Semesterarbeit Kryptographie - Gruppenarbeit PKCS

Aregger Thomas thomas.aregger@students.ffhs.ch
Dünki Marc marc.duenki@students.ffhs.ch
Gutknecht Jürg juerg.gutknecht@students.ffhs.ch
Daniel David david.daniel@students.ffhs.ch

3. Januar 2014

Inhaltsverzeichnis

1 Einleitung

Dieses Dokument liefert einen Überblick über das Themengebiet der Public-Key Cryptography Standards und entstand als Teil der Semesterarbeit des Moduls Kryptologie bei Herrn Josef Schuler.

1.1 Gruppe

Die Gruppe der an dieser Arbeit mitwirkenden Studenten umfasst:

- Thomas Aregger
- David Daniel
- Marc Dünki (Projektleiter)
- Jürg Gutknecht

1.2 Vorgehensweise

Datum	**Fortschritt**
7.9.2013	Gruppenbildung
8.9.2013 - 4.10.2013	Individuelles Erarbeiten der Aufgabenstellung
4.10.2013	Aufteilen der Themen innerhalb der Gruppe
4.10.2013 - 28.12.2013	Individuelles Bearbeiten der Inhalte gemäss Themenverteilung
28.12.2013	Besprechen des Fortschritts
28.12.2013 - 4.1.2014	Individuelles Finalisieren der Inhalte gemäss Themenverteilung
4.1.2014	Verteilen der Aufgaben zum Abschliessen der Arbeit
4.1.2014 - 9.1.2014	Abschliessen der Arbeit
	Abgabe der Arbeit

Tabelle 1: Vorgehensweise

1.3 Arbeitsteilung

1.3.1 Verteilung der Themen

PKCS#	Person
1	Thomas Aregger
3	Marc Dünki
5	Thomas Aregger
6	Jürg Gutknecht
7	David Daniel
8	Thomas Aregger
9	David Daniel
10	Jürg Gutknecht
11	Jürg Gutknecht
12	Marc Dünki
13	David Daniel
15	Marc Dünkis

Tabelle 2: Verteilung der Themen

1.3.2 Weitere Aufgaben

Aufgabe	Person
Zusammenführung der Beiträge in einem Dokument	David Daniel
Zusammenfassung der Beiträge in einer Präsentation	
Abgabe der Aufgabe	

Tabelle 3: Verteilung weiterer Aufgaben

2 Public-Key Cryptography Standards

Die Public Key Cryptography Standards (PKCS) sind eine von RSA Laboratories [RSA-Lab] seit den 1990er- Jahren veröffentlichte Reihe an Standards zum Themengebiet der asymmetrischen Kryptographie (Public-Key Kryptographie).

Die Gruppe der PKCS besteht aus den nachfolgend aufgelisteten Standards, welche in diesem Kapitel detailliert betrachtet werden.

PKCS #	Titel
1	RSA Cryptography Standard
3	Diffie-Hellman Key-Agreement Standard
5	Password-Based Cryptography Standard
6	Extended-Certificate Syntax Standard
7	Cryptographic Message Syntax Standard
8	Private-Key Information Syntax Standard
9	Selected Object Classes and Attribute Types
10	Certification Request Syntax Standard
11	Cryptographic Token Interface Standard
12	Personal Information Exchange Syntax
13	Elliptic Curve Cryptography Standard
14	Pseudo Random Number Generation
15	Cryptographic Token Information Syntax Standard

Tabelle 4: Übersicht der PKCS

Die PKCS #2 und #4 wurden in den PKCS #1 überführt und sind nicht mehr eigener Bestandteil der PKCS-Familie [KAL91, S.1].

PKCS #14 befindet sich zur Zeit in Entwicklung, wird auf den PKCS-Seiten der RSA Laboratories [PKCS-Standards] nicht erwähnt und wird hier ebenfalls nicht genauer betrachtet.

2.1 PKCS #1: RSA Cryptography Standard

Das PKCS Dokument #1 gibt Empfehlungen zur Implementierung von Public Key Kryptografie auf Basis des RSA Algorithmus. Dabei werden hauptsächlich die folgenden Aspekte abgedeckt, welche nachfolgend genauer erläutert werden:

- Kryptografische Primitive
- Verschlüsselungs Schemas (Encryption schemes)
- Signierungs Schemas (Signature schemes)

2.1.1 Key Typen

Bei RSA existieren 2 unterschiedliche Key Typen, welche zusammen als RSA Key Pair bezeichnet werden:

- *RSA Public Key* wird zum verschlüsseln bzw. verifizieren benutzt.
- *RSA Private Key* wird zum entschlüsseln bzw. signieren benutzt.

2.1.2 RSA Public Key

Der Public Key besteht aus zwei Komponenten, nämlich einem Modulo und einem öffentlichen Exponenten (Public exponent). Beides sind positive Integer.

n = RSA Modulo. Der Modulo ist ein Produkt von mindestens 2 Primzahlen. In Lehrbüchern und Beispielen werden häufig nur 2 Primzahlen multipliziert,

es können aber auch mehr verwendet werden. $(r_1, r_2, \ldots, r_3)$ r_1 und r_2 werden häufig auch als p und q bezeichnet.

e = Public exponent. Zahl zwischen 3 und $n-1$, welche relativ Prim (teilerfremd) zu $(r_1 - 1) * (r_2 - 1) * (r_3 - 1)$ ist.

Der Einfachheit halber wird in dieser Zusammenfassung immer davon ausgegangen, dass bei der Generierung von n nur 2 Primzahlen verwendet wurden.

2.1.3 RSA Private Key

Der Private Key besteht aus dem selben Modulo n wie der Public Key. Zusätzlich existiert ein privater Exponent d:

n = RSA Modulo (siehe Kapitel 2.1.2 (RSA Public Key)).

d = Private Exponent. $e * d \equiv 1 \mod (p-1)(q-1)$. d ist also das Inverse von $e \mod (p-1)(q-1)$.

Bemerkung: Würden mehr als 2 Primzahlen verwendet werden, würde der Private Key noch einige Komponenten mehr enthalten.

2.1.4 Kryptographische Primitive

Kryptographische Primitive bezeichnen grundlegende mathematische Operationen auf denen die kryptographischen Schemas aufbauen. Es werden 4 Typen von Primitiven spezifiziert:

- Verschlüsselung & Entschlüsselung
- Signierung und Verifikation

Verschlüsselungs und Entschlüsselungs Primitive Ein Verschlüsselungsprimitiv erstellt unter Verwendung des Public Keys aus einer Nachricht ein Chiffrat. Das Entschlüsselungsprimitiv gewinnt unter Verwendung des Private Keys wieder die Nachricht aus dem Chiffrat. RSA definiert die beiden Schemes RSAEP und RSADP (RSA Encryption/Decryption Primitiv). Beide verwenden die selben mathematischen Operationen, wobei einfach unterschiedlicher Input verwendet wird.

