Security and Cryptography 2019

Mirosław Kutyłowski

grading criteria:

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

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

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

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

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

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

I. ACCESS CONTROL

ACCESS CONTROL (AC)

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

  - access exactly when needed

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

- practical problems:

  - diversity of application scenarios, no unique generalizations

  - administrator/system provider/product provider may be malicious

  - safety issues (access control to SCADA systems for industrial installations, ...)

  - constraints must respect future states

  - designing a system much harder: hardware, software design are much easier to test, in access control one cannot create a test environment

  - cultural differences (the users may react differently, this may lead to severe misunderstanding the situation by the AC designer)

Formal description

- goal: define secure and insecure states, secure state is such that the principals get access to objects as claimed by some predefined axiomatic properties,

- (there might be some gray zone where the state is neither secure nor insecure)

- AC defines rules define a dynamic system (changing access rights by principals are possible)

- question: given: a secure system with some access rights

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

Theorem – Undecidability

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

Proof idea

- a Turing machine can be regarded as an access control system:

  - a head may be treated as a principal

  - the tape represents objects (each entry is a data)

  - transition function of TM specifies what is allowed concerning data and security status

- ask if a Turing machine can reach some state+tape contents (which is declared as insecure)

- the problem is in general undecidable

□

Corollary

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

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

Main Models

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

AM

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

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

problems of AM:

i. lack of scalability, no flexibility,
ii. enormous effort, unless an AC system is tiny
iii. central administration, single point of failure (attack the administrator to get unrestricted control over the system)

ACL

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

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

advantages of ACL:

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

problems of ACL:

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

so ACL is not much useful for large and complicated systems

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

RBAC

- organization:
    - *subjects* assigned to *roles*
    - *permissions* assigned to *roles*
- a textbook example: a hospital system with differents roles: doctor, nurse, patient, accounting staff, family members of a patient, insurance company controller, ...

    i. a patient may read own record but not write or erase it
    ii. a doctor can write/read a record (but not erase)
    iii. a nurse can write records specifying activities, but not diagnosis part (reserved for the doctor's role)
    iv. ...

    But: even in this example we get into troubles: how to meet the requirements of GDPR: how to formalize "patient's own medical record"? Impossible with roles only.

- RBAC1:
    - hierarchy of roles (a directed acyclic graph)
    - a role gets all permissions from its lower roles in the hierarchy,

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

    disadvantages: expressive power the same as for RBAC

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

LATTICE models

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

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

Denning's Lattice Model

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

Explicit and implicit data flows

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

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

Security rules

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

Binding to levels

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

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

Decentralized Label Model

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

Chinese Wall (the name is misleading)

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

- an axiomatized approach enabling formal verification is possible

AC for group of principals

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

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

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

ABAC

recommended architecture:

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

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

- major problems:
    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

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

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

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

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

     $$e_{\mathrm{PICC}}=\mathrm{EM}(K,r_{\mathrm{PICC}},r_{\mathrm{PCD}},K_{\mathrm{PICC}})$$
  4. The terminal decrypts and verifies r_{PCD}
  5. both sides derive $K_{\mathrm{Enc}},K_{\mathrm{Mac}}$ from the master key

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

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

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

1. Introduction

aimed for customers looking for proper products, overview

1.1 PP reference

basic data, registration data

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

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

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

General Status: Final

Version Number: 1.10

Registration: BSI-CC-PP-0055

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

1.2 TOE Overview

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

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

- Usage and security features

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

- life cycle

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

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

2. Conformance Claim

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

EAL packages:

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

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

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

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

- levels:

  - EAL1: Functionally Tested:

    - correct operation, no serious threats

    - minimal effort from the manufacturer

  - EAL2: Structurally Tested

    - delivery of design information and test results,

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

  - EAL3: Methodically Tested and Checked

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

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

  - EAL4: Methodically Designed, Tested and Reviewed

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

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

  - EAL5: Semiformally Designed and Tested

  - EAL6: Semiformally Verified Design and Tested

  - EAL7: Formally Verified Design and Tested

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

- → tests
    - ATE_COV 1 2 2 2 3 3 Coverage
    - ATE_DPT 1 1 3 3 4 Depth
    - ATE_FUN 1 1 1 1 2 2 Functional tests
    - ATE_IND 1 2 2 2 2 2 3 Independent testing
- → vulnerability assesment
    - AVA_VAN 1 2 2 3 4 5 5 Vulnerability analysis

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

  - *The procedures for processing reported security flaws shall provide safeguards that any corrections to these security flaws do not introduce any new flaws.*

  - *The flaw remediation guidance shall describe a means by which TOE users report to the developer any susp ected security flaws in the TOE.*

- ALC_FLR.3:

  - first 5 as before

  - *The flaw remediation procedures shall include a procedure requiring timely response and the automatic distribution of security flaw reports and the associated corrections to registered users who might be affected by the security flaw.*

  - next 3 as before

  - *The flaw remediation guidance shall describe a means by which TOE users may register with the developer, to be eligible to receive security flaw reports and corrections.*

  - *The flaw remediation guidance shall iden tify the specific points of contact for all reports and enquiries about security issues involving the TOE.*

CEM -Common Evaluation Methodology

- given CC documentation, EAL classification etc, perform a check

- idea: evaluation by non-experts, semi-automated, mainly paper work

- mapping:

  - assurance class ⇒ activity

  - assurance component ⇒ sub-activity

  - evaluator action element ⇒ 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).

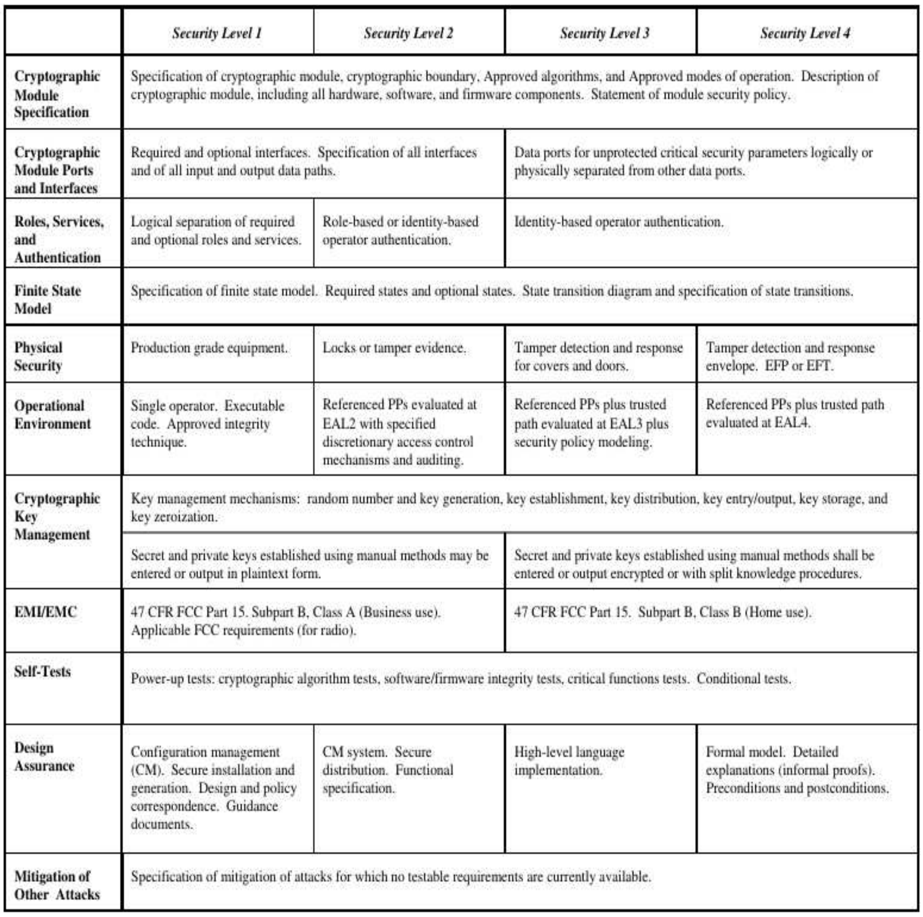
III. FIPS:

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:

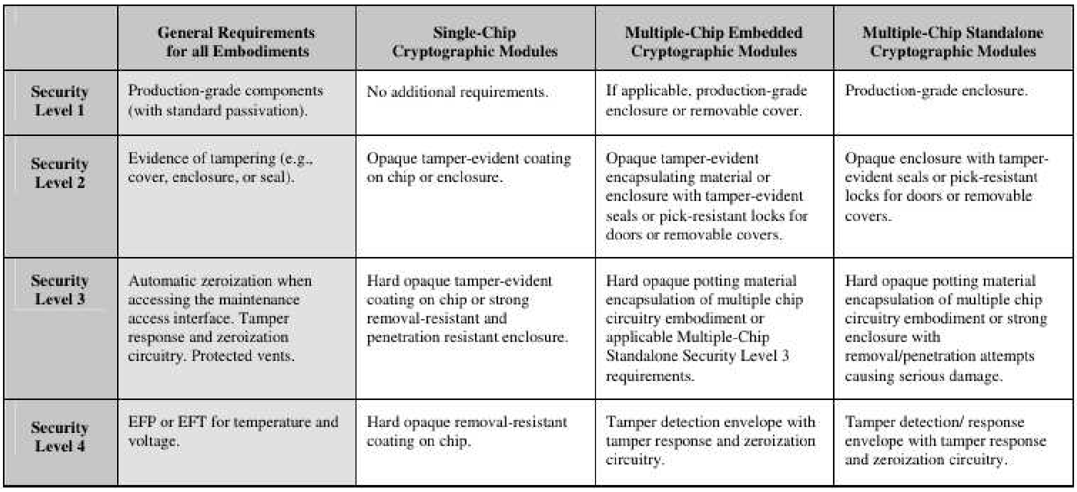


	Security Level 1	*Security Level 2*	*Security Level 3*	*Security Level 4*
Cryptographic Module Specification	Specification of cryptographic module, cryptographic boundary, Approved algorithms, and Approved modes of operation. Description of cryptographic module, including all hardware, software, and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

