Security and Cryptography 2016

Mirosław Kutyłowski

grading criteria: 50% exam, 50% assignments

skills to be learned: developing end-to-end security systems, they must be flawless!

rules: do not memorize the standards, they come and go. Only the skills are important

presence: obligatory during the lectures

exam date: TBA (quickly to enable internships in February)

I. EXAMPLE TO LEARN FROM: PKI FAILURE

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 (according to Bruce Schneier):

1. whom we trust and for what? why CA should be trusted??
2. who is using my key? (private key - there are really no clones??)
3. how secure is the verifying computer? (no cryptography can help is the verifier software is cheating)
4. who is the signer? (ambiguity unless there is a trustworthy ID registry)
5. is CA an authority? (really not an authority for data contained in the certificate. Certificate based on fake documents...)
6. is the user part of the security design? (no, the user is free to behave in a stupid way)
7. separation CA and RA brings new threats
8. How did CA verify the certificate holder? (certificate issued for ..., but how to know that this was really this person)
9. How secure are the certification practices? (revocation, etc)

10. the customers wish to run single-sign-on

reasons for PKI failure (according to me):

a nice concept of digital signatures but

1. big infrastructure: time to build,
2. scope of necessary coordination,
3. lack of interoperability (sometimes as business goal)
4. necessary trust in roots
5. registration: single point of fraud, (e.g. with fake breeding documents)
6. responsibility of CA
7. cost - who will pay? For the end user the initial cost is too high.
8. legal strength of signatures
9. unsolved problem of revocation: possible to check the status in the past but not now

MAJOR PROBLEM: how to design/buy sound systems?

II. COMMON CRITERIA FRAMEWORK

http://www.commoncriteriaportal.org/

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:** custemer 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 crucial data

Idea of Common Criteria Framework:

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

Value:

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

Example for PP: BAC (Basic Access Control)

- used to secure wireless communication between a reader and a 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 master key

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

and 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 guid ance
- AGD_PRE 1 1 1 1 1 1 1 Preparative procedures

→ life-cycle support

- ALC_CMC 1 2 3 4 4 5 5 CM capabilities
- ALC_CMS 1 2 3 4 5 5 5 CM scope
- ALC_DEL - 1 1 1 1 1 1 Delivery
- ALC_DVS - - 1 1 1 2 2 Development securit
- 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

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

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

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

– 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

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

  – but not when they are 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. EIDAS REGULATION

goals:

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

tools:

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

technical concept:

- Member State provides an online system enabling identification and authentication with means from ths member state used abroad
- notification scheme for national systems
- if notified (some formal and technical conditions must be fulfilled), then every member state must admit 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 zoo of solutions
- breakthrough claimed, but likely to fail

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 proofing and verification (for mnatural 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

- electornic 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 different 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.

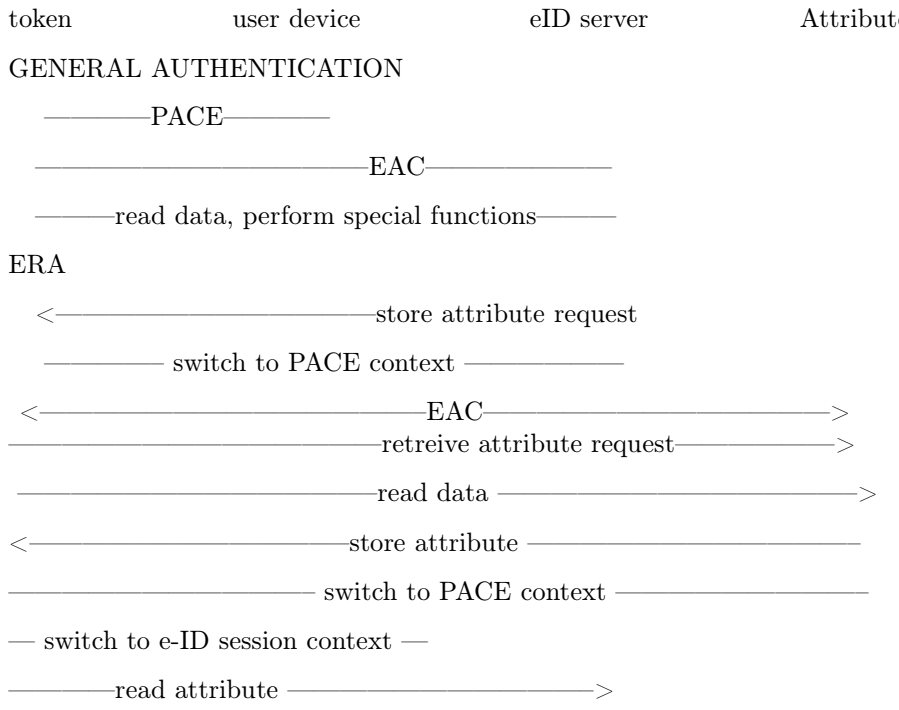
  2. Where a scheme is directly managed by a government body, it is audited in accordance with the national law.

IV. eIDAS TOKEN SPECIFICATION, BSI

- Technical guideline, security mechanisms for electronic travel documents, not focused on readers
- cryptographic mechanisms:
  - Password Authenticated Connection Establishment (PACE)
  - Terminal Authentication Version 2 (TA2).
  - Chip Authentication Version 3 (CA3)
  - Restricted Identification (RI)
  - Pseudonymous Signatures (PS)
- procedures
  - General Authentication Procedure (GAP)
  - Enhanced Role Authentication (ERA)
  - PIN Management
- terminal types:
  - inspection system
  - authentication terminal - government or private, terminal rights to be checked, GAP must be used
  - attribute terminal- extension of Authentication Terminal, ERA must be used
  - signature management terminal - key creation, signature creation
  - signature terminal - GAP must be used
  - priviledged terminals: category: inspection terminals and some authentication terminals explicitly authorized. Signature terminals are never priviledged
- user credentials:
  - MRZ-Password
  - CAN
  - PIN - always blocking (RC reaches 0 then blocked)
  - PUK - blocking or non-blocking
- password blocking: RC=0 password blocked, RC=1 - password suspended and the correct CAN must be entered during the same session to resume the password. Resume is volatile.
- switching session context: a stack of protocols, when terminating a protocol we return to the context on the top of the stack

- password authentication:
  - PACE- global passwords, VERIFY-application local
  - Inspection terminal SHALL use CAN or MRZ
  - authentication terminal SHALL use PIN, but the CAN can be allowed by the terminal
  - signature terminal: PIN, CAN or PUK
- Extended Access Control:
  - 1. Terminal Authentication v2: terminal SHALL generate ephemeral keys used later for Chip Authentication, only standard parameters, ephemeral keys authenticated, result: read/write access granted
  - 2. Passive Authentication: terminal reads and verifies Security Objects, compares the data obtained before PACE
  - 3. Chip Authentication v2 or v3: afterwards secure channel restarted
- General Authentication Procedure:
  i. password verification - PACE
  ii. EAC
  iii. read/write data
- Enhanced Role Authentication – authentication terminal with proper rules can proceed as follows:
  i. authentication terminal sends an ATTRIBUTE REQUEST to eIDAS token. token makes a link between the request and the terminal's sector
  ii. restore session context of PACE, store context of Chip authentication
  iii. EAC with attribute provider
  iv. proceed attribute request, write the resulting attributes to the eIDAS token, the access rights restricted to terminals with proper rights
  v. restore session context: PACE, then Chip Authentication
  vi. terminal may read the stored attributes
- online authentication:
  - eID server: remote part of authentication terminal
  - user device: interacts with user, eIDAS token and eID server, but not authorized to read eIDAS token data, access rights only after authentication with the eID server

  Protocol chart:



- unauthenticated terminals:
  i. password verification based on PACE:
     – terminal does not show its type
     – can choose password type
     – after authentication secure messaging
  ii. authentication with CAN resumes PIN
  iii. updating retry counter

- authenticated terminals: after terminal authentication the terminal becomes authenticated

Cryptographic building blocks:

– hash $\boldsymbol{H}(m)$
– compression function for public key: $\boldsymbol{Comp}$(PK)
– projected representation of a public key $\boldsymbol{\Pi}$(PK)
– symmetric key algorithms:
  – deriving key for encryption $K_{\text{Enc}} = \boldsymbol{KDF}_{\text{Enc}}(K, [r])$
  – $K_{\text{Mac}} = \boldsymbol{KDF}_{\text{Mac}}(K, [r])$

  - $K_\pi = \boldsymbol{KDF}_\pi(\pi)$
  - encryption and decryption
  - MAC
- asymmetric algorithms:
  - domain parameters
  - keys (page 19):
    - eIDAS: ephemeral on both sides
    - Chip authentication: static on side of the chip
    - Chip authentication version 3: ephemeral on both sides based on static Chip's key
    - Restricted Identification: token uses a static key, sector public key, sector specific identifier
  - KA - key agreement (like DH)
  - signatures, mapping to RSA and ECDSA described

Pseudonymous signature:

- used for anonymous signature and for Chip Authentication v3
- 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 **Π** omitted here)

iv. compute

- $s_1 = K_1 - c \cdot \mathrm{SK}_{\mathrm{ICC},1}$
- $s_1 = K_2 - c \cdot \mathrm{SK}_{\mathrm{ICC},2}$

v. output (c, s_1, s_2)

- verification:

  compute

  - $Q_1 = (\mathrm{PK}_{\mathrm{ICC}})^c \cdot g^{s_1} \cdot (\mathrm{PK}_M)^{s_2}$
  - $A_1 = (I^{\mathrm{sector}}_{\mathrm{ICC},1})^c \cdot (\mathrm{PK}_{\mathrm{sector}})^{s_1}$
  - $A_2 = (I^{\mathrm{sector}}_{\mathrm{ICC},2})^c \cdot (\mathrm{PK}_{\mathrm{sector}})^{s_2}$
  - recompute c and check against the c from the signature

- why it works?

