chip's nonce, r_{PCD} is the terminal's randomly chosen nonce, and K_{PCD} is keying material for the generation of the session keys.
  3. The MRTD chip decrypts and verifies $r_{\mathrm{PI\mathbb{C}C}}$, responds with

$$e_{\mathrm{PICC}} = \mathrm{EM}(K, r_{\mathrm{PICC}}, r_{\mathrm{PCD}}, K_{\mathrm{PICC}})$$

  4. The terminal decrypts and verifies r_{PCD}
  5. both sides derive $K_{\mathrm{Enc}}, K_{\mathrm{Mac}}$ from the master key

$$K_{\mathrm{PICC}} \,\mathrm{XOR}\, K_{\mathrm{PCD}}$$

     and a sequence number derived from the random nonces (key derivation function)
- ***K* derived from information available on the machine readable zone (optical reader applied, not available via wireless connection)**
- implementation: biometric passports.
- a simple system. Really?

Common Criteria Protection Profile Machine Readable Travel Document with ICAO Application, Basic Access Control BSI-CC-PP-0055

1. Introduction

`aimed for customers looking for proper products, overview`

1.1 PP reference

`basic data, registration data`

Title: Protection Profile - Machine Readable Travel Document with ICAO Application and Basic Access Control (MRTD-PP)

Sponsor: Bundesamt für Sicherheit in der Informationstechnik CC Version: 3.1 (Revision 2)

Assurance Level: The minimum assurance level for this PP is EAL4 augmented.

General Status: Final

Version Number: 1.10

Registration: BSI-CC-PP-0055

Keywords: ICAO, machine readable travel document, basic access control

1.2 TOE Overview

- Target of Evaluation
- "is aimed at potential consumers who are looking through lists of evaluated TOEs/Products to find TOEs that may meet their security needs, and are supported by their hardware, software and firmware"
- important sections:
    - Usage and major security features of the TOE
    - TOE type
    - Required non-TOE hardware/software/firmware
- Definition, Type

  which parts, which general purpose, which functionalities are present and which are missing, e.g. ATM card with no contactless payments

- Usage and security features

  crucial properties of the system (high level) and security features from the point of view of the security effect and not how it is achieved

- life cycle

  the product in the whole life cycle including manufacturing, delivery and destroying

- Required non-TOE hardware/software/firmware: other components that can be crucial for evaluation

2. Conformance Claim

- CC Conformance Claim: version of CC
- PP claim: other PP taken into account in a plug-and-play way
- Package claim: which EAL package level

EAL packages:

- The CC formalizes assurance into 6 categories (the so-called "assurance classes" which are further subdivided into 27 sub-categories (the so-called "assurance families"). In each assurance family, the CC allows grading of an evaluation with respect to that assurance family.

- 7 predefined ratings, called evaluation assurance levels or EALs. called EAL1 to EAL7, with EAL1 the lowest and EAL7 the highest

- Each EAL can be seen as a set of 27 numbers, one for each assurance family. EAL1 assigns a rating of 1 to 13 of the assurance families, and 0 to the other 14 assurance families, while EAL2 assigns the rating 2 to 7 assurance families, the rating 1 to 11 assurance families, and 0 to the other 9 assurance families

- monotonic: EALn+1 gives at least the same assurance level as EALn in each assurance families

- levels:

    - EAL1: Functionally Tested:

        - correct operation, no serious threats

        - minimal effort from the manufacturer

    - EAL2: Structurally Tested

        - delivery of design information and test results,

        - effort on the part of the developer than is consistent with good commercial practice.

    - EAL3: Methodically Tested and Checked

        - maximum assurance from positive security engineering at the design stage without substantial alteration of existing sound development practices.

        - developers or users require a moderate level of independently assured security, and require a thorough investigation of the TOE and its development without substantial re-engineering.

    - EAL4: Methodically Designed, Tested and Reviewed

        - maximum assurance from positive security engineering based on good commercial development practices which, though rigorous, do not require substantial specialist knowledge, skills, and other resources.

        - the highest level at which it is likely to be economically feasible to retrofit to an existing product line.

    - EAL5: Semiformally Designed and Tested

    - EAL6: Semiformally Verified Design and Tested

    - EAL7: Formally Verified Design and Tested

- assurance classes:
  - → development:
    - ADV_ARC - 1 1 1 1 1 1 architecture requirements
    - ADV_FSP 1 2 3 4 5 5 6 functional specifications
    - ADV_IMP - - - 1 1 2 2 implementation representation
    - ADV_INT - - - - 2 3 3 "is designed and structured such that the likelihood of flaws is reduced and that maintenance can be more readily performed without the introduction of flaws"
    - ADV_SPM - - - - - 1 1 security policy modeling
    - ADV_TDS - 1 2 3 4 5 6 TOE design
  - → guidance documents
    - AGD_OPE 1 1 1 1 1 1 1 Operational user guidance
    - AGD_PRE 1 1 1 1 1 1 1 Preparative procedures
  - → life-cycle support
    - ALC_CMC 1 2 3 4 4 5 5 Configuration Management capabilities
    - ALC_CMS 1 2 3 4 5 5 5 Configuration Management scope
    - ALC_DEL - 1 1 1 1 1 1 Delivery
    - ALC_DVS - - 1 1 1 2 2 Development security
    - ALC_FLR - - - - - - - Flaw remediation
    - ALC_LCD - - 1 1 1 1 2 Life-cycle definition
    - ALC_TAT - - - 1 2 3 3 Tools and techniques
  - → security target evaluation
    - ASE_CCL 1 1 1 1 1 1 1 Conformance claims
    - ASE_ECD 1 1 1 1 1 1 1 Extended components definition
    - ASE_INT 1 1 1 1 1 1 1 ST introduction
    - ASE_OBJ 1 2 2 2 2 2 2 Security objectives
    - ASE_REQ 1 2 2 2 2 2 2 Security requirements
    - ASE_SPD - 1 1 1 1 1 1 Security problem definition
    - ASE_TSS - 1 1 1 1 1 1 TOE summary specification

→ tests

- ATE_COV 1 2 2 2 3 3 Coverage
- ATE_DPT 1 1 3 3 4 Depth
- ATE_FUN 1 1 1 1 2 2 Functional tests
- ATE_IND 1 2 2 2 2 2 3 Independent testing

