# Public-Key Cryptography Standards

Thomas Aregger

David Daniel

Marc Dünki

Jürg Gutknecht

# Inhalt

- Übersicht
- Die einzelnen Standards im Überblick
- Ressourcen
- Q&A

# Übersicht

- Was sind Public-Key Cryptography Standards?
  - Eine Reihe von Standards zur Public-Key-Kryptografie
  - Von RSA Laboratories seit den 1990er-Jahren veröffentlicht
  - Es gibt zur Zeit 12 Standards, ein weiterer befindet sich in Entwicklung

# PKCS #1: RSA Cryptography

- Key Typen
- Kryptografische Primitive
- Verschlüsselungs-Schemas
- Signierungs Schemas

# Key Typen

## Public Key

n = RSA Modulo (Produkt aus mindestens 2 Primzahlen)

e = Public Exponent (Zahl zwischen 3 und n-1)

## Private Key

n = RSA Modulo

d = Private Exponent (Inverses Element von e mod (p-1)(q-1))

# Kryptografische Primitive

- Verschlüsselung (RSAEP)
- Entschlüsselung (RSADP)
- Signatur (RSASP1)
- Verifikation (RSAVP1)

# Verschlüsselungs Schemas

- Verschlüsselungs- und Entschlüsselungsoperation

**RSAES-OAEP**

1. Encoding der Nachricht (EME-OAEP)
2. Konvertieren in Integer Version (OS2IP)
3. Anwenden der RSAEP Primitive

# Signatur Schemas

- Signaturgenerierung und Signaturverifikations Operation
- Bsp: Signaturgenerierung mittels RSASSA-PSS

1. Anwenden des Encodings (EMSA-PSS-ENCODE)
2. Konvertieren in Integerversion (OS2IP)
3. Anwenden von RSASP1
4. Konvertieren in Oktet-String (I2OSP)
5. Output der Signatur

# PKCS #3:
# Diffie-Hellman Key-Agreement

- Gründe für die Verwendung des DH Key Agreement:
  - Austausch von geheimen Schlüsseln zwischen zwei Kommunikationsparteien, welche oftmals für symmetrische Kryptosysteme verwendet werden
  - Schlüsselaustausch sicherer als z.B. Bei RSA (Session-Keys)
  - Leicht zu implementieren
- Nachteile des DH Key Agreement:
  - Ohne zusätzlichen Einsatz von Authentifizierungs-Mechanismen is ein Man-In-The-Middle Attacke unbemerkbar möglich

# PKCS #3:
# Diffie-Hellman Key-Agreement

- Funktionsweise des DH-Key-Agreement:
  - Vorbereitung:
    - Zentrale Authorität bestimmt eine Primzahl ***p*** sowie eine Primitivwurzel ***g*** mit:
  - Phase 1:
    - Beide Parteien wählen eine Geheimzahl ***x*** mit:
    - Beide Parteien berechnen den öffentlichen Wert ***y*** mit:
    - Beide Parteien tauschen den berechneten Wert ***y*** aus

# PKCS #3: Diffie-Hellman Key-Agreement

- Funktionsweise des DH-Key-Agreement (Fortsetzung):
  - Phase 2:
    - Beide Parteien berechnen nun mit dem erhaltenen Wert ***y*** , der eigenen Geheimzahl ***x*** sowie dem öffentlich bekannten Wert ***p*** das geminsame Geheimnis ***z*** mit:

Dieser Wert ***z*** kann ab sofort von beiden Parteien für das Ver- sowie Entschlüsseln verwendet werden.

# PKCS #3:
# Diffie-Hellman Key-Agreement

- Der Standard beschreibt eine Methode zur Implementierung des DH-Schlüsselaustauschs
- Das Protokoll stützt sich auf das Problem des diskreten Logarithmus
- Deckt lediglich das Schutzziel der Vertraulichkeit ab
- Es sollte daher mit Mechanismen der Authentizität kombiniert werden

# PKCS #5: Password-Based Cryptography

- Passwörter sind als Schlüssel nicht geeigenet
  - Grund: Teil eines zu kleinen Zeichenraums (Anzahl möglicher Passwörter ist klein)
- Abhilfe: Verwendung von Schlüsselableitungsfunktionen (mit Hilfe von Salt und Wiederholungszähler)
- **Salt** ist zufälliger Wert welchem dem Passwort angehängt wird
- **Wiederholungszähler** bestimmt wie oft die Ableitungsfunktion ausgeführt wird, und dient dazu die Operation rechenintensiver zu machen (und somit auch das Knacken erschweren)