RSAEP

Input	(n, e)	Public Key
	m	Nachricht
Output	c	Chiffrat

RSADP

Input	(n, d)	Private Key
	c	Chiffrat
Output	m	Nachricht

Für die genaue Berechnung sei auf PKCS #1 [PKCS1, Kapitel 5.1] verwiesen.

RSASP1

Input	(n, d)	Private Key
	m	Nachricht
Output	s	Signatur

RSAVP1

Input	(n, e)	Public Key
	s	Signatur
Output	m	Nachricht

Signatur und Verifikation Primitive Das Signatur Primitiv erstellt unter Verwendung des Private Keys aus einer Nachricht [1] eine Signatur. Das Verifikations Primitiv gewinnt unter Verwendung des Public Keys aus der Signatur wieder die Nachricht. Die Primitiven RSASP1 und RSAVP1 (RSA Signature/Verification Primitives) funktionieren gleich wie RSADP und RSAEP, mit dem einzigen Unterschied, dass die Input und Output Argumente unterschiedlich sind.

2.1.5 Schemas

Ein Schema kombiniert kryptographische Primitive mit anderen Techniken um ein bestimmtes Schutzziel zu erreichen. In PKCS #1 werden Verschlüsselungs und Signatur Schemas spezifiziert. Die Schemas definieren nur die Schritte welche unternommen werden um die Daten zu verarbeiten. Es wird z.Bsp. keine Schlüssel generiert oder validiert.

2.1.6 Verschlüsselungs Schemas

Ein Verschlüsselungs Schema besteht aus einer Verschlüsselungs- und einer Entschlüsselungs Operation. Ein solches Schema kann für verschiedene Zwecke verwendet werden. Häufig wird es verwendet um z.Bsp. einen symmetrischen Schlüssel auszutauschen. In PKCS #1 werden zwei unterschiedliche Schemas spezifiziert. RSAES-OAEP und RSAES-PKCS1-v1_5. Für neuere Applikationen sollte nur noch ersteres verwendet werden, deshalb wird das letztere hier nicht weiter berücksichtigt.

Grundsätzlich wird bei den Schemas zuerst ein Encoding der Nachricht durchgeführt. Die codierte Nachricht wird dann in eine Integerversion der Nachricht konvertiert. Auf diese Integerversion der Nachricht wird anschliessend die Verschlüsselungs Primitve angewendet um das Chiffrat zu erstellen. Genau umgekehrt funktioniert das Verfahren um wieder den Klartext zu erhalten.

RSAES-OAEP RSAES-OAEP kombiniert die RSAEP/RSADP Primitive mit der EME-OAEP Encoding Methode. OAEP steht für Optimal Asymmetric Encryption Padding und dient dazu das Kryptosystem gegen Chosen-Plaintext Attacken zu schützen in dem bei jeder Nachricht ein Zufalls-Padding vorgenommen wird. Dieses sehr wichtige Padding wird im Schulbuch RSA selten erwähnt.

[1] Wie später noch gezeigt wird, muss es sich bei der Nachricht nicht um den gleichen Text handeln wie bei der Verschlüsselung. Es kann z.Bsp. auch der Hashwert der verschlüsselten Nachricht sein.

Um den Rahmen dieses Dokumentes nicht zu sprengen, wird im folgenden nur die Verschlüsselung beschrieben. Die Entschlüsselung funktioniert nach einem ähnlichen Prinzip.

Grobes Verfahren:

1. Länge der Nachricht prüfen
2. Encoding vornehmen (Details siehe EME-OAEP 2.1.6)
3. Konvertieren in Integer Version, mithilfe von OS2IP (Octet String to Integer Primitive)
4. Anwenden der RSAEP Primitive

EME-OAEP

1. Generieren des Hashwertes lHash des Labels L [2] (welches in dieser Version der Spezifikation leer ist. Somit wird hier immer der gleiche Hashwert verwendet, welcher jedoch natürlich abhängig von der verwendeten Hashfunktion ist)
2. Generieren eines Null Oktet Strings PS der Länge k - mLen - 2hLen
3. Zusammenfügen von lHash, PS, einem Oktet `0x01` und der Nachricht M. Wird zusammen als data block DB bezeichnet.
4. Generieren eines Random Strings seed
5. Anwenden einer MGF [3] auf dem seed. Output = dbMask
6. maskedDB = DB xor dbMask
7. Anwenden einer MGF auf maskedDB. Output = seedMask
8. maskedSeed = seed xor seedMask
9. Das Oktet `0x00`, maskedSeed und maskedDB bilden zusammen die codierte Nachricht.

k	Länge des Modulo n
mLen	Länge der Nachricht
hLen	Länge des Hashwertes

[2] In PKCS #1 werden für RSAES-OAEP lediglich die Hashfunktionen SHA-1 und SHA-256/384/512 empfohlen.

[3] MGF = Mask Generation Functions werden hier nicht näher erläutert. Für Details sei auf das Kapitel B.2 in PKCS #1 verwiesen

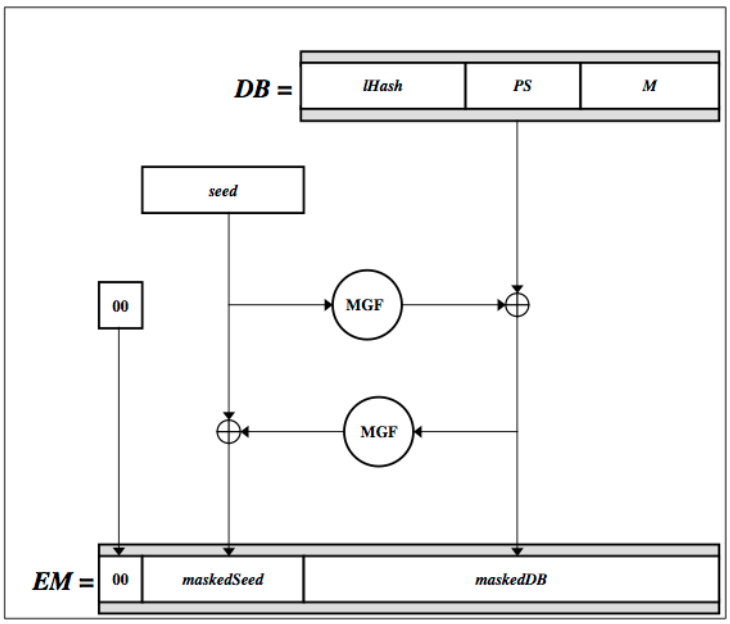

Abbildung 1: Grafische Ansicht des EME-OAEP. Quelle: PKCS #1

2.1.7 Signatur Schemas

Ein Signatur Schema besteht aus einer *signature generation operation* und einer *signature verifcation operation*. Die Erstere wird von der unterzeichnenden Partie mit Hilfe seines privaten Schlüssels verwendet um eine Signatur zu erstellen. Letztere wird von der prüfenden Partie mit Hilfe des öffentlichen Schlüssels der unterzeichnenden Partie verwendet um die Signatur zu überprüfen. Die in PKCS #1 spezifizierten Schemas werden auch als Signatur Schemas mit Anhang bezeichnet, da sie die Nachricht selbst ebenfalls benötigen um die Signatur zu verifizieren. Im Gegensatz dazu existieren auch Signatur Schemas mit Möglichkeit zur Nachricht-Wiederherstellung.

