The art of cryptography
Secure internet & e-passports
Summer term 2017

Michael Nüsken

Bonn-Aachen International Center for Information Technology

21 April 2017 – 21 August 2017
Global overview

Organizational
Preliminaries
Toolbox cryptography

Secure internet
Secure connections
TLS
Attacks on TLS
Security for TLS
Provable security for TLS

ePassports and identity cards
Introduction
MRTDs (Machine Readable Travel Documents)
eMRTDs
Security mechanisms
eMRTD PKI
ePassport application (BSI summary and extension)
eIDAS application (BSI TR03110)
Security?
German ID card and residence permit (BSI TR03127)
European projects, regulations, applications
Section Overview

Organizational
  Webpage & mailing list
  Time & place
  Hand-in & exam

Preliminaries

Toolbox cryptography
Course page

https://cosec.bit.uni-bonn.de/students/teaching/17ss/17ss-taoc/

Mailing list for discussions

17ss-taoc@lists.bit.uni-bonn.de
Subscribe today!
Organizational:

Time & place

Lectures

- Monday, 12^{30}-14^{00} sharp, b-it bitmax.
- Thursday, 12^{30}-14^{00} sharp, b-it bitmax.

Tutorial

- Monday, 14^{15}-15^{45} sharp, b-it bitmax.
Exercises

- Out: Typically, Monday, 18\textsuperscript{00}.
- In: Friday, 12\textsuperscript{00} (noon).
- ≥ 50\%: Admitted to the exam.
- ≥ 70\%: Exam bonus: One third.
- ≥ 90\%: Exam bonus: Two third.

Final exam

- 23 August 2017 (Check webpage!).
- ≥ 50\% of all points necessary to pass.
- If and only if you pass, the bonus is applied.
Organizational

Preliminaries

Overview
Barrier
Secure emails?
PGP / GnuPG
GnuPG
Attacks and defenses

Toolbox cryptography
Expectations

What do you expect to learn in this course?
What made you come here?
Expectations
Practical application of toolbox
Security applications
Cryptography

Learn more about!

Theory of cryptography that practice
Real-world cryptography

E-identities
Group security
Proofs/proof techniques
Practical cryptography

Improve/improve course

Fun

Can't stay away from it?

Bike rides
Secure internet.
  secure channels.
    In practice:
      TLS.
      ssh, scp, ...
      IPsec.
      OTS, axolotl (OpenWhisperSystems), ...
        Signal, WhatsApp,
        ...
      EMV.
  TLS.
    Definition.
    Correctness, efficiency.
    Security: definitions, proof.
    Attacks.
  You: IPsec, ssh.
  e-passports, identity cards, ...
    Standards.
    Security?
    Attacks?
click here
What is email?

- 1965, MIT: Users logging into a central system via dial-up exchanged messages by creating files like to-tom in a shared file systems.
- 1964/5: proposal CTSS MAIL.
  Anecdote: “The idea of sending “letters” using [MIT Computation Center’s] CTSS was initially resisted by management, as a waste of resources.”
- 1971: ‘first email’. Ray Tomlinson: SNDMSG appended text to the end of a file on another computer. He also proposed the ‘@’ sign.
- 1972: Unix mail
- ...
- 1982, RFC 822: A protocol!
SMTP (Simple Mail Transport Protocol)

- Goal:
  - Send a bare text message.
  - ... to a specific offline user.
  - ... from some user.
  - Fast.
  - Simple.
  - Reliable.
  - Asynchronous.
SMTP (Simple Mail Transport Protocol)

Format:

- Pure text, namely strict 7-bit ASCII, only chars 32–126, 10, 13.
- Simple format:
  
  From <sender> <timeinfo>
  <header>
  <blank line>
  <body>

  where <header> consists of lines
  <keyword>: <value>

  Particular keywords are Message-ID, From, To, Date, Subject.
How to find the recipient?

- Read DNS (Domain Name System).
- ...uses plain text channel (telnet, tcp).
Infrastructure

1. To: bob@b.org
   From: alice@a.org
   Dear Bob. ...

2. MX for b.org?

3. mx.b.org

4. To: bob@b.org
   From: alice@a.org
   Dear Bob. ...

5. To: bob@b.org
   From: alice@a.org
   Dear Bob. ...

smtp.a.org

mx.b.org

mx.b.org

ns.b.org

The Internet

Preliminaries:
Secure emails?

Security?

No security!

...so far.
Security

- What do we have?
- What do we want?
- History and design...
Desired features

- Confidentiality. Nobody apart from the intended recipient can read the message.
- Non-repudiation. The sender cannot deny having sent the message.
- Authenticity. The recipient can check that the message was created and sent by a certain sender.
- Integrity. The message was not changed during transport. ⇐ authenticity.
- Availability.
- Pseudonymity/Anonymity. The recipient cannot know who the sender is.
- Protection against malware or spam.
- Retract a message...
- Acknowledgement of reading.
- Acknowledgement of receipt.
Preliminaries:
Secure emails?

Security vs. Safety

Safety (reliability)
- random attacks
- format, syntax

Security
- content, semantics
- intentional attacks
Preliminaries:

PGP / GnuPG
History

- 1991 **Phil Zimmermann.** Pretty Good Privacy provides strong cryptography for everyone.
- 1993. Criminal investigations against Zimmermann for “munitions export without a license”: Cryptosystems with keys larger than 40 bits is considered munitions and subject to US export restrictions.
- 1995 PGPi. Typed from the book “PGP Source Code and Internals”, circumventing the US export restrictions since books are “free speech” and thus protected by the First Amendment.
- 1998 the case against Zimmermann was dropped.
- 1998: GnuPG founded by **Werner Koch.**

GnuPG is free software and open source from start until now!
Preliminaries:
PGP / GnuPG

*History II*

- 1997: NAI (Network associates, Inc. = merger of McAfee Associates, PGP Corporation, …) add disk encryption, desktop firewalls, intrusion detection, IPsec VPNs. **Export strategy: release source code.**
- 2000: The export restrictions are liberalized.
- 2000 NAI stops releasing source code.
- 2001: Zimmermann leaves NAI.
- 2002: PGP Corporation, a new foundation by some ex-PGP team members, buy PGP assets back from NAI.
- 2010: …
GnuPG

... is the open source and free software solution for protecting email (and files). It still offers:

Confidentiality of content by providing hybrid encryption of the mail body or any file.

Authenticity and integrity of content with respect to a user associated key pair by a signature of the mail body or any file.

Key infrastructure by a web of trust connecting key pairs with user information, like name, email address, picture.
**Usability**

... is provided by interfaces to the command line gpg to various email clients and other.

The trickiest part is the **key management**.

- Private keys should remain with the user only!
- Must connect identity and private key!
- The user needs to decide
  - ... which public keys he trusts.
  - ... and how: Other public keys may
    - ... sign a message.
    - ... certify a further public key connected to a specified identity.
- Loss of private keys?
- Revocation?
- Private key on several devices?
- Certify other entity’s keys? → **PGP signing party.**
GnuPG is not invincible

- An implementation feature called sliding windows is implemented in libgcrypt.

Juli 2017

Bernstein, Breitner, Genkin, Bruinderink, Heninger, Lange, van Vredendaal
‘Sliding right into disaster’.
They can use the induced timings to break RSA-1024 and even
13% of the RSA-2048 keys. The attack requires access to a
timing side channel, e.g. using software on the same CPU as
the attacked GnuPG instance.
<table>
<thead>
<tr>
<th>Attack</th>
<th>Defense</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network sniffing, eavesdropper in the middle</td>
<td><strong>Encryption &amp; PKI</strong></td>
</tr>
<tr>
<td>Hacking the mail platform</td>
<td>Keep your private key elsewhere.</td>
</tr>
<tr>
<td>Hacking the mail account</td>
<td>Use a randomly generated password. Keep your private key elsewhere.</td>
</tr>
<tr>
<td>Impersonation, modifier in the middle</td>
<td><strong>Signature &amp; PKI</strong></td>
</tr>
<tr>
<td>Phishing, stealing passwords, ...</td>
<td>Education!</td>
</tr>
<tr>
<td>Denial of Service</td>
<td>Redundancy, acknowledgments, cookies</td>
</tr>
<tr>
<td>Sending malware</td>
<td>Education!</td>
</tr>
<tr>
<td></td>
<td>Antivirus software, SPAM filters, (firewalls).</td>
</tr>
</tbody>
</table>
Section Overview

Organizational

Preliminaries

Toolbox cryptography
  Foundations
  Public-key encryption
  Symmetric encryption
  Hybrid Encryption and the KEM/DEM Paradigm
  Public-key signatures
  Message authentication codes
  Key exchange
  PKI and certificates
  Trust anchor
  Randomness
Primitives and security notions

For each type of cryptosystem we consider:

- Blackbox description and concrete implementation.
- Correctness.
- Efficiency.
- Security.
  → security notion/game.
Blackbox view, asymmetric/public-key

- Public-key encryption:

\[ m \xrightarrow{\operatorname{Enc}} c \xrightarrow{\operatorname{Dec}} m' \]

- Public-key signature:

\[ 1^\kappa \xrightarrow{\text{KeyGen}} m \xrightarrow{\operatorname{Sign}} s \xrightarrow{\operatorname{Vrfy}} \{\text{TRUE}, \text{FALSE}\} \]
Blackbox view, symmetric

- Symmetric-key encryption:

  - $m \rightarrow Enc \rightarrow c \rightarrow Dec \rightarrow m'$

  - KeyGen

- Message authentication:

  - $m \rightarrow KeyGen \rightarrow t \rightarrow Vrfy \rightarrow \{TRUE, FALSE\}$

  - $1^\kappa$

  - $m'$

  - $c$

  - $c$
Toolbox cryptography:
Public-key encryption

- Blackbox description:

- Correctness. For each message $m$ and $(K, k) \leftarrow \text{KeyGen}(1^\kappa)$ we have $\text{Dec}_k \text{Enc}_K(m) = m$.

- Efficiency. All algorithms/protocols are efficient, i.e. polynomial time or use at most a specified time.

- Security. Each ‘fast’ attacker can ‘break’ the scheme at most with ‘small’ advantage.

‘break’ = solve a task defined by a game
  ▶ providing certain input and oracles and
  ▶ expecting an answer.

‘fast’ = polynomial time or given time $t$.
‘small’ = negligible or at most given $\epsilon$.

Precisely: OW-CPA, IND-CPA, IND-CCA, ...
Onewayness game $G^{OW-CPA}$

- Pick key pair $(K, k) \leftarrow \text{KeyGen}(1^\kappa)$. 
- Choose a hidden message $m \leftarrow \mathcal{M}_K$ uniformly random.
- Prepare an encryption oracle $\mathcal{O}_{\text{Enc}}$. When called with $m \in \mathcal{M}_K$ the oracle returns $c \leftarrow \text{Enc}_K(m)$.
- Prepare a one-time oracle $\mathcal{O}_{\text{Chal}}$. When called with no argument the oracle returns $c^* \leftarrow \text{Enc}_K(m)$.
- Call the attacker $\mathcal{A}$ with input $1^\kappa$, public key $K$ and the oracles $\mathcal{O}_{\text{Enc}}$ and $\mathcal{O}_{\text{Chal}}$. Await a guess $m' \in \mathcal{M}_K$.
- If $m = m'$ then ACCEPT else REJECT.

**Definition**

A public-key encryption scheme $\Pi$ is $(t, \epsilon)$-OW-CPA secure iff for each probabilistic $t$-time attacker $\mathcal{A}$ the advantage

$$\text{adv}^{O^{W-CPA}}(\mathcal{A}) = \left| \text{prob} \left( G^{O^{W-CPA}}(\mathcal{A}) = \text{ACCEPT} \right) - 0 \right|$$

is at most $\epsilon$. (0 or $\frac{1}{\#\mathcal{M}_K}$ . . . )

**Theorem**

For each public-key encryption scheme: $\text{OW-POA} = \text{OW-CPA}$.
Toolbox cryptography:
Public-key encryption

Indistinguishability game $G^{\text{IND-CPA}}$

- Pick key pair $(K, k) \leftarrow \text{KeyGen}(1^\kappa)$.
- Choose a hidden bit $h \leftarrow \{0, 1\}$ uniformly random.
- Prepare an encryption oracle $\mathcal{O}_{\text{Enc}}$. When called with $m \in \mathcal{M}_K$ the oracle returns $c \leftarrow \text{Enc}_K(m)$.
- Prepare a one-time oracle $\mathcal{O}_{\text{Test}}$. When called with $m_0^*, m_1^* \in \mathcal{M}_K$ the oracle returns $c^* \leftarrow \text{Enc}_K(m_h^*)$.
- Call the attacker $\mathcal{A}$ with input $1^\kappa$, public key $K$ and the oracles $\mathcal{O}_{\text{Enc}}$ and $\mathcal{O}_{\text{Test}}$. Await a guess $h' \in \{0, 1\}$.
- If $h = h'$ then ACCEPT else REJECT.

Definition

A public-key encryption scheme $\Pi$ is $(t, \epsilon)$-IND-CPA secure iff for each probabilistic $t$-time attacker $\mathcal{A}$ the advantage

$$\text{adv}^{\text{IND-CPA}}(\mathcal{A}) = \left| \text{prob} \left( G^{\text{IND-CPA}}(\mathcal{A}) = \text{ACCEPT} \right) - \frac{1}{2} \right|$$

is at most $\epsilon$.

Theorem

For each public-key encryption scheme: $\text{IND-POA} = \text{IND-CPA}$. 
**Indistinguishability game $G^{\text{IND-CCA}}$**

- Pick key pair $(K, k) \leftarrow \text{KeyGen}(1^\kappa)$.
- Choose a hidden bit $h \leftarrow \{0, 1\}$ uniformly random.
- Prepare an encryption oracle $O_{\text{Enc}}$. When called with $m \in M_K$ the oracle returns $c \leftarrow \text{Enc}_K(m)$.
- Prepare a decryption oracle $O_{\text{Dec}}$. When called with $c \in C$ the oracle returns $m \leftarrow \text{Dec}_k(c)$.
- Prepare a one-time oracle $O_{\text{Test}}$. When called with $m_0^*, m_1^* \in M_K$ the oracle returns $c^* \leftarrow \text{Enc}_K(m_h^*)$.
- Call the attacker $A$ with input $1^\kappa$ and the oracles $O_{\text{Enc}}, O_{\text{Dec}}$ and $O_{\text{Test}}$. Await a guess $h' \in \{0, 1\}$.
- If the decryption oracle has even been called with the (first) output $c^*$ of the test oracle as input then randomly ACCEPT or REJECT.
- If $h = h'$ then ACCEPT else REJECT.

**Definition**

A public-key encryption scheme $\Pi$ is $(t, \epsilon)$-IND-CCA secure iff for each probabilistic $t$-time attacker $A$ the advantage

$$\text{adv}^{\text{IND-CCA}}(A) = \left| \text{prob} \left( G^{\text{IND-CCA}}(A) = \text{ACCEPT} \right) - \frac{1}{2} \right|$$

is at most $\epsilon$. 
Toolbox cryptography:
Public-key encryption

Security landscape for public-key encryption

- Non-Malleability (NM)
- Indistinguishability (IND)
- Onewayness (OW)
- Unbreakability (UBK)

Public Only Attack (POA)
Chosen Plaintext Attack (CPA)
Plaintext Checking Attack (CCA)
Chosen Ciphertext Attack (CCA)
Toolbox cryptography:  
Public-key encryption

Reductions

- Given an efficient attacker $A$: runtime $t(n)$, success $\epsilon(n)$.
  - It assumes to play a given security game $G$, which describes a break for some scheme $\Pi$. 
Reductions

- We let it play against our reduction $\mathcal{R}$.
  - We must ensure that $\mathcal{A}$ cannot detect a difference.
  - The reduction manipulates input and oracles.
  - The reduction uses the answer.
  - The reduction $\mathcal{R}$ tries to solve some problem $X$. 
Toolbox cryptography:
Public-key encryption

Reductions

- Assume: the reduction solves problem $X$ with probability at least $\frac{1}{n^c}$ provided the attacker wins the original game.
- Runtime polynomial, success $\frac{\epsilon(n)}{n^c}$.
- Thus: If $A$ is successful, i.e. $\epsilon(n)$ is not negligible, then also $R$ is successful, i.e. $\frac{\epsilon(n)}{n^c}$ is not negligible.
Toolbox cryptography:
Public-key encryption

Reductions

Short
If $A$ is successful then we can solve the problem $X$.

Theorem (Relative security)

*If the problem $X$ is $(t', \epsilon')$-hard then the scheme is $(t, \epsilon)$-secure in the sense of the security game $G$.**
Toolbox cryptography:
Public-key encryption

Reductions form landscape

Theorem

\[
\begin{align*}
NM-^\ast \text{ secure} & \quad \downarrow \\
IND-^\ast \text{ secure} & \quad \downarrow \\
OW-^\ast \text{ secure} & \quad \downarrow \\
UBK-^\ast \text{ secure} & \\
\end{align*}
\]

\[
\begin{align*}
\ast\text{-CCA secure} & \quad \downarrow \\
\ast\text{-PCA secure} & \quad \downarrow \\
\ast\text{-CPA secure} & \quad \downarrow \\
\ast\text{-POA secure} & \\
\end{align*}
\]
Example: RSA encryption

- OW-CPA secure *if* its encryption is a one-way function.
- Not IND-CPA secure since the encryption is deterministic.
Example: ElGamal encryption

- IND-CPA secure iff DDH is hard relative to the generated group as chosen in ElGamal KeyGen.
- Not IND-CCA secure since homomorphic.
Toolbox cryptography:
Public-key encryption

Example: Cramer-Shoup encryption

- IND-CCA secure if
  - DDH is hard relative to the generated group and
  - the hash function is collision-resistant
as chosen in its KeyGen.
Toolbox cryptography:
Symmetric encryption

- Blackbox description:

- Correctness. For each message $m$ and $k \leftarrow \text{KeyGen}(1^\kappa)$ we have $\text{Dec}_k(\text{Enc}_k(m)) = m$.

- Efficiency. All algorithms/protocols are efficient, i.e. polynomial time or use at most a specified time.

- Security. No ‘fast’ attacker can ‘break’ the scheme with ‘large’ advantage.
  - The security notions are essentially identical.
  - Here, POA $\neq$ CPA, since the encryption is only accessible via an oracle.
Example: OTP (one-time pad)

- IND-POA.
- Not UBK-CPA.
Toolbox cryptography:
Symmetric encryption

For block ciphers, like AES, one usually hopes that the encryption is a pseudo-random function (PRF).

Example: AES

- OW-CPA if AES encryption is PRF.
- Not IND-CPA since the encryption is deterministic.

Example: AES-CTR with random initialization vector

- IND-CPA if AES encryption is PRF.
- Not IND-CCA since homomorphic.
Hybrid encryption

\[ \begin{align*}
&k \\
&\text{Encap} \\
&m \\
&\text{Enc} \\
\end{align*} \quad \begin{align*}
&c \\
&\text{Decap} \\
&m' \\
&\text{Dec} \\
\end{align*} \quad \begin{align*}
&k \\
&\text{KeyGen} \\
&\kappa \\
\end{align*} \]
Toolbox cryptography:
Hybrid Encryption and the KEM/DEM Paradigm

Key encapsulation and data encryption mechanism
Theorem

\[
\begin{align*}
KEM \text{ ROR-CPA secure} \\
DEM \text{ IND-POA secure}
\end{align*}
\implies KEM/DEM IND-CPA secure
\]

Notice that it suffices for the symmetric encryption to satisfy a weaker security. Intuitively, the reason is that a fresh uniform key is chosen for each and every message.

Theorem

\[
\begin{align*}
KEM \text{ ROR-CCA secure} \\
DEM \text{ IND-CCA secure}
\end{align*}
\implies KEM/DEM IND-CCA secure
\]
In toolbox cryptography, public-key signatures achieve:

- **Blackbox description:**
  - $1^{\kappa}$ is input.
  - **KeyGen** produces $(K, k)$.
  - $m$ is a message.
  - $s$ is a signature.
  - $\text{Vrfy}_k(m, \text{Sign}_K(m))$ evaluates.

- **Correctness.** For each message $m$ and $(K, k) \leftarrow \text{KeyGen}(1^{\kappa})$ we have $\text{Vrfy}_k(m, \text{Sign}_K(m)) = \text{TRUE}$.

- **Efficiency.** All algorithms/protocols are efficient, i.e. polynomial time or use at most a specified time.

- **Security.** No 'fast' attacker can 'break' the scheme with 'large' advantage.
  - Precisely: EUF-CMA, sEUF-CMA, \ldots
Toolbox cryptography:
Public-key signatures

Signature forge game $G^{sEUF-CMA}$

- Prepare a key pair $(K, k) \leftarrow \text{KeyGen}(1^\kappa)$.
- Prepare a signing oracle $O_{\text{Sign}}$. When called with $m \in \mathcal{M}_K$ the oracle returns $s \leftarrow \text{Sign}_K(m)$.
- Call the attacker $\mathcal{A}$ with input $1^\kappa$, $K$ and the oracle $O_{\text{Sign}}$. Await a pair $(m^*, s^*)$.
- If the signing oracle has been called with input $m^*$ and output $s^*$ then REJECT.
- If $\text{Vrfy}_K(m^*, s^*) = \text{TRUE}$ then ACCEPT else REJECT.

