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):
    1. identify policies applicable to the request
    2. the request evaluated against each rule from these policies
    3. the results are combined within a policy
    4. 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:
    1. creating policies is a complicated task, errors are likely
    2. 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 (webservers...)

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 ⇒ 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
  $$\mathrm{EM}(K,S) = \mathrm{Enc}(\mathrm{KB_{Enc}}, S) \| \mathrm{MAC}(\mathrm{KB_{Mac}}, \mathrm{Enc}(\mathrm{KB_{Enc}}, S), S)$$
  where the key K is $(\mathrm{KB_{Enc}}, \mathrm{KB_{Mac}})$
- 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

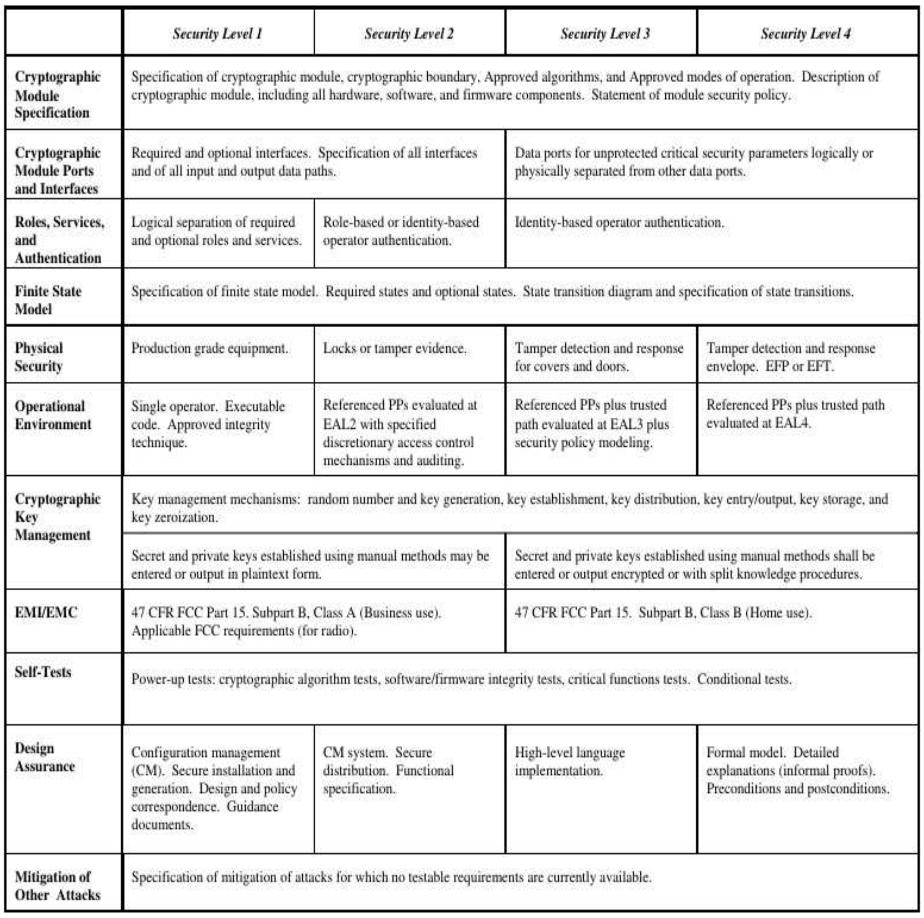
- The SARs are a description of how the TOE is to be evaluated. This description uses a standardised language (to provide exact description, to allow comparison between two PP).

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:

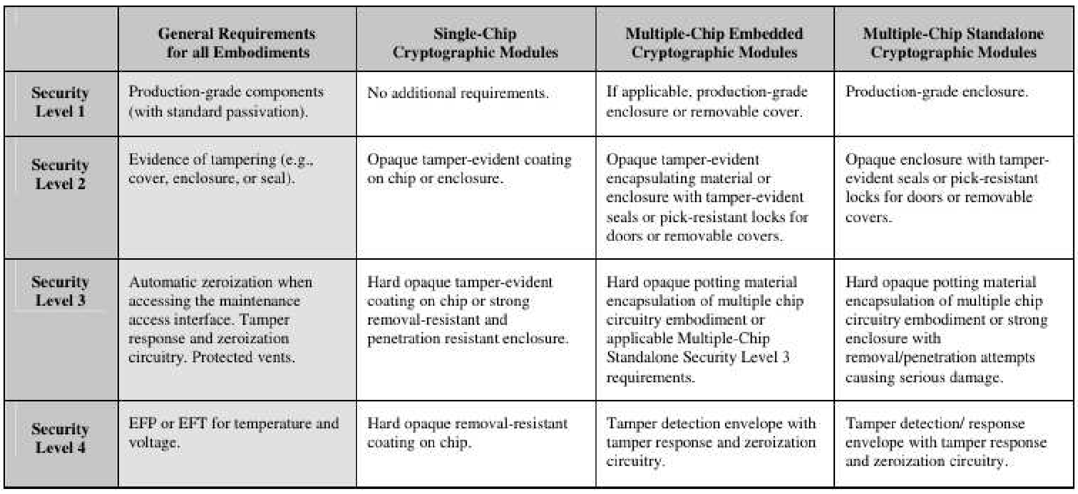


	Security Level 1	*Security Level 2*	*Security Level 3*	*Security Level 4*
Cryptographic Module Specification	Specification of cryptographic module, cryptographic boundary, Approved algorithms, and Approved modes of operation. Description of cryptographic module, including all hardware, software, and 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

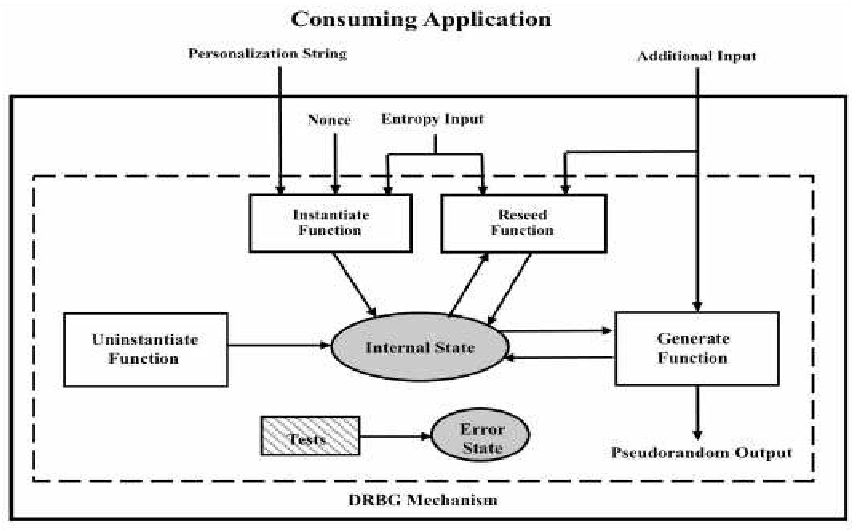
- key management:
    - non-approved RNG can be used for IV or as input to approved RNG
    - list of approved RNG: refers to an annex and annex to NIST document from 2016 (with a link to 2015)
    - list of approved key establishment methods - again links
    - key in/out: automated (encrypted) or manual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

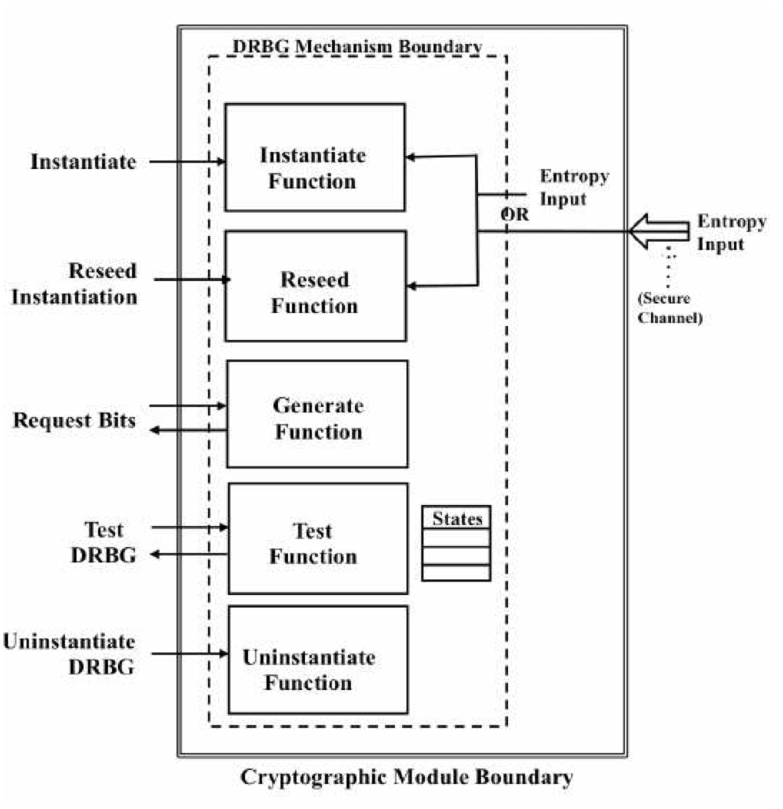
Functions executed:

- 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: $h_1 = \mathrm{Hash}(V)$,$h_2 = \mathrm{Hash}(h_1)$, $h_3 = \mathrm{Hash}(h_2)$, ...

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 = $\frac{requested - no - of - bits}{outlen}$
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 = $0^{keylen}$
4. V = $0^{blocklen}$
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$\neq$ 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

→ **initialization - 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

→ **generate (generating bits and changing the internal state):**

  1. input the current state $s_{i-1}=(K,V_{i-1})$ and the current 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 $s_1 < s_2 < \ldots$ and $M_1, M_2, \ldots$ 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

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

Defense strategies for PRNG

- two sources of randomness: PRNG and a deterministic generator fully configured by the user. Example for DH

  - the device's PRNG creates a and g^a

  - g^a is sent to the users's unit,

  - the user's unit holds a key K and it computes $a' := \mathrm{HMAC}(K, g^a)$

  - the final result is $(g^a)^{a'}$

  neither the PRNG nor the user's unit are of control of the randomness,

  they can self control themselves

- two PRNG with seeds set by different parties. If the user retains his seed then can check that there are no irregularities

- hashing the output of PRNG is a good idea according to the KEA1 assumption, correlated input hash assumption, this disables resuse

---

V. RFC DOCUMENTS

RFC ="Request for Comments"

- by Internet Engineering Task Force (IETF) and the Internet Society

- semi-standard, developed from rfc from ARPANET
- based on voluntary work, RFC created by small groups of authors, the authors are frequently independent
- peer review, some RFC's reach the status of an "Internet Standards"
- RFC editor controls the publication process
- streams:
  - Internet Engineering Task Force (IETF) - current issues
    - BCP Best Current Practice;
    - FYI For Your Information; informational
    - STD Standard: with 2 maturity levels (Proposed Standard, Internet Standard)
  - Internet Research Task Force (IRTF) - more long term issues
  - Internet Architecture Board (IAB) (a body over task forces)
  - independent authors
- Status:
  - FYI: informational
  - experimental
  - BCP: best current practice
  - STD standard: Draft Standard, Proposed Standard, Internet Standard

RFC as an example of specification of a protocol

A. Exemplary RFC: draft of TLS 1.3 spec.

B. Required level of detail - ensuring unambiguous implementation.

C. Structure of the document: from general view to detailed description - to facilitate reading many datastructures and technicalities are shifted to the appendices.

- Abstract: What the document is about?
- Status: Internet draft, expiration date
- Copyright notice
- Table of contents:
  1. Introduction
     - a note for RFC editors (present in the draft only)

- the goal of the protocol (authentication, confidentiality, integrity + definitions of the three terms, to let the non-cryptographers understand the document)
- the high level view - primary components
- conventions and terminology - for clear and precise understanding
- change log - for editors (present in the draft only)
- major differences from previous version of the protocol (TLS 1.2)

2. Protocol Overview. Handshake: what must be negotiated, what are the basic key exchange modes?

3. Presentation language: Big or little endian? basic block size? etc.

4.-9. Protocol components, further details.

10. Security considerations

D. Exemplary protocol detail: certificate request from server side (dnames of CAs) cross-certification.

EXAMPLE: RFC2560

Network Working Group

Category: Standards Track

authors ... June 1999

title: *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

This document specifies a protocol useful in determining the current status of a digital certificate without requiring CRLs. Additional mechanisms addressing PKIX operational requirements are specified in separate documents.

... contents of sections

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document (in uppercase, as shown) are to be interpreted as described in [RFC2119].

–

MUST=REQUIRED=SHALL: an absolute requirement

MUST NOT=SHALL NOT: an absolute prohibition of the specification

SHOULD=RECOMMENDED: ‘*‘there may exist valid reasons in particular circumstances to ignore, but implications must be understood and carefully weighed before choosing a different course*”

SHOULD NOT=NOT RECOMMENDED: negation of SHOULD (think twice before implementing it in this way!)

MAY=OPTIONAL: real option, **but** implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option,

2. Protocol Overview

– supplement to periodical checking CRL

– enables to determine the state of an identified certificate

– more timely, with more information

– RFC defines data exchanged

2.1 Request

– protocol version – service request – target certificate identifier – optional extensions which MAY be processed by the OCSP Responder

OCSP Responder checks:

1. request well formed
2. responder configured to serve such request
3. all necessary data given in the request

otherwise: error message

2.2 Response

– type+actual response

– basic type MUST be supported

– *"All definitive response messages SHALL be digitally signed."*

– signer MUST be one of: CA who issued the certificate, or a Trusted Responder of the requester, CA Designated Responder (Authorized Responder) - agent of CA with a certificate from CA

– response message: version of the response syntax – name of the responder – responses for each of the certificates in a request – optional extensions – signature algorithm OID – signature computed across hash of the response

– for each target certificate: certificate status value – response validity interval – optional extensions

- values:
  - good: *"At a minimum, this positive response indicates that the certificate is not revoked, but does not necessarily mean that the certificate was ever issued or that the time at which the response was produced is within the certificate's validity interval. Response extensions may be used to convey additional information on assertions made by the responder regarding the status of the certificate such as positive statement about issuance, validity, etc."*
  - revoked: the certificate has been revoked (permanantly or temporarily (on hold))
  - unknown: responder has no data

2.3 Exception Cases

- error messages not signed
- types: – malformedRequest – internalError – tryLater – sigRequired – unauthorized
- "internalError" = responder reached an inconsistent internal state. The query should be retried
- "tryLater" = temporarily unable to respond
- "sigRequired"= the server requires the client sign
- "unauthorized"=the client is not authorized to make this query

2.4 Semantics of thisUpdate, nextUpdate and producedAt

- thisUpdate = time at which the indicated status is known to be correct
- nextUpdate= time at or before which newer information will be available about the certificate status
- producedAt = time at which the OCSP signed this response.

2.5 Response Pre-production

''*OCSP responders MAY pre-produce signed responses specifying the status of certificates at a specified time. The time at which the status was known to be correct SHALL be reflected in the thisUpdate field of the response. The time at or before which newer information will be available is reflected in the nextUpdate field, while the time at which the response was produced will appear in the producedAt field of the response.*"

- means that OCSP is not checking the status of the certificate but status on the CRL!

2.6 OCSP Signature Authority Delegation

- the OCSP might be an agent of CA explicitely apointed,
- signing key must allow signing it

2.7 CA Key Compromise

- if CA's private key compromised, then OCSP MAY return the revoked state for all certificates issued by that CA.