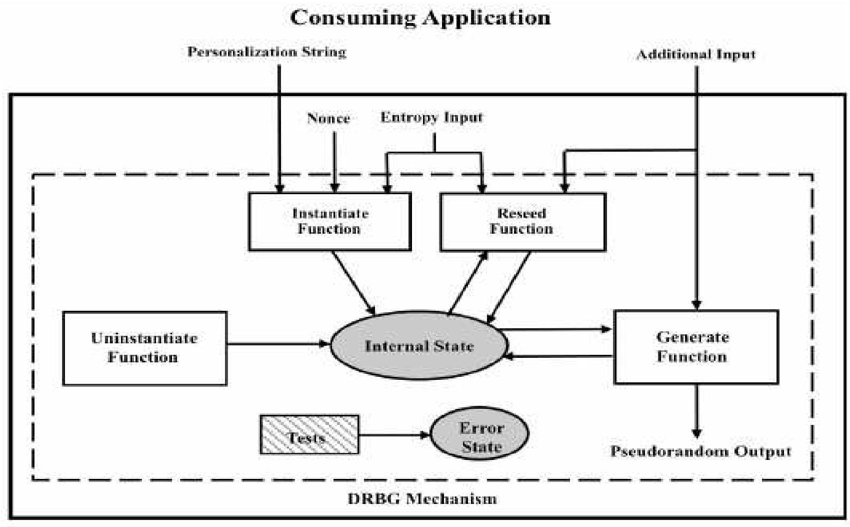
- key management:
    - non-approved RNG can be used for IV or as input to approved RNG
    - list of approved RNG: refers to an annex and annex to NIST document from 2016 (with a link to 2015)
    - list of approved key establishment methods - again links
    - key in/out: automated (encrypted) or manual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

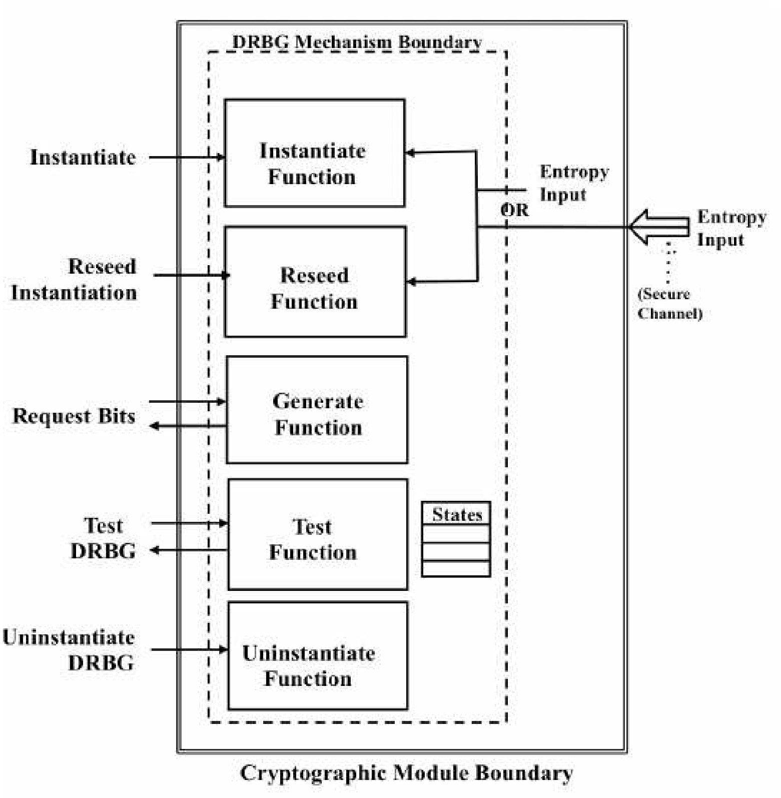
Functions executed:

- 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

variants:

– hash algorithms: SHA-1 up tp 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 $2^{\text{seedlen}}$
3. (returned_bits) = Hashgen (requested_number_of_bits, V)
4. H = Hash (0x03 || V)
5. V = (V + H + C + reseed_counter) mod $2^{\text{seedlen}}$
6. reseed_counter = reseed_counter + 1
7. Return (SUCCESS, returned_bits, V, C, reseed_counter)
```

Hashgen:

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

HMAC_DRBG

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

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

```
4. Key = HMAC (Key, V || 0x01 || provided_data)
5. V = HMAC (Key, V)
6. Return (Key, V)
```

Instantiate:

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

Reseed:

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

Generate bits:

```
1. If reseed_counter > reseed_interval, then return "reseed required"
2. If additional_input ≠ 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 $2^{\text{ctrlen}}$.
      2.1.2 V = leftmost (V, blocklen-ctr_len) || inc
   Else V = (V+1) mod $2^{\text{blocklen}}$
   2.2 output_block = Block_Encrypt (Key, V)
   2.3 temp = temp || output_block
3. temp = leftmost (temp, seedlen)
4.  temp = temp $\oplus$ provided_data
5. Key = leftmost (temp, keylen)
6. V = rightmost (temp, blocklen).
```

Instantiate:

```
1. pad personalization_string with zeroes
2.   seed_material = entropy_input $\oplus$ personalization_string
3. Key = $0^{\text{keylen}}$
4. V = $0^{\text{blocklen}}$
5. (Key, V) = CTR_DRBG_Update (seed_material, Key, V).
6. reseed_counter = 1
7. Return (V, Key, reseed_counter).
```

reseeding is similar

Generate:

```
1. If reseed_counter > reseed_interval, then “reseed required”
2. If (additional_input$\neq$ 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 = $0^{\text{seedlen}}$
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 $2^{\text{ctrlen}}$
      4.1.2 V = leftmost (V, blocklen-ctr_len) || inc
   Else V = (V+1) mod $2^{\text{blocklen}}$
   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. $R:=\mathrm{AES}_U(9)$

IV. 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.

V. GDPR

Some discussion from the lecture missing here

DOMAIN SIGNATURES

Definition of Domain Signatures

some discussion from the lecture missing here

Domain signatures and applications

1. replacing login+ password mechanisms. Like Restricted Identification from German Personal Identity cards. Advantages
    1. one PIN per person
    2. high entropy, not predictable
    3. no problem for
2. anonymization in datasets
3. recommendations,
4. ”randomness” and self-organization

Pseudonymous Signature:

- BSI standard
- 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 (called manager)
    - for a token the manager 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 $I^{\mathrm{sector}}_{\mathrm{ICC},1}$ and $I^{\mathrm{sector}}_{\mathrm{ICC},2}$ of the eIDAS token in the sector: $I^{\mathrm{sector}}_{\mathrm{ICC},1} = (\mathrm{PK}_{\mathrm{sector}})^{\mathrm{SK}_{\mathrm{ICC},1}}$ and $I^{\mathrm{sector}}_{\mathrm{ICC},2} = (\mathrm{PK}_{\mathrm{sector}})^{\mathrm{SK}_{\mathrm{ICC},2}}$
- signing: with keys $\mathrm{SK}_{\mathrm{ICC},1}$, $\mathrm{SK}_{\mathrm{ICC},2}$ and $I^{\mathrm{sector}}_{\mathrm{ICC},1}$ and $I^{\mathrm{sector}}_{\mathrm{ICC},2}$ 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. $c = \mathrm{Hash}(Q_1, I^{\mathrm{sector}}_{\mathrm{ICC},1}, A_1, I^{\mathrm{sector}}_{\mathrm{ICC},2}, A_2, \mathrm{PK}_{\mathrm{sector}}, m)$

    (variant parameters and $\mathbf{\Pi}$ 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

  - $Q_1 = (\mathrm{PK}_{\mathrm{ICC}})^c \cdot g^{s_1} \cdot (\mathrm{PK}_M)^{s_2}$
  - $A_1 = (I_{\mathrm{ICC},1}^{\mathrm{sector}})^c \cdot (\mathrm{PK}_{\mathrm{sector}})^{s_1}$
  - $A_2 = (I_{\mathrm{ICC},2}^{\mathrm{sector}})^c \cdot (\mathrm{PK}_{\mathrm{sector}})^{s_2}$
  - 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 $I_{\mathrm{ICC},1}^{\mathrm{sector}}, I_{\mathrm{ICC},2}^{\mathrm{sector}}$

- 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
    - domain trapdoor: $T_{\mathrm{domain},u} = \mathrm{PK}_{\mathrm{domain}}^{x_u}$ and s_u
    - then one can conclude that nym_1 and nym_2 correspond to user u, if:

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

VI. STANDARS VERSUS SECURITY

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

Standardization process:

- representatives of countries, not necessarily specialists
- strong representation of interests of industry
- target: a unified solution
- no open evaluation as in case of e.g. NIST competitions
- long process, many standards never used in practice

Example: ANSI X9.31 PRG

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

Algorithm

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

  – K used for the lifetime of the device

  – V changes

→ **next:**

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

Problems with seeding:

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

  → certification documentation may skip the problem of generating K

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

  and there is no reason to refuse certificate

- attack 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 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 padd the hash value before applying RSA signing operation, after padding the input 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
  - ○ ASN.1 data
  - ○ hash

    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:
  - hash not right adjusted (padding 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 root ... and this is the signature!
- attack variants: some fields declared but not checked. then Bleichenbacher's freedom to adjust the number to become a cube even if the hash is 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 not in 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.
     - observe $c_0^e = m \cdot s$ and it starts with msb: 00, 02,

- let k the byte length of n, $B = 2^{8(k-2)}$
- then $2B \leq m \cdot s < 3B$
- 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

   ○ 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

   ○ 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

   ○ M_i consists of all intervals

   $[\max\left(a, \frac{2B + r_i \cdot n}{s_i}\right), \min\left(b, \frac{3B - 1 + r_i \cdot n}{s_i}\right)]$ for $[a, b]$ 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 set $m = a \cdot s_0^{-1} \bmod n$

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

$$a \cdot s_i - (3B - 1) \leq r \cdot n \leq b \cdot s_i - 2B$$

- on the other hand

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

The attack exploits such vulnerabilities like

- error messages returned by the attacked device when decryption fails on different stages of the decryption algorithm
- 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).