# Schlüsselableitungsfunktion PBKDF2

- Der abgeleitete Schlüssel besteht aus verschiedenen Blöcken
- Für jeden Block wird eine Pseudozufalls Funktion mehrere Male ausgeführt (gemäss Wiederholungszähler) und mit dem vorherigen Resultat XORed (Input der Funktion bei der ersten Ausführung ist das Passwort, der Salt sowie die Nummer des Blocks (siehe Schritt 1). Bei allen weiteren Ausführungen das Passwort und das Resultat der letzten Iteration.
- Am Schluss werden die einzelnen Blöcke zusammengesetzt und bilden den abgeleiteten Schlüssel

# PKCS #6: Extended-Certificate Syntax

- Der Standard beschreibt eine Syntax für erweiterte Zertifikate (extended certificates)
- Ein erweitertes Zertifikat ist ein X.509-Public-Key-Zertifikat, erweitert um eine Menge von Attributen
- Es wird nicht mehr nur das Zertifikat verifiziert, sondern auch in den Attributen enthaltene Informationen des Zertifikatshalters (E-Mail-Adresse etc.)

# PKCS #6: Extended-Certificate Syntax

- Erstellen eines erweiterten Zertifikats
  1) Eine `ExtendedCertificateInfo` wird von der Ausgabestelle erstellt
  2) Der Wert der `ExtendedCertificateInfo` wird mit dem privaten Schlüssel der Ausgabestelle signiert
  3) Informationen, Signatur und der Algorithm Identifier werden in einem `ExtendedCertificate` zusammengeführt

# PKCS #6:
# Extended-Certificate Syntax

```
ExtendedCertificateInfo ::= SEQUENCE {
   version Version,
   certificate Certificate,
   attributes Attributes
}
Version ::= INTEGER
Attributes ::= SET OF Attribute
```

# PKCS #6:
# Extended-Certificate Syntax

```
ExtendedCertificate ::= SEQUENCE {

   extendedCertificateInfo ExtendedCertificateInfo,

   signatureAlgorithm SignatureAlgorithmIdentifier,
   signature Signature

}
SignatureAlgorithmIdentifier ::= AlgorithmIdentifier
Signature ::= BIT STRING
```

# PKCS #7:
# Cryptographic Message Syntax

- Definiert eine Syntax zur Beschreibung von kryptographischen Inhalten (Signaturen, Schlüssel etc.)
- Anwendungen:
  - Signieren von Nachrichten
  - Digest (hash) von Nachrichten
  - Authentisierung von Nachrichten (MAC)
  - Verschlüsselung digitaler Inhalte

# PKCS #7: Aufbau

- Zwei Klassen von Inhalten
  - **base:** Enthält “plain data” (keine kryptographischen Erweierungen)
  - **enhanced:** Enthält Inhalt eines bestimmten Typs (evtl. Verschlüsselt) und weitere kryptographische Erweiterungen

# PKCS #7:
# Inhalts Typen

- Insgesamt werden 6 Inhalts Typen definiert
  - data (einziger Typ der Kategorie “base”)
  - signed data
  - enveloped data
  - signed and enveloped data
  - digested data
  - encrypted data

# PKCS #7: Definition des Inhaltes

```
ContentInfo ::= SEQUENCE {
    contentType ContentType,
    content [0]
       EXPLICIT ANY DEFINED BY contentType OPTIONAL
}
```

- contentType benennt den Typen des Inhaltes (Object identifier, data, signed etc.)
- content ist der Inhalt der Nachricht – optional, der Inhalt kann auch anderweitig definiert werden (contentType)

# PKCS #7:
# Inhalt signieren

- Für jeden Signierer wird der Message Digest des Inhaltes gemäss Algorithmus des Signierers erstellt.
- Für jeden Signierer wird mit dessen privatem Schlüssel jeder Message Digest und die zugehörigen Angaben verschlüsselt.
- Für jeden Signierer werden der verschlüsselte Message Digest und andere Signierer-spezifische Angaben in einem SignerInfo Objekt abgelegt. Zertifikate und certificate-revocation listst werden in diesem Schritt ermittelt.
- Sämtliche Message-Digest Algorithmen und `SignerInfo` Objekte werden mit dem Inhalt im `SignedDataValue` Objekt abgelegt.