3. Functional Requirements

3.1 Certificate Content

- CAs SHALL provide the capability to include the AuthorityInfoAccess extension in certificates that can be checked using OCSP
- accessLocation for the OCSP provider may be configured locally at the OCSP client
- CAs supporting OCSP MUST "*provide for the inclusion of a value for a uniformResourceIndicator (URI) accessLocation and the OID value id-ad-ocsp for the accessMethod in the AccessDescription SEQUENCE*"
- accessLocation field in the subject certificate defines the transport (e.g. HTTP) used to access OCSP responder and data (e.g. a URL)

3.2 Signed Response Acceptance Requirements

Before accepting response clients SHALL confirm that:

1. certificate in response=certificate asked
2. signature valid
3. signature of the responder
4. responder authorized
5. thisUpdate sufficiently recent
6. nextUpdate is greater than the current time

4. Detailed Protocol

- data to be signed encoded using ASN.1 distinguished encoding rules (DER)
- ASN.1 EXPLICIT tagging as a default
- "terms imported from elsewhere are: Extensions, CertificateSerialNumber, SubjectPublicKeyInfo, Name, AlgorithmIdentifier, CRLReason"

4.1 Requests

4.1.1 Request Syntax

OCSPRequest ::= SEQUENCE { tbsRequest TBSRequest, optionalSignature [0] EXPLICIT Signature OPTIONAL }

TBSRequest ::= SEQUENCE {version [0] EXPLICIT Version DEFAULT v1, requestorName [1] EXPLICIT GeneralName OPTIONAL, requestList SEQUENCE OF Request, requestExtensions [2] EXPLICIT Extensions OPTIONAL }

Signature ::= SEQUENCE { signatureAlgorithm AlgorithmIdentifier, signature BIT STRING, certs [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL}

Version ::= INTEGER { v1(0) }

Request ::= SEQUENCE { reqCert CertID, singleRequestExtensions [0] EXPLICIT Extensions OPTIONAL }

CertID ::= SEQUENCE { hashAlgorithm AlgorithmIdentifier, issuerNameHash OCTET STRING, – Hash of Issuer's DN issuerKeyHash OCTET STRING, – Hash of Issuers public key serialNumber CertificateSerialNumber }

- public key hashed together with name (names may repeat, public key must not)
- Support for any specific extension is OPTIONAL
- ”Unrecognized extensions MUST be ignored (unless they have the critical flag set and are not understood)”.
- requestor MAY sign the OCSP request, data included for easy verification (name:SHALL, certificate: MAY)

4.2 Response Syntax

OCSPResponse ::= SEQUENCE { responseStatus OCSPResponseStatus, responseBytes [0] EXPLICIT ResponseBytes OPTIONAL }

OCSPResponseStatus ::= ENUMERATED { successful (0), –Response has valid confirmations malformedRequest (1), –Illegal confirmation request internalError (2), –Internal error in issuer tryLater (3), –Try again later –(4) is not used sigRequired (5), –Must sign the request unauthorized (6) –Request unauthorized }

The value for responseBytes consists of an OBJECT IDENTIFIER and a response syntax identified by that OID encoded as an OCTET STRING.

ResponseBytes ::= SEQUENCE { responseType OBJECT IDENTIFIER, response OCTET STRING }

For a basic OCSP responder, responseType will be id-pkix-ocsp-basic.

id-pkix-ocsp OBJECT IDENTIFIER ::= { id-ad-ocsp } id-pkix-ocsp-basic OBJECT IDENTIFIER ::= { id-pkix-ocsp 1 }

4.3 Mandatory and Optional Cryptographic Algorithms

- clients SHALL: DSA sig-alg-oid specified in section 7.2.2 of [RFC2459]
- clients SHOULD: RSA signatures as specified in section 7.2.1 of [RFC2459]
- responders SHALL: SHA1

4.4 Extensions

4.4.1 Nonce

nonce against replay:

- nonce as one of the requestExtensions in requests
- in responses it would be included as one of the responseExtensions
- object identifier id-pkix-ocsp-nonce

4.4.2 CRL References

if revoked then indicate CRL where revoked

id-pkix-ocsp-crl OBJECT IDENTIFIER ::= { id-pkix-ocsp 3 }

CrlID ::= SEQUENCE { crlUrl [0] EXPLICIT IA5String OPTIONAL, crlNum [1] EXPLICIT INTEGER OPTIONAL, crlTime [2] EXPLICIT GeneralizedTime OPTIONAL }

For the choice crlUrl, the IA5String will specify the URL at which the CRL is available. For crlNum, the INTEGER will specify the value of the CRL number extension of the relevant CRL. For crlTime, the GeneralizedTime will indicate the time at which the relevant CRL was issued.

4.4.3 Acceptable Response Types

d-pkix-ocsp-response OBJECT IDENTIFIER ::= { id-pkix-ocsp 4 }

AcceptableResponses ::= SEQUENCE OF OBJECT IDENTIFIER

4.4.4 Archive Cutoff

– specifies how many years after expiration the revocation inforamation is retained, this si"archive cutoff" date

4.4.5 CRL Entry Extensions

All the extensions specified as CRL Entry Extensions - in Section 5.3 of [RFC2459] - are also supported as singleExtensions.

4.4.6 Service Locator

OCSP server receives a request and reroutes it to another OCSP

serviceLocator request extension used

d-pkix-ocsp-service-locator OBJECT IDENTIFIER ::= { id-pkix-ocsp 7 }

ServiceLocator ::= SEQUENCE { issuer Name, locator AuthorityInfoAccessSyntax OPTIONAL }

Values defined in certificate asked

5. Security Considerations

– flood of queries,

– signed and unsigned both enable DOS

– precomputation helps

– HTTP caching might be risky: “*Implementors are advised to take the reliability of HTTP cache mechanisms into account when deploying OCSP over HTTP.*’

VI. 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

changes regarding electronic signature:

- electronic seal with the same conditions as electornic signature,
- the seal is aimed for legal persons
- weakening conditions for qualified electronic signatures: admitting server signatures and delegating usage of private keys

new:

- electronic registered delivery service
- Webpage authentication

Example of requirements (electronic seal):

Definition:

“electronic seal creation device” means configured software or hardware used to create an electronic seal;

“qualified electronic seal creation device” means an electronic seal creation device that meets mutatis mutandis the requirements laid down in Annex II;

Art. 36

An advanced electronic seal shall meet the following requirements:

(a) it is uniquely linked to the creator of the seal;

(b)it is capable of identifying the creator of the seal;

(c)it is created using electronic seal creation data that the creator of the seal can, with a high level of confidence under its control, use for electronic seal creation; and

(d) it is linked to the data to which it relates in such a way that any subsequent change in the data is detectable.

Annex II:

(a) the confidentiality of the electronic signature creation data used for electronic signature creation is reasonably assured;

(b) the electronic signature creation data used for electronic signature creation can practically occur only once;

(c) the electronic signature creation data used for electronic signature creation cannot, with reasonable assurance, be derived and the electronic signature is reliably protected against forgery using currently available technology;

(d) the electronic signature creation data used for electronic signature creation can be reliably protected by the legitimate signatory against use by others.

2. Qualified electronic signature creation devices shall not alter the data to be signed or prevent such data from being presented to the signatory prior to signing.

3. Generating or managing electronic signature creation data on behalf of the signatory may only be done by a qualified trust service provider.

4. Without prejudice to point (d) of point 1, qualified trust service providers managing electronic signature creation data on behalf of the signatory may duplicate the electronic signature creation data only for back-up purposes provided the following requirements are met:

(a) the security of the duplicated datasets must be at the same level as for the original datasets;

(b) the number of duplicated datasets shall not exceed the minimum needed to ensure continuity of the service.

Art. 30

1. Conformity of qualified electronic signature creation devices with the requirements laid down in Annex II shall be certified by appropriate public or private bodies designated by Member States.

notification system:

An electronic identification scheme eligible for notification if:

(a) issued by the notifying state

(b) at least one service available in this state;

(c) at least assurance level low;

(d) ensured that the person identification data is given to the right person

(e) ...

(f) availability of authentication online, for interaction with foreign systems (free of charge for public services), no specific disproportionate technical requirements

(g) description of that scheme published 6 months in advance

(h) meets the requirements from the implementing act

Assurance levels:

- regulation, Sept. 2015, implementation of eIDAS
- reliability and quality of
  - enrolment
  - *electronic identification means* management
  - authentication
  - management and organization
- authentication factors
  - posession based
  - knowledge based
  - inherent (physical properties)
- enrolment: (for all levels):

  1. Ensure the applicant is aware of the terms and conditions related to the use of the electronic identification means.

  2. Ensure the applicant is aware of recommended security precautions related to the electronic identification means.

  3. Collect the relevant identity data required for identity proofing and verification.

- identity proving and verification (for natural persons):

  low:

  1. The person can be assumed to be in possession of evidence recognised by the Member State in which the application for the electronic identity means is being made and representing the claimed identity.

  2. The evidence can be assumed to be genuine, or to exist according to an authoritative source and the evidence appears to be valid.

  3. It is known by an authoritative source that the claimed identity exists and it may be assumed that the person claiming the identity is one and the same.

  substantial: low plus:

  1. The person has been verified to be in possession of evidence recognised by the Member State in which the application for the electronic identity means is being made and reprensenting the claimed identity

and

the evidence is checked to determine that it is genuine; or, according to an authoritative source, it is known to exist and relates to a real person

and

steps have been taken to minimise the risk that the person's identity is not the claimed identity, taking into account for instance the risk of lost, stolen, suspended, revoked or expired evidence; or

2. options related to other trustful sources

high: substantial plus

(a) Where the person has been verified to be in possession of photo or biometric identification evidence recognised by the Member State in which the application for the electronic identity means is being made and that evidence represents the claimed identity, the evidence is checked to determine that it is valid according to an authoritative source; and the applicant is identified as the claimed identity through comparison of one or more physical characteristic of the person with an authoritative source; or ...

- electronic identification means management:

  low:

  1. The electronic identification means utilises at least one authentication factor.

  2. The electronic identification means is designed so that the issuer takes reasonable steps to check that it is used only under the control or possession of the person to whom it belongs.

  substantial:

  1. The electronic identification means utilises at least two authentication factors from differŋ ent categories.

  2. The electronic identification means is designed so that it can be assumed to be used only if under the control or possession of the person to whom it belongs.

  high:

  1. The electronic identification means protects against duplication and tampering as well as against attackers with high attack potential

  2. The electronic identification means is designed so that it can be reliably protected by the person to whom it belongs against use by others.

- Issuance , delivery and activation:

  low:

  After issuance, the electronic identification means is delivered via a mechanism by which it can be assumed to reach only the intended person.

  substantial:

  After issuance, the electronic identification means is delivered via a mechanism by which it can be assumed that it is delivered only into the possession of the person to whom it belongs.

  high:

  The activation process verifies that the electronic identification means was delivered only into the possession of the person to whom it belongs.

- suspencion, revocation and reactivation:

all levels:

1. It is possible to suspend and/or revoke an electronic identification means in a timely and effective manner.

2. The existence of measures taken to prevent unauthorised suspension, revocation and/or reactivation.

3. Reactivation shall take place only if the same assurance requirements as established before the suspension or revocation continue to be met.

- authentication mechanism:

  substantial:

  1. The release of person identification data is preceded by reliable verification of the electronic identification means and its validity.

  2. Where person identification data is stored as part of the authentication mechanism, that information is secured in order to protect against loss and against compromise, including analysis offline.

  3. The authentication mechanism implements security controls for the verification of the electronic identification means, so that it is highly unlikely that activities such as guessing, eavesdropping, replay or manipulation of communication by an attacker with enhanced-**basic attack potential** can subvert the authentication mechanisms.

  high:

  .... by an attacker with **high attack potentia**l can subvert the authentication mechanisms.

- audit:

  low:

  The existence of periodical internal audits scoped to include all parts relevant to the supply of the provided services to ensure compliance with relevant policy.

  substantial:

  The existence of periodical **independent** internal or external audits ....

  high:

  1. The existence of periodical i**ndependent external audits** scoped to include all parts relevant to the supply of the provided services to ensure compliance with relevant policy.

  2. Where a scheme is directly managed by a government body, it is audited in accordance with the national law.

VII. HARDWARE TROJANS

goal of a Trojan:

change hardware so that the chip functionally seems to work as claimed, but it opens doors for the attacker

attack moment:

– chip planning (easy)

- chip manufacturing (hard)
- hardware components from third parties (easy)
- outsourcing fabrication (likely to occur due to production line costs)

methods of testing:

- functional tests (not really helpful for trapdoors, the most dangerous are hidden faults that do not disrupt operation)
- internal tests circuitry (putting some values and observing results on single components along so called test path, or dedicated tests like checking CRC of memory contents)
- distructive - chemical-mechanical polishing and inspection under microscope etc, it can detect modifications on layout level, costly
- side channel information (especially comparing with a "golden chip" behavior – the chip that is ideal and follows the specification) - delay path analysis, static current analysis, transient current analysis

classical attacks: the trojans should remain undetected during the testing phase, so the attack has to be triggered by an unlikely event. Methods used:

- an attack triggered by an unlikely event known to the attacker but not to the evaluator
- there is a counter, the attack starts when the counter reaches a certain value
- attack occurs due to aging or via a random event (e.g. for enabling fault analysis)

some countermeasures:

- regions: design a chip so that it is covered by "regions"
  - for each region there must be a test path so that the activities are concentrated in this region while the rest stays almost idle,
  - then the side channel (such as energy usage) may be attributed to that region
  - a hardware Trojan should be concentrated in some region and then substantially change the side channel of that region
- removing rare-triggered nets
- configurable security monitors
- variant- based parallel execution of the same function

Analog attack: A2

goal: in a certain situation change a priviledge bit (the rest of the attack follows some scenario)

limitation:

- no change in digital circuit, only some analog parts added

- very limited regarding area (so playground for ASICs, which are less optimized – less compressed)
- trojans preferably in layer 1 to avoid collisions with routing etc

construction idea:

- a single capacitor added,
- the capacitor is loaded each time a triggering event occurs
- if triggering events occur in a short period of time, then the capacitor loaded to a certain voltage causing a flip-flop operation to occur (changing a bit to a predefined value)
- the capacitor discharged gradually so if triggering events occur infrequently, then the flip-flop operation never executed

a more robust construction:

- choosing relative capacity of capacitors one can control the number of triggering events needed

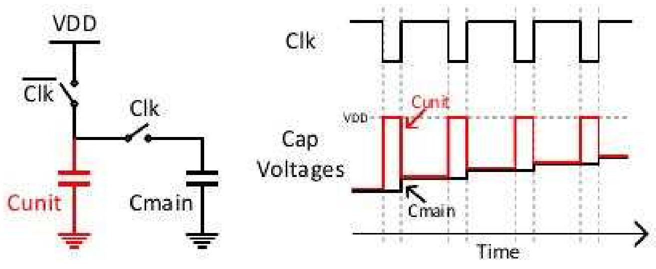


Figure 4: Design concepts of analog trigger circuit based on capacitor charge sharing.

Figure 1.

(from paper: A2: Analog Malicious Hardware, Kaiyuan Yang, Matthew Hicks, Qing Dong, Todd Austin, Dennis Sylvester)

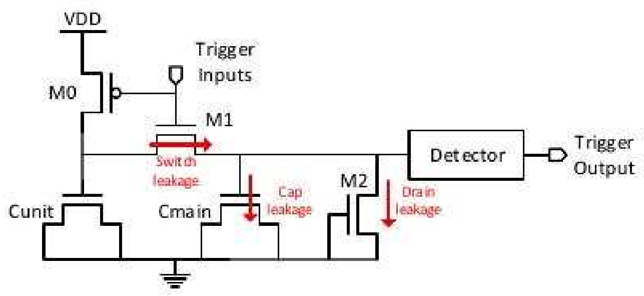


Figure 5: Transistor level schematic of analog trigger circuit.