Definition

A signature scheme $\Pi = (\text{KeyGen}, \text{Sign}, \text{Vrfy})$ is $(t, \epsilon)$-sEUF-CMA secure (strongly existentially unforgeable under an adaptive chosen-message attack) iff for each probabilistic $t$-time attacker $\mathcal{A}$ the success probability

$$\text{adv}^{sEUF-CMA}(\mathcal{A}) = \text{prob}\left(G^{sEUF-CMA}(\mathcal{A}) = \text{ACCEPT}\right)$$

is at most $\epsilon$. 
Toolbox cryptography:
Public-key signatures

Example: RSA-FDH

- Verify\(_{h,N,e}(m,s)\) checks whether \(h(m) = s^e\) in \(\mathbb{Z}_N\).
- EUF-CMA secure in the ROM if the RSA problem is hard.
- sEUF-* = EUF-* since the signing is deterministic.
Random Oracle Model

Random oracle model
The Hash function is modelled as a random function via an oracle. A successful attacker must therefore be good for any function:

$$\exists A : \forall h : A^h \text{ is successful}$$

Standard model
The hash function is given and not even the game has any influence on its values. If that function is left unknown, we ask for:

$$\forall h : \exists A : A^h \text{ is successful}$$

Informal Theorem (Canetti, Goldreich & Halevi 2004)
There exists encryption and signature schemes that are secure in the Random Oracle Model, but have NO SECURE IMPLEMENTATION in the [standard model]. That is, implementing these secure ideal schemes [...] results in insecure schemes.
Toolbox cryptography:
Public-key signatures

Example: ElGamal signature

TODO: ...
Example: Schnorr signature

- Verify\([\pi, h, A](m, [c, r])\) checks whether with \(I \leftarrow g^r A^{-c}\) in \(G\) we have \(c = h(I, m)\) in \(\mathbb{Z}_q\).

- EUF-CMA secure in the ROM if DL is hard relative to the group generated in KeyGen.

- sEUF-CMA is not automatic since the signing is randomized.
Toolbox cryptography:
Message authentication codes

- Blackbox description:

- Correctness. For each message $m$ and $k \leftarrow \text{KeyGen}(1^\kappa)$ we have $\text{Vrfy}_k(m, \text{Mac}_k(m)) = \text{TRUE}$.

- Efficiency. All algorithms/protocols are efficient, i.e. polynomial time or use at most a specified time.

- Security. No ‘fast’ attacker can ‘break’ the scheme with ‘large’ advantage.
Precisely: EUF-CMA, sEUF-CMA, ...
Example: HMAC-SHA1

- As long as SHA1 is collision resistant(?) and SHA1’s compression function is a PRF, HMAC-SHA1 is heuristically EUF-CMA secure. (Quantify?!)

- As long as SHA1’s compression function is a PRF, HMAC-SHA1 is heuristically EUF-CMA secure.

- Here, sEUF ≡ EUF since Mac is deterministic.
Example: AES-CBC-MAC, -CMAC

- As long as AES encryption is a PRF and the message length is fixed, AES-CBC-MAC is EUF-CMA secure.
- Here, sEUF ≡ EUF since Mac is again deterministic.
Toolbox cryptography:

Key exchange

- Blackbox description:

- Correctness. For any $\kappa$, after the real protocol $k_{\text{init}} = k_{\text{resp}}$.
- Efficiency. All algorithms/protocols are efficient, i.e. polynomial time or use at most a specified time.
- Security. No ‘fast’ attacker can ‘break’ the scheme with ‘large’ advantage.

At least: (passive) ROR-POA, namely:
- distinguish between a Real Or Random key
- under a passive Public Only Attack.
Real-or-random game $G_{\Pi}^\text{ROR-POA}$

- Choose parameters $\pi \leftarrow \text{Gen}(1^\kappa)$ (mostly not randomized).
- Let Alice and Bob given the parameters $\pi$ execute the key exchange protocol $\Pi$. We obtain the transcript $t$ and the shared key $k_0$.
- Pick a random key $k_1 \leftarrow K_\pi$.
- Pick a hidden bit $h \leftarrow \{0, 1\}$.
- Call the attacker with the parameters $\pi$, the transcript $t$ and $k_h$. Await a guess $h' \in \{0, 1\}$.
- If $h' = h$ then ACCEPT else REJECT.

**Definition**

A key exchange $\Pi$ is passively $(t, \epsilon)$-ROR-POA secure iff

for each probabilistic $t$-time attacker $A$ the advantage

$$\text{adv}_{\Pi}^\text{ROR-POA}(A) = \left| \text{prob} \left( G_{\Pi}^\text{ROR-POA}(A) = \text{ACCEPT} \right) - \frac{1}{2} \right|$$

is at most $\epsilon$. 
Example: Diffie-Hellman

Based on $\kappa$ fix a group $(G, \cdot)$ and an element $g \in G$ of order $q$.

$$1^\kappa, (G, \cdot), g, q$$

$\kappa$,

$1^\kappa, (G, \cdot), g, q$

\[ a \leftarrow \mathbb{Z}_q \]

$A \leftarrow g^a$

$$A$$

$B \leftarrow g^b$

$$B$$

$k_{\text{Alice}} \leftarrow B^a$$

$\triangleright$ Correctness: $k_{\text{Alice}} = g^{ab} = k_{\text{Bob}}$.

$\triangleright$ Efficiency: ok, if the group operation is.

$\triangleright$ Security: passive ROR-POA iff DDH is hard relative to the group generated in Gen.
Decisional Diffie-Hellman (DDH) Game

- Pick $\pi = (G, \cdot, g, q) \leftarrow \text{Gen}(1^\kappa)$.
- Choose $a, b, c_1 \leftarrow \mathbb{Z}_q$, compute $c_0 = ab$.
- Pick a hidden bit $h \leftarrow \{0, 1\}$.
- Call the player with $\pi$, $g^a$, $g^b$, $g^{c_1 h}$. Await a guess $h' \in \{0, 1\}$.
- If $h' = h$ then ACCEPT else REJECT.

Theorem

If the Decisional Diffie-Hellman problem is $(t', \epsilon)$-hard
then the Diffie-Hellman key exchange is $(t, \epsilon)$-ROR-POA secure.
And vice versa with only a small time loss.

Proof. ...
Moderator-in-the-middle

Alice: $a \leftarrow \mathbb{Z}_q$

Mo: $\tilde{b}, \tilde{a} \in \mathbb{Z}_q$

Bob: $b \leftarrow \mathbb{Z}_q$

$k \leftarrow (g^{\tilde{b}})^a$

$k \leftarrow (g^a)^{\tilde{b}}, k' \leftarrow (g^b)^{\tilde{a}}$

Theorem

Basic Diffie-Hellman is never secure against an active attacker.
RSA key exchange

- TODO:
- Correctness: ok.
- Efficiency: ok.
- Security:
  - Not secure without padding.
  - ROR-POA secure if TODO: padding good.
Toolbox cryptography:
PKI and certificates

- Hierarchical PKI (public key infrastructure). Used for:
  - Websites. Mail server. TLS partners.
  - Communication entities.
  - Passports:
    - Document authentication.
    - Reader authentication.
  - ...

Side remark: Bundesdruckerei or Deutsche Telekom issue SSL certificates starting at about 150 Euro/year.

- Web of trust. Used for:
  - PGP.
- Identity-based cryptography. Certificateless cryptography.
  - Idea: Use identity as a public-key.
- Blockchain based ...
Certificate (X.509)

We, the Deutsche Telekom Root CA 2, hereby grant that

DFN Verein PCA Global - G01

has

- the right to certify with a maximum of 2 intermediates
- verifiable
- by PKCS #1 RSA-Verschlüsselung
- with the public key

Modulus (2048 Bits):
- e9 9b c3 67 85 79 0d ae f5 84 54 c3 96 50 35 34
- 62 e9 6e 4c ed 94 d7 00 5b 95 22 74 d4 20 eb 34
- 8f d6 ec c0 31 04 0b 99 81 e2 a6 14 d2 52 a0 28
- 23 84 8b 74 89 04 5e 5b e0 e2 78 c1 78 cb 16 cb
- 2b 35 39 7b 2d 90 45 d0 ed a0 00 7a 7c bf 4a 0a
- 1b 00 c3 86 e9 5c 2b 31 11 7b 0c f3 82 24 43 8c
- 1c 38 8b 6a 68 00 9a ee de 4f 78 ab d2 c6 13 9b
- 76 ad ee de 26 e8 ef 01 af 74 0f c1 09 a2 f6 6b
- ce bd d3 cd 14 30 4f 5f e5 e3 a4 c6 2b 9b 82 la
- 03 27 30 0d 02 6b e0 4d ed d1 09 23 2a 96 35 8a
- 27 d3 76 c6 71 b6 90 1d cd ef ff 35 86 7d 6f 33
- b3 db 0f c5 11 c2 8a 83 a1 94 5d 41 6b d8 d2 10
- f5 4c fd ca 51 ac d9 bd ef 92 83 bb da eb 8b 16
- 56 56 43 cf a1 d5 13 3d a6 1f 27 30 cd 49 54 db
- c9 13 34 9a 71 75 c5 6c ea a7 0b 98 f9 21 9d 27

Exponent (24 Bits):
- 65537

Valid
- from 22 July 2014, 14:08:26 GMT,
- until 10 July 2019, 01:59 GMT.

Signed with PKCS #1 SHA-256 mit RSA-Verschlüsselung:

Format:
Hierarchical PKI

- Telekom
  - DFN
  - Uni Bonn
    - www.bit
    - cosec.bit

Hierarchical PKI, management needs

- **Need trust anchor**: root certificate.
- Create certificates.
  - CA must verify $ld \leftrightarrow K$ and that $ld$ knows $k$.
  - Validity periods?!
- Revoke certificates (before validity ends).
  - Necessary: some $k$ interaction.
  - Put on Certificate Revocation List (CRL). Problematic:
    - How to deploy?
    - Can it revoke certs since yesterday?
    - If root cert is revoked, is CRL then valid?
- Check certificates. Entity must check:
  - Not on a CRL. Need unforged up-to-date CRL.
  - Signatures valid for chain. Need unforged root certificate.
  - Validity. Need unforged clock.
Web of trust
Web of trust, discussion

- **Need trust anchor**: identities signed by own key.
- Decentralized. Thus independent.
- Revocation.
- Trust transfer difficult to understand.
  - Trust in relation Id $\leftrightarrow K$. Owner trust.
  - Trust in competence and honesty of signers.

Trust chains difficult to find and one needs to trust in the honesty and competence of all chain members.

- **Key servers**: increase usability but decrease privacy.
  - An uploaded key can be revoked.
  - ... but never be deleted.
Identity-based encryption

- **Blackbox description:**
  - Idea: use the identity dst as public-key.
  - A third party, namely the KGC, serves the recipient the corresponding private key after verifying authenticity and access conditions.
  - To enable that the third party generates a key and distributes its public key to all senders. The authority’s private key is used to generate private keys for recipients.
  - The recipient needs its served private key to decrypt.
Identity-based encryption

- Blackbox description:

- Correctness: Always $m' = m$.
- Efficiency.
Identity-based encryption

- **Need trust anchor**: KGC near recipient.
- Access restrictions may be included in the identity, say:
  - nuesken@bit.uni-bonn.de, but only after 2017-12-31.
- KGC can decrypt everything.
  - Pro: Company may see all traffic.
  - Con: Single point of attack.
- Certificateless: combine with standard public-key encryption to decrease power and vulnerability of KGC.
Blockchain based PKI

TODO: An emerging approach for PKI is to use the blockchain technology commonly associated with modern cryptocurrency. Since blockchain technology aims to provide a distributed and unalterable ledger of information, it has qualities considered highly suitable for the storage and management of public keys. Emercoin is an example of a blockchain-based cryptocurrency that supports the storage of different public key types (SSH, GPG, RFC 2230, etc.) and provides open source software that directly supports PKI for OpenSSH servers.[citation needed]
Toolbox cryptography:
Trust anchor

- Certification authority.
  - Shall verify \( \text{Id} \leftrightarrow K \) and that \( \text{Id} \) knows \( k \).
  - Must check trustability of id with respect to granted rights.
  - Must protect its own private key.
  - ...

- Various different forms of authorities.
  - Honesty.
  - Competence.
  - ...

- Shared secret key.
  - Secrecy.

- ...

- Shared random string.
  - Secrecy.
Debian bug, openssl fiasco (May 2008)

- Due to debugging of valgrind complaints,
  - eg. access to uninitialized memory,
    a version of openssl was released where only 15 bits from the
    process id and of 3 cases seeded the pseudo-random generator.
    - Only $3 \cdot 2^{15}$ cases,
    - ie. entropy $16.6T$.

  Used in Diffie-Hellman each case results in one key pair. . .
- The bug remained about 14 months in the wild.

Lessons learned:
- Randomness is vital!
- Open software is good! Always multi-eyes.
- . .

See also Schneier’s blog (2008) or Cox’ Lessons (2008).
Toolbox cryptography:
Randomness

In the rush to clean up the Debian-OpenSSL fiasco, a number of other major security holes have been uncovered:

<table>
<thead>
<tr>
<th>Affected System</th>
<th>Security Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fedora Core</td>
<td>Vulnerable to certain decoder rings</td>
</tr>
<tr>
<td>Xandros (EEE PC)</td>
<td>Gives root access if asked in stern voice</td>
</tr>
<tr>
<td>Gentoo</td>
<td>Vulnerable to flattery</td>
</tr>
<tr>
<td>OLPC OS</td>
<td>Vulnerable to Jeff Goldblum’s Powerbook</td>
</tr>
<tr>
<td>Slackware</td>
<td>Gives root access if user says elvish word for “friend”</td>
</tr>
<tr>
<td>Ubuntu</td>
<td>Turns out distro is actually just Windows Vista with a few custom themes</td>
</tr>
</tbody>
</table>
Part I

Secure internet

Secure connections

TLS

Attacks on TLS

Security for TLS

Provable security for TLS
Secure connections
  Secure channel
  Model, placement

TLS

Attacks on TLS

Security for TLS

Provable security for TLS
Secure connections:

Secure channel

- Want: ‘Secure’ channel from Alice to Bob which is
  - confidential and
  - ensures integrity and
  - authenticated.

- Need: Some trust anchor
  - a (hierarchical) PKI root certificate,
  - a shared secret or
  - a shared random string.

- Want additionally:
  - Easy to use.
  - Interoperable.
  - Transparent to the user?
  - Transparent to the application?
  - Configurable for the paranoid?
  - Anonymous to third parties?
  - Completely anonymous?
  - . . .
  - ‘Secure’?
Secure connections:
Model, placement

**OSI vs. TCP/IP**

<table>
<thead>
<tr>
<th>OSI Model</th>
<th>TCP/IP Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host layers</td>
<td></td>
</tr>
<tr>
<td>Application</td>
<td>Application</td>
</tr>
<tr>
<td>Presentation</td>
<td>Transport</td>
</tr>
<tr>
<td>Session</td>
<td>Internet</td>
</tr>
<tr>
<td>Transport</td>
<td>Link</td>
</tr>
<tr>
<td>Media layers</td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td></td>
</tr>
<tr>
<td>Data link</td>
<td></td>
</tr>
<tr>
<td>Physical</td>
<td></td>
</tr>
<tr>
<td>telnet, http, ftp, smtp, imap, DNS, ssh, ...</td>
<td>IPv4, IPsec, IPv6, ICMP, ...</td>
</tr>
<tr>
<td>TLS/SSL, ...</td>
<td>ARP, PPP, Ethernet, DSL, ISDN, ...</td>
</tr>
<tr>
<td>sockets, ...</td>
<td></td>
</tr>
<tr>
<td>TCP, UDP, ...</td>
<td></td>
</tr>
</tbody>
</table>

For more details see wiki/OSI_model.
Secure connections: Model, placement

Placement pro/con

**The higher**
the more control the application has about the security.

**The lower**
the more transparent the security is to applications.
And so often an application uses it even if it is not aware of it.

**The higher**
the easier to achieve

- network address translation (NAT).
- filtering in firewalls.
- ...

...
Secure connections

**TLS**
- History
- Statistics and more
- Structure
- Protocol details
  - Record layer
  - Handshake
  - Computing the master secret
  - Further key exchange options

**OWASP Transport Layer Protection Cheat Sheet**

**Attacks on TLS**

**Security for TLS**

**Provable security for TLS**
SSL = Secure Socket Layer

TLS = Transport Layer Security
Modern versions by IETF, 1999–.
Nov.1995 SSL 3.0. Merged from SSL 2.0 and PCT.¹
  1999 TLS 1.0 (RFC 2246). Now fully managed by IETF.
2006 TLS 1.1 (RFC 4346).
2008 TLS 1.2 (RFC 5246).
  ▶ AEAD added.

2018? TLS 1.3. Lessons learned:
  ▶ Various things deprecated due to attacks.
  ▶ Additional improvements due to security reduction efforts.

¹Taher ElGamal, chief scientist at Netscape Communications from 1995 to 1998, is recognized as the “father of SSL”. 
TLS:
Statistics and more

Secure servers acc.to
Trustworthy Internet Movement: SSLpulse (TIM 2017).
Mind that the security considerations change over time facing different known attacks
Notice the drop in SSL 3.0 after discovery of the Poodle attack
At startup the current state has no key and the ciphersuite

```
TLS_NULL_WITH_NULL_NULL
```

- key exchange
- encryption
- authentication
Connection state

Security parameters
- Connection end.
  - Server or client.
- PRF algorithm.
- Bulk encryption algorithm.
  - Eg. NULL, RC3, 3DES, AES.
  - Cipher type.
    - Stream, block or AEAD.
  - Encryption key length. Block length. Fixed IV length, record IV length.
- MAC algorithm.
  - NULL, HMAC_MD5, HMAC_SHA*.
  - Mac length, mac key length.
- Compression algorithm.
- Master secret (384 bits).
- Client random (256 bits).
- Server random (256 bits).

Further parameters, for each direction
- Cipher state.
  - Eg. encryption key, IV.
- MAC key.
- Sequence number (64 bits).
Record layer

In case of a block cipher each record has the format

<table>
<thead>
<tr>
<th>type</th>
<th>version</th>
<th>len</th>
</tr>
</thead>
<tbody>
<tr>
<td>content ($\leq 2^{14}$ bytes)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAC (optional)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>padding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>padlen</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x14</td>
<td>ChangeCipherSpec</td>
</tr>
<tr>
<td>0x15</td>
<td>Alert</td>
</tr>
<tr>
<td>0x16</td>
<td>Handshake</td>
</tr>
<tr>
<td>0x17</td>
<td>Application</td>
</tr>
<tr>
<td>0x18</td>
<td>Heartbeat</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>version</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x03, 0x00</td>
<td>SSL 3.0</td>
</tr>
<tr>
<td>0x03, 0x01</td>
<td>TLS 1.0</td>
</tr>
<tr>
<td>0x03, 0x02</td>
<td>TLS 1.1</td>
</tr>
<tr>
<td>0x03, 0x03</td>
<td>TLS 1.2</td>
</tr>
</tbody>
</table>
Record layer

... if a block cipher, in TLS always in CBC-mode, is used:

```
... data ...
```

- fragment
- compress
- mac
- encrypt

This is an instance of AtE (authenticate-then-encrypt). The sequence number must be kept in sync by the communicating parties.
Handshake

ClientHello

ServerHello
Certificate*
ServerKeyExchange*
CertificateRequest*
ServerHelloDone

Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished

[ChangeCipherSpec]
Finished
Handshake

Each handshake message is preceded by four bytes:

<table>
<thead>
<tr>
<th>msg_type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>hello_request</td>
</tr>
<tr>
<td>1</td>
<td>client_hello</td>
</tr>
<tr>
<td>2</td>
<td>server_hello</td>
</tr>
<tr>
<td>11</td>
<td>certificate</td>
</tr>
<tr>
<td>12</td>
<td>server_key_exchange</td>
</tr>
<tr>
<td>13</td>
<td>certificate_request</td>
</tr>
<tr>
<td>14</td>
<td>server_hello_done</td>
</tr>
<tr>
<td>15</td>
<td>certificate_verify</td>
</tr>
<tr>
<td>16</td>
<td>client_key_exchange</td>
</tr>
<tr>
<td>20</td>
<td>finished</td>
</tr>
</tbody>
</table>
ClientHello

client_version  random.gmt  random.random

len  len  cipher_suites

len  compression_methods
Some TLS cipher suites (RFC 5246, . . .)

00 00  TLS_NULL_WITH_NULL_NULL
00 01  TLS_RSA_WITH_NULL_MD5
00 02  TLS_RSA_WITH_NULL_SHA
00 3B  TLS_RSA_WITH_NULL_SHA256
00 04  TLS_RSA_WITH_RC4_128_MD5
00 05  TLS_RSA_WITH_RC4_128_SHA
00 0A  TLS_RSA_WITH_3DES_EDE_CBC_SHA
00 2F  TLS_RSA_WITH_AES_128_CBC_SHA
00 35  TLS_RSA_WITH_AES_256_CBC_SHA
00 3C  TLS_RSA_WITH_AES_128_CBC_SHA256
00 3D  TLS_RSA_WITH_AES_256_CBC_SHA256
C0 2B  TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
C0 2C  TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
C0 2F  TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
C0 30  TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

For the full list of currently 326 cipher suites see IANA (2017) TLS parameters.
ServerHello

```
server_version  random.gmt  random.random

len  session_id
    cipher_suite  cm
```
ServerCertificate

len

ASN.1Cert certificate_list
ServerKeyExchange

In case of dhe_dss, dhe_rsa:

<table>
<thead>
<tr>
<th>len</th>
<th>ServerDHPrams.dh_p</th>
</tr>
</thead>
<tbody>
<tr>
<td>len</td>
<td>ServerDHPrams.dh_g</td>
</tr>
<tr>
<td>len</td>
<td>ServerDHPrams.dh_Ys</td>
</tr>
<tr>
<td>algorithm</td>
<td>len</td>
</tr>
</tbody>
</table>

The signature signs client_random, server_random and ServerDHPrams with a key matching the supplied certificate. In case of dh_anon it consists only of the ServerDHPrams. In case of rsa, dh_dss, dh_rsa it is omitted.
Certificate types are with

- an RSA key,
- a DSA key,
- a static DH key.

Supported signature algorithms for verification of the certificate.
Certificate authorities which are accepted. An empty list here means that no a priori restriction is made.
ServerHelloDone

...is an empty message (apart from type and length).
ClientCertificate

len

ASN.1Cert certificate_list
**ClientKeyExchange**

If the key exchange is RSA then this message consists of the RSA ciphertext of the 48 byte PreMasterSecret consisting of the 16 bit client_version and a 368 bit random bit string ciphered into

In most other cases it's the client's explicit Diffie-Hellman public key.

