



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Diplomarbeit

Fachgebiet: Theoretische Informatik - Kryptographie und Computeralgebra

Entwurf und Implementierung von SCVP in Java

Simple Certificate Validation Protocol

Arbeit von: **Mohamed Belhilali**

Gutachter: Prof. Dr. Johannes Buchmann

Betreuer: Dr. Vangelis Karatsiolis

*Mein Vater hat mir Authentifizierungsprozesse schon als Kind beigebracht...
Damit ich mir bei einigen Händlern in unserem Viertel auch in seiner Abwesenheit
bestimmte Waren holen konnte, musste ich mich und damit auch meine Anfrage an den
Händler „authentifizieren“. Dies geschah, indem ich dem Händler ein a priori zwischen
ihm und meinem Vater vereinbartes „Passwort“ nannte. Das Passwort wurde in
sporadischen Zeitabständen gewechselt, sodass wir „Kinder“ selten die Möglichkeit hatten,
eine Ware zu holen, von der mein Vater nicht gewusst hat... Eine besondere Freude kam
mir auf, als ich meinem Vater Authentifizierungsprozesse in der IT erzählt habe und er mir
damit geantwortet hat, dass er sich solche Techniken von seinem Vater eignet hat..*

*Im Namen Allahs, des Allerbarmers, des Barmherzigen!
Lies im Namen deines Herrn, Der erschuf. (1) Er erschuf den Menschen aus einem
Blutklumpen. (2) Lies; denn dein Herr ist Allgütig, (3) Der mit dem Schreibrohr lehrt, (4)
lehrt den Menschen, was er nicht wusste... Koran (Sure 96)*

Ehrenwörtliche Versicherung

Ich versichere hiermit, daß ich die vorliegende Arbeit selbstständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Stellen, die anderen Werken entnommen wurden, habe ich in jedem einzelnen Fall durch die Angabe der Quelle als solche kenntlich gemacht.

Diese Diplomarbeit ist noch nicht veröffentlicht worden. Sie ist somit weder anderen Interessenten zugänglich gemacht, noch einer anderen Prüfungsbehörde vorgelegt worden.

Darmstadt, 01. Februar. 2010

Mohamed Belhilali

Danksagung

Natürlich gilt mein aller größter Dank dem Schöpfer der Welten...

Diese Diplomarbeit entstand am Fachgebiet: Theoretische Informatik - Kryptographie und Computeralgebra der technischen Universität Darmstadt unter der Leitung von Herrn Prof. Johannes Buchmann. Besonders möchte ich mich bei Herrn Buchmann für die Möglichkeit bedanken, in seiner Arbeitsgruppe diese Diplomarbeit anzufertigen. Nicht zuletzt durch die freundliche und engagierte Betreuung hat mir diese Arbeit viel Freude bereitet.

Außerdem gilt mein Dank Dr. Vangelis Karatsiolis, der mir immer für Rat und Tat zur Seite stand, für die großartige Unterstützung sowie für seine Geduld.

Meiner gesamten Familie, besonders meinen Eltern möchte ich einen besonderen Dank ausdrücken..

Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation und Ziel der Arbeit	1
1.2	Gliederung der Arbeit	3
2	Public Key Infrastruktur (PKI)	5
2.1	Kryptographische Methoden	5
2.1.1	Verschlüsselung	5
2.1.1.1	Symmetrische Verfahren - secret key cryptography:	6
2.1.1.2	Asymmetrische Verfahren - public key cryptography	7
2.1.1.3	Hybride Verschlüsselung	7
2.1.2	Hashfunktionen	8
2.1.3	Digitale Signaturen	9
2.2	Zertifikate in Public Key Infrastruktur	10
2.2.1	Certification Authority	10
2.2.2	Zertifikate	11
2.2.2.1	X.509-Zertifikat	12
2.2.3	Zertifikat Überprüfung	14
2.2.3.1	Sperrlisten	14
2.2.3.2	Gültigkeitsmodelle	15
2.3	Online Certificate Status Protocol: Online-Sperrprüfung	18
3	Simple Certificate Validation Protocol	19
3.1	ASN.1	19
3.2	SCVP	22
3.2.1	Validation Policies	24
3.2.2	Validation Algorithmus	25
3.2.3	Validation Requirements:	26

3.2.4	Validation Request	26
3.2.5	Server Policy Request	34
3.2.6	Validation Response	35
4	Entwurf und Implementierung in Java	39
4.1	Webserver	40
4.2	Die Datenbank-Komponente	41
4.2.1	DB und DB-Schema	41
4.2.2	Die Datenbankverbindung	43
4.3	SCVP-ASN1-Modul: implementierte Klassen	47
4.3.1	BouncyCastle	47
4.3.2	Die Basis-Klasse ASN1Encodable:	48
4.3.3	SCVP-ASN1-Modulklassen:	49
4.3.3.1	Beispielklasse „Query“	50
4.3.3.2	Beispiel einer CVRequest	54
4.4	Bearbeitungslogik für ankommende Anfragen	56
5	Nachwort	64
6	Literaturverzeichnis	65

Abbildungsverzeichnis

2.1	Symmetrische Ver- und Entschlüsselung	6
2.2	Asymmetrische Ver- und Entschlüsselung	7
2.3	Entstehung einer digitalen Signatur	9
2.4	Prüfung einer digitalen Signatur	10
2.5	Das Schallenmodell	16
2.6	Das Kettenmodell	17
3.1	Datenaustausch auf ASN.1-Basis	19
3.2	Fragment eines X.509v3-Zertifikats als ASN.1 Struktur	22
3.3	Signed Data gekapselter CVRequest	28
4.1	Komponenten des SCVP-Prototyps	39
4.2	Die Klasse zur Verbindung mit der Derby Datenbank	43
4.3	UML und ASN1 Struktur der Klasse „Query“	50
4.4	CVRequest und einige Abhängigkeiten (UML)	54

1 Einleitung

1.1 Motivation und Ziel der Arbeit

Informationstechnologie (IT) und Telekommunikation (TK) haben in den letzten Jahrzehnten immer an Wichtigkeit gewonnen, so dass das moderne Leben ohne sie kaum vorstellbar ist.

Die Existenz moderner Großunternehmen wird bedroht, sobald ihre IT-Systeme länger als einen bis zwei Tage nicht funktionstüchtig sind, oder nicht wie erwartet funktionieren. Die Toleranzfrist bei kleinen und mittelständischen Unternehmen beträgt ca. 7 Tage¹.

Die Unversehrtheit der Daten und die Erhaltung der Funktionalität gemäß Spezifikation während und nach der Informationsverarbeitung und der Geschäftsprozesse sind in diesem Zusammenhang wesentliche Faktoren der Unternehmensplanung geworden. Hier gilt es, Risiken und Bedrohungen zu definieren, Schäden zu reduzieren und Schwachstellen zu minimieren oder gar zu beseitigen.