# PKCS #8: Private-Key Information Syntax

- Beschreibt die Syntax zu Speicherung von Private-Key Informationen

```
PrivateKeyInfo ::= SEQUENCE {
    version Version,
    privateKeyAlgorithm PrivateKeyAlgorithmIdentifier,
    privateKey PrivateKey,
    attributes [0] IMPLICIT Attributes OPTIONAL
}
Version ::= INTEGER
PrivateKeyAlgorithmIdentifier ::= AlgorithmIdentifier
PrivateKey ::= OCTET STRING
Attributes ::= SET OF Attribute
```

# Verschlüsselte Private Key Information Syntax

```
EncryptedPrivateKeyInfo ::= SEQUENCE {
    encryptionAlgorithm EncryptionAlgorithmIdentifier,
    encryptedData EncryptedData
}
EncryptionAlgorithmIdentifier ::= AlgorithmIdentifier
EncryptedData ::= OCTET STRING
```

# PKCS #9: Selected Object Classes & Attribute Types

- Behandelt allgemeine Objektklassen und Attribute, welche von anderen PKCS Dokumenten referenziert werden.
- Definiert zwei neue Objektklassen und deren Attribute:
  - `pkcsEntity`
  - `NaturalPerson`
- Für die Verwendung in Verzeichnis-Diensten (LDAP)

# PKCS #9:
# `pkcsEntity`

- Soll die Möglichkeit bieten, Attribute von beliebigen PKCS Entitäten zu beherbergen.
- Minimaler Aufbau:

```
pkcsEntity OBJECT-CLASS ::= {
    SUBCLASS OF { top }
    KIND auxiliary
    MAY CONTAIN { PKCSEntityAttributeSet }
    ID pkcs-9-oc-pkcsEntity
}
```

- Neben der ID kann die Entity ein Attribut aus dem `PKCSEntityAttributeSet` beinhalten

# PKCS #9:
## `PKCSEntityAttributeSet`

- `pKCS7PDU` beinhaltet die in PKCS #7 definierten geschützten Daten (enveloped, signed etc.)
- `userPKCS12` enthält die Angaben über die persönliche Identität wie in PKCS #12 definiert.
- `pKCS15Token` wird für die Verwendung der in PKCS #15 definierten kryptographischen Tokens verwendet.
- `encryptedPrivateKeyInfo` enthält das in PKCS #8 definierte Format für verschlüsselte private Schlüssel.

# PKCS #9:
# `naturalPerson`

- Objektklasse – beherbergt Attribute von Menschen (natürlichen Personen).
- Gleicher Aufbau wie pkcsEntity
  - Andere ID (`pkcs-9-oc-naturalPerson`)
  - Anderes Attribut-Set (`NaturalPersonAttributeSet`)

# PKCS #9: `NaturalPersonAttributeSet`

- `emailAddress`
  - 1 oder mehrere *unstrukturierte* E-Mail Adressen
  - EQUALITY MATCHING RULE pkcs9CaseIgnoreMatch (beim Vergleich zweier Werte wird die Gross- Kleinschreibung nicht beachtet)
- `unstructuredName`
  - Unstrukturierter Name (ebenfalls pkcs9CaseIgnoreMatch)
- `unstructuredAddress`
  - Unstrukturierte Adresse (pkcs9CaseIgnoreMatch)
- `dateOfBirth`
  - In Form der GeneralizedTime
  - Darf nur einmal vorkommen (SINGLE VALUE TRUE)
- `placeOfBirth`
  - Unstrukturierter Text, welcher den Geburtsort nennt (case sensitive)
- `gender`
  - Darf nur einmal vorkommen - “F”, “f”, “M”, “m”
- `countryOfCitizenship, countryOfResidence, pseudonym, serialNumber`

# PKCS #9: ausgewählte Attribute, ein Auszug

- `contentType`
  - Spezifiziert den Inhalts-Typen des in PKCS #7 (oder S/MIME) signierten ContentInfo Objektes
- `messageDigest`
  - Spezifiziert den Message Digest der Inhalte des content Feldes des ContentInfo Objektes. Zwingend, falls authentifizierte Attribute aus PKCS #7 verwendet werden.
- `randomNonce`
  - Ein einmaliger Zufallswert – kann zwecks Schutz vor replay Attacken verwendet werden.
  - Oktett-String (mind. 4 Byte lang)
  - Darf nur einmal vorkommen
- `counterSignature`
  - Erlaubt die Signatur von Signaturen (kann beliebig verschachtelt werden)