Figure 2.

detector is for instance an inverter

extensions:

- use a few such analog circuits and combine them
- e.g.: both must “fire” (AND operation), one of them suffices (“OR”) – in theory any circuit possible however the attacker is limited by space available

Dopant Trojans

CMOS inverter: (image Wikipedia)

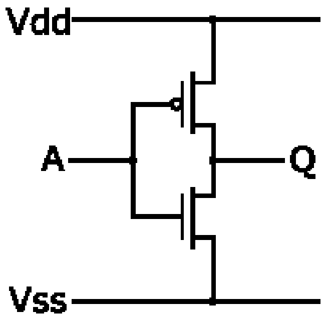


Figure 3.

where: A is the source, Vdd positive supply , Vss is ground

upper transistor: PMOS (allows current flow at low voltage)

lower transistor: NMOS (allows current flow at high voltage)

how it works:

- if voltage is low, then the lower transistor (NMOS) is in high resistance state and the current from Vdd flows to Q (high voltage)
- if voltage is high, then the upper transistor MOS) is in high resistance state and the current from Vss flows to Q while Vdd has low voltage

PMOS: in dopant area “holes” (positive) playing the role of conductor, low voltage creates depletion area, high voltage attracts them

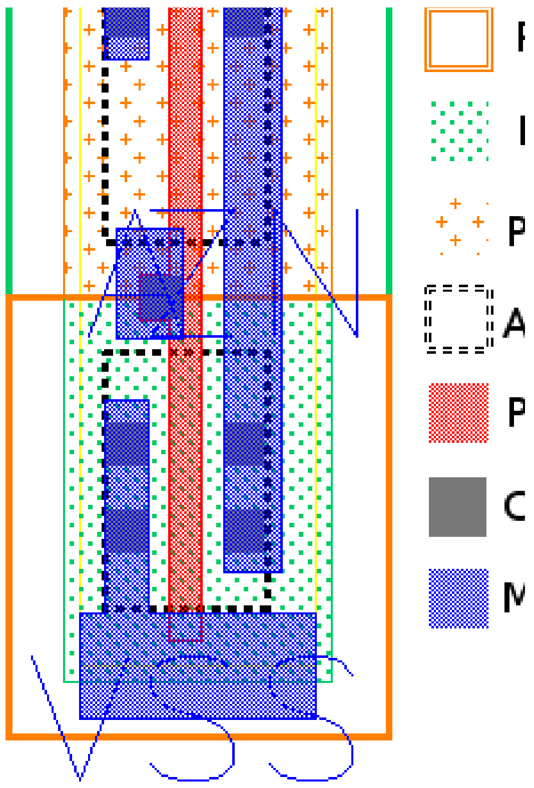
NMOS: in dopant area electrons (negative) playing the role of conductor, high voltage pushes the electrons out

For physical realization of a transistor see excellent videos from

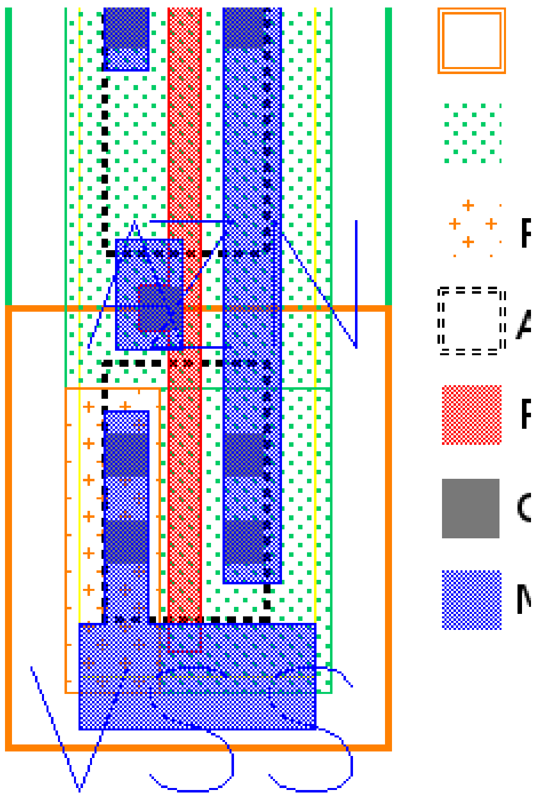
https://www.youtube.com/watch?v=7ukDKVHnac4&t=116s

https://www.youtube.com/watch?v=stM8dgcY1CA

CMOS inverter in the “bird eye perspective”:

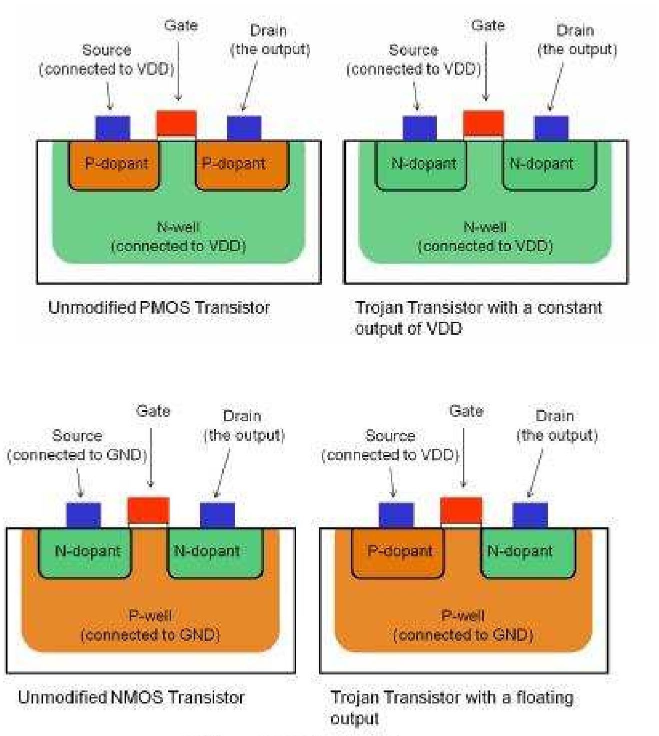


Trojan design:



The idea is to inject wrong dopant and thereby disable or enable connection regardless of the voltage

- whatever happens the VDD is connected to the output
- whatever happens the VSS is disconnected with the output


Gate
Source (connected to VDD)
Drain (the output)
P-dopant
P-dopant
N-well (connected to VDD)
Unmodified PMOS Transistor
Gate
Source (connected to VDD)
Drain (the output)
N-dopant
N-dopant
N-well (connected to VDD)
Trojan Transistor with a constant output of VDD
Gate
Source (connected to GND)
Drain (the output)
N-dopant
N-dopant
P-well (connected to GND)
Unmodified NMOS Transistor
Gate
Source (connected to VDD)
Drain (the output)
P-dopant
N-dopant
P-well (connected to GND)
Trojan Transistor with a floating output

(b) n-MOS Transistor

Trojan TRNG

TRNG consists of

- entropy source (physical)
- self test circuit (OHT - online health test)
- deterministic RNG, Intel version:
    - conditioner (computes seeds to rate matcher) and rate matcher (computes 128 bit numbers)
    - derivation, internal state (K, c):
        1. $c := c + 1$, $r := \mathrm{AES}_K(c)$
        2. $c := c + 1$, $x := \mathrm{AES}_K(c)$
        3. $c := c + 1$, $y := \mathrm{AES}_K(c)$
        4. $K := K \oplus x$
        5. $c := c \oplus y$

- conditioner (reseeding)

  1. $c := c+1, \quad x := \mathrm{AES}_K(c)$
  2. $c := c+1, \quad y := \mathrm{AES}_K(c)$
  3. $K := K \oplus x \oplus s$
  4. $c := c \oplus y \oplus t$

- attack: fix K by applying Trojan transistors, if K is known, then it is easy to find internal state c from r and then the consecutive random numbers r
- fix all but n bits of c then only n bits of entropy and the output r has only n entropy bits - to the attack does not need to see anything, just prediction possible (helpful e.g. against randomized signature schemes)
- problem with Built-In-Self-Test implemented according to FIPS: after power-up the RNG is tested against aging:
  - known LFSR creates bits strings for conditioner and rate matcher, entropy source disabled, a 32-bit CRC from the result computed and checked against a known value,
  - knowing the test one can find K and c by exaustive search

Side channel Trojan:

- side channel resistant logic: Masked Dual Rail Logic

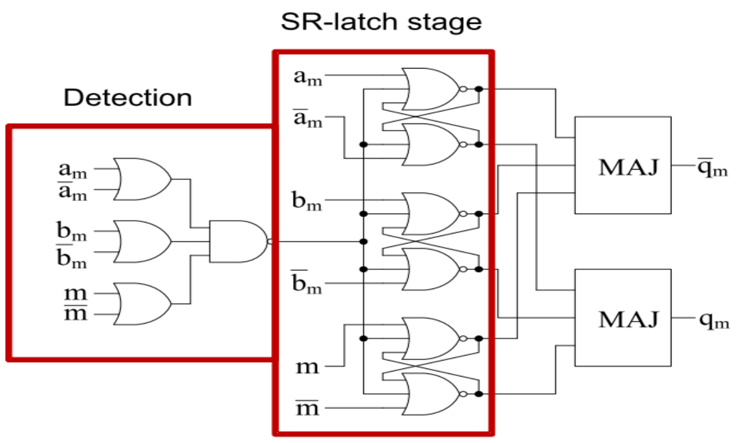
  i. for each a both a and negation of a computed

  ii. precharge: each phase preceded by charging all gates

  iii. masking operations by random numbers:

  computing $a \wedge b$:

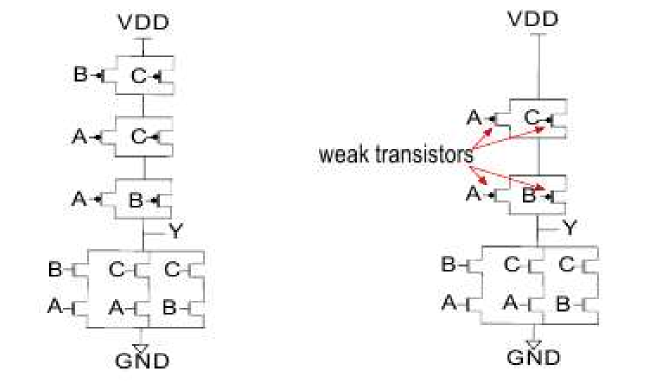
  - input $a \oplus m$, $a \oplus \neg m$, $b \oplus m$, $b \oplus \neg m$, m, $\neg m$
  - detection, SR-latch stage and majority gate



gates on the picture: OR – 3 gates in the detection , AND - the right gate in the Detection, NOR (output 1 if all inputs 0)- the OR gate with a dot

SR-latch is a bi-stable circuit. It remains stable in the state (0,1) and in (1,0). These values encode two bitvalues

attacking not-majority gate:



a) Trojan free AOI222 Gate b) Trojan AOI222 Gate

Idea: instead of cutting output there is low voltage in a certain situation

– the same behavior except for $A = 0$ and B, $C = 1$, where good output but high power consumption due to connection between VDD and VSS

– the upper pair of transistors do not disappear from the layout but are changed so that in fact constant connections are created.

– weakness of the transistors is created via reducing dopant areas (dopant creates free electrons or hole that may “jump”. Alone reducing the size of active area makes a transistor weak.

– computing majority works as normal except for the case that $a_m = 0, b_m = 1, m = 1$ or $\bar{a}_m = 0, \bar{b}_m = 1, \bar{m} = 1$. In both cases we have $a = 1, b = 0$

– in the last case the weak transistors are too weak to change the result (connection to the ground is stronger and determines the final voltage)

Artificial aging

make some transistors disfuctional (as ithe case of PRNG)

method:

– apply to high voltage at certain areas

– the electrons accelerate and break barrier - damages

– effect the same as of aging a chip

– the transistor changes its operational characteristic

Defense methods:

- problem: Trojan may be triggered by some particular event, detection becomes harder
- problem: Trojan may work in very particular physical conditions, e.g. temperature, voltage
- on-chip checks: detection of unexpected behavior, e.g. delay characteristics: workload path and a shadow path that provides result after fixed time, + comparison
- ring osscilators on the chip detecting nonstandard behavior
- methods to enable activation in certain areas only
- inserting PUFs, (either randomize as much a s possible - noise over trojan information)
- keep algorithm deterministic
- secure coding: take into account the situation that certain components are not working properly
- external watchdogs techniques

VIII. QUANTUM COMPUTING AND QUANTUM DEVICES

as there are problems to guarantee security of devices in the traditional way, maybe there is a way out using physics? three directions:

1) quantum based random number generators

2) key transport

3) quantum cryptanalysis

Qubit

the concept is as follows:

- instead of a bit with discrete states 0 and 1 we have a linear combination of basis vectors denoted by $|0\rangle$ and $|1\rangle$:

  $\alpha \cdot |0\rangle + \beta \cdot |1\rangle$
- with α, β complex numbers
- a measurement of $\alpha \cdot |0\rangle + \beta \cdot |1\rangle$ yields $|0\rangle$ with pbb $|\alpha|^2$ and $|1\rangle$ with pbb $|\beta|^2$ – this is quite annoying but ...
- measurement may be performed only for orthogonal basis. The basis can be different from $|0\rangle$ and $|1\rangle$. E.g.:

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \quad , \quad \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

- measuring $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ in basis $|0\rangle$ and $|1\rangle$ yields both 0 and 1 with perfect probability 0.5: it seems to be a perfect source of random bits:
  - generate fotons $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$
  - measure them in basis $|0\rangle$ and $|1\rangle$
- moreover: reading changes the state to the state read: if the result is $|0\rangle$ then the physical state becomes $|0\rangle$ as well. There is no state $\alpha \cdot |0\rangle + \beta \cdot |1\rangle$ anymore.
- **In fact, this is the core of Shor'a algorithm - a reading operation creates a change in a physical system that would be infeasible to compute on a classical computer**
- instead of a single bit we may have strings of qubits, say of length l where $l > n$

Random Number Generators

Problems:

- high price (e.g 990EUR +VAT)
- while physical source might be ok, reading circuit introduces high bias, very poor results (2017) in the standard randomness tests for devices available on the market
- bias can be removed via additional logic, but extra hardware may mean place for Trojans and the whole advantage is gone

Quantum key transport, BB84

- Charles Bennett and Gilles Brassard, 1984, 1st quantum protocol, even implemented
- key agreement immune to eavsdropping (reading qubits is detectable)
- two bases used: $|0\rangle$ and $|1\rangle$ (denoted $+$) or $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ (denoted $\times$)
- encoding of bits:

  basis $+$: $|0\rangle = 0$ and $|1\rangle = 1$

  basis $\times$: $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = 0, \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = 1$
- Steps:
  1. Alice chooses at random bitstrings a and b of length n
  2. for $i \leq n$ Alice encodes a_i according to basis indicated by b_i (0 indictes $+$, 1 indicates $\times$)
  3. Alice sends n photons (codes for a)
  4. Bob chooses at random string $\hat{b}$ of n bits,
  5. For the ith photon, Bob measures the photons using the basis indicated by $\hat{b_i}$
  6. Alice sends b to Bob over a traditional (public) channel, Bob sends $\hat{b}$ to Alice

7. Alice and Bob take the substring K of bits a_i such that $b_i = \hat{b_i}$

8. Alice chooses a subset of 50% of bit of K and discloses them to Bob

9. Bob checks how many of them disagree with his measurement. If above some threshold then it is likely that an adversary has measured the transmission as well and the protocol is aborted (environment may also create inconsistencies)

10. the unpublished substring of K may differ between Alice and Bob: an error correcting procedure applied (error correction attracts the bitstrings to the closest codewords, so if the strings of Alice and Bob differ slightly, then they result in the same codeword)

11. "privacy amplification": hashing to a much shorter string

– effect of eavesdropping:

  - say Alice chooses basis $\times$ to encode 0,
  - eavesdropper Eve chooses a different basis for the measurement: namely $+$
  - Eve gets $|0\rangle$ with pbb .5 and $|1\rangle$ with pbb .5, say $|0\rangle$ has been obtained
  - at the same time the photon converted to $|0\rangle$ (important!)
  - Bob measures the photon according to basis $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$
  - $|0\rangle = \frac{1}{\sqrt{2}}\Big(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) + \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\Big)$, so both results of the measurements (i.e $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ or $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ are equally probably for Bob, so the measurement of Bob indicated 1 with pbb .5
  - corollary: eavesdropping creates inconsistency between Alice and Bob with pbb .5 once Eve chooses a different basis than Alice and Bob

– quantum hacking: in theory wonderful, but the problems come with physical realization

  → sending many photons to Bob at the time when his hardware already set for a measurement. reflected photons show the basis used

Eckert algorithm:

– entangled pair of photons: measurement of one of them makes the mirror change of the other photon

– procedure:

1. generate entangled qubits $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ by some source

2. measure them by Alice and Bob on both ends of the channel

Properties:

– long distance transmissions possible, even to moving airplane

- over optical fibre or in free space (better vacuum)
- 1203 km between ground stations over satelite (China)
- both BB84 and Eckert used
- high price
- does not solve man-in-the-middle issue

Quantum computing and Shor factorization algorithm

Problem and its algebraic context:

- given an RSA number $n = p \cdot q$ for prime factors p and q of a similar size, the goal is to find p or q
- many modern crypto products are based on difficulty of this factorization problem. There are many software systems and embedded devices with RSA, no update is possible
- in order to break factorization problem it suffices to learn a nontrivial root r of 1:
    - $r \neq -1$
    - $r^2 = 1 \bmod n$

  indeed

    - $r^2 - 1 = (r-1)(r+1) = 0 \mod p \cdot q$
    - therefore p divides either $r-1$ or $r+1$
    - if p divides $r-1$ then q cannot divide $r-1$ as then $r-1$ would be at least n, but $r-1 < n$
    - in this situation we compute $\text{GCD}(n, r-1)$, the result must be p
    - if p divides $r+1$ then q cannot divide $r+1$ and therefore q must divide $r-1$. In this case $\text{GCD}(n, r-1)$ yields q
- if for a given $a < n$ we find s such that $a^s = 1$, then with probability ≥ 0.5 we get $a^{s/2}$ as a nontrivial root of 1. Indeed:
    - by Chinese Reminder Theorem a number $a < n$ is represented by $a_p = a \bmod p$ and $a_q = a \bmod q$
    - given a and b we may compute representation of $a \cdot b \bmod n$ by computing $a_p \cdot b_p \bmod p$ and $a_q \cdot b_q \bmod q$
    - there are two roots of 1 modulo prime number p: 1 and $p-1$
    - if $a^s = 1 \bmod n$, while $a^{s/2} \neq 1 \bmod n$, then $a^{s/2} \bmod p$ is 1 or -1

- there are the following cases:

  1. $a^{s/2} = 1 \bmod p$, $a^{s/2} = -1 \bmod q$
  2. $a^{s/2} = -1 \bmod p$, $a^{s/2} = 1 \bmod q$
  3. $a^{s/2} = -1 \bmod p$, $a^{s/2} = -1 \bmod q$

     the last case corresponds to $-1 \bmod n$, the first two ones to a nontrivial roots of -1

- so it suffices to find such an s - the order of a. By repeating the procedure for different a's we finally find a nontrivial root of $-1 \bmod n$

Quantum operations and gates

- a quantum computer should perform some operations on qubits, technical realization is a challenge, but in theory possible
- we consider $l -$ qubit numbers as representing numbers mod 2^l (well, this is fuzzy as each bit is fuzzy as a qubit), in this way we a get quantum state for each $a < q = 2^l$
- Hadamard transformation: an easy way to create a quantum state such that takes any value a (denoted $|a\rangle$) with the same probability. The way to achieve this is:
  - create the state $|0....0\rangle$
  - apply Hadamard transformation gate to it. That is, each ccordinate is transformed by

    $$\frac{1}{\sqrt{2}}\begin{pmatrix} 1,1 \\ 1,-1 \end{pmatrix}$$

    so $|0\rangle$ is transformed to

    $$\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

- Quantum Fourier transform:
  - regular FT: $(x_1, ..., x_N)$ transformed to $(y_1, ..., y_N)$ where
    $y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \cdot e^{(2\pi i \cdot j \cdot k)/N}$
  - quantum:
    $\sum x_i \cdot |i\rangle$ transformed to $\sum y_i \cdot |i\rangle$ where
    $y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \cdot e^{(2\pi i \cdot j \cdot k)/N}$
  - in other words:
    $|j\rangle \to \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{(2\pi i \cdot j \cdot k)/N} \cdot |k\rangle$

  an „efficient implementation” is based on similar algebra as for DFT

Shor's algorithm (based on presentation of Eric Moorhouse)

1. fix q such that $2n^2 < q < 3n^2$, $q = 2^l$ (or a product of small primes)

   we use states with $2l$ qubits, notation $|a, b\rangle$ or $|a\rangle|b\rangle$

2. prepare state $|0, 0\rangle$ and apply Hadamard transformation to the first register. Its result is a state
$$|\psi\rangle = \frac{1}{\sqrt{q}} \cdot \sum_{a=0}^{q-1} |a, 0\rangle$$

3. fix $x < n$ at random

4. to the state $|\psi\rangle$ apply the quantum transformation
$$|a, 0\rangle \to |a, x^a \bmod n\rangle$$
   the result is
$$\frac{1}{\sqrt{q}} \cdot \sum_{a=0}^{q-1} |a, x^a \bmod n\rangle$$
   (there is a theory how to make such a computation with quantum gates)

5. measure the second register. The result is some k. But then the measured state changes to
$$\frac{1}{\sqrt{M}} \cdot \sum_{a \in A} |a, k\rangle$$
   where A is the set of all a such that $x^a = k \bmod n$

   so $A = \{a_0, a_0 + r, a_0 + 2r....\}$ where r is the order of x and $M = |A|$ (so $M \approx q/r$)
$$\frac{1}{\sqrt{M}} \cdot \sum_{d=0}^{M-1} |a_0 + d \cdot r, k\rangle$$

6. apply the DFT to the first register. This changes the state
$$\frac{1}{\sqrt{M}} \cdot \sum_{d=0}^{M-1} |a_0 + d \cdot r, k\rangle$$
   to
$$\frac{1}{\sqrt{q \cdot M}} \cdot \sum_{c=0}^{q-1} \sum_{d=0}^{M-1} e^{2\pi i \cdot c(a_0 + d \cdot r)/q} \cdot |c, k\rangle$$
   which is equal to
$$\sum_{c=0}^{q-1} \frac{e^{2\pi i \cdot c \cdot a_0/q}}{\sqrt{q \cdot M}} \sum_{d=0}^{M-1} e^{2\pi i \cdot c \cdot d \cdot r/q} \cdot |c, k\rangle$$
$$\sum_{c=0}^{q-1} \frac{e^{2\pi i \cdot c \cdot a_0/q}}{\sqrt{q \cdot M}} \sum_{d=0}^{M-1} \zeta^d \cdot |c, k\rangle$$
   where
$$\zeta = e^{2\pi i \cdot c \cdot r/q}$$

7. measure the first register (this is the key moment!!)

   - which c is read depends on the values of $\sum_{d=0}^{M-1} \zeta^d$ which in turn corresponds to the probability
   - if $c \cdot r/q$ is not very close to an integer, then the sum is $\frac{1-\zeta^M}{1-\zeta}$
   - if $c \cdot r/q$ is an integer, then we sum up M ones
   - so the former case is unlikely and the readings are concentrated around values c such that

     $$c/q \approx d/r$$

     for an integer d
   - the rest is a classical computation involving c, q and trying different d's. The search space is relatively narrow

CATACRYPT catastrophy cryptography

- what happens if assumptions broken (e.g. DL solvable for some group)?
- "post-quantum crypto"

reality:

- post-quantum is at early stage, no industrial products, logistically impossible to replace
- no plans, scenarios, ...
- consequences of building a quantum computer or anything that breaks the current mechanisms:

  i. option 1: the situation is well hidden, the party controlling the means uses it without leaving traces but still claims the system is secure

  ii. option 2: disclosed, call for massive conversion to new products (may occur even if there is no quantum computer but it works for increasing sale numbers)

  iii. option 3: disclosed, neglected
- catastrophy is already there

IX. GDPR

Protection of personal data

- declared as fundamental right in EU, but technically fundamental for cybersecurity
  - identity theft e.g. for financial criminality
  - mobbing, discrimination, social abuse
  - lack of protection is a threat for economy and national security

Frameworks

- EU and European Economic Area, adopted by many countries, some recognized as equivalent by EU
- Privacy Shield: https://www.privacyshield.gov/
- California Consumer Privacy Act

SEE THE SET OF SLIDES for DISCUSSION on CONTENTS OF GDPR

TECHNICAL TOOLS for PRIVACY PROTECTION

Pseudonymization

Symmetric methods:

- hashing the identifier: pseudonym = Hash(identifier)

  problem: it is impossible to compute the identifier from the pseudonym, however hashing all possible identifiers and brute force reveals the link between the pseudonym and identifier
- encryption with a (secret) symmetric key: unlinkability, however the user cannot compute the pseudonym himself and the owner of the secret key can link all pseudonyms
- hashing with a key: as above, the party holding the secret key has to perform brute force to link back the pseudonym to the identifier

Asymmetric methods:

- based on Diffie Hellman Problem:
  - a domain (service provider, database, etc) holds a pair of keys $(d, D = g^d)$
  - a user Alice holds a pair $(x, X = g^x)$
  - the pseudonym of Alice corresponding to D is $g^{x \cdot d}$, which is computed as X^d by the domain manager, and D^x by the user
  - nobody but the user and the domain manager can compute the pseudonym, for a third person deciding whether X^d corresponds to X in domain D requires solving DDH Problem
- a variant based on domain and a central Authority:
  - the key d is not known to the domain authority
  - $d = d_A + d_{\text{domain}}$, where d_A is known by the authority and d_{domain} is known by the domain manager
  - steps of generating the pseudonym by the authority:
    1. the Authority computes $X' := X^{d_A}$ and presents X' to the domain manager
    2. the domain manager computes pseudonym as $(X')^{d_{\text{domain}}}$

    - linking a pseudonym with the starting public key is a reverse process but **both the domain manager and the Authority must participate** in it

- a variant from German personal identity cards (Restricted Identification):
    - the pseudonym is Hash(D^x)
    - the pseudonym is presented by the ID card over a secure channel, no proof that the pseudonym is correct, but a smart card can create only one pseudonym per domain
    - revocation by computing Hash($(X^{d_A})^{d_{\text{domain}}}$) jointly by the Authority and the domain manager and putting the result on the blacklist
    - blacklisting a black sheep based on the domain pseudonym requires brute force and recomputing all pseudonyms
- more flexibility if pairing groups are available but be careful: DDH might be easy and so the above methods do not work

Advantages and disadvantages:

- different pseudonyms generated automatically is
    - user friendly
    - makes re-identification based solely on data related to the pseudonym much harder
- problems:
    - converting a pseudonym in domain D_1 to a pseudonym in domain D_2 might be hard or infeasible, and require cooperation with the user and/or an authority

      (problem area: moving pseudonymized medical records)

DATABASES and handling queries

the main problem is answering queries: does a query result disclose personal data?

Approach: **anonymity set**

- query accepted if the number of record used to answer the query is at least k (and each concerns a different person)
- the method is naive: the attack is to ask for two sets of records: one including Alice and one excluding Alice to know the value for Alice

Approach: **differential privacy**

here we classify the algorithms (queries) and may say that an algorithm A satisfies ϵ-differential privacy. The following conditions must be met for any two databases D and D' that differ by elimination of one record:

- for any subset of the image of A the property from the next point must be fulfilled
- the property

$$\Pr(A(D) \in S) \leq \exp(\epsilon) \cdot \Pr(A(D') \in S)$$

  where the probability is over the random choices of A

Then:

- $\epsilon = 0$ is the ideal for privacy, but the result is a noise that does not depend on the database contents

- so it is necessary to find balance between privacy (ϵ as small as possible) and information in the response (ϵ as big as possible)

Pseudonymous Signature:

Application areas:

- while having the pseudonyms, how to authenticate digital data? Digital signatures would solve the problem

- implementing GDPR rights in practice: a data subject can authenticate the request (e.g. for data rectification) in a database with pseudonyms by sending a request with a signature corresponding to the pseudonym

BSI Pseudonymous Signature:

- keys:

  - domain parameters D_M and a pair of global keys $(\mathrm{PK}_M, \mathrm{SK}_M)$

  - public key $\mathrm{PK}_{\mathrm{ICC}}$ for a group of eIDAS tokens, the private key $\mathrm{SK}_{\mathrm{ICC}}$ known to the issuer of eIDAS tokens

  - assigning the private keys for a user: the issuer chooses $\mathrm{SK}_{\mathrm{ICC},2}$ at random, then computes $\mathrm{SK}_{\mathrm{ICC},1}$ such that $\mathrm{SK}_{\mathrm{ICC}} = \mathrm{SK}_{\mathrm{ICC},1} + \mathrm{SK}_M \cdot \mathrm{SK}_{\mathrm{ICC},2}$

  - a sector (domain) holds private key $\mathrm{SK}_{\mathrm{sector}}$ and public key $\mathrm{PK}_{\mathrm{sector}}$.

  - a sector has revocation private key $\mathrm{SK}_{\mathrm{revocation}}$ and public key $\mathrm{PK}_{\mathrm{revocation}}$

  - sector specific identifiers $I_{\mathrm{ICC},1}^{\mathrm{sector}}$ and $I_{\mathrm{ICC},2}^{\mathrm{sector}}$ for the user:

    $I_{\mathrm{ICC},1}^{\mathrm{sector}} = (\mathrm{PK}_{\mathrm{sector}})^{\mathrm{SK}_{\mathrm{ICC},1}}$ and $I_{\mathrm{ICC},2}^{\mathrm{sector}} = (\mathrm{PK}_{\mathrm{sector}})^{\mathrm{SK}_{\mathrm{ICC},2}}$

- signing: with keys $\mathrm{SK}_{\mathrm{ICC},1}$, $\mathrm{SK}_{\mathrm{ICC},2}$ and $I_{\mathrm{ICC},1}^{\mathrm{sector}}$ and $I_{\mathrm{ICC},2}^{\mathrm{sector}}$ for $\mathrm{PK}_{\mathrm{sector}}$ and message m

  i. choose K_1, K_2 at random

  ii. compute

     - $Q_1 = g^{K_1} \cdot (\mathrm{PK}_M)^{K_2}$

     - $A_1 = (\mathrm{PK}_{\mathrm{sector}})^{K_1}$

     - $A_2 = (\mathrm{PK}_{\mathrm{sector}})^{K_2}$

  iii. $c = \mathrm{Hash}(Q_1, I_{\mathrm{ICC},1}^{\mathrm{sector}}, A_1, I_{\mathrm{ICC},2}^{\mathrm{sector}}, A_2, \mathrm{PK}_{\mathrm{sector}}, m)$

     (variant parameters omitted here)

  iv. compute

     - $s_1 = K_1 - c \cdot \mathrm{SK}_{\mathrm{ICC},1}$

    – $s_1 = K_2 - c \cdot \mathrm{SK}_{\mathrm{ICC},2}$

  v. output (c, s_1, s_2)

- verification:

  compute

  – $Q_1 = (\mathrm{PK}_{\mathrm{ICC}})^c \cdot g^{s_1} \cdot (\mathrm{PK}_M)^{s_2}$

  – $A_1 = (I^{\mathrm{sector}}_{\mathrm{ICC},1})^c \cdot (\mathrm{PK}_{\mathrm{sector}})^{s_1}$

  – $A_2 = (I^{\mathrm{sector}}_{\mathrm{ICC},2})^c \cdot (\mathrm{PK}_{\mathrm{sector}})^{s_2}$

  – recompute c and check against the c from the signature

- why it works?

$$\begin{aligned}(\mathrm{PK}_{\mathrm{ICC}})^c \cdot g^{s_1} \cdot (\mathrm{PK}_M)^{s_2} &= (\mathrm{PK}_{\mathrm{ICC}})^c \cdot g^{K_1} \cdot (\mathrm{PK}_M)^{K_2} \cdot g^{-c \cdot \mathrm{SK}_{\mathrm{ICC},1}} \cdot (\mathrm{PK}_M)^{c \cdot \mathrm{SK}_{\mathrm{ICC},2}} \\ &= (\mathrm{PK}_{\mathrm{ICC}})^c \cdot g^{K_1} \cdot (\mathrm{PK}_M)^{K_2} \cdot g^{-c \cdot \mathrm{SK}_{\mathrm{ICC},1}} \cdot (g)^{-c \cdot \mathrm{SK}_M \cdot \mathrm{SK}_{\mathrm{ICC},2}} \\ &= (\mathrm{PK}_{\mathrm{ICC}})^c \cdot g^{K_1} \cdot (\mathrm{PK}_M)^{K_2} \cdot g^{-c \cdot \mathrm{SK}_{\mathrm{ICC}}} = g^{K_1} \cdot (\mathrm{PK}_M)^{K_2} = Q_1\end{aligned}$$