Des Weiteren nahm die internationale Verflechtung in nahezu allen Gesellschaftsbereichen (Wirtschaft, Politik, Kultur, Umwelt, Kommunikation etc.) in einem sehr großen Maße zu. Diese Intensivierung der globalen Beziehungen geschieht auf der Ebene von Individuen, Gesellschaften, Institutionen und Staaten und hat daher die Sicherheit in der Informationstechnologie zum Gegenstand vieler Normen, Gesetzen und Richtlinien werden lassen. IT-Sicherheit hat sich innerhalb der letzten 30 Jahre zu einem der wichtigsten Forschungsgebiete der Informatik entwickelt.

Die klassische Grundlage der IT-Sicherheit in diesem Sinne ist durch Sicherheitsziele bestimmt, die es zu „garantieren“ gilt. Die folgende Auflistung zeigt die Ziele davon, die von dieser Arbeit unmittelbar betroffen sind:

- Authentizität: entspricht der Sicherstellung der Identität eines Subjektes. Hier ist ein

¹<http://www.t-systems-zert.de/>

Subjekt als ein Benutzer, ein Prozess, ein System, eine Maschine oder gar eine Information zu verstehen. Authentizität ist die Voraussetzung für Verbindlichkeit.

- Vertraulichkeit: umfasst den Schutz von übertragenen, gespeicherten und verarbeiteten Informationen gegen die unberechtigte Kenntnisnahme. Dies gilt auch für Daten, die als gefahrlos und sicherheitsunkritisch erscheinen, jedoch dazu benutzt werden können, um Zugriff auf vertrauliche Informationen zu erhalten (z.B. Systemkonfiguration). In manchen Fällen kann das sogar bedeuten, dass selbst das Stattfinden einer Kommunikation vertraulich bleiben soll.
- Integrität: bedeutet die Sicherstellung der Korrektheit (Vollständigkeit, Unversehrtheit, und Richtigkeit) von Informationen. Systemintegrität ist die Sicherstellung der korrekten Funktionsweise eines Systems. Diese Forderungen schließen auch Dateiattribute, Sicherungskopien und Dokumentationen ein.

Andere Sicherheitsziele sind:

- Verfügbarkeit: Ist gewährleistet wenn zu jedem Zeitpunkt alle an dem Systembetrieb beteiligten Komponenten funktionsbereit und die zu verarbeitenden Daten in erwarteten Zustand und Qualität bereit stehen.
- Verbindlichkeit: Darunter versteht man die Unleugbarkeit einer getätigten Aktion.
- Datiertheit: In einigen Prozessen wird dieses Sicherheitsziel dafür gefordert, um den Zeitpunkt einer durchgeführten Aktion festzuhalten bzw. zu bestätigen. Zum Beispiel ist eine Antwort auf eine Ausschreibung fristgerecht einzureichen, und ist der Zeitpunkt der Antwort für eine spätere Prüfung nachzuweisen.
- Nicht-Vermehrbarkeit (Non-Propagation): Dieses Ziel wäre am ehesten dem Punkt Integrität zuzuordnen. Dabei dürfen Informationen von Unerlaubten bzw. Unberechtigten nicht kopiert werden können, z.B. bei „Replay-Angriffen“ unbemerkt wiederholt werden können.
- Anonymität (Anonymity): bedeutet den Schutz gegen Identifizierung. Kann durch bestimmte Verfahren (blinde Signaturen) gelöst werden, wie sie z.B. bei elektronischen Wahlen eingesetzt werden.

Wie oben erwähnt worden ist, stellen Vertraulichkeit, Integrität und Authentizität Sicherheitsziele dar, die von dieser Arbeit unmittelbar betroffen sind. Dies ist der Fall, da bei dieser Arbeit das Simple Certificate Validation Protocol analysiert und dafür prototypisch Konzeption bzw.

Entwurf einer Implementierung erstellt wird.

Das SCVP ist ein Internet-Protokoll, mit dessen Hilfe die Gültigkeitsüberprüfung von X.509v3-Zertifikaten und Zertifikatsketten zentralisiert werden kann. Die Gültigkeit von X.509v3-Zertifikaten stellt die Authentizität von Öffentlichen Schlüsseln sicher, mit denen basierend auf kryptographischen Verfahren wiederum Vertraulichkeit und Integrität von Daten innerhalb einer Public Key Infrastruktur garantiert werden kann.

1.2 Gliederung der Arbeit

Diese Arbeit beschäftigt sich als erstes im analytischen Teil damit, wie PKI-Ansätze und -Ziele zu verstehen sind und die Rolle, die dabei das Simple Certificate Validation Protocol spielt. Hierbei wird zuerst auf den allgemeinen Einsatz Kryptographischer Methoden und digitaler Zertifikate, ins Besondere X.509v3, eingegangen. Daraufhin wird ein Überblick über Vertrauensmodelle, Gültigkeit von Zertifikaten und Zertifizierungsstellen verschafft.

Der analytische Teil wird mit einer Einführung in das Simple Certificate Validation Protocol abgeschlossen: Nachdem das Online Certificate Status Protocol kurz angesprochen wird, werden ASN.1 und seine Konstrukte, die als syntaktische Grundlagen für SCVP gelten, vorgestellt.

Die Möglichkeiten von Kryptographie, OpenSSL und ASN.1 alle aufzuzählen würde den Rahmen dieser Arbeit sprengen, daher sind nur die für das Verständnis des Eigenentwurfs relevanten Teile erklärt.

Im praktischen Teil dieser Diplomarbeit ist das Ziel ein Prototyp für einen SCVP-Server zur Validierung von Zertifikaten bzw. zur Zusammenstellung von Zertifizierungspfaden. Hierbei wurde die RFC-Spezifikation berücksichtigt und abgebildet, sodass das Prototyp sowohl vom Konzept als auch von der Implementierung her den Standard repräsentiert.

Es ist von einer PKI-Struktur in Form von mehreren Certification Authorities und mit deren Hilfe ausgeteilten X.509v3-Zertifikaten ausgegangen. Dies erfolgte unter Verwendung von OpenSSL.

Für die Datenhaltung ist eine Datenbank erstellt worden, die im Client-Server-Modus (Stand-Alone-Konfiguration) von der verwendenden Anwendungen (z.B SCVP Server) angesprochen

wird. Bei der zu entwickelnden Anwendung handelt es sich um eine Servlet-Webanwendung, der der Webcontainer Tomcat von der Apache Foundation zu Grunde liegt.

Zum Testen der Anwendung ist wieder von den RFC-SCVP-Anforderungen ausgegangen. Hierbei werden Pfadbildungs-, Validierungs-anfragen und weitere SCVP Anfragen an den Webserver geschickt und die Antwort von ihm entsprechend verarbeitet und als SCVP-Response zurückgeliefert.

2 Public Key Infrastruktur (PKI)

2.1 Kryptographische Methoden

Die PKI baut auf Techniken der Kryptographie die man in drei Methoden bzw. Strategien unterteilen kann:

- Verschlüsselung
- Hashfunktionen
- digitale Signaturen

Es wird im Folgenden überblickartig auf die drei Methoden näher eingegangen.