Sometimes this public share is in the client's certificate. Then this packet is marked 'implicit':

(2017-05-22) 98+165
CertificateVerify

where the signature signs the concatenation of all handshake messages so far.
This is sent to explicitly verify a client certificate having signing capability...
Finished

which is computed as

\[ \text{PRF}(\text{master\_secret}, \text{finished\_label} | \text{hash(\text{handshake\_messages})}) \]

with “client finished” or “server finished” as finished\_label.
Computing the master secret

\[
\text{master\_secret} = \text{PRF(\text{pre\_master\_secret,}}\\
\text{"master secret"|}}\\
\text{ClientHello.random|ServerHello.random}}\\
\)[0..47]
\]

The \text{pre\_master\_secret} is either

- generated by the client and sent RSA encrypted or
- the Diffie-Hellman shared key.
## Further key exchange options

<table>
<thead>
<tr>
<th>type</th>
<th>Key exchange</th>
<th>Certificate key type</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA, RSA_PSK</td>
<td>RSA</td>
<td>RSA public (encryption) key</td>
</tr>
<tr>
<td>DHE_RSA, ECDHE_RSA</td>
<td>DH</td>
<td>RSA public (signature) key</td>
</tr>
<tr>
<td>DHE_DSS</td>
<td>DH</td>
<td>DSA public key</td>
</tr>
<tr>
<td>DH_DSS, DH_RSA</td>
<td>DH</td>
<td>Diffie-Hellman public key &amp;...</td>
</tr>
<tr>
<td>ECDH_ECDSA, ECDH_RSA</td>
<td>DH</td>
<td>ECDH capable public key</td>
</tr>
<tr>
<td>ECDHE_ECDSA</td>
<td>DH</td>
<td>ECDSA public key</td>
</tr>
</tbody>
</table>
Benefits
- Protection against unauthorized disclosure
- Integrity
- Server authentication for the client
- Client authentication by certificate or login/password
- Replay prevention

Rules
- Use TLS.
- All pages/files must be served over HTTPS. A single unprotected HTTP request may enable a man-in-the-middle attack.
- Use HSTS (HTTP Strict Transport Security).
- Mark cookies as secure.

Basic requirements
- Access to a PKI for certificates.
- Access to a directory or an Online Certificate Status Protocol (OCSP) responder for certificate revocation status.

FIPS 140-2 validated cryptomodule?

Secure Server Design
- Use TLS everywhere
- Do not provide non-TLS pages for secure content
- Do not mix TLS and non-TLS content
- Use ‘Secure’ cookie flag
- Keep sensitive data out of the URL
- Prevent caching of sensitive data
- Use HSTS
- Use public key pinning

Server certificate
- Use strong keys and protect them
- Use a certificate with all required domain names
- Use fully qualified names in certificates
- Do not use wildcard certificates
- Do not use private addresses (RFC 1918)
- Use an appropriate CA for your audience
- Always provide all needed certificates (chain!)
- Be aware if and have a plan for SHA-1 deprecation

Server protocol and cipher configuration
- Only support strong protocols (i.e., no SSL2.0 or SSL3.0)
- Prefer ephemeral key exchanges
- Only support strong cryptographic ciphers
  - volatile, lots of things to consider
- Support TLS-PSK (pre-shared key) and TLS-SRP (secure remote password, PAKE) for mutual authentication
- Only support secure renegotiation
- Disable compression
- Update your crypto library

Test your overall TLS setup and your certificate
Test with various client configurations

Additional controls
- Extended validation certificates
- Client-side certificates
- Certificate and public key pinning

1 OWASP = Open Web Application Security Project
Secure connections

TLS

Attacks on TLS
  Heartbeat and heartbleed
  CRIME (2012)
  Lucky13 (2013)
  Attack history

Security for TLS

Provable security for TLS
Heartbeat, a TLS feature

A heartbeat message is a message of type HEARTBEAT

<table>
<thead>
<tr>
<th>req/resp</th>
<th>payload_length</th>
<th>payload</th>
<th>padding</th>
</tr>
</thead>
</table>

where the at least 16-byte long padding field has 
\((\text{TLSPlaintext} \cdot \text{length} - \text{payload} \cdot \text{length} - 3)\) bytes. 
The response (2) to a reply must send an exact copy of the payload of the request (1).
Heartbleed attack (2014)

A tiny request with huge payload_length, say 65535, is sent. The affected versions of OpenSSL (1.0.1–1.0.1f) allocate a memory buffer based on the length field TLSPlaintext.length of the request. Say, three bytes are stored as received payload. But when creating the answer payload payload_length bytes are read. That sends up to 64 kilobytes beyond the end of the buffer. Memory that was likely to have been used previously for storing sensitive data like private keys.

Bottom line

- The attack leaves no traces.
- Buffer overflows are dangerous!
Inject an additional cookie MyCookie into the login message

MyCookie=Password=MyS*

GET http://.../login?
Password=MySecretPassword

and loop over all allowed characters in a password for *. If * = e the compression probably works better and the resulting ciphertext is shorter. Observing the ciphertext length finds the best candidate for the next letter.

Bottom line

- We need a small application inside the browser.
- Compression may be harmful. Turn it off.
Attacks on TLS:
Lucky13 (2013)

Padding can be misused!
Consider TLS with a AES_128_CBC and HMAC-SHA-1, ie.block length 16 bytes, mac length 20 bytes. TLS padding, eg.05 05 05 05 05 05.

Thus modify $c_{-2}$ and observe $p_{-1}$ by server reaction.
Consider a packet of 85 bytes, i.e. $t|v|\ell|c_0|...|c_4$:

MACed length: $57 - \text{length}_{\text{padding}}$.
Modify $c_{-2}$. If modified $p_{-1}$ ends with

- 00: One byte padding. MACed length: 56 bytes.
- 01 01: Two byte padding. MACed length: 55 bytes.
Attacks on TLS:
Lucky13 (2013)

SHA-1 adds at least 1 byte padding and 8 bytes length before applying its compression function on blocks of 64 bytes. HMAC-SHA-1 prepends one key block and processes the 20 byte result again with a modified key block.

► So 55 bytes end up as one SHA-1 block, 2 bytes padding.
  \[\Rightarrow 4\text{ SHA-1 compressions}.\]
► And 56 or 57 bytes end up as two SHA-1 blocks, 0 or 1 bytes padding.
  \[\Rightarrow 5\text{ SHA-1 compressions}.\]

This may cause a small timing difference on the server.
Attacks on TLS:
Lucky13 (2013)

In response to a similar, older attack RFC 5246 says:

- “[…] compute the MAC even if the padding is incorrect […]”
- “This leaves a small timing channel, since MAC performance depends to some extent on the size of the data fragment, but it is not believed to be large enough to be exploitable, due to the large block size of existing MACs and the small size of the timing signal.”

**Lucky 13**

…shows that statistics over suitably many, say 128, repetitions makes this small timing difference recognizable.
Attacks on TLS:
Lucky13 (2013)

- By trying all 65536 values for the last two bytes we can find out the correct last 2 bytes of $\text{dec}_k c_{-1}$.
- Similarly, by trying another 256 values for the third but last byte and assuring that the last two bytes come out 02 02 we can find also the third but last byte of $\text{dec}_k c_{-1}$.
- And so on...
- We need $65536 + 14 \cdot 256$ tries to find the entire $\text{dec}_k c_{-1}$.
- This can be done for any ciphertext block put as $c_{-1}$!

Bottom line

- Full plaintext recovery
- due to a supposedly tiny hole.
Attacks on TLS:
Attack history

1994/06  SSL v2
1994/11  SSL v2
1995/06  PCT
1996/11  SSL v3
1999/01  TLS 1.0
2006/04  TLS 1.1
2008/05  Debian bug
2008/08  TLS 1.2
2009/08  Insecure renegotiation
2009/08  sslstrip
2009/08  NUL byte attacks
2011/06  BEAST
2012/04  SSL Pulse
2012/05  Flame
2012/09  CRIME
2013/02  Lucky 13
2013/03  RC4 biases
2013/03  TIME
2013/05  Snowden
2013/08  BREACH
2013/08  TLS 1.3 begins
2013/09  Bullrun and Edgehill
2013/09  Dual EC
2014/03  Triple Handshake attack
2014/04  Heartbleed
2014/09  BERserk
2014/10  POODLE (SSL3.0)
2014/12  POODLE TLS (TLS1.0)
2015/02  Superfish
2015/02  RC4 deprecated
2015/03  SMACK
2015/03  FREAK
2015/05  Logjam
2015/06  SSL 3.0 deprecated
2015/07  More POODLEs
2016/01  SLOTH
2016/02  TLS 1.3 Ready or Not?
2016/03  DROWN
2016/08  SWEET32
2018/07  PCI (payment card industry) deprecates TLS 1.0

... and lots of bad certificates and various further positive events. See Ristić (2017).
Secure connections

TLS

Attacks on TLS

Security for TLS

  Perfect Forward Secrecy
    Beagle Boys attack
    Escrow attack
    Finer-grained attacks

  Denial of Service
    DoS attack
    DoS Mitigation

  Replay protection and Live partner reassurance
    Replay attack
    Replay mitigation
    Live partner reassurance

  Security aspects +
    Typical secure connection protocol

Provable security for TLS
Beagle Boys attack

- They record the encrypted session. Ciphertext is always public!
- Later, they steal the long-term secret(s) and try to decrypt.

Definition

If the Beagle Boys attack always fails then we have Perfect Forward Secrecy (PFS).
**TLS_RSA mode: no PFS**

Beagle boys can decrypt since

- ClientKeyExchange contains
  - the pre_master_secret
  - encrypted with RSA using the long-term public key from the server’s certificate.
- The corresponding private key is a long-term secret.

**TLS_DH or TLS_ECDH mode: no PFS**

The same holds since the server’s private Diffie-Hellman key share is a long-term secret as the corresponding public key share is in its certificate.

**TLS_DHE mode: PFS possible**

We may have PFS.
Escrow attack

The escrow attack is even stronger than the Beagle Boys attack: the long-term secrets are corrupted even before the communication starts.

**TLS_RSA mode:** no PFS, no escrow security

**TLS_DH or TLS_ECDH mode:** no PFS, no escrow security

**TLS_DHE mode:** PFS possible, escrow security possible
Finer-grained attacks

Even when the attacker gets the session key for one message that shall not enable him to decrypt any prior message.

Yes, under, say, DDH hardness, it is possible to be that secure.

...
DoS attack

Attackers try to overload the resources of the victim.

DDoS and DoS amplification

Distributed DoS attack: the attacker upscales its attack by using amplification or using a botnet or both.

- SYN flood using reflection to amplify.
- UDP-based amplification.
- 2016: attacks with more than a tebibit per second ($2^{40}$ bits/sec) have been observed.
- Meanwhile even full-featured tools exists and criminals sell DoS attacks.
DoS Mitigation

DoS attacks only work where the victim needs much more resources for answering than the attacker.

- Increase the costs for the attacker: Defer storage and computation until the other side has revealed some identity information, at least its IP address.
  - Good if the attacker is not willing to identify.
  - Worthless against a botnet.
- Analyse traffic patterns.
  $\rightsquigarrow$ allows threat detection.
- Filter.
- Automate to be fast and flexible.

This is a constant threat! See eg. Kaspersky Cyberthreat Map, Norse Corp. Attack Map, Google’s Digital Attack Map.
Replay attack

- First, the attacker records messages exchanged between potential victims.
- Later, he replays these messages to provoke further actions.
- Or more sophisticated: the attacker may send modifications of the recorded messages.
Replay mitigation

- Nonce = number to be used once. Using nonces makes sure that each session key is fresh.
- In TLS: the random numbers ClientHello.random and ServerHello.random serve as nonces. Remember:

\[
\text{master\_secret} = \text{PRF(pre\_master\_secret,}
\]

\[
\text{“master secret”|}
\]

\[
\text{ClientHello.random|ServerHello.random}
\]

\[
)[0..47]
\]

- Now, a successful replay of a previous message even with potential modifications is only possible by breaking underlying primitives.
Live partner reassurance

- Clearly, each entity wants/needs to be sure that the other side is live.
- This can be assured by making sure that not all of the communication is predictable by the other side.

So, the nonces should be — at least partially — unpredictable by the other side. This is slightly stronger than replay protection.
Security for TLS:
Security aspects +

1. Perfect forward secrecy (Beagle Boys attack, Escrow attack)
2. Denial of Service mitigation
3. Replay protection and live partner reassurance
4. Endpoint identifier hiding (anonymity vs. third parties)
   - Can a passive eavesdropper tell who is talking?
   - Can an active eavesdropper tell who is talking?
5. Plausible deniability (à la non-repudiation)
   - What does a session log prove?
6. Negotiating parameters
   - Which algorithms are used?
   - Are they negotiated?
7. Session key agreement
   - How long? Random? Unpredictable?
   - Do both parties contribute?
   - Person in the middle?
8. Stream protection
   - Is the data stream confidential and authentic?
Security for TLS:
Typical secure connection protocol

Algorithm negotiation. Often incorporated in following stuff.

- Mostly Diffie-Hellman or non-ephemeral RSA key exchange.
  Need, say for 128 bit security:
  - For Diffie-Hellman: secure group where DDH is hard.
    - $\mathbb{Z}_p^\times \ni g, \text{ord } g \text{ with } p \approx 2^{3072}, q \approx 2^{256}$.
    - Elliptic curve $E \ni P$ defined over $\mathbb{F}_\ell, q = \text{ord } P$ with prime power $\ell \approx 2^{256}, q \approx \ell$.
  - For RSA key exchange: secure RSA key pair.
    - $N \approx 2^{3072}$
  - Secure pseudo random number generator and enough entropy (i.e. number of unpredictable bits).
  \textit{Beware}: Debian bug.

Need:
- Secure public key signatures to authenticate.
- Secure certificates and a PKI.

Need:
- Fast algorithms for authenticated encryption, eg. AES-CBC with HMAC-SHA3, AES-XCBC-MAC, AES-GCM.
Secure connections

TLS

Attacks on TLS

Security for TLS

Provable security for TLS

Security notion
  Better than passively ROR-POA?
  Problematic situations
  What can we do?

AKE Security
  Environment and oracles
  Game AKE
  Implications
    Stateful length-hiding authenticated encryption
    TLS cannot be AKE secure

ACCE

Used primitives
Handshake details
Security result
Game Hopping Lemma (Shoup 2004, Dent 2006)
About the proof: Case splitting
About the proof: LORA
Further security results
Attacks vs. security proof
This is by far not all
Better than passively ROR-POA?

We need to replace the static attacker input with something that gives the attacker more options.

- Attacker should be able to control the messages.
  
  So provide oracles for Client and Server:
  
  - $O_{\text{Client}}$ is a several times oracle. On first call, it takes no input and produces the first message.
  - $O_{\text{Server}}$ also is a several times oracle. On first call, it expects a first message and produces the second message.
  - On second call of $O_{\text{Client}}$, it expects a second message and produces a third.
  - ...
  - Eventually, each oracle accepts or rejects and from that point only outputs ACCEPT or REJECT.
Provable security for TLS: 
Security notion

Better than passively ROR-POA?
Better than passively ROR-POA?

- To allow a real-or-random attack, we also supply a test oracle $\mathcal{O}_{\text{Test}}$ which either returns
  - the session key found by client or server or
  - a random key.

- A uniformly chosen hidden bit $h \in \{0, 1\}$ determines whether it is real or random.

- Whose key do we consider? Well, let the attacker determine.

- What if that instance has not accepted?
  Too bad: $\mathcal{O}_{\text{Test}}$ will always output a random key.
  Yet, the attacker knows whether an instance has accepted.
Provable security for TLS:
Security notion

Better than passively ROR-POA?

Game RORA+1

Input: Security parameters.
Output: ACCEPT or REJECT.

1. Prepare oracles $O_{\text{Client}}, O_{\text{Server}}$ that perform the protocol.
2. Choose a hidden bit $h \leftarrow \{0, 1\}$.
3. Prepare a test oracle $O_{\text{Test}}$ that on input Client or Server returns either
   - the session key of that protocol instance if $h = 0$ and
   - a random key if $h = 1$.
4. Call the attacker with the oracles $O_{\text{Client}}, O_{\text{Server}}$ and $O_{\text{Test}}$ and expect its answer $h' \in \{0, 1\}$.
5. If $h = h'$ then
   Return ACCEPT.
Else
   Return REJECT.

Eureka?
Better than passively ROR-POA?

No, epic fail!

Consider the attacker:

1. Let $O_{\text{Client}}$ talk to a simulation of the server protocol until both accept. The fully attacker-controlled simulation obtains the true session key $k$.
2. Ask $O_{\text{Test}}$ for the client’s key or a random one: $k^*$.
3. If $k = k^*$ return 0 (real) else return 1 (random).

This attacker is fast and has advantage $\frac{1}{2}$. 
Better than passively ROR-POA?

And on

- We are asking too much. No key exchange protocol can be that secure.
- Pro: Our game allows a person in the middle.
- Pro: It allows the attacker to be active.
- Con: It does not prevent the attack.
- But how to escape from that problem?
- We might add conditions up to requiring that a second oracle with the same transcript exists.
  😞 But then the attacker is again passive. 😞
- What could prevent the attacker from completely simulating the Server or the Client?

Server and Client need a verifiable secret!
Better than passively ROR-POA?

- So let’s supply a signature key pair to each party.
- For verification the respective partner needs a valid copy of the associated public key.
- So distribute these uncorruptably?
  
  Well, that’s not practical...

- Let’s put an identifier and the public key in a certificate.
  
  ...signed by some CA whose public key is embedded in each protocol instance.
Better than passively ROR-POA?

Game RORA+2

Input: Security parameters.
Output: ACCEPT or REJECT.

1. For each party $P \in \{\text{Client}, \text{Server}\}$ do
2. Prepare a signature key pair $(K_P, k_P) \leftarrow \text{KeyGen}^{\text{Sign}}(\kappa)$.
   Certify the identity $P$ with the public key $K_P$.
3. Prepare oracles $O_{\text{Client}}$, $O_{\text{Server}}$ that perform the protocol and can check certificates.
4. Choose a hidden bit $h \leftarrow \{0, 1\}$.
5. Prepare a test oracle $O_{\text{Test}}$ that on input Client or Server returns either
   - the session key of that protocol instance if $h = 0$ and the protocol instance has accepted or
   - a random key if $h = 1$.
6. Call the attacker with the oracles $O_{\text{Client}}$, $O_{\text{Server}}$ and $O_{\text{Test}}$ and expect its answer $h' \in \{0, 1\}$.
7. If $h = h'$ then
   Return ACCEPT.
Else
   Return REJECT.

Eureka?
Better than passively ROR-POA?

- The previous attack can now be defeated by requiring and inspecting a certificate.

Fine.
- What about more parties?
- Can a party run several protocol instances?
Provable security for TLS:
Security notion

Better than passively ROR-POA?

ℓ parties \( P \in \mathcal{L} \).

q protocol instances \( \pi \in \mathcal{Q} \).
Better than passively ROR-POA?

Game RORA+3

Input: Security parameters, parties $\mathcal{L}$, protocol instances $\mathcal{Q}$.
Output: ACCEPT or REJECT.

1. For each party $P \in \mathcal{L}$ do
2. Prepare a signature key pair $(K_P, k_P) \leftarrow \text{KeyGen}^\text{Sign} (\kappa)$.
   Certify the identity $P$ with the public key $K_P$.
3. Prepare an oracle $O_{\text{Send}}$ that performs the protocol and can check certificates.
4. Choose a hidden bit $h \leftarrow \{0, 1\}$.
5. Prepare a test oracle $O_{\text{Test}}$ that on input protocol instance $\pi$ returns either
   - the session key of that protocol instance if $h = 0$ and the protocol instance has accepted or
   - a random key if $h = 1$.

6. Call the attacker with the oracles $O_{\text{Send}}$ and $O_{\text{Test}}$ and expect its answer $h' \in \{0, 1\}$.
7. If $h = h'$ then
   Return ACCEPT.
Else
   Return REJECT.
Better than passively ROR-POA?

- All parties are honest. Is that realistic?
- Does the game ensure PFS?
  In other words: Could a party be corrupted after the protocol?
- Could a party be corrupted in more situations?
- What about revealing session keys?
Better than passively ROR-POA?

$\ell$ parties $P \in \mathcal{L}$. 
$q$ protocol instances $\pi \in \mathcal{Q}$.
Better than passively ROR-POA?

Game RORA+4

Input: Security parameters, parties $\mathcal{L}$, protocol instances $Q$.
Output: ACCEPT or REJECT.
1. For each party $P \in \mathcal{L}$ do
2. Prepare a signature key pair $(K_P, k_P) \leftarrow \text{KeyGen}^{\text{Sign}}(\kappa)$.
   Certify the identity $P$ with the public key $K_P$.
3. Prepare an oracle $\mathcal{O}_{\text{Send}}$ that performs the protocol and can check certificates.
4. Further, set up oracles $\mathcal{O}_{\text{Corrupt}}$ and $\mathcal{O}_{\text{Reveal}}$ that corrupt parties or reveal the session key of a protocol instance, respectively.
5. Choose a hidden bit $h \leftarrow \{0, 1\}$.
6. Prepare a test oracle $\mathcal{O}_{\text{Test}}$ that on input protocol instance $\pi$ returns
   ▶ the session key of that protocol instance if $h = 0$ and the protocol instance has accepted or
   ▶ a random key if $h = 1$.
7. Call the attacker with the oracles $\mathcal{O}_{\text{Send}}$, $\mathcal{O}_{\text{Corrupt}}$, $\mathcal{O}_{\text{Reveal}}$ and $\mathcal{O}_{\text{Test}}$ and expect its answer $h' \in \{0, 1\}$. The tested protocol instance $\pi'$ belongs to the party $\overline{P}'$.
8. If $h = h'$ then
   Return ACCEPT.
   Else
   Return REJECT.