# PKCS #10: Certification Request Syntax

- Der Standard beschreibt eine Syntax für Zertifizierungsanfragen
- Die Anfragen werden an eine Zertifizierungsstelle (CA) geschickt
- Die CA wandelt die Anfrage in ein X.509-Zertifikat um

# PKCS #10: Certification Request Syntax

- Erstellen einer Anfrage

  1) Eine `CertificationRequestInfo` wird durch die Anfragestelle generiert

  2) `CertificationRequestInfo` wird mit dem privaten Schlüssel der Anfragestelle signiert

  3) Zusammenführen der drei Bestandteile in ein `CertificationRequest`

# PKCS #10: Certification Request Syntax

- Die CA erhält die Anfrage, authentisiert die Anfragestelle und überprüft die Signatur
- Ist die Anfrage gültig, wird ein X.509-Zertifikat erstellt
- Die Art und Weise, in welcher die CA das Zertifikat zustellt, wird im Standard nicht definiert.
  - Möglichkeiten: Als kryptografische Nachricht oder in Papierform

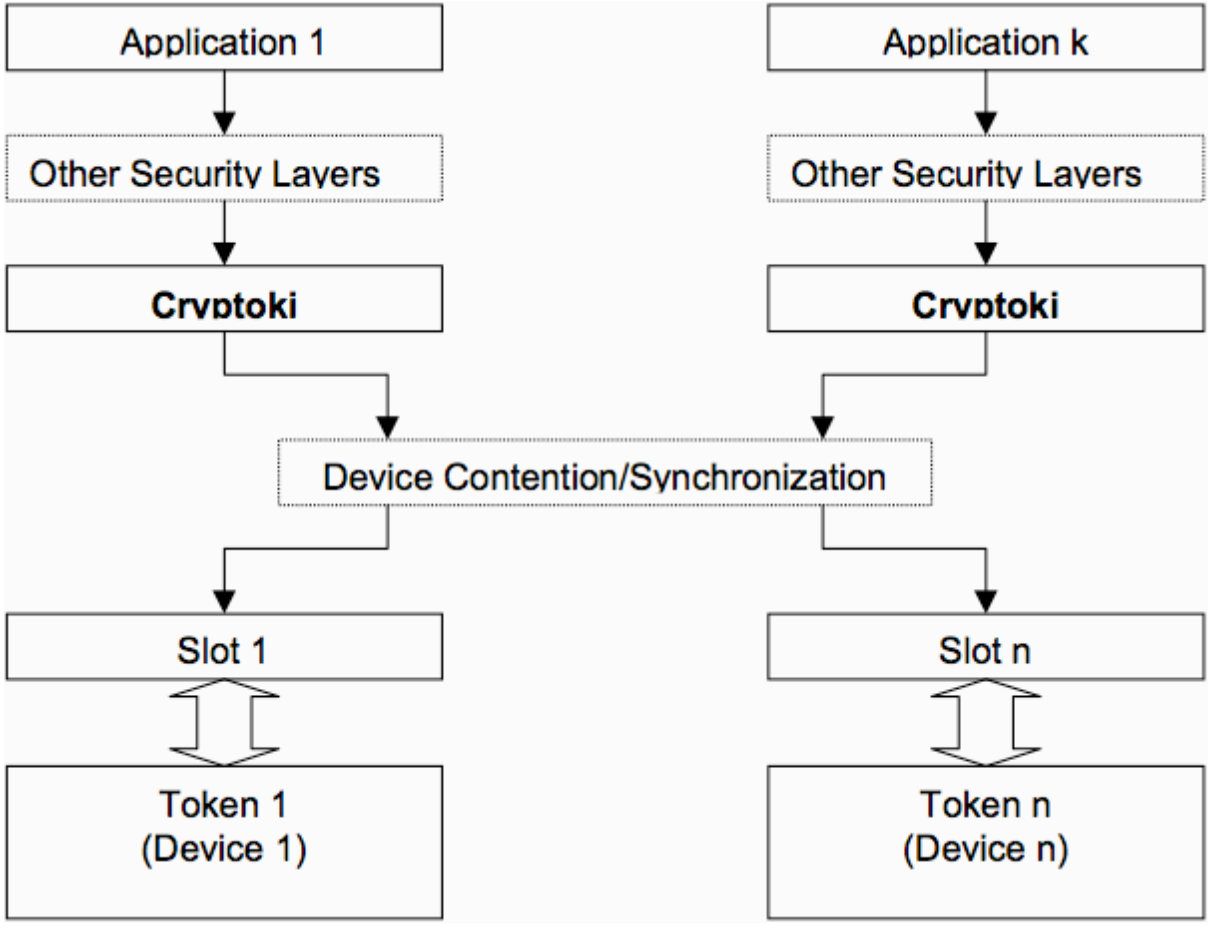
# PKCS #10:
# Certification Request Syntax

```
CertificationRequest ::= SEQUENCE {
   certificationRequestInfo CertificationRequestInfo,
   signatureAlgorithm AlgorithmIdentifier{{ SignatureAlgorithms}},
   signature BIT STRING
}
AlgorithmIdentifier {ALGORITHM:IOSet } ::= SEQUENCE {
   algorithm ALGORITHM.&id({IOSet}),
   parameters ALGORITHM.&Type({IOSet}{@algorithm}) OPTIONAL
}
SignatureAlgorithms ALGORITHM ::= {
   ... -- add any locally defined algorithms here --
}
```