2.1.1 Verschlüsselung

Es handelt sich bei der Verschlüsselung um eine Art Kodierung bzw. Transformation, die gewisse Bedingungen erfüllen muss: Mit der Anwendung eines Verschlüsselungsverfahrens werden die im Klartext vorliegenden Daten so verändert, dass der Originaltext in einer Form oder Struktur gebracht wird, die keine Rückschlüsse auf die enthaltenen Informationen erlaubt.

Die Verschlüsselung muss so stark sein, dass der Aufwand für die unerlaubte Entschlüsselung möglichst hoch ausfällt oder sogar deren Unrealisierbarkeit erreicht wird. Dazu muss es immer gewährleistet werden, dass seitens befugter Subjekte immer und ohne Informationsverlust die Informationsrückgewinnung (Entschlüsselung) möglich ist.

Grundsätzlich lassen sich die Verschlüsselungsverfahren in drei Kategorien klassifizieren:

- Symmetrische Verfahren
- Asymmetrische Verfahren
- Hybride Verschlüsselung

2.1.1.1 Symmetrische Verfahren - secret key cryptography:

Diese Art der Verschlüsselung Abb. [2.1] greift sowohl zum Verschlüsseln als auch zum späteren Entschlüsseln auf den selben Schlüssel.

Die Sicherheit Symmetrischer Chiffrierverfahren beruht damit auf der Geheimhaltung des Schlüssels. Die Abbildung In dem Szenario wo eine Verschlüsselte Kommunikation stattfindet

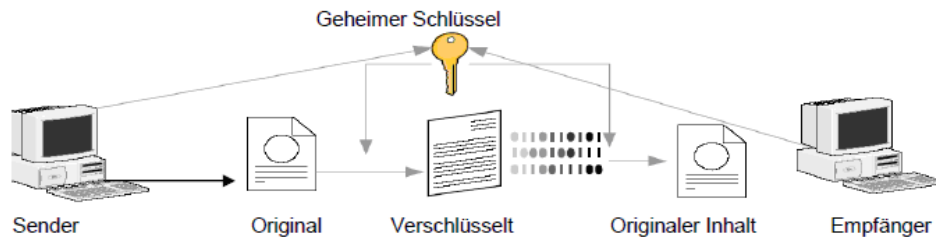


Abbildung 2.1: Symmetrische Ver- und Entschlüsselung

den soll, müssen die Kommunikationspartner für die Geheimhaltung der Schlüssel sorgen und ihn über gesicherten Weg austauschen.

Dies stellt den größten Nachteil dieses Typs der Kodierung: Zum einen muss der Schlüssel auf einen sicheren Weg dem Kommunikationspartner zugestellt werden, was einen entsprechenden Aufwand erfordert. Zum anderen wächst mit der Anzahl der Teilnehmer die Anzahl der zu verwaltenden Schlüssel. Dies wird in der Literatur als Schlüsselverteilungsproblem bezeichnet: da Theoretisch, für jeden Kommunikationspartner bzw. Kommunikationsverbindung zwischen zwei Subjekten ein eigener Schlüssel generiert und verwaltet sein muss. Formal ergibt das für n Teilnehmer jeweils $n \times (n - 1) \div 2$ Schlüsseln.

Einige der am meisten verbreiteten Symmetrischen Chiffrierverfahren sind:

- DES: (Data Encryption Standard) auch als Luzifer benannt
- Triple DES: eine weitere Verbesserung von DES, 3-mal langsamer, garantiert aber eine Steigerung der Sicherheit um einige Größenordnungen
- AES: (Advanced Encryption Standard) oder Rijndael. Ein Verschlüsselungsstandard, das als Nachfolger von DES gilt

2.1.1.2 Asymmetrische Verfahren - public key cryptography

Hier kommen zum Ver- und Entschlüsseln verschiedene Schlüssel zum Einsatz. Jeder Benutzer besitzt einen geheim gehaltenen vertraulichen Schlüssel (Private Key) und einen den Kommunikationspartnern bekannten Schlüssel (Public Key). Aus dem öffentlichen darf kein Rückschluss auf Art und Inhalt des geheimen Schlüssels möglich sein. Wendet man beide Schlüssel nacheinander (in beliebiger Reihenfolge) auf eine Nachricht an, so erhält man die ursprüngliche Nachricht.

Sollte eine Kommunikation mit asymmetrischer Verschlüsselung gesichert werden, dann chiffriert der Sender die zu sendenden Daten unter Einsatz vom öffentlichen Schlüssel des Empfängers. Dieser entschlüsselt dann die empfangenen Daten durch Anwendung seines geheimen Schlüssels und des selben Verfahrens (Funktion), wie der Sender. Abbildung [2.2] veranschaulicht die Schritte solch einer asymmetrischen Verschlüsselung/entschlüsselung.

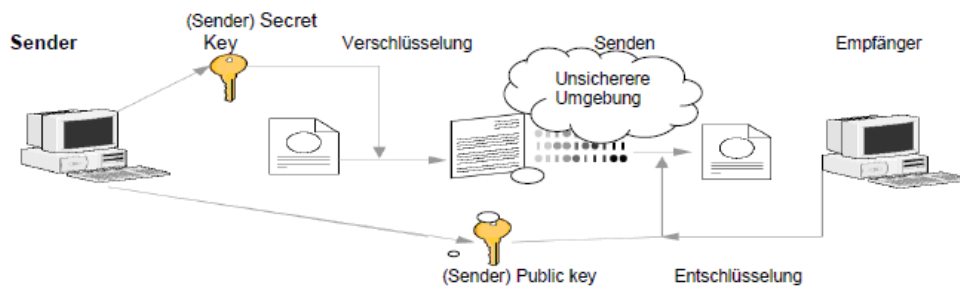


Abbildung 2.2: Asymmetrische Ver- und Entschlüsselung

Wichtige Vertreter bzw. Realisierungen dieser Technik sind:

- RSA
- ElGamal

2.1.1.3 Hybride Verschlüsselung

Die Funktionsweise dieses Verfahrens geht nach folgendem Schema; es wird der Klartext mit einem symmetrischen Schlüssel (Session Key) verschlüsselt. Danach wird dieser Schlüssel selbst asymmetrisch mit dem öffentlichen Schlüssel des Senders verschlüsselt. Anschließend

werden die Ergebnisse beider Verschlüsselungen gesendet.

Bei dem Empfänger angekommen wird der Session Key mit Hilfe des geheimen Schlüssels des Empfängers entschlüsselt. Nach diesem Schritt kann die eigentliche Nachricht entschlüsselt werden.

Das resultierende Verfahren weist folgende Eigenschaften auf:

- die verschlüsselten Daten sind kompakt
- die Lösung ist sicher solange die verwendeten Algorithmen sicher sind
- das Verfahren ist skalierbar
- eine vorherige Kontaktaufnahme ist nicht nötig
- die Verschlüsselung ist schnell
- es gibt keine unsichere Übertragung von Schlüsseln

2.1.2 Hashfunktionen

Hashfunktionen bilden große Datenmengen auf vergleichsweise kleinere Datenbestände einer festen vorgegebenen Länge ab.

Hashfunktionen sind Einwegfunktionen, d.h. sie sind im Sinne der Komplexitätstheorie schwer umzukehren.