Eureka?
Wait…

- The attacker can corrupt the alleged partner party and run the protocol with its credentials.
- The attacker can reveal the protocol instance’s session key and use it for its decision.
- The attacker can reveal the alleged partner protocol instance’s session key.

All these situations are unavoidable!

So, we exclude them.
Game RORA+5

Input: Security parameters, parties $\mathcal{L}$, protocol instances $Q$.
Output: ACCEPT or REJECT.
1. For each party $P \in \mathcal{L}$ do
2. Prepare a signature key pair $(K_P, k_P) \leftarrow \text{KeyGen}^{\text{Sign}}(\kappa)$.
   Certify the identity $P$ with the public key $K_P$.
3. Prepare an oracle $O_{\text{Send}}$ that performs the protocol and can check certificates.
4. Further, set up oracles $O_{\text{Corrupt}}$ and $O_{\text{Reveal}}$ that corrupt parties or reveal the session key of a protocol instance, respectively.
5. Choose a hidden bit $h \leftarrow \{0, 1\}$.
6. Prepare a test oracle $O_{\text{Test}}$ that on input protocol instance $\pi$ returns
   - the session key of that protocol instance if $h = 0$ and the protocol instance has accepted or
   - a random key if $h = 1$.
7. Call the attacker with the oracles $O_{\text{Send}}$, $O_{\text{Corrupt}}$, $O_{\text{Reveal}}$ and $O_{\text{Test}}$ and expect its answer $h' \in \{0, 1\}$. The tested protocol instance $\widehat{\pi}'$ belongs to the party $\widehat{P}'$.
8. Determine the alleged partner protocol instance $\widehat{\pi}'$ and the party $\widehat{P}'$ it belongs to.
9. If the corrupt oracle was used on $\widehat{P}'$ before $\widehat{\pi}'$ accepted then
   Return randomly ACCEPT or REJECT.
10. If the reveal oracle was used on $\widehat{\pi}'$ then
    Return randomly ACCEPT or REJECT.
11. If the reveal oracle was used on $\widehat{\pi}'$ then
    Return randomly ACCEPT or REJECT.
12. If $h = h'$ then
    Return ACCEPT.
Else
    Return REJECT.

Eureka?
Wait again...

- Which one is the **partner** party?
- Which one is the **partner** protocol instance?
- Is the **partner** protocol instance unique?
Problematic situations

- After the attacker is done, there is one protocol instance that has accepted and either:
  - **Two or more** other protocol instances have seen the exact same messages.
  - **No** other has seen the same messages.
- What happens to the message that is sent after a protocol instance has accepted?
  In other words: Who is the partner of the sender of the last message?

Well, the identification of the partner protocol instance or party is tricky.
What can we do?

Basically: identification only by the seen messages.

- Are we talking about all messages?
- Or just some?
- Or just a part of them?

We choose the basis for the decision to accept:

- Namely, consider all messages seen up to acceptance.
- An instance with opposite rôle is a partner instance if it has seen the very same messages.
- Notice that this is asymmetric.
Authenticated Key Exchange

A security notion.
Environment and oracles

- One Attacker $\mathcal{A}$.
- One CA.
- A set $\mathcal{L}$ of $\ell$ parties.
- A set $\mathcal{Q}$ of $q$ protocol instances. $q \gg \ell$.
- Four oracles:

  $\mathcal{O}_{\text{Send}} : (\pi, m_{\text{in}}, P) \mapsto \begin{cases} [m_{\text{out}}], & \text{WAIT} \\ \text{ACCEPT}, & \text{REJECT} \end{cases}$

  $\mathcal{O}_{\text{Corrupt}} : P \mapsto k_P$.

  $\mathcal{O}_{\text{Reveal}} : \pi \mapsto \begin{cases} s_\pi & \text{if } \pi \text{ has accepted,} \\ \text{FAIL} & \text{otherwise.} \end{cases}$

  $\mathcal{O}_{\text{Test}} : \pi \mapsto \begin{cases} s_\pi & \text{if } \pi \text{ has accepted and } h = 0, \\ r & \leftarrow \mathcal{K} \text{ otherwise}. \end{cases}$
Environment and oracles

- One Attacker $\mathcal{A}$.
- One CA.
- A set $\mathcal{L}$ of $\ell$ parties.
- A set $\mathcal{Q}$ of $q$ protocol instances. $q \gg \ell$.
- Four oracles:
  \[ O_{\text{Send}} : (\pi, m_{\text{in}}, P) \mapsto [m_{\text{out}}], \]
  \[ \begin{cases} \text{WAIT} & \text{if } \pi \text{ has accepted,} \\ \text{ACCEPT} & \text{otherwise.} \end{cases} \]
  \[ O_{\text{Corrupt}} : P \mapsto k_P. \]
  \[ O_{\text{Reveal}} : \pi \mapsto \begin{cases} s_{\pi} & \text{if } \pi \text{ has accepted,} \\ \text{FAIL} & \text{otherwise.} \end{cases} \]
  \[ O_{\text{Test}} : \pi \mapsto \begin{cases} s_{\pi} & \text{if } \pi \text{ has accepted and } h = 0, \\ r & \text{otherwise.} \end{cases} \]
Game AKE

Input: Security parameters.

Output: ACCEPT or REJECT.

First, prepare the environment:

2. Pick RandomDecision $\leftarrow \{\text{ACCEPT, REJECT}\}$.
3. Pick $\bar{\pi}^* \leftarrow Q, \bar{\pi}^* \leftarrow Q \setminus \{\bar{\pi}^*\}$ uniformly.
4. Preset $\bar{\pi}' \leftarrow Q, \bar{\pi}' \leftarrow Q \setminus \{\bar{\pi}'\}$ uniformly. (Notice that this will be overwritten in all meaningful cases, but ensures certain independence claims.)
5. Generate a key pair $(K_{\text{CERT}}, k_{\text{CERT}}) \leftarrow \text{SIG.Keygen}(1^\kappa)$ for the certification authority and prepare a signing oracle $\mathcal{O}_{\text{EUF-CMA}}^{\text{SIGNCERT}}$ that produces signatures using $k_{\text{CERT}}$.
6. For $P \in \mathcal{L}$ do 7–8
7. Equip party $P$ with a key pair $(K_P, k_P) \leftarrow \text{SIG.Keygen}(1^\kappa)$ and define its signing oracle $\text{SIG.Sign}(k_P, .)$. 
8. Construct a certificate $\text{CERT}_P$ containing $P$’s identity $\text{ID}_P \leftarrow P$ and the public key $K_P$ and sign it using $\mathcal{O}_{\text{EUF-CMA}}$. Equip party $P$ with this certificate.

Next, call the attacker:

19. Choose a hidden bit $h \xleftarrow{\$} \{0, 1\}$.
20. Instantiate the oracles $\mathcal{O}_{\text{Send}}, \mathcal{O}_{\text{Reveal}}, \mathcal{O}_{\text{Corrupt}}, \mathcal{O}_{\text{Test}}$.
21. Call the attacker with the four oracles $\mathcal{O}_{\text{Send}}, \mathcal{O}_{\text{Reveal}}, \mathcal{O}_{\text{Corrupt}}$ and $\mathcal{O}_{\text{Test}}$ and expect its answer $h' \in \{0, 1\}$. 
After the attack, first check for an authentication break:

22. For each $\pi \in Q$ do 23–29
23. If $\pi$ has not accepted then next.
24. Let $\hat{P}$ be the alleged partner $\pi.\Pi$ of $\pi$.
25. If the corrupt oracle was used on $\hat{P}$ before $\pi$ accepted then next.
26. If there is exactly one protocol instance $\hat{\pi}$ with the opposite rôle and with all messages leading to $\pi$ accepting coincide then next.
29. Detect: there is an authentication break with protocol instance $\pi$ and alleged partner $\pi.\Pi = \hat{P}$. And break.
30. If no authentication break detected then $\pi \leftarrow Q$.
31. If an authentication break was detected then 32–42
39. If $\pi$ has $\rho =$ Initiator then 40–40
40. Return ACCEPT.
41. If $\pi$ has $\rho =$ Responder then 42–42
42. Return ACCEPT.
Second, check for a real-or-random attack:

43. Once do 44–51
44. If the test oracle $O_{\text{Test}}$ was not called then next.
45. Let $\pi'$ be the tested protocol instance.
46. Overwrite $\widehat{P}'$ with the alleged partner $\pi'.\Pi$ of $\pi'$.
47. If the corrupt oracle was used on $\widehat{P}'$ before $\pi'$ accepted then next.
48. Overwrite $\widehat{\pi}'$ such that it is the unique partner instance of $\pi'$ according to step 26.
49. If the reveal oracle was used on $\overline{\pi}'$ then next.
50. If the reveal oracle was used on $\overline{\pi}'$ then next.
51. Detect: there is a real-or-random attack. And break.
52. If the game detected a real-or-random attack then 53–54
54. If $h = h'$ then
   Return ACCEPT.
Else
   Return REJECT.
Finally, state the attacker’s defeat:

55. Return \textbf{REJECT}.

We define

$$\text{adv}^{\text{Game AKE}}(\mathcal{A})$$

$$= \text{prob (\mathcal{A} wins Game AKE | ABI)} \cdot \text{prob (ABI)} +$$

$$\text{prob (\mathcal{A} wins Game AKE | ABR)} \cdot \text{prob (ABR)} +$$

$$\left| \text{prob (\mathcal{A} wins Game AKE | RORA)} - \frac{1}{2} \right| \cdot \text{prob (RORA)} +$$

$$0 \cdot \text{prob (NOATTACK)}.$$
Implications

Does AKE security imply …

- Perfect forward secrecy?
  - Yes, due to steps 25 and 47.
- Denial of Service mitigation?
  - No.
- Replay protection? Live partner reassurance?
  - Yes, due to step 26.
- Endpoint identifier hiding?
  - No.
- Plausible deniability? Stream protection?
  - No, since the data transport is not considered at all.
  - Yes, if combined with a ‘secure’ data transport.
Unfortunately…
TLS cannot be AKE secure

Consider the following attacker:

1. Let two protocol instances of different parties execute the entire protocol until both accept.
2. Ask the test oracle for a candidate session key $s$.
3. Test whether the authenticated encryption in the server’s Finished message is authentic.
4. If so return 0 (real) otherwise return 1 (random).

This attacker has advantage almost $\frac{1}{2}$ and it is fast.

😊 Another failure...
TLS cannot be AKE secure

How to heal that?

- Solution 1: Consider TLS without the Finished messages. *But that’s not the entire protocol, authentication of large parts of the handshake is missing and we won’t be able to exclude the person-in-the-middle.*

- Solution 2: Consider truncated TLS, with unencrypted Finished messages. *But that’s not the true protocol.*

- Solution 3: Change the security notion. *AKE asks the session key to be real-or-random. That’s much more than what we want.*

We want...
Authenticated & Confidential Channel Establishment

A security notion.
Provable security for TLS:
ACCE

- Real-or-random is nice but what we need is this:
  The *data transport* is authenticated and confidential.

- Real-or-random is nice because:

  **Theorem**

  \[ \text{AKE} \land \text{sLHAE} \Rightarrow \text{ACCE}. \]

  But it is not inevitable, ie. \( \text{AKE} \not\iff \text{ACCE} \).

- Modify AKE as follows. Replace the RORA task by the sLHAE task:
  - Drop \( \mathcal{O}_{\text{Test}} \) in favour of two oracles \( \mathcal{O}_{\text{Encrypt}}, \mathcal{O}_{\text{Decrypt}} \) that reflect the sLHAE notion.
  - The hidden bit now distinguishes *left or right* rather than *real or random*. 
Input: Security parameters.

Output: ACCEPT or REJECT.

First, prepare the environment:

2. Pick RandomDecision $\xleftarrow{\xleftarrow{}\mathcal{R}} \{\text{ACCEPT, REJECT}\}$.
3. Pick $\pi^* \xleftarrow{\xleftarrow{}\mathcal{R}} Q$, $\tilde{\pi}^* \xleftarrow{\xleftarrow{}\mathcal{R}} Q \setminus \{\pi^*\}$ uniformly.
4. Preset $\pi'$ $\xleftarrow{\xleftarrow{}\mathcal{R}} Q$, $\tilde{\pi}' \xleftarrow{\xleftarrow{}\mathcal{R}} Q \setminus \{\pi'\}$ uniformly. (Notice that this will be overwritten in all meaningful cases, but ensures certain independence claims.)
5. Generate a key pair $(K_{\text{CERT}}, k_{\text{CERT}}) \leftarrow \text{SIG.Keygen}(1^\kappa)$ for the certification authority and prepare a signing oracle $O_{\text{Sign}_{\text{CERT}}}^{\text{EUF-CMA}}$ that produces signatures using $k_{\text{CERT}}$.
6. For $P \in \mathcal{L}$ do 7–8
7. Equip party $P$ with a key pair $(K_P, k_P) \leftarrow \text{SIG.Keygen}(1^\kappa)$ and define its signing oracle $\text{SIG.Sign}(k_P, \cdot)$.
8. Construct a certificate $\text{CERT}_P$ containing $P$’s identity $\text{ID}_P \leftarrow P$ and the public key $K_P$ and sign it using $O_{\text{Sign}_{\text{CERT}}}^{\text{EUF-CMA}}$. Equip party $P$ with this certificate.

12.
17.
Next, call the attacker:

20. Choose a hidden bit \( h \leftarrow \{0, 1\} \).
21. Instantiate the oracles \( \mathcal{O}_{\text{Send}}, \mathcal{O}_{\text{Reveal}}, \mathcal{O}_{\text{Corrupt}}, \mathcal{O}_{\text{Encrypt}}, \mathcal{O}_{\text{Decrypt}} \).
22. Call the attacker with the five oracles \( \mathcal{O}_{\text{Send}}, \mathcal{O}_{\text{Reveal}}, \mathcal{O}_{\text{Corrupt}}, \mathcal{O}_{\text{Encrypt}} \) and \( \mathcal{O}_{\text{Decrypt}} \) and expect its answer \( \pi' \in Q \cup \{\perp\}, h' \in \{0, 1\} \).
After the attack, first check for an authentication break:

23. For each $\pi \in Q$ do 24–30
24. If $\pi$ has not accepted then next.
25. Let $\hat{P}$ be the alleged partner $\pi.\Pi$ of $\pi$.
26. If the corrupt oracle was used on $\hat{P}$ before $\pi$ accepted then next.
27. If there is exactly one protocol instance $\hat{\pi}$ with the opposite rôle and with all messages leading to $\pi$ accepting coincide then next.
30. Detect: there is an authentication break with protocol instance $\pi$ and alleged partner $\pi.\Pi = \hat{P}$. And break.
31. If no authentication break detected then $\pi \leftarrow Q$.
32. If an authentication break was detected then 33–43
40. If $\pi$ has $\rho = \text{Client}$ then 41–41

41. Return ACCEPT.
42. If $\pi$ has $\rho = \text{Server}$ then 43–43
43. Return ACCEPT.
Second, check for a left-or-right attack against the authenticated encryption:

44. If $\pi' \in \mathcal{Q}$ then 45–50
45. Overwrite $\hat{P}'$ with the alleged partner $\overline{\pi}'$.II of $\pi'$.
46. If the corrupt oracle was used on $\hat{P}'$ before $\pi'$ accepted then break.
47. If the reveal oracle was used on $\pi'$ then break.
48. For each $\hat{\pi}'$ which is a partner instance of $\pi'$ do
49. If the reveal oracle was used on $\hat{\pi}'$ then break.
50. Detect: there is a left-or-right attack.
51. If the game detected a left-or-right attack then 52–53
52. If $h = h'$ then
   Return ACCEPT.
   Else
   Return REJECT.

Finally, state the attacker’s defeat:

54. Return REJECT.
Provable security for TLS:
Used primitives

Game EUF-CMA

Parameters: Signature scheme \((\text{KeyGen}, \text{Sign}, \text{Vrfy})\).
Input: Security parameters.
Output: \text{ACCEPT} or \text{REJECT}.

1. Prepare a key pair \((K, k) \leftarrow \text{KeyGen}(1^\kappa)\).
2. Prepare a signing oracle \(\mathcal{O}_{\text{Sign}}\). When called with \(m \in \mathcal{M}\) the oracle returns \(s \leftarrow \text{Sign}_{sk}(m)\).
3. Call the attacker \(\mathcal{A}\) with input \(1^\kappa, K\) and the oracle \(\mathcal{O}_{\text{Sign}}\). Await a pair \((m^*, s^*)\).
4. If the signing oracle has been called with input \(m^*\) then \text{REJECT}.
5. If \(\text{Vrfy}_{pk}(m^*, s^*) = \text{TRUE}\) then \text{ACCEPT}.
   Else \text{REJECT}.
Game PRF

Parameter: A (family of) keyed function(s) $F: \{0, 1\}^{κ_0} \rightarrow \{0, 1\}^{κ_1} \rightarrow \{0, 1\}^{κ_2}$.

Input: Security parameters.

Output: ACCEPT or REJECT.

1. Choose $h_{\text{PRF}} \leftarrow \{0, 1\}$.
2. Pick $k \leftarrow \{0, 1\}^{κ_0}$.
   Prepare $O_{\text{or}}^{\text{PRF}}$
   - In $w \in \{0, 1\}^{κ_1}$.
   - Out $r \in \{0, 1\}^{κ_2}$.
     - If a previous call with input $w$ exists Return old answer.
     - Else If $h_{\text{PRF}} = 0$ then Return $F_k(w) \in \{0, 1\}^{κ_2}$.
     - Else If $h_{\text{PRF}} = 1$ then Return $r \leftarrow \{0, 1\}^{κ_2}$.

3. Call the player $D$ with the oracle $O_{\text{or}}^{\text{PRF}}$ and await its guess $h'_{\text{PRF}} \in \{0, 1\}$.
4. If $h_{\text{PRF}} = h'_{\text{PRF}}$ then
   ACCEPT
   Else
   REJECT.
Game PRF-OODH

Parameter: A (family of) keyed function(s) $F: \{0, 1\}^{\kappa_0} \to \{\{0, 1\}^{\kappa_1} \to \{0, 1\}^{\kappa_2}\}$.

Input: Security parameters.

Output: ACCEPT or REJECT.

1. Pick $\pi = (G, \cdot, g, q) \leftarrow \text{Gen}(1^\kappa)$. Pick $k \leftarrow \{0, 1\}^{\kappa_0}$.
2. Choose $a, b \leftarrow \mathbb{Z}_q$.
3. Pick a hidden bit $h_{\text{PRF-OODH}} \leftarrow \{0, 1\}$.

4. Prepare an oracle $O_{\text{PRF-OODH}}$.
   \begin{itemize}
   \item \textbf{In} $m$.
   \item \textbf{Out} $s$.
   \begin{itemize}
   \item If $m$ has been seen then
     Return old answer.
   \item If $h = 0$ then
     Return $s \leftarrow F_k(g^{ab}, m)$.
   \item Else
     Return $s \leftarrow \{0, 1\}^{\kappa_2}$.
   \end{itemize}
   \end{itemize}

5. Prepare a one-time oracle $O_{\text{PRF-OODH}}$.
   \begin{itemize}
   \item \textbf{In} $B, m$.
   \item \textbf{Out} $s$.
   \begin{itemize}
   \item If $B \neq g^b$ then
     Return $F_k(B^a, m)$.
   \item Else
     Return $F_k(B^b, m)$.
   \end{itemize}
   \end{itemize}

6. Prepare a one-time oracle $O_{\text{RODH}}$.
   \begin{itemize}
   \item \textbf{In} $A, m$.
   \item \textbf{Out} $s$.
   \begin{itemize}
   \item If $A \neq g^a$ then
     Return $F_k(A^b, m)$.
   \item Else
     Return $F_k(A^a, m)$.
   \end{itemize}
   \end{itemize}

5. Call the player with $\pi$, $g^a$, $g^b$ and the oracles $O_{\text{PRF-OODH}}$, $O_{\text{PRF-OODH}}$, $O_{\text{PRF-OODH}}$.
   Await a guess $h'_{\text{PRF-OODH}} \in \{0, 1\}$.

6. If $h'_{\text{PRF-OODH}} = h_{\text{PRF-OODH}}$ then ACCEPT else REJECT.
Game WCR

Parameter: A (family of) keyed function(s) $F: \{0, 1\}^{K_0} \to \{\{0, 1\}^{K_1} \to \{0, 1\}^{K_2}\}$.
Input: Security parameters.
Output: ACCEPT or REJECT.
1. Pick $k \leftarrow \{0, 1\}^{K_0}$. Prepare $O_{\text{real function}}$ which on input $w \in \{0, 1\}^{K_1}$ does
   $F_k(w) \in \{0, 1\}^{K_2}$.
2. Call the player $D$ with the oracle $O_{\text{real function}}$ and
   await its answer $w_0', w_1' \in \{0, 1\}^{K_1}$.
3. If $w_0' \neq w_1'$ and $O_{\text{real function}}(w_0') = O_{\text{real function}}(w_1')$ then
   ACCEPT
   Else
   REJECT.

Game CR

In this stronger game the player additionally gets the key $k$.
However, for hash functions like MD5, SHA-1, SHA-256 they are equivalent if we treat their initialization vector as key.

Proof. …
**Game sLHAE**

Parameter: An authenticated encryption scheme \( \text{AE} = (\text{AE.Keygen}, \text{AE.Enc}, \text{AE.Dec}) \).

Input: Security parameters.

Output: ACCEPT or REJECT.