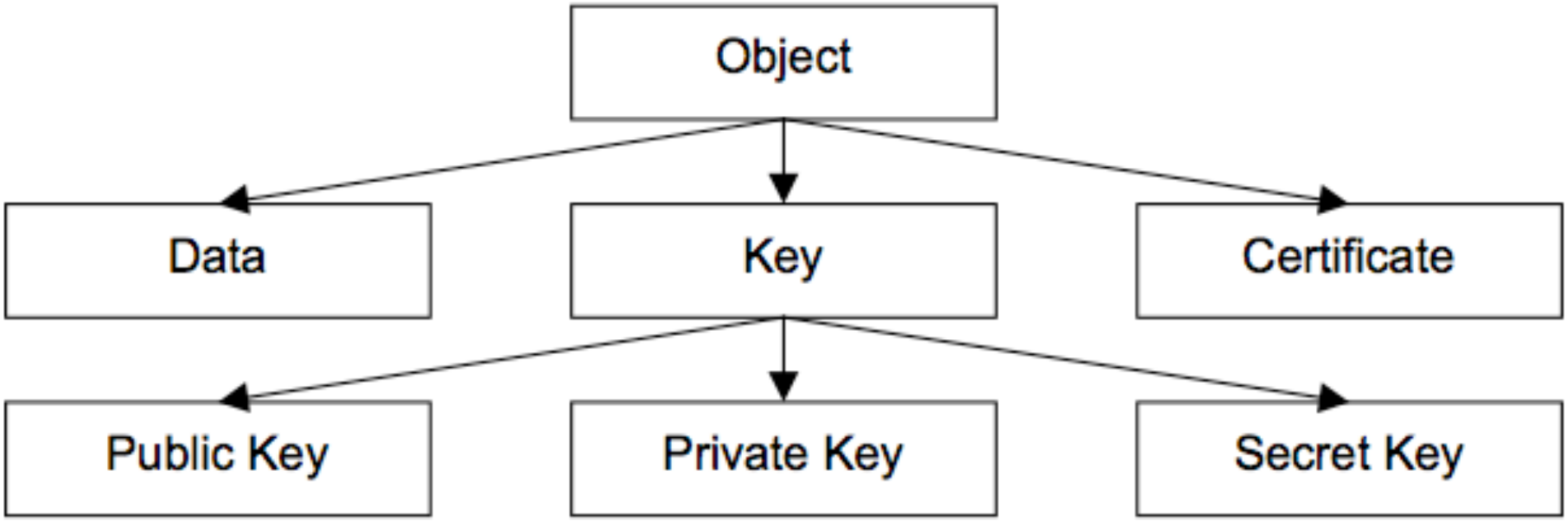
# PKCS #11:
# Cryptographic Token Interface

- Der Standard beschreibt eine API für den Zugriff auf Geräte, welche kryptografische Informationen enthalten oder kryptografische Funktionen ausführen (SmartCards etc.)
- Es werden die bereitgestellten Datentypen und Funktionen für die Sprache C spezifiziert

# PKCS #11:
# Cryptographic Token Interface



# PKCS #11:
# Cryptographic Token Interface



# PKCS #11: Cryptographic Token Interface

- Verwendung von Cryptoki
  - `C_Initialize()` aufrufen
  - Authentisierung des Benutzers
  - Starten von Sessions (Read-Only oder Read-Write)
  - Lesen/schreiben von Objekten
  - Ausführen kryptografischer Funktionen (z.B. RSA-Schlüsselpaargenerierung)
  - Beenden der Sessions und Logout
  - `C_Finalize()`