Hashfunktionen sind Kollisionsfrei. Eine Hashfunktion f ist *Kollision frei*, wenn für alle (x,y) Paare aus dem Urbildbereich gilt:

$$x \neq y \Leftrightarrow f(x) \neq f(y)$$

Der Hashwert $f(x)$ einer Nachricht x ist eine für diese Nachricht eine charakteristische Prüfsumme, daher auch als Fingerabdruck bezeichnet. Hashfunktionen werden im Zusammenhang mit der Erstellung elektronischer Signaturen verwendet. Anstatt eine lange Nachricht zu signieren, wird der im Allgemeinen viel kürzere Hashwert der Nachricht signiert. Was eine

effiziente Erhöhung der Performance bedeutet.

2.1.3 Digitale Signaturen

Analog zu den Unterschriften auf dem Papier, sind digitale Unterschriften bzw. Signaturen als Identitätsmerkmal zu verstehen. Mit dem Einsatz digitaler Signaturen will man erreichen dass, das Unterschieben gefälschter Dokumente durch dritte Instanzen, z.B. Notardienst, prüfbar und ggf. feststellbar wird.

Wie Abbildung [2.3] zeigt, wird zuerst mit einem Hashalgorithmus eine Prüfsumme über das zu signierende Dokument erzeugt. Diese Prüfsumme gilt als Fingerabdruck des Dokumentes bzw. der Nachricht, d.h. wenn sich nur ein Bit in der Quellnachricht ändert, macht sich diese Änderung auch auf den Hashwert bemerkbar. Als nächster Schritt wird die Prüfsumme mit dem privaten Schlüssel des Signierers verschlüsselt und an die eigentliche Nachricht angehängt. Somit ist die Signatur abgeschlossen.

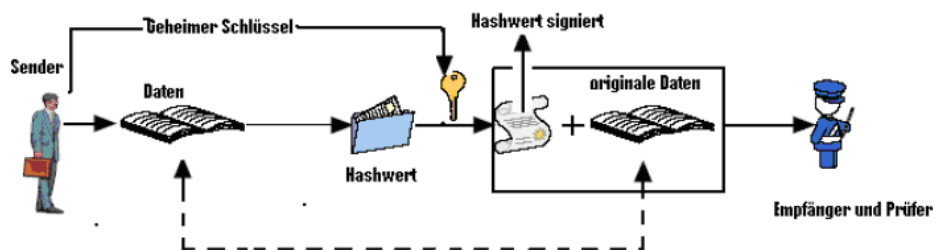


Abbildung 2.3: Entstehung einer digitalen Signatur

Um das Signierte Dokument zu prüfen Abb. [2.4], wird sein Nutzteil (eigentliche Nachricht) von der Signatur getrennt. Dann wird eine aktuelle Prüfsumme von dem Dokument mit demselben Hashalgorithmus, wie beim Signieren, berechnet. Danach entschlüsselt man die Signatur mit Hilfe des öffentlichen Schlüssels des Signierers. Das Ergebnis dieser Entschlüsselung wird dann mit der aktuellen Prüfsumme verglichen. Sind die beiden Hashwerte gleich, wird die Signatur akzeptiert, ansonsten deutet dies auf Manipulation, oder fehlerhafte Übertragung hin. In diesem Fall wird die Signatur nicht anerkannt.

Die Authentizität durch die Prüfung der digitalen Signatur des Dokuments ist gewährleistet

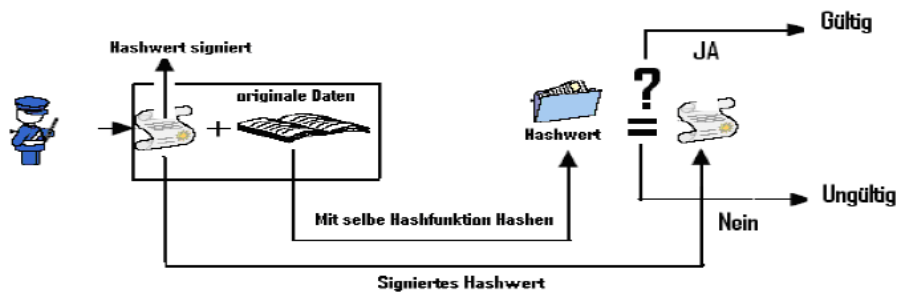


Abbildung 2.4: Prüfung einer digitalen Signatur

und die Integrität des Dokumentes ist auch durch die Prüfsummen prüfbar.

Es sei an dieser Stelle auf die Klassifizierung von digitalen Signaturen aufmerksam gemacht, wonach deren Verbindlichkeit je nach Verwendungskontext unterschiedlich bewertet wird. Die digitalen Signaturen sind Gesetzlich in mehrere Klassen geteilt, :

- Normale oder einfache Signatur
- Fortgeschrittene Signatur
- Qualifizierte Signatur

2.2 Zertifikate in Public Key Infrastruktur

2.2.1 Certification Authority

Die Bezeichnung Public Key Infrastruktur (PKI) bedeutet übersetzt „Die Infrastruktur öffentlicher Schlüssel“ und wird auch in der deutschen Literatur in der englischen Form verwendet. Der Begriff Schlüssel in Zusammenhang mit IT-Sicherheit ist wiederum der Kryptographie entnommen worden.

Eine PKI setzt optimal kryptographische Anwendungen ein, und stellt ein Hilfsmittel für die Absicherung kritischer IT-Prozesse, insbesondere in Bezug auf Authentifizierung, Autorisierung und Verschlüsselung dar.

Mit der PKI wird dafür gesorgt, dass die Schlüsselpaare teilnehmender Kommunikationspartner innerhalb einer Organisation sorgfältig verwaltet werden. Im Detail bedeutet dies, dass mit einer PKI die Schlüsselpaare, solange sie verwendet werden dürfen, immer folgende Eigenschaften aufweisen:

- Integrität
- Vertraulichkeit
- Authentizität

Der Aufbau einer PKI basiert auf das so genannte hierarchische Vertrauensmodell. Bei diesem Modell handelt es sich um eine Vertrauensinstanz (Wurzel-Instanz, eng. Root-CA), auf die sich die Teilnehmer als solche einigen. Diese Instanz hat im Wesentlichen für die folgenden Punkte Sorge zu tragen:

- richtige Zuordnung der Schlüsselpaare zu den Teilnehmern: über diese Zuordnung wird ein Zertifikat erzeugt und mit dem privaten Schlüssel der Wurzel-Instanz signiert. Auf Zertifikate wird später detaillierter eingegangen.
- Schlüsselsperrung und Sperrinformationen zur Verfügung stellen: das findet nur dann statt, wenn es entsprechend begründet ist.
- ggf. Schlüsselerzeugung und Festlegen des Schlüsselzwecks: Es ist zum Beispiel von Vorteil, dass zwischen Signier- und Chiffrierschlüssel unterschieden wird.

Dadurch, dass ein oder mehrere Teilnehmer auch als Wurzel-Instanz anderer Organisationen fungieren können, entstehen Vertrauensbäume. Jeder Vertrauensbaum repräsentiert in der Praxis die PKI seiner Organisation. Die Wurzel-Instanz wird in der Literatur meist auch als TrustCenter oder Zertifizierungsstelle „Certification Authority“ (CA) bezeichnet.