In PKCS#1 werden zwei Schemas spezifiziert. RSASSA-PSS und RSASSA-PKCS1-v1_5. Für neuere Applikationen wird RSASSA-PSS empfohlen, weshalb in diesem Dokument nur auf dieses eingegangen wird.

RSASSA-PSS RSASSA-PSS kombiniert die Primitive RSASP1/RSAVP1 mit dem EMSA-PSS Encoding (Auf EMSA-PSS wird in diesem Dokument nicht näher eingegangen). RSASSA-PSS ist probabilistisch da ein zufallsgeneriertes Salt eingefügt wird (PSS = Probabilistic Signature Scheme). Die Sicherheit beruht jedoch nicht auf diesem Salt und kann immer noch als gewährleistet betrachtet werden, wenn ein solcher Salt nicht generiert werden kann.

Input	K	Private Key der signierenden Partie
	M	Nachricht
Output	S	Signatur

Signature generation operation Schritte:

1. Anwenden des Encodings EMSA-PSS-ENCODE
2. Konvertieren der codierten Nachricht in Integer (OS2IP)
3. Anwenden von RSASP1 Primitive mit Hilfe von K und M
4. Konvertieren Integer Version in Oktet-String (I2OSP)
5. Output von Signatur S

Input	(n, e)	Public Key der signierenden Partie
	M	Nachricht
Output		valide/invalide Signatur

Signature verification operation Schritte:

1. Prüfung der Signatur Länge (Muss gleich Lang wie der Modulo n sein)
2. Konvertieren der Signatur S in Integer (OS2IP)
3. Anwenden der RSAVP1 Primitive mit Hilfe vom Public Key (n,e) und der Nachricht M.
4. Konvertieren in Oktet-String (I2OSP)
5. Verifikation mittels EMSA-PSS-VERIFY
6. Ouput valide/invalide Signatur

Bei der Verifikation wird zuerst der Salt wiederhergestellt, danach der Hashwert erneut berechnet und mit dem Hashwert welcher in der Signatur enthalten ist verglichen. Bei Übereinstimmung ist die Signatur valide.

Die Abbildung 2 illustriert das EMSA-PSS-ENCODING.

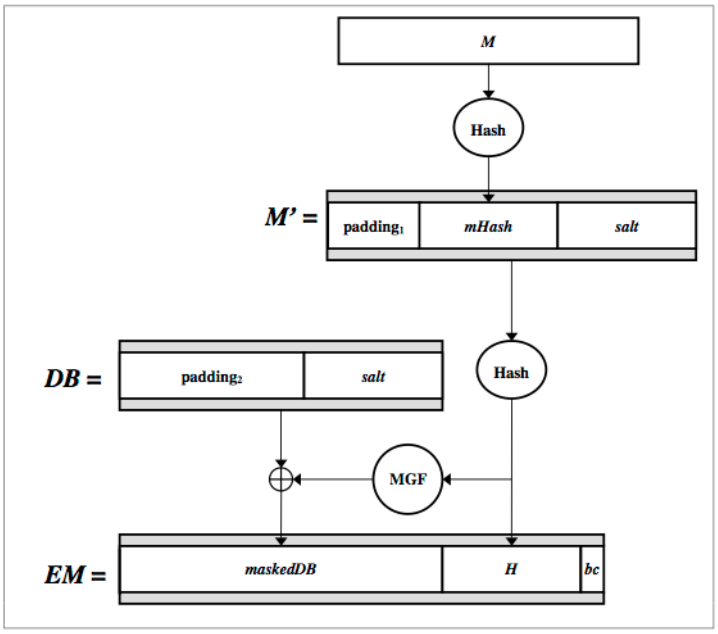

Abbildung 2: EMSA-PSS-ENCODING. Quelle: PKCS #1

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

2.2.1 Einleitung

Aktuelle Version des Standards	1.4
Datum	Revised November 1, 1993
Seitenzahl	8

Kurzbeschreibung RSA Laboratories

> *This standard describes a method for implementing Diffie-Hellman key agreement. The intended application of this standard is in protocols for establishing secure communications.*

2.2.2 Zusammenfassung

Scope Das Diffie-Hellmann-Schlüsselaustausch Protokoll wird in der Kryptografie verwendet, um zwischen zwei Kommunikationspartner einen geheimen Schlüssel zu erzeugen. Dieser wird dann vornehmlich als geheimer Schlüssel für synchrone Verschlüsselungsmechanismen verwendet. Mit diesem Protokoll soll eine grosse Problematik der synchronen Verschlüsselung, nämlich der des Austauschs des geheimen Schlüssels, gelöst werden. Da es sich beim Diffie-Hellann um ein Protokoll handelt, welches sich auf das Problem des diskreten Logarithmus stützt, kann man für den Einsatz zum Beispiel Elliptische Kurven (ECC) verwenden.

Sicherheit Beide Kommunikationspartner senden sich eine Nachricht über ein nicht gesichertes Netz zu (Internet). Falls nun ein Angreifer diese beiden Nachrichten abfängt und aus diesen beiden Nachrichten den geheimen Schlüssel generieren möchte, muss er das Diffie-Hellmann-Problem lösen. Von diesem wird jedoch angenommen, dass es praktisch unlösbar ist. Allerdings, sobald ein Angreifer diese Nachrichten verändern kann (Man-in-the-Middle-Attack), ist der Diffie- Hellmann-Schlüsselaustausch nicht mehr sicher. Das impliziert, dass dieses Protokoll mit Mechanismen der Authentiziät kombiniert werden sollte, welche dann sicherstellen, dass Nachrichten nicht umbemerkt verändert werden können. Alternativ kann auch ein Zero-Knowledge- Beweis zum Einsatz kommen, wie z.B. das Fiat-Shamir-Verfahren. Dieses Verfahren ermöglicht es einem Kommunikationspartner zu beweisen, dass er das Geheimnis kennt, ohne es aber zu verraten.