- there is a version without A_1, A_2 and the pseudonyms $I^{\mathrm{sector}}_{\mathrm{ICC},1}, I^{\mathrm{sector}}_{\mathrm{ICC},2}$

- Problems:

  – the authorities know the private keys (there is a way to solve it when the user gets two pairs of keys on the device and takes their linear combination)

  – breaking into just 2 devices reveals the system keys

  – possible to create a trapdoor for enabling to link pseudonyms

    – apart from $\mathrm{SK}_{\mathrm{ICC}} = \mathrm{SK}_{\mathrm{ICC},1} + \mathrm{SK}_M \cdot \mathrm{SK}_{\mathrm{ICC},2}$ there is a another relationship for the user u

$$x_u = \mathrm{SK}_{\mathrm{ICC},1} + s_u \cdot \mathrm{SK}_{\mathrm{ICC},2}$$

    – x_u and s_u are dedicated for user u - maybe not in the database but derived from a secret key, say Z

    – domain trapdoor: $T_{\mathrm{domain},u} = \mathrm{PK}^{x_u}_{\mathrm{domain}}$ and s_u (it can be derived from Z alone)

    – then one can conclude that nym_1 and nym_2 correspond to user u, if:

$$T_{\mathrm{domain},u} = \mathrm{nym}_1 \cdot \mathrm{nym}_2^{s_u}$$

ANONYMOUS CREDENTIALS

two commertial products (libraries): Idemix (IBM) and UProve (Microsoft)

some details concerning Idemix

components:

– **actors:** issuer, recipient, verifier, trusted party

- **attributes**: for each attribute there is: `name`, `value` and `type`. The types are `int`, `string`, `date`, `enum` (enumeration). The attributes concern the recipient.
- **credentials**: given by the issuer to the recipient
  i. known (A_k): the issuer knows the value of an attribute
  ii. commited (A_c): the issuer knows a commitment to the attribute but not the commitment itself
  iii. hidden (A_h): the attribute is completely hidden to the issuer
- **pseudonyms:**
  - pseudonyms (for each domain any number of unlinkable pseudonyms can be generated)
  - a single domain pseudonym for each domain (as usual, generated by the master key m_1 as dom^{m_1}, where dom is the public key of a domain
- **keys:**
  - single master key for each user (m_1)
  - single master key for the Issuer – for creation of CL signatures

Cryptographic schemes used by Idemix

CL signatures:

- RSA group, special choice of primes: $p = 2p' + 1$, $q = 2q' + 1$, where p' and q' are primes
- choose at random quadratic residues: $R_1, ..., R_l, Z, S$
- public key: $(n, R_1, ..., R_l, Z, S)$, private key: p, q
- security based on Strong RSA assumption (inability to compute e-roots for $e > 2$
- signature for messages $m_1, ..., m_l$:
  - choose v at random and a prime e of length higher than each $m_1, ..., m_l$
  - $A := ((Z/(S^v \cdot \prod R_i^{m_i}))^{1/e}$
  - the signature is (A, e, V)
- verification: check if Z is the result of the expression:

$$A^e \cdot S^v \cdot \prod R_i^{m_i}$$

Issuing a certificate for values $m_1, ..., m_l$

- somewhat complicated since the Issuer can learn only some attribute signed
- method: a two-party protocol to compute CL signature of the Issuer, algorithm:
  - issuer sends a challenge n_1

- the user chooses v' at random and computes $U := S^{v'} \cdot \prod R_i^{m_i}$ where the product over attributes hidden and commited as well as a ZKP that it has been created in this way, where
  - the user knows hidden attributes
  - the user uses the same attributes as commited
- the issuer checks the proof
- the issuer chooses a random prime e and v''
- the issuer computes

  $Q := Z / \big(U \cdot S^{v''} \cdot \prod_{m_i \text{ known}} R^{m_i}\big)$ and $A := Q^{1/e}$
- (A, e, v'') sent to the user together with a proof that it has been computed in this way
- the user computes $v := v' + v''$, checks the proof and validity of signature (A, e, v)

Presenting a credential

complicated: also involves proofs over encrypted values and the range of attributes. Some attributes may be revealed, but some stay hidden. Moreover, the certificate must not be revealed.

some details for verification of certificate without revealing it:

- values $\widetilde{m_i}$ chosen for each hidden attribute m_i, where $i \in A_{\bar{r}}$
- the user chooses r_A at random and randomizes (A, e, v):
  - $A' := A \cdot S^{r_A}$, $v' := v - e \cdot r_A$
- so called t-values computed:
  - chosen at random: $\tilde{e}$,$\tilde{v}'$
  - $\tilde{Z} := (A')^{\tilde{e}} \cdot S^{\tilde{v}'} \cdot \prod R^{\widetilde{m_i}}$
- these t-values $\tilde{Z}$ and t values from other proofs plus some other data are hashed to get challenge c
- signatures components (s-values) are derived:
  - $\hat{e} := \tilde{e} + c \cdot e$
  - $\hat{v}' := \tilde{v}' + c \cdot v'$
  - $\hat{m}_i := \widetilde{m_i} + c \cdot m_i$

Credential verification - based on recomputation of t-values and recomputing c.

$\tilde{Z}$ recomputed as:

$$(A')^{\hat{e}} \cdot \prod_{i \in A_{\bar{r}}} R_i^{\widehat{m_i}} \cdot S^{\hat{v}'} / \left(\frac{Z}{\prod_{i \in A_{\bar{r}}} R^{m_i}} \right)^c$$

(we remove some parts from Z, what is left must be raised to power c to eliminate the effect of components $c \cdot e$, $c \cdot v'$, $c \cdot m_i$ when using the exponents $\hat{e}$, $\hat{v}'$, $\hat{m}_i$

ranges have to be checked, etc

...

IDENTIFICATION

running wireless communication protocol may enable tracing a user. Threats:

- explicit exchange of identifiers: an eavsdropper learns who is communicating with whom

- strong cryptographic proofs created during identification: can be misused for proving presence to the third parties

elimination of explicit identifiers:

- at each communication round Alice and Bob create random nonce (nonces) for the next round

- even more secure: if n is such a nonce, then Alice uses n' where n' is the same as n except for one bit at a random position

deniability:

- the idea is that a transcript of a communication (including the answer from the Prover created with his private key) can be simulated

  consequence: a third party has no grounds to believe the communication transcript presented to him

- wrong example: challenge-response algorithm with digital signature:

  1. the Verifier selects x at random and sends to the Prover

  2. the Prover returns his signature s over x

     unfortunately s can serve as a proof of the claim of the Verifier: "I have talked to Prover" if x is a signature of the Verifier or somthing that only could be created by the Verifier

- good example: static Diffie-Hellman protocol

- good example: Stinson-Wu for Prover with the key pair $(a, A = g^a)$

  1. the Verifier chooses x at random, computes $X: = g^x$ and $Y := \mathrm{Hash}(A^x)$ and sends X, Y to the Prover

  2. the Prover computes $Z := X^a$ and aborts if $Y \neq \mathrm{Hash}(Z)$

  3. the Prover sends Z

  4. the Verifier accepts iff $Z = A^x$

- Stinson-Wu does not create an oracle for DH Problem, the Verifier must send a challege for which somebody knows x

- however: our attack works Eve provides the Prover correct X, Y and x encrypted with the $\mathrm{Hash}(Z)$. After the interaction with the Prover the Verifier returns x to Eve as a proof of interaction

X. COMMUNICATION SECURITY – SSL/TLS

many mistakes in practice:

- risk of common (standard) groups
- cryptanalysis: most efficient number field sieve (NFS):
  - complexity subexponential (for $\mathbb{Z}_{\mathbb{P}}$ it is

    $$\exp(1.93 + o(1))\big(\log p^{1/3} (\text{loglog}\, p)^{2/3}\big)$$

  - most time precomputation independent from the target number y (where $\log y$ to be computed in a given group)
  - the time dependant from y can be optimized to subexponential but much lower
  - 512-bit groups can be broken, MitM attack can be mounted
- standard safe primes – seem to be ok, but attacker can amortize the cost over many attacks
- TLS with DH: frequently "export-grade" DH with 512 bit primes, about 5% of servers support DHE_EXPORT, most servers (90% and more) use a few primes of a given length, after a precomputation breaking for a given prime: reported as 90 sec
- TLS: client wants DHE, server offers DHE_EXPORT, but one can manipulate the messages exchanged, so that the client treats the (p_{512}, g, g^b) as a response to DHE – it is not an implementation bug!
- sometimes non safe prime used (the number $\frac{p-1}{2}$ is composite), Pohling-Hellman method can be used
- DH-768 breakable on academic level, claims: DH-1024 for state agencies in some countries
- recommendations:
  - avoid fixed prime groups
  - transition to EC (partially withdrawn due to required transition to post-quantum instead)
  - deliberately do not downgrade security, even if seems to be ok
  - follow the progress in computer algebra

Padding attack (Serge Vaudenay)

Scenario:

- for encryption the plaintext should have the length as a multiply of b

- padding is always applied (even if not necessary)
- if i positions have to be padded, write i times the number i . de-padding is then obvious
- encrypt the resulting padded plaintext $x_1, \ldots, x_N$ in the CBC mode with IV (fixed or random) and a block cipher Enc:

$$y_1 = \mathrm{Enc}(\mathrm{IV} \oplus a_1), \quad y_i = \mathrm{Enc}(y_{i-1} \oplus x_i)$$

- properties of CBC:
  - efficiency
  - a warning about confidentiality weakness: if IV fixed, then one can check that two plaintexts have the same prefix of a given size

attack:

- manipulate the ciphertext
- destination node decrypts, it can detect incorrect padding
- decision: what to do if the padding is incorrect? Each reaction will turn out to be wrong:
  - $\to$ reaction “reject”: creates padding oracle (attacker tests the behavior)
  - $\to$ reaction “proceed”: enables manipulation of the plaintext data

last word oracle:

- goal: compute $a = \mathrm{Dec}(y)$ for a block y
- create an input for padding oracle:
  - create a 2 block ciphertext: $r = r_1 \ldots r_b$ chosen at random, $c := r|y$
  - oracle call: if $\mathrm{Oracle}(c) = \mathrm{valid}$, then $\mathrm{Dec}(y) \otimes r$ should yield a correct padding. whp this happens if $a_b = r_b \oplus 1$ (that is, if the padding consists of a single “1”). Other options – “22”, “333”, “4444”,.... are less probable
  - it may happen that the oracle says `valid` because of other correct padding. The following procedure solves the problem (idea: change conseqtive words in the padding until invalid:

```
1. pick $r_1, r_2 \ldots, r_b$ at random, take $i = 0$
2. put $r = r_1 r_2 \ldots r_{b-1}(r_b \oplus i)$
3. run padding oracle on $r|y$, if the result invalid then increment $i$ and goto (2)
4. $r_b := r_b \oplus i$  /* now we have a correct padding of an unknown length
5. for $j = b$ to 2:
   $r := r_1 \ldots r_{b-j}(r_{b-j+1} \oplus 1) r_{b-j} \ldots r_b$
   /* attempting to disturb padding, from left to right
```

ask padding oracle for $r|y$, if `invalid` then output $(r_{b-j+1} \oplus j) \ldots (r_b \oplus j)$ and halt

6. output $r_b \oplus 1$ /* last choice, manipulating all positions except the rightmost has not created an error so the padding has length 1, so $y_b \oplus r = 1$ or $y_b = r_b \oplus 1$

block decryption oracle

let $a_1 \ldots a_b$ be the plaintext of y

decryption:

- get a_b via the last word oracle
- proceed step by step learning a_{j-1} once $a_j, \ldots, a_b$ are already known
  1. set $r_k := a_k \oplus (b-j+2)$ for $k = j, \ldots, b$ /* preparing the values so that the padding values $(b-j+2)$ appear at the end)
  2. set $r_1, \ldots, r_{j-1}$ at random, $i := 0$ /* search for the value that makes a proper padding
  3. $r := r_1 \ldots r_{j-2}(r_{j-1} \oplus i) r_j \ldots r_b$
  4. if output on $r|y$ is `invalid`, then $i := i+1$ and goto 3
  5. output $r_{j-1} \oplus i \oplus (b-j+2)$

decryption oracle

- block by block, (after decryption we have to XOR with the previous ciphertext block due to CBC construction)
- the only problem is the first block if IV is secret

bomb oracles:

- padding oracle in SSL/TLS breaks the connection if a padding error occurs , so it can be used only once
- bomb oracle: try a longer part at once, execute many trials

other paddings:

easy to adjust the attack in the following cases (the reason isthat we KNOW hat to expect on a given position of the pading) :

- $00 \ldots 0n$ instead of $nn \ldots n$
- $12 \ldots n$ instead of $nn \ldots n$

the padding where it would not work is a one with random data on the added positions

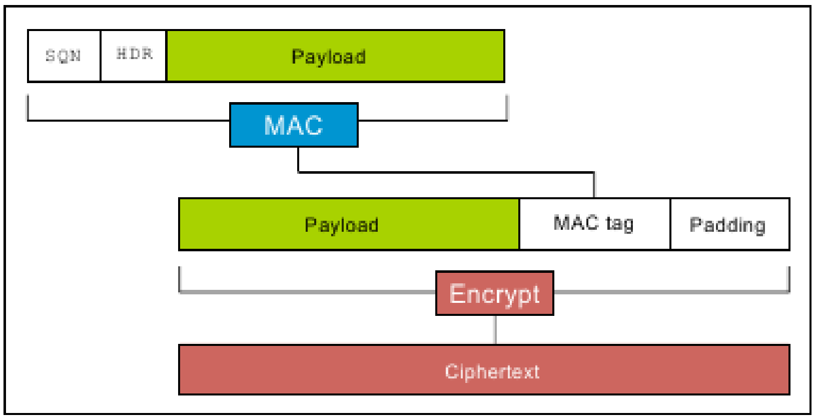
Applications for (old) versions of SSL/TLS, ...

- if MAC applied before padding, then padding oracle techniques can be applied

- wrong MAC and wrong padding create the same error message - from SSL v3.0, debatable whether it is impossible to recognize situation via side channel (response time)
- TLS attempts to hide the plaintext length by variable padding:
  - checking the length of padding: take the last block y, send $r|y$ where the last word of r is $n \oplus 1$. acceptance indicates that the padding is of length n
  - checking paddings longer than a block: send ry_1y_2 where y_1y_2 are the last blocks
- IPSEC: discards message with a wrong padding, no error message, but there might be other activities to process errors (they may leak information)
- WTLS: decryption-failed message in clear (!) session not interrupted
- SSH: MAC after padding (+)

Lucky Thirteen

- concerns DTLS (similar to TLS for UDP connections)
- MAC-Encode-Encrypt paradigm (MEE), MAC is HMAC based



- 8-byte SQN, 5-byte HDR (2 byte version field, 1 byte type field, 2 byte length field)
- size of the MAC: 16 bytes (HMAC-MD5), 20 bytes (HMAC-SHA1), 32 bytes (HMAC-SHA-256)
- padding: $p+1$ copies of p, at least one byte must be added
- after receiving: checking the details: padding, MAC, (underflow possible if padding manipulated and removing blindly)
- HMAC of M:

  $T := H((K_a \oplus \text{opad})||H((K_a \oplus \text{ipad})||M))$
- **Distinguishing attack:**
  - → M_0 : 32 arbitrary bytes followed by 256 copies of 0xFF

→ M_1: 287 bytes followed by $0x00$

→ both 288 bytes, 18 plaintext blocks

→ encoded $M_d||T||\text{pad}$, we aim to guess d

→ C – the ciphertext

→ create a ciphertext C' by truncating all parts corresponding to $T||\text{pad}$

→ give HDR$||C'$ for decryption

→ if M_0: the 256 copies of 0xFF interpreted as padding and removed, remaining 32 bytes as short message and MAC, calculating MAC: 4 hash computed, then typically error returned to the attacker

→ if M_1: 8 hash evaluations

Plaintext recovery attacks

- C^* – the block of ciphertext to be broken, C' – the ciphertext block preceding it
- we look for P^*, where $P^* = \text{Dec}(C^*) \oplus C'$
- assume CBC with known IV, $b = 16$ (as for AES). $t = 20$ (as for HMAC-SHA-1)
- let Δ be a block of 16 bytes, consider

$$C^{\text{att}}(\Delta) = \text{HDR}||C_0||C_1||C_2||C' \oplus \Delta||C^*$$

  it represents 4 non-IV blocks in the plaintext, the last block is:

$$P_4 = \text{Dec}(C^*) \oplus (C' \oplus \Delta) = P^* \oplus \Delta$$

- case 1: P_4 ends with 0x00 byte:
  - 1 byte of padding is removed, the next 20 bytes interpreted as MAC, 43 bytes left - say R. MAC computed on SQN|HDR|R of 56 bytes
- case 2: P_4 ends with padding pattern of ≥ 2 bytes:
  - at least 2 bytes of padding removed, 20 bytes interpreted as MAC, at most 42 bytes left, MAC over at most 42+13=55 bytes
- case 3: P_4 ends with no valid padding:
  - according to RFC of TLS 1.1, 1.2 treated as with no padding , 20 bytes treated as MAC, verification of MAC over 44+13=57 bytes

    – MAC is computed to avoid other timing attack!
- time: case 1 and 3: 5 evaluations of SHA-1, case 2: 4 evaluations of SHA-1, detection of case 2 possible in LAN
- in case 2: most probable is the padding 0x01 0x01, all other paddings have probability about $\approx \frac{1}{256}$ of probability of 0x01 0x01, so we may assume that $P_4 = P^* \oplus \Delta$ ends with 0x01 0x01. Then we derive the last two bytes of P^*.

repeat the attack with Δ' that has the same last two bytes as Δ to check if the padding has the length bigger than 2 (we are changing the byte 3 and observe whether the case 2 occurs, if it is so, then padding has length 2).