2.2.2 Zertifikate

Das sog. digitale Zertifikat stellt eine Verknüpfung einer Identität mit ihrem Schlüsselpaar dar. Anders formuliert ist das Ziel eines digitalen Zertifikats das Garantieren der Zusammengehörigkeit eines Subjekts (Person, Behörde, Maschine, Programm etc...) zu seinem öffentlichen Schlüssels.

Ein digitales Zertifikat kann analog zu einem Personalausweis angesehen werden:

- Hat ein bestimmtes Format
- Enthält identifizierende Informationen über den Inhaber
- Wird von einer vertrauten und dazu befugten Behörde (CA) ausgestellt
- Trägt den Stempel (Signatur) des Ausstellers
- Hat einen Gültigkeitszeitraum

2.2.2.1 X.509-Zertifikat

Wir haben das primäre Ziel eines Zertifikats gesehen, nämlich das Anbinden des öffentlichen Schlüssels zu seinem Inhaber.

Es wurde schnell erkannt dass, einheitliche Standard Zertifikatsformate benötigt werden, um die Unabhängigkeit von den jeweiligen Zertifizierungsstellen zu gewährleisten. Der mittlerweile meist verbreitete X.509-Standard wurde schon 1988, von der Internationale Fernmeldeunion¹, eine Unterorganisation der Vereinten Nationen veröffentlicht.

Ein X.509v3 -Zertifikat enthält Standard-Attribute, die im ASN.1-Format codiert sind. Seit der dritten Version v3 bietet sich die Möglichkeit, Erweiterungen (engl. Extensions); von der jeweiligen Zertifizierungsstelle freie selbst definierte Einträge, einzufügen. Auch hier gibt es vordefinierte Standard-Extensions.

Die Standardfelder bzw. -Attribute sind:

- `version`: Versionsnummer
- `serialNumber`: Seriennummer des Zertifikates, eindeutig für jeden ausgestellten Zertifikat bei demjenigen Aussteller
- `signature`: Signaturverfahren womit das Zertifikat signiert wurde
- `issuer`: Name des Ausstellers, als X.500 Distinguished Name
- `validity`: Gültigkeitszeitraum des Zertifikates von - bis
- `subject`: Name des Zertifikatsinhabers, als X.500 Distinguished Name

¹<http://www.itu.int/>

- subjectPublicKeyInfo: öffentlicher Schlüssel des Zertifikatinhabers, plus technische Informationen zum Schlüssel z.B Algorithmus Name...
- subjectUniqueID: eindeutige Nummer für Zertifikastinhaber
- issuerUniqueID: : eindeutige Nummer für Aussteller
- Extensions: Erweiterungen

Es folgt ein Ausschnitt aus einer Textdarstellung eines Zertifikats nach X.509v3

```
Certificate:
Data:
Version: 3 (0x2)
Serial Number: 1 (0x1)
Signature Algorithm: md5WithRSAEncryption
Issuer: C=XY, ST=USA, L=NY, O=U&M Ltd, OU=Best CA, CN=CA/Email=ca@u&mca.dom
Validity
  Not Before: Oct 29 17:39:10 2000 GMT
  Not After : Oct 29 17:39:10 2001 GMT
Subject: C=EN, ST=USA, L=NY, O=Home, OU=Web Lab, CN=XY.com/Email=xy@XY.com
Subject Public Key Info:
  Public Key Algorithm: rsaEncryption
  RSA Public Key: (1024 bit)
  Modulus (1024 bit):
    00:c4:40:4c:6e:14:1b:61:36:84:24:b2:61:c0:b5:
    d7:e4:7a:a5:4b:94:ef:d9:5e:43:7f:c1:64:80:fd:
    9f:50:41:6b:70:73:80:48:90:f3:58:bf:f0:4c:b9:
    90:32:81:59:18:16:3f:19:f4:5f:11:68:36:85:f6:
    1c:a9:af:fa:a9:a8:7b:44:85:79:b5:f1:20:d3:25:
    7d:1c:de:68:15:0c:b6:bc:59:46:0a:d8:99:4e:07:
    50:0a:5d:83:61:d4:db:c9:7d:c3:2e:eb:0a:8f:62:
    8f:7e:00:e1:37:67:3f:36:d5:04:38:44:44:77:e9:
    f0:b4:95:f5:f9:34:9f:f8:43
  Exponent: 65537 (0x10001)
X509v3 extensions:
  X509v3 Subject Alternative Name:
    email:xyz@anywhere.com
  Netscape Comment:
    mod_ssl generated test server certificate
  Netscape Cert Type:
    SSL Server
Signature Algorithm: md5WithRSAEncryption
12:ed:f7:b3:5e:a0:93:3f:a0:1d:60:cb:47:19:7d:15:59:9b:
3b:2c:a8:a3:6a:03:43:d0:85:d3:86:86:2f:e3:aa:79:39:e7:
82:20:ed:f4:11:85:a3:41:5e:5c:8d:36:a2:71:b6:6a:08:f9:
```

```
cc:1e:da:c4:78:05:75:8f:9b:10:f0:15:f0:9e:67:a0:4e:a1:
4d:3f:16:4c:9b:19:56:6a:f2:af:89:54:52:4a:06:34:42:0d:
d5:40:25:6b:b0:c0:a2:03:18:cd:d1:07:20:b6:e5:c5:1e:21:
44:e7:c5:09:d2:d5:94:9d:6c:13:07:2f:3b:7c:4c:64:90:bf:
ff:8e
```

X.509 Zertifikate werden von verschiedenen Anwendungen unterstützt, diese dienen unterschiedlichen Zwecken darunter:

- Email-Verschlüsselung
- Email-Signatur
- Datei-Verschlüsselung
- Datei-Signatur
- Dokumentensignatur
- Benutzerauthentifikation
- VPN Authentifizierung
- SSL Serverzertifikat für Webserver

Das Wurzelzertifikat, auch als Rootzertifikat bezeichnet, ist das Zertifikat des obersten Ausstellers, der vertrauenswürdigen Instanz der PKI. Es ist zwar ein normales Zertifikat, mit dem in ihm enthaltenen Schlüssel werden aber generell alle Daten (Sperrlisten, andere Zertifikate...) auf Authentizität und Integrität geprüft. Von seiner Echtheit hängt das Vertrauen in der gesamten Infrastruktur ab. Man muss ihm immer direkt vertrauen können. Daher muss man je nach Sicherheitslevel, während des gesamten Prozedurablaufs, also von der Wurzelzertifikaterstellung bis hin zu seiner Auslieferung bzw. Installation bei dem Endbenutzer, besonders sichern.

2.2.3 Zertifikat Überprüfung

2.2.3.1 Sperrlisten

Um Informationen über den Status eines Zertifikates zu erhalten bedarf es Veröffentlichungen von der Ausstellenden Instanz. Denn ein Zertifikat kann aus unterschiedlichen Gründen Z.B. Schlüssel können Kompromittiert werden, etc... innerhalb seiner Gültigkeitszeit widerrufen werden. Daher werden vom Aussteller verschiedene Möglichkeiten angeboten, damit zu jeder

Zeit den Status eines Zertifikates abgefragt werden kann.