VII. RFC DOCUMENTS (actual lecture given by dr Kubiak could be different)

RFC

”Request for Comments”

- by Internet Engineering Task Force (IETF) and the Internet Society
- semi-standard, developed from rfc from ARPANET
- authors of RFC versus standards with commitees
- peer review, some reach status of “Internet Standards”
- RFC editor provided
- streams:
  - Internet Engineering Task Force (IETF) - current issues
    - BCP Best Current Practice;
    - FYI For Your Information; informational
    - STD Standard: with 2 maturity levels
  - Internet Research Task Force (IRTF) - more long term issues
  - Internet Architecture Board (IAB) (a body over task forces)
  - independent
- Status:
  - informational
  - experimental
  - best current practice
  - standard: Proposed Standard, Draft 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 very nice 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*."

VIII. HARDWARE TROJANS

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)

- optical inspection (distructive) - can detect modifications on layout level, costly

Idea: change properties that are not visible under microscope: increase aging effects, manipulate transistors so that the output is fixed or has other different characteristics

Dopant Trojans

CMOS inverter: (image Wikipedia)

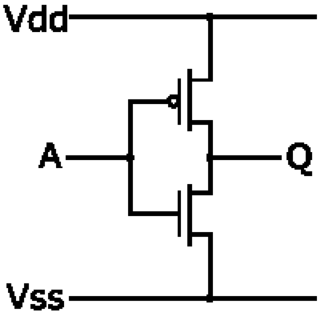


Figure 1.

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

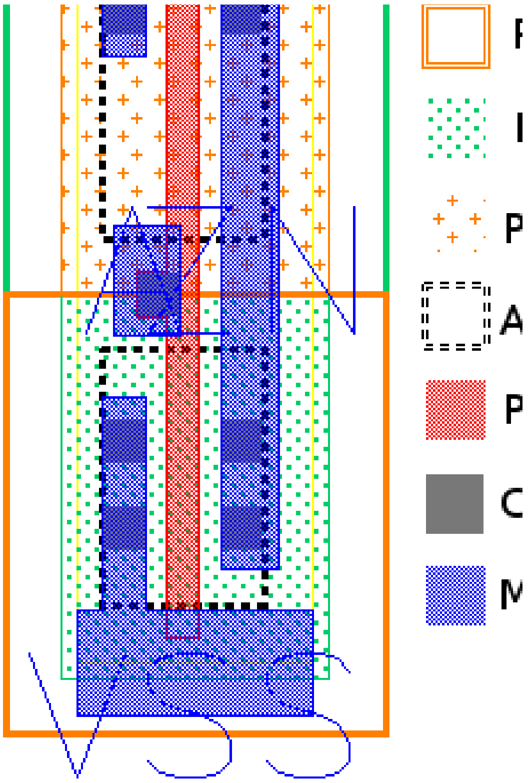
PMOS: in dopant area "holes" (positive) playing the role of conductor, low voltage creates depletion area, high voltage attracts them

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

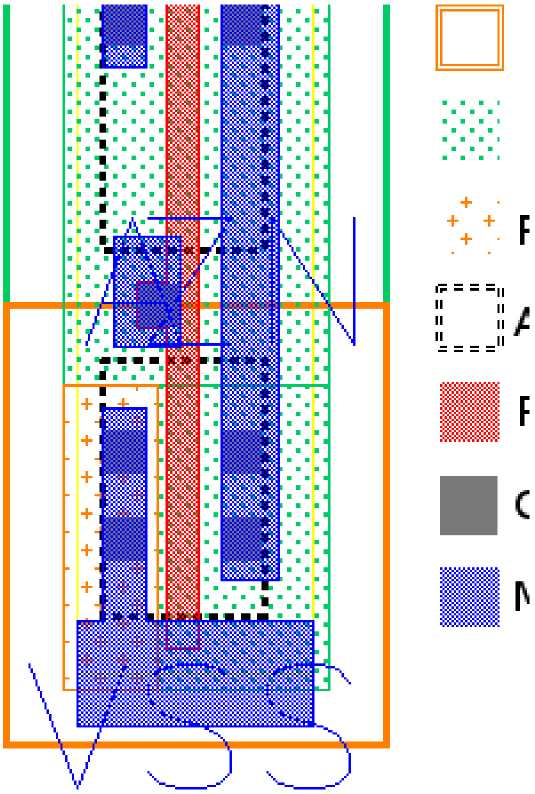
For physical realization of a transistor see excellent videos from

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

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

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. $c := c + 1$, $x := \mathrm{AES}_K(c)$
    2. $c := c + 1$, $y := \mathrm{AES}_K(c)$
    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. againt randomized signature schemes)
- problem with OHT: tests with some values have to create known outputs (32 CRC from the last 4 outputs), knowing the test one can find K and c by exaustive search

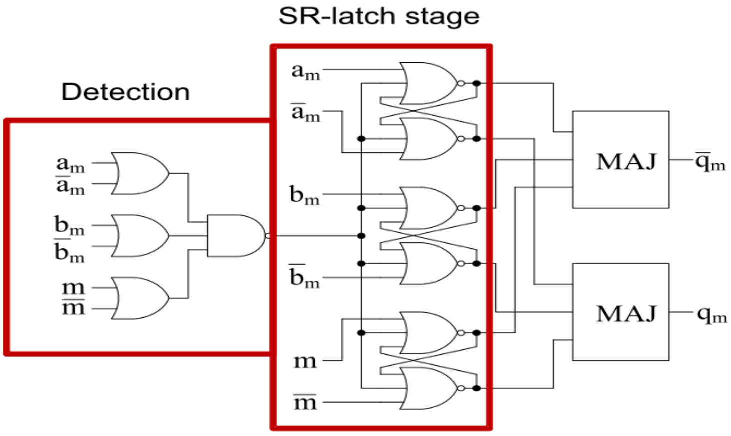
Side channel Trojan:

- 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 $a \wedge b$:

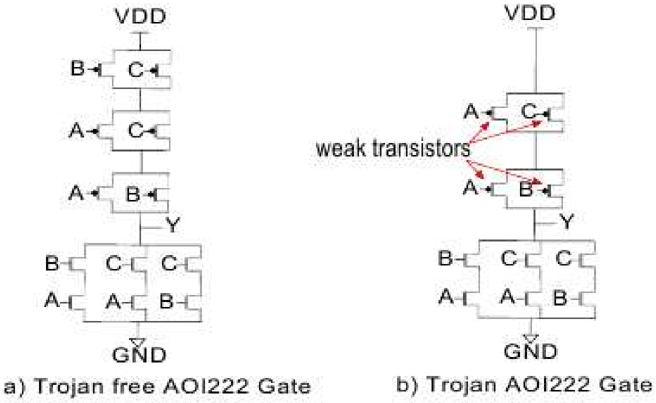
- input $a \oplus m$, $a \oplus \neg m$, $b \oplus m$, $b \oplus \neg m$, m, $\neg m$
- detection, SR-latch stage and majority gate



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:



Idea: instead of cutting output a low voltage

- 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 stornger 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

IX. 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 (1000 EUR and more)

- 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. Bob measures the photons using basis indicated by $\hat{b_i}$ for the ith photon
  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 × 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 $|0\rangle$!!!!!!
  - 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 sattelite (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 $\mathrm{GCD}(n, r-1)$, 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 $\mathrm{GCD}(n, r-1)$ 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
  - 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:

    $|j\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{(2\pi i \cdot j \cdot k)/N} \cdot |k\rangle$

  „efficient implementation” 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

$$|a, 0\rangle \rightarrow |a, x^a \bmod n\rangle$$

   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

$$\frac{1}{\sqrt{M}} \cdot \sum_{a \in A}^{M-1} |a, k\rangle$$

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

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

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

    - 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 ($\frac{\text{p-1}}{2}$ 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
- always pad something
- 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, ...,x_N in the CBC mode with IV (fixed or random) and a block cipher Enc:

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

- 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:
  - → reaction “reject”: creates padding oracle (attacker tests the behavior)
  - → reaction “proceed”: enables manipulation of the plaintext data

last word oracle:

- goal: compute Dec(y) for a block y
- create an input for padding oracle:
  - create a 2 block ciphertext: $r = r_1 ... r_b$ chosen at random, $c := r|y$
  - oracle call: if Oracle(c) = valid, then Dec(y) $\otimes$ r should yield a correct padding. whp this happens if $y_b = r_b \oplus 1$ (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 $r_1, r_2 ..., r_b$ at random, take $i = 0$
    2. put $r = r_1 r_2 ... r_{b-1} (r_b \oplus i)$

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:

   $r := r_1 ... r_{b-j}(r_{b-j+1} \oplus 1)r_{b-j} ... r_b$

   /* attempting to disturb padding, from left to right

   ask padding oracle for $r|y$, if “invalid” then output $(r_{b-j+1} \oplus j) ... (r_b \oplus j)$ 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 $a_1 ... a_b$ be the plaintext of y

decryption:

- get a_b via the last word oracle
- proceed step by step learning a_{j-1} once $a_j, ..., a_b$ are already known

  1. set $r_k := a_k \oplus (b - j + 2)$ for $k = j, ..., b$ /* preparing the values so that the padding values $(b - j + 2)$ appear at the end)
  2. set $r_1, ..., r_{j-1}$ at random, $i := 0$ /* search for the value that makes a proper padding
  3. $r := r_1 ... r_{j-2}(r_{j-1} \oplus i)r_j ... r_b$
  4. if $O(r|y) = \text{invalid}$, 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 expct on a givn positio of the pading) :

- $00....0n$ instead of $nn....n$

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

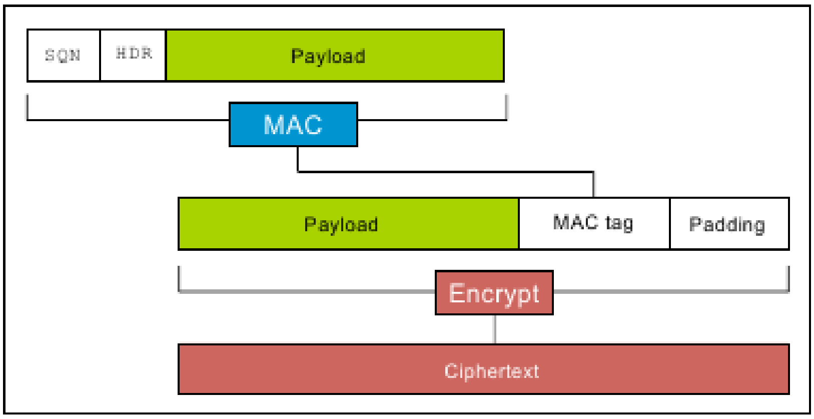
the padding where it would not work is a one with random data on the added positions