Ablauf Schlüsselerzeugung

Generierung der Parameter Eine zentrale Authorität wählt eine ungerade Primzahl „p“. Ebenfalls wird eine Primitivwurzel „g“ gewählt, welche erfüllt dass, $0 < g < p$ gilt. Bei einigen Methoden hängt der Aufwand für das berechnen des diskreten Logarithmus vo der länge der Primzahl ab. Für andere ist die länge der privaten Geheimzahl ausschlaggebend. Deshalb kann gerade für dieses Verfahren, durch die zentrale Autorität eine maximale Länge der privaten Geheimzahl vorgegeben werden. Dies ermöglicht die Zeit für das Berechnen relativ klein zu halten, während trotzdem ein gewisses Niveau von Sicherheit gewährleistet werden kann.

Vorgehen zur Schlüsselerzeugung Der Ablauf der Schlüsselerzeugen teilt sich in zwei Phasen ein, wobei beide Phasen in drei Schritten ablaufen. Beide Kommunikationspartner durchlaufen diese beiden Phasen gleichzeitig. Für die erste Phase nehmen beide Kommunikationspartner die öffentlich bekannten Zahlen „p“ (Primzahl) und „g“ (Primitivwurzel) als Input.

Phase 1

Schritt 1 Erzeugen der privaten Geheimzahl

- Es wird eine natürliche Zahl „x“ gewählt, so dass $0 < x < p - 1$ gilt

Schritt 2 Potenzieren

- Es wird ein öffentlicher int Wert „y“ errechnet mit: $y = g^x \mod p$, $0 < y < p$

Schritt 3 Konvertierung von Integer-to-octet-string

- Der öffentliche int Wert „y“ wird in ein octet-string PV (Public Value) mit Länge „k“ gewandelt mit $y = \sum_{i=1}^{k} 2^{8(k-i)PV_i}$

Nach der ersten Phase tauschen die beiden Kommunikationspartner ihren öffentlichen Wert (PV) aus, welcher soeben errechnet wurde. Für die zweite Phase nehmen beide Kommunikationspartner den soeben erhaltenen Public Value PV' des Kommunikationspartners, wie auch die eigene private Geheimzahl als Input.

Phase 2

Schritt 1 Konvertierung von Octet-string-to-integer

- Der öffentliche octet-string PV' des Kommunikationspartners wird umgewandelt in ein int-Wert „y“ mit: $y = \sum_{i=1}^{k} 2^{8(k-i)PV_i'}$

Schritt 2 Potenzieren

- Die errechnete Zahl „y“ wird nun mit der eigentlichen Geheimzahl „x“ potenziert und mod „p“ gerechnet: $Z = (y')^x \mod p$, $0 < z < p$.

Die daraus erzeugte Zahl „z“ ist nur der gemeinsame Geheimschlüssel

Schritt 3 Konvertierung von Integer-to-octet-string

- Der int-Wert „z“ wird nun noch in einen octet-string SK(Secret Key) umgewandelt: $z = \sum_{i=1}^{k} 2^{8(k-i)SK_i}$

Der in der zweiten Phase von beiden erzeugte Octet-string, auch bekannt als secret key (SK), kann nun von beiden für das Verschlüsseln von Nachrichten angewendet werden.

2.2.3 Vergleich zur Vorlesung Krypt

Die im Vorgehen mehrfach vollzogene Umwandlung von Integer in String und umgekehrt, wurde in unseren Vorlesungen nicht betrachtet. Dieser Schritt ist jedoch für die Theorie nicht zentral und lediglich wichtig für eine konkrete Implementierung in der Praxis von DH. Zudem werden in diesem PKCS die Domain Werte, also Primzahl p sowie eine Primitivwurzel g, nicht von den beiden Kommunikationsparteien sondern von einer zentralen Authorität gewählt.

2.3 PKCS #5

PKCS Dokument #5 gibt Empfehlungen für die Implementierung von Passwort basierter Kryptografie. Passwörter sind nicht direkt als Schlüssel verwendbar und müssen deshalb angepasst werden, um als solchen verwendet werden zu können. Bei dieser Umwandlung in einen Schlüssel muss auch beachtet werden, dass Passwörter meist Teil von einem kleineren Raum sind. Die Anzahl verschiedener Werte ist also viel kleiner als z.Bsp. bei richtigen Schlüssel.

Ein Ansatz zur Passwort basierten Kryptografie ist die Verwendung eines sogenannten Salt um einen Schlüssel zu produzieren. Der Salt ist ein zufälliger Wert welcher dem Passwort angehängt wird und somit verhindert, dass aus einem sogenannten Dictionary von bereits berechneten Hashwerten das Passwort ausgelesen werden kann.

Ein anderer Ansatz ist die Verwendung einer Funktion zur Ableitung eines Passworts, welche unterschiedlich oft ausgeführt wird (abhängig vom sogenannten Wiederholungszähler) Die in PKCS #5 beschriebenen Methoden implementieren beide Ansätze, dementsprechend ist die Passwort basierte Schlüssel Ableitung eine Funktion von einem Passwort, einem Salt und dem Wiederholungszähler.

In PKCS #5 wird die Verwendung dieser Ableitung in Zusammenhang mit Verschlüsselung und Message Authentication definiert obwohl auch andere Verwendungszwecke denkbar wären (z.Bsp. bei der Speicherung von Passwörtern).

2.3.1 Salt und Wiederholungszähler

Der Salt dient dazu für ein einzelnes Passwort eine grosse Menge von möglichen Keys generieren zu können. Bei einem 64 Bit langen Salt gibt es also für jedes Passwort 2^64 verschiedene Keys. Zudem ist es unwahrscheinlich, dass für das selbe Passwort zwei mal derselbe Key generiert wird.

Der Wiederholungszähler dient dazu die Erstellung des Keys rechenaufwändiger zu machen, und somit die Schwierigkeit einer Attacke zu erhöhen. Es wird eine Anzahl von mindestens 1000 Iterationen empfohlen.

Salt und Wiederholungszähler müssen nicht geheim sein!

2.3.2 Schlüsselableitungsfunktion PBKDF2

Für neuere Applikationen wird PBKDF2 empfohlen, weshalb hier die PBKDF1 ausser Acht gelassen wird. Grob zusammengefasst funktioniert PBKDF2 folgendermassen.

1. Der abgeleitete Schlüssel wird in einzelne Blöcke unterteilt
2. 2. Für jeden Block wird eine Pseudozufalls Funktion mehrere Male ausgeführt (Anzahl gemäss Wiederholungszähler) und mit dem vorherigen Resultat der Funktion 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.
3. 3. Am Schluss werden die einzelnen Blöcke zusammengesetzt und das ist dann der abgeleitete Schlüssel.

Als Pseudozufallsfunktionen kommen HMAC-SHA-1 oder HMAC-SHA-2 in Frage.