Die so genannte Revokationsliste auch Sperr-, oder Widerrufs-Liste genannt² ist eine vom Zertifikatsaussteller, generierte Liste mit Seriennummern und ggf. die Sperrgründe aller gesperrten Zertifikate. Diese kann von jedem Teilnehmer, jederzeit geladen werden, er kann mit deren Hilfe vor der Verwendung eines Zertifikates überprüfen können, ob die Seriennummer der zu Prüfenden Zertifikat nicht auf dieser Liste gelistet ist.

Es wird bei jeder Sperrung oder in periodischen Zeitabstände eine neue Sperrliste erstellt, das könnte eventuell zur Inkonsistenzen zwischen lokal vorliegender und der zentral bei dem Aussteller gehaltenen CRL führen. Dieses Szenario lässt sich nicht ganz vermeiden. Durch das öfters Updaten der Lokale Listen vom Benutzern, kann das Problem weitestgehend abgefangen werden, allerdings führt dies zu Problemen der Skalierbarkeit.

2.2.3.2 Gültigkeitsmodelle

Da nur der Zertifikatsaussteller die Zertifikate produzieren kann und die von ihm ausgestellten Zertifikate zurückziehen (Revokation) darf, hat jeder Teilnehmer einer PKI die Möglichkeit, mit relativ einfachem Aufwand Manipulationen aufzudecken und die Gültigkeit der Zertifikate zu überprüfen, wenn er im Besitz des authentischen Wurzelzertifikats ist.

Bei der Prüfung eines Zertifikates muss das festgelegte Gültigkeitsmodell beachtet werden. Denn neben der Prüfung der digitalen Signatur des Teilnehmerzertifikats muss auch das überstehende Zertifikat der Ausstellenden bzw. der Zertifizierungsinstanz geprüft werden. Es wird rekursiv entlang der Zertifizierungskette weiter geprüft bis zur Prüfung des Zertifikats vom Vertrauensanker. Das Ergebnis dieser Prüfung kann von einem Gültigkeitsmodell zum anderen variieren. D.h. es kann unter einen Gültigkeitsmodell gültig sein während beim Prüfen unter einem anderen Gültigkeitsmodell ungültig wird.

Wann ein Zertifikat für gültig erklärt wird, und wann eine Signatur als beweiskräftig zu betrachten ist, gibt es im Allgemeinen nicht. Daher wird hier auf die Bestimmungen der BSI³ zurückgegriffen:

- Eine digitale Signatur ist genau dann gültig,

²engl: Certificate Revocation List, CRL

³Bundesamt für Sicherheit in der Informationstechnik, www.bsi.bund.de

- wenn sie mathematisch richtig ist und
- wenn zum Zeitpunkt der Signatursübung der zugehörige Signaturschlüssel gültig war
- Eine digitale Signatur ist genau dann beweiskräftig,
 - wenn sie zum Zeitpunkt der Prüfung entsprechend dem verwendeten Gültigkeitsmodell als gültig anerkannt wird und
 - wenn der zugehörige Signaturschlüssel nicht kompromittiert ist

Zur Verifikation gibt's zwei grundlegende Modelle, nämlich das Schallenmodell⁴ und das Kettenmodell⁵.

- Schallenmodell: Die Prüfung unter dem Schallenmodell sieht vor, dass der Gültigkeitszeitraum des ausstellenden Zertifikats (CA-Zertifikat) im Zeitpunkt der Prüfung umfasst den Gültigkeitszeitraum des ausgestellten Zertifikats, d.h. der Gültigkeitszeitraum jedes Zertifikats der Kette ist vollständig eingeschlossen vom Gültigkeitszeitraum aller über ihn bzw. ihm übergeordneten Zertifikates. Sollte ein CA-Zertifikat zurückgenommen (Revokation) werden, impliziert dies den Widerruf aller ihm untergeordneten Zertifikate.

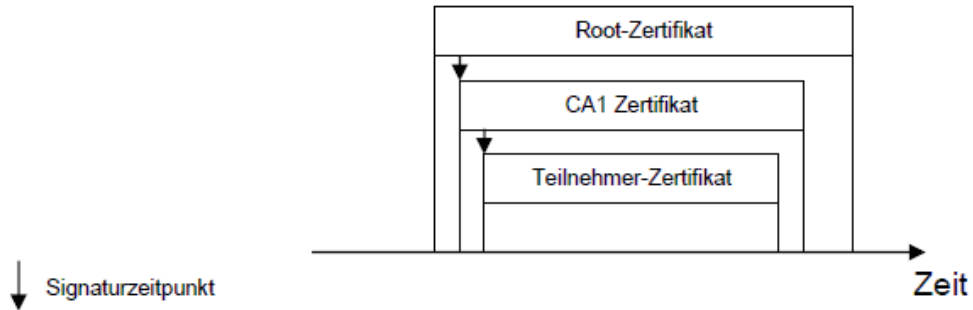


Abbildung 2.5: Das Schallenmodell

⁴eng. shell
⁵engl. chain

- Kettenmodell: Ein Zertifikat in einer Zertifikatskette ist gültig, wenn es innerhalb des Gültigkeitszeitraums des jeweiligen ausstellenden bzw. übergeordneten Zertifikats ausgestellt wurde. Das ausgestellte Zertifikat darf eine Gültigkeitsdauer besitzen, die über die Gültigkeitsdauer des ausstellenden Zertifikates hinausgeht. Dem Signaturzeitpunkt und ggf. den Widerrufsinformationen fallen aber besondere Bedeutungen zu, denn bei einer Signaturprüfung wird der Erstellungszeitpunkt mit einem Widerrufszeitpunkt verglichen. Daher müssen diese Informationen auch nach Ende des Gültigkeitszeitraums eines Zertifikates abrufbar sein.

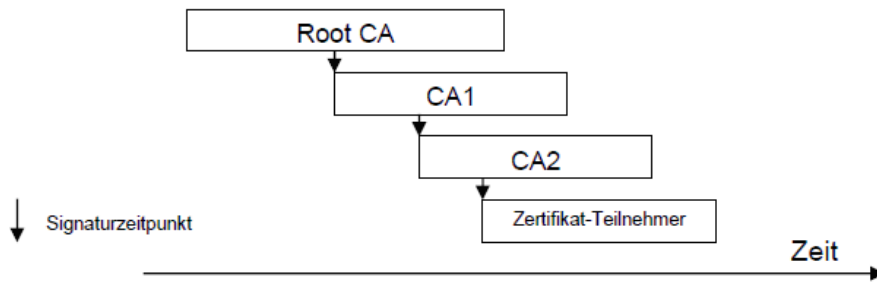


Abbildung 2.6: Das Kettenmodell

2.3 Online Certificate Status Protocol: Online-Sperrprüfung

Mit Hilfe eines OCSP⁶ kann der Status eines Zertifikats beim OCSP Server abgefragt werden. Dieser sog. OCSP Responder wird in den meisten Fällen von der Organisation, die das Zertifikat erstellt hat (Zertifizierungsstelle), betrieben. Mögliche Antworten sind:

- `good`: d. h. Zertifikat ist nicht gesperrt
- `revoked`: Zertifikat ist gesperrt
- `unknown`: der Status konnte nicht ermittelt werden

OCSP-Response-Daten werden in der Regel vom Server signiert.