- after recovery of the last two bytes the rest recovered byte by byte from right to left:
  - the original padding attack
  - e.g. to find 3rd rightmost byte set the last two bytes Δ so that P_4 ends with 0x02 0x02, then try different values for the Δ_{13} so that Case 2 occurs (meaning that P_4 ends with 3 bytes 0x02
  - average time: $14 \cdot 2^7$ trials
- practical issues:
  - → for TLS after each trial connection broken, so multi-session scenario
  - → timing difference small, so necessary to gather statistical data
  - → complexity in fact lower, since the plaintexts not from full domain: e.g. http username and password are encoded Base64
  - → partial knowledge may speed up the recovery of the last 2 bytes
  - → less efficient configuration of the lengths for HMAC-MD5 and HMAC-SHA-256

BEAST

attack, phase 0:

1. P to be recovered (e.g. a password, cookie, etc), requires ability to force Alice to put secret bits on certain positions
2. force Alice to send $0...0P_0$ (requires malware on Alice computer) – of course encrypted
3. eavesdrop and get $C_p = \text{Enc}(C_{p-1} \oplus 0...0P_0)$
4. guess a byte g
5. force Alice to send encrypted plaintext $C_{i-1} \oplus C_{p-1} \oplus 0...0g$:

   then Alice sends $C_i = \text{Enc}(C_{i-1} \oplus C_{i-1} \oplus C_{p-1} \oplus 0...0g) = \text{Enc}(C_{p-1} \oplus 0...0g)$
6. if $C_i = C_p$ then $P_0 = g$

attack phase 1:

1. P_0 already known
2. force Alice to send $0...0P_0P_1$ and proceed as in phase 0

last phase: we get the test for the whole $P_0...P_{15}$

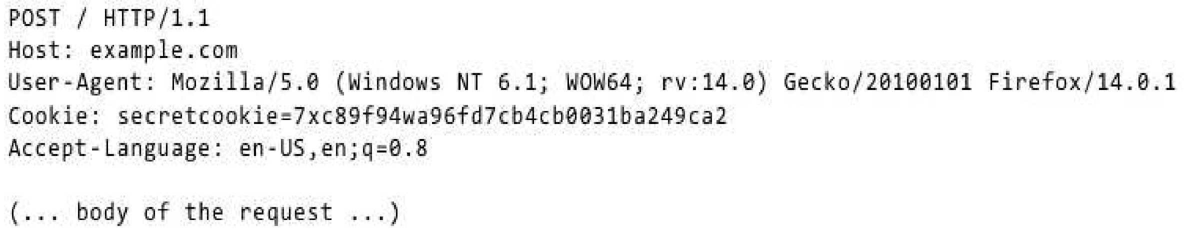
protection: browser must be carefully designed and do not admit injecting plaintexts (SOP- Same Origin Protection). Some products do not implement it.

CRIME (2012)

– based on compression algorithm used by some (more advanced) versions of TLS

– compression: LZ77 and then Huffman encoding, LZ77- sliding window approach: instead of a string put a reference to a previous occurence of the same substring

– idea of recovering cookie:

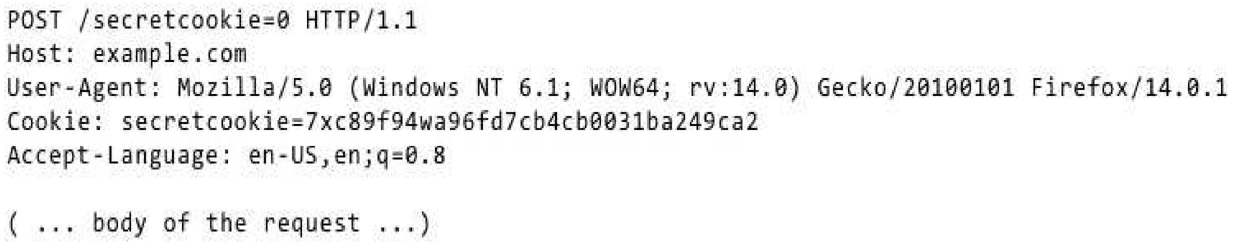


```
POST / HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:14.0) Gecko/20100101 Firefox/14.0.1
Cookie: secretcookie=7xc89f94wa96fd7cb4cb0031ba249ca2
Accept-Language: en-US,en;q=0.8

(... body of the request ...)
```

Listing 1: *HTTP request of the client*

modified POST:



```
POST /secretcookie=0 HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:14.0) Gecko/20100101 Firefox/14.0.1
Cookie: secretcookie=7xc89f94wa96fd7cb4cb0031ba249ca2
Accept-Language: en-US,en;q=0.8

( ... body of the request ...)
```

Listing 2: *HTTP request modified by the attacker*

LZ77 compresses the 2nd occurence of secretcookie= or secretcookie=0. We try all

secretcookie=i to find out the case when compression is easier (secretcookie=7)

when the first character recovered the attacker repeats the attack for the second character (trying all "secretcookie=7i" in the preamble)

TIME

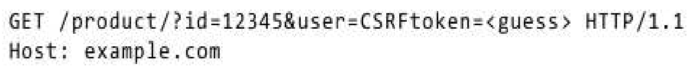
- again based on compression but now on the server side (from the client to the server compression might be disabled and CRIME fails)
- works if the server includes the client's request in the response (most do!)
- works even if SOP is enabled. SOP does not control data with the tag `img`, so the attacker can manipulate the length and therefore influence the number of blocks for block encryption
- the attacker requires malicious Javascript on the client's browser
- the attacker tries to get the secret value sent from the server to the client

- mechanism:
  - → as in CRIME, the request sends "secretvalue=x" where x varies
  - → the response is compressed, so it takes either "secretvalue=" or "secretvalue=x"
  - → the length manipulated so that either one or two packets are sent – connection specific data must be used: Maximum Transmission Unit
  - → RTT (round trip time) measured
- independent on the browser, it is not an implementation attack!
- countermeasure: restrict displaying images

BREACH

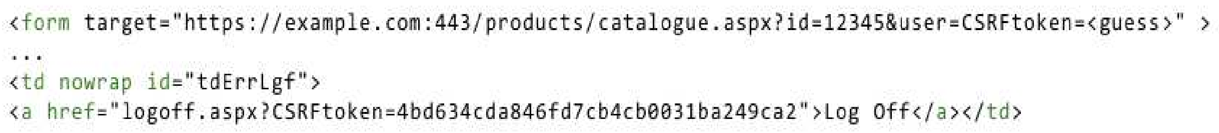
Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext

- attack against HTTP compression and not TLS compression as in case of CRIME
- a victim visits attacker-controlled website (phishing etc).
- force victim's computer to send multiple requests to the target website.
- check sizes of responses



```
GET /product/?id=12345&user=CSRFtoken=<guess> HTTP/1.1
Host: example.com
```

Listing 4: *Compromised HTTP request*



```
<form target="https://example.com:443/products/catalogue.aspx?id=12345&user=CSRFtoken=<guess>" >
...
<td nowrap id="tdErrLgf">
<a href="logoff.aspx?CSRFtoken=4bd634cda846fd7cb4cb0031ba249ca2">Log Off</a></td>
```

Listing 5: *HTTP response*

- requirements: application supports http compression, user's input in the response, sensitive data in the response
- countermeasures:
  - → disabling compression
  - → hiding length (randomizing the length of the output – it makes the attacks only harder if the attack can be repeated many times)
  - → no secrets in the same response as the user's data
  - → masking secret: instead of S send $R||S \oplus R$ for random R (fresh in each response)
  - → trace behaviour of requests and warn the user

POODLE (2014)

in SSL v.3.0 using technique from BEAST:

- padding is not covered by MAC so the attacker can manipulate it
- padding non-deterministic: padding 1 to L bytes ($L=$ block length), the last byte denotes the number of preceding padding random bytes
- encrypted POST request:

  POST /path Cookie: name=value... ⟨r\n\r\n⟩ body ||20-byte MAC||padding
- manipulations such that:
  - the padding fills the entire block (encrypted to C_n)
  - the last unknown byte of the cookie appears as the last byte in an earlier block encrypted into C_i
- attack: replace C_n by C_i and forward to the server

  usually reject

  accept if $\mathrm{Dec}_K(C_i)[15] \oplus C_{n-1}[15] = 15$, thereby $P_i[15] = 15 \oplus C_{n-1}[15] \oplus C_{i-1}[15]$

  proceed in this way byte by byte
- downgrade dance: provoke lower level of protection by creating errors say in TLS 1.0, and create connection with SSL v3.0
- the attack does not work with weak (!) RC4 because of no padding

Weaknesses of RC4

- known weaknesses:
  - → the first 257 bytes of encryption strongly biased, ≈200 bytes can be recovered if ≈232 encryptions of the same plaintext available

    simply gather statistics as in case of Ceasar cipher
  - → at some positions (multiplies of 256) if a zero occurs, then the next position more likely to contain a zero
- broadcast attack: force the user to encrypt the same secret repeatedly and close to the beginning
- countermeasure: no secrets in the initial part!

TLS 1.2

differences with TLS 1.1 and TLS 1.0:

- explicit IV instead of implicit IV
- IDEA and DES 64bit removed
- MD5/SHA-1 is replaced with a suite specified hash function – SHA-256 for all TLS 1.2 suites, but in the future also SHA-3, ...

- digitally-signed element includes the hash algorithm used
- Verify_data length is no longer fixed length $\Rightarrow$ TLS 1.2 can define SHA-256 based cipher suites
- new encryption modes allowed: CCM, GCM

CCM encryption mode, just to avoid patent threats (triggered by request to patent OCB mode - patented in USA, exempt for general public license for non-commertial use)

Prerequisites: block cipher algorithm; key K; counter generation function; formatting function; MAC length $Tlen$

Input: nonce N; payload P of $Plen$ bits; valid associated data A

Computation: Steps:

1. formatting applied to (N, A, P), result: blocks $B_0, ..., B_r$
2. $Y_0 := \mathrm{Enc}_K(B_0)$
3. for $i = 1$ to r: $Y_i := \mathrm{Enc}_K(B_i \oplus Y_{i-1})$
4. $T := \mathrm{MSB}_{Tlen}(Y_r)$
5. generate the counter blocks $\mathrm{Ctr}_0, \mathrm{Ctr}_1, ..., \mathrm{Ctr}_m$ for $m = Plen/128$
6. for $j = 0$ to m: $S_j := \mathrm{Enc}_K(\mathrm{Ctr}_j)$
7. $S := S_1 || ... || S_m$
8. $C := (P \oplus \mathrm{MSB}_{Plen}(S)) || (T \oplus S_0)$

Decryption:

1. return INVALID, if $Clen < Tlen$
2. generate the counter blocks $\mathrm{Ctr}_0, \mathrm{Ctr}_1, ..., \mathrm{Ctr}_m$ for $m = Plen/128$
3. for $j = 0$ to m: $S_j := \mathrm{Enc}_K(\mathrm{Ctr}_j)$
4. $S := S_1 || ... || S_m$
5. $P := \mathrm{MSB}_{Clen}(C) \oplus \mathrm{MSB}_{Plen}(S)$
6. $T := \mathrm{LSB}_{Tlen}(C) \oplus \mathrm{MSB}_{Tlen}(S_0)$
7. If N, A or P invalid, then return INVALID, else reconstruct $B_0, ..., B_r$
8. recompute $Y_0, ..., Y_r$
9. if $T \neq \mathrm{MSB}_{Tlen}(Y_r)$, then return INVALID, else return P.

properties:

- data length must be known in advance
- two passes

GCM (The Galois/Counter Mode)

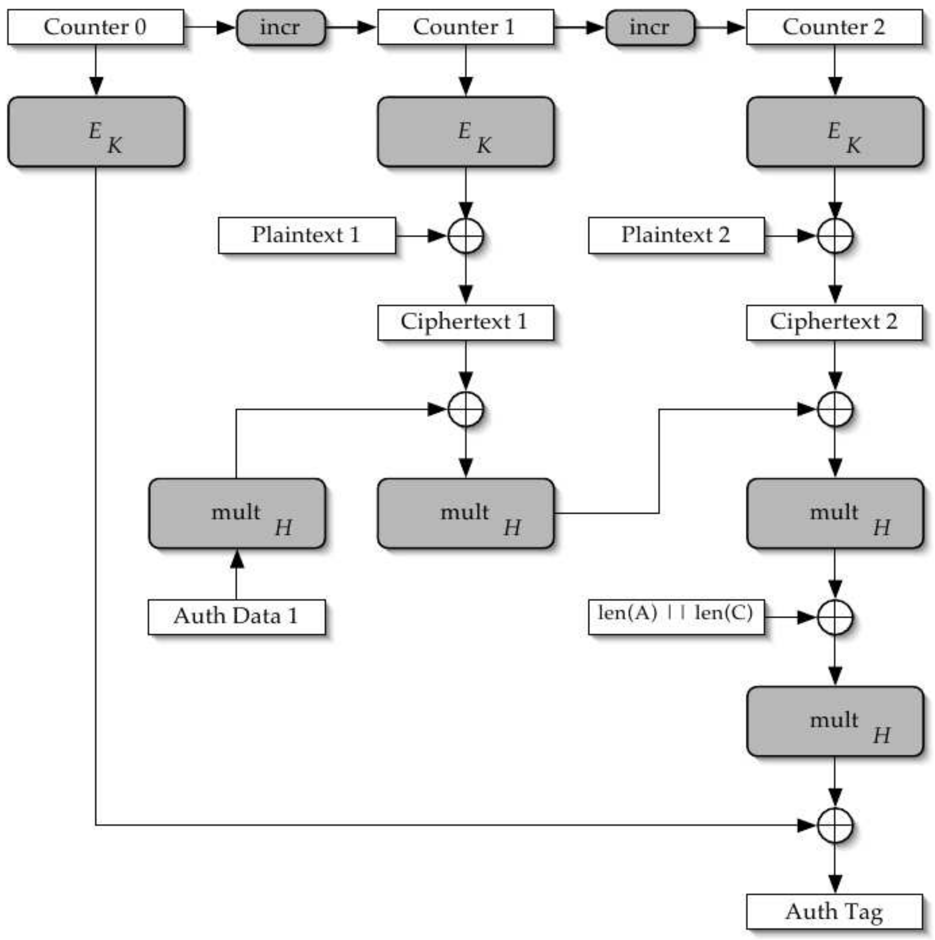
background:

- popular, as replacement for CBC mode (because of attacks presented) and weaknesses of RC4 (forbidden in the current TLS)
- received fundamental critics already before standardization
- finally (April 2018) Google decided to remove it until April 2019
- operations over GF(2^{128}), addition in the field represented by xor ($\oplus$)

Computation: Steps:

1. $H := \mathrm{Enc}_K(0^{128})$
2. $Y_0 := \mathrm{IV} || 0^{31}1$ if length of IV should be 96

   or $Y_0 := \mathrm{GHASH}(H, \{\}, \mathrm{IV})$
3. $Y_i := \mathrm{incr}(Y_{i-1})$ for $i = 1, \ldots, n$ (counter computation)
4. $C_i := P_i \oplus \mathrm{Enc}_K(Y_i)$ for $i = 1, \ldots, n-1$ (counter based encryption)
5. $C_n^* := P_n \oplus \mathrm{MSB}_u(\mathrm{Enc}_K(Y_n))$ (the last block need not to be full)
6. $T := \mathrm{MSB}_t(\mathrm{GHASH}(H, A, C)) \oplus \mathrm{Enc}_K(Y_0)$



Details of computation of the tag

$\mathrm{GHASH}(H,A,C)=X_{m+n+1}$ where m is the length of authenticating information A, and:

X_i equals:

0	for $i=0$
$(X_{i-1}\oplus A_i)\cdot H$	for $i=1,\ldots,m-1$
$((X_{i-1}\oplus(A_m^*\|\|0^{128-v}))\cdot H$	for $i=m$
$(X_{i-1}\oplus C_i)\cdot H$	for $i=m+1,\ldots,m+n-1$
$((X_{m+n-1}\oplus(C_m^*\|\|0^{128-u}))\cdot H$	for $i=m+n$
$((X_{m+n}\oplus(\mathrm{len}(A)\|\mathrm{len}(C)))\cdot H$	for $i=m+n+1$

Decryption:

1. $H:=\mathrm{Enc}_K(0^{128})$
2. $Y_0:=\mathrm{IV}\|\|0^{31}1$ if length of IV should be 96

   or $Y_0:=\mathrm{GHASH}(H,\{\},\mathrm{IV})$
3. $T':=\mathrm{MSB}_t(\mathrm{GHASH}(H,A,C))\oplus\mathrm{Enc}_K(Y_0)$, is $T=T'$?
4. $Y_i:=\mathrm{incr}(Y_{i-1})$ for $i=1,\ldots,n$
5. $P_i:=C_i\oplus\mathrm{Enc}_K(Y_i)$ for $i=1,\ldots,n$
6. $P_n^*:=C_n^*\oplus\mathrm{MSB}_u(\mathrm{Enc}_K(Y_n))$

Fundamental flaws (by Nils Ferguson)

- engineering disadvantages: message size up to $2^{36}-64$ bytes, arbitrary bit length (instead of byte length)
- collisions of IV: the same pseudorandom string for encryptions
- collisions of Y_0 also possible. Due to birthday paradox 2^{64} executions might be enough for 128-bit values, for massive use in TLS 2^{64} is maybe too small

Ferguson attack via linear behavior

- authenticating tag computed as leading bits of $T=K_0+\prod_{i=1}^{N} D_i\cdot H^i$
- representing elements of $\mathrm{GF}(2^{128})$: X – as an abstract element of the field, $\mathrm{Poly}(X)$–as a polynomial over GF(2) with coefficients $X_0,X_1,\ldots,X_{127}$
- multiplication by a constant D: $X\to D\cdot X$ can be expressed by multiplication by a matrix:

$$(D\cdot X)^T=M_D\cdot X^T$$

- squaring is linear: $(A+B)^2=A^2+B^2$ (field of characteristic 2), so

$$(X^2)^T=M_S\cdot X^T$$

  where S is a fixed matrix

- the goal is to find C' such that

$$\sum_{i=1}^{N} C_i \cdot H^i = \sum_{i=1}^{N} C'_i \cdot H^i$$

  or at least leading bits are the same (as they are taken to MAC)

- let $C_i - C'_i = E_i$, so we need $\sum_{i=1}^{N} E_i \cdot H^i = 0$

- we confine ourselves to $E_i = 0$ except for i which is a power of 2. Let $D_i = E_{2^i}$. Let $2^n = N$

$$E^T = \prod_{i=1}^{n} M_{D_i} \cdot (M_S)^i \cdot H^T$$

- let

$$A_Z = \prod_{i=1}^{n} M_{D_i} \cdot (M_S)^i$$

- then we have $E^T = A_D \cdot H^T$

- write equations to force a row of A_D to zero, equation for each bit, so 128 linear equations for a row

- there are $128 \cdot n$ free variables describing the values D_i

- we can find a nonzero solution for $n-1$ rows

- consider messages of length 2^{17}, $D_0 = 0$ due to issues like not changing the length,

- $D_1, D_2, .., D_{17}$ can be chosen so that 16 rows of A_D are zero,

- GCM used with 32 bits MAC, so still 16 bits might be non-zero, so the chance of forgery is 2^{-16}

Recovering *H*

- from a collision we have 16 linear equations for H, so we may describe H by a sequence of 112 unknown bits H' and expression

$$H^T = X \cdot H'^T$$

  where X is a matrix 128x112.

$$E^T = A_D \cdot X \cdot H'^T$$

- now repeat the same with H' - the attack is easier as there are only 112 bits and not 128, there are 112 equations per row and $17 \cdot 128$ free variables, so one can zeroize 19 rows and get the chance of forgery of 2^{-13}. If succeeds, then 13 new variables of H known

- repeat until all bits of H known.

- **finally it is possible to forge MAC for any message**

ChCha

- stream cipher, Chacha extends 256 bit stream key into 2^{64} randomly accessible streams, each of 2^{64} blocks of 64 bytes
- Daniel Berstein's construction,
- used by Google for communication between mobile devices and Google websites, also RFC, in libraries (OpenSSL,...)
- variant of SALSA from European ECRYPT competition
- faster than AES
- working on four 32-bit words
- quarter-round of SALSA 20 for inputs a, b, c, d
  1. $b = b$ xor $(a + d) \lll 7$
  2. $c = c$ xor $(b + a) \lll 9$
  3. $d = a$ xor $(c + b) \lll 13$
  4. $a = a$ xor $(d + c) \lll 18$
- quarter-round of ChaCha20 (better diffusion)
  1. $a = a + b$; $d = d \operatorname{xor} a$; $d = d \lll 16$
  2. $c = c + d$; $b = b \operatorname{xor} c$; $b = b \lll 12$
  3. $a = a + b$; $d = d \operatorname{xor} a$; $d = d \lll 8$
  4. $c = c + d$; $b = b \operatorname{xor} c$; $b = b \lll 7$
- Chacha matrix 4x4: (where 'input' = 'block counter'+nonce)

const	const	const	const
key	key	key	key
key	key	key	key
input	input	input	input

- round: 8 quarter-rounds:
  - 1st column, 2nd column, 3rd column, 4th column
  - quarter-round on diagonals

    QUARTERROUND($x0, x5, x10, x15$),

    QUARTERROUND($x1, x6, x11, x12$)

    QUARTERROUND($x2, x7, x8, x13$)

    QUARTERROUND($x3, x4, x9, x14$)
- ChaCha20 - 20 rounds

Poly1035

- designed by Bernstein, no patent, fast
- MAC algorithm, 16 byte MAC
- variable message length, 16 byte AES key, 16 byte additional key r, 16 byte nonce
- works with AES, not weaker than AES, but if AES fails, then one can reuse safely with a different encryption scheme
- the only way to break Poly is to break AES
- per message overhead is low (even for short messages)
- no long lookup tables, therefore it fits into cache memory even if multiple keys used
- keys k - for AES, r - 16 byte, treated as little endian 128-bit number
- some limitations on r because of efficiency of implementation

  $r = r_0 + r_1 + r_2 + r_3$ where

  $r_0 \in \{0, 1, ..., 2^{28} - 1\}$, $r_1/2^{32} \in \{0, 4, 8, 12, ..., 2^{28} - 4\}$,...
- nonces: 16 bit, encrypted by AES
- message: divided into 16 byte chunks. Each chunk treated as 17-byte number with little-endian, where the most significant byte is an added 1 or 0, result: $c_1, ..., c_q$
- authenticator

  $$(((c_1 r^q + c_2 r^{q-1} + ... + c_q r^1) \bmod 2^{130} - 5) + \mathrm{AES}_k(\text{nonce})) \bmod 2^{128}$$

  denoted also $H_r(m) + \mathrm{AES}_k(\text{nonce})$
- $2^{130} - 5$ is a prime,
- a nonce must be used at most once
- security depends on the fact that for two random messages m, m' of length L pbb that $H_r(m) = H_r(m') + g$ is at most $8\lceil L/16\rceil / 2^{108}$ – that is, all differentials have small probability

TLS 1.3 (August 2018)

- list of symmetric algorithm contain now only AEAD (authenticated Encryption with Associated Data)
- separating key agreement and authentication from record protection (key length...)
- zero round-trip time (0-RTT) added (for some application data to be added encrypted with the first message)
- static DH and RSA for negotiation of keys removed (now forward security achieved)

- after ServerHello all handshake messages are encrypted
- key derivation function HKDF (hash key derivation function: first derive PRK via hashing of the shared secret+salt+user input, from PRK derive the secrets by hashing with sequence number)
- handshake state machine restructured to be more consistent and with no superfluous messages
- elliptic curve algorithms in base specification, EdDSA included, point format negotiation removed (one point format)
- RSA padding changed to RSASSA-PSS
- no support for some elliptic curves, MD5, SHA-224
- no compression
- prohibiting RC4 and SSL negotiation for backwards compatibility
- negotiation mechanism removed, instead a version list provided in an extension
- authentication with DH, or PSK (pre-shared key), or DH with PSK
- session resumption with PSK
- added: Chacha20 stream cipher with Poly1305 authentication code
- Addition of the Ed25519 and Ed448 digital signature algorithms
- Addition of the x25519 and x448 key exchange protocols

XI. CERTIFICATES and – SSL/TLS

“Certified Lies”

- rogue certificates + MitM attack: the user believes that is directed elsewhere
- no control over root CA’s worldwide, indicated either by operating system or the browser
- compelled assistance from CA’s ?

ROGUE Certificates and MD5

- target: create a certificate (webserver, client) that has not been issued by CA
- not forging a signature contained in the certificate but:
    i. find two messages that $\mathrm{Hash}(M_0) = \mathrm{Hash}(M_1)$ and M_0 as well as M_1 have some common prefix that you expect in a certificate (e.g. the CA name)

ii. submit a request corresponding to M_0, get a certificate with the signature over Hash(M_0)

iii. copy the signature from the certificate concerning M_0 to a certificate based on M_1

- problems: some data in M_0 are to be guessed : sequential number, validity period,

  some other are known in advance: distinguished name, ...

legitimate website certificate		rogue CA certificate
serial number issuing CA validity period		serial number issuing CA validity period
domain name	chosen prefixes	rogue CA name 1024 bit RSA public key extensions “CA=true”
2048 RSA public key	collision bits	tumor
extension “CA=false”	identical suffix	

Table.

- finding M_0 and M_1 has to be fast (otherwise the guess about the serial number and validity will fail) - e.g. a day over the weekend

- attack on MD5, general picture:

message A		message B
prefix P		prefix P'
padding S_r		padding S'_r
birthday blocks S_b		birthday blocks S'_b
near-collision block $S_{c,1}$		near-collision block $S'_{c,1}$
near-collision block $S_{c,2}$		near-collision block $S'_{c,2}$
...		...
near-collision block $S_{c,r}$	←collision→	near-collision block $S'_{c,r}$
suffix		suffix

Table.

- identical prefix, birthday bits, near collision blocks:

  - birthday bits: 96, end at the block boundary, they are RSA bits – in the genuine certificate, “tumor” (ignored part by almost all software- marked as a comment extension) – in the rogue certificate

    birthday bits make the difference of intermediate hash values computed for both certificates fall into a *good class*

  - then apply 3 near-collision blocks of 512-bits. website: we have “consumed” 208 + 96 + 3·512 = 1840 bits of the RSA modulus. Rogue certificate: all bits concerned are in the “tumor”

- after collision bits: 2048-1840 = 208 bits needed to complete the RSA modulus of the webpage – we have to generate an RSA number with the prefix of 1840 bits already fixed

  – continued so that two prime factors obtained:

  → B denotes the fixed 1840-bit part of the RSA modulus followed by 208 ones

  → select at random 224-bit integer q until $B \bmod q < 2^{208}$, continue until both q and $p = \lfloor B/q \rfloor$ are prime. Then

    – $p \cdot q$ is an RSA number

    – $p \cdot q < B$, $B - p \cdot q = B - q \cdot \lfloor B/q \rfloor < 2^{208}$. Hence $p \cdot q$ has the same 1840 most significant bits as B

  → this RSA number is not secure, but still factorizing it is not feasible and cannot be checked by CA before signing (as the smallest factor is more than 67-digit prime)

  → ... one can create RSA signature for the webpage for the certificate request

- attack complexity (number of hash block evaluations) for a chosen prefix MD5: 2^{49} at 2007, 2^{39} in 2009, not much motivation for more work - remove MD5 certificates! (For a collision: 2^{16})

  for SHA-1 still 2^{77} in 2012 (for a collision: 2^{65})

- history:

  → attack found

  → real collision computed as a proof-of-concept

  → CA informed and given time

  → publication

  → code available

FLAME

- malware discovered 2012, 20MB, sophisticated code, mainly in Middle East, government servers attacked

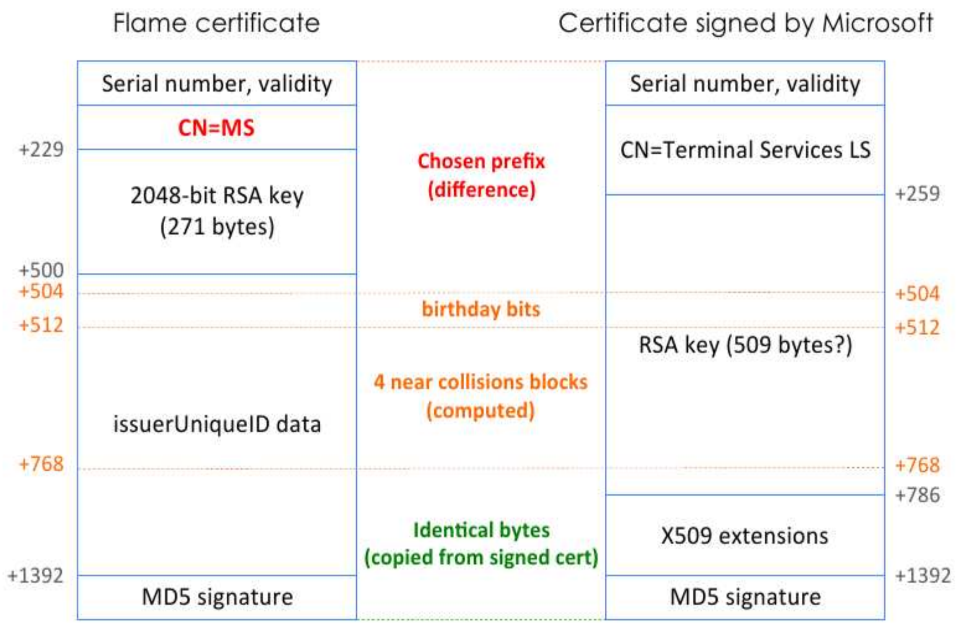
- draft of the attack:

  – client attempts to resolve a computer name on the network, in particular makes WPAD (Web Proxy Auto-Discovery Protocol) requests

  – Flame claims to be WPAD server, provides wpad.dat configuration file

  – victim that gets wpad.dat sets its proxy server to a Flame computer (later no sniffing necessary!)