# PKCS #12: Personal Information Exchange Syntax

- Standardisiert eine Syntax für den direkten Austausch von persönlichen Identifikations Informationen wie z.B.:
  - Private Schlüssel
  - Zertifikate
  - Sonstige geheime Daten
  - Erweiterungen

# PKCS #12: Personal Information Exchange Syntax

- Gründe für die Verwendung:
  - Kann sowohl in Software als auch in Hardware zum Einsatz kommen
  - Anwendungen, Browser, etc. welche diesen Standard unterstützen ermöglichen es Usern, ein einzelnes Set von persönlichen Identitätsinformationen zu importieren und exportieren
  - Erweiterung von PKCS #8, wobei noch eine höhrere Sicherheitsstufe ermöglicht wird durch den Einsatz von Public-Key-Verfahren

# PKCS #12: Personal Information Exchange Syntax

- Einsatz:
  - Es werden zwei Sicherheitsstufen unterstützt
    - Höchste Stufe: Public/Private Key Paare für Verschlüsselung und Authentizität
    - Tiefere Stufe: Passwort basierte Privacy und Integrity Modi, wenn die höchste Stufe nicht eingesetzt werden kann
  - Es ergeben sich insgesamt vier mögliche Kombinationen:
    - Privacy-Modi: Public-Key-Privacy Modus, Password Privacy Modus
    - Integrity-Modi: Public-Key-Integrity Modus, Password Integrity Modus

# PKCS #13:
# Elliptic Curve Cryptography

- Bis heute erst ein Entwurf
- Soll die folgenden Punkte abdecken:
  - Parameter und Schlüssel Erzeugung und Validierung
  - Digitale Signaturen
  - Public Key Verschlüsselung
  - Key Agreement
  - ASN.1 Syntax
  - Überlegungen zur Sicherheit

# PKCS #13:
# In Anlehnung an...

- Es sind bereits Standards in Arbeit, welche die Kryptographie mit elliptischen Kurven adressieren:
  - ANSI X9.62
    - Standard für digitale Signaturen (in Entwicklung)
  - ANSI X9.63
    - Standard für Key Agreement (in Entwicklung)
  - IEEE P1363
    - Generelle Referenz für Public Key Verfahren, inkl. elliptische Kurven (in Entwicklung)
- PKCS #13 soll:
  - bestehende Standards vervollständigen.
  - ein Profil der anderen Standards im PKCS Format liefern.
  - eine Anleitung für die Integration in andere PKCS Anwendungen (z.B. PKCS #7) bieten.

# PKCS #13:
# Aktueller Umfang

- Zurzeit existieren lediglich Vorschläge:
  - Mögliche Key Agreement Schemes
  - Mögliche Signature Schemes
  - Mögliche Encyption Schemes
  - Mögliche Punkt-Repräsentationen

# PKCS #15: Cryptographic Token Information Syntax

- Spezifiziert sowohl ein Datei- als auch ein Pfadformat, um sicherheitsrelevante Informationen auf krypt. Tokens zu speichern
- Die Realisierung kann sowohl in Hardware für Smartcards, als auch in Software für virtuelle Smartcards erfolgen
- Ist jedoch veraltet und wird durch ISO/IEC 7816-15 abgelöst

# PKCS #15:
# Cryptographic Token Information Syntax

- Ziele von PKCS#15 :
  - Platform Interoperabilität zwischen Komponenten, welche auf verschiedenen Platformen laufen, zu erreichen
  - Produkte Unabhängigkeit
  - Applikationsunabhängigkeit
  - Konsistenz von bestehender, verwendeter Standarts erhalten und diese wenn nötig weiter entwickeln

# PKCS #15: Cryptographic Token Information Syntax

- Profile für das Datenformat/Zugrifssberechtigungen
  - Definieren die Verzeichnisstrukturen und Dateizugriffsbedingungen für den Einsatz von Tokens
  - Sind nötig um die Konformität zu gewähren
  - Um ein Token zu verwenden, wird ein EID (Electronic Identification Profile) benötigt, wobei auch private Schlüssel u.A. für die PIN-Verifizierung zum Einsatz kommen
- Ein EID besteht aus:
  - Verzeichnisstruktur
    - Definiert das Layout resp. die Hierarchieeben von Masterfile, Elementaryfiles und Directoryfiles für das Profile
  - Dateizugriffsbedingungen
    - Regelt die Rechte von Read / Write Zugriffen auf die Files für das Profil

# Ressourcen

- PKCS-Seiten der RSA Laboratories
- http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/public-key-cryptography-standards.htm

# Q&A

?