OCSP-Requests mit mehreren Zertifikaten sind in dem Protokoll auch vorgesehen. Die Response ist dann eine Zertifikatsstatusliste für die jeweiligen Zertifikaten.

Da unter Umständen für die Gültigkeit des Signierens mit einem Zertifikat, der Status dieses Zertifikats im Zeitpunkt der Signierung (Vergangenheit) und nicht zum Prüfungszeitpunkt (Zeitpunkt der OCSP-Abfrage) relevant ist, werden bei den OCSP-Antworten für gesperrte Zertifikate auch den Sperrzeitpunkt angegeben. Das macht es möglich, den Status im Bezug auf einen bestimmten Zeitpunkt zu ermitteln. Damit kann die Frage, ob das Zertifikat im Zeit X noch gültig war, beantwortet werden. Problematisch wird es aber wenn die Zertifizierungsstelle provisorischer Sperrungen⁷ erlaubt. In so einem Fall darf man die OCSP Antwort `good` nicht vertrauen.

OCSP-Responder können Sperrinformationen in Echtzeit liefern, wenn sie Zugriff auf aktuelle Datenquellen der Zertifizierungsstelle haben (z. B. die CA-Datenbank). Bei dem OCSP-Request geht es um eine Anfrage, die der Teilnehmer an das TrustCenter sendet, darin ist die Seriennummer von einem bestimmten oder mehreren Zertifikat(en) enthalten. Daraufhin überprüft das Ausstellende TrustCenter intern ob das zu der jeweiligen Seriennummer passende Zertifikat gesperrt wurde. Die Antwort sendet es signiert, dabei kann es optional auch den Sperrgrund hinzufügen.

⁶Online Certificate Status Protocol, RFC2560

⁷Suspendierungen

3 Simple Certificate Validation Protocol

3.1 ASN.1

Es wird an diesem Abschnitt auf ASN.1 eingegangen, da es für SCVP die Basis der Datenstrukturen darstellt.

Die Abbildung [3.1] illustriert den Datenaustausch auf ASN.1-Basis zwischen zwei Anwendungen, die auf unterschiedliche Systeme (Hardware, Betriebssystem...) laufen. ASN.1 ist ein Datenstandard, mit dem Standardempfehlungen präzise, kompakt und mit

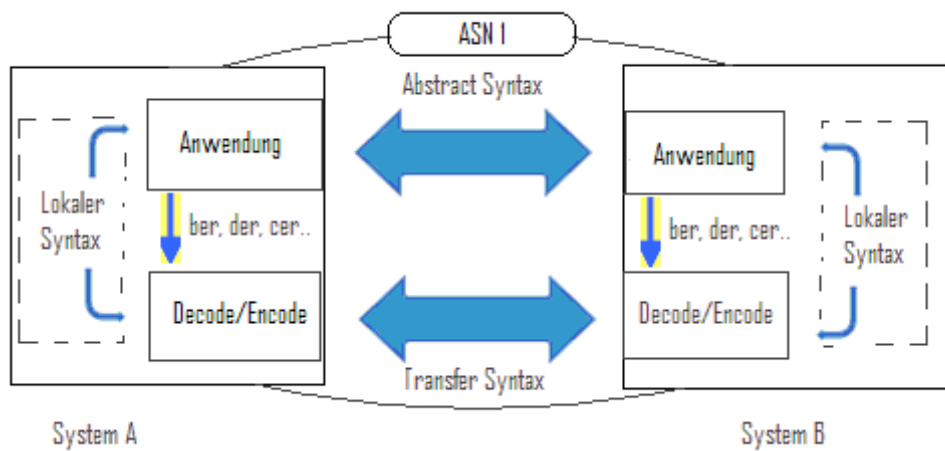


Abbildung 3.1: Datenaustausch auf ASN.1-Basis

einem hohen Grad an Abstraktion beschrieben werden. Auf Basis von ASN.1 lassen sich ebenfalls Datenstrukturen eindeutig spezifizieren. Vorteilhaft ist, dass Datenstrukturen,

die mittels ASN.1 verfasst worden sind, sowohl für den Mensch als auch maschinell lesbar sind.

In vielen Fachliteraturquellen wird die Ähnlichkeit zu höheren Programmiersprachen hervorgehoben und an ihr angelehnt ASN.1 erläutert. Denn wie die gängigen Programmiersprachen bietet auch ASN.1 vorgegebene Datentypen, die beim Spezifizieren Einsatz finden können, sowie Schlüsselwörter und Zuweisungsmechanismen.

Auch wenn ASN.1 nicht als Programmiersprache bezeichnet werden kann, bietet sie zusätzlich vier verschiedene Klassen von Datenstrukturen:

- UNIVERSAL
- APPLICATION
- PRIVATE
- CONTEXT-SPECIFIC

Die Universal Klasse stellt verschiedene Arten an Datentypen zur Verfügung:

- `atomic-types`: öfters auch als `primitiv-types`, sind Datentypen die nur einen Typ enthalten, und nicht weiter unterteilbar sind, darunter sind :

- BOOLEAN
- INTEGER
- ENUMERATED
- REAL
- BIT STRING
- OCTET STRING
- NULL
- PrintableString
- UTF8String
- Object Identifier

Diese Datentypen sind analog zu anderen primitiven Datentypen in anderen Programmiersprachen zu verstehen, wobei `Object Identifier` ein primitiver Datentyp ist, der eine Art Objektinformation zur weltweit eindeutigen Identifizierung von Objekten wie z.B. ASN.1-Modulen, Organisationen, kryptographische Algorithmen, Attributen usw. darstellt.

- `structured-types` : oder zusammengesetzte Typen sind auch unter ASN.1 möglich darunter sind :
 - Wiederholung (`SEQUENCE`) : Folge von mehreren Komponenten, die verschiedene Typen besitzen. Die Reihenfolge der Komponenten spielt eine Rolle.
 - Wiederholung (`SEQUENCE OF`) : Folge von mehreren Komponenten, die alle den gleichen Typ besitzen; die Reihenfolge der Komponenten spielt eine Rolle.
 - Liste von Feldern (`SET`) : Folge von mehreren Komponenten, die verschiedene Typen besitzen; die Reihenfolge der Komponenten spielt keine Rolle.
 - Liste von Feldern (`SET OF`) : Folge von mehreren Komponenten, die alle den gleichen Typ besitzen; die Reihenfolge der Komponenten spielt keine Rolle.
 - Markierte od. gekennzeichnete Objekte (`TAGGED TYPES`) : sind von anderen Typen abgeleitet. Hierbei wird eine Typkennungen bzw. Markierung eingeführt, um Kodierung sowie Unterscheidungen zu ermöglichen.
 - andere
 - * `CHOICE` : ein oder mehrere Alternativen
 - * `ANY` : ein beliebiger Wert eines beliebigen type

Die Abbildung [3.2] zeigt ein Fragment eines X.509v3-Zertifikats als ASN.1 Struktur.