- Windows updates provided by the Flame computer. The main problem is that the updates must be properly signed to be installed!
- signatures obtained for terminal Services, certificates issued by Microsoft LSRA PA. No Extended Key Usage restrictions – allows code signing, (except for Microsoft Hydra X.509 extension – this cannot be used for code-signing on Vista and Windows 7)
- till 2012 still signatures with MD5 hash used
- MD5 collision necessary to remove extension



XII. CACHE ATTACKS

idea:

- applies to multiprocess architectures, with strict separation between processes offered by the system: hypervisor and virtualization, sandboxing, ...
- trying to get secrets from one processes by another process with no priviledges
- despite separation protection the processes share cache
- there is a strict control over the cache content but **cache hits and cache misses** might be detected by **timing for the attacker's process** (and not of the victim process)
- the timing for cache access should somehow depend on the sensitive information to be retreived
- difficulty: other than in the classical cryptanalysis – access to plaintext or ciphertext might be impossible (they belong to the victim process) - the attacker can only predict something

cache:

- cache is necessary: gap between CPU speed and latency of memory access, innermost cache access $\approx$0.3ns, main memory access $\approx$50ns to 150ns
- set-associative memory cache:
  - cache line (cache block) of B bytes
  - a row consisting of W cache lines
  - S cache sets, each consisting of a row
  - when a cache miss occurs, then a memory block is copied into one of cache lines evicting its previous contents
  - a memory block with address a can be cached only into the cache set with the index i such that $i = \lfloor a/B \rfloor \bmod S$ — **this is crucial for the attack**
- cache levels: slight complication to the attacks but differences of timing enable to recognize the situation

---

CASE STUDY: AES encryption

AES software implementation:

- particularly vulnerable because of its design
- AES defined in algebraic terms, but lookup table is typically faster
- there are arguments against alggebraic implementations as the execution time may provide a side channel
- key expansion: round zero: simply the key bytes directly, other rounds: key expansion reversable (details irrelevant for the attack)
- fast implementation based on lookup tables T_0, T_1, T_2, T_3 and $T_0^{(10)}, T_1^{(10)}, T_2^{(10)}, T_3^{(10)}$ for the last round (with no `MixColumns`)
- round operation

$$\left(x_0^{(r+1)}, x_1^{(r+1)}, x_2^{(r+1)}, x_3^{(r+1)}\right) := T_0(x_0^r) \oplus T_1(x_5^r) \oplus T_2(x_{10}^r) \oplus T_3(x_{15}^r) \oplus K_0^{(r+1)}$$

$$\left(x_4^{(r+1)}, x_5^{(r+1)}, x_6^{(r+1)}, x_7^{(r+1)}\right) := T_0(x_4^r) \oplus T_1(x_9^r) \oplus T_2(x_{14}^r) \oplus T_3(x_3^r) \oplus K_1^{(r+1)}$$

$$\left(x_8^{(r+1)}, x_9^{(r+1)}, x_{10}^{(r+1)}, x_{11}^{(r+1)}\right) := T_0(x_8^r) \oplus T_1(x_{13}^r) \oplus T_2(x_2^r) \oplus T_3(x_7^r) \oplus K_2^{(r+1)}$$

$$\left(x_{12}^{(r+1)}, x_{13}^{(r+1)}, x_{14}^{(r+1)}, x_{15}^{(r+1)}\right) := T_0(x_{12}^r) \oplus T_1(x_1^r) \oplus T_2(x_6^r) \oplus T_3(x_{11}^r) \oplus K_3^{(r+1)}$$

attack notation:

- $\delta = B/$entrysize of lookup table, typically: entrysize=4bytes, $\delta = 16$, (so δ entries of a lookup table are within the same cache line – this is a complication for the attack!)

- for a byte y let $\langle y\rangle = \lfloor y/\delta\rfloor$, it indicates a memory block of y in T_l
- if $\langle y\rangle = \langle z\rangle$ then x and y correspond to requests to the same memory block of the lookup table and therefore to the same cache line
- $Q_k(p,l,y) = 1$ iff AES encryption of plaintext p under key K accesses memory block of index y in T_l at least once in 10 rounds
- $M_k(p,l,y)$ a measurement that has expected value bigger in case when $Q_k(p,l,y) = 1$ then in case when $Q_k(p,l,y) = 0$

"synchronous attack"

- plaintext random but known, one can trigger encryption (e.g. for VPN with unknown key, dm-crypt of Linux)
- phase 1: measurements, phase 2: analysis
- from experiments: AES key recovered using 65 ms of measurements (800 writes) and 3 sec analysis
- **round-one attack:** the first round attacked
  i. accessed indices are simply $x_i^{(0)} = p_i \oplus k_i$ for $i = 0, \ldots, 15$
  ii. finding information $\langle k_i\rangle$ of k_i – test candidates $\bar{k_i}$
  iii. if $\langle k_i\rangle = \langle \bar{k_i}\rangle$ and $\langle y\rangle = \langle p_i \oplus \bar{k_i}\rangle$, then $Q_k(p,l,y) = 1$ for the lookup $T_l\left(x_i^{(0)}\right)$
  iv. if $\langle k_i\rangle \neq \langle \bar{k_i}\rangle$, then there is no lookup in block y for T_l during the first round, but
     - there are $4 \cdot 9 - 1 = 35$ other accesses affected by other plaintext bits during the entire encryption (4 per round, 9 rounds in total as the last round uses different look-up tables)
     - probability that none of them accesses block y for T_l is $\left(1 - \frac{\delta}{256}\right)^{35} \approx 0.104$ for $\delta = 16$
  v. few dozens of samples required to find a right candidate for $\langle \bar{k_i}\rangle$
  vi. together we determine $\log(256/\delta) = 4$ bits of each byte of the key
  vii. no more possible for the first round, not enough to start brute force (still 64 bits to be found!)
  viii. in reality more samples needed due to noise in measurements $M_k(p, l, y)$ and not $Q_k(p,l,y)$
- **two-round attack:** the second round attack because of the missing bits
  i. exploiting equations derived from Rijndeal specification:

     $x_2^{(1)} = s(p_0 \oplus k_0) \oplus s(p_5 \oplus k_5) \oplus 2\bullet s(p_{10} \oplus k_{10}) \oplus 3\bullet s(p_{15} \oplus k_{15}) \oplus s(k_{15}) \oplus k_2$

     $x_5^{(1)} = s(p_4 \oplus k_4) \oplus 2\bullet s(p_9 \oplus k_9) \oplus 3\bullet s(p_{14} \oplus k_{14}) \oplus s(p_3 \oplus k_3) \oplus s(k_{14}) \oplus k_1 \oplus k_5$

$x_8^{(1)} = ...$

$x_{15}^{(1)} = ...$

where s stands for the Rijndael Sbox, and $\bullet$ means multiplication in the field with 256 elements

ii. lookup for $T_2\left(x_2^{(1)}\right)$:

  - $\langle k_0\rangle, \langle k_5\rangle, \langle k_{10}\rangle, \langle k_{15}\rangle, \langle k_2\rangle$ already known
  - low level bits of $\langle k_2\rangle$ influence only low bits of $x_2^{(1)}$ so not important for cache access pattern
  - the upper bits of $x_2^{(1)}$ can be determined after guessing low bits of k_0, k_5, k_{10}, k_{15}: there are δ^4 possibilities ($=16^4$)
  - a correct guess yields a lookup in the right place
  - an incorrect guess: some $k_i \neq \bar{k_i}$ so

    $x_2^{(1)} \oplus \bar{x}_2^{(1)} = c\bullet s(p_i \oplus k_i) \oplus c\bullet s(p_i \oplus \bar{k_i}) \oplus ...$

    (for c depending on i) where ... depends on different random plaintext bits and therefore random

    differential properties of AES studied for AES competition:

    $\Pr\left[c\bullet s(p_i \oplus k_i) \oplus c\bullet s(p_i \oplus \bar{k_i}) \neq z\right] > 1 - \left(1 - \frac{\delta}{256}\right)^3$

    so the false positive for lookup:

    - $\left(1 - \frac{\delta}{256}\right)^3$ for computing $T_2\left(x_2^{(1)}\right)$
    - $\left(1 - \frac{\delta}{256}\right)$ for computing each of the remaining T_2
    - together no access with pbb about $\left(1 - \frac{\delta}{256}\right)^{38}$
  - this yields about 2056 samples necessary to eliminate all wrong candidates
  - it has to repeated 3 more times to get other nibbles of key bytes

iii. optimization: guess $\Delta = k_i \oplus k_j$ and take $p_i \oplus p_j = \Delta$, then i.e. $s(p_0 \oplus k_0) \oplus s(p_5 \oplus k_5)$ cancels out and we have to guess less bits (4 instead of 8)

- **similar attack: last round** - created ciphertext must be known to the attacker, otherwise similar. Subkey from the last round learnt, but keyschedule is reversable

- **measurement: Evict+Time**

  i. procedure:

     1. trigger encryption of p
     2. evict: access memory addresses so that one cache set overwritten completely

        3. trigger encryption of p

    ii. in the evicted cache set one cache line from T_l

    iii. measure time: if long, then cache miss and the encryption refers to evicted δ positions from the lookup table

    iv. practical problem: triggering may invoke other activities and timing is not precise

- **measurement: Prime+Probe**

    i. procedure

        1. (prime) read A: a contiguous memory of the size of the cache – results in overwriting the entire cache

        2. trigger an encryption of p (partial eviction at places where lookup used)

        3. (probe:) read memory addresses of A that correspond to $M_k(p, l, y)$

    ii. easier: timing for probe suffice to check if encryption used a given cache set

- **complications in practice:**

    i. address of lookup tables in the memory - how they are loaded to the cache remains unknown – offset can be found by considering all offsets and then statistics for each offset (experiments show good results even on noisy environment)

    ii. hardware prefetcher may disturb the effects. Solution: read and write the addresses of A according to a pseudorandom permutation

- **practical experiments:** e.g. Athlon 64, no knowledge of adresses mapping, 8000 encryptions with Prime & Probe

  Linux dm-crypt (disk, filesystem, file encryption): with knowledge of addressing, 800 encryptions (65 ms), 3 seconds analysis, full AES key

- **extensions of the attack:**

    - on some platforms timing shows also position of the cache line (better resolution for one-round attack)

    - remote attacks (VPN, IPSec): with requests that trigger immediate response (situation yet unclear about practicality)

"asynchronous attrack"

- no knowledge of plaintext, no knowledge of ciphertext

- one-round attack

- based on frequency F of bytes in e.g. English texts, frequency score for each of $\frac{256}{\delta}$ blocks of length δ

- F is nonuniform: most bytes have high nibble equal to 6 (lowercase characters "a" through "o")

- find j such that j is particularly frequent indicates $j = 6 \oplus \langle k_i \rangle$ and shows $\langle k_i \rangle$
- complication: this frequency concerns at the same time k_0, k_5, k_{10}, k_{15} affecting T_0 so we learn 4 nibbles but not their actual allocation to k_0, k_5, k_{10}, k_{15}
- the number of bits learnt is roughly: $4 \cdot (4 \cdot 4 - \log 4!) \approx 4 \cdot (16 - 3.17) \approx 51$ bits
- experiment: OpenSSL, measurements 1 minute, 45.27 bits of information on the 128-bit key gathered

Bernstein's attack

- an alternative way of computing AES, algorithm applied in OpenSSL:
  - → two constant 256-byte tables: S and S'
  - → expanded to 1024-byte tables T_0, T_1, T_2, T_3

    $T_0[b] = (S'[b], S[b], S[b], S[b] \oplus S'[b])$

    $T_1[b] = (S[b] \oplus S'[b], S'[b], S[b], S[b])$

    ....
  - → AES works with 16-byte arrrays x and y, where x initialized with the key k, y initialized with $n \oplus k$, where n is the plaintext
  - → AES computation is modifications of x and y:
    - i. x viewed as (x_0, x_1, x_2, x_3) (4 bytes parts)
    - ii. $e := (S[x_3(1) \oplus 1], S[x_3(2)], S[x_3(3)], S[x_3(0)])$
    - iii. replace (x_0, x_1, x_2, x_3) with $(e \oplus x_0, e \oplus x_0 \oplus x_1, e \oplus x_0 \oplus x_1 \oplus x_2, e \oplus x_1 \oplus x_2 \oplus x_3)$
    - iv. modify y viewed as (y_0, y_1, y_2, y_3), replace it with

      $$(T_0[y_0[0]] \oplus T_1[y_1[1]] \oplus T_2[y_2[2]] \oplus T_3[y_3[3]] \oplus x_0,$$
      $$(T_0[y_1[0]] \oplus T_1[y_2[1]] \oplus T_2[y_3[2]] \oplus T_3[y_0[3]] \oplus x_1,$$
      $$(T_0[y_2[0]] \oplus T_1[y_3[1]] \oplus T_2[y_0[2]] \oplus T_3[y_1[3]] \oplus x_2,$$
      $$(T_0[y_3[0]] \oplus T_1[y_0[1]] \oplus T_2[y_1[2]] \oplus T_3[y_2[3]] \oplus x_3$$
    - v. the next round uses $\oplus 2$ instead of $\oplus 1$ for x, otherwise the same. Similar changes corresponding to rounds up to 9
    - vi. in round 10 use $S[], S[], S[], S[]$ instead of $T's$
    - vii. y is the final output
- it is embarassing how simple the attack is:
  - → it has been checked in practice that execution depends on $k[i] \oplus n[i]$ - which is a position in the table:
    - try many plaintexts
    - collect statistics for each byte for $n[i]$

    - the maximum occurs for z
    - the maximum corresponds to a fixed value for $k[i] \oplus n[13]$, say c
    - compute $k[13] = c \oplus z$

- $\rightarrow$ for different bytes different statistics observed: for some t a few values $k[t] \oplus$ plaintext$[t]$, where substantially higher time observed
- $\rightarrow$ statistic gathered, different packet lengths
- $\rightarrow$ finally brute force checking all possibilites, nonce encrypted with the server key

Countermeasures

- ”no reliable and practical countermeasure” so far
- implementation based on no-lookup but algebraic algorithm (slow!!!) or bitslice implementation (sometimes possible and nearly as efficient as lookup)
- alternative lookup tables: if smaller than less date leaks (but for cryptanalysis bigger Sboxes increase security)
- data-independent access to memory blocks - every lookup causes a redundant read in all memory blocks, generally: oblivious computation possible theoretically but overhead makes it inpractical
- masking operations: ≈“we are not aware of any method that helps to resist our attack”
- cache state normalization: load all lookup tables - equires deep changes in OS and reduces efficiency, even then LRU cache policy may leak information which part has been used!
- process blocking: again, deep changes in OS
- disable cache sharing: deep degradation of performance
- ”no-fill” mode during encryption:
    - preload lookup tables
    - activate “no-fill”
    - encrypt
    - deactivate “no-fill”

  the first two steps critical and no other process is allowed to run

  possible only in priviledged mode, cost of operation prohibitive
- dynamic table storage: e.g. many copies of each table, or permute tables

  details architecture dependent and might be costly
- hiding timing information: adding random values to timing makes the statistical analysis harder but still feasible

- try to protect some rounds (the first 2 and the last one) with any mean – but may be there are other attack techniques...
- cryptographic services at system level: good but unflexible
- sensitive status for user processes: erasing all data when interrupt
- specialized hardware support: seems to be the best choice

  but the problem is not limited to AES or crypto – many sensitive data operations are not cryptographic and a coprocessor does not help

Meltdown – a similar attack on modern processors

- out-of-order execution
- kernel, checking access rights
- core instruction sequence

  1; rcx = kernel address

  2; rbx = probe array

  3 retry:

  4 mov al, byte [rcx]

  5 shl rax, 0xc

  6 jz retry

  7 mov rbx, qword [rbx + rax]

The idea is that:

– the instruction 4 leads to violation of access rights and consequently it will be interrupted, with temporary values erased

– in the meantime instructions 5-7 might be executed in advance, all results deleted – except that the value from rbx has appeared in the cache and it is still there

– before executing the code above the rbx is evicted from the cache, after execution the whole rbx is fetched, for one page the time is shorter . This occurs for the value of rax, which is the kernel address times 4096