1. \((k, \text{st}^{\text{Enc}}, \text{st}^{\text{Dec}}) \leftarrow \text{AE.Keygen}(1^\kappa)\).
2. \(h_{\text{AE.Enc}} \leftarrow \{0, 1\}, \ h_{\text{AE.Dec}} \leftarrow \{0, 1\}\).
3. Prepare the oracle \( \mathcal{O}_{\text{Enc}} \) (à la LOR-CPA):
   \[
   \begin{align*}
   \text{In} & \quad \ell, x, m_0, m_1. \\
   \text{Out} & \quad \text{A ciphertext } c \text{ or FAIL.} \\
   & \quad (c_0, \text{st}^{\text{Enc}}_0) \leftarrow \text{AE.Enc}(k, \ell, x, m_0, \text{st}^{\text{Enc}}). \\
   & \quad (c_1, \text{st}^{\text{Enc}}_1) \leftarrow \text{AE.Enc}(k, \ell, x, m_1, \text{st}^{\text{Enc}}). \\
   & \quad \text{If } c_0 = \text{FAIL or } c_1 = \text{FAIL then} \\
   & \quad \quad \text{Return FAIL.} \\
   & \quad \quad \text{Put } \text{st}^{\text{Enc}} \leftarrow \text{st}^{\text{Enc}}_{\text{AE.Enc}}. \\
   & \quad \quad \text{Return } c_{h_{\text{AE.Enc}}}. \\
   \end{align*}
   \]

4. Invoke the player with input \((\mathcal{O}_{\text{Enc}}, \mathcal{O}_{\text{Dec}})\) to obtain a choice \( c \in \{\text{Enc, Dec}\} \) a bit \( h'_{\text{AE}} \in \{0, 1\}\).
5. If \( h_{\text{AE},c} = h'_{\text{AE}} \) then
   \[
   \text{ACCEPT}
   \]
   Else
   \[
   \text{REJECT.}
   \]

Prepare the oracle \( \mathcal{O}_{\text{Dec}} \) (by EUF-CMA):

\[
\begin{align*}
\text{In} & \quad x, c. \\
\text{Out} & \quad \text{A message } m \text{ or FAIL.} \\
& \quad \text{If } h_{\text{AE.Dec}} = 0 \text{ then} \\
& \quad \quad \text{Return FAIL.} \\
& \quad \quad (m, \text{st}^{\text{Dec}}) \leftarrow \text{AE.Dec}(k, x, c, \text{st}^{\text{Dec}}). \\
& \quad \quad \text{If } c \text{ and all prior ones were created by} \\
& \quad \quad \quad \mathcal{O}_{\text{dec}} \text{ in this order} \\
& \quad \quad \quad \text{then} \\
& \quad \quad \quad \quad \text{Return FAIL.} \\
& \quad \quad \text{Return } m.
\end{align*}
\]
Provable security for TLS:

Used primitives

- Signatures: EUF-CMA.
- Pseudo-random function: PRF.
- Diffie-Hellman group: PRF-OODH.
- Hash function: collision-resistant (tricky!).
- Data transport: sLHAE.
Provable security for TLS:  
Handshake details

Client → Server

ClientHello
ServerHello
Certificate*
ServerKeyExchange*
CertificateRequest*
ServerHelloDone

Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished

[ChangeCipherSpec]
Finished
**Provable security for TLS:**

**Handshake details**

**Client Request**
1. $\rho \leftarrow \text{Client}$.
2. $r_{\text{Client}} \leftarrow \{0, 1\}^\lambda$.
3. $m_1 \leftarrow (r_{\text{Client}}, \text{cs-list})$.

**Server Response**
1. $\rho \leftarrow \text{Server}$.
2. $r_{\text{Server}} \leftarrow \{0, 1\}^\lambda$.
3. $t_{\text{Server}} \leftarrow z_q, T_{\text{Server}} \leftarrow g^{t_{\text{Server}}} \text{ in } \mathbb{Z}_p^\times$.
4. $\sigma_{\text{Server}} \leftarrow \text{SIG.Sign}(k_{\text{Server}}, r_{\text{Client}}, r_{\text{Server}}, p, g, T_{\text{Server}})$.
5. $m_2 \leftarrow (r_{\text{Server}}, \text{cs-choice})$.
6. $m_3 \leftarrow \text{cert}_{\text{Server}}$.
7. $m_4 \leftarrow (p, g, T_{\text{Server}}, \sigma_{\text{Server}})$.
8. $m_5 \leftarrow \text{get-cert}$.
9. $m_6 \leftarrow \text{done}$.

**Client Response**
3. $\Pi \leftarrow \text{Server determined from cert}_{\text{Server}}$.
4. If not $\text{SIG.Vfy}(K_{\Pi}, \sigma_{\text{Server}}, r_{\text{Client}}, r_{\text{Server}}, p, g, T_{\text{Server}})$ then
   $\Lambda \leftarrow \text{REJECT}$ and abort.
5. $t_{\text{Client}} \leftarrow z_q, T_{\text{Client}} \leftarrow g^{t_{\text{Client}}} \text{ in } \mathbb{Z}_p^\times$.
6. $m_7 \leftarrow \text{cert}_{\text{Client}}, m_8 \leftarrow T_{\text{Client}}$.
7. $\Sigma_{\text{Client}} \leftarrow \text{SIG.Sign}(k_{\text{Client}}, m_1, \ldots, m_8)$.
8. $\text{prf} \leftarrow T_{\text{Server}}^{r_{\text{Client}}} \text{ in } \mathbb{Z}_p^\times$,
   $\text{ms} \leftarrow \text{PRF}(\text{prf}, r_{\text{Client}}, r_{\text{Server}})$.
9. $\text{sessionkey} \leftarrow \text{PRF}(\text{ms}, r_{\text{Client}}, r_{\text{Server}})$.
10. $m_9 \leftarrow \Sigma_{\text{Client}}, m_{10} \leftarrow \text{change-cipher}$
11. $\text{fin}_{\text{Client}} \leftarrow \text{PRF}(\text{ms}, m_{10}, H(m_1, \ldots, m_{10}))$.
12. $(m_{11}, \text{stEnc}) \leftarrow \text{AE.Enc}((\text{sessionkey}_{\text{Client}}, \ell, H, \text{fin}_{\text{Client}}, \text{stEnc})$.

**Server Accept**
4.1 $\Pi \leftarrow \text{Client determined from cert}_{\text{Client}}$.
4.2 If not $\text{SIG.Vfy}(K_{\Pi}, \sigma_{\text{Client}}, m_1, \ldots, m_8)$ then
   $\Lambda \leftarrow \text{REJECT}$ and abort.
4.3 $\text{prf} \leftarrow T_{\text{Client}}^{t_{\text{Server}}} \text{ in } \mathbb{Z}_p^\times$,
   $\text{ms} \leftarrow \text{PRF}(\text{prf}, r_{\text{Client}}, r_{\text{Server}})$.
4.4 $\text{sessionkey} \leftarrow \text{PRF}(\text{ms}, \text{label}_2, r_{\text{Client}}, r_{\text{Server}})$.
4.5 $m_{12} \leftarrow \text{change-cipher}$
4.6 $\text{fin}_{\text{Server}} \leftarrow \text{PRF}(\text{ms}, \text{label}_3, H(m_1, \ldots, m_{10}), m_{12})$.
4.7 $\text{fin}_{\text{Server}} \leftarrow \text{PRF}(m_{12}, \text{stEnc}, H(m_1, \ldots, m_{10}), \text{fin}_{\text{Client}}, m_{12})$
4.8 $(m_{13}, \text{stEnc}) \leftarrow \text{AE.Enc}((\text{sessionkey}_{\text{Server}}, \ell, H, \text{fin}_{\text{Server}}, \text{stEnc})$.
4.9 $(\text{fin}_{\text{Client}}, \text{stDec}) \leftarrow \text{AE.Dec}((\text{sessionkey}_{\text{Client}}, \ell, H, m_{11}, \text{stDec})$.
4.10 If $\text{fin}_{\text{Client}} \neq \text{fin}_{\text{Server}}$ then
   $\Lambda \leftarrow \text{REJECT}$ and abort.
4.11 $\Lambda \leftarrow \text{ACCEPT}$ and return sessionkey.

**Client Accept**
5.1 $\text{fin}_{\text{Server}} \leftarrow \text{PRF}(m_{13}, \text{stDec}, m_{11}, \text{fin}_{\text{Server}}, m_{12})$.
5.2 $\text{fin}_{\text{Server}} \leftarrow \text{AE.Dec}(\text{sessionkey}_{\text{Server}}, H, m_{13}, \text{stDec})$.
5.3 If $\text{fin}_{\text{Server}} \neq \text{fin}_{\text{Server}}$ then
   $\Lambda \leftarrow \text{REJECT}$ and abort.
5.4 $\Lambda \leftarrow \text{ACCEPT}$ and return sessionkey.
Theorem (Jager, Kohlar, Schäge & Schwenk 2011)

Assume that

- Each primitive is \((t, \epsilon^X)\)-secure...
- The attacker controls the entire network and triggers any actions.
- Each party has \(d = \frac{q}{\ell}\) protocol instances.

Then in time \(t - u\) the attacker’s advantage is bounded:

\[
\text{adv}^\text{ACCE}(\mathcal{A}) \leq 4q \left( \frac{q}{2^\lambda} + \ell \epsilon_{\text{EUF-CMA}} + \epsilon_{\text{DDH}} + q \epsilon_{\text{PRF-ODH}} + \epsilon_{\text{PRF}} + \epsilon_{\text{sLHAE}} + (q + 1) \left( \epsilon_{\text{PRF}} + \epsilon_{H} + \epsilon_{\text{sLHAE}} + \frac{1}{2^\mu} \right) \right).
\]
Get together with your group.

1. Prepare a poster for your protocol as a basis for a presentation.
2. Present your protocol to the other groups.

The aim of this project is to give all participants

- insight on the respective protocol, i.e. how it works, which resources it uses, where it is located, how it is used, and so on,
- a first idea of its security.
Interludium: Student project conference:
ssh: secure shell

SSH: SECURE SHELL

What is SSH?

- Protocol for secure remote login (insecure network)

Example usage?

- Manage a remote Linux server
- Quick info?

- Server authentication
- Confidentiality
- Integrity
- (authenticate client to server)

How does it work?

Client
- Identification string
  - ident. string
  - algorithm req. list

Server
- DH request
  - DH params
- $a \in \mathbb{Z}_q$
  - \( A = g^a \)
- \( \text{host key, cert, id, H} \)
- Sign (H)
  - (new sess) processed

Data exchange:
- encrypt length padding length payload padding
- mac
  - Seq.nr, unencrypted packet content
Interludium: Student project conference:
Signal protocol: X3DH and double ratchet
Layer 3
- Security independent of applications
- Below TCP
- Less control

Security Associations
- Specify crypto algorithms, keys, etc
- Separate SAs for incoming & outgoing traffic
- Separate for AH & ESP protected data
- Set up & managed manually or automatically

DOS
- Too many half-open IKE SAs
  - Respondes includes a cookie in IKE_SA_INIT response
  - Defers crypto calculation until Resp. gets the cookie back

IKE
- Internet Key Exchange
- Establishing and managing SAs
  - HDR, SAi1, KEi, Ni

IKE consists of two exchanges: IKE_SA_INIT, IKE_AUTH
IKE_SA_INIT establishes a SA that protects cipher negotiation
IKE_AUTH provides authentication of both parties and establishes the first SA to be used during the subsequent conversation
- Uses DHE key exchange (DH values are KEi and KEj)
- PFS
  - Protection against MITM

Endpoint ID hiding
- Respondes is hidden
- Active attacks can learn initiator identity

Nonces provide randomness to each new key exchange (Nonces are Ni and Nj)
- Live partner reassurance
(i) Indistinguishability hop

In the situation

\[
\begin{array}{c|c|c|c}
 \uparrow & G & \downarrow \\
 \hline
 G_0 & \mathcal{D} & G_1 \\
\end{array}
\]

with \( \text{adv}^G(\mathcal{D}) = \left| \text{prob}(\mathcal{D} \text{ wins } G) - \frac{1}{2} \right| \). Then

\[
\text{prob}(\mathcal{A} \text{ wins } G_0) - \text{prob}(\mathcal{A} \text{ wins } G_1) \leq 2 \text{adv}^G(\mathcal{A}\mathcal{D})
\]

and

\[
\text{adv}^{G_0}(\mathcal{A}) - \text{adv}^{G_1}(\mathcal{A}) \leq 2 \text{adv}^G(\mathcal{A}\mathcal{D}).
\]
(ii) Small error hop

Assume \( S_0 \Delta S_1 \subset E \).
Then
\[
| \text{prob} (S_0) - \text{prob} (S_1) | \leq \text{prob} (E).
\]

- Use if we know \( \text{prob} (E) \leq B \).
- Say \( G \) is a game with constant guessing probability \( \alpha \) and \( \text{adv}^G (\mathcal{P}) = | \text{prob} (\mathcal{P} \text{ wins } G) - \alpha | \). If we have a reduction \( \mathcal{R} \) to \( G \) that on \( E \) always wins and outside \( E \) always guesses then

\[
\text{prob} (E) \leq \frac{1}{1 - \alpha} \text{adv}^G (A \mathcal{R}).
\]
(iii) Large error hop

Assume $S_0 \Delta S_1 \subset E$ and $(S_0, E)$ are independent.

1. If additionally $\text{prob}(S_1 \mid E) = 0$ then

$$\text{prob}(S_0) = \frac{1}{\text{prob}(\neg E)} \text{prob}(S_1).$$

Note: $S_1 = S_0 \cap \neg E \implies S_0 \Delta S_1 \subset E \land \text{prob}(S_1 \mid E) = 0$.

2. If additionally $\text{prob}(S_1 \mid E) = \alpha \neq 0$ then

$$|\text{prob}(S_0) - \alpha| = \frac{1}{\text{prob}(\neg E)} |\text{prob}(S_1) - \alpha|.$$
Define events LORA, ABC, ABS and NOATTACK by reaching certain steps in the game. And, say, define Game LORA by making all other decisions REJECT. Put

$$\text{adv}^{\text{Game AKE}}(\mathcal{A}) = \text{adv}^{\text{Game LORA}}(\mathcal{A}) +$$
$$\text{adv}^{\text{Game ABC}}(\mathcal{A}) +$$
$$\text{adv}^{\text{Game ABS}}(\mathcal{A}) +$$
$$0 \cdot \text{prob}(\text{NOATTACK})$$

with

$$\text{adv}^{\text{Game LORA}}(\mathcal{A}) = |\text{prob}(\mathcal{A} \text{ wins Game AKE} | \text{LORA}) - \frac{1}{2}| \cdot \text{prob}(\text{LORA}),$$

$$\text{adv}^{\text{Game ABC}}(\mathcal{A}) = |\text{prob}(\mathcal{A} \text{ wins Game AKE} | \text{ABC}) - 0| \cdot \text{prob}(\text{ABC}) = \text{prob}(\text{ABC}),$$

$$\text{adv}^{\text{Game ABS}}(\mathcal{A}) = |\text{prob}(\mathcal{A} \text{ wins Game AKE} | \text{ABS}) - 0| \cdot \text{prob}(\text{ABS}) = \text{prob}(\text{ABS}).$$
Provable security for TLS:
About the proof: LORA

Modified steps for Game A1 based on Game LORA via $\mathcal{R}_{\text{LORA} \rightarrow \text{A1}}^{\pi^*, \hat{\pi}^*}$.

52. 

Theorem

For all attackers $A$ we have

$$\text{adv}^{\text{Game LORA}}(A) \leq q^2 \cdot \text{adv}^{\text{Game A1}}(A).$$

Proof.

Use ???.

$\square$
9. Recall that each server protocol instance computes in step 2.3:
   \( t_{\text{Server}} \leftarrow Z_q, T_{\text{Server}} \leftarrow g^{t_{\text{Server}}} \in \mathbb{Z}_p^x \).
   Next, each client protocol instance computes in steps 3.3, 3.6 and 3.7:
   \( t_{\text{Client}} \leftarrow Z_q, T_{\text{Client}} \leftarrow g^{t_{\text{Client}}} \in \mathbb{Z}_p^x \).

   pms \leftarrow T_{\text{Server}} \cdot t_{\text{Client}} \in \mathbb{Z}_p^x \),
   ms \leftarrow \text{PRF}(\text{pms, label}_1 \mid t_{\text{Client}} \mid T_{\text{Server}}),
   \text{sessionkey} \leftarrow \text{PRF}(\text{ms, label}_2 \mid t_{\text{Client}} \mid T_{\text{Server}}).

   Further, each server computes in step 4.3 and 4.4:
   pms \leftarrow T_{\text{Client}} \cdot t_{\text{Server}} \in \mathbb{Z}_p^x \),
   ms \leftarrow \text{PRF}(\text{pms, label}_1 \mid t_{\text{Client}} \mid T_{\text{Server}}),
   \text{sessionkey} \leftarrow \text{PRF}(\text{ms, label}_2 \mid t_{\text{Client}} \mid T_{\text{Server}}).

**Proof:**

Use Game Hopping Lemma (i)...
Modified steps for Game A3 based on Game A2 via $R_{A2 \rightarrow A3}^\text{PRF}$.

12. Initialize $ms^0$ with a never occurring special value $\#$.

Just after the (first) client $\pi^*$ among the guessed protocol instances $\tilde{\pi}^*$ and $\hat{\pi}^*$ has executed step 3.7 and if it has $ms$ modified then read off the shared key seed in $\pi^*$: $ms^0 \leftarrow \pi^*ms$, $r_{\text{Client}}^0 \leftarrow \pi^*r_{\text{Client}}$, $r_{\text{Server}}^0 \leftarrow \pi^*r_{\text{Server}}$.

14. Recall that each protocol instance computes in step 3.7 or 4.4:

\[ \text{sessionkey} \leftarrow \text{PRF}(ms, label_2 | r_{\text{Client}} | r_{\text{Server}}). \]

Embed the challenge $O_{\text{ror}}$ of Game PRF:

Modify the behavior of each protocol instance $\pi$ in step 3.7 and 4.4:

\[ \begin{align*}
\text{If } ms \text{ modified and } \\
r_{\text{Client}} | r_{\text{Server}} = r_{\text{Client}}^0 | r_{\text{Server}}^0 \\
\text{and so } ms = ms^0 \\
\text{then} \\
& \text{sessionkey} \leftarrow O_{\text{ror}}(\text{label}_1 | r_{\text{Client}} | r_{\text{Server}}). \\
\text{Else} \\
& \text{sessionkey} \leftarrow \text{PRF}(ms, label_2 | r_{\text{Client}} | r_{\text{Server}}).
\end{align*} \]

Game A3, Game C6 or Game S6

Prepare a truly random function $\text{trf}_2$.

Modify the behavior of each protocol instance $\pi$ in step 3.7 and 4.4:

\[ \begin{align*}
\text{If } ms \text{ modified and } \\
r_{\text{Client}} | r_{\text{Server}} = r_{\text{Client}}^0 | r_{\text{Server}}^0 \\
\text{and so } ms = ms^0 \\
\text{then} \\
& \text{sessionkey} \leftarrow \text{trf}_2(\text{label}_1 | r_{\text{Client}} | r_{\text{Server}}). \\
& * \text{sessionkey modified} * \\
\text{Else} \\
& \text{sessionkey} \leftarrow \text{PRF}(ms, label_2 | r_{\text{Client}} | r_{\text{Server}}).
\end{align*} \]

Proof.

Use Game Hopping Lemma (i)…

\[ \square \]
Modified steps for Game A4 based on Game A3 via $\mathcal{R}_{A3 \rightarrow A4}$.

17. Initialize sessionkey$^0$ with a never occurring special value $\texttt{\textbackslash x}$. When first protocol instance $\pi^*$ among the guessed ones $\widehat{\pi}^*$ and $\tilde{\pi}^*$ reaches step 3.7 or 4.4 and if it has sessionkey modified then let sessionkey$^0$ be the constructed session key $\pi^*$sessionkey.

18. \[ \mathcal{R}_{\text{sLHAE}} \]

Embed the challenge $O_{\text{sLHAE}}^{\text{Enc}}$ and $O_{\text{sLHAE}}^{\text{Dec}}$ into $\pi^*$ and $\tilde{\pi}^*$, respectively, provided the available session key is sessionkey$^0$.

Game A4

Modify $O_{\text{Enc}}$ in $\pi^*$ and $O_{\text{Dec}}$ in $\tilde{\pi}^*$ provided the available session key is sessionkey$^0$ to not reveal any information about $h_{\text{AE}}$, by using a fake hidden bits $h_{\text{AE}, \text{Enc}}$ in $O_{\text{Enc}}$, it returns an encryption of $m_{h_{\text{AE}, \text{Enc}}}$, and $O_{\text{Dec}}$ always returns FAIL.

Proof.

Use Game Hopping Lemma (ii).
Theorem

For all attackers $A$ we have $\text{adv}^{\text{Game } A^4}(A) = 0$.

Proof. ...
Lemma (Left-or-right security; \( \approx \)Lemma 5, Jager, Kohlar, Schäge & Schwenk 2011)

For all attackers \( \mathcal{A} \) on Game LORA we have

\[
\text{adv}^{\text{Game LORA}}(\mathcal{A}) \leq q^2 \cdot (2\epsilon_{\text{PRF-OODH}} + 2\epsilon_{\text{PRF}} + 2\epsilon_{\text{sLHAE}}).
\]
Krawczyk, Paterson & Wee (2013)
On the Security of the TLS Protocol: A Systematic Analysis

- Server only authentication: from ACCE to SACCE.
- Selective security.
  - Attacker commits to attack goal before doing anything else.
- TLS labeled KEMs.
  - Key exchange constructed from a KEM.
  - Consider IND-CCA for KEMs.

CCCA = constrained chosen ciphertext attack. Constraint is
  - a predicate specified by attacker under condition that it is ‘difficult’ to fulfill or
  - mostly fixed a priori, eg. by —like here— a pattern with some part that the attacker has to specify.
Theorem

If

- $\text{tlskem}$ is IND-CCA secure,
- $\text{casig}$ is an existentially unforgeable signature scheme and
- $\text{stE}$ is $sLHAE$-secure

then $\text{TLS}$ is SACCE-secure.
Provable security for TLS:
Further security results

- TLS-RSA (OW-PCA)
- TLS-CCA (IND-PCA)
- TLS-DH (PRF-ODH)
- TLS-DHE (PRF-ODH)

IND-CCCA

+PRF +RO

SACCE

server-only auth

+slHAE +SIG

ACCE

mutual auth

+PRF ?+?