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

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

PACE (Password Authenticated Connection Establishment)

- ICAO Doc 9303: Basic Access Control/PACE and EAC v1 (=Chip Authentication v1+ Terminal Authentication v1) MUST be used
- password based authentication protocol
- password on the side of the token: stored, on the terminal: input by the user
- steps:

i. token chooses s at random

ii. token computes $z = \mathrm{Enc}(K_\pi, s)$, where $K_\pi = \mathrm{KDF}(\pi)$ and sends z to the reader together with the parameters D_{PICC}

iii. the reader recovers s

iv. the reader and the token compute $D_{\mathrm{Mapped}} = \mathrm{Map}(D_{\mathrm{PICC}}, s)$ (mapping function)

v. the reader and the token perform anonymous Diffie-Hellman key agreement based on the ephemeral domain parameters (ephemeral values based on D_{Mapped} as an generator), shared secret K obtained

vi. they create session keys $K_{\text{Mac}} = \text{KDF}_{\text{Mac}}(K)$ and $K_{\text{Enc}} = \text{KDF}_{\text{Enc}}(K)$

vii. exchange and verification of tokens: $T_{\text{PCD}} = \text{MAC}(K_{\text{MAC}}, \text{ephemeral key of PICC})$
$T_{\text{PICC}} = \text{MAC}(K_{\text{MAC}}, \text{ephemeral key of PCD})$

viii. Secure Messaging restarted

Terminal authentication v2

- Chip Authentication MUST be performed after Terminal Authentication (condition repeated in the description of CHA v2 only)
- simple challenge-response algorithm, undeniable, resistant to replay
- ephemeral public key for ChA as a side effect
- steps:

  i. the terminal send the certificate chain to eIDAS token, it has to confirm the key PK_{PCD}

  ii. the token checks PK_{PCD}

  iii. the terminal creates ephemeral pair of keys, sends the compressed version of $\widetilde{\text{PK}_{\text{PCD}}^{\text{CA}}}$ to the token

  iv. token replies with a random nonce r_{PICC}

  v. the terminal signs with SK_{PCD} the following data: r_{PICC}, compressed version of $\widetilde{\text{PK}_{\text{PCD}}^{\text{CA}}}$

  vi. the token checks the signature

Chip authentication v2

- static DH authentication with the ephemeral key of the terminal
- steps:

  i. the token sends its public key PK_{PICC}

  ii. the terminal sends ephemeral public key from TA (uncompressed)

  iii. static DH key agreement with SK_{PICC} and ephemeral public key on side of the token, and PK_{PICC} and ephemeral secret key on side of the terminal, master key K generated

  iv. token chooses r_{PICC}, computes $K_{\text{Enc}} = \text{KDF}_{\text{Enc}}(K, r_{\text{PICC}})$, $K_{\text{Mac}} = \text{KDF}_{\text{Mac}}(K, r_{\text{PICC}})$

  v. token computes the tag $T_{\text{PICC}} = \text{MAC}(K_{\text{Mac}}, \text{ephemeral public key of the terminal})$

vi. the terminal checks the tag

vii. secure messaging restarted using K_{Enc} and K_{Mac}

Chip authentication v3

- alternative to Chip authentication v2 and RI
- claimed: "message-deniable strong authentication", "pseudonymity without using the same key on several chips", "possibility of whitelisting eIDAS tokens"
- scheme:

  i. phase 0: terminal authentication, ephemeral key for terminal in phase 1 chosen and signed

  ii. phase 1: key agreement like DH with ephemeral keys on both sides, restarting secure messaging with new keys

  iii. phase 2:

  – static keys on the side of the chip: $\text{SK}_{\text{ICC},1}, \text{SK}_{\text{ICC},2}, \text{PK}_{\text{ICC}}$ and the parameters

  – terminal sends $\text{PK}_{\text{sector}}$ to the chip, the chip compares it with the "compressed" version received during Terminal Authentication

  – chip reconstructs $I^{\text{sector}}_{\text{ICC},1} = (\text{PK}_{\text{sector}})^{\text{SK}_{\text{ICC},1}}$ and $I^{\text{sector}}_{\text{ICC},2} = (\text{PK}_{\text{sector}})^{\text{SK}_{\text{ICC},2}}$

  – chip creates pseudonymous signature using $I_{\text{ICC},1}$, $I_{\text{ICC},2}$ as pseudonym and the secret keys $\text{SK}_{\text{ICC},1}$, $\text{SK}_{\text{ICC},2}$ over the ephemeral key given by the terminal

- If PACE GM used before ChA v3 then one can reuse the ephemeral key from the terminal
- checking the key PK_M is obligatory (otherwise it would be easy to forge the token)

Restricted Identification

- optional
- depending on the version, deanonymization might be possible or not (depending on $\text{PK}_{\text{sector}}$)
- executed after Terminal Authentication and Chip Authentication (not specified which version, but with v3 it does not makes sense)
- sector specific identifier computed as Hash(key computed via DH from $\text{PK}_{\text{sector}}$ and SK_{ID})
- blacklisting impossible in case of group key compromise (from ChA v2)

Pseudonymous Signature as replacement of RI

- whitelisting possible in case of group key compromise (claimed as new but possible for RI)

- the second part from ChA v3, the key PK_{sector} used as sector public key

PSA - Pseudonymous Signature Authentication

- the sector public key = the ephemeral public key from ephemeral DH key agreement (now DH explicitly mentioned)

PSM - Pseudonymous Signature of a Message

- TA and ChA must be executed before
- message to be signed comes from the terminal
- public key unspecified

PSC - Pseudonymous Signature of Credentials

- used in combination with ERA
- Attribute Terminal involved, but eIDAS token creates the signature himself (after breaking group key one can also create the PSC)
- public key unspecified
- terminal rights to get the attributes are to be checked

PROBLEMS:

- security properties not stated, they can be derived via tedious analysis
- lack of security proofs
- underspecified (details may turn the token to be insecure)
- powerful adversary able to break into the token may crate fake ID's, unless whitelist approach used

---

V. STANDARS VERSUS SECURITY

Bleichenbacherr's RSA signature forgery based on implementation error

The attack works for PKCS-1 padding.

The PKCS-1 padding consists of a byte of 0, then 1, then a string of 0xFF bytes,
then a byte of zero, then the "payload" which is the hash+ASN.1 data.
Graphically:

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

The signature verifier first applies the RSA public exponent to reveal this PKCS-1 padded data, checks and removes the PKCS-1 padding, then compares the hash with its own hash value computed over the signed data.

The error that Bleichenbacher exploits is if the implementation does not check that the hash+ASN.1 data is right-justified within the PKCS-1 padding. Some implementations remove the PKCS-1 padding by looking for the high bytes of 0 and 1, then the 0xFF bytes, then the zero byte; and then they start parsing the ASN.1 data and hash. The ASN.1 data encodes the length of the hash within it, so this tells them how big the hash value is. These broken implementations go ahead and use the hash, without verifying that there is no more data after it. Failing to add this extra check makes implementations vulnerable to a signature forgery, as follows.

An attacker forges the RSA signature for an exponent of 3 by constructing a value which is a perfect cube. Then he can use its cube root as the RSA signature. He starts by putting the ASN.1+hash in the middle of
the data field instead of at the right side as it should be. Graphically:

```
00 01 FF FF ... FF 00  ASN.1  HASH  GARBAGE
```

This gives him complete freedom to put anything he wants to the right of the hash. This gives him enough flexibility that he can arrange for the value to be a perfect cube.

There are some other variations of this attack, for example some implementations of the signature verification algorithm neglect to check if the field "parameters" inside "digestAlgorithm" field is NULL. In such a case an attacker may put some GARBAGE here, making the attack still possible even if the algorithm verifies that the HASH is right-justified.

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

An attacker has access to an oracle that, for any chosen ciphertext, indicates whether the corresponding plaintext $C^d \bmod n$ has the correct format according to the encryption standard.

Then after querying the oracle with about 2^{20} adaptively chosen ciphertexts the attacker is able to calculate the plaintext correpsonding to the original ciphertext.

The oracle might be for example a HSM that receives ciphertexts resulting from the key-wrap algorithm.

The attack exploits such vulnerabilities like

- different 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).

VI. FORMAL SECURITY PROOFS

Security model e.g. for PACE

Background (Hanzlik, MK)

data confidentiality: nobody can understand any data from the communication between an eID and a terminal, except for this eID and this terminal. By "data" we mean:

- workload data to be transmitted via the channel established according to the protocol,
- partner specific data (such as partner identity) - if sending them (explicitly or implicitly) results from the protocol execution.

data integrity: a third person cannot manipulate without detection the data exchanged between the eID and the terminal. This concerns in particular manipulating identity data.

session integrity: if a party A accepts at some moment a session executed presumably with a single partner, then indeed this interaction of A emerged in interaction with a single partner.

partner authentication: if a partner A accepts a session as a session withparty C, then A indeed has been talking with C until this moment (maybe with somebody playing man-in-the-middle, but only passively). Partner authentication might be mutual or one-sided. In case of PACE, there is one-sided authentication of an eID.

owner's consent: eID is used only when the user agrees and with the terminal chosen by the user.

proof non-transferability: a party A interacting with a party B cannot prove against a third party C that it is interacting with B, and cannot authenticate in this way the data received from B. This should be understood that executing the protocol does not provide additional cryptographic evidence over the data mentioned in the data confidentiality condition.

Case study: KEA

- Diffie-Hellman based key-exchange protocol, mutual authentication for the parties
- developed by NSA, declassified in 1998, no security analysis
- attacked in 2005, Lauter, Mityagin, extension KEA+ proposed, security proven by reduction proofs
- naive protocol:
  - party A chooses x at random and sends to B:
    - g^x and $\text{sign}_A(g^x, B)$
  - party B chooses y at random and sends to BA:
    - g^y and $\text{sign}_B(g^y, A)$
  - both: verify the signature, compute $g^{x \cdot y}$ as for DH protocol
  - attack:
    - if ephemeral x of A from communication between A and B revealed, then ...
    - the adversary resends g^x and $\text{sign}_A(g^x, B)$ to C and can impersonate A as he can compute the session key

- KEA:
  - A and B hold, respectively, the keys: private a and b, and public keys g^a and g^b
  - A and B select ephemeral secret keys x and y at random and exchange g^x and g^y
  - each party computes $g^{a \cdot y}$ and $g^{b \cdot x}$ (static DH protocol)
  - session key computed as $F(g^{a \cdot y} \text{ xor } g^{b \cdot x})$ (just like Blake-Wilson, D. Johnson, and A. Menezes: Hash($g^{a \cdot y}$, $g^{b \cdot x}$))
- Unknown Key Share (UKS) – a formal attack on KEA:
  - Mallet registers the same key g^a as Alice
  - Alice starts a session with Bob but session intercepted by Mallet
  - Mallet starts a session with Bob as Mallet
  - Mallet forwards the values g^x and g^y
  - therefore Alice and Bob compute the same session key
  - Mallet corrupts one sesson and get a session key for the second one - contradicting AKE security
- KEA+
  - session key computed as $F(g^{a \cdot y}, g^{b \cdot x}, A, B)$
- KEA+C
  - keys as for KEA
  - A chooses x at random and sends g^x
  - B chooses y at random, computes $L = \text{Hash}(g^{a \cdot y}, g^{b \cdot x}, A, B)$
  - B responds with g^y and $\text{MAC}_L(0)$
  - A computes L, checks $\text{MAC}_L(0)$and responds with $\text{MAC}_L(1)$
  - B checks $\text{MAC}_L(1)$
- security properties:
  - AKE (Authenticated Key Exchange)-
    - the adversary controls all communication
    - the adversary can corrupt some of the parties.
    - the adversary must select an uncorrupted session called a test session and then he is given a challenge, which is either the session key of the test session or a randomly selected key.
    - the adversary wins if can distinguish between these 2 cases.