2.3.3 Verschlüsselungs Schema PBES2

Eine typische Anwendung von einem **P**assword **B**ased **E**ncryption **S**cheme (wie PBES2 [4]) ist der Schutz von Private Keys, also von Nachrichten welche Private Key Informationen enthalten. PBES2 kombiniert die Schlüsselableitungsfunktion PBKDF2 mit einem Verschlüsselungs-Schema (z.Bsp. AES-CBC-Pad)

Etwas vereinfacht werden bei Ver- und Entschlüsselung folgende Schritte durchgeführt:

Verschlüsselung

1. Wählen von Salt und Wiederholungszähler
2. Erstellen des Schlüssels mit Hilfe von PBKDF2
3. Verschlüsseln der Nachricht mit z.Bsp. AES-CBC-Pad

[4] Neben PBES2 existiert auch noch PBES1, welches jedoch nicht mehr verwendet werden soll.

Entschlüsselung

1. Erlangen von Salt und Wiederholungszähler
2. Erstellen des Schlüssels mit Hilfe von PBKDF2
3. Entschlüsseln der Nachricht mithilfe des abgeleiteten Schlüssel aus Schritt 2

Wie man sieht, wird mit dem gleichen Schlüssel ver- und entschlüsselt (schliesslich ist AES ja auch ein symmetrisches Verfahren).

2.3.4 Message Authentication Schema PBMAC1

Eine Message Authentication Schema besteht aus einer MAC Erzeugungs-Operation und einer MAC Verifikations-Operation. Dabei wird (anders als bei einer Signatur) die Erzeugung sowie die Verifikation des MACs (Message Authentication Code) mit dem selben Schlüssel vorgenommen.

PBMAC1 kombiniert die Schlüsselableitungsfunktion PBKDF2 mit einem Message Authentication Schema. Namentlich sind das HMAC-SHA-1 oder HMAC-SHA-2, welche beide auf den SHA-1 respektive SHA-224, SHA-256, SHA-384 oder SHA-512 Hash Funktionen basieren.

Die beiden Operationen werden hier etwas vereinfach beschrieben:

PBMAC1 Erzeugungsoperation

1. Wählen von Salt und Wiederholungszähler
2. Erstellen des Schlüssels mit Hilfe von PBKDF2
3. Erstellen des MAC mithilfe der erzeugten Schlüssels und dem zugrunde liegenden Message Authentication Schemas (z.B. HMAC-SHA-2)

PBMAC1 Verifikationsoperation

1. Erlangen von Salt und Wiederholungszähler
2. Erstellen des Schlüssels mit Hilfe von PBKDF2
3. Erstellen des MAC mithilfe der erzeugten Schlüssels und dem zugrunde liegenden Message Authentication Schemas (z.B. HMAC-SHA-2)
4. Vergleichen des eben erzeugten MACs und des zu verifizierenden MACs
5. Bei Übereinstimmung ist der Output „correct“, ansonsten „incorrect“

2.4 PKCS #6 Extended-Certificate Syntax Standard

2.4.1 Einleitung

Aktuelle Version des Standards	1.5
Datum	1. November 1993
Seitenzahl	11

Kurzbeschreibung RSA Laboratories

> *This standard describes syntax for extended certificates, consisting of a certificate and a set of attributes, collectively signed by the issuer of the certificate. The intended application of this standard is to extend the certification process beyond just the public key to certify other information about the given entity [PKCS6].*

2.4.2 Zusammenfassung

Was beschreibt der Standard? Das Dokument beschreibt einen Syntax für «extended certificates» (Erweiterte Zertifikate).

Ein erweitertes Zertifikat ist ein X.509 Public-Key-Zertifikat und eine Menge von Attributen, welche zusammen vom Aussteller des Zertifikats signiert werden.

Durch die Verwendung der zusätzlichen Attribute wird im Zertifizierungsprozess nicht mehr nur das Public-Key-Zertifikat, sondern weitere Informationen des Zertifikathalters verifiziert, z.B. E-Mail-Adressen.

Anwendungen des Standards Gemäss [PKCS6] v1.5 wird die hauptsächliche Anwendung des Standards in PKCS #7 beschrieben, wobei weitere Anwendungen erwartet werden [PKCS6, S.1].

Gründe für die Verwendung von erweiterten Zertifikaten Die Standarddokumentation nennt vier Gründe für die Verwendung von erweiterten Zertifikaten:

1. Änderungen an der X.509-Zertifikat-Syntax „are easily followed“
2. Das X.509-Zertifikat kann einfach aus dem erweiterten Zertifikat exportiert werden
3. Sowohl das erweiterte Zertifikat, wie auch das darin enthaltene X.509-Zertifikat können mit einer einzigen Public-Key-Operation verifiziert werden, da beide zusammen von der selben Ausgabestelle signiert wurden.
4. Wenig Redundanz zwischen X.509-Zertifikat und erweitertem Zertifikat, da die Informationen des X.509-Zertifikats bereits in dessen enthalten sind. Das erweiterte Zertifikat enthält nur zusätzliche Informationen in seinen Attributen.

Mögliche Attribute Einige möglicherweise sinnvolle Attribute werden in PKCS #9 definiert.

2.4.3 Syntax

Ein erweitertes Zertifikat besteht aus drei Teilen:

1. Informationen zum erweiterten Zertifikat („extended-certificate information“)
2. ein „signature algorithm identifier“
3. eine digitale Signatur der Informationen zum erweiterten Zertifikat

Die Informationen zum erweiterten Zertifikat bestehen aus einem bereits vom Aussteller signierten X.509-Zertifikat und einer Menge von Attributen, welche weitere Informationen zur im X.509-Zertifikat identifizierten „entity“ enthalten.

Das erweiterte Zertifikat wie das X.509-Zertifikat werden von der selben Ausgabestelle (CA) signiert.

Erstellen eines erweiterten Zertifikats

1. Ein Wert für den Datentyp ExtendedCertificateInfo wird (von der Ausgabestelle) erstellt, enthaltend ein X.509-Zertifikat sowie die Menge der zusätzlichen Attribute.
2. Der Wert von ExtendedCertificateInfo wird mit dem privaten Schlüssel der Ausgabestelle signiert.
3. Die oben genannten drei Bestandteile werden in den Datentyp Extended-Certificate zusammengeführt.

Datentypen

ExtendedCertificateInfo

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

Attribut	Beschreibung
version	Versionsnummer des Standards, aktuell 0
certificate	Ein X.509-Zertifikat
attributes	Eine Menge möglicher Attribute mit weiteren Informationen

Tabelle 5: Attribute von ExtendedCertificateInfo

ExtendedCertificate

```
ExtendedCertificate ::= SEQUENCE {
    extendedCertificateInfo ExtendedCertificateInfo,
    signatureAlgorithm SignatureAlgorithmIdentifier,
    signature Signature
}
SignatureAlgorithmIdentifier ::= AlgorithmIdentifier
Signature ::= BIT STRING
```

Attribut	Beschreibung
extendedCertificateInfo	Die eigentliche „extended-certificate information“, welche signiert wird.
signatureAlgorithm	Der Signierungsalgorithmus, mit dem die Informationen signiert werden. Bsp.: md2WithRSAEncryption, md5WithRSAEncryption (siehe PKCS #1)
signature	Die resultierende Signatur der Informationen

Tabelle 6: Attribute von ExtendedCertificate

2.5 PKCS #7 - Standard zur Syntax kryptographischer Nachrichten

PKCS #7 definiert eine generelle Syntax zur Beschreibung von Inhalten welche in Verbindung mit kryptographischen Verfahren stehen können, zum Beispiel Signaturen. Die Syntax erlaubt Rekursion, so dass beispielsweise Inhalte signiert werden können, welche zuvor von einer anderen Instanz signiert wurden etc.