+PRF ?+SIG?
Provable security for TLS:
Further security results

Bhargavan, Fournet, Kohlweiss, Pironti, Strub & Zanella-Béguelin (2014)
Proving the TLS Handshake Secure (as it is)

- Agile security, i.e. includes algorithm negotiation.
  - New problem: Cross-ciphersuite attacks.
- Unilateral and mutual authentication.
- Key exchanges considered:
  - RSA.
  - PSK.
  - DHE.
  - ECDHE.
- Also: session resumption and renegotiation.
- Handshake coded in 3600 lines F# (≈45pp).
- Automated proof: 3000 lines EasyCrypt (≈38pp).
- Problem: TLS releases algorithms and keys to record layer before handshake is complete.
- Complements Bhargavan, Fournet, Kohlweiss, Pironti & Strub (2013) on miTLS.
How can attacks and security proof coexist?

- Heartbleed
  - Implementation error.
- CRIME
  - With compression AE not length-hiding.
  - Additional oracles.
- Lucky13
  - Additional oracles, eg. timing.
- RC4 weakness
  - RC4 is simply not sLHAE secure.
Provable security for TLS:
This is by far not all

- DTLS (Datagram TLS / UDP).
- EAP-TLS @ WPA, WPA2. (Extensible Authentication Protocol)
- TLS-SRP (secure remote password), PAKE (password authenticated key exchange).
- TLS Extensions. Session resumption, ...
- ...
Part II

ePassports and identity cards

Introduction

MRTDs (Machine Readable Travel Documents)

eMRTDs

Security mechanisms

eMRTD PKI

ePassport application (BSI summary and extension)
Introduction

ICAO Doc 9303
BSI TR03110

MRTDs (Machine Readable Travel Documents)

eMRTDs

Security mechanisms

eMRTD PKI

ePassport application (BSI summary and extension)

eIDAS application (BSI TR03110)

Security?

German ID card and residence permit (BSI TR03127)

European projects, regulations, applications
Introduction

click here

click here
Who standardizes travel documents?

Globally:

▶ ICAO (International Civil Aviation Organization).
▶ ...part of the UN.

Per country, example Germany:

▶ BSI (Bundesamt für Sicherheit in der Informationstechnik, Bonn)
Introduction:
ICAO Doc 9303

- Specifies features of any kind of MRTD.
  - TD1 size: card, MRZ on back side, eg. German identity card.
  - TD2 size: card.
  - TD3 size: passport, eg. German passport.
- 12 parts.
  1. Introduction.
  3. Specifications common to all MRTDs.
  4. Specifications for Machine Readable Passports (MRPs) and other TD3 size MRTDs.
  5. Specifications for TD1 size Machine Readable Official Travel Documents (MROTDs).
  8. Emergency Travel Documents.
  9. Deployment of Biometric Identification and Electronic Storage of Data in MRTDs.
  10. Logical Data Structure (LDS) for Storage of Biometrics and Other Data in the Contactless Integrated Circuit (IC).
  12. Public Key Infrastructure for MRTDs.

Part 4: Applications and Document Profiles

Part 1: ePassports

Part 2: Protocols for electronic IDentification, Authentication and trust Services (eIDAS)

Part 3: Common Specifications
Introduction

MRTDs (Machine Readable Travel Documents)
  - Main threats
  - Physical passport layout
  - Biometrics
  - MRZ

eMRTDs

Security mechanisms

eMRTD PKI

ePassport application (BSI summary and extension)

eIDAS application (BSI TR03110)

Security?

German ID card and residence permit (BSI TR03127)

European projects, regulations, applications
MRTDs (Machine Readable Travel Documents):
Main threats

- Counterfeiting a complete travel document.
- Photo substitution.
- Deletion/alteration of data in the visual or machine readable zone of the MRP data page.
- Construction of a fraudulent document, or parts thereof, using materials from legitimate documents.
- Removal and substitution of entire page(s) or visas.
- Deletion of entries on visa pages and the observations page.
- Theft of genuine document blanks.
- Impostors (assumed identity; altered appearance).
- Tampering with the contactless IC (where present) either physically or electronically.
Detection of security features can be at any or all of the following three levels of inspection:

- Level 1 — Cursory examination for rapid inspection at the point of usage (easily identifiable visual or tactile features).
- Level 2 — Examination by trained inspectors with simple equipment.
- Level 3 — Inspection by forensic specialists.
MRTDs (Machine Readable Travel Documents):
Physical passport layout

- Feature types (ICAO Doc9303v7, part 2, §3.1, §A.5).
  - Structure.
  - Substance.
  - Data.

- Security features and techniques (ICAO Doc9303v7, part 2, §A.5).
  - Substrate materials.
  - Security design and printing.
  - Protection against copying, counterfeiting or fraudulent alteration.
  - Personalization techniques.
MRTDs (Machine Readable Travel Documents):

Biometrics

- Primary biometric: facial image.
  - ...stored as JPEG compressed to approx. 12kB
- Additional biometrics:
  - Fingerprint.
    - ...stored as [10kB per finger] JPEG plus optional template.
  - Iris.
    - ...stored as image plus optional template.
MRTDs (Machine Readable Travel Documents):
MRZ

click here
Introduction

MRTDs (Machine Readable Travel Documents)

*eMRTDs*
- Radio connection
- Pros & cons
- Security goals
- Data on chip

Security mechanisms

*eMRTD PKI*

*ePassport application (BSI summary and extension)*

*eIDAS application (BSI TR03110)*

Security?

*German ID card and residence permit (BSI TR03127)*

*European projects, regulations, applications*
eMRTDs:
Radio connection
Symbol and use.

Contactless IC (acc.to ISO/IEC14443 Type A or Type B). RFID (radio frequency identification) w/o own power supply.

Terminal, aka. inspection system, IFD (InterFace Device), PCD (Proximity Coupling Device).

Required read range at least 10 cm.
De facto: $< 1.5 \text{ m}$, $\sim 20 \text{ cm}$

Transmission speed $\geq 424 \text{ kbit/sec}$.

Data storage MUST be at least 32kB.

LDS (logical data structure).

Security and privacy of the stored data.
eMRTDs: 
Pros & cons

- IC is an additional security feature.
- Radio conversation may be overheard.
- Contactless channel may be read without notice.
eMRTDs:
Security goals

- Prevent skimming of data from the contactless IC.
- Prevent eavesdropping on the communication between contactless IC and reader.
- Provide authentication of the data stored on the contactless IC based on the Public Key Infrastructure (PKI).
- Provide authentication of the contactless IC itself.
eMRTDs:
Data on chip

- ePassport application’s elementary file EF. DG1–DG16. MUST be write protected.
- Security object EF. SO_D. SignedData, RFC 3369. ...signs a list of DG hashes. Important since eg. DG3 and DG4 are not readable for any device.
Section Overview

Introduction

MRTDs (Machine Readable Travel Documents)

eMRTDs

Security mechanisms
  Tasks
    Accessing
      Gain access and establish secure messaging
        BAC
        PACE
  Authenticating
    PA
    AA
    CA
    PACE with Chip Authentication Mapping
  Secure Messaging

eMRTD PKI

ePassport application (BSI summary and extension)

eIDAS application (BSI TR03110)

Security?

German ID card and residence permit (BSI TR03127)

European projects, regulations, applications
Security mechanisms:

Tasks

- Passive authentication.
  
  \(\ldots\) uses a digital signature to authenticate data stored in the data groups on the MRTD chip.

- Active authentication.
  
  \(\ldots\) is a digital security feature that prevents cloning by introducing a chip-individual key pair [enabling that] the chip can prove knowledge of this private key in a challenge-response protocol\(\ldots\).

- Access control.
  
  \(\ldots\) is not only required for privacy reasons but also mitigates the risk of cloning attacks.
Security mechanisms:

Accessing

1. Read EF.CardAccess.
2. Gain access.
   ▶ BAC (Basic Access Control).
   ▶ PACE (Password Authenticated Connection Establishment).
3. Authentication of data.
   ▶ PA (Passive Authentication).
4. Authentication of the chip.
   ▶ AA (Active Authentication).
   ▶ CA (Chip Authentication).
5. Gain access to additional biometrics.
   ▶ EAC (Extended Access Control).
   ▶ Keep some data encrypted.
6. Read data.
   ▶ Secure Messaging.
7. Comparision of conventional MRZ (OCR-B) and IC-based MRZ (LDS).
8. ...
Preconditions on the inspection system for

- **BAC (Basic Access Control).**
  - Read MRZ via OCR and derive document basic access keys.
  - Support BAC and Secure Messaging.
- **PACE (Password Authenticated Channel Establishment).**
  - Read MRZ or CAN via OCR.
  - Support PACE and Secure Messaging.
- **PA (Passive Authentication).**
  - Securely stored copies of each state’s Country Signing CA Certificate $C_{CSCA}$.
  - Or, ‘all’ Document Signer Certificates $C_{DS}$.

Before use: establish trust!

- **AA (Active Authentication).**
  - Read MRZ via OCR.
  - Support AA.
- **CA (Chip Authentication).**
  - Read MRZ via OCR.
  - Support CA.
- **EAC (Extended Access Control) to additional biometrics.**
  - Depends on state’s definitions.
- **Decryption of additional biometrics.**
  - Depends on state’s definitions.
Security mechanisms:
Gain access and establish secure messaging

BAC

**IFD: Get challenge**
1.1 Derive Document Basic Access Keys \( K_{Enc}, K_{MAC} \) from the MRZ (or CAN).
1.2 Send GET CHALLENGE.

**IC: Send challenge**
2.1 Pick RND.IC ∈ \( \{0, 1\}^{64} \).
2.2 Send challenge.

**IFD: External authentication**
3.1 Choose a nonce RND.IFD ∈ \( \{0, 1\}^{64} \) and a key K.IFD ∈ \( \{0, 1\}^? \).
3.2 \( S \leftarrow \text{RND.IFD} | \text{RND.IC} | \text{K.IFD} \).
3.3 Encrypt \( E_{\text{IFD}} \leftarrow E(K_{Enc}, S) \).
3.4 Compute \( M_{\text{IFD}} \leftarrow \text{MAC}(K_{MAC}, E_{\text{IFD}}) \).
3.5 Send EXTERNAL AUTHENTICATE \( E_{\text{IFD}} | M_{\text{IFD}} \).

**IC: Respond**
4.1 Check \( M_{\text{IFD}} \).
4.2 Decrypt \( E_{\text{IFD}} \).
4.3 Check RND.IC in \( S \).
4.4 Generate \( K.IC \in \{0, 1\}^? \).
4.5 \( R \leftarrow \text{RND.IC} | \text{RND.IFD} | \text{K.IC} \).

**IFD: Check**
5.1 Check \( M_{\text{IC}} \).
5.2 Decrypt \( E_{\text{IC}} \).
5.3 Check RND.IFD in \( R \).

**Both: Derive keys**
6.1 \( K_{Enc} | K_{MAC} \leftarrow \text{KDF}(K.IC \oplus K.IFD) ∈ \{0, 1\}^{256} \).
Security mechanisms:
Gain access and establish secure messaging

BAC

- Document Basic Access Keys for BAC are

\[ K_{\text{Enc}}^{64} | K_{\text{MAC}}^{64} | *^{32} \leftarrow \text{SHA-1(MRZ\_information)}. \]

MRZ\_information consists of document number, date of birth and date of expiry including check digits.

\⇒\ At best 56–73 bits of entropy, much less if document number is not random or the attacker can restrict ranges for the dates. Paranoid scenario: no entropy.

- Encryption: 3DES-CBC with zero IV without any padding.

- Authentication: DES-CBC-Retail-MAC with zero IV, certain padding, 8 bytes.
Security mechanisms:
Gain access and establish secure messaging

PACE

- Provides DH key exchange. BAC does not!
- Allows to use CAN as password.
  - CAN (card access number) MUST be randomly or pseudo-randomly chosen. \(\leadsto\) more entropy.
  - CAN is easier to type manually than the MRZ.
Security mechanisms:
Gain access and establish secure messaging

PACE: EC, generic

**PCD: Init**
1.1 Derive the pass key
  \( K_\pi \leftarrow \text{KDF(MRZ)} \) or
  \( K_\pi \leftarrow \text{KDF(CAN)} \).
1.2 Manage Security
  Environment: choose algorithms and password
  (MRZ or CAN).

**IC: Send nonce**
2.1 Static domain pars \( D_{IC} \).
2.2 Derive \( K_\pi \) from MRZ or
  CAN.
2.3 Pick \( s \in \mathbb{Z}_q \).
2.4 \( z \leftarrow E(K_\pi, s) \).
2.5 Send challenge.

**PCD: Start generic mapping**
3.1 \( s \leftarrow D(K_\pi, z) \).
3.2 \( k_{\text{Map},\text{PCD}} \leftarrow \mathbb{Z}_q \).
3.3 \( K_{\text{Map},\text{PCD}} \leftarrow k_{\text{Map},\text{PCD}}P \).

**IC: Generic Mapping**
4.1 \( k_{\text{Map},\text{IC}} \leftarrow \mathbb{Z}_q \).
4.2 \( K_{\text{Map},\text{IC}} \leftarrow k_{\text{Map},\text{IC}}P \).
4.3 \( H \leftarrow k_{\text{Map},\text{IC}}K_{\text{Map},\text{PCD}} \).

**PCD: Finish generic mapping**
5.1 \( D \leftarrow \text{Map}(D_{IC}, s, H) \),
  namely: \( \hat{P} \leftarrow sP + H \).
5.2 \( K \leftarrow k_{\text{DH},\text{PCD}}K_{\text{DH},\text{IC}} \).
5.3 Both: Derive keys
6.1 \( K_{\text{MAC}} \leftarrow \text{KDF}_{\text{MAC}}(K) \).
6.2 \( K_{\text{Enc}} \leftarrow \text{KDF}_{\text{Enc}}(K) \).

**PCD: Start DH**
7.1 \( k_{\text{DH},\text{PCD}} \leftarrow \mathbb{Z}_q \).
7.2 \( K_{\text{DH},\text{PCD}} \leftarrow k_{\text{DH},\text{PCD}}\hat{P} \).

**IC: Respond DH**
8.1 \( k_{\text{DH},\text{IC}} \leftarrow \mathbb{Z}_q \).
8.2 \( K_{\text{DH},\text{IC}} \leftarrow k_{\text{DH},\text{IC}}\hat{P} \).
8.3 \( K \leftarrow k_{\text{DH},\text{IC}}K_{\text{DH},\text{PCD}} \).

**PCD: Authenticate**
11.1 \( T_{\text{PCD}} \leftarrow \text{MAC}(K_{\text{MAC}}, K_{\text{IC}}) \).

**IC: Authenticate**
12.1 Check \( T_{\text{PCD}} \).
12.2 \( T_{\text{IC}} \leftarrow \text{MAC}(K_{\text{MAC}}, K_{\text{PCD}}) \).

**PCD: Verify**
13.1 Check \( T_{\text{IC}} \).

\( (2017-07-17) \ 218+45 \)
Security mechanisms:
Gain access and establish secure messaging

PACE: EC, generic with Chip Authentication Mapping

### PCD: Init
1.1 Derive the pass key $K_\pi \leftarrow \text{KDF(MRZ)}$ or $K_\pi \leftarrow \text{KDF(CAN)}$.
1.2 Manage Security Environment: choose algorithms and password (MRZ or CAN).

### IC: Send nonce
2.1 Static domain pars $D_{IC}$.
2.2 Derive $K_\pi$ from MRZ or CAN.
2.3 Pick $s \leftarrow \{0,1\}^\ell$.
2.4 $z \leftarrow E(K_\pi, s)$.
2.5 Send challenge.

### PCD: Start generic mapping
3.1 $s \leftarrow D(K_\pi, z)$.
3.2 $k_{Map.PCD} \leftarrow \mathbb{Z}_q$.
3.3 $K_{Map.PCD} \leftarrow k_{Map.PCD}P$.

### IC: Generic Mapping
4.1 $K_{Map.IC} \leftarrow \mathbb{Z}_q$.
4.2 $K_{Map.IC} \leftarrow k_{Map.IC}P$.
4.3 $H \leftarrow k_{Map.IC}K_{Map.PCD}$.

### PCD: Finish generic mapping
5.1 $H \leftarrow k_{Map.PCD}K_{Map.IC}$.
5.2 $H \leftarrow k_{DH.PCD}K_{DH.IC}$.

### PCD: Start DH
7.1 $k_{DH.PCD} \leftarrow \mathbb{Z}_q$.
7.2 $K_{DH.PCD} \leftarrow k_{DH.PCD}P$.

### IC: Respond DH
8.1 $k_{DH.IC} \leftarrow \mathbb{Z}_q$.
8.2 $K_{DH.IC} \leftarrow k_{DH.IC}P$.
8.3 $K \leftarrow k_{DH.IC}K_{DH.PCD}$.

### PCD: Finish DH
9.1 $K \leftarrow k_{DH.PCD}K_{DH.IC}$.
9.2 $K_{MAC} \leftarrow K_{MAC}(K)$.

### PCD: Authenticate
11.1 $T_{PCD} \leftarrow \text{MAC}(K_{MAC}, K_{IC})$.

### IC: Authenticate & CA
12.1 Check $T_{PCD}$.
12.2 $T_{IC} \leftarrow \text{MAC}(K_{MAC}, K_{PCD})$.
12.3 $A_{IC} \leftarrow k^{-1}_{CA.IC}k_{Map.IC}$.

### PCD: Verify & CA
13.1 Check $T_{IC}$.
13.2 Decrypt $A_{IC}$ and check $K_{Map.IC} = CA_{IC}K_{CA.IC}$.
Security mechanisms:
Gain access and establish secure messaging

PACE: EC, integrated

PCD: Init
1.1 Derive the pass key
   $K_\pi \leftarrow \text{KDF}(\text{MRZ})$ or
   $K_\pi \leftarrow \text{KDF}(\text{CAN})$.
1.2 Manage Security
   Environment: choose
   algorithms and password
   (MRZ or CAN).

IC: Send nonce  
2.1 Static domain pars $D_{IC}$.
2.2 Derive $K_\pi$ from MRZ or
   CAN.
2.3 Pick $s \leftarrow \{0,1\}^\ell$.
2.4 $z \leftarrow E(K_\pi, s)$.
2.5 Send challenge.

PCD: Start integrated
   mapping
3.1 $s \leftarrow D(K_\pi, z)$.
3.2 $t \leftarrow \{0,1\}^k$.

Both: Update domain.
6.1 $D \leftarrow \text{Map}(D_{IC}, s, t)$,
   namely $P \leftarrow f_P(R_p(s, t))$
   where $R_p: \{0,1\}^\ell \times \{0,1\}^k \rightarrow Z_p$
   is pseudo-random and $f_P: Z_p \rightarrow \langle P \rangle$.

PCD: Start DH
7.1 $k_{DH, PCD} \leftarrow \hat{\text{KDF}}(Z_q^\times)$.
7.2 $K_{DH, PCD} \leftarrow k_{DH, PCD} P$.

IC: Respond DH
8.1 $k_{DH, IC} \leftarrow \hat{\text{KDF}}(Z_q^\times)$.
8.2 $K_{DH, IC} \leftarrow k_{DH, IC} \hat{P}$.
8.3 $K \leftarrow k_{DH, IC} K_{DH, PCD}$.

PCD: Finish DH
9.1 $K \leftarrow k_{DH, PCD} K_{DH, IC}$.

Both: Derive keys
10.1 $K_{\text{Enc}} \leftarrow \text{KDF}_{\text{Enc}}(K)$.
10.2 $K_{\text{MAC}} \leftarrow \text{KDF}_{\text{MAC}}(K)$.

PCD: Authenticate
11.1 $T_{PCD} \leftarrow \text{MAC}(K_{\text{Mac}}, K_{IC})$.

IC: Authenticate
12.1 Check $T_{PCD}$.
12.2 $T_{IC} \leftarrow \text{MAC}(K_{\text{MAC}}, K_{PCD})$.