- PFS (Perfect Forward Security):
  - AKE experiment
  - the adversary can corrupt a party A (reveal the long-term secret key),
  - test session: a session of A occurred before corrupting A
- KCI (Key Compromise Impersonation)
  - the adversary gets a long-term secret key of A
  - attempt to impersonate as other party to A
  - of course, the adversary can impersonate A to anyone
- advantage of the adversary A running algorithm $\mathcal{A}$:
  $|\Pr(\mathcal{A}(\text{data}, \text{real key}) = 1) - \Pr(\mathcal{A}(\text{data}, \text{random key}))|$
  the advantage should be "negligibly small"

- reduction proofs:
  - → assume that that there is an advarsary A breaking scheme U
  - → choose a cryptographic assumption P
  - → from a case p for P construct a case u for U
  - → show how to run A on u
    - → the environment need not to behave exactly as the scheme U
    - → the difference between real U and the simulated one should be impossible to detect by A
  - → breaking u should lead to breaking p with a fair probability
  - → finally: compute the advantage of the resulting adversary breaking p
- modelling via oracles:
  - atomic actions that can be initiated by the adversary
  - all interactions with the system defined by the oracles
  - specification of adversary's power
- typical oracles:
  - Reveal: reveal ephemeral key
  - Reveal: reveal session key
  - Corrupt: reveal long-time key
  - Execute(A, B): make A and B execute the protocol

- Send: send a message to A and get its reaction (if any) – the messages may come form the protocol, but might be faulty
- Test: a session ends after key establishment, no workload communication (this can be added with the tested key), must concern a *fresh session*
- *fresh session:* exclude situation where for instance via corruptions it is possible to break the session

- AKE for KEA+:
  - reduction via Gap Diffie-Hellman (CDH under assumption that DDH easy)
  - ROM for hash function
  - ways to distinguish between the random from real key: hash value must be asked
  - possibilities for the real key K to appear in the experiment:
    1. Forging: enforce Hash on the tuple $(\mathrm{CDH}(A,Y), \mathrm{CDH}(B,X), A, B)$
    2. Key-replication attack. succeed to create another session with the same "signature" $(\mathrm{CDH}(A,Y), \mathrm{CDH}(B,X), A, B\ )$ and so the same secret key
  - key replication: impossible, since $A' = A$ and $\mathrm{CDH}(A', Y') = \mathrm{CDH}(A, Y)$ implies $Y = Y'$. Similarly $X = X'$ and the sessions are identical
  - forging: case of a single session:
    - adversary observes a single session between honest A and B
    - problem GDH for (X_0, Y_0)
    - the long term key of A chosen as X_0, the response of B chosen as Y_0, the rest executed as in the scheme description
    - learning the key requires asking hash oracle about $(\mathrm{CDH}(X_0, Y_0), g^{b\cdot x}, A, B)$
  - forging the in general case: problem since A involved in many interactions but we do not know the secret key. Idea: replace with a random key
    - all users initialized according to the scheme, except for A
    - Hash simulated by HSim
    - sessions not involving A executed according to the protocol (and HSim)
    - a session $(\boldsymbol{A}, \boldsymbol{C}, \mathrm{role})$:
      - C public key of $\boldsymbol{C}$
      - if $\boldsymbol{A}$ initiator, then it chooses x at random, sends g^x, gets reply Y, session key $\mathrm{HSpec}(1, Y, C^x, A, C)$
      - if $\boldsymbol{A}$ responder, then it waits for X, chooses y at random, sends g^y, gets reply Y, session key $\mathrm{HSpec}(2, X, C^y, C, A)$

- a session $(\boldsymbol{C}, \boldsymbol{A}, \text{role})$:
  - as in the scheme description
  - except for test session where Y_0 sent and the session key not computed
- reveal and corrupt key: as described by the scheme
- $\text{HSim}(Z_1, Z_2, B, C)$ – random oracle on valid signatures
  - if asked before, then repeat the answer
  - check all previous $\text{HSpec}(i, Y, Z, B, C) = v$ and check if $Z = Z_{3-i}$ and $\text{DDH}(X_0, Y, Z_i) = \text{true}$. If yes, then return v.
  - if not found then return random w and remember it
- $\text{HSpec}(i, Y, Z, B, C)$ - random oracle for cases when adversary does not know the secret key of A. For input (Z_1, Z_2, B, C), where $Z_i = \text{CDH}(X_0, Y)$ and $Z_{3-i} = Z$

VII. 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, ...
- catastrophy is already there

TLS and DH real security

mistakes:

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

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

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

VIII. HARDWARE TROJANS

methods of testing:

- functional tests
- internal tests circuitry
- optical inspection (distructive) - can detect modifications on layout level

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

Dopant Trojans

CMOS inverter: (image Wikipedia)

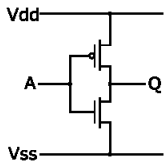


Figure 1.

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)

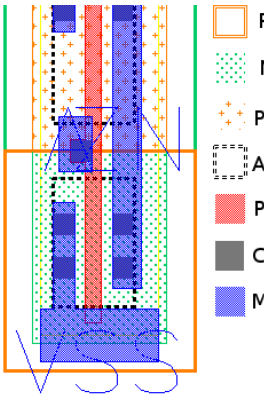
how it works:

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

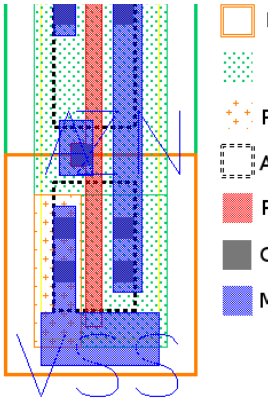
PMOS: in dopant area "holes" (positive) playing the role of conductor, low voltage creates depletion area, high voltage attracts them

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

CMOS inverter in the "bird eye perspective":



Trojan design:



- whatever happens the VDD is connected to the output

Trojan TRNG

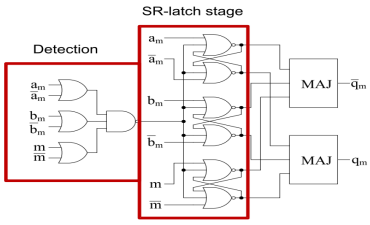
TRNG cosnsists of

- entropy source (physical)
- self test circuit (OHT - inline 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. $c := c \oplus y$
- 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
- 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 by exaustive search

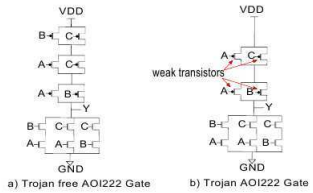
Side channel Trojan:

- 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 gat



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

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
- methods to enable activation in certain areas only
- inserting PUFs, (either randomize as much a s possible - noise over trojan information) or keep deterministic

---

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

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 respone

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)

IX. MODELLING UNSECURITY

attacks:

- hit-and-run
- hit-and-stay
- insider

life-cycle of a solution based on key secrecy:

1. T_0: key created
2. T_1 : forensics-based key non-compromised
3. T_2: key compromise (known to the adversary)
4. T_3 : forensics-based key already compromised
5. T_4: key ceased from operation

[T_1,T_4] is a gray period. Should be as short as possible

Example contermeasure: DSAS framework

archive for signatures,

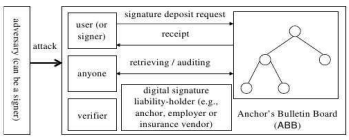


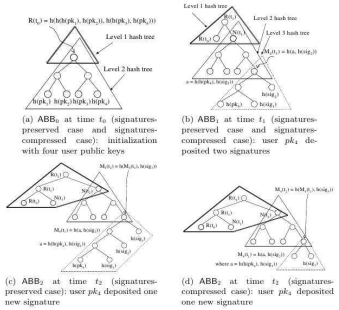
ABB:

— 3 levels of trees

— bottom level: signatures corresponding to one user in a separate tree, leftmost leaf holds public key, the root is a leaf in the level 2 tree

— middle level trees: binary tree for all user, the root is a leaf in top level tree

— top tree has nonleaf nodes corresponding to old roots and old leaves of trees of level 2



(a) ABB_0 at time t_0 (signatures-preserved case and signatures-compressed case): initialization with four user public keys

(b) ABB_1 at time t_1 (signatures-preserved case and signatures-compressed case): user pk_4 deposited two signatures

(c) ABB_2 at time t_2 (signatures-preserved case): user pk_4 deposited one new signature

(d) ABB_2 at time t_2 (signatures-compressed case): user pk_4 deposited one new signature

X. CRYPTOGRAPHIC FORENSICS

how to detect that a card has been cloned:

fail stop signatures: work if the clone created by cryptanalysis and deriving secret key and not by card inspection

- key generation by a trusted party: p, q chosen as for RSA, a - odd integer such that $\frac{p-1}{2a}$ is a prime and $q-1$ and a are coprime
- user chooses secret keys $\mathrm{sk}_1, \mathrm{sk}_2 \in Z_n^2$, public keys: $\mathrm{pk}_1 = \alpha^a$, $\mathrm{pk}_2 = \mathrm{sk}_1^a \bmod n^2$
- signing m: $s := \mathrm{sk}_1 \cdot \mathrm{sk}_2^m$
- verification: $s^a := \mathrm{pk}_1 \cdot \mathrm{pk}_2^m$
- fail-stop idea: there are many solutions for sk_1 and sk_2 (namely a for each)

  a different solution yields $s'^a = s^a \bmod n$, then $s'^a = s^a \bmod q$, $s' = s \bmod q$ since a is invertible modulo q

ctrl-signatures:

actors:

Inspection Authority: IA has a long period secret key k_{master}. For a user U, IA determines the control key $c_U := \mathit{Hash}_1(U, k_{master})$, and a pair of inspection keys: the private key $i_U = \mathit{Hash}_2(U, k_{master})$, and the public key $I_U = g^{i_U}$.
Card Issuer: for a user U, Card Issuer obtains the keys c_U and I_U from IA and installs them in the SSCD issued for U.
Signatories: the SSCD of a user U holds the preinstalled keys c_U and I_U, as well as the private signature key x_U created at random by the SSCD, and the public key $X_U = g^{x_U}$. (Note that the SSCD does not hold the key i_U.)
Certification Authority: CA has standard keys for issuing certificates for the public keys of the users, just as in PKI built according to the X.509 framework.

footprints:

Generating $f_U(k)$ - a hidden footprint for k and user U.
input: I_U, k
$f := \mathit{Hash}_3(I_U^k)$;
output d least significant bits of f

For the inspection procedure carried out by Inspection Authority there is an alternative way for computing $f_U(k)$ (this is essential, since parameter k is present only on the SSCD):

Alternative generation of $f_U(k)$.
input: $i_U, r = g^k$
$f := \mathit{Hash}_3(r^{i_U})$;
output d least significant bits of f

Creating the ith signature by SSCD of user U for message M.
input: a message M
"choose k at random so that $f_U(k) = \rho_U^i$"
proceed with the signing algorithm Sign with
the first signature component $r = g^k$

Inspection

Below we describe inspection of the signature list created by a user U.