Folgendes sind Beispiele von Anwendungen, welche dieser Standard adressiert:

- Signieren von digitalen Nachrichten
- Digest (hash) von digitalen Nachrichten
- Authentisierung von Nachrichten (MAC)
- Verschlüsselung digitaler Inhalte

2.5.1 Aufbau

Es werden generell zwischen zwei Klassen von Inhalts-Typen unterschieden:

base data enthält „plain data“, also Daten, welche keine kryptographischen Erweiterungen („enhancements“) aufweisen.

enhanced data enthält Inhalt eines bestimmten Typs (evtl. verschlüsselt) und weitere kryptographische Erweiterungen.

Insgesamt werden vom Standard sechs Inhalts-Typen definiert, weitere könnten in der Zukunft hinzu kommen:

- data
- signed data
- enveloped data
- signed-and-enveloped data
- digested data
- encrypted data

Nur „data“ gehört zur Klasse „base“, alle weiteren Typen gehören zur Kategorie „enhanced“. Da die Typen der enhanced Klasse selbst Inhalte anderer Typen beinhalten, wird auch von dem sogenannten „inner“ und „outer“ content gesprochen. Der „inner“ content ist somit der vom „outer“ content erweiterte Inhalt.

2.5.2 Der Inhalt

Der Standard exportiert schliesslich einen Typen `ContentInfo`, welcher den entsprechenden Inhalt zu repräsentieren vermag. Eine Nachricht, ein Element resp. ein Objekt dieses Standards weist die folgende Syntax auf:

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

contentType benennt den Typ des Inhaltes. Es handelt sich um einen Object identifier, welcher den Inhalts-Typen gemäss obiger Liste 2.5.1 (data, signed data etc.) enthält. Er weist die folgende Syntax auf:

```
ContentType ::= OBJECT IDENTIFIER
```

content ist der Inhalt der Nachricht. Das Feld ist optional und falls es nicht enthalten ist, muss der gewünschte Inhalt anderweitig zur Verfügung gestellt werden (wird mittels `contentType` kommuniziert).

2.5.3 Die Inhalts-Typen

Data Der data content type ist ein arbiträrer octet string, welcher generell keine interne Struktur aufweist (jedoch aufweisen kann) und in Form einer ASCII Zeichenketten betrachtet wird. Die genaue Interpretation des Inhaltes wird der Anwendung überlassen.

Signed-data Der signed-data content type besteht aus dem signierten Inhalt eines beliebigen Typs sowie verschlüsselter Digests des Inhaltes, welche von einer beliebigen Anzahl Instanzen signiert wurden.

Die Syntax des signet-data content type ist folgendermassen gegeben [PKCS7, S.10]:

```
SignedData ::= SEQUENCE {
    version Version,
    digestAlgorithms DigestAlgorithmIdentifiers,
    contentInfo ContentInfo,
    certificates [0] IMPLICIT ExtendedCertificatesAndCertificates
        OPTIONAL,
    crls [1] IMPLICIT CertificateRevocationLists OPTIONAL,
    signerInfos SignerInfos
}
```

```
DigestAlgorithmIdentifiers ::= SET OF DigestAlgorithmIdentifier
SignerInfos ::= SET OF SignerInfo
```

Der Prozess, wie signierter Inhalt erzeugt wird, wird folgendermassen festgehalten [PKCS7, S.10]:

1. Pro jede signierende Instanz wird der Message Digest des Inhaltes gemäss dem Algorithmus der signierenden Instanz erstellt.
2. Pro signierende Instanz wird mit dessen privatem Schlüssel jeder Message Digest dieser Instanz und dessen zugehörigen Angaben verschlüsselt.
3. Pro signierende Instanz werden der verschlüsselte Message Digest und andere Instanz-spezifische Informationen in einem `SignerInfo` Objekt abgelegt. Die Zertifikate und certificate-revocation Listen werden in diesem Schritt ermittelt.
4. Sämtliche Message-Digest Algorithmen und `SignerInfo` Objekte für alle signierenden Instanzen werden mit dem Inhalt im `SignedDataValue` Objekt abgelegt.

Enveloped Der enveloped-data content type beinhaltet verschlüsselte Daten sowie die verschlüsselten Schlüssel für eine beliebige Anzahl Empfänger, womit der Inhalt wieder entschlüsselt werden kann. Die Idee ist ein sogenanntes „Digital Envelope“, dadurch können die Vorteile des Public Key Algorithmus mit den Vorzügen der symmetrischen Verschlüsselung genutzt werden (hybrid).

Das Element enthält neben einer Version und den verschlüsselten Inhalten eine Menge von Empfänger Angaben. Darin werden der verschlüsselte Schlüssel sowie die Angaben über den zugrundeliegenden Algorithmus hinterlegt:

```
EnvelopedData ::= SEQUENCE {
    version Version,
    recipientInfos RecipientInfos,
    encryptedContentInfo EncryptedContentInfo
}
RecipientInfos ::= SET OF RecipientInfo
EncryptedContentInfo ::= SEQUENCE {
    contentType ContentType,
    contentEncryptionAlgorithm
    ContentEncryptionAlgorithmIdentifier,
    encryptedContent
    [0] IMPLICIT EncryptedContent OPTIONAL
}
EncryptedContent ::= OCTET STRING
```

Signed and enveloped Der signed and enveloped content type kombiniert quasi die signed und die enveloped Typen. Im Besonderen wird der genaue Prozess zur Erstellung des entsprechenden Inhaltes genannt [PKCS7, S.22] und der genaue Aufbau des Typs beschrieben [PKCS7, S.23].

Digested Der digested content type beinhaltet einen Inhalt beliebigen Typs sowie einen Message Digest dazu. Dies dient grundsätzlich dazu, die Integrität des Inhaltes zu gewährleisten. Dieser Inhalt wird daher typischerweise in einen enveloped content type integriert.

Encrypted Der encrypted content type Inhalt enthält selbst keine Empfänger oder Schlüssel. Der Typ ist eher dazu gedacht, für lokale Verschlüsselung verwendet zu werden. Der Typ beinhaltet lediglich eine Version und den verschlüsselten Inhalt wie er Bestandteil des enveloped content type ist (Syntax 2.5.3, `EncryptedContentInfo`).