→ vulnerability assesment

- AVA_VAN 1 2 2 3 4 5 5 Vulnerability analysis

- for example, a product could score in the assurance family developer test coverage (ATE_COV):
  - 0: It is not known whether the developer has performed tests on the product;
  - 1: The developer has performed some tests on some interfaces of the product;
  - 2: The developer has performed some tests on all interfaces of the product;
  - 3: The developer has performed a very large amount of tests on all interfaces of the product
- example more formal: ALC_FLR
  - ALC_FLR.1:
    - *The flaw remediation procedures documentation shall describe the procedures used to track all reported security flaws in each release of the TOE.*
    - *The flaw remediation procedures shall require that a description of the nature and effect of each security flaw be provided, as well as the status of finding a correction to that flaw.*
    - *The flaw remediation procedures shall require that corrective actions be identified for each of the security flaws.*
    - *The flaw remediation procedures documentation shall describe the methods used to provide flaw information, corrections and guidance on corrective actions to TOE users.*
  - ALC_FLR.2:
    - ALC_FLR.1 as before
    - *The flaw remediation procedures shall describe a means by which the developer receives from TOE users reports and enquiries of suspected security flaws in the TOE.*
    - *The procedures for processing reported security flaws shall ensure that any reported flaws are remediated and the remediation procedures issued to TOE users.*

- *The procedures for processing reported security flaws shall provide safeguards that any corrections to these security flaws do not introduce any new flaws.*
- *The flaw remediation guidance shall describe a means by which TOE users report to the developer any susp ected security flaws in the TOE.*

- ALC_FLR.3:
  - first 5 as before
  - *The flaw remediation procedures shall include a procedure requiring timely response and the automatic distribution of security flaw reports and the associated corrections to registered users who might be affected by the security flaw.*
  - next 3 as before
  - *The flaw remediation guidance shall describe a means by which TOE users may register with the developer, to be eligible to receive security flaw reports and corrections.*
  - *The flaw remediation guidance shall iden tify the specific points of contact for all reports and enquiries about security issues involving the TOE.*

CEM -Common Evaluation Methodology

- given CC documentation, EAL classification etc, perform a check
- idea: evaluation by non-experts, semi-automated, mainly paper work
- mapping:
  - assurance class $\Rightarrow$ activity
  - assurance component $\Rightarrow$ sub-activity
  - evaluator action element $\Rightarrow$ action
- responsibilities:
  - sponsor: requesting and supporting an evaluation. different agreements for the evaluation (e.g. commissioning the evaluation), providing evaluation evidence.
  - developer: produces TOE, providing the evidence required for the evaluation on behalf of the sponsor.
  - evaluator: performs the evaluation tasks required in the context of an evaluation, performs the evaluation sub-activities and provides the results of the evaluation assessment to the evaluation authority.
  - evaluation authority: establishes and maintains the scheme, monitors the evaluation conducted by the evaluator, issues certification/validation reports as well as certificates based on the evaluation results
- verdicts: pass, fail, inconclusive
- parts:
  - evaluation input task (are all documents available to perform evaluation?)

- evaluation sub-activities
- evaluation output task (deliver the Observation Report (OR) and the Evaluation Technical Report (ETR)).
- demonstration of the technical competence task

Example of a sub-activity: ACE_CCL.1

Objectives

`The objective of this sub-activity is to determine the validity of various conformance claims. These describe how the PP-Module conforms to the CC Part 2 and SFR packages.`

Input

The evaluation evidence for this sub-activity is:

a) the PP-Module;

b) the SFR package(s) that the PP claims conformance to.

Action ACE_CCL.1.1E

ACE_CCL.1.1C

The conformance claim shall contain a CC conformance claim that identifies the version of the CC to which the PP-Module claims conformance.

ACE_CCL.1-1 The evaluator shall check that the conformance claim contains a CC conformance claim that identifies the version of the CC to which the PP-Module claims conformance.

The evaluator determines that the CC conformance claim identifies the version of the CC that was used to develop this PP-Module. This should include the version number of the CC and, unless the International English version of the CC was used, the language of the version of the CC that was used.

ACE_CCL.1.2C

The CC conformance claim shall describe the conformance of the PP-Module to CC Part 2 as either CC Part 2 conformant or CC Part 2 extended.

ACE_CCL.1-2 The evaluator shall check that the CC conformance claim states a claim of either CC Part 2 conformant or CC Part 2 extended for the PP-Module.

ACE_CCL.1.3C

The conformance claim shall identify all security functional requirement packages to which the PP-Module claims conformance.

ACE_CCL.1-3 The evaluator shall check that the conformance claim contains a package claim that identifies all security functional requirement packages to which the PP-Module claims conformance.

If the PP-Module does not claim conformance to a security functional requirement package, this work unit is not applicable and therefore considered to be satisfied.

...

3 Security Problem Definition

- **Object Security Problem (OSP)**: "The security problem definition defines the security problem that is to be addressed.
  - `axiomatic:` deriving the security problem definition outside the CC scope

– `crucial:` the usefulness of the results of an evaluation strongly depends on the security problem definition.

– `requires work:` spend significant resources and use well-defined processes and analyses to derive a good security problem definition.

- good example:

  Secure signature-creation devices must, by appropriate technical and operational means, ensure at the least that:

  1) The signature-creation-data used for signature-creation can practically occur only once, and that their secrecy is reasonably assured;

  2) The signature-creation-data used for signature-creation cannot, with reasonable assurance, be derived and the signature is protected against forgery using currently available technology;

  3) The signature-creation-data used for signature-creation can be reliably protected by the legitimate signatory against the use of others

- **assets:** entities that someone places value upon. Examples of assets include: - contents of a file or a server; - the authenticity of votes cast in an election; - the availability of an electronic commerce process; - the ability to use an expensive printer; - access to a classified facility.

  no threat no asset!

- **Threats:** threats to assets, what can happen that endengers assets

- **Assumptions:** assumptions are acceptable, where certain properties of the TOE environment are already known or can be assumed

  this is NOT the place for putting properties derived from specific properties of the TOE

4. Security objectives

- "The security objectives are a concise and abstract statement of the intended solution to the problem defined by the security problem definition. Their role:

  - a high-level, natural language solution of the problem;

  - divide this solution into partwise solutions, each addressing a part of the problem;

  - demonstrate that these partwise solutions form a complete solution to the problem.