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

- MAC applied before padding, so 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 means 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:

  $T\colon = H((K_a \oplus \text{opad})||H((K_a \oplus \text{ipad})||M))$

- **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 to check if the padding has the length bigger than 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 the plaintext $C_{i-1} \oplus C_{p-1} \oplus 0...0g$

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

7. 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 $P_0...P_{15}$

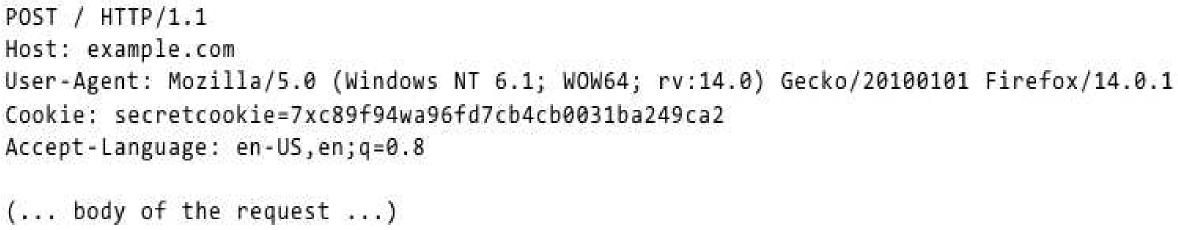
protection: browser must be carefully designed and do not admit injecting plaintexts (SOP- Same Origin Protection). Some products do not implement it.

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:

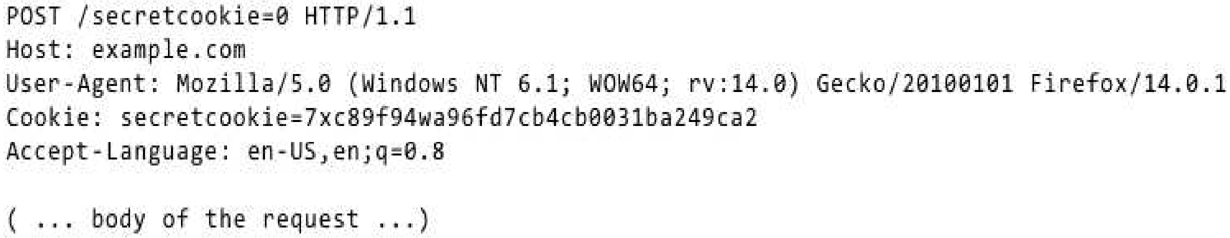


```
POST / HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:14.0) Gecko/20100101 Firefox/14.0.1
Cookie: secretcookie=7xc89f94wa96fd7cb4cb0031ba249ca2
Accept-Language: en-US,en;q=0.8

(... 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

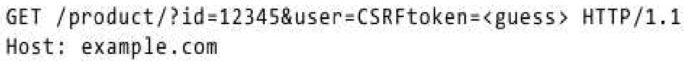
- again based on compression but now on the server side (from the client to the server compression might be disabled and CRIME fails)

- 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 two or one packets – 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

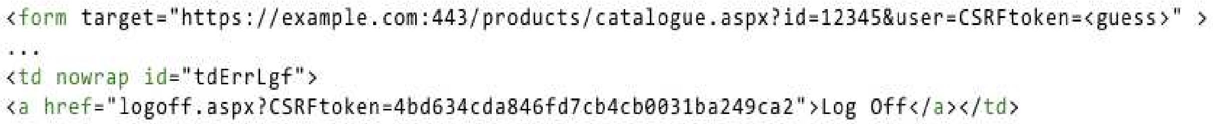
Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext

- 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 PRF 65 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 ⇒ 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.

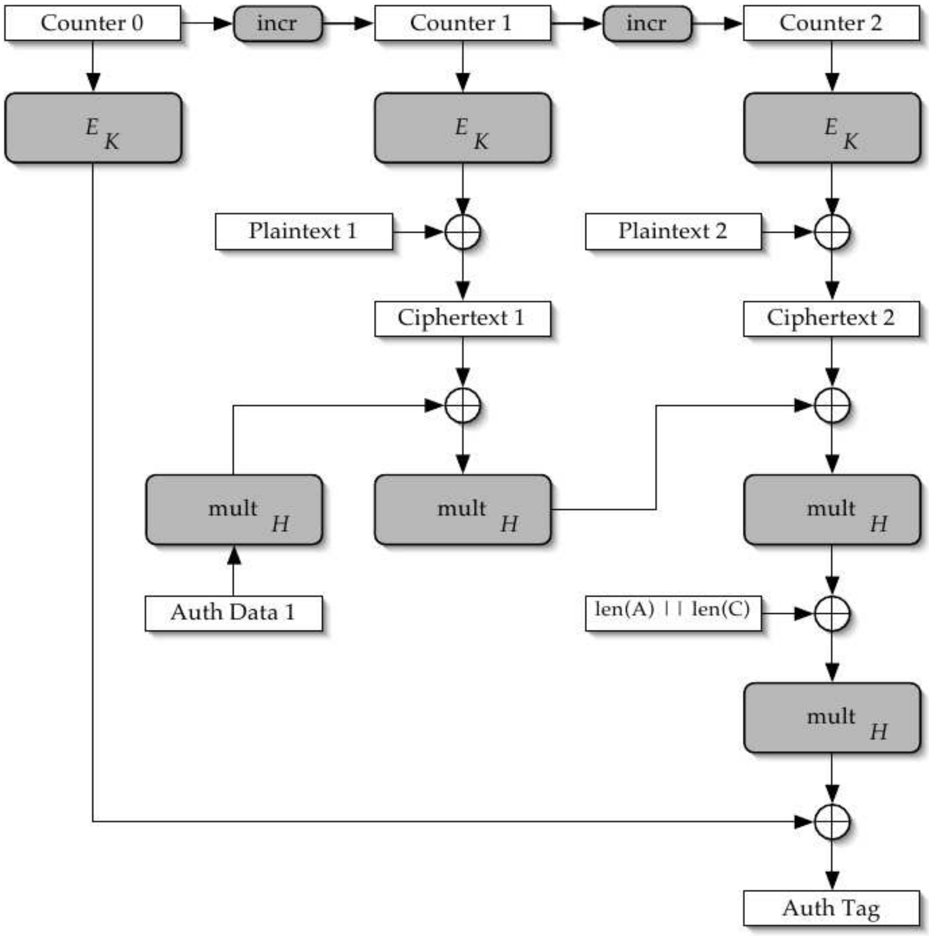
GCM (The Galois/Counter Mode)

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:

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

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), lookup tables
- 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 $\mathrm{GF}(2)$ with coefficients $X_0, X_1, \ldots., X_{127}$
- multiplication by a constant D: $X \rightarrow D \cdot X$ 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 might be non-zero, so the chance to 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
- variant of SALSA from European ECRYPT competition
- 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. $a = a + b$; $d = d \,\mathrm{xor}\, a$; $d = d \lll 16$
  2. $c = c + d$; $b = b \,\mathrm{xor}\, c$; $b = b \lll 12$
  3. $a = a + b$; $d = d \,\mathrm{xor}\, a$; $d = d \lll 8$
  4. $c = c + d$; $b = b \,\mathrm{xor}\, c$; $b = b \lll 7$
- 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

- 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 reuse safely with a different encryption scheme
- the only way to break Poly is to break AES
- low per message overhead (even for short messages)
- no long lookup tables, therefore fit 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, result: $c_0, c_1, ..., c_q$
- 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 messages m, m' of length L pbb that $H_r(m) = H_r(m') + g$ is at most $8\lceil L/16 \rceil / 2^{108}$ – small probabilities of differentials

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 rond-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 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 Hash(M_0) = 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 make WPAD (Web Proxy Auto-Discovery Protocol) requests

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

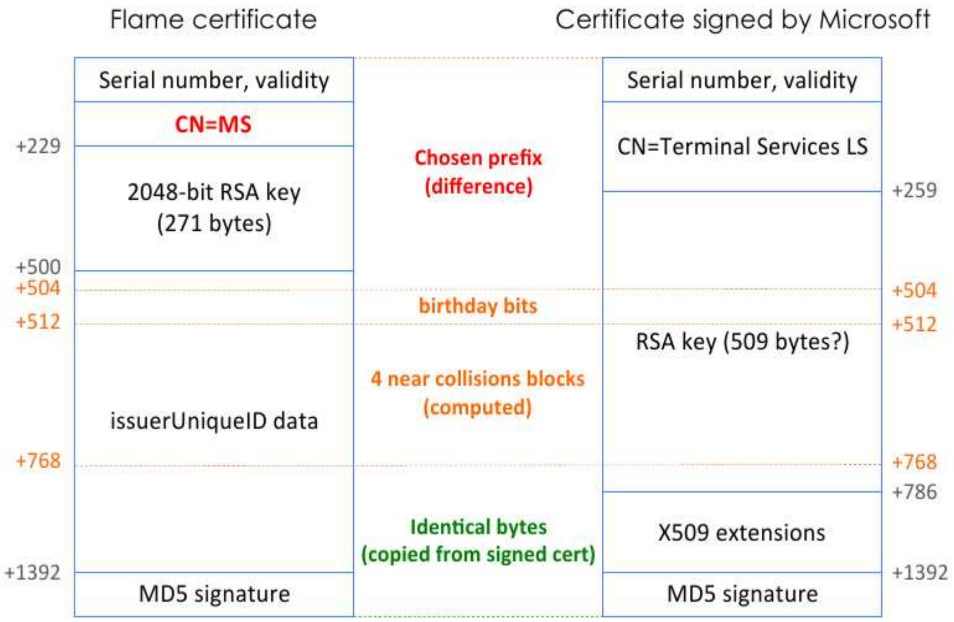
  - victim that gets wpad.dat sets its proxy server to a Flame computer (later no sniffing necessary!)

- Windows updates provided by the Flame computer. The main problem is that the updates must be properly signed!

- 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 ≈0.3ns, main memory access ≈50ns to 150ns
- set-associative memory cache:
  - cache line of B bytes
  - a row consisting of W cache lines
  - S cache sets, each consisting of a line
  - 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 $i = \lfloor a/B \rfloor$ — **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:

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

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

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

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

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

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

  (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. adress 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

    $$\begin{gathered}(T_0[y_0[0]] \oplus T_1[y_1[1]] \oplus T_2[y_2[2]] \oplus T_3[y_3[3]] \oplus x_0,\\ (T_0[y_1[0]] \oplus T_1[y_2[1]] \oplus T_2[y_3[2]] \oplus T_3[y_0[3]] \oplus x_1,\\ (T_0[y_2[0]] \oplus T_1[y_3[1]] \oplus T_2[y_0[2]] \oplus T_3[y_1[3]] \oplus x_2,\\ (T_0[y_3[0]] \oplus T_1[y_0[1]] \oplus T_2[y_1[2]] \oplus T_3[y_2[3]] \oplus x_3\end{gathered}$$

    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

- embarassing simple attack:

  → timing of execution depends on $k[i] \oplus n[i]$:

    - try many plaintexts

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

→ for different bytes different statistics observed: for some t a few values $k[t] \oplus$ plaintext$[t]$, where substantially higher time observed

→ statistic gathered, different packet lengths

→ 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

Spectre