2.6 PKCS #8

Der Standard in PKCS #8 beschreibt die Syntax um Private Key Informationen zu speichern. Dies kann sowohl unverschlüsselt als auch verschlüsselt geschehen (z.Bsp. mit den in PKCS #5 beschriebenen Algorithmen). Wie in allen PKCS Dokumenten wird zur Beschreibung der Syntax die ASN.1 [5] verwendet.

2.6.1 Private Key Information Syntax

Die Syntax für den ASN.1 Typ PrivateKeyInfo:

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

Bedeutung der Felder:

version Syntax Versionsnummer dient der Kompatibilität von zukünftigen Überarbeitungen dieses Standards. Für diese Version des Standards ist sie 0.

privateKeyAlgorithm Bezeichnet der verwendete Algorithmus. z.Bsp. rsaEncryption aus PKCS #1

privateKey Dies ist der Private Key

attributes Eine Menge von Attributen, wie sie z.Bsp. in PKCS #9 definiert sind.

[5] Abstract Syntax Notation One: http://www.itu.int/ITU-T/asn1/

2.6.2 Verschlüsselte Private Key Information Syntax

Die Syntax für den ASN.1 Typ EncryptedPrivateKeyInfo:

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

Bedeutung der Felder

encryptionAlgorithm Bezeichnet den verwendeten Algorithmus um die Private Key Information zu verschlüsseln (Bsp. aus PKCS #5: pbeWithMD5AndDES-CBC)

encryptedData Die verschlüsselte Private Key Information (PrivateKeyInfo aus Abschnitt 2.6.2)

2.7 PKCS #9 - Ausgewählte Objektklassen und Attribute

In PKCS#9 werden einige Objektklassen und Attribute behandelt, welche Bestandteile anderer PKCS Dokumente sind und von unterschiedlichen Dokumenten gleichermassen referenziert werden.

Der Standard führt die folgenden zwei neuen Objekt-Klassen und deren zugehörigen Attribute ein:

- `pkcsEntity`
- `naturalPerson`

2.7.1 pkcsEntity

Die `pcksEntity` Objekt-Klasse ist dazu gedacht, Attribute von beliebigen PKCS Entitäten zu beherbergen. Sie wurde für die Verwendung von LDAP basierten Verzeichnis-Diensten entwickelt.

Die Syntax ist folgendermassen gegeben:

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

Die Klasse sieht eine ID sowie ein optionales Attribut vor. Die folgenden Attribute werden allesamt dafür verwendet, die zugehörigen Informationen in einem Verzeichnis-Dienst abzulegen.

pKCS7PDU Die in PKCS#7 definierten geschützten Daten (enveloped, signed etc.) werden mit diesem Attribut verwendet.

userPKCS12 In PKCS#12 wird ein Format für den Austausch von Angaben über die persönliche Identität definiert. Dieses Attribut kann hierfür verwendet werden.

pKCS15Token In PKCS#15 wird ein Format für kryptographische Tokens definiert, welche in diesem Attribut abgelegt werden können.

encryptedPrivateKeyInfo PCKS#8 definiert ein Format für verschlüsselte private Schlüssel, welche in diesem Attribut enthalten sein können.

2.7.2 naturalPerson

Die Objekt-Klasse `naturalPerson` wurde wie die Klasse `pkcsEntity` für die Verwendung in Verzeichnis-Diensten erstellt. `naturalPerson` ist dafür gedacht, Attribute von natürlichen Personen (Menschen) zu beherbergen.

Die Syntax und der Aufbau ähneln stark derjenigen der `pkcsEntity`:

```
naturalPerson OBJECT-CLASS ::= {
    SUBCLASS OF { top }
    KIND auxilary
    MAY CONTAIN { NaturalPersonAttributeSet }
    ID pkcs-9-oc-naturalPerson
}
```

Für `naturalPerson` sind die folgenden Attribute definiert:

emailAddress spezifiziert eine oder mehrere E-Mail Adressen in Form einer unstrukturierten ASCII Zeichenkette. Es liegt an der Anwendung, die Adressen zu interpretieren. Speziell an diesem Element ist die `EQUALITY MATCHING RULE` als `pkcs9CaseIgnoreMatch`, welche besagt, dass falls zwei E-Mail Adressen miteinander verglichen werden, wird die Gross-Kleinschreibung ignoriert.

unstructuredName spezifiziert den oder die Namen eines Subjektes als unstrukturierte ASCII Zeichenkette. Ein unstrukturierter Name kann mehrere Attribut-Werte enthalten, es liegt auch hier an der Anwendung, den Namen zu interpretieren. Wie bei der E-Mail Adresse wird beim Vergleich zweier unstrukturierter Adressen die Gross-Kleinschreibung ignoriert.

unstructuredAddress nennt die Adresse des Subjektes. Auch hierbei handelt es sich um vollkommen unstrukturierten Inhalt. Auch hier gilt, dass beim Vergleich die Gross-Kleinschreibung ignoriert wird. Wie der Name kann auch die Adresse mehrere Attribut-Werte enthalten und es liegt an der Anwendung, diesen Inhalt zu interpretieren.

dateOfBirth wird in Form der `GeneralizedTime` geführt. Es wird ferner verlangt, dass dieses Attribut maximal einmal vorkommt (`SINGLE VALUE TRUE`).

placeOfBirth darf wie `dateOfBirth` maximal einmal vorkommen und nennt in einem unstrukturierten Text den Geburtsort des Subjektes. Der Geburtsort wird allerdings unter Berücksichtigung der Gross-Kleinschreibung behandelt.

gender nennt das Geschlecht mittels „F“, „f“, „M“ oder „m“. Dieses Attribut darf ebenfalls maximal einmal vorkommen.

countryOfCitizenship zählt alle Staatsangehörigkeiten des Subjektes auf, folglich darf das Attribut mehrmals vorkommen. Das Land wird als 2-stelliger Länder-Code gemäss ISO/IEC 3166-1 („CH“ für Schweiz, „DE“ für Deutschland etc.) hinterlegt.

countryOfResidence nennt alle Länder der Aufenthaltsorte des Subjektes, kann also mehrfach vorkommen. Das Land wird ebenfalls als zweistelliger ISO Code hinterlegt.

pseudonym spezifiziert ein Pseudonym des Subjektes. Neben einer ID enthält es das Pseudonym als Zeichenkette, welches unter Berücksichtigung der Gross-Kleinschreibung behandelt wird.

serialNumber Auf dieses Attribut wird nicht eingegangen, es ist definiert in ISO/IEC 9594-6.