- bridge between the security problem and Security Functional Requirements (SFR)

- **mapping objectives to threats**: table, each threat shoud be covered, each objective has to respond to some threat

  answers to questions:

  – what is really needed?

  – have we forgot about something?

- **rationale:** verifiable explanation why the mapping is sound

5. Extended Component Definition

- In many cases the security requirements (see the next section) in an ST are based on components in CC Part 2 or CC Part 3.

- in some cases, there may be requirements in an ST that are not based on components in CC Part 2 or CC Part 3.

- in this case new components (extended components) need to be defined

6.1 **SFR (Security Functional requirements)**

- *The SFRs are a translation of the security objectives for the TOE. They are usually at a more detailed level of abstraction, but they have to be a complete translation (the security objectives must be completely addressed) and be independent of any specific technical solution (implementation). The CC requires this translation into a standardised language for several reasons: - to provide an exact description of what is to be evaluated. As security objectives for the TOE are usually formulated in natural language, translation into a standardised language enforces a more exact description of the functionality of the TOE. - to allow comparison between two STs. As different ST authors may use different terminology in describing their security objectives, the standardised language enforces using the same terminology and concepts. This allows easy comparison.*

- predefined classes:

  - Logging and audit class FAU

  - Identification and authentication class FIA

  - Cryptographic operation class FCS

  - Access control families FDP_ACC, FDP_ACF

  - Information flow control families FDP_IFC, FDP_IFF

  - Management functions class FMT

  - (Technical) protection of user data families FDP_RIP, FDP_ITT, FDP_ROL

  - (Technical) protection of TSF data class FPT

  - Protection of (user) data during communication with external entities families FDP_ETC, FDP_ITC, FDP_UCT, FDP_UIT, FDP_DAU, classes FCO and FTP

- There is no translation required in the CC for the security objectives for the operational environment, because the operational environment is not evaluated

- customizing SFRs: `refinement` (more requirements), `selection` (options), `assignment` (values), `iterations` (the same component may appear at different places with different roles)

- rules:

  check dependencies between SFR - In the CC Part 2 language, an SFR can have a dependency on other SFRs. This signifies that if an ST uses that SFR, it generally needs to use those other SFRs as well. This makes it much harder for the ST writer to overlook including necessary SFRs and thereby improves the completeness of the ST.

  security objectives must follow from SFR's - Security Requirements Rationale section (Sect.6.3) in PP

if possible, use only standard SFR's

6.2 Security Assurance Requirements

- The SARs are a description of how the TOE is to be evaluated. This description uses a standardised language (to provide exact description, to allow comparison between two PP).

III. FIPS:

FIPS PUB 140-2, SECURITY REQUIREMENS FOR CRYPTOGRAPHIC MODULES

- Federal Information Processing Standards, NIST, recommendations and standards based on the US law
- for sensitive but unclassified information
- levels: 1-4
- Cryptographic Module Validation Program (certification by NIST and Canadian authority)
- need to use "approved security functions" if to be used in public sector, waivers concerning some features are possible (only if the application of the standard makes more harm)
- CSP: Critial security parameters – focus on protecting them by e.g. separation - logical of physical
- Levels:
  - Level 1: cryptographic module with at least one approved algorithm, no physical protection (like a PC)
  - Level 2:
    - tamper evident seals for access to CSP (critical security parameters)
    - role base authentication for operator,
    - refers to PPs, EAL2 or higher, or secure operating system
  - Level 3:
    - protection against unauthorized access and attempts to modify cryptographic module, detection probability should be high,
    - CSP separated in a physical way from the rest
    - identity based authentication+ role based of an identified person (and not solely role based as on level 2)

- CSP input and output - encrypted
- components of cryptographic module can be executed in a general purpose operating system if
  - PP fulfilled, Trusted Path fulfilled
  - EAL 3 or higher
  - security policy model (ADV.SPM1)
- or a trusted operating system

– Level 4:

- like level 3 but at least EAL4

- a more detailed overview:

	Security Level 1	*Security Level 2*	*Security Level 3*	*Security Level 4*
Cryptographic Module Specification	Specification of cryptographic module, cryptographic boundary, Approved algorithms, and Approved modes of operation. Description of cryptographic module, including all hardware, software, and firmware components. Statement of module security policy.			
Cryptographic Module Ports and Interfaces	Required and optional interfaces. Specification of all interfaces and of all input and output data paths.		Data ports for unprotected critical security parameters logically or physically separated from other data ports.	
Roles, Services, and Authentication	Logical separation of required and optional roles and services.	Role-based or identity-based operator authentication.	Identity-based operator authentication.	
Finite State Model	Specification of finite state model. Required states and optional states. State transition diagram and specification of state transitions.			
Physical Security	Production grade equipment.	Locks or tamper evidence.	Tamper detection and response for covers and doors.	Tamper detection and response envelope. EFP or EFT.
Operational Environment	Single operator. Executable code. Approved integrity technique.	Referenced PPs evaluated at EAL2 with specified discretionary access control mechanisms and auditing.	Referenced PPs plus trusted path evaluated at EAL3 plus security policy modeling.	Referenced PPs plus trusted path evaluated at EAL4.
Cryptographic Key Management	Key management mechanisms: random number and key generation, key establishment, key distribution, key entry/output, key storage, and key zeroization.			
	Secret and private keys established using manual methods may be entered or output in plaintext form.		Secret and private keys established using manual methods shall be entered or output encrypted or with split knowledge procedures.	
EMI/EMC	47 CFR FCC Part 15. Subpart B, Class A (Business use). Applicable FCC requirements (for radio).		47 CFR FCC Part 15. Subpart B, Class B (Home use).	
Self-Tests	Power-up tests: cryptographic algorithm tests, software/firmware integrity tests, critical functions tests. Conditional tests.			
Design Assurance	Configuration management (CM). Secure installation and generation. Design and policy correspondence. Guidance documents.	CM system. Secure distribution. Functional specification.	High-level language implementation.	Formal model. Detailed explanations (informal proofs). Preconditions and postconditions.
Mitigation of Other Attacks	Specification of mitigation of attacks for which no testable requirements are currently available.			