1. User U presents a list $S_1, S_2, \ldots, S_t$ of allegedly all signatures created with SSCD of U, where the signatures appear on the list in the order in which they have been created. (If the signing time is included in the signatures, it is not necessary to specify the order of creating signatures.)
2. Apart from the regular verification of each signature S_i, the Inspection Authority checks all footprints. Namely, for each signature $S_j = (r_j, s_j), j \le t$, IA computes the footprint $\omega_j := f_U(r_j)$.
3. If $(\omega_1, \omega_2, \ldots, \omega_t) = (\rho_U^1, \rho_U^2, \ldots, \rho_U^t)$, then inspection result is positive.

XI. COMMUNICATION SECURITY – SSL/TLS

Padding attack (Serge Vaudenay)

Scenario:

- for encryption the plaintext should have the length as a multiply of b
- pad the plaintext with n occurences of n, always pad something
- the resulting padded plaintext $x_1, ..., x_N$ encrypt in CBC mode with IV (fixed or random) and a block cipher:

$$y_1 = \text{Enc}(\text{IV} \oplus x_1), \quad y_i = \text{Enc}(y_{i-1} \oplus x_i)$$

- CBC:
  - efficiency
  - confidentiality limits: if IV fixed one can check that two plaintexts have the same prefix of a given size
  - CBC-MAC has security flaws: m_1 and m_2 augment by extra blocks: due to birthday paradox we might create the same MAC

attack:

- manipulate the ciphertext
- destination node decrypts, can see incorrect padding
- decision: what to do if padding incorrect?
  - → reject: creates padding oracle
  - → proceed: enables manipulation of the data

last word oracle:

- goal: compute $\text{Dec}(y)$
- create an input for padding oracle:
  - $r = r_1 ... r_b$ chosen at random, $c := r|y$
  - oracle call: if $O(c) = \text{valid}$, then $y_b = r_b \oplus 1$ whp
  - recognizing other cases:
    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 result "invalid" then increment i and goto (2)
4. $r_b := r_b \oplus i$
5. for $j = b$ to 2:

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

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

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 $k = j, ..., b$
  2. set $r_1, ..., r_{j-1}$ at random, $j := 0$
  3. $r := r_1 ... r_{j-2}(r_{j-1} \oplus i) r_j ... r_b$
  4. if $O(r|y) = 0$, then $i := i + 1$ and goto 3
  5. output $r_{j-1} \oplus i \oplus (b - j + 2)$

decryption oracle

- block by block
- the only problem with the first block if IV is secret

bomb oracles:

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

other paddings:

- 00....0n instead of $nn....n$ – also vulnerable
- 12....n instead of $nn....n$ – also vulnerable

– <random> n instead of $nn....n$ – last word only, possible detection of padding length 1 (if encrypted twice with fixed IV)

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 longer paddings: send ry_1y_2 where y_1y_2 are the last blocks

– IPSEC: discards message with a wrong padding, no error message

– WTLS: decryption-failed message in clear (!) session not interrupted

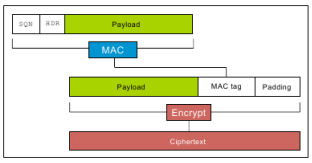
– SSH: MAC after padding (+)

Lucky Thirteen

– concerns DTLS (similar to TLS for UDP connections)

– MAC-Encode-Encrypt paradigm (MEE), MAC is HMAC based

–



– 8-byte SQN, 5-byte HDR (2 byte version field, 1 byte type field, 2 byte length field)

– size of the MAC: 16 bytes (HMAC-MD5), 20 bytes (HMAC-SHA1), 32 bytes (HMAC-SHA-256)

– padding: $p+1$ copies of p, at least one byte must be added

– after receiving: checking the details: padding, MAC, (underflow possible if padding manipulated and removing blindly)

– HMAC of M:

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

to M append the length field encoded

– **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 $M_d||T||$pad, we aim to guess d

→ C – the ciphertext

→ create a ciphertext C' by truncating all parts corresponding to $T||$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 $P^* = \mathrm{Dec}(C^*) \oplus C'$
- 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

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

4 non-IV blocks in plaintext, the last:

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

- 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 $\geq$2 bytes:
  - at least 2 bytes of padding removed, 20 bytes interpreted as MAC, at most 42 bytes left, MAC over at least 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 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)
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 length
- attacker requires malicious Javascript on the client's browser
- 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
  - → 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:

- 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 becouse 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 (Edukacja runs with 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

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. $C := (P \oplus \mathrm{MSB}_{Plen}(S)) || (T \oplus \mathrm{MSB}_{Plen}(S))$

Decryption:

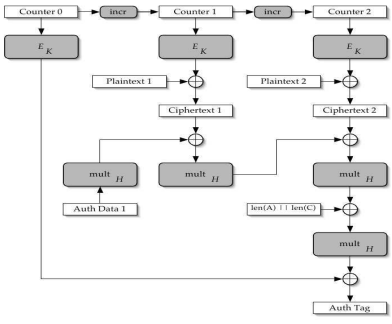
1. return INVALID, if $Clen < Tlen$
2. generate the counter blocks $\mathrm{Ctr}_0, \mathrm{Ctr}_1, ..., \mathrm{Ctr}_m$ for $m = Plen/128$
3. for $j = 0$ to m: $S_j := \mathrm{Enc}_K(\mathrm{Ctr}_j)$
4. $S := S_1 || ... || S_m$
5. $P := \mathrm{MSB}_{Clen}(C) \oplus \mathrm{MSB}_{Plen}(S)$
6. $T := \mathrm{LSB}_{Tlen}(C) \oplus \mathrm{MSB}_{Tlen}(S_0)$
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)

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 $i = 1, ..., n$ (counter computation)
4. $C_i := P_i \oplus \mathrm{Enc}_K(Y_i)$ for $i = 1, ..., n-1$ (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

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

X_i equals:

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

Decryption:

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

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

XII. 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 but:

  i. find two messages that $\text{Hash}(M_0) = \text{Hash}(M_1)$ 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 $\text{Hash}(M_0)$

iii. copy the signature to a certificate based on M_1

- problems: some data in M_0 are to be guessed : sequential number, validity period, other are known in advance: distinguished name, ...

legitimate website certificate		rogue CA certificate
serial number		serial number
issuing CA		issuing CA
validity period		validity period
domain name	chosen prefixes	rogue CA name
		1024 bit RSA public key
		extensions
		"CA=true"
	.....	tumor
2048 RSA public key	collision bits	
	.....	
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, RSA bit in certificate, tumor (ignored part by almost all software) in rogue

    birthday bits make the difference of intermediate hash value fall into a good class

  - then 3 near-collision 512-bit blocks. website $208 + 96 + 3{\cdot}512 = 1840$ bits of RSA modulus. rogue certificate: tumor

  - after collision bits, 2048-1840 = 208 bits needed to complete the RSA modulus of the webpage.

    – continued so that two prime factors:

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

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

→ (purely esthetic reasons: smallest fact 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 chosen prefix MD5: 2^{49} at 2007, 2^{39} in 2009 (computation costly 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,

- 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

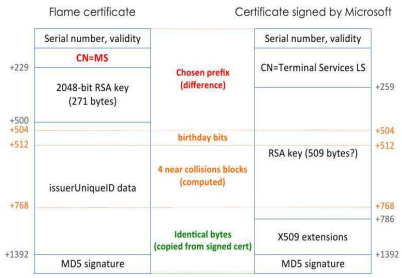
  – victim that gets wpad.dat sets its proxy server to Flame computer (later no sniffing necessary!)

  – Windows updates provided (but must be signed) – main problem!

  – signatures obtained for terminal Services, certificates issued by Microsoft LSRA PA. No Extended Key Usage restrictions – allows code signing, but Microsoft Hydra X.509 extension – 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



MD5 attack draft

MD5:

- padding to the length 448 mod 512 with 10..., then the length of the message as 64 bit
- partition into 512 bit blocks
- IHV_i after block i, consist of four 32-bit numbers a_i, b_i, c_i, d_i. the initial values a_0, b_0, c_0, d_0 are fixed
- $\mathrm{IHV}_i = \mathrm{MD5Compress}(\mathrm{IHV}_{i-1}, M_i)$
- output IHV_N (reformatted)
- compression function:
  - steps 0,...,63 (4 rounds of 16 steps)
  - each step involves modular addition, left rotation, nonlinear function f_t, involves addition constant AC_t and rotation constant RC_t
  - nonlinear function: $f_t(x, y, z) =$

$$\begin{aligned} F(x,y,z) &= (x \wedge y) \oplus (\bar{x} \wedge z) && \text{for } 0 < t < 16 \\ G(x,y,z) &= (z \wedge x) \oplus (\bar{z} \wedge y) && \text{for } 16 \le t < 32 \\ H(x,y,z) &= x \oplus y \oplus z && \text{for } 32 \le t < 48 \\ I(x,y,z) &= y \oplus (x \vee \bar{z}) && \text{for } 48 \le t < 64 \end{aligned}$$

  - w_t composed of message blocks: repetitions occur (important for security):

    W_t equals:

$$\begin{aligned} m_t & \quad \text{for } 0 \le t < 16 \\ m_{(1+5t) \bmod 16} & \quad \text{for } 16 \le t < 32 \\ m_{(5+3t) \bmod 16} & \quad \text{for } 32 \le t < 48 \\ m_{(7t) \bmod 16} & \quad \text{for } 48 \le t < 64 \end{aligned}$$

  - rolling notation: the values kept are Q_t, Q_{t-1}, Q_{t-2}, Q_{t-3} , computed Q_{t+1} and retained $Q_{t+1}, Q_t, Q_{t-1}, Q_{t-2}$. Computation:

$$\begin{aligned} F_t &= f_t(Q_t, Q_{t-1}, Q_{t-2}), \\ T_t &= F_t + Q_{t-3} + \mathrm{AC}_t + W_t \\ R_t &= \mathrm{RL}(T_t, \mathrm{RC}_t) \\ Q_{t+1} &= Q_t + R_t \end{aligned}$$

– final step: (quite important! enables elimination of differences step by step)

$$\text{MD5Compress}(\text{IHV}, B) = (a + Q_{61}, b + Q_{64}, c + Q_{63}, d + Q_{62})$$

Notation:

– primes for second copy

– $\delta X = X - X'$

– ΔX is δX in BSDR notation: a 32-bit word X is defined as $(k_i)_{i=0}^{31}$, where $X = \sum_{i=1}^{31} 2^i \cdot k_i$, and $i \in \{-1, 0, 1\}$

– many such representations, taken the one with maximal number of zeroes

Birthday part

- constructed near-collision blocks (based on Xiaoyun Wang observation): they cannot erase a difference in a, only identical differences can be removed from c and d parts
- assumption: before using near-collision block we adjust so that the differences are $\delta\text{IHV}_n = (0, \delta b, \delta c, \delta c)$
- birthday search: we aim to have a property on 64 bits,
- average number of calls to MD5 compression function $\approx \sqrt{\pi} 2^{32}$
- in the original paper: more conditions (relationship between δb and δc) - time-space trade-off – so that the time for the birthday part and near collision part are of the same order
- the search for collision: usual technique of a pseudorandom walk and storing only characteristic points on a walk
- a collision found in this way is not always useful as the prefix might be the same, number of pairs of near-collision blocks must be small, etc

Near-collision blocks

- basic differential path for a near-collision block:

Table 2 Family of partial differential paths using $\delta m_{11} = \pm 2^{p-10 \bmod 32}$, where $s_0, \ldots, s_{w'} \in \{-1, 0, +1\}$ and $w' = \min(w, 31 - p)$ for a fixed $w \geq 0$

t	δQ_t	δF_t	δW_t	δT_t	δB_t	RC_t
31	$\mp 2^{p-10 \bmod 32}$					
32	0					
33	0					
34	0	0	$\pm 2^{p-10 \bmod 32}$	0	0	16
35 – 60	0	0	0	0	0	·
61	0	0	$\pm 2^{p-10 \bmod 32}$	$\pm 2^{p-10 \bmod 32}$	$\pm 2^p$	10
62	$\pm 2^p$	0	0	0	0	15
63	$\pm 2^p$	0	0	0	0	21
64	$\pm 2^p$ $+ \sum_{i=0}^{w'} s_i 2^{p+21 \text{ b.b } \bmod 32}$					

Note: Interesting values for the parameter w are between 2 and 5.