PCD: Verify
13.1 Check $T_{IC}$.
## Security mechanisms:
Gain access and establish secure messaging

### PACE: Algorithms

<table>
<thead>
<tr>
<th>OID</th>
<th>Mapping</th>
<th>Sym. Cipher</th>
<th>Keylength</th>
<th>Secure Messaging</th>
<th>Auth. Token</th>
</tr>
</thead>
<tbody>
<tr>
<td>id-PACE-DH-GM-3DES-CBC-CBC</td>
<td>Generic</td>
<td>3DES</td>
<td>112</td>
<td>CBC / CBC</td>
<td>CBC</td>
</tr>
<tr>
<td>id-PACE-DH-GM-AES-CBC-CMAC-128</td>
<td>Generic</td>
<td>AES</td>
<td>128</td>
<td>CBC / CMAC</td>
<td>CMAC</td>
</tr>
<tr>
<td>id-PACE-DH-GM-AES-CBC-CMAC-192</td>
<td>Generic</td>
<td>AES</td>
<td>192</td>
<td>CBC / CMAC</td>
<td>CMAC</td>
</tr>
<tr>
<td>id-PACE-DH-GM-AES-CBC-CMAC-256</td>
<td>Generic</td>
<td>AES</td>
<td>256</td>
<td>CBC / CMAC</td>
<td>CMAC</td>
</tr>
<tr>
<td>id-PACE-DH-IM-3DES-CBC-CBC</td>
<td>Integrated</td>
<td>3DES</td>
<td>112</td>
<td>CBC / CBC</td>
<td>CBC</td>
</tr>
<tr>
<td>id-PACE-DH-IM-AES-CBC-CMAC-128</td>
<td>Integrated</td>
<td>AES</td>
<td>128</td>
<td>CBC / CMAC</td>
<td>CMAC</td>
</tr>
<tr>
<td>id-PACE-DH-IM-AES-CBC-CMAC-192</td>
<td>Integrated</td>
<td>AES</td>
<td>192</td>
<td>CBC / CMAC</td>
<td>CMAC</td>
</tr>
<tr>
<td>id-PACE-DH-IM-AES-CBC-CMAC-256</td>
<td>Integrated</td>
<td>AES</td>
<td>256</td>
<td>CBC / CMAC</td>
<td>CMAC</td>
</tr>
<tr>
<td>id-PACE-ECDH-GM-3DES-CBC-CBC</td>
<td>Generic</td>
<td>3DES</td>
<td>112</td>
<td>CBC / CBC</td>
<td>CBC</td>
</tr>
<tr>
<td>id-PACE-ECDH-GM-AES-CBC-CMAC-128</td>
<td>Generic</td>
<td>AES</td>
<td>128</td>
<td>CBC / CMAC</td>
<td>CMAC</td>
</tr>
<tr>
<td>id-PACE-ECDH-GM-AES-CBC-CMAC-192</td>
<td>Generic</td>
<td>AES</td>
<td>192</td>
<td>CBC / CMAC</td>
<td>CMAC</td>
</tr>
<tr>
<td>id-PACE-ECDH-GM-AES-CBC-CMAC-256</td>
<td>Generic</td>
<td>AES</td>
<td>256</td>
<td>CBC / CMAC</td>
<td>CMAC</td>
</tr>
<tr>
<td>id-PACE-ECDH-IM-3DES-CBC-CBC</td>
<td>Integrated</td>
<td>3DES</td>
<td>112</td>
<td>CBC / CBC</td>
<td>CBC</td>
</tr>
<tr>
<td>id-PACE-ECDH-IM-AES-CBC-CMAC-128</td>
<td>Integrated</td>
<td>AES</td>
<td>128</td>
<td>CBC / CMAC</td>
<td>CMAC</td>
</tr>
<tr>
<td>id-PACE-ECDH-IM-AES-CBC-CMAC-192</td>
<td>Integrated</td>
<td>AES</td>
<td>192</td>
<td>CBC / CMAC</td>
<td>CMAC</td>
</tr>
<tr>
<td>id-PACE-ECDH-IM-AES-CBC-CMAC-256</td>
<td>Integrated</td>
<td>AES</td>
<td>256</td>
<td>CBC / CMAC</td>
<td>CMAC</td>
</tr>
<tr>
<td>id-PACE-ECDH-CAM-AES-CBC-CMAC-128</td>
<td>Chip Authentication</td>
<td>AES</td>
<td>128</td>
<td>CBC / CMAC</td>
<td>CMAC</td>
</tr>
<tr>
<td>id-PACE-ECDH-CAM-AES-CBC-CMAC-192</td>
<td>Chip Authentication</td>
<td>AES</td>
<td>192</td>
<td>CBC / CMAC</td>
<td>CMAC</td>
</tr>
<tr>
<td>id-PACE-ECDH-CAM-AES-CBC-CMAC-256</td>
<td>Chip Authentication</td>
<td>AES</td>
<td>256</td>
<td>CBC / CMAC</td>
<td>CMAC</td>
</tr>
</tbody>
</table>
Security mechanisms:
Authenticating

PA

\[ \text{IC} \leftarrow \text{SO}_D \rightarrow \text{IFD} \]

IFD: Read $\text{SO}_D$

IC: Send $\text{SO}_D$

IFD: Detailed verification.
3.1 Check certificate and signature of $\text{SO}_D$ including validity and consistency:
   - Verify the chain of the document signer certificate $C_{DS}$, check against the trusted master list and CRLs.
   - Verify the signature on $\text{SO}_D$.
   - Best practice:
     - Check that certificate rights are correct.
     - Check DG1 and compare to visual MRZ.
     - Check consistency of country codes in the certificate chain, DG1 and visual MRZ.
   - Optionally, read data groups and ensure that its hash is in $\text{SO}_D$.
   - Verify that the issuing date is in the usage period of the certificate.

+ Data authentic and unchanged.
− No chip secret involved $\implies$ Cloning not prevented.
Security mechanisms:
Authenticating

AA

\[
\text{IFD: Internal authentication}
\]

1.1 \( \text{RND.IFD} \leftarrow \{0, 1\}^{64} \)

\[
\text{IC: Internal authenticate}
\]

2.1 Note \( \text{RND.IC} \in \{0, 1\}^{c-4} \).
2.2 Generate message \( M = \text{RND.IC} | \text{RND.IFD} \).
2.3 Sign (with message recovery) the hash of \( M \) with the 
active authentication private key: \( \sigma \).

\[
\text{IFD}
\]

3.1 Obtain the chip’s active authentication public key from 
\( \text{DG15} \).
3.2 Check the signature...

Together with PA: Cloning prevented.

Transferable (device-in-the-middle).

Challenge semantics.

\[
\begin{align*}
\text{Attacker obtains signature on any challenge from IC.} \\
\text{Terminal could use with message recovery: } c &= \text{Sign}(k_{\text{PCD}}, \text{ID}_{\text{IC}} | \text{Date} | \text{Time} | \text{Location}). \\
\text{For internal use, eg. as proof for immigration.} \\
\text{Powerful tracking that cannot be faked due to the signatures.}
\end{align*}
\]
Security mechanisms:
Authenticating

CA

IC:  
2.1 Send static Diffie-Hellman public key $K_{IC}$, namely DG14, and domain parameters $D_{IC}$.

PCD:  
3.1 Generate ephemeral Diffie-Hellman key pair $k_{DH,PCD}$, $K_{DH,PCD}$.
3.2 $K = k_{DH,PCD}K_{DH,IC}$.

IC:  
4.1 $K = k_{IC}K_{DH,PCD}$.

Both:  
5.1 Establish secure messaging:
$K_{Enc} \leftarrow KDF_{Enc}(K)$, $K_{MAC} \leftarrow KDF_{MAC}(K)$.

Together with PA: Cloning prevented.
Non-transferable.
No challenge semantics.
Strong session keys.
Security mechanisms:
Authenticating

PACE with Chip Authentication Mapping

**PCD: Init**
1.1 Derive the pass key $K_\pi \leftarrow \text{KDF(MRZ)}$ or $K_\pi \leftarrow \text{KDF(CAN)}$.
1.2 Manage Security Environment: choose algorithms and password (MRZ or CAN).

**IC: Send nonce**
2.1 Static domain pars $D_{IC}$.
2.2 Derive $K_\pi$ from MRZ or CAN.
2.3 Pick $s \leftarrow \{0,1\}^\ell$.
2.4 $z \leftarrow E(K_\pi, s)$.
2.5 Send challenge.

**PCD: Start generic mapping**
3.1 $s \leftarrow D(K_\pi, z)$.
3.2 $k_{\text{Map.PCD}} \leftarrow \pi Z_q^\pi$.
3.3 $K_{\text{Map.PCD}} \leftarrow k_{\text{Map.PCD}} P$.

**IC: Generic Mapping**
4.1 $k_{\text{Map.IC}} \leftarrow \pi Z_q^\pi$.
4.2 $K_{\text{Map.IC}} \leftarrow k_{\text{Map.IC}} P$.
4.3 $H \leftarrow k_{\text{Map.IC}} K_{\text{Map.PCD}}$.

**PCD: Finish generic mapping**
5.1 $H \leftarrow k_{\text{Map.PCD}} K_{\text{Map.IC}}$.
Both: Update domain
6.1 $D \leftarrow \text{Map}(D_{IC}, s, H)$, namely $\hat{P} \leftarrow sP + H$.

**PCD: Start DH**
7.1 $k_{\text{DH.PCD}} \leftarrow \pi Z_q^\pi$.
7.2 $K_{\text{DH.PCD}} \leftarrow k_{\text{DH.PCD}} P$.

**IC: Respond DH**
8.1 $k_{\text{DH.IC}} \leftarrow \pi Z_q^\pi$.
8.2 $K_{\text{DH.IC}} \leftarrow k_{\text{DH.IC}} \hat{P}$.
8.3 $K \leftarrow k_{\text{DH.IC}} K_{\text{DH.PCD}}$.

**PCD: Finish DH**
9.1 $K \leftarrow k_{\text{DH.PCD}} K_{\text{DH.IC}}$.
Both: Derive keys
10.1 $K_{\text{Enc}} \leftarrow \text{KDF}_{\text{Enc}}(K)$.
10.2 $K_{\text{MAC}} \leftarrow \text{KDF}_{\text{MAC}}(K)$.

**PCD: Authenticate**
11.1 $T_{\text{PCD}} \leftarrow \text{MAC}(K_{\text{MAC}}, K_{\text{IC}})$.

**IC: Authenticate & CA**
12.1 Check $T_{\text{IC}}$.
12.2 $T_{\text{IC}} \leftarrow \text{MAC}(K_{\text{MAC}}, K_{\text{PCD}})$.
12.3 $A_{\text{IC}} \leftarrow k_{\text{CA.IC}}^{-1} k_{\text{Map.IC}}$, $A_{\text{IC}} = E(K_{\text{Enc}}, A_{\text{IC}})$.

**PCD: Verify & CA**
13.1 Check $T_{\text{IC}}$.
13.2 Decrypt $A_{\text{IC}}$ and check $K_{\text{Map.IC}} = A_{\text{IC}} K_{\text{CA.IC}}$. 

\(2017-07-20\) 222+41
Security mechanisms:

Secure Messaging

- Established by BAC, PACE or CA.
- Based on 3DES or AES in EtA mode (encrypt-then-authenticate) with SSC (send sequence counter).
  - **3DES.**
    - **Encryption:** Two key 3DES in CBC mode with zero IV, zero padding.
    - **Message Authentication:** MAC algorithm 3 (Retail MAC, ISO/IEC 9797) with DES, zero IV and padding method 2. Tag length: 64 bits.
  - **SSC:** 64 bits, initialized either by BAC with 32 least significant bits of RND.IC and RND.IFD or to zero.

- **AES.**
  - **Encryption:** AES in CBC mode with \( IV \leftarrow E(K_{Enc}, SSC) \).
  - **Message Authentication:** AES in CMAC-mode. Tag length: 64 bits.
  - **SSC:** 128 bits, initialized to zero.

- Terminated on any error or plain message: IC must delete session keys and terminal’s access rights.
Introduction

MRTDs (Machine Readable Travel Documents)

eMRTDs

Security mechanisms

eMRTD PKI
  Roles and responsibilities
  Key management
  Distribution mechanisms

ePassport application (BSI summary and extension)

eIDAS application (BSI TR03110)

Security?

German ID card and residence permit (BSI TR03127)

European projects, regulations, applications
eMRTD PKI:
Roles and responsibilities

- CSCA (country signing certification authority).
  - Exactly one per state.
  - Issues certificates for document signers and optionally mast list signers and deviation list signers.
  - Issues periodic CRLs (certificate revocation lists).
  - CSCA key pairs should be generated and stored in a *highly protected, off-line CA infrastructure.*
  - MUST be forwarded to the PKD (public key directory).

- DS (document signer).
  - Signs eMRTD data. \( \rightsquigarrow \text{SO}_D \).
  - DS key pairs should be generated and stored in a *highly protected infrastructure.*
  - SHOULD be forwarded to the PKD (public key directory).

- IS (inspection system).
  - PA, i.e. verifies signatures including certification path and CRLs.

- MLS (master list signer).
  - Signs a list of CSCA certificates under bilateral distribution.

- DLS (deviation list signers).
  - Issues periodically and signs a list of deviations of issued MRTDs from Doc 9303.
eMRTD PKI:
Key management

- Usage and Validity.

<table>
<thead>
<tr>
<th></th>
<th>Use of private key</th>
<th>Public key validity</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSCA</td>
<td>3-5 years</td>
<td>13-15 years</td>
</tr>
<tr>
<td>DS</td>
<td>≤3 months</td>
<td>10 years 3 months</td>
</tr>
<tr>
<td>IC</td>
<td></td>
<td>5-10 years</td>
</tr>
</tbody>
</table>

...assuming a MRTD validity period of 10 years.

- Re-key.
  - MUST be notified to receiving states 90 days in advance of the key rollover.
  - If new: out-of-band authentication. (Eg. in a diplomatic courier.) Otherwise: CSCA link certificates link old and new key.

- Revocation.
  - At least one CRL every 90 days, but no more frequently than every 48 hours.
  - If a certificate is revoked, a new CRL MUST be distributed within 48 hours.

- Algorithms.
  - RSASSA-PSS (RFC 4055).
  - RSASSA-PKCS1_v15; not recommended, but must be implemented.
  - DSA (FIPS 186-4).
  - ECDSA (BSI TR03111).
  - SHA-224, SHA-256, SHA-384, SHA-512 (FIPS 180-2).
eMRTD PKI:
Distribution mechanisms

- Master list.
  - is a digitally signed list of the CSCA certificates that are trusted by the receiving state and MUST include the own CSCA certificate and the MLS certificate.
- PKD (public key directory).
  - contains
    - DS certificates and CRLs.
    - CSCA master lists (including CSCA and MSL certificate).
- Writeable only for participants.
  Readable for participants and non-participants.

- Bilateral exchange.
  - Diplomatic courier.
  - Email exchange.
  - Download from a website of the issuing CSCA.
  - Download from an LDAP server of the issuing CSCA.
- Deviation lists.
- eMRTD contactless IC.
Section Overview

Introduction

MRTDs (Machine Readable Travel Documents)

eMRTDs

Security mechanisms

eMRTD PKI

ePassport application (BSI summary and extension)
  - Tasks
  - Inspection
  - More subprotocols
    - TA version 1
  - Country verification PKI

eIDAS application (BSI TR03110)

Security?

German ID card and residence permit (BSI TR03127)

European projects, regulations, applications
ePassport application (BSI summary and extension):

Tasks

- Passive authentication.
  
  \[\ldots\] uses a digital signature to authenticate data stored in the data groups on the MRTD chip.

- Read \( \text{SO}_D \).
- Retrieve corresponding \( C_{DS} \), trusted \( C_{CSCA} \) and corresponding CRL.
- Verify all that.
- Compute hashes of read data groups and compare to \( \text{SO}_D \).

- Active authentication.
  
  \[\ldots\] is a digital security feature that prevents cloning by introducing a chip-individual key pair [enabling that] the chip can prove knowledge of this private key in a challenge-response protocol[\ldots].

- Read public key from DG15.
- Corresponding private key stored in secure memory on the chip and only used internally.

- Access control.
  
  \[\ldots\] is not only required for privacy reasons but also mitigates the risk of cloning attacks.

- Less-sensitive data, eg. MRZ and facial image, is protected by BAC.
- BAC has limited strength.
- PACE provides strong session keys independent of the password’s entropy.
- Sensitive data, eg. fingerprints, must only be available to authorized terminals. Protected by Extended Access Control.
Chip Access Procedure

- Read CardAccess (symmetric ciphers, key agreement algorithms, domain parameters, mappings). If missing try BAC.
- PACE with MRZ or CAN.
  If successful the chip starts and requires secure messaging, grants access to less-sensitive data, at least DG1, DG2, DG14, DG15, SO$_D$.
- Select ePassport application.
- BAC unless PACE application has been used.

Standard ePassport Inspection Procedure

Execute the Chip Access Procedure and

- PA: Read SO$_D$, verify at least DG14.
- AA (optional): Verify DG15 and perform AA.
- Read and authenticate data.

Advanced ePassport Inspection Procedure

Execute the Chip Access Procedure and

- CA: Read DG14 and perform CA unless PACE-CAM was used.
- PA: Read SO$_D$, verify at least DG14, check CardAccess against DG14.
- AA (optional): Verify DG15 and perform AA.
- TA (terminal authentication version 1).
  If successful the chip additionally grants access to data groups acc.to the terminal’s access rights and requires secure messaging with the keys from CA authenticated via TA.
- Read and authenticate data.
ePassport application (BSI summary and extension): More subprotocols

TA version 1

Both: Prepare ID and C.
1.1 If BAC used then $\text{ID}_{IC} \leftarrow \text{MRZ Document Number}$.
   If PACE used then $\text{ID}_{IC} = \text{Comp}(K_{\text{DH,IC}})$ (dynamic binding).
   Some states have issued MRTDs with static binding where $\text{ID}_{IC} \leftarrow \text{MRZ Document Number}$ if MRZ is used as password for PACE or $\text{ID}_{IC} \leftarrow \text{CAN}$ if CAN is used as password for PACE.

1.2 $C \leftarrow \text{Comp}(K_{\text{DH,PCD}})$ from CA or PACE-CAM or PACE.

PCD: Announce.
2.1 Send the certificate chain starting with a CVCA certificate trusted by the chip and ending with the terminal certificate.

IC: Challenge.
3.1 Verify the certificate chain and extract the terminal’s public key $K_{\text{PCD}}$.
3.2 Choose a challenge $r_{IC} \leftarrow \{0, 1\}^s$.

PCD: Response.
4.1 $s_{PCD} \leftarrow \text{Sign}(k_{PCD}, \text{ID}_{IC} | r_{IC} | C)$.

IC: Verify.
5.1 Verify $s_{PCD}$.

- MUST be preceded by CA or PACE-CAM.
- IC grants access acc.to the terminal’s certificate restricted to secure messaging with the authenticated ephemeral public key $K_{\text{DH,PCD}}$.
- IC needs to store one or more CVCA certificates as trust anchor.
- Does not modify session keys.
- IC knows that terminal has been verifiably granted extra rights.
- Need a verification PKI for verification of terminals.
ePassport application (BSI summary and extension):
Country verification PKI

- Each CVCA certifies DV with certain access rights and validity periods.
- Each DV certifies terminals with restricted access rights and validity periods.
- Terminal certificates MUST have a short validity period, possibly depending on the associated DV.
Fact

*The chip must be able to verify these verification certificates.*

- Certificate format.
- Signature algorithm, domain parameters and key sizes are determined by the CVCA of the issuing state. In particular, the entire tree MUST use the same ones.
- CVCA link certificates can be used to switch to new parameters.
- The terminal needs a certificate tracing back to (one of) the CVCA certificate(s) stored in the chip!
  - The chip MUST update its trust point(s) acc.to received valid link certificates.
- If the IC has no internal clock, the current date is approximated.
  - Namely, the current date is at least the most recent effective date in a valid received CVCA link certificate, DV certificate or terminal certificate.

**Note:** The IC only verifies that a certificate is *apparently* recent, ie. with respect to the approximated current date.
Introduction

MRTDs (Machine Readable Travel Documents)

eMRTDs

Security mechanisms

eMRTD PKI

ePassport application (BSI summary and extension)

**eIDAS application (BSI TR03110)**
- eID data
- Again more subprotocols
- Extended General Authentication Procedure
- TA version 2
- CA version 2
- RI
- Sector specific revocation of an IC
- Pseudonymous signatures
- PS
- CA version 3

Security?