2.7.3 Generelle Attribute

Neben der beiden neu definierten Objekt-Klassen `pkcsEntity` und `naturalPerson` wird im Besonderen auf einige spezifische Attribute eingegangen. Von diesen sollen nun einige genauer betrachtet werden:

contentType Das Attribut `contentType` spezifiziert den Inhalts-Typen des in PKCS#7 (oder S/MIME) signierten `ContentInfo` Objektes. In solchen Inhalten ist das Attribut `contentType` zwingend, falls authentifizierte Attribute aus PKCS#7 vorhanden sind.

messageDigest spezifiziert den Message Digest der Inhalte des `content` Feldes des `ContentInfo` Objektes. Der Message Digest wird anhand dem Algorithmus der signierenden Instanz berechnet. Dieses Attribut ist zwingend, falls authentifizierte Attribute aus PKCS#7 Verwendung finden.

signingTime benennt die Zeit, wann die Signatur erstellt wurde. Die Zeit wird gemäss ISO/IEC 9594-8 notiert, wobei Daten vor dem 1.1.1950 oder nach dem 31.12.2049 müssen in Form der `GeneralizedTime` codiert werden, alle anderen Zeiten als `UTCTime` [PKCS9, S.12].

randomNonce ist ein Attribut, welches es ermöglicht, gegen spezifische Attacken zu schützen. Beispielsweise kann ein Unterzeichner `signingTime` unterdrücken, um replay Attacken zu unterbinden. Das Attribut dient signierten Daten aus PKCS#7 und darf nur einmal vorkommen. Das Element `RandomNonce` ist ein Oktett-String und muss mind. 4 Bytes lang sein.

counterSignature erlaubt es, eine Signatur zu signieren. Das Attribut hat dieselbe Bedeutung wie `SignerInfo` [PKCS7, S.12], ausser:

- Das Feld `authenticatedAttributes` muss ein Attribut `messageDigest` aufweisen, falls es irgendwelche andere Attribute aufweist.

- Der Inhalt des Message Digest ist der Inhalt des `signatureValue` Feldes des `SignerInfo` Objektes. Das bedeutet, dass der signierende Prozess (welcher die Signatur signiert) den originalen Inhalt nicht zu kennen braucht. Zudem kann eine `counterSignature` selbst wieder eine `counterSignature` beinhalten, so lassen sich beliebig lange Ketten von `counterSignature` Objekten erstellen.

challengePassword spezifiziert ein Passwort, mit welchem eine Entität die Annullierung eines Zertifikates verlangen kann. Die Interpretation des Inhaltes ist wiederum der Anwendung überlassen, er wird jedoch unter Berücksichtigung von Gross-Kleinschreibung verglichen. Es wird bemerkt, dass der Inhalt als `PrintableString` encodiert werden soll, falls Internationalisierung dies nicht ermöglicht, sollte stattdessen `UTF8String` verwendet werden.

2.8 PKCS #13 - Elliptische Kurven

Die Kryptographie mit elliptischen Kurven erfährt eine zunehmende Popularität, da eine vergleichbare Sicherheit zu etablierten Public Key Verfahren mit kleineren Schlüsseln ermöglicht wird. Verbesserungen in der Implementierung wie der Erzeugung von elliptischen Kurven machen das Verfahren praxistauglicher als bei seiner Einführung in den 1980-er Jahren.

PCKS#13 ist bis heute noch kein definitiver Standard. Der Standard ist noch immer in Entwicklung. Der Standard soll die folgenden Aspekte der Kryptographie mit elliptischen Kurven abdecken [PKCS13-proj]:

- Parameter und Schlüssel Erzeugung und Validierung
- Digitale Signaturen
- Public Key Verschlüsselung
- Key Agreement (anstelle des verwundbaren anonymen Key-Exchanges wie z.Bsp. Diffie-Hellman)
- ASN.1 Syntax
- Überlegungen zur Sicherheit

Es sind bereits Standards in Arbeit, welche sich mit der Kryptographie mit elliptischen Kurven befassen:

ANSI X9.62 Ist ein in Entwicklung befindlicher Standard für digitale Signaturen.

ANSI X9.63 Ist ein in Entwicklung befindlicher Standard für Key Agreement.

IEEE P1363 Soll eine generelle Referenz für Public Key Verfahren verschiedener Techniken, inkl. elliptischer Kurven werden.

PKCS#13 soll die anderen Standards vervollständigen, ein Profil der anderen Standards im PKCS Format liefern und eine Anleitung für die Integration in andere PKCS Anwendungen (wie beispielsweise PKCS#7) bieten.

2.8.1 Aktueller Umfang

Zurzeit umfasst der Standard Grundlagen der folgenden Bereiche, welche allerdings keine konkreten Implementierungen oder Standards nennt. Diese Themen können als Grundlagen der Kryptographie mit elliptischen Kurven betrachtet werden.

Functions Grundlegende Definition einer Funktion: „A function f from a set A to a set B assigns to each element a in A a unique element b in B.“ [PKCS13-func]

Modular arithmetic Grundlagen der modularen Arithmetik

Groups Grundlagen der diskreten Mathematik ($\mathbb{Z}_p$ und $\mathbb{Z}_p^*$) in Bezug auf Gruppen.

Fields and rings Ebenfalls mathematische Grundlagen.

Vector spaces and lattices Behandelt die Grundlagen der linearen Algebra und der Vektorräume.

Boolean expressions Die Betrachtung logischer Ausdrücke als Funktionen.

Time estimations and some complexitiy Betrachtungen zur Komplexität.

Generell sind keine konkreten Vorgaben zu diesem Standard vorhanden. Zurzeit existiert lediglich ein Vorschlag, welcher mögliche Key Agreement Schemes, Signature Schemes, Encryption Schemes und Point Representations nennt.

A Quellen

[RSA-Lab] RSA Laboratories: http://www.emc.com/domains/rsa/index.htm, 1.1.2014

[KAL91] Kalinski & Burton S.: An Overview of the PKCS Standards, 1991

[PKCS-Standards] RSA Laboratories: PUBLIC-KEY CRYPTOGRAPHY STANDARDS, http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/public-key-cryptography-standards.htm, 1.1.2014

[PKCS1] RSA Laboratories: PKCS#1, ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf

[PKCS6] RSA Laboratories: PKCS#6, http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs-6-extended-certificate-syntax-standard.htm, 1.1.2014

[PKCS7] RSA Laboratories: PKCS#7 Cryptographic Message Syntax Standard; 1993

[PKCS9] RSA Laboratories: PKCS #9 v2.0: Selected Object Classes and Attribute Types; 2000

[PKCS13] RSA Laboratories: PKCS #13: ELLIPTIC CURVE CRYPTOGRAPHY STANDARD http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs-13-elliptic-curve-cryptography-standard.htm

[PKCS13-proj] RSA Laboratories: PKCS#13 Project overview, http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/project-overview.htm, 1.1.2013

[PKCS13-func] RSA Laboratories: A.1 Functions, http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/functions.htm, 1.1.2013