-
  ```
  if (x<array1_size)
  y= array2[aray1[x]*4096];
  ```

Meltdown – 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]
  ```

XIII. DISK ENCRYPTION

Problems

- sector size (512, 4096 bytes versus blocksize of encryption)

- some mode to be used: ECB obviously wrong

  CBC and other modes require IV
- different IV or tweaking inside each sector
- no extra space in a sector

Malleability

CBC: if plaintext known then one can change every second block to desired value (every second would be junk):

- $C_i = E_K(C_{i-1} \oplus P_i)$
- replace C_{i-1} with $C_{i-1} \oplus P_i \oplus P'$
- then the ith block decrypts to P' (while block P_{i-1} will become junk)

creating IV vectors (should not repeat!): Encrypted salt-sector initialization vector (ESSIV)

- $\mathrm{IV}_n = \mathrm{Enc}_K(n)$ where $K = \mathrm{Hash}(K_0)$ and n is the sector number
- the same or separate key

Encryption algorithms

- attempts to use sector- size block ciphers - not popular
- tweaking traditional block ciphers (tweakable narrow-block encryption)
- LRW Liskov, Rivest, and Wagner
  - $C = \mathrm{Enc}_K(P \oplus X) \oplus X$
  - $X = F \otimes I$
  - F is the additional key, I is the index of the block
  - the issue of "red herrings": encryption of block $F||0^n$

    $C_0 = \mathrm{Enc}_K(F \oplus F \otimes 0) + F \otimes 0 = \mathrm{Enc}_K(F)$

    $C_1 = \mathrm{Enc}_K(0 \oplus F \otimes 1) + F \otimes 1 = \mathrm{Enc}_K(F) \oplus F$
- Xor–encrypt–xor (XEX)
  - $X_J = \mathrm{Enc}_K(I) \otimes \alpha^J$
  - $C_J = \mathrm{Enc}_K(P \oplus X) \oplus X$
  - I is the sector number, J is the block numer in the sector and α is a generator
- XEX-based tweaked-codebook mode with ciphertext stealing (XTS)
  - IEEE P1619
  - different key for IV than for encryption ("through misunderstanding XEX specification")

  - deals with the sector size not divisible by the block size
  - for the last block
    1. expands the k byte plaintext with the last bytes of the ciphertext of the previous block,
    2. the resulting ciphertext stores in place of the ciphertext of the previous block
    3. the ciphertext from the previous block truncates to k bytes and stores as the last ciphertext

    for decryption: the missing $n-k$ bytes are recovered from decryption of the ciphertext of the last (originally) block
  - problem: no MAC, one can manipulate blocks, something will be recovered!

Generating key for disk encryption form the password

Password-Based Key Derivation Function 2 (PBDKF2)

- $\text{DerivedKey} = \text{PBDKF2}(\text{PRF}, \text{Password}, \text{Salt}, c, \text{dkLen})$
- c is the number of iterations requested
- $\text{DerivedKey} = T_1 || T_2 || \ldots || T_{\text{dklen/hlen}}$
- $T_i = F(\text{Password}, \text{Salt}, c, i)$
- $F(\text{Password}, \text{Salt}, c, i) = U_1 \,\text{xor}\, U_2 \,\text{xor}\, \ldots \text{xor}\, U_c$
- $U_1 = \text{PRF}(\text{Password}, \text{Salt}, i)$
- $U_j = \text{PRF}(\text{Password}, U_{j-1})$ for $1 < j < c$

slight problem: if password too long then first processed by hash, then some trivial collisions

password hashing competition

- organized by a group of people
- Argon2 winner
- some controversies
- design goals:
  - fills memory fast
  - tradeoff resilience (smaller area higher time - but potentially compensated by ASIC)
  - scalability of parameters
  - number of threads can be high
  - GPU/FPGA/ASIC unfirendly

    - optimized for current processors

Argon2

inputs:

- message P (up to $2^{31}-1$ bytes)
- nonce S (up to $2^{31}-1$ bytes)
- parameters: degree of parallelism p, tag length τ, memory size m from $8p$ to $2^{32}-1$ kB, number of iterations t, secret K (up to 32 bytes), associated data X (up to $2^{32}-1$ bytes)

extract-then-expand

- extract a 64 byte value: $H_0 = \text{Hash}(p, \tau, m, t, v, y, <P>, P, <S>, S, <K>, K, <X>, X)$ where $<A>$ denotes the length of A
- expand using a variable length hash H':
    - initialize blocks $B[i, j]$ with p rows ($i = 0, ..., p-1$) and $q = \lfloor\frac{m}{4p}\rfloor \cdot 4$ columns, each $B[i, j]$ of 1kB
    - $B[i, 0] = H'(H_0||0000||i)$
    - $B[i, 1] = H'(H_0||1111||i)$
    - $B[i, j] = G(B[i, j-1]||B[i', j'])$) where i', j' depend on the version

    t iterations:

    - $B[i, 0] = G(B[i, q-1], B[i, j])$
    - $B[i, j] = G(B[i, j-1], B[i, j])$

    final

    - $B_{\text{final}} = B[0, q-1] \oplus B[1, q-1] \oplus .... \oplus B[p-1, q-1]$
    - $\text{Tag} = H'(B_{\text{final}})$

- variable length hashing
    - H_x a hash function with output of length x
    - if $\tau \leq 64$, then $H'(X = H_\tau(\tau||X))$
    - if $\tau > 64$:

        $r = \lceil\tau/32\rceil - 2$

        $V_1 = H_{64}(\tau||X)$

        $V_2 = H_{64}(V_1)$

        ...

        $V_r = H_{64}(V_{r-1})$

$V_{r+1} = H_{\tau - 32r}(V_{r-1})$

$H'(X) = A_1 || A_2 || ... || A_r || V_{r+1}$

where A_i are the first 32 bits of V_i

- compression function G
  - Blake2b round function P used
  - $G(X, Y)$ on 1kB blocks X, Y:
    - $R = X \oplus Y$, R treated as 8×8 matrix of 16-byte registers $R_0, ..., R_{63}$
    - $(Q_0, ..., Q_7) = P(R_0, ..., R_7)$

      $(Q_8, ..., Q_{15}) = P(R_8, ..., R_{15})$

      ...

      $(Q_{56}, ..., Q_{63}) = P(R_{56}, ..., R_{63})$
    - $(Z_0, Z_8, ..., Z_{56}) = P(Q_0, Q_8, ..., Q_{56})$

      $(Z_1, Z_9...., Z_{57}) = P(Q_1, Q_9, ..., Q_{57})$

      ...

      $(Z_7, Z_{15}, ..., Z_{63}) = P(Q_7, Q_{15}, ..., Q_{63})$
    - finally output

      $Z \oplus R$

XIV. ANONYMITY IN COMMUNICATION

- who is communicating with whom is a sensitive data

protection methods:

- broadcast channel
- token ring
- dining cryptographers -DC nets
- onion protocols and TOR

DC-nets

- protection of the message source. One of cryptographers is sending a 0 or 1.

- protocol for 3 cryptographers sitting at a round table:

    1. each pair of neighbors establish a shared bit at random
    2. each cryptographer that is not transmitting computes XOR of the shared bits that they knows,
    3. the sender computes the same XOR but swaps it if the bit transmitted is 1
    4. each cryptographer reveals the computed result
    5. every cryptographer computes XOR of all bits published

Steganography

- hiding transmission in innocent data (images, sound)
- steganography versus watermarking: watermarking is not annoying but hard to remove, steganography is invisible
- procedure:

    1. original picture - stego image
    2. marking algorithm (maybe with a secret key) applied to message and stego image
    3. outcome transmitted/published
    4. retreiving the covered message

- invisibility property: without a key impossible to decide whether there is a message hidden
- typical applications: copyright protection, DRM,
- concrete techniques:

    i. changing lsb bits of gray scale

    ii. JPEG encoding: cosinus transform, high frequency components are manipulated anyway for compression

    iii. other digital transforms

    iv. interference watermark

    v. echoing

    vi. audio encoding: transformation and assigning coefficients to waves – manipulations of certain coefficients undetectable for listeners

    vii. watermarking network flow:

        - 1 based versus 0/1 based (in the first case the change is 1 no change encodes 0)
        - all random parameters transmitted in clear may contain watermarks
        - simple timing: interpacket delays, departure times

- mean balancing:
  - $2d$ probes divided into two sets A and B.
  - the expected values of A and B are the same with no watermarking
  - changing the characteristics so that expected values differ in some direction (the direction is the watermarked value)
- kinds of mean balancing watermarks:
  - interpacket delays
  - interval centroid: divide into $2d$ intervals, in each compute mean arrival time
  - interval counting: divide into $2d$ intervals, in each compute the number of packets
- size: packet size (hard if encryption...), object size (https, malicious Javascript generating watermarked size data)
- network rate:
  - influencing with dummy packets to change characteristics
  - burst traffic (against TOR)
  -

response times in transmission, packet order etc

- defense: (problematic or illegal due to intelectual property rights)

  i. compression

  ii. transforms and random distortions

  iii. streching (Stirmark)

  iv. printing and scanning, input to an analog device and digitalize again

- effectiveness measured by relative entropy:

$$D(P_1|P_2|) = \sum_{q \in \text{space}} \Pr_1(q) \cdot \frac{\Pr_1(q)}{\Pr_2(q)}$$

relative entropy of plaintexts and ciphertext should e smaller than some small ϵ

Onions

- limiting transmission overhead
- onions:
  - → an onion created for a route going through servers A, B, C, ..., Z to the final destination Γ