- overview:

– only δm_{11} not equal to 0. It occurs only at steps 34 (to erase a difference δQ_{31}), however it occurs at step 61 as well

- then the difference propagates to Q_{62}, Q_{63} (affecting the change on c and d) as well as Q_{64} (affecting b), difference obtained

$$\pm\left(0, 2^p + \sum_{\lambda=0}^{w'} s_\lambda \cdot 2^{p+21+\lambda \bmod 32}, 2^p, 2^p\right)$$

- different characteristics for different w' - for large w more differences might be removed but also pbb lower

- starting differences: $\delta c = \sum_i k_i \cdot 2^i$ and $\delta b - \delta c = \sum_i l_i \cdot 2^i$ (expressed as NAFs)

- let $k_i \neq 0$: use differential path with $m_{11} = k_i \cdot 2^{i-10 \bmod 32}$ to eliminate the difference $k_i \cdot 2^i$ in c and d. A side effect: change of δb by

$$k_i 2^i + \sum_{\lambda=i+21}^{i+21+w'} l_\lambda \cdot 2^{\lambda \bmod 32}$$

  in this expression we carefully choose w to get coordinates l_λ

- finally δc=0, but there are some differences in b, say we get $\delta\hat{b} = \sum_{\lambda=0}^{31} e_\lambda l_\lambda 2^\lambda$, where $e_\lambda = 0$ (if the coordinate has been nullified) or $e_\lambda = 1$, the weight of $\delta\hat{b}$ is not higher than the weight of δb

- the differences from δb eliminated as follows:

  - let $\mathrm{NAF}(\delta\hat{b}) = \sum_{\lambda=0}^{31} \hat{l}_\lambda \cdot 2^\lambda$. Choose j such that $\hat{l}_j \in \{-1, 1\}$ and $j - 21 \bmod 32$ is minimal

  - Then the difference $\sum_{i=j}^{j+w'} \hat{l}_i 2^i$ with $w' = \min(w, 31 - (j - 12 \bmod 32))$can be eliminated from $\delta\hat{b}$ with $m_{11} = 2^{j-31 \bmod 32}$

  - side effect: a new difference $2^{j-21 \bmod 32}$ in b, c and d

  - the side effect eliminated using $\delta m_{11} = 2^{31} \bmod 32$

  - result: a new difference vector $(0, \delta\bar{b}, 0, 0)$with weight of $\mathrm{NAF}(\delta\bar{b})$ smaller than the weight of $\mathrm{NAF}(\delta b)$

- construction of near-collision blocks:

  - the key is so-called *differential path* for MD5Compress, for IHV, IHV' and δb

  - $\delta F_t = f_t(Q'_t, Q'_{t-1}, Q'_{t-2})$-$f_t(Q_t, Q_{t-1}, Q_{t-2})$

    $\delta T_t = \delta F_t + \delta Q_{t-3} + \delta W_t$

    $\delta R_t = \mathrm{RL}(T'_t, \mathrm{RC}_t) - \mathrm{RL}(T_t, \mathrm{RC}_t)$

    $\delta Q_{t+1} = \delta Q_t + \delta R_t$

  - δF and δR cannot be uniquely determined given $(\delta Q_t, \delta Q_{t-1}, \delta Q_{t-2})$ and δT_t

  - differentials: corresponding to each step in the rolling notation. We start from

IHV $=(Q_{-3}, Q_0, Q_{-1}, Q_{-2})$ and IHV' $=(Q'_{-3}, Q'_0, Q'_{-1}, Q'_{-2})$ and δB and leading to $(\delta Q_{61}, \delta Q_{62}, \delta Q_{63}, \delta Q_{64})$

- bitconditions:

Table 5 Boolean function bitconditions

$q_t[i]$	*Condition on* $\{Q_t[i], Q'_t[i]\}$	*Direct/indirect*	*Direction*
0	$Q_t[i] = Q'_t[i] = 0$	Direct	
1	$Q_t[i] = Q'_t[i] = 1$	Direct	
^	$Q_t[i] = Q'_t[i] = Q_{t-1}[i]$	Indirect	Backward
v	$Q_t[i] = Q'_t[i] = Q_{t+1}[i]$	Indirect	Forward
!	$Q_t[i] = Q'_t[i] = \overline{Q_{t-1}[i]}$	Indirect	Backward
y	$Q_t[i] = Q'_t[i] = \overline{Q_{t+1}[i]}$	Indirect	Forward
n	$Q_t[i] = Q'_t[i] = Q_{t-2}[i]$	Indirect	Backward
w	$Q_t[i] = Q'_t[i] = Q_{t+2}[i]$	Indirect	Forward
#	$Q_t[i] = Q'_t[i] = \overline{Q_{t-2}[i]}$	Indirect	Backward
h	$Q_t[i] = Q'_t[i] = \overline{Q_{t+2}[i]}$	Indirect	Forward
?	$Q_t[i] = Q'_t[i] \wedge (Q_t[i] = 1 \vee Q_{t-2}[i] = 0)$	Indirect	Backward
q	$Q_t[i] = Q'_t[i] \wedge (Q_{t+2}[i] = 1 \vee Q_t[i] = 0)$	Indirect	Forward

- constructing differential paths:
  i. for steps 1-11: forward
  ii. for steps 64-16 backwards
  iii. then try to fill the gap between 11 and 16
- very subtle case specific techniques

XIII. 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 byte

- S cache sets, each consisting of W cache lines
- when a cache miss occurs, then a memory block is copied into one of cache lines evicting its previous contents
- a memory block of 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 typically the fastest
- key expansion: round zero: simply the key bytes directly, other rounds: key expansion reversable (details irrelevant for the attack)
- fast implementation based on 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:

– $\delta = B/\text{entrysize of lookup table}$, typically: entrysize=4bytes, $\delta = 16$

– 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 request to the same memory block of the lookup table

– $Q_k(p, l, y) = 1$ 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 $i = 0, ..., 15$
  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 $T_l\Big(x_i^{(0)}\Big)$
  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
    - probability that none of them accesses block y for T_l is $\Big(1 - \frac{\delta}{256}\Big)^{35} \approx 0.104$ for $\delta = 16$
  v. few dozens of samples required to find a right candidate for $\langle \bar{k_i} \rangle$
  vi. together we determine $\log(256/\delta) = 4$ bits of each byte of the key
  vii. no more possible for the first round, not enough to start brute force (still 64 bits to be found!)
  viii. in reality more samples needed due to noise in measurements $M_k(p, l, y)$ and not $Q_k(p, l, y)$
- **two-round attack:** the second round attack because of the missing bits
  i. exploiting equations derived from Rijndeal specification:

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

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

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

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

  where s stands for the Rijndael Sbox, and $\bullet$ means multiplication in the field with 256 elements
  ii. lookup for $T_2\Big(x_2^{(1)}\Big)$:
    - $\langle k_0 \rangle, \langle k_5 \rangle, \langle k_{10} \rangle, \langle k_{15} \rangle, \langle k_2 \rangle$ already known
    - low level bits of $\langle k_2 \rangle$ influence only low bits of $x_2^{(1)}$ so not important for cahce access pattern
    - the upper bits of $x_2^{(1)}$ can be determined after guessing low bits of k_0, k_5, k_{10}, k_{15}: there are δ^4 possibilities ($=16^4$)
    - a correct guess yields a lookup in the right place

- an incorrect guess: some $k_i \neq \bar{k_i}$ so

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

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

  differential properties of AES studied for AES competition:

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

  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 $\left(1 - \frac{\delta}{256}\right)^{38}$

- 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, otehrwise 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 known δ positions

  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 - unknown so how they are loaded to the cache 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}

roughly number of bits learnt: $4 \cdot (4 \cdot 4 - \log 4!) \approx 4 \cdot (16 - 3.17) \approx 51$ 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, 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, y initialized with plaintext $\oplus\, x$

→ modification of x:

– embarassing simple attack:

→ timing of execution depends on $k[13] \oplus$ plaintext[13]:

– try many plaintexts

– collect statistics for each byte as plaintext[13]

– the maximum occurs for z

– the maximum corresponds to a fixed value for $k[13] \oplus$ plaintext[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 (soemtimes 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 inforamation which part has been used!

- process blocking: again, deep changes in OS
- disable cahce 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 are 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

---

XIV. KLEPTOGRAPHY - CASE STUDY: KLEPTOGRAPHIC RSA KEY GENERATION

The idea of the attack

The aims of the attack:

- The public RSA modulus n will contain a backdoor.
- The backdoor shall be visible only by the author of the implementation of the RSA-key generation procedure.
- For others the keys generated shall look trully randomly (even if they suspect the attack) – there should be no statistical properties indicating implementation of the attack.
- The backdoor is based on asymmetric keys – opening the implementation does give no additional power for e-forensic analyst.

Techniques used:

- The backdoor is based on the ephemeral-static Diffie-Hellman protocol: the asymmetric key (usually called public) Y_a is hidden inside the contaminated implementation, the corresponding private key x_a is held by the author of the implementation.
- The ephemeral key $x(Q_a)$ of length ℓ is transferred inside the upper part of the RSA modulus, and **is uniformly distributed over the bitstrings of length** ℓ. The key is an x coordinate of elliptic curve point Q_a.
- To ensure the uniform distribution twisted elliptic curves defined over binary field are used. Two base procedures are defined: GenDHParamAndDHSecret(), RecoverDHSecret($x(Q_a), x_0, x_1$) – see below.
- The result of the ephemeral-static DH protocol is used as a seed of the pseudorandom number generator, from which the upper half of one of the primes (namely p_1) is derived.
- Reconstructing the upper half of one of the primes the author of the implementation may utilize Algorithm [1], [2] to factorize n.
- The attack requires that both prime factors are congruent to $3 \bmod 4$, hence it is assumed that a quarter of the keys generated by the malicious implementation shall be infected.