Table 1: *Summary of security requirements*

- more details:
  - roles: user, crypto officer, maintenance

- services: to operator: show status, perform self-tests, perform approved security function, bypassing cryptographic operations must be documented etc.
- auhentication: pbb of a random guess $<\frac{1}{1000000}$, one minute attemps: $<\frac{1}{100000}$, feedback obscured
- physical security:
  - full documentation,
  - if maintenance functionalities, then many features including erasing the key when accessed
  - protected holes, you cannot put probing devices through the holes
  - level 4: environmental failure protection (EFP) features or undergo environmental failure testing (EFT) – prevent leakage through unusual conditions

	General Requirements for all Embodiments	Single-Chip Cryptographic Modules	Multiple-Chip Embedded Cryptographic Modules	Multiple-Chip Standalone Cryptographic Modules
Security Level 1	Production-grade components (with standard passivation).	No additional requirements.	If applicable, production-grade enclosure or removable cover.	Production-grade enclosure.
Security Level 2	Evidence of tampering (e.g., cover, enclosure, or seal).	Opaque tamper-evident coating on chip or enclosure.	Opaque tamper-evident encapsulating material or enclosure with tamper-evident seals or pick-resistant locks for doors or removable covers.	Opaque enclosure with tamper-evident seals or pick-resistant locks for doors or removable covers.
Security Level 3	Automatic zeroization when accessing the maintenance access interface. Tamper response and zeroization circuitry. Protected vents.	Hard opaque tamper-evident coating on chip or strong removal-resistant and penetration resistant enclosure.	Hard opaque potting material encapsulation of multiple chip circuitry embodiment or applicable Multiple-Chip Standalone Security Level 3 requirements.	Hard opaque potting material encapsulation of multiple chip circuitry embodiment or strong enclosure with removal/penetration attempts causing serious damage.
Security Level 4	EFP or EFT for temperature and voltage.	Hard opaque removal-resistant coating on chip.	Tamper detection envelope with tamper response and zeroization circuitry.	Tamper detection/ response envelope with tamper response and zeroization circuitry.

Table 2: *Summary of physical security requirements*

- more details:
  - operational environment:
    - L1: separation of processes, concurrent operators excluded, no interrupting cryptographic module, approved integrity technique
    - L2: operating system control functions under EAL2, specify roles to operate, modify,..., crypto software withing cryptographic boundary, audit: recording invalid operations, capable of auditing the following events:

– operations to process audit data from the audit trail,

– requests to use authentication data management mechanisms,

– use of a security-relevant crypto officer function,

– requests to access user authentication data associated with the cryptographic module,

– use of an authentication mechanism (e.g., login) associated with the cryptographic module,

– explicit requests to assume a crypto officer role,

– the allocation of a function to a crypto officer role.

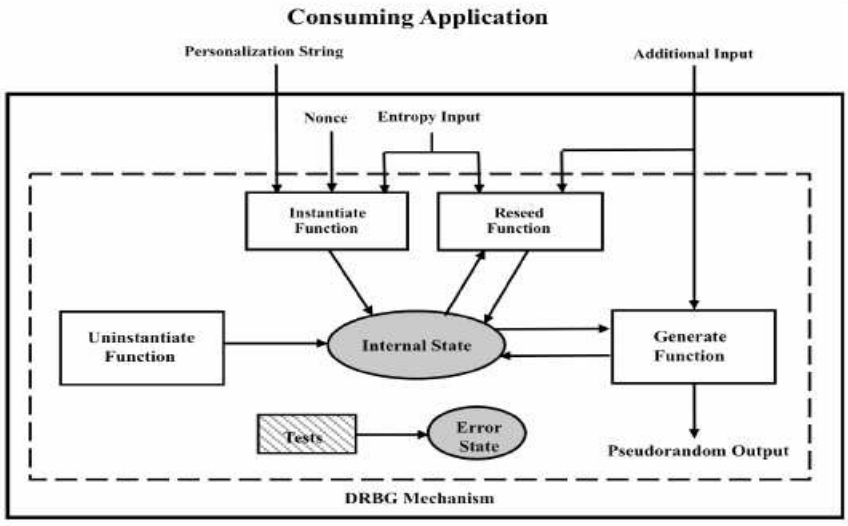
- L3: EAL3, trusted path (also included in audit trail)
- L4: EAL4

- key management:
  - non-approved RNG can be used for IV or as input to approved RNG
  - list of approved RNG: refers to an annex and annex to NIST document from 2016 (with a link to 2015)
  - list of approved key establishment methods - again links
  - key in/out: automated (encrypted) or manual (splitted in L3 and L4)

- tests: self-test and power-up. No crypto operation if something wrong. tests based on known outputs
  - Pair-wise consistency test (for public and private keys).
  - Software/firmware load test.
  - Manual key entry test.
  - Continuous random number generator test.
  - Bypass test – proper switching between bypass and crypto

FIPS Approved Random Number Generators

- nondeterministic generators not approved
- deterministic: special NIST Recommendation,
- first approved entropy source creates a seed , then deterministic part



Instantiation:

- the seed has a limited time period, after that period a new seed has to be used
- reseeded function requires a different seed
- different instantiations of a DRNG can exist at the same time, they MUST be independent in terms of the seeds and usage

Internal state:

- it contains cryptographic chain value AND the number of requests so far (each request corresponds to an output)
- different instantiations of DRBG must have separate internal states

Instantiation strength:

- formally defined as "112, 128, 192, 256 bits", intuition: number of bits to be guessed
- `Security_strength_of_output` = min(`output_length`, `DRBG_security_strength`)