German ID card and residence permit (BSI TR03127)

European projects, regulations, applications
eIDAS (electronic IDentification, Authentication and trust Services)

- eID: electronic identification.
- eSIGN: electronic signatures.
- Online authentication.
<table>
<thead>
<tr>
<th>DG</th>
<th>Content</th>
<th>FID</th>
<th>SFID</th>
<th>ASN.1 type</th>
<th>R/W</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>DG1</td>
<td>Document Type</td>
<td>0x0101</td>
<td>0x01</td>
<td>DocumentType</td>
<td>R</td>
<td>PACE + TA + CA</td>
</tr>
<tr>
<td>DG2</td>
<td>Issuing State, Region and Municipality</td>
<td>0x0102</td>
<td>0x02</td>
<td>IssuingEntity</td>
<td>R</td>
<td>PACE + TA + CA</td>
</tr>
<tr>
<td>DG3</td>
<td>Date of Expiry</td>
<td>0x0103</td>
<td>0x03</td>
<td>Date</td>
<td>R</td>
<td>PACE + TA + CA</td>
</tr>
<tr>
<td>DG4</td>
<td>Given Names</td>
<td>0x0104</td>
<td>0x04</td>
<td>GivenNames</td>
<td>R</td>
<td>PACE + TA + CA</td>
</tr>
<tr>
<td>DG5</td>
<td>Family Names</td>
<td>0x0105</td>
<td>0x05</td>
<td>FamilyNames</td>
<td>R</td>
<td>PACE + TA + CA</td>
</tr>
<tr>
<td>DG6</td>
<td>Nom de Plume</td>
<td>0x0106</td>
<td>0x06</td>
<td>NomDePlume</td>
<td>R</td>
<td>PACE + TA + CA</td>
</tr>
<tr>
<td>DG7</td>
<td>Academic Title</td>
<td>0x0107</td>
<td>0x07</td>
<td>AcademicTitle</td>
<td>R</td>
<td>PACE + TA + CA</td>
</tr>
<tr>
<td>DG8</td>
<td>Date of Birth</td>
<td>0x0108</td>
<td>0x08</td>
<td>DateOfBirth</td>
<td>R</td>
<td>PACE + TA + CA</td>
</tr>
<tr>
<td>DG9</td>
<td>Place of Birth</td>
<td>0x0109</td>
<td>0x09</td>
<td>PlaceOfBirth</td>
<td>R</td>
<td>PACE + TA + CA</td>
</tr>
<tr>
<td>DG10</td>
<td>Nationality</td>
<td>0x010A</td>
<td>0x0A</td>
<td>Nationality</td>
<td>R</td>
<td>PACE + TA + CA</td>
</tr>
<tr>
<td>DG11</td>
<td>Sex</td>
<td>0x010B</td>
<td>0x0B</td>
<td>Sex</td>
<td>R</td>
<td>PACE + TA + CA</td>
</tr>
<tr>
<td>DG12</td>
<td>Optional Data</td>
<td>0x010C</td>
<td>0x0C</td>
<td>OptionalData</td>
<td>R</td>
<td>PACE + TA + CA</td>
</tr>
<tr>
<td>DG13</td>
<td>Birth Name</td>
<td>0x010D</td>
<td>0x0D</td>
<td>BirthName</td>
<td>R</td>
<td>PACE + TA + CA</td>
</tr>
<tr>
<td>DG14</td>
<td>Written Signature</td>
<td>0x010E</td>
<td>0x0E</td>
<td>WrittenSignature</td>
<td>R</td>
<td>PACE + TA + CA</td>
</tr>
<tr>
<td>DG15</td>
<td>Date of Issue</td>
<td>0x010F</td>
<td>0x0F</td>
<td>Date</td>
<td>R</td>
<td>PACE + TA + CA</td>
</tr>
<tr>
<td>DG16</td>
<td>--</td>
<td>0x0110</td>
<td>0x10</td>
<td>RFU</td>
<td>R</td>
<td>PACE + TA + CA</td>
</tr>
<tr>
<td>DG17</td>
<td>Normal Place of Residence (multiple)</td>
<td>0x0111</td>
<td>0x11</td>
<td>PlaceOfResidence</td>
<td>R/W</td>
<td>PACE + TA + CA</td>
</tr>
<tr>
<td>DG18</td>
<td>Municipality ID</td>
<td>0x0112</td>
<td>0x12</td>
<td>MunicipalityID</td>
<td>R/W</td>
<td>PACE + TA + CA</td>
</tr>
<tr>
<td>DG19</td>
<td>Residence Permit I</td>
<td>0x0113</td>
<td>0x13</td>
<td>ResidencePermitI</td>
<td>R/W</td>
<td>PACE + TA + CA</td>
</tr>
<tr>
<td>DG20</td>
<td>Residence Permit II</td>
<td>0x0114</td>
<td>0x14</td>
<td>ResidencePermitII</td>
<td>R/W</td>
<td>PACE + TA + CA</td>
</tr>
<tr>
<td>DG21</td>
<td>Phone Number</td>
<td>0x0115</td>
<td>0x15</td>
<td>PhoneNumber</td>
<td>R/W</td>
<td>PACE + TA + CA</td>
</tr>
<tr>
<td>DG22</td>
<td>Email Address</td>
<td>0x0116</td>
<td>0x16</td>
<td>EmailAddress</td>
<td>R/W</td>
<td>PACE + TA + CA</td>
</tr>
</tbody>
</table>

Table 3: Data Groups of the eID Application
eIDAS application (BSI TR03110):
Again more subprotocols

- Password Authenticated Connection Establishment (PACE).
  - Used before CA, rather than afterwards.
  - Generates ephemeral DH key share for use in later CA2 or CA3.
- Chip Authentication Version 2 (CA2).
  - To provide pseudonymity, the same key must be used for a sufficiently large group of eIDAS tokens.
- Chip Authentication Version 3 (CA3).
  - ...uses a pseudonymous signature combined with RI, instead.
- Restricted Identification (RI).
- Pseudonymous Signatures (PS).
Based on these protocols the following procedures are defined:

- General Authentication Procedure (GAP).
- Enhanced Role Authentication (ERA).
- PIN Management.
eIDAS application (BSI TR03110):
Extended General Authentication Procedure
Chip Access Procedure

- Read CardAccess (symmetric ciphers, key agreement algorithms, domain parameters, mappings).
- PACE.
  If successful the chip starts and requires secure messaging, grants access to less-sensitive data.
- Select eIDAS application.
- 

GAP

Execute the Chip Access Procedure and

\[
\text{EAC} \left\{ \begin{array}{c}
\text{TA2 with eID server.} \\
\text{PA: Read SO}_D, \text{ verify at least DG14.} \\
\text{CA2 or CA3: Read DG14 and perform CA with terminal key pair generated in TA2.} \\
\text{Read data, perform special functions.}
\end{array} \right.
\]

ERA (optional)

- Store attribute request.
  - EAC with attribute provider.
  - Retrieve attribute request.
  - Read data, perform special functions.
  - Store attribute.
  - Switch back.
- Read attribute.
eIDAS application (BSI TR03110):
TA version 2

Both: Prepare ID.
1.1 \( \text{ID}_{IC} \leftarrow \text{Comp}(K_{DH,IC}) \).

PCD: Announce.
2.1 Generate ephemeral Diffie-Hellman key pair \( k_{DH,PCD}, K_{DH,PCD} \) for use in subsequent(!) CA.
2.2 \( C \leftarrow \text{Comp}(K_{DH,PCD}) \).
2.3 Send the certificate chain starting with the CVCA certificate stored on the chip and ending with the terminal certificate.

IC: Challenge.
3.1 Verify the certificate chain and extract the terminal’s public key \( K_{PCD} \).
3.2 Choose a challenge \( r_{IC} \leftarrow \{0,1\}^2 \).

PCD: Response.
4.1 \( s_{PCD} \leftarrow \text{Sign}(k_{PCD}, \text{ID}_{IC} | r_{IC} | C | [A_{PCD}]) \).

IC: Verify.
5.1 Verify \( s_{PCD} \).

TA2 is identical to TA with exception of:
- Additional data \( A_{PCD} \) can be added in the signed text.
- The ephemeral PCD key pair for CA is chosen and required to be used in a following CA.
eIDAS application (BSI TR03110): CA version 2

IC:
2.1 Send static Diffie-Hellman public key $K_{IC}^{group}$ and domain parameters $D_{IC}$.

PCD:
3.1 If not done earlier, generate ephemeral Diffie-Hellman key pair $k_{DH,PCD}^{K_{DH,PCD}}$.
3.2 $K \leftarrow k_{DH,PCD}^{K_{DH,PCD}} K_{IC}^{group}$.

IC:
4.1 $K \leftarrow k_{IC}^{K_{DH,PCD}}$.
4.2 Choose $r_{IC} \leftarrow \mathbb{Z}_2$.
4.3 $K_{Enc} \leftarrow \text{KDF}_{Enc}(K, r_{IC})$, $K_{MAC} \leftarrow \text{KDF}_{MAC}(K, r_{IC})$.
4.4 Compute $T_{IC} \leftarrow \text{MAC}(K_{MAC}, K_{DH,PCD})$.

PCD:
5.1 Establish secure messaging:
    $K_{Enc} \leftarrow \text{KDF}_{Enc}(K, r_{IC})$, $K_{MAC} \leftarrow \text{KDF}_{MAC}(K, r_{IC})$.
5.2 Verify $T_{IC}$.

- For *pseudonymity* the static key pair must not be personalized. So it must be used by a large group of devices.
- Still, the chip must prove itself.
- Open: how to revoke a ‘chip’ now?
eIDAS application (BSI TR03110):

\[ K_S, D \]

\[ I_{ID}^S \]

IC:  
2.1 Verify \( K_S \).  
2.2 Compute the sector-specific identifier: \( I_{ID}^S \leftarrow H(k_{ID} K_S) \).

- The terminal must be able to verify chip authenticity.  
- Mind that any preceding communication MUST not reveal identity information.  
- Revocation?
eIDAS application (BSI TR03110): Sector specific revocation of an IC

Revocation request

\[ K_{ID} \]

\[ d = k_{revoc} K_{ID} \]

\[ s_j = H(k_{Sj} d) \]

\[ s_j = H(k_{Sj} k_{revoc} K_{ID}) \]

Sector 1

\[ C_{DV_1}(T_1, DV_1, \ldots, K_{S1}) \]

\[ I^{S2}_{ID} = H(k_{ID} K_{S2}) \]

\[ K_{ID} = k_{ID}^P \]

Sector 2

\[ C_{DV_2}(T_2, DV_1, \ldots, K_{S2}) \]

Sector 3

\[ C_{DV_3}(T_3, DV_2, \ldots, K_{S3}) \]

\[ I^{S3}_{ID} = H(k_{ID} K_{S3}) \]
eIDAS application (BSI TR03110):
Pseudonymous signatures

DomainKeyGen

Input: $1^k$.
Output: Domain parameters $D_M$, a domain key pair $k_M$, $K_M$ and a group key pair $k_{\text{group}}$, $K_{\text{group}}$.

- Choose domain parameters $D_M$ specifying a group, a generator $P$ and its order $q$.
- Choose $k_M \leftarrow \mathbb{Z}_q$, $K_M \leftarrow k_MP$ within $D_M$.
- Choose $k_{\text{group}} \leftarrow \mathbb{Z}_q$, $K_{\text{group}} \leftarrow k_{\text{group}}P$ within $D_M$.

RecovationKeyGen

Input: Domain parameters $D_M$ and a public key $K_M$.
Output: A private key $k_{\text{revo}}$ and a public key $K_{\text{revo}}$.

- Choose $k_{\text{revo}} \leftarrow \mathbb{Z}_q$, $K_{\text{revo}} \leftarrow k_{\text{revo}}P$ within $D_M$.

PSign

Input: $(D_M, K_M)$, $K_S$, $(k_{1IC,1}, k_{1IC,2})$, $I_{1IC,1}^S$, $I_{1IC,2}^S$, $lD_{DSI}$, $M$.
Output: $ps_{1IC} = (c, s_1, s_2)$.

- Assert $I_{1IC,1}^S = k_{1IC,1}K_S$, $I_{1IC,2}^S = k_{1IC,2}K_S$.
- Choose $k_1, k_2 \leftarrow \mathbb{Z}_q$.
- Compute $Q_1 \leftarrow k_1P + k_2K_M$.
- Compute $A_1 \leftarrow k_1K_S$, $A_2 \leftarrow k_2K_S$.
- Compute a witness (challenge)
  $$c \leftarrow H(Q_1, I_{1IC,1}^S, A_1, I_{1IC,2}^S, A_2, K_S, lD_{DSI}, M).$$
- Compute $s_1 \leftarrow k_1 - ck_{1IC,1}$, $s_2 \leftarrow k_2 - ck_{1IC,2}$.
- Return $(c, s_1, s_2)$.

SectorKeyGen

Input: Domain parameters $D_M$ and a public key $K_M$, the revocation public key $K_{\text{revo}}$, a sector id $S$.
Output: A private key $k_S$ and a public key $K_S$.

- Choose $k_S \leftarrow \mathbb{Z}_q$, $K_S \leftarrow k_SK_{\text{revo}}$ within $D_M$.

KeyGen

Input: Domain parameters $D_M$ and a public key $K_M$, a domain private key $k_M$, a group key pair $k_{\text{group}}$, $K_{\text{group}}$.
Output: A private key $(k_{1IC,1}, k_{1IC,2})$ and a public key $K_{1IC}$ and a semi-public key $(K_{1IC,1}, K_{1IC,2})$ for revocation.

- Choose $k_{1IC,2} \leftarrow \mathbb{Z}_q$.
- Compute $k_{1IC,1}$ such that $k_{\text{group}} = k_{1IC,1} + k_{1IC,2}k_M$.
- Compute $K_{1IC,1} \leftarrow k_{1IC,1}P$, $K_{1IC,2} \leftarrow k_{1IC,2}P$.
- Put $K_{1IC} \leftarrow K_{\text{group}}$. Note that $K_{1IC} = k_{1IC,1}P + k_{1IC,2}K_M$.

PVerify

Input: $(D_M, K_M)$, $K_S$, $K_{1IC}$, $I_{1IC,1}^S$, $I_{1IC,2}^S$, $lD_{DSI}$, $M$,
output: $ps_{1IC} = (c, s_1, s_2)$.
Output: ACCEPT or REJECT.

- Compute $Q_1 \leftarrow cK_{1IC} + s_1P + s_2K_M$.
- Compute $A_1 \leftarrow cI_{1IC,1}^S + s_1K_S$, $A_2 \leftarrow cI_{1IC,2}^S + s_2K_S$.
- Recompute the witness (verification)
  $$v \leftarrow H(Q_1, I_{1IC,1}^S, A_1, I_{1IC,2}^S, A_2, K_S, lD_{DSI}, M).$$
- If $v = c$ then ACCEPT else REJECT.
eIDAS application (BSI TR03110):

**IC:**
1.1 static keys \( (k_1^I, k_2^I, K^I, (D_M, K_M)) \)

**PCD:**

\[ [M], K^S \]
\[ I_1^S, I_2^S, ps^I \]

\[ \leftarrow \]

IC:
1. Verify \( M \) using authenticated auxiliary data \( A_{PCD} \) sent during TA.
2. Verify \( K^S \).
3. Derive sector-specific pseudonyms:
   \( I_1^S \leftarrow k_1^I K^S \).

\[ \rightarrow \]

PCD:
4. Check
   \[ PSign((D_M, K_M), K^S, k_1^I, k_1^I, I_1^S, I_2^S, ID_{DSI}, M) = \text{ACCEPT} \]

\[ \text{I}_{IC,2} \leftarrow k_2^I K^S. \]

3.4 Compute the pseudonymous signature over \( (D_M, K_M) \) on \( M \) and the digital signature information \( ID_{DSI} \):
   \[ ps^I \leftarrow \]
   \[ \text{PSign}((D_M, K_M), K^S, k_1^I, k_1^I, I_1^S, I_2^S, ID_{DSI}, M). \]

\[ \text{PVerify}((D_M, K_M), K^S, k_1^I, I_1^S, I_2^S, ID_{DSI}, M, ps^I) = \text{ACCEPT}. \]

The protocol augments the Restricted Identification to allow for signatures under a sector-specific pseudonym.

- The eIDAS token authenticates the sector-specific identifier towards the terminal by means of the pseudonymous signature.
- The protocol allows for whitelisting of original eIDAS token, even in case of a group key compromise.
eIDAS application (BSI TR03110):
CA version 3

IC:  
1. Previously in TA2: Receive $D_{IC}$ and generate ephemeral Diffie-Hellman key pair $k_{DH,IC}$, $K_{DH,IC}$.

IC:  
2.1 Generate ephemeral Diffie-Hellman key pair $k_{DH,IC}$, $K_{DH,IC}$.
2.2 $K \leftarrow k_{DH,IC} K_{DH,IC}$.

PCD:  
3.1 $K \leftarrow k_{DH,IC} K_{DH,IC}$.

Both:  
4.1 Establish secure messaging with keys derived from $K$: $K_{SEnc} \leftarrow KDF_{Enc}(K)$, $K_{MAC} \leftarrow KDF_{MAC}(K)$.

PCD:  
6.1 Verify $PVerify((D_M, K_M), K_S, K_{IC}, I_{IC,1}^S, I_{IC,2}^S, ID_{DSI}, M, ps_{IC}) = ACCEPT$.

IC:  
5.1 Static keys: $(k_{IC,1}, k_{IC,2}, K_{DH,IC}, (D_M, K_M))$.
5.2 Compute $I_{IC,1}^{Sector} = k_{IC,1} K_{Sector}$.
5.3 Compute $I_{IC,2}^{Sector} = k_{IC,2} K_{Sector}$.
5.4 Compute $ps_{IC} \leftarrow PSign((D_M, K_M), K_S, k_{IC,1}, k_{IC,2}, I_{IC,1}^S, I_{IC,2}^S, ID_{DSI}, M)$.

This version grants pseudonymity and authenticity by the use of a pseudonymous signature.

Open: how to revoke a ‘chip’ now?
Section Overview

Introduction

MRTDs (Machine Readable Travel Documents)

eMRTDs

Security mechanisms

eMRTD PKI

ePassport application (BSI summary and extension)

eIDAS application (BSI TR03110)

Security?

German ID card and residence permit (BSI TR03127)

European projects, regulations, applications
Security assumptions?
- DDH for the used groups.
- PRF for key derivation functions.
- EUF-CMA for used signatures.
- ...
- Secure platform.

Attacker model?
- Passive eavesdropper.
- Active eavesdropper.
- MRTD forger.
- Terminal forger.
- Active person-in-the-middle.
- ...

Security reductions?
- PACE.
- ...
- Combined system?
Introduction

MRTDs (Machine Readable Travel Documents)

eMRTDs

Security mechanisms

eMRTD PKI

ePassport application (BSI summary and extension)

eDAS application (BSI TR03110)

Security?

**German ID card and residence permit (BSI TR03127)**

- Authentication methods
- Applications

European projects, regulations, applications
German ID card and residence permit (BSI TR03127): Authentication methods

- PACE, TA2, PA, CA2.
- Resident permits only: BAC, TA1, CA1.
- Not implemented: AA, for data protection reasons, namely challenge semantics.
German ID card and residence permit (BSI TR03127):
Authentication methods

GAP

- Read EF.CardAccess.
- Enter/read PACE password, ie. eID PIN, CAN or MRZ.
- PACE.
- Transmit certificate chain and TA2.
- Read EF.CardSecurity.
- PA(EF.CardSecurity).
- CA2.
- Optional on authentication terminal:
  - Document validity query.
  - Read revocation token.
  - Check revocation list.
- Inspection system:
  - Read \( SO_D \) (EF.SOD).
  - \( PA(SO_D) \).
  - Subsequently, compare the hash of each read data group with \( SO_D \).
- Read approved data or exercise special rights.

For residence permits additionally the standard or advanced ePassport inspection procedure may be used.
German ID card and residence permit (BSI TR03127):
Applications

- biometrics (ePassport).
- eID.
  - Document valid?
  - Read revocation token.
  - Special functions:
    - Generate signature key pair.
    - Set eID PIN, activate/deactivate eID application.
    - Pseudonym (service and card specific identifier).
    - Age verification.
    - Community ID verification.
- eSIGN.
  - Qualified signature.
  - Set signature PIN.
- Unauthenticated.
  - Change eID PIN using the current one.
  - Reset RC of eID PIN or signature PIN with PUK.
- Online authentication.
  - GAP/ERA.
Introduction

MRTDs (Machine Readable Travel Documents)

eMRTDs

Security mechanisms

eMRTD PKI

ePassport application (BSI summary and extension)

eIDAS application (BSI TR03110)

Security?

German ID card and residence permit (BSI TR03127)

European projects, regulations, applications

Europe promotes eIDAS

eIDAS: become a provider

eIDAS: German ID card application provider
European projects, regulations, applications:
Europe promotes eIDAS

- eIDAS (electronic IDentification, Authentication and trust Services) is an EU regulation on electronic identification and trust services for electronic transactions in the internal market.

  - Advanced electronic signature.
  - Qualified electronic signature.
  - Qualified digital certificate for electronic signature.
  - Trust service.
    - Electronic signatures.
    - Time-stamps.
    - Seals.
    - Certificates
European projects, regulations, applications:
Europe promotes eIDAS

click here
European projects, regulations, applications: Europe promotes eIDAS
European projects, regulations, applications:

eIDAS: become a provider

2017/06 “Gesetz zur Förderung des elektronischen Identitätsnachweises” (German law for promotion of eID)

- Usage options:
  - Client management.
  - Age verification.
  - Pseudonym access.
  - Access control.
Examples of applications.

- Portal solutions, insurance companies.
  - Allianz Insurance Company: Registration for the online service “Meine Allianz”.
  - CosmosDirekt Insurance Company: Substitute for Postident procedure for registering and concluding contracts at “mein CosmosDirekt”.
  - HUK24 – The Online Insurance: Registration in the service section “Meine HUK24”.
  - LVM Versicherung (insurance company): Registration at the client portal “Meine LVM”.
  - WGV-Versicherungen (insurance company): Registration at the client portal.
- Portal solutions, health insurance companies.
  - KKH Kaufmännische Krankenkasse (health insurance): First registration and login at the client portal “Online-Servicezentrum”.
- Portal solutions, other.
  - DATEV: Employees online — Portal for wage and salary statements.
- German Emissions Trading Authority: Online registration of an electronic mailbox.
- Christoph Kroschke: Electronic vehicle registration.
- SCHUFA Holding (Protective association for general collateral security): Registration and login at the portal “meineSCHUFA.de”.

- Supporting solutions.
  - DB Vertrieb GmbH: Sign-up for direct debiting.
  - Fabasoft Distribution: Folio Cloud — The cloud platform for business solutions.
- De-Mail.
  - Registration of an account.
  - High security login.
- AusweisApp2.
  - OpenPGP-eID certifies your OpenPGP key against your identity card.
  - ... many more online ...
Global overview

Organizational
Webpage & mailing list
Time & place
Hand-in & exam
Preliminaries
Overview
Barrier
Secure emails?
PGP / GnuPG
GnuPG
Attacks and defenses
Toolbox cryptography
Foundations
Public-key encryption
Symmetric encryption
Hybrid Encryption and the KEM/DEM Paradigm
Public-key signatures
Message authentication codes
Key exchange
PKI and certificates
Trust anchor
Randomness

Secure internet
Secure connections
Secure channel
Model, placement
TLS
History
Statistics and more
Structure
Protocol details
Record layer
Handshake
Computing the master secret
Further key exchange options
OWASP Transport Layer Protection Cheat Sheet
Attacks on TLS
Heartbeat and heartbleed
CRIME (2012)
Lucky13 (2013)
Attack history
Security for TLS

Perfect Forward Secrecy
Beagle Boys attack
Escrow attack
Fine-grained attacks
Denial of Service
DOS attack
DOS Mitigation
Replay protection and Live partner reassurance
Replay attack
Replay mitigation
Live partner reassurance
Security aspects +
Typical secure connection protocol

Pros & cons
Security goals
Data on chip

Security mechanisms
Tasks
Accessing
Gain access and establish secure messaging
BAC
PACE
Authenticating
PA
AA
CA
PACE with Chip Authentication Mapping
Secure Messaging

eMRTD PKI
Roles and responsibilities
Key management
Distribution mechanisms

ePassport application (BSI summary and extension)
Tasks
Inspection
More subprotocols
TA version 1
Country verification PKI
eIDAS application (BSI TR03110)
eID data
Again more subprotocols
Extended General Authentication Procedure
TA version 2
CA version 2
RI
Sector specific revocation of an IC
Pseudonymous signatures
PS
CA version 3

Security?
German ID card and residence permit (BSI TR03127)
Authentication methods
Applications

European projects, regulations, applications
Europe promotes eIDAS
eIDAS: become a provider
eIDAS: German ID card application provider
Part III

Winter 2017/2018

Cryptography (4+2)
Symmetric and public-key cryptography in the light of security reductions.

SATiC: Seminar Advanced Topics in Cryptography (2)
Current research.

Master theses, lab
Any time ... just ask me. Some topics:
https://cosec.bit.uni-bonn.de/students/theses/.