$O_1 = \mathrm{Enc}_A(B, \mathrm{Enc}_B(C, \mathrm{Enc}_C(...(\mathrm{Enc}_Z(\Gamma, M))....)))$

(by encryption Enc_X we mean encryption with the public key of server X)

→ O_1 sent from the origin machine to A,

→ server A decrypts with its private key and gets B and $O_2 = \mathrm{Enc}_B(C, \mathrm{Enc}_C(...(\mathrm{Enc}_Z(\Gamma, M))....))$

→ A sends O_2 to B

→ server B decrypts O_2 with its private key and gets C and $O_2 = \mathrm{Enc}_C(...(\mathrm{Enc}_Z(\Gamma, M))....)$

→ the process is continued in the same way until server Z finds Γ, M and forwards the message M to machine Γ

each processing steps is like peeling off one layer of an onion

- idea: if two or more onion enter a server at the same time, get partially decrypted, then forwarded, then it is impossible to say which incomming onion corresponds to which outcoming onion - **node mixing**
- traffic analysis: assigning probabilities to permutations ($\pi(i) = j$ means that the ith sender has a message to the jth receiver)
- it is not enough to say that $\pi(i) = j$ with ppb $\approx \frac{1}{n}$ - example:
  - let us assume that the adversary knows that π is a circular shift
  - the adversary gets extra knowledge: if source i is talking to destination j, then $i+1$ is talking to destination $j+1$. However, still the property $\pi(i) = j$ is fulfilled

- question: necessary length of the onions? restricted case of n senders and n receivers, messages sent simultaneously
  - $O(\log^2 n)$ if adversary has a full knowledge of the system (not likely to have a better estimation unless ... big progres in math), assumption: uniform distribution for choosing destinations
  - $O(\log n)$ if the adversary can see only a constant fraction of nodes - tight bound, assumption: sender i may have non-uniform distribution of destination points

meaning of the results: *traffic analysis does not improve our prior knowledge in a significant way* (e.g. if we know in advance that source i always sends to destination j, then onions cannot hide this fact

guarantees given in terms of total variation distance of two probability distributions:

$\|\pi, \mu\| = \frac{1}{2}\sum_{\omega \in \Omega} |\pi(\omega) - \mu(\omega)|$, where Ω is the set of all events

Problems with onions

- replay attacks: just send the same onion (or partially decrypted onion) for the second time. the same subonions will appear along the forwarding path defense: universal reencryption, example based on Elgamal:

- ciphertext of m:

  $(\beta^r, m \cdot g^r, \beta^s, g^s)$ for r, s chosen at random

- renecryption:

  1. choose α, β at random
  2. replace (y_1, y_2, y_3, y_4) by $\left(y_1 \cdot y_3{}^{\alpha}, y_2 \cdot y_4^{\alpha}, y_3^{\beta}, y_4^{\beta}\right)$

     thereby we get $(\beta^{r+\alpha s}, m \cdot g^{r+\alpha s}, \beta^{s\beta}, g^{s\beta})$

     if order of the group is a prime number then this is equivalent with choosing $(\beta^{r'}, m \cdot g^{r'}, \beta^{s'}, g^{s'})$ for random r', s'

- local view - not common lists of servers - long routes do not increase security:

  - give user A a list of servers with 50% of server used by nobody else
  - no matter how long is the routing path, it is likely that close to destination the path goes through a rogue server
  - a few destinations available from this rogue server (50% of cases the rogue one send directly to the destination)
  - an onion going through the rogue server originates from the attacked source

- timing at nodes - delays necessary

  defense: collecting enough onions and flashing them at once. (slowdown!!!)

- sparse traffic = no protection

TOR

- free BSD licence
- connection based protocol, new connection established periodically ("10 minutes or so")
- routes limited to 3 TOR nodes
- onion based forwarding the symmetric keys

  i. each node on the path learns only the predecessor and the successor

  ii. the path established step by step:

  - after establishing a subpath $X_0, X_1, \ldots., X_k$ the subpath is used to send an encrypted message over the channel to X_k stating that the next node is X_{k+1}.
  - the sender and X_{k+1} negotiate a new connection key via DH key exchange

  iii. after making a connection the message is encrypted symmetrically with the keys:

  $\mathrm{AES}_{\mathrm{relay1}}(\mathrm{AES}_{\mathrm{relay2}}(\mathrm{AES}_{\mathrm{relay3}}(m)))$

each relay node removes one layer

iv. the messages back to the sender: instead of decryption: encryption with keys shared with the sender. The sender has to decrypt the onion

problems:

- exit node knows the plaintext
- traffic correlation
- application level attacks
- Heartbleed - change of public keys, some clients use old keys, ...

XV. WIFI

standards:

- evolution
- little interaction with academic community
- underspecified,
- sometimes not literally implemented, lack of documentation
- sometimes formal security proofs – like for WPA, but nevertheless ... attacks

Learning from early mistakes: WEP

Attack thanks to CRC32

Attack through weaknesses of RC4

RC4 KSA (key Scheduling Algorithm) for key K

```
for i=0 to 255 do
    S[ i ] := i
end
j := 0
for i= 0 to 255 do
    j:= j+S[i]+K[i mod len(K)] mod 256
    swap(S, i , j )
end
i:=0, j:=0
```

RC4 PRNG, execute the following loop as long as output needed:

```
i:=i + 1 mod 256
j:= j + S[i] mod 256
swap(S, i , j )
```

```
return S[ S[ i ] + S[j] mod 256 ]
```

FMS Attack

- assumption: the adversary already knows the first l bytes of the key AND the first output byte of RC4 PRNG
  - so the adversary can perform the first l steps of Key Scheduling Algorithm
  - the goal will be to learn one more byte of the key (like for ChopChop attack)
- assumptions about the state of KSA after l steps for a given initial vector:
  - → $S_l[1] < l$
  - → $S_l[1] + S_l[S_l[1]] = l$
  - → $S_l^{-1}[X(0)] \neq 1, S_l[1]$
- validity of the assumption for a given initial vector can be checked by simulation of the first l steps
- **ASSUME THAT:**
  - $S_l[1], S_l[S_l[1]], S_{l+1}[l]$ did not participate in any swaps during the rest of the KSA
- **THEN**:
  - for the generation of the first output byte we take

$$S_{n+1}[\,S_{n+1}[1] + S_{n+1}[S_n[1]]\,]$$

  - if the assumption was ok then this is the same as

$$S_{l+1}[\,S_l[1] + S_l[S_l[1]]\,] = S_{l+1}[l] = S_l[j_{l+1}]$$

    the last equation follows from the fact that at the step $l+1$ there is a swap at positions l and j_{l+1}

  - from the output byte we derive the candidate for j_{l+1} (the position of the output byte in S_l)
  - on the other hand: $j_{l+1} = j_l + K[l] + S_l[l]$, so we may derive a candidate for $K[l]$
  - the first swap of PRNG will swap $S[1]$ and $S[S[1]]$ and this does not change the value of $S[1] + S[S[1]]$, HOWEVER the value would be affected by the swap if $S[1] + S[S[1]] = 1$ or $S[1] + S[S[1]] = S[1]$
    - → case: $S[1] + S[S[1]] = 1$

      then: since the values has not been changed (assumption), we would have $S_l[1] + S_l[S_l[1]] = 1$ as well. But it is equal to l by another assumption. The case is impossible to occur!

    - → case: $S[1] + S[S[1]] = S[1]$

then: $S_l[1]+S_l[S_l[1]]=S_l[1]$, so $S_l[S_l[1]]=0$. But $S_l[1]<l$ and $S_l[1]+S_l[S_l[1]]=l$, so we must have $S_l[S_l[1]]>0$. The case is impossible.

–

Krack

- attack based on crypto assumption: "no IV used twice"
- works despite "provable security", but the proofs have not modelled all scenarios
- effects depend on particular implementation. Most cases:
  - decryption due to reuse of the same string in stream cipher
  - or just making mess by replay attack (e.g. against NTP- network time protocol)

4-way handshake

- "supplicant"= user, "authenticator"=Access Point
- PMK Pairwise Master Key is preshared
- PTK (Pairwise Tansient Key) derived as a session key
- PTK=f(PMK, ANonce, SNonce), PTK splitted into TK (Temporal Key), KCK (Key Confirmation Key), KEK (Key Encryption Key)
- for WPA2 also GPK (Group Temporal Key) transported to the supplicant (used by AP for transmissions)
- frames: EAPOL consisting of
  - header - determines which message in the handshake
  - replay counter – used to detect replayed frames, replay counter increased
  - nonce - nonces (of supplicant and authenticator) to generate new keys
  - RSC Receive Sequence Counter - starting packet number of group key
  - MIC - contains Message Integrity Check created with KCK
  - Key Data - contains group key encrypted with KEK
- encryption schemes used: AES-CCMP, GCM , MAC: Michael (weak), GHASH (from GCM)

handshake:

- notation: after ";" the data are encrypted
- green background = "sometimes"

association stage	supplicant		authenticator
		Authentication request →	
		← Authentication response	
4-way handshake		←Msg1(r,Anonce)	
	derive PTK		
		Msg2(r,Snonce) →	
		←Msg3(r+1,GTK)	
			derivePTK
		Msg4(r+1) →	
	install PTK,GTK		install PTK
group key		← $\text{Enc}^{x}_{\text{PTK}}(\text{Group1}(r+2;\text{GTK}))$	
handshake		$\text{Enc}^{y}_{\text{PTK}}(\text{Group2}(r+2))$ →	
	install GTK		install GTK

Table 2.