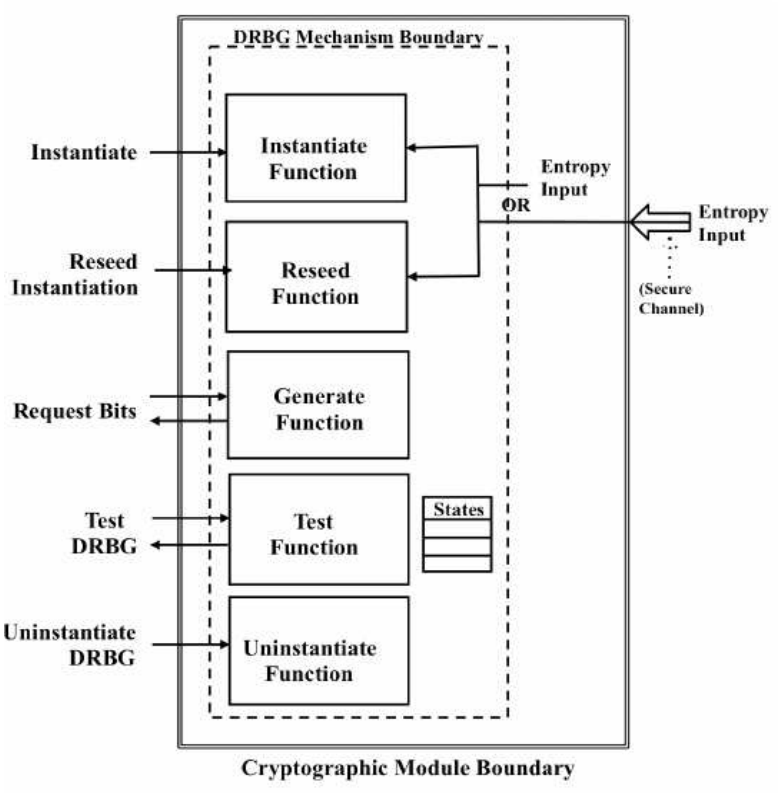
Functions executed:

- instantiate: initializing the internal state, preparing DRNG to use
- generate: generating output bits as DRNG
- reseed: combines the internal state with new entropy to change the seed
- uninstantiate: erase the internal state
- test: internal tests aimed to detect defects of the chip components

DRBG mechanism boundary:

- this is not a cryptographic module boundary

- DRBG internal state and operation shall only be affected according to the DRBG mechanism specification
- the state exists solely within the DRBG mechanism boundary, it is not accessible from outside
- information about the internal state is possible only via specified output



Seed:

- entropy is obligatory, entropy strength should be not smaller than the entropy of the output
- *approved randomness source* is obligatory as an entropy source
- reseeding: a nonce is not used, the internal state is used
- nonce: it is not a secret. Example nonces:
  - a random value from an approved generator
  - a trusted timestamp of sufficient resolution (never use the same timestamp)
  - monotonically increasing sequence number
  - combination of a timestamp and a monotonically increasing sequence number, such that the sequence number is reset iff the timestamp changes
- not used for any other purposes

reseed operation:

- "for security"
- argument: it might be better than `uninstantiate` and `instantiate` due to aging of the entropy source

personalization:

- not security critical, but the adversary might be unaware of it (analogous to a login)

resistance:

- backtracking resistance: given internal state at time t it is infeasible to distinguish between the output for period $[1, t-1]$ and a random output
- prediction resistance: *"Prediction resistance means that a compromise of the DRBG internal state has no effect on the security of future DRBG outputs. That is, an adversary who is given access to all of the output sequence after the compromise cannot distinguish it from random output with less work than is associated with the security strength of the instantiation; if the adversary knows only part of the future output sequence, he cannot predict any bit of that future output sequence that he does not already know (with better than a 50-50 chance).* – **refers only to reseeding** (before reseeding the output is predictable)

  distinguishability from random input or predicting missing output bits

specific functions:

- (status, entropy_input) = Get_entropy_input (min_entropy, min_ length, max_ length, prediction_resistance_request),
- Instantiation:
  - → checks validity of parameters
  - → determines security strength
  - → obtains entropy input and a nonce
  - → runs instantiate algorithm to get the initial state
  - → returns a handle to this DRNG instantiation

  Instantiate_function(requested_instantiation_security_strength, prediction_resistance_flag, personalization_string)

  prediction_resistance_flag determines whether consuming application may request reseeding
- Reseed:
  - → there must be an explicit request by a consuming application,
    - if prediction resistance is requested
    - if the upper bound on the number of genereted outpus reached

– due to external events

steps:

→ checks validity of the input parameters,

→ determines the security strength

→ obtains entropy input, nonce

→ runs reseed algorithm to get a new initial state

- Generate function (outputs the bits)
  Generate_function(state_handle,requested_number_of_bits,requested_security_strength, prediction_resistance_request, additional_input)
- Removing a DRBG Instantiation:
  Uninstantiate_function (state_handle)
  internal state zeroized (to prevent problems in case of a device compromise)

Hash_DRBG

comment:

– hash function considered as pseudorandom function

– one can use a random walk: $h_1 = \text{Hash}(V), h_2 = \text{Hash}(h_1), h_3 = \text{Hash}(h_2), ...$

variants:

– hash algorithms: SHA-1 up to SHA-512

– parameters determined, e.g. maximum length of personalization string

– seed length typically 440 (but also 888)

state:

→ value V updated during each call to the DRBG

→ constant C that depends on the seed

→ counter `reseed_counter`: storing the number of requests for pseudorandom bits since new entropy_input was obtained during instantiation or reseeding

instantiation:

1. `seed_material` = `entropy_input || nonce || personalization_string`
2. `seed` = `Hash_df (seed_material, seedlen)` (hash derivation function)
3. `V = seed`
4. `C = Hash_df ((0x00 || V), seedlen)`

5. Return (V, C, reseed_counter)

reseed:

1. seed_material = 0x01 || V || entropy_input || additional_input
2. seed = Hash_df (seed_material, seedlen)
3. V = seed
4. C = Hash_df ((0x00 || V), seedlen)
5. reseed_counter = 1
6. Return (V, C, reseed_counter).

generating bits:

1. If reseed_counter > reseed_interval, then return "reseed required"
2. If (additional_input $\neq$ Null), then do
   2.1 w = Hash (0x02 || V || additional_input)
   2.2 V = (V + w) mod 2^{seedlen}
3. (returned_bits) = Hashgen (requested_number_of_bits, V)
4. H = Hash (0x03 || V)
5. V = (V + H + C + reseed_counter) mod 2^{seedlen}
6. reseed_counter = reseed_counter + 1
7. Return (SUCCESS, returned_bits, V, C, reseed_counter)

Hashgen:

1. m = $\frac{\text{requested} - \text{no} - \text{of} - \text{bits}}{\text{outlen}}$
2. data = V
3. W = Null string
4. For i = 1 to m
   4.1 w = Hash (data).
   4.2 W = W || w
   4.3 data = (data + 1) mod 2seedlen
5. returned_bits = leftmost (W, requested_no_of_bits)
6. Return (returned_bits)