The Main Tool: EC Diffie-Hellmann with uniformly distributed binary strings as ephemeral public keys

Twisted Elliptic Curves over binary field

Let $E_{a,b}$ denote an elliptic curve defined over a binary field $\mathbb{F}_{2^\ell}$, given by equation

$$y^2 + xy = x^3 + ax^2 + b. \tag{1}$$

For any $E_{a,b}$ there are points $P_{a,1}, P_{a,2} \in E_{a,b}$ such that $E_{a,b} = \langle P_{a,1}\rangle \times \langle P_{a,2}\rangle$ and $\mathrm{ord}P_{a,2} | \mathrm{ord}P_{a,1}$. If $\mathrm{ord}P_{a,1}$ has a single large prime factor q_a, then some point G_a of order q_a can be determined, as well as some $P'_{a,1}$ such that $\langle P_{a,1}\rangle = \langle G_a\rangle \times \langle P'_{a,1}\rangle$. From now on we assume that both $\mathrm{ord}P'_{a,1}$, $\mathrm{ord}P_{a,2}$ are small. Under these assumptions Algorithm 2 from [3], which finds points $P_{a,1}$, $P_{a,2}$, is fast (it must factorize $\#E_{a,b}$, the number of points of $E_{a,b}$).

Mallet will use a pair of twisted curves $E_{0,b}$, $E_{1,b}$ over $\mathbb{F}_{2^\ell}$, where ℓ is prime (in [4], $\ell = 163$). Clearly, point $(0, \sqrt{b})$ belongs to both curves. At the same time, for each nonzero x there are two *different* points ((x, y) and $(x, x+y) = -(x, y)$) on exactly one of the curves $E_{0,b}$, $E_{1,b}$. Hence each $x \in \mathbb{F}_{2^\ell}$ occurs twice in $E_{0,b} \cup E_{1,b}$. On the other hand, apart from points (x, y) satisfying (1) there is also a point $\mathcal{O}_{a,b} \in E_{a,b}$, called "point at infinity", which is the neutral element of group $E_{a,b}$. Consequently, the number $2 \cdot 2^\ell$ corresponds to the number of points in the set $(E_{0,b} \setminus \{\mathcal{O}_{0,b}\}) \cup (E_{1,b} \setminus \{\mathcal{O}_{1,b}\})$.

Altogether, after obtaining $E_{0,b}$, $E_{1,b}$ and points G_0, $P'_{0,1}$, $P_{0,2}$, G_1, $P'_{1,1}$, $P_{1,2}$, any point $Q_a \in E_{a,b}$, for $a \in \{0, 1\}$, might be expressed as

$$Q_a := [u]G_a + [v_1]P'_{a,1} + [v_2]P_{a,2} \tag{2}$$

for $0 \le u < \mathrm{ord}G_a$, $0 \le v_1 < \mathrm{ord}P'_{a,1}$, $0 \le v_2 < \mathrm{ord}P_{a,2}$, where $[k]B$ stands for point B multiplied by scalar k. Note that according to the choice of $E_{a,b}$ described above, $\mathrm{ord}P_{a,2} | \mathrm{ord}P'_{a,1}$.

On the other hand, having received the nonzero x-coordinate $x(Q_a)$ of Q_a, Mallet may determine $a \in \{0, 1\}$ ($a = \mathrm{Tr}_{\mathbb{F}_{2^\ell}/\mathbb{F}_2}(x(Q_a) + b \cdot (x(Q_a))^{-2})$ for odd ℓ). Then he may calculate any of the two y-coordinates for $x(Q_a)$. Next he multiplies the resulting point $\pm Q_a$ by a scalar k such that $k = 1 \bmod \mathrm{ord}G_a$ and $k = 0 \bmod \mathrm{ord}P'_{a,1}$. The outcome is one of the points $\pm[u]G_a$. If Mallet has received $x(Q_a) = 0$, then he knows that this is the x-coordinate of the point $Q_a = (0, \sqrt{b})$ of both curves, and that the order of Q_a is equal 2 (this is because $(0, \sqrt{b}) = -(0, \sqrt{b})$). Hence u in $[u]G_a$ must be equal 0.

According to [4], Mallet chooses curves $E_{0,b}$, $E_{1,b}$ with orders $4q_0$, $2q_1$, respectively, where q_0 and q_1 are prime (so the conditions imposed are quite sharp). Consequently, $|E_{0,b} \setminus \{\mathcal{O}_{0,b}\}| = 4q_0 - 1$, $|E_{1,b} \setminus \{\mathcal{O}_{1,b}\}| = 2q_1 - 1$. The point of order 2 on $E_{1,b}$ is obviously the point $P'_{1,1} = (0, \sqrt{b})$. Point $P'_{0,1} = (\sqrt[4]{b}, \sqrt{b})$ is a point of order 4 on $E_{0,b}$. It follows that points $P_{a,2}$ disappear from (2). Next, Mallet finds points G_0, G_1 such that $G_a \in E_{a,b}$ and $\mathrm{ord}G_a = q_a$ for $a = 0, 1$. To finalize his public key generation Mallet chooses $x_a \in \{2, \ldots, q_a - 1\}$ uniformly at random and assigns $Y_a = [x_a]G_a$ for $a = 0, 1$. Finally, definition of $\mathbb{F}_{2^\ell}$, value b, and (G_0, G_1, Y_0, Y_1) or their x-coordinates, stand for his public key (are hidden inside the implementation).

GenDHParamAndDHSecret()

To encode a message, the contaminated software sets $a = 0$ with probability $\frac{4q_0-1}{2^{\ell+1}}$, and $a = 1$ with probability $\frac{2q_1-1}{2^{\ell+1}}$ (neutral elements $O_{a,b}$ will not be generated). Then it selects $u \in \{0, 1, \ldots, q_a - 1\}$ uniformly at random, calculates $[u]G_a$, $[u]Y_a$, chooses $v \in \{0, \ldots, 2^{2-a} - 1\}$ uniformly at random and assigns

$$Q_a := [u]G_a + [v]P'_{a,1}. \tag{3}$$

If $u = v = 0$ the selection is repeated from the beginning. The crucial point is that this procedure ensures uniform distribution of values $x(Q_a)$ in the set $\{0, \ldots, 2^\ell - 1\}$. The 163 bits of the x-coordinate $x(Q_a)$ will be transmitted, and $\mathcal{R}(H(x([u]Y_a)))$ will be used by the device as a source of random data.

RecoverDHSecret($x(Q_a), x_0, x_1$):

To decode the secret, Mallet gathers the bits of $x(Q_a)$ at first, and then reconstructs $\pm[u]G_a$. Using his private key x_a he determines $\pm[u]Y_a$, and since the x-coordinate is independent of the sign of the point, he obtains $x([u]Y_a)$.

RSA key generation klepto-procedure [5]

$N/2$ is the size in bits of each prime factor of n, and e is the RSA exponent (it may be $2^{16} - 1$ for example). π is a permutation defined over the set $\{0, 1\}^\theta$

Procedure GetPrimes$_{N,e}(x(Q_a), x([u]Y_a))$:
Input: $x(Q_a), x([u]Y_a) \in \{0, 1\}^\ell$
Output: A pair of acceptable RSA primes (p_1, q_1)

1. set $p_1 = \text{GenPrimeWithOracle}(x([u]Y_a), N/2, e)$
2. choose $s_0 \in_R \{0, 1\}^{\theta-\ell}$
3. compute $t = \pi(s_0||x(Q_a))$
4. choose $r_2 \in_R \{0, 1\}^{N-\theta}$
5. set $n_c = (t||r_2)$ (so n_c has length N)
6. solve for (q_1, r_c) in $n_c = q_1 p_1 + r_c$ (i.e., find quotient q_1)
7. if (IsAcceptablePrime$(e, N/2, q_1)$ = false) then goto step 2
8. output (p_1, q_1) and halt

References

[1] Coppersmith, D.: Finding a small root of a bivariate integer equation; factoring with high bits known. In Maurer, U.M., ed.: EUROCRYPT. Volume 1070 of LNCS., Springer (1996) 178–189. Available from: http://dsns.csie.nctu.edu.tw/research/crypto/HTML/PDF/E96/178.PDF [retrieved on December 13, 2006] (Cited on page 1.)

[2] Coron, J.S.: Finding small roots of bivariate integer polynomial equations revisited. In Cachin, C., Camenisch, J., eds.: EUROCRYPT. Volume 3027 of LNCS., Springer (2004) 492–505. Available from: http://www.eleves.ens.fr/home/coron/publications/bivariate.pdf [retrieved on December 13, 2006] (Cited on page 1.)

[3] Miller, V.S.: The Weil pairing, and its efficient calculation. J. Cryptology **17**(4) (2004) 235–261. Available from: http://dx.doi.org/10.1007/s00145-004-0315-8 [retrieved on December 13, 2006] (Cited on page 2.)

[4] Möller, B.: A public-key encryption scheme with pseudo-random ciphertexts. In Samarati, P., Ryan, P.Y.A., Gollmann, D., Molva, R., eds.: ESORICS. Volume 3193 of LNCS., Springer (2004) 335–351. Available from: http://www.bmoeller.de/pdf/pke-pseudo-esorics2004.pdf [retrieved on December 13, 2006] (Cited on pages 2 and 3.)

[5] Young, A., Yung, M.: A space efficient backdoor in RSA and its applications. In Preneel, B., Tavares, S.E., eds.: Selected Areas in Cryptography. Volume 3897 of LNCS., Springer (2005) 128–143 (Cited on page 3.)

XV. PKI STANDARDS

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

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

- 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.*"

RFC 3820: Proxy Certificate Profile

- certificates for delegation of authentication within PKI by the end entity (user),
- proxies of the user, solving single-sign on and delegation problem
- restricted proxies - for security reasons
- unique identity for proxies - because of policies,
- derive a new identity from existing identity of the end entity (the simplest approach)
- properties:

    1. *It is signed by either an X.509 End Entity Certificate (EEC), or by another PC. This EEC or PC is referred to as the Proxy Issuer (PI).*
    2. *It can sign only another PC. It cannot sign an EEC.*
    3. *It has its own public and private key pair, distinct from any other EEC or PC.*
    4. *It has an identity derived from the identity of the EEC that signed the PC. When a PC is used for authentication, in may inherit rights of the EEC that signed the PC, subject to the restrictions that are placed on that PC by the EEC.*
    5. *Although its identity is derived from the EEC's identity, it is also unique. This allows this identity to be used for authorization as an independent identity from the identity of the issuing EEC, for example in conjunction with attribute assertions as defined in [i3].*
    6. *It contains a new X.509 extension to identify it as a PC and to place policies on the use of the PC. This new extension, along with other X.509 fields and extensions, are used to enable proper path validation and use of the PC.*

- creation and use of proxy certificate:

    1) A new public and private key pair is generated.

2) That key pair is used to create a request for a Proxy Certificate that conforms to the profile described in this document.