- state automaton definded, states for the supplicant:

  A) PTK-INIT:

  - entered when 4 way handshake started
  - exit to state PTK-START with Msg1 received
  - operations: PMK- preshared master key

  B) PTK-START:

  - exit: self loop with MSg1 received, with proper Msg3 to state PTK-NEGOTIATING (proper= MIC correct and no replay)
  - operations:
    - TPTK=CalcPTK(PMK,ANonce,SNonce)
    - Send Msg2(SNonce)

  C) PTK-NEGOTIATING:

  - exit: unconditional to PTK-DONE
  - operations:
    - PTK=TPTK
    - Send Msg4

  D) PTK-DONE:

  - exit: to PTK-START if Msg1 received, to PTK-NEGOTIATING if proper Msg3 received attack 1 - plaintext retransmission of Msg3

supplicant		adv		authenticator
	← Msg1(r,Anonce)		← Msg1(r,Anonce)	
derive PTK				
	Msg2(r,Snonce) →		Msg2(r,Snonce) →	
	← Msg3(r+1;GTK)		← Msg3(r+1;GTK)	
				derivePTK
	Msg4(r+1) →			
install PTK,GTK				
	$\text{Enc}^1_{\text{PTK}}\{\text{Data}(A....)\} \rightarrow$			
	← Msg3(r+2;GTK)		← Msg3(r+2;GTK)	
	$\text{Enc}^2_{\text{PTK}}\{\text{Msg4}(r+1)\} \rightarrow$			
resinstall PTK,GTK				
			$\text{Enc}^2_{\text{PTK}}\{\text{Msg4}(r+1)\} \rightarrow$	(rejected)
			Msg4(r+1) →	
				install PTK
	$\text{Enc}^1_{\text{PTK}}\{\text{Data}(B....)\} \rightarrow$		$\text{Enc}^1_{\text{PTK}}\{\text{Data}(....)\} \rightarrow$	

mechanism:

- according to the 802.11 standard Msg4(r+1) will be accepted as it is checked that r+1 is a replay counter used before

- the problem is that $\text{Enc}^1_{\text{PTK}}\{\text{Data}(A....)\}$ and $\text{Enc}^1_{\text{PTK}}\{\text{Data}(B....)\}$ use the same IV but security of the encryption modes used collapse in this case

attack 2 - only ciphertext retransmission of Msg3 accepted

CPU		NIC		adversary
	← Msg1(r,Anonce)		← Msg1(r,Anonce)	
	Msg2(r,Snonce) →		Msg2(r,Snonce) →	
			← Msg3(r+1;GTK)	
			← Msg3(r+2;GTK)	
	← Msg3(r+1;GTK)			
	← Msg3(r+2;GTK)			
	Msg4(r+1) →			
	install keys →		Msg4(r+1) →	
		install PTK,GTK		
	Msg4(r+2) →			
	install keys →		$\text{Enc}^1_{\text{PTK}}\{\text{Msg4}(r+2)\} \rightarrow$	
		reinstall PTK,GTK		
	Data(....) →			
			$\text{Enc}^1_{\text{PTK}}\{\text{Data}(....)\} \rightarrow$	

mechanism:

- assumption: encryption and decryption offloaded to NIC

- main CPU does not decrypt messages and always receives messages already decrypted by NIC,

- so it cannot distinguish the case when Msg3(r+2;GTK) has been received as plaintext or already encrypted. In both cases the reaction is the same and asks NIC to install keys
- adversary holds the first Msg3 until the authenticator sends another one because of no response
- finally two ciphertexts created with the same IV
- the problem is that in fact there are two state machines - one for main CPU and one for NIC and collectively they are not equivalent to the original machine from the standard

attack 3 - in some systems (MacOS) the message Msg3 has to be encrypted

attack when refreshing the key

CPU		NIC		adversary
	← Msg1(r,Anonce)		← Msg1(r,Anonce)	
	Msg2(r,Snonce) →		Msg2(r,Snonce) →	
			← $\mathrm{Enc}^{x}_{\mathrm{ptk}}$(Msg3(r+1;GTK))	
			← $\mathrm{Enc}^{x+1}_{\mathrm{ptk}}$(Msg3(r+2;GTK))	
	← Msg3(r+1;GTK)			
	← Msg3(r+2;GTK)			
	Msg4(r+1) →			
	install keys →		Msg4(r+1) →	
		install PTK,GTK		
	Msg4(r+2) →			
	install keys →		$\mathrm{Enc}^{1}_{\mathrm{PTK}}\{\mathrm{Msg4}(r+2)\}$ →	
		reinstall PTK,GTK		
	Data(....) →			
			$\mathrm{Enc}^{1}_{\mathrm{PTK}}\{\mathrm{Data}(....)\}$ →	

mechanism:

- the countermeasure was that when refreshing then the Msg3 must be encrypted
- intention was that encryption with the new key so after reinstallation new key used and no problem that the counter starts again from 1
- the mistake is that it is not checked under which key the messeage has been encrypted

attack 4 - group key reinstallation

challenge:

when to reinstall the key for AP? Options:

a) right after sending information to the supplicants

b) after receiving ack from all supplicants

each scenario leads to problems

attack 4a - group key reinstallation immediately after sending group message

supplicant		adv		authenticator
				refresh GTK
	$\leftarrow \mathrm{Enc}^{x}_{\mathrm{PTK}}\{\mathrm{Group1}(r; \mathrm{GTK})\}$		$\leftarrow \mathrm{Enc}^{x}_{\mathrm{PTK}}\{\mathrm{Group1}(r; \mathrm{GTK})\}$	
install GTK				install GTK
	$\mathrm{Enc}^{y}_{\mathrm{PTK}}\{\mathrm{Group2}(r)\} \rightarrow$			
			$\leftarrow \mathrm{Enc}^{x+1}_{\mathrm{PTK}}\{\mathrm{Group1}(r+1; \mathrm{GTK})\}$	
	$\leftarrow \mathrm{Enc}^{1}_{\mathrm{GTK}}\{\mathrm{GroupData}(...)\}$		$\leftarrow \mathrm{Enc}^{1}_{\mathrm{GTK}}\{\mathrm{GroupData}(...)\}$	
	$\leftarrow \mathrm{Enc}^{x+1}_{\mathrm{PTK}}\{\mathrm{Group1}(r+1; \mathrm{GTK})\}$			
reinstall GTK				
	$\leftarrow \mathrm{Enc}^{1}_{\mathrm{GTK}}\{\mathrm{GroupData}(...)$			

mechanism:

- after key reinstallation one can replay the old message as the index 1 will be accepted again

attack 4b - group key reinstallation after ack from all supplicants

supplicant		adv		authenticator
				refresh GTK
	$\leftarrow \mathrm{Enc}^{x}_{\mathrm{PTK}}\{\mathrm{Group1}(r; \mathrm{GTK})\}$		$\leftarrow \mathrm{Enc}^{x}_{\mathrm{PTK}}\{\mathrm{Group1}(r; \mathrm{GTK})\}$	
install GTK				
	$\mathrm{Enc}^{y}_{\mathrm{PTK}}\{\mathrm{Group2}(r)\} \rightarrow$			
			$\leftarrow \mathrm{Enc}^{x+1}_{\mathrm{PTK}}\{\mathrm{Group1}(r+1; \mathrm{GTK})\}$	
			$\mathrm{Enc}^{y}_{\mathrm{PTK}}\{\mathrm{Group2}(r)\} \rightarrow$	
				install GTK
	$\leftarrow \mathrm{Enc}^{1}_{\mathrm{GTK}}\{\mathrm{GroupData}(...)\}$		$\leftarrow \mathrm{Enc}^{1}_{\mathrm{GTK}}\{\mathrm{GroupData}(...)\}$	
	$\leftarrow \mathrm{Enc}^{x+1}_{\mathrm{PTK}}\{\mathrm{Group1}(r+1; \mathrm{GTK})\}$			
reinstall GTK				
	$\leftarrow \mathrm{Enc}^{1}_{\mathrm{GTK}}\{\mathrm{GroupData}(...)\}$			

mechanism:

- again, after key reinstallation at the supplicant (victim) one can replay the old message as the index 1 will be accepted again

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 very nice 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.

– –

XVI. PKI INFRASTRUCTURES

global versus local names

- OpenPGP
- X.509
- SPKI

anonymous authorization

- PKI
- BSI standards

X.509

- revocation: CRL, OCSP
- time limited validity (models: Austrian versus German to refresh validity of a document)

OpenPGP

- web of trust

flat infrastructure for biometric passports

- short time certificates for Document Verifier
- anchor points in passports for time

SPKI

- keys are principals for authentication
- basic name: key +identifier, key defines name space, identifier can refer to somebody in the name space
- compound names: key identifier1 identifier2 ... identifier$n$

- name certificate: $\mathrm{Sing}_K(K, A, S, T)$, where K is the public key of name space, A is a basic name (no compound names), S is a subject (typically key or name), T is validity time
- no single certificate with given A, creating groups possible
- authorization certificates: $\mathrm{Sign}_K(K, S, d, T, V)$ where K is the public key of name space, S is a local name or a key that receives authorization, d is delegation flag, T is a tag (interpretation subject to application), V is validity specification (time, condition, ...)
- chains of certificates, rewrite rules: each certificate is a directed edge: issuer gives the right to the subject, in fact transitive closure computed
- S-expressions - LISP like semantics for names, tags, ... There is logic to interpret multiple tags (intersection of rights)
- trust model unspecified – left for application level (X.509 explicit trust model)
- health certificate and suicide note

__________________________________-

VIII. HOW TO CREATE A SYSTEM IN THE WORST POSSIBLE WAY?

Example: CHIP AUTHENTICATION PROGRAMME

“optimisation is the process of taking something that works and replacing it with something that almost works but is cheaper”

CAP - idea:

- cheap, Chip&PiN device
- keyboard, display, chip reader
- protecting PIN (it does not go to the PC)
- CAP device not personalized, one can use own card with CAP in a bank, or borrow from somebody
- recommended to use own CAP device
- optimized to be as much as possible based on EMV (standard for electronic purse)

used in UK, Master Card initiative

operation modes:

- identify: returns one-time code (like RSA-token) (based on symmetric key)
- respond: responds to a challenge using symmetric key

– sign: just as respond, however takes account number and value to generate the response

Protocol overview:

1. select application of the card (CAP has some fixed identifiers)
2. read records: account number, certificates , ... but important: CDOL1, CDOL2 (card object lists) and CAP (bit filter defining the protocol execution)
3. PIN verification
4. ciphertext generation: GENERATE AC command, response: Authorisation Request Cryptogram (ARQC), then the reader asks for Application Authentication Cryptogram (AAC) indicating cancelling the transaction (according to EMV)

challenge: AA (Authorized Amount), UN (Unpredictable number)