HMAC_DRBG

Update (used for instantiation and reseeding) HMAC_DRBG_Update (provided_data, Key, V):

1. `Key = HMAC (Key, V || 0x00 || provided_data)`
2. `V = HMAC (Key, V)`
3. If (`provided_data = Null`), then return `Key` and `V`
4. `Key = HMAC (Key, V || 0x01 || provided_data)`
5. `V = HMAC (Key, V)`
6. Return `(Key, V)`

Instantiate:

1. `seed_material = entropy_input || nonce || personalization_string`
2. `Key = 0x00 00...00`
3. `V = 0x01 01...01`
4. `(Key, V) = HMAC_DRBG_Update (seed_material, Key, V)`
5. `reseed_counter = 1`
6. `Return (V, Key, reseed_counter)`

Reseed:

1. `seed_material = entropy_input || additional_input`
2. `(Key, V) = HMAC_DRBG_Update (seed_material, Key, V)`
3. `reseed_counter = 1`
4. Return `(V, Key, reseed_counter)`.

Generate bits:

1. If `reseed_counter > reseed_interval`, then return "reseed required"
2. If `additional_input` $\neq$ `Null`, then
   `(Key, V) = HMAC_DRBG_Update (additional_input, Key, V)`
3. `temp = Null`
4. While `len (temp)` $<$ `requested_number_of_bits` do:
   4.1 `V = HMAC (Key, V)`
   4.2 `temp = temp || V`
5. `returned_bits = leftmost (temp, requested_number_of_bits)`
6. `(Key, V) = HMAC_DRBG_Update (additional_input, Key, V)`

7. `reseed_counter = reseed_counter + 1`

8. Return (`SUCCESS, returned_bits, Key, V, reseed_counter`).

CTR_DRBG

this generator is based on an encryption function, choice: 3DES with 3 keys or AES 128, 192, 256

internal state:

- value `V` of `blocklen` bits, updated each time another `blocklen` bits of output are produced
- value `Key` of `keylen-bit` bits, updated whenever a predetermined number of output blocks is generated
- `counter (reseed_counter)` = the number of requests for pseudorandom bits since instantiation or reseeding
- `ctr_len` is a parameter depending on implementation, counter field length, at least 4, at most `ctr_len`≤`blocklen`, for example important when 3DES is used: `ctr_len` is only 64

Update Process: `CTR_DRBG_Update (provided_data, Key, V)`:

where `provided_data` has length `seedlen`

1. `temp = Null`
2. While (`len (temp)`< `seedlen`, do

   2.1 If `ctr_len` < `blocklen` /* *comment: counter increased in the suffix)*

   2.1.1 `inc = (rightmost (V, ctr_len) + 1) mod` 2^{ctrlen}.

   2.1.2 `V = leftmost (V, blocklen-ctr_len) || inc`

   Else `V = (V+1) mod` 2^{blocklen}

   2.2 `output_block = Block_Encrypt (Key, V)`

   2.3 `temp = temp || output_block`
3. `temp = leftmost (temp, seedlen)`
4. `temp = temp` $\oplus$ `provided_data`
5. `Key = leftmost (temp, keylen)`
6. `V = rightmost (temp, blocklen)`.

Instantiate:

1. pad `personalization_string` with zeroes
2. `seed_material = entropy_input` $\oplus$ `personalization_string`
3. `Key` $= 0^{\text{keylen}}$
4. `V` $= 0^{\text{blocklen}}$

5. `(Key, V) = CTR_DRBG_Update (seed_material, Key, V)`.
6. `reseed_counter = 1`
7. `Return (V, Key, reseed_counter)`.

reseeding is similar

Generate:

1. If `reseed_counter > reseed_interval`, then "`reseed required`"
2. If (`additional_input`≠ `Null`), then
   2.1 `temp = len (additional_input)`.
   2.2 If (`temp`< `seedlem`) then pad `additional_input` with zeroes
   2.3 `(Key, V) = CTR_DRBG_Update (additional_input, Key, V)`.
   Else additional_input = 0^{seedlen}
3. `temp = Null`
4. While `(len (temp)<requested_number_of_bit)`, do
   4.1 If `ctr_len< blocklen`
   4.1.1 `inc = (rightmost (V, ctr_len) + 1) mod` 2^{ctrlen}
   4.1.2 `V = leftmost (V, blocklen-ctr_len) || inc`
   Else `V = (V+1) mod` 2^{blocklen}
   4.2 `output_block = Block_Encrypt (Key, V)`.
   4.3 `temp = temp || output_block`
5. `returned_bits = leftmost (temp, requested_number_of_bits)`
6. `(Key, V) = CTR_DRBG_Update (additional_input, Key, V)`
7. `reseed_counter = reseed_counter + 1`
8. `Return (SUCCESS, returned_bits, Key, V, reseed_counter)`.

Models and solutions based on AES

data:

inside the generator:

- key
- state

from outside:

- input

leakage:

- only data from computations are leaked
- bounded leakage: λ entropy bits per iteration, some (probabilistic) leakage function
- non-adaptive leakage: leakage function fixed in advance
- simultable leakage: there is always some leakage output. The best situation when the real leakage can be simulated and the adverary cannot distinguish if it is a simulation

knowledge of adversary (some options):

- Chosen-Input Attack (CIA): key hidden, state known, input chosen by adversary
- Chosen-state Attack (CSA): key hidden, state chosen by the adversary, input known
- Known-Key Attack (KKA): key known, state hidden, inputs known

almost not considered: key known, state known, inputs with low entropy and somewhat predictable

Some constructions

Construction 1 (against CIA, CSA,KKA)

- setup: 128-bit string X chosen at random
- initialize: 128 bit strings K and S chosen at random
- generate function (with input I):
  1. $U := K \cdot X^2 + S \cdot X + I \bmod 2^{128}$
  2. $S := \mathrm{AES}_U(1)$
  3. output $\mathrm{AES}_U(2)$

Construction 2 (separating entropy extraction and output generation - separating information theoretic arguments from cryptographic procedures)

- setup: 1024-bit strings X, X' chosen at random
- initialize: 1024-bit string S chosen at random
- refresh with I:
  1. $S := S \cdot X + I$
- next (with input I):
  1. $U := [X' \cdot S]_{256}$
  2. $S := \mathrm{AES}_U(1)\mathrm{AES}_U(2)....\mathrm{AES}_U(8)$
  3. $R := \mathrm{AES}_U(9)$