3) A Proxy Certificate, signed by the private key of the EEC or by another PC, is created in response to the request. During this process, the PC request is verified to ensure that the requested PC is valid (e.g., it is not an EEC, the PC fields are appropriately set, etc).

When a PC is created as part of a delegation from entity A to entity B, this process is modified by performing steps #1 and #2 within entity B, then passing the PC request from entity B to entity A over an authenticated, integrity checked channel, then entity A performs step #3 and passes the PC back to entity B.

Path validation of a PC is very similar to normal path validation, with a few additional checks to ensure, for example, proper PC signing constraints.

- features:
  - ease of integration with PKI: X509 as authentication method recognized in software, patha validation as in X509,
  - ease of use: smartcard used once (often more secure)
  - key hidden: not like proxy signature schemes form literature

RFC 2693 SPKI Certificate Theory

experimental, SPKI Working Group, authorization rather than authentication.

- ACL: an Access Control List: a list of entries, "list of root keys" that may start certification chain for a resource, ACL not signed
- CERTIFICATE: signed, gives the rights. contains: Issuer, Subject, [validity conditions, authorization, delegation information]. categories:
  - ID (mapping ⟨name,key⟩),
  - Attribute (mapping ⟨authorization,name⟩),
  - Authorization (mapping ⟨authorization,key⟩)

  rights transferable or not
- ISSUER: the signer of a certificate
- KEYHOLDER: entity that controls a given private key.
- PRINCIPAL: a cryptographic key, capable of generating a digital signature.
- SPEAKING: A principal "speaks" by means of signed messages ("speak for" the Keyholder)
- SUBJECT: thing empowered by a certificate or ACL entry: key, name, set of keys
- S-EXPRESSION: LISP- like parenthesized expression, no empty list, 1st element is a "type" string

- THRESHOLD SUBJECT: K out of N threshold scheme. only $1/K$ power to a single Subject, K of them necessary to get the rights

Name Certification

- classical PKI: biding names and public keys
- PGP: key rings
- SPKI: rethinking global names,
  - not much added value for security
  - unique in local domain,
  - global directions dangerous
  - identifiers should be random of a proper length

Inescapable Identifiers

inescapable identifiers: e.g. from ID cards, commertial CA: disputable

Local Names

SDSI 1.0: how to use local names globally .

Basic SDSI Names

SDSI 2.0 name is an S-expression with two elements: the word "name" and the chosen name.

george: (name fred)

name "fred" in the name space defined by george.

2.6.2 Compound SDSI Names

If fred defines a name

fred: (name sam)

and george defined fred, then george referes to sam as:

george: (name fred sam)

2.7 Sources of Global Identifiers

- public keys
- hash of public key

2.8 Fully Qualified SDSI Names

- name space defined by public key
- chain of names in a name space

– certificate: CA's public key is the name space

2.9 Fully Qualified X.509 Names

X.509: (name ⟨root⟩key ⟨leaf⟩name)

(name ⟨root⟩key ⟨CA(1)⟩ ⟨CA(2)⟩ ... ⟨CA(k)⟩ ⟨leaf⟩name)

2.10 Group Names

– more than one key per name admitted

– this might be a group

3.1 Attribute Certificates

X.9.57:

authorization → name → key

X.509v3 Extensions

authorization → name

authorization → key

SPKI Certificates:

authorization → key

or

authorization → key → name (the name irrelevant for security)

3.4 ACL Entries

SPKI ACL grants authorization to names.

– like attribute certificate,

– not signed since local

– as local data need not to be standardized

4. Delegation

ability to delegate authorizations from one person to another without bothering the owner of the resource(s) involved.

a simple permission (e.g., to read some file) or issue the permission to delegate that permission

issues:

– to limit depth of delegation

– separating delegators from those who can exercise the delegated permission

4.1 Depth of Delegation

no control, boolean control and integer control

4.1.1 No control

free delegation, no limitations

4.1.2 Boolean control

boolean control to specify an inability to delegate

e.g. export restrictions

4.1.3 Integer control

depth up to k

4.3 Delegation of Authorization vs. ACLs

flexibility in delegation, mimicking real life processes

5. Validity Conditions

- optional
- traditional: not-before, not-after
- online tests: CRL, re-validation, one-time
- dependent on the issuer!

5.1 Anti-matter CRLs

traditional CRL have non-deterministic dissemination proces – **such CRL excluded in SPKI**

5.2 Timed CRLs

result must be deterministic. conditions:

1. The certificate must list the key (or its hash) that will sign the CRL and may give one or more locations where that CRL might be fetched.

2. The CRL must carry validity dates.

3. CRL validity date ranges must not intersect. That is, ***one may not issue a new CRL to take effect before the expiration of the CRL currently deployed****.*

5.3 Timed Revalidations

CRLs are negative statements.

Revalidation reverses the decision.

Process must be deterministic

5.4 Setting the Validity Interval

risk managements determines the period

5.5 One-time Revalidations

Validity intervals of length zero are not possible.

For those who want to set the validity interval to zero, SPKI defines a one-time revalidation.

no lifetime beyond the current authorization computation. One applies for this on-line, one-time revalidation by submitting a request containing a nonce. That nonce gets returned in the signed revalidation instrument, in order to prevent replay attacks.

5.6 Short-lived Certificates

5.7.2 Rivest's Reversal of the CRL Logic

validity condition model is flawed because it assumes that the issuer (or some entity to which it delegates this responsibility) decides the conditions under which a certificate is valid. – like for military model

in the commercial space, the verifier takes the risk. It should therefore be the verifier who decides what level of assurance he needs before accepting a credential.

not reflected in the SPKI structure definition.

6. Tuple Reduction

way of processing the information

automatic verification of certificates, path recognition, threshold issues, cooperation with PGP, X.509, ...

7. Key Management

keys not revoked, certificates with a limited lifetime

suicide note+ health certificate : not in the RFC

RFC 3647: Certificate Policy and Certification Practices Framework

- certificate policies or certification practice statements
- X.509v3 certificate: optional field declaring certificate policies that apply to that certificate
- CP: "a named set of rules that indicates the applicability of a certificate to a particular community and/or class of applications with common security requirements."
- CPS: "detailed description of the practices followed by a CA in issuing and otherwise managing certificates ... published by or referenced by the CA"
- not always a part of agreement, but jhelp to make the decision by the "relying party'
- notions:
  - Activation data - Data values, other than keys, that are required to operate cryptographic modules and that need to be protected
  - Authentication - The process of establishing that individuals, organizations, or things are who or what they claim to be.
  - Identification - The process of establishing the identity of an individual or organization:

    (1) establishing that a given name of an individual or organization corresponds to a real-world identity of an individual or organization, and

    (2) establishing that an individual or organization applying for or seeking access to something under that name is, in fact, the named individual or organization.

  - PKI Disclosure Statement (PDS) - supplements a CP or CPS by disclosing critical information about the policies and practices of a CA/PKI.

  - Policy qualifier - Policy-dependent information that may accompany a CP identifier in an X.509 certificate. (e.g. URL)

- categories:

  - ”applicability of a certificate to a particular community"

  - ”applicability of a certificate to a . . . class of application with common security requirements."

- X.509 Certificate Fields

  - Certificate Policies extension: list of CPs applicable

  - Policy Mappings extension: own policies defined as equivalent to external policies

  - Policy Constraints extension: flag to impose the same restrictions on the rest of the certification path, flag to disable further policy mapping

- Policy qualifiers: text

- CPS

  CPS Summary (or CPS Abstract) - only some data published

- differences between CP and CPS:

  - CP: states requirements and standards imposed by the PKI. CP establishes what participants must do.

  - CPS: how a CA and other participants implement procedures and controls to meet the requirements from CP.

  - scope of CP: possibly many CAs, CPS: only one CA

- document framework:

  - Introduction

  - Overview

  - Document Name and Identification (ASN.1 object identifier)

  - PKI Participants: CA, Registration Authorities, Subscribers, Relying Parties, other

  - Certificate usage: Certificate Usage

    This subcomponent contains:

    * A list or the types of applications for which the issued certificates are suitable,

* A list or the types of applications for which use of the issued certificates is prohibited.

- Policy Administration: who responsible
- Definitions and Acronyms
- Publication and Repository Responsibilities: who, how often, access information, ...
- Identification and Authentication:
  - naming conventions
  - Initial Identity Validation

    e.g.

    ** Identification and authentication requirements for an individual subscriber or a person acting on behalf of an organizational subscriber or participant (CA, RA, in the case of certificates issued to organizations or devices controlled by an organization, the subscriber, or other participant),(5) including:*

    ** Type of documentation and/or number of identification credentials required;*

    ** How a CA or RA authenticates the identity of the organization or individual based on the documentation or credentials provided;*

    ** If the individual must personally present to the authenticating CA or RA;*

    ** How an individual as an organizational person is authenticated, such as by reference to duly signed authorization documents or a corporate identification badge.*

  - what not checked
  - what needed for re-key requests
  - what needed for revocation
- Certificate Life-Cycle
  - Certificate Application
  - Certificate Application Processing
  - Certificate Issuance
  - Certificate Acceptance
  - Key Pair and Certificate Usage
  - Certificate Renewal
  - Certificate Re-key
  - Certificate Modification
  - Certificate Revocation and Suspension
  - Certificate Status Services

- End of Subscription
- Key Escrow and Recovery

- Management, Operational, and Physical Controls
  - Physical Security Controls
  - Procedural Controls
  - Personnel Security Controls
  - Audit Logging Procedures:

    Types of events recorded, such as certificate lifecycle operations, attempts to access the system, and requests made to the system;

    ** Frequency with which audit logs are processed or archived, for example, weekly, following an alarm or anomalous event, or when ever the audit log is n% full;*

    ** Period for which audit logs are kept;*

    ** Protection of audit logs:*

    - Who can view audit logs, for example only the audit administrator;

    - Protection against modification of audit logs, for instance a requirement that no one may modify or delete the audit records or that only an audit administrator may delete an audit file as part of rotating the audit file; and

    - Protection against deletion of audit logs.

    ** Audit log back up procedures;*

  - Records Archival: * Types of records that are archived, for example, all audit data, certificate application information, and documentation supporting certificate applications;

    * Retention period for an archive;

    * Protection of an archive:

    - Who can view the archive

    - Protection against modification of the archive

    - Protection against deletion of the archive;

    - Protection against the deterioration of the media on which the archive is stored

    - Protection against obsolescence of hardware, operating systems, and other software

    * backup procedures;

    * Requirements for time-stamping;

    * internal or external;

    * Procedures to obtain and verify archive information (e.g. two independent copies)

  - Key Changeover

    - Compromise and Disaster Recovery

- Compromise and Disaster Recovery

  * listing of the applicable incident and compromise reporting and handling procedures.

  * The recovery procedures used if computing resources, software, and/or data are corrupted or suspected to be corrupted.

  * The recovery procedures used if the entity key is compromised.

  * The entity's capabilities to ensure business continuity following a natural or other disaster.