- for identify both are 0
- for respond: AA=0, UN=challenge
- for sign: AA=transaction value, UN= destination account

Response:

- based on the following data: ATC (application transaction counter), CID (Cryptogram Identification Data), IAD (Issuer Application Data - contains result of PIN verification), AC (Application Cryptogram - MAC (3DES CBC MAC) of the rest)
- CAP filter used to determine which bits to take
- NatWest: 5 least significant bits of ATC, 20 least significant bits of MAC, 1 bit from IAD
- Barclay: top bit of CID, 8 least significant bits of ATC, 17 least significant bits of MAC
- HBOS: top bit of CID, 7 least significant bits of ATC, 17 bits of MAC (not in one block), 1 bit from IAD

Verification: recomputed with the secret key shared with the card

Application:

- bank decides how to use (mode + semantic field)
- NatWest: respond mode, 8 bits of challenge, 4 random, 4 =last 4 digits of destination account, not used for login, transaction value not authenticated
- Barclay: identify necessary for login, for transaction: sign with destination account and transaction value (no freshness from bank, only ACT against replay – but might be played later)

Serious mistakes:

→ checking PIN, result available on the device (mugging threat) – this concerns also cards of other banks

→ the same PIN for ATM and online authentication – some keys on the CAP clean and some used - after stealing it one has 3 trials, 24 permutations on 4 keys, pbb to guess PIN to ATM becomes $\frac{1}{8}$

→ CAP has no secret, infected PC may emulate CAP

→ GSM in CAP to transmit secrets

→ complicated instruction manual, the user may insert something else than intended account number

→ overloading: sign with transaction with 0 value is valid for response (for a random account-nounce)

→ NatWest: nonce as 4 digits in respond challenge, Chip&PIN terminal requests a number of responses from the card, later number of challenges from the online bank, there would be a match due to birthday paradox

(there are info indicating the attack: the number of requests, the change of transaction counter)

critical mistake: MITM regarding PIN verification

- PIN verification result never explicitly stated. Info to the bank contained in TVR (terminal verification results) and IAD (Issuer Application Data)
- TVR states possible failure conditions for authentication, in success not indicated which method used
  - → bit8=1: carholder verification was not successful
  - → bit7=1: unrecognized CVM
  - → bit6=1: PIN Try limit exceeded
  - → bit5=1: PIN required and PIN pad not present
  - → bit4=1:PIN required, PIN pad present and PIN not entered
  - → bit3=1: online PIN entered
- IAD may contain info on whether the PIN has been verified, but cannot be read by the terminal (proprietary format), So terminal can have a different picture of the situation
  - → bi4=1: Issuer Authentication performed and failed
  - → bit3=1: offline PIN performed
  - → bit2=1: offline PIN verification performed and failed
  - → bit1=1: unable to go online
- attack:
  1. tricking the terminal by sending 0x9000 to `Verify` without sending PIn to the card

2. card thinks that the terminal is not supporting PIN and skip PIN or uses signature
3. card does not increase PIN retry counter
4. issuer thinks that the terminal was not supporting PIN and accepts

- practical case (as described in 2015 paper after 2011 case in Belgium)
  - credit cards stolen, used in Belgium, police used intersection analysis (card usage, SIM cards in the proximity) to identify the criminals
  - ”minimal effort design”, just to work. Implementation of the attack with MiTM
  - hardware: FUN chip attached to the original chip, wires connected (contacts of the FUN with contacts of the original chip), the card has traces of manipulation. thickness: .82 mm (instead of .76mm)
  - functional: data embossed on the card does not match the data from the chip, accepts any PIN, some wrong responses

What went wrong:

- no evaluation, no public certification report
- no reaction to S&P paper from 2011
- specification EMV: thousands of pages
- certification costs
- designing a solution: chaos, no sufficiently detailed documentation and regime
- CC very likely to fail:
  - → asset: PIN, password, protected against use on a PC
  - → no methodology to answer the question: **what are side-effects of protecting one asset**
  - → important: security is **not monotonic**: improving situation with respect to one threat may worsen situation to another one. **Not reflected by CC framework**.
  - → **optimization is necessary, but may lead to situation that is worse than the original one**

  (other solution: a shadow PIN for the case of mugging)

I. FAILURE EXAMPLES TO LEARN FROM

I.1. PKI for Signing Digital Documents

PKI - Public Key Infrastructure

- strong authentication of digital documents with digital signatures seems to be possible
- in fact we get an evidence that the holder of a private key has created a signature
- who holds the key? PKI has to provide a certified answer to this question
- PKI is not a cryptographic solution - it is an organizational framework (using some crypto tools)

PKI, X.509 standard

- a certificate binds a public key with an ID of its alleged owner,
- a couple of other fields, like validity date, key usage, certification policy, ...
- certificate signed by CA (Certification Authority)
- tree of CA's (or directed acyclic graph), with roots as "root of trust"
- status of a certificate may change - revocation
- checking status methods: CRL, OCSP

reasons for PKI failure:

a nice concept of digital signatures but

1. big infrastructure required:
   - big effort and cost
   - long time planning needed (so possible in China, but not in Europe)
   - unclear financial return
2. scope of necessary coordination,
   - in order to work must be designed at least for the Common Market
   - example of killing the concept: link to certification policy in Polish
3. lack of interoperability (sometimes as business goal)
   - companies make efforts to eliminate competition
   - standarization may be focused on having market shares

4. necessary trust in roots
    - how do you know that the root is honest?
5. registration: single point of fraud, (e.g. with fake breeding documents)
    - once you get a certificate you may forge signatures
6. responsibility of CA
    - fiancial risk – based on risk or responsibility
7. cost - who will pay? For the end user the initial cost is too high.
    - certificates are too expensive for just a few signatures (at least initially)
8. legal strength of signatures
    - if scheme broken or signing devices turn out to be insecure you are anyway responsible for the signatures
9. unsolved problem of revocation: possible to check the status in the past but not now

reason: mismatch of requirements and interests with the designed solution

I.2. Clickjacking on Android

Overlay mechanism:

- apps are separated in their sandboxes - security design mechanism
- all apps display informations on the screen on the same time - they are overlays
- overlays:
    - require Android permissions
    - clickable or paththrough
    - opaque or transparent
    - combining: parameter α defines weights: for each color of RGB the new pixel value is

$$\text{old}\cdot\alpha + \text{new}\cdot(1-\alpha)$$

- basic clickjacking:
    - on top an opaque overlay with something innocent (game...)
    - button is in fact a batton but paththrough overlay
    - below is an unvisible button of an attacked app

  defense: Google's "obscure flag" - an app checks if at the momnet of clicking there is an overlay above it

- context-hiding clickjacking: overlay covers everything but not the button

  defense: Google's "hide overlays" (applicable only in case of settings etc as the users like overlays)

- examples:

  → Google play: after installing the app it asks for "open app", but acceptance is for installing and opening anything else

  → Browser: cover the context and make the user click (e-voting?)

  → gmail: prepare a message (available tools), cover it with overlay and ask for accepting "send" button

  → whatsapp: send messages, send SMS to chosen destinations (and learn attacked SIM subscriber number)

  → Google Authenticator: "long click" copies token to clipboard (and makes it available to other apps)

  → Facebook, Tweeter: unprotected, possibilities to insert likes, send tweets, ...

  → Lokout Mobile Security: 3 clicks and anti-virus protection disabled

- remedy?

  - no effective protection mechanism known
  - architecture separates apps so it is impossible to ask what the othe apps are showing
  - some overlays must be tolerated because of the requests of the users
  - pressure to run security critical apps (banking...)

  Clickshield proposal:

  - requires minor changes in the Android framework
  - concentrates on the central region (as on margin no critical buttons observed)
  - attempts to check whether between the app and the screen final value there is a uniform "delta"
    - choose two points
    - solve equations with unknowns α,δ :

      $$r_1 = \alpha \cdot d_1 + (1-\alpha) \cdot \delta$$

      $$r_2 = \alpha \cdot d_2 + (1-\alpha) \cdot \delta$$

    then substract the values D - screen of the app- from the final screen values R. The result Δ should be homogenous. Otherwise there is an context switching.

I.2. Easy Fishing on Android

- mobile password managers:
  - associate app (package name) with a domain name
  - for a domain name associated with the app insert the user's credentials when the app accesses this URL
  - credentials are related to the domain name (facebook.com, etc) and not to the app
  - in fact: improves protection against fishing (difference between facebook and faceb00k detected, a human may make a mistake)
- Instant Apps: instead of downloading the whole app fetch only a small app that emulates the full version with accessing an URL
  - gets a full control over the screen –e.g. it may hide browser's security information
- limitations: at most one app with the same package name on an Android device, and at most one in Play Store
- attack:
  - create an app
  - choose the package name so that the mapping points to the attacked domain
  - include developer's URL for Instant App purposes (it is not checked by the Play Store)
  - lure the user to run Instant App
    - → the credentials will be included by the Password Manager
    - → the information will go to the developer's URL
    - → Instant App will show something different on the screen (information from password manager need not to be visible, browser information will be covered as well)
    - → the user should dislike the app and consequently the app will not be downloaded (no traces of forgery)
- mappings:
  - most important associations on a kind of whitelist
  - one-to-one would be more secure but frequently many apps to one domain
  - (insecure) heuristics:
    1. Keeper: finds a corresponding entry on play.google.com and takes "app developer website field" , it autosuggests this name to the user,

attacks: write malicious app “developer website field”

2. Dashlane: hardcopied 81 mappings, rest: autosuggestion heuristic based on at least 3 matching characters: xxx.face.yyy will be mapped to facebook.com

   attack: use similar package names

3. Lastpass: translates directly (aaa.bbb.ccc to bbb.aaa), if not existing then consult from crowdsourced mapping (distributed database created by users – where it is easy to inject something)

4. 1Password: does not provide mapping but presents suggestions and enables the user – a human to choose (and confuse FACEB00K with FACEBOOK)

5. Google Smart Lock – burden of mapping to the developer (at the time not automated process) based on Digital Asset Links (on the website a list of permitted apps, authenticated with hashes of the signing key)