DUAL EC -standardized backdoor

story:

- NIST, ANSI, ISO standard for PRNG, from 2006 till 2014 when finally withdrawn
- problems reported during standardization process: bias that would be unacceptable for constructions based on symmetric crypto, finally 2007 a paper of Dan Shumow and Niels Ferguson with an obvious attack based on kleptography (199*)
- DUAL EC dead for crypto community since 2007 but not in industry
  - deal NSA -RSA company (RSA was paid to include DUAL EC)
  - products with FIPS certification had to implement Dual EC, no certificate when P and Q generated by the device
  - discouraging generation of own P and Q by NIST
  - used in many libraries: BSAFE, OpenSSL, ...
  - in 2007 an update of Dual EC that that makes the backdoor more efficient
  - changes in the TCP/IP to ease the attack (increasing the number of consecutive random bits sent in plaintext)

algorithm:

- basic scheme:
  - → state $s_{i+1} = f(s_i)$, where s_0 is the seed
  - → generating bits: $r_i := g(s_i)$
  - → both f anf g must be one-way functions in a cryptographic sense
- Dual EC, basic version:
  - → points P and Q "generated securely" by NSA but information classified,
  - → $s_{i+1} := x(s_i \cdot P)$ (that is, the "x" coordinate of the point on an elliptic curve)
  - → $r_i := x(s_i \cdot Q)$
  - → this option used in many libraries
- Dual EC with additional input:
  - → if additional input given then update is slightly different:
  - → $t_i := s_i \oplus H(\text{additonalinput}_i), s_{i+1} := x(t_i \cdot P)$

Attack: with a backdoor d, where $P = d \cdot Q$

- for basic version:
  - → from r_i reconstruct the EC point R_i (immediate, two options)

→ compute s_{i+1} as $x(d \cdot R_i)$ (no knowledge of the internal state s_i required)

- for additional input:
  - it does not work in this way since the $\oplus$ operation is algebraically incompatible with scalar multiplication with the points of elliptic curve
  - however it does not help much: frequently more than one block r_i is needed by the consuming application and simply the next step(s) is executed without additional input – at this moment the adversary learns the internal state
  - the attacker have problems if cannot trace the additional input: gradually looses control over the state of PRNG

Dual EC 2007:

- an update to "increase security"
- an extra step after request for bits, before using additional input:
  - → $s_{i+1} := x(s_i \cdot P)$,
  - → $t_{i+1} := s_{i+1} \oplus H(\text{additional input}_{i+1})$
  - → $s_{i+2} := x(t_{i+1} \cdot P)$
  - → $r_{i+2} := x(s_{i+2} \cdot Q)$
- attack:
  - reconstruct $s_{i+1} := x(d \cdot R_i)$
  - compute t_{i+1} and s_{i+2} for guessed additional input, then check against r_{i+2} (the test works also if r_{i+2} is used as an exponent for DH and only the result of exponentiation is visible for the attacker

Practical attack issues:

- some products do not use entire r_i and skip some number of bits. Frequently this is 16 bits – which makes the attack 2^{16} times longer. Truncating say 100 bits would secure the design, but this is not done
- some protocols disclose the original PRNG output. Then increasing the size of such a block eases the attack, as some steps are executed without additional input and the time complexity goes down
- Extended Random proposals: not standardized but finally BSafe uses as an option. Some TLS servers ask for Extended Random.

IV. STANDARS VERSUS SECURITY

It is not true that a standard solution is by definition a secure solution.

Standardization process:

- representatives of countries, not necessarily specialists

- strong representation of interests of industry
- target: a unified solution
- no open evaluation as in case of e.g. NIST competitions
- long process, many standards never used in practice

Example: ANSI X9.31 PRG

- approved PRNG by FIPS and NIST between 1992 and 2016
- now deprecated by NIST
- many devices based on X9.31 have FIPS certificates, widely used

Algorithm

→ **seeding**: select initial seed $s = (K, V)$, with random V and pre-generated key K

- K used for the lifetime of the device
- V changes

→ **next:**

1. input the current state $s_{i-1} = (K, V_{i-1})$ and timestamp T_i
2. intermediate value: $I_i := \mathrm{Enc}_K(T_i)$
3. output: $R_i := \mathrm{Enc}_K(I_i \oplus V_{i-1})$
4. state update: $V_i := \mathrm{Enc}_K(R_i \oplus I_i)$

Problems with seeding:

- NIST standard says: "This K is reserved only for the generation of pseudo-random numbers", and explains length,
- NIST standard does not say how K is generated
- consequences:

  → certification documentation may skip the problem of generating K

  → in some cases the key is encoded in software or hardware and **the same** for all devices

  and there is no reason to reject application for a certificate

- an attack is based on key recovered from software

  1. observe R_i and R_{i+1}
  2. guess timestamp T_i, T_{i+1} and check:

$$\mathrm{Dec}_K(\mathrm{Dec}_K(R_{i+1}) \oplus \mathrm{Enc}_K(T_{i+1})) = R_i \oplus \mathrm{Enc}_K(T_i)$$

indeed, this is equivalent to

$$\mathrm{Dec}_K(R_{i+1}) \oplus \mathrm{Enc}_K(T_{i+1}) = \mathrm{Enc}_K(R_i \oplus \mathrm{Enc}_K(T_i))$$

$$(I_{i+1} \oplus V_i) \oplus I_{i+1} = \mathrm{Enc}_K(R_i \oplus I_i)$$

$$V_i = V_i$$

3. if the test shows equality, then the timestamps are ok and both sides show V_i
4. having K and V_i one can recover states forwards and backwards each time adjusting the guesses for timestamp – as long as the (portions) of the generated sequence are available. For backwards:

   → $R_t = \mathrm{Enc}_K(I_t \oplus V_{t-1})$, so $V_{t-1} = \mathrm{Dec}_K(R_t) \oplus I_t$

   → having V_{t-1} compute $R_{t-1} = \mathrm{Dec}_K(V_{t-1}) \oplus I_{t-1}$

- the attack requires the key K and guessing two consecutive timestamps

  → implementations do not care about it and use consecutive outputs e.g. for DH exponent, separating them would help

  → presenting two output blocks of PRNG is necessary – so presenting at most one block would help