- CA and RA Termination

- Technical Security Controls:

  - Key Pair Generation and Installation:

    1. Who generates the entity public, private key pair? How?
    2. secure private key delivery
    3. public key presentation to the certification authority
    4. delivery of CA public keys
    5. key size
    6. public key parameters, who checks the quality of the parameters?
    7. purpose of the keys, restrictions of use (X.509 certificates with flags)

  - Private Key Protection and Cryptographic Module Engineering Controls

    - standards
    - multiparty control
    - escrow
    - key back up, archive
    - key export from crypto device?
    - how stored in the device
    - how to activate private key
    - how to deactivate the private key
    - how to destroy private key
    - cryptographic module boundary, input/output, roles and services, finite state machine, physical security, software security, operating system security, algorithm compliance, electromagnetic compatibility, and self tests.

  - Activation Data (lifecycle, protection,...)

    - Computer Security Controls
    - Life Cycle Security Controls
    - Network Security Controls (firewalls, ...)
    - Time-Stamping

- Certificate and CRL Profiles
    - * Version number(s) supported;
      * Certificate extensions populated and their criticality;
      * Cryptographic algorithm object identifiers;
      * Name forms used for the CA, RA, and subscriber names;
      * Name constraints used and the name forms used in the name constraints;
      * Applicable CP OID(s);
      * Usage of the policy constraints extension;
      * Policy qualifiers syntax and semantics; and
      * Processing semantics for the critical CP extension.
- CRL Profile

  * Version numbers supported for CRLs; and

  * CRL and CRL entry extensions populated and their criticality.
- OCSP Profile
- Compliance Audit and Other Assessment
    - topics covered
    - frequency
    - identity and qualifications of auditors
    - degree of independance
    - audit results and follow-up
    - who could see the assesment
- other business and legal matters
- fees
- financial responsibility
- Confidentiality of Business Information
- Privacy of Personal Information

- Intellectual Property Rights
- Representations and Warranties
- Disclaimers of Warranties
- Limitations of Liability
- Indemnities: *a CPS may say that a CA uses a relying party agreement, under which relying parties are responsible for indemnifying a CA for losses the CA sustains arising out of use of a certificate without properly checking revocation information or use of a certificate for purposes beyond what the CA permits.*
- Term and Termination
- Individual notices and communications with participants
- Amendments
- Dispute Resolution Procedures
- Governing Law
- Compliance with Applicable Law

ETSI TS 101 903 V1.2.2 (2004-04) XML Advanced Electronic Signatures (XAdES)

- mandate from EU Commision, Directive on Electronic Signatures
- based on RFC 3369 "CMS signatures" (RFC on "Cryptographic Message Syntax")
- goal: long time validation, format interoperability
- no new techniques and designs – rather how to organize them in the signature format
- new ASN.1 types: information for qualifying the CMS signatures
- XML format
- trusted parties:
  - CA
  - Registration Authorities;
  - Repository Authorities (e.g. a directory);
  - Time-Stamping Authorities
  - Time-Marking Authorities;
  - Signature Policy Issuers;

  - Attribute Authorities
  - Arbitrator (arbiter in disputes between signer and verifier)

- signature properties and forms:

  CompleteCertificateRefs - references to the CA certificates used to validate the signature

  CompleteRevocationRefs- references to the full set of revocation information used for the verification of the electronic signature

  AttributeCertficateRefs - references to the full set of Attribute Authorities certificates that have been used to validate the attribute certificate

  CertificateValues– the values of certificates used to validate the signature

  RevocationValues - the full set of revocation information used for the verification of the electronic signature

- time stamping properties:
  - **SignatureTimeStamp** - covers the digital signature value element
  - **AllDataObjectsTimeStamp**- covers all the signed data objects
  - **IndividualDataObjectsTimeStamp** - covers selected signed data objects
  - **SigAndRefsTimeStamp**- covers the signature and references to validation data
  - **RefsOnlyTimeStamp**- covers only references to validation data
  - **ArchiveTimeStamp** -covers signature and other properties required for providing long-term validity

- other properties:
  - **SigningCertificate** - certificate confirming signer's public key
  - **SigningTime**
  - **DataObjectFormat** - to avoid misinterpretation of the signed data
  - **CommitmentTypeIndication** - according to Signing Policy
  - **SignatureProductionPlace -**
  - **SignerRole**
  - **CounterSignature** - signature on a signature

- electronic signature forms:
  - **Basic electronic signature (XAdES-BES)**: MUST contain
    - SigningCertificate signed property. – This property MUST contain the reference and the digest value of the signing certificate. MAY contain references and digests values of other certificates

- KeyInfo: if SigningCertificate in the signature, then no restrictions. Otherwise MUST include a X509Data containing the signing certificate
- SignedInfo: MUST contain a Reference element referencing

  KeyInfo. In this way, the signing certificate is

  secured by the signature.

  MAY contain
- SigninTime signed property;
- DataObjectFormat signed property;
- CommitmentTypeIndication signed
- SignerRole signed property;
- SignatureProductionPlace signed property;
- one or more IndividualDataObjectsTimeStamp

  or AllDataObjectTimeStamp signed properties
- one or more CounterSignature unsigned properties.

- **Explicit policy electronic signatures (XAdES-EPES)**
  - MUST contain signed property SignaturePolicyIdentifier
- **Electronic signature with time (XAdES-T)**
  - the SignatureTimeStamp as an unsigned property added to the electronic signature;
  - or a time mark of the ES provided by a trusted service provider.
- **Electronic Signature with Complete Validation Data**

  (XAdES-C)
  - adds CompleteCertificateRefs and CompleteRevocationRefs unsigned properties
  - if there are attribute certificatesin the signature, then also

    AttributeCertificateRefs and AttributeRevocationRefs
  - *CompleteCertificateRefs element contains a sequence of references to the full set of CA certificates that have been used to validate the electronic signature up to (but not including) the signing certificate.*
  - similarly for CompleteRevocationRefs
  - **references allow to reduce the size of a stored electronic signature format**

- XML structures defined

FIPS PUB 140-2, SECURITY REQUIREMENS FOR CRYPTOGRAPHIC MODULES

- Federal Information Processing Standards, NIST, recommendations and standards based on 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
- 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 fetures 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 (HMAC?)
    - L2: operating system control functions under EAL2, specify roles to operate, modify,..., crypto software withing cryptographic boundary, audit: recording invalid operatons, 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,

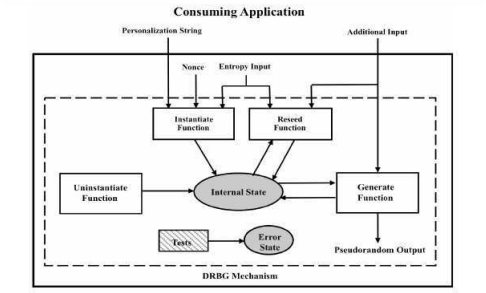
      explicit requests to assume a crypto officer role,

      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 - again links
    - key in out: automated (encrypted) or manual (splitted in L3 or 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:

- seed has a limited period
- reseeded requires a different seed

Internal state:

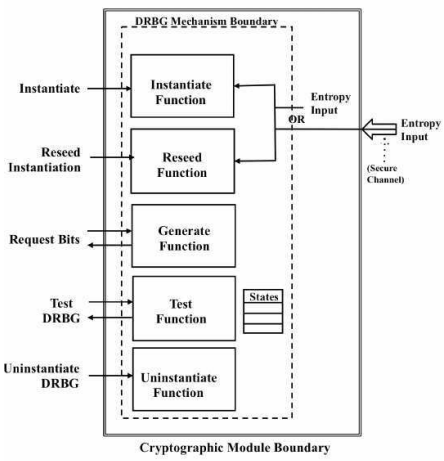
- contains cryptographic chain value AND the number of requests so far
- different instantiations of DRBG must have separate internal states

Instantiation strength:

- ”112, 128, 192, 256 bits”
- Security_strength_of_output = min(output_length, DRBG_security_strength)

DRBG mechanism boundary:

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



Seed:

- entropy is obligatory, entropy strength should be at least as the entropy of the output
- approved randomness source obligatory for entropy source
- nonce: not secret. Example nonces:
  - random value form approved generator
  - a trusted timestamp of sufficient resolution
  - monotonically increasing sequence number
  - combination of a timestamp and a monotonically increasing sequence number, such that the sequence number is reset when and only when the timestamp change

reseed:

- ”for security”

personalization:

- not security critical, but the adversary might be unaware of it (anaogous 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**

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, nonce

  → runs instantiate algorithm to get initial state

  → returns a handle

  Instantiate_function(requested_instantiation_security_strength, prediction_resistance_flag, personalization_string)

  prediction_resistance_flag determines whether consuming application may request reseeding

- Reseed:

  → Explicit request by a consuming application,

  → if prediction resistance is requested

  → also if the upper bound on the number of genereted outpus reached

  → also due to external events

  steps:

  → checks validity of the input parameters,

  → determines the security strength

  → obtains entropy input, nonce

  → runs reseed algorithm to get initial state

- Generate function:

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

Hash_DRBG

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)
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, and reseed_counter).

generating bits:

1. If reseed_counter > reseed_interval, then return "reseed required"
2. If (additional_input ≠ Null), then do
   2.1 w = Hash (0x02 || V || additional_input).

2.2 V = (V + w) mod 2seedlen.

3. (returned_bits) = Hashgen (requested_number_of_bits, V)
4. H = Hash (0x03 || V)
5. V = (V + H + C + reseed_counter) mod 2seedlen
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 = the 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

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

internal state:

- value V of blocklen bits, updated each time another blocklen bits of output are produced
- keylen-bit Key, updated whenever a predetermined number of output blocks are generated
- counter (reseed_counter) = the number of requests for pseudorandom bits since instantiation or reseeding

Update Process:

1. temp = Null

2. While (len (temp)< seedlen, do

2.1 If ctr_len < blocklen

2.1.1 inc = (rightmost (V, ctr_len) + 1) mod 2ctr_len.

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

Else V = (V+1) mod 2blocklen

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. ...

3. seed_material = entropy_input $\oplus$ personalization_string

4. Key = 0^{keylen}

5. V = $0^{blocklen}$

6. (Key, V) = CTR_DRBG_Update (seed_material, Key, V).

7. reseed_counter = 1

8. Return (V, Key, reseed_counter).

Generate:

1. If reseed_counter > reseed_interval, then "reseed required"

2. If (additional_input$\neq$ Null), then

2.1 temp = len (additional_input).

2.2 If (temp< seedlem) then pad additional_input with zeroes

2.3 (Key, V) = CTR_DRBG_Update (additional_input, Key, V).

Else additional_input = $0^{seedlen}$

3. temp = Null

4. While (len (temp)<requested_number_of_bit), do

4.1 If ctr_len< blocklen

4.1.1 inc = (rightmost (V, ctr_len) + 1) mod 2ctr_len.

4.1.2 V = leftmost (V, blocklen-ctr_len) || inc

Else V = (V+1) mod 2blocklen.

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).