  → it would help to use DH exponent as a hash of the output of PRNG and some data hard to guess by the attacker, but many protocols do not do it

  → attacking either side may help for DH, but for RSA key transport the party choosing the secret must be affected

Example: Bleichenbacher's RSA signature forgery based on implementation error

- background: one has to pad the hash value before applying RSA signing operation, after padding the input it is no more a small number
- PKCS#1: RSA Laboratories standard for formats of RSA signatures (currently 1.5 and OEAP (optimal Asymmetric encryption padding)
- The attack works for PKCS-1.5 padding:
  - a byte 0
  - a byte 1
  - string of 0xFF bytes (their number depends on RSA modulus and the rest)

- a byte 0
- some ASN.1 data
- hash value of the signed document

  00 01 FF FF FF ... FF 00 ASN.1 HASH

- RSA signature verification: exponentiation with the public key, remove padding, check the hash
- implementation based on the standard: recognize the structure 00 01 FF FF FF ... FF 00 and after them parse the hash
- attack mechanism:
  - the hash is not right adjusted (the padding is too short), after the hash there is a part that is not parsed (it could be anything)
  - concern RSA systems with public key 3 (sometimes it is done so to speed-up verification) – according to strong RSA assumption computing roots of degree 3 is generally infeasible without the secret key
  - part after the hash adjusted so that the resulting number is a cube as an integer
  - compute the third root ... and this would be accepted as a valid signature for software not checking adjustment of the hash!
- attack variants: some fields after padding are declared but not checked. So adjust these parts so that the number becomes a cube (in this case the hash might be right justified)

Chosen Ciphertext Attacks Against Protocols Based on RSA Encryption Standard PKCS-1 – the Million Message Attack.

- RSA decryption device, returns an error message if the ciphertext is not in the PKCS-1 format (HSM,...) - for the attack it is enough, we do not need to see the result of decryption (if the ciphertext is correct)
- the standard format for encryption:

  00||02||PS||00||data, where PS is a string of nonzero bytes. The length of PS such that the resulting number has size proper for RSA
- the ciphertext c to be broken is manipulated many times and based on error messages we narrow the set of choices for the plaintext
- attack (find m such that $m^d = c \bmod n$):
  1. phase: blind the ciphertext: $c_0 := c \cdot s^e \bmod n$ by choosing s such that c_0 is a valid PKCS-1 ciphertext.
     - determining that c_0 is a valid ciphertext is done with the oracle device
     - observe $c_0^e = m \cdot s$ and it starts with msb: 00, 02,
     - let k be the byte length of n, $B = 2^{8(k-2)}$
     - then $2B \leq m \cdot s < 3B$ and it suffices to learn $m \cdot s$

– let $M_0 = [2B, 3B-1]$ – we know in advance that any plaintext falls into this interval because of PKCS-1 formatting

2. phase: narrowing the set of intervals defining $s_1 < s_2 < ...$ and $M_1, M_2, ...$ such that $M_{i+1} \subset M_i$, each M_i is a set of intervals

   Step $i > 1$:

   - if M_{i-1} is not a single interval, then find the smallest $s_i > s_{i-1}$ such that $c_0 \cdot s_i^e$ is PKCS-1 conforming
   - if M_{i-1} is a single interval $[a, b]$ then choose small values s_i and r_i such that

     $r_i > 2\frac{b \cdot s_{i-1} - 2B}{n}$ and $\frac{2B + r_i n}{b} < s_i < \frac{3B + r_i n}{a}$

     and $c_0 \cdot s_i^e$ is PKCS-1 conforming

   narrowing the choice (computing the set of intervals M_i):

   - M_i consists of all intervals

     $[\max\left(a, \frac{2B + r \cdot n}{s_i}\right), \min\left(b, \frac{3B - 1 + r \cdot n}{s_i}\right)]$ for (a, b, r) such that

     $[a, b]$ is one of the intervals from M_{i-1} and

     $\frac{a \cdot s_i - 3B + 1}{n} \leq r \leq \frac{b \cdot s_i - 2B}{n}$

   when eventually $M_i = [a, a]$ then the plaintext is a and the sought plaintext of c is $a \cdot s_0^{-1} \bmod n$

why it works:

- assume $m \in [a, b]$ from M_i
- as $m \cdot s_i$ is PKCS conforming, there is r such that

$$2B \leq m \cdot s_i - r \cdot n < 3B$$

hence

$$-2B > -m \cdot s_i + r \cdot n \geq -3B$$

$$m \cdot s_i - 2B > r \cdot n \geq m \cdot s_i - 3B$$

$$b \cdot s_i - 2B > r \cdot n \geq a \cdot s_i - 3B$$

- on the other hand

$$\frac{2B + r \cdot n}{s_i} \leq m < \frac{3B + r \cdot n}{s_i}$$

The attack exploits information leakages:

- error messages returned by the attacked device when decryption fails on different stages of the decryption algorithm, or
- different timings of execution of the decryption algorithm when the PKCS-1 encryption padding is correct and when it is incorrect.

If a device supports the PKCS-1 encryption padding and the implementation of the PKCS-1 decryption on the device is vulnerable, then the million message attack works also when

- the ciphertext is calculated according to a padding different than PKCS-1
- the "ciphertext" is the plaintext for which we want to obtain a signature (dangerous for a situation when the same key is used for decryption and for signatures, and decryption is not PIN protected).

V. LEGAL FRAMEWORK - EXAMPLE: **EIDAS REGULATION**

goals:

- interoperability, comparable levels of trust
- merging national systems into pan-European one
- trust services, in particular: identification, authentication, signature, electronic seal, time-stamping, electronic delivery, Web authentication
- supervision system
- information about security breaches
- focused on public administration systems. However, the rules for all trust services except for closed systems (not available to anyone). Private sector encouraged to reuse the same means.

tools:

- common legal framework
- supervision system
- obligatory exchange of information about security problems
- common understanding of assurance levels

technical concept:

- each Member State provides an online system enabling identification and authentication with means from this Member State to be used abroad
- a notification scheme for national systems
- if notified (some formal and technical conditions must be fulfilled), then every member state must implement it in own country within 12 month

identification and authentication:

- eID cards – Member States are free to introduce any solution, the Regulation attempts to change it and build a common framework from a variety of (incompatible) solutions
- breakthrough claimed, but likely to fail