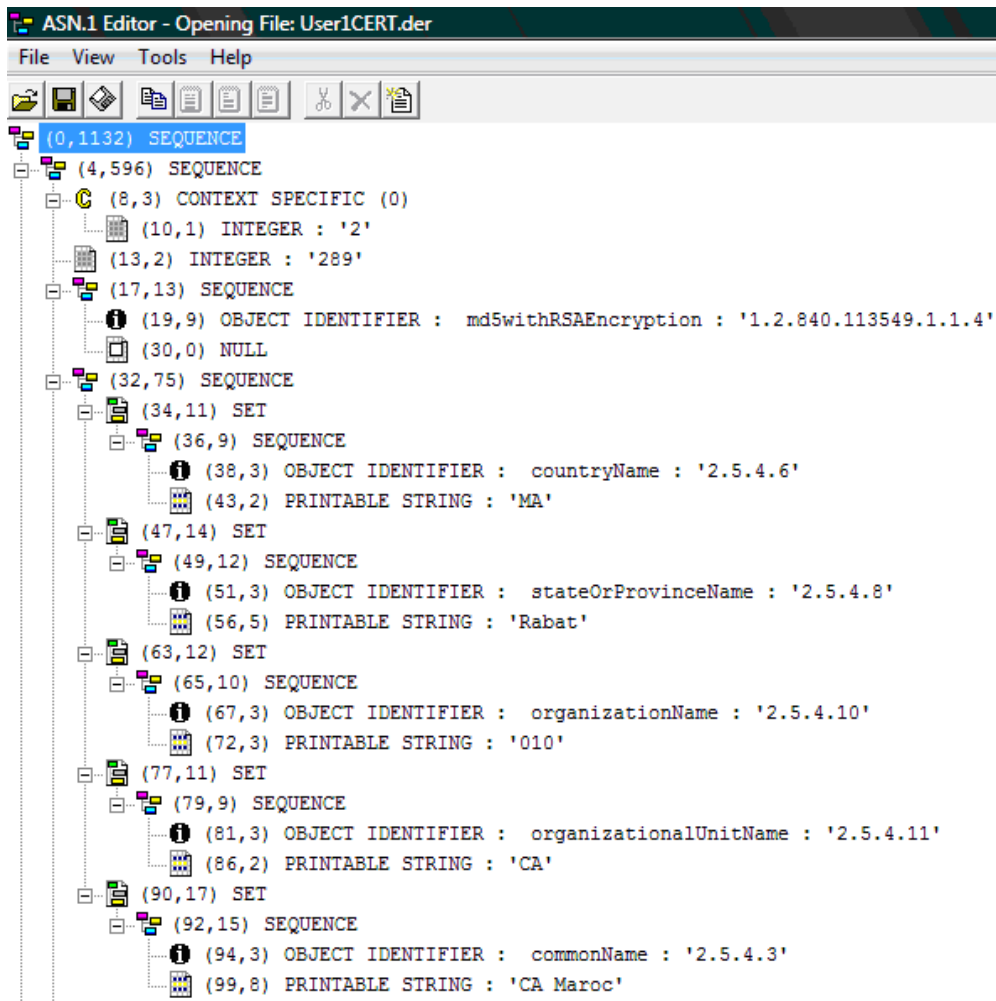


Abbildung 3.2: Fragment eines X.509v3-Zertifikats als ASN.1 Struktur

3.2 SCVP

Der primäre erklärte Ziel vom SCVP ist es, die Benutzung von PKI fähige Applikation einfacher zu gestalten und somit zu breiterer Akzeptanz zu verhelfen: Durch das Delegierungsprinzip, wird die aufwändige (Zertifikat) Pfad Bildung und/oder die Gült-

tigkeitsprüfung auf einen Server, den SCVP-Server ausgelagert. Dadurch ergibt sich die Möglichkeit die (`Validation Policies`) innerhalb einer Organisation die eigenen Regeln, und Bedingungen für die Zertifikatsgültigkeitsprüfung zentral zu verwalten (serverseitig).

Wie schon eingangs erwähnt, bedeuten die Dienste eines SCVP-Servers eine Entlastung für Clients. Dies gilt besonders für Clients, die mit der Pfadbildung und der Validierung aufgrund fehlender Ressourcen oder Protokolle überlastet sind oder deren Systeme auf andere Ziele fokussiert sind, sodass sie Pfadbildung- bzw. Validierungsdienste delegieren sollen.

Die erste Klasse von Clientanwendungen verlangt zwei Informationen vom SCVP nämlich :

- eine Bestätigung, dass der public key zu der enthaltenen Identität im Zertifikat gehört
- eine Bestätigung, dass der public key für den beabsichtigten Zweck¹ benutzt werden kann.

Solchen Clients delegieren sowohl die Pfadbildung als auch die Validierung vollständig an den SCVP Server. Dieser Prozess wird `delegated path validation (DPV)` genannt.

Die zweite Klasse von Anwendungen delegiert nur die Pfadbildung, während die Validierung lokal durchgeführt wird. Dies wird mit `delegated path discovery (DPD)` bezeichnet.

Generell gibt es zwei Gründe warum Clients auf SCVP setzen können:

1. Clients bzw. Anwendungen wollen es vermeiden, für jeden Zertifikat den sie bekommen die Prozesse der Pfadbildung und der Validierung durchzuführen.
2. Clients sind in einer Organisation, wo man auf eine zentrale Verwaltung für `Validation Policies` setzt.

¹Signatur, Qualifizierte Signatur, etc...

Die typische Nutzung von SCVP ist über das http Protokoll², jedoch kann jedes anderes Protokoll benutzt werden. Vorausgesetzt, es unterstützt den Transport bzw. die Übermittlung von digital signierten Objekten. Die Datenübermittlung zwischen Server und Clients läuft über die http-Post Methode.

Es sei im Folgenden auf Konstrukte eingegangen, die beim Verständnis von SCVP relevant sind.

3.2.1 Validation Policies

Sie stellen die Regeln und Parameter dar, die vom SCVP Server beim Validieren von Zertifikaten bzw. Zertifikatsketten benutzt werden.

Die vom SCVP Server zu verwendende Validation Policy³ kann vom Clients in seinen Requests entweder völlig (keine Parameterübertragung) oder partiell referenziert (mit additional Parametern) werden.

Definiert die Validation Policy alle notwendigen Parameter, so hat die SCVP-Anfrage (request) lediglich das zu validierende Zertifikat oder gar nur eine Referenz darauf (PKCReference), eine Referenzierte Validation Policy (OID) und die run-time Parametern zu enthalten.

Ein Server veröffentlicht die Referenzen zu den Validation policies, die er unterstützt. Sollte es erlaubt sein, einige Parameter zu überschreiben, dann werden die Standardwerte dieser Parameter ebenfalls veröffentlicht. Ein Client kann dann, in seiner Request diese Parameter überschreiben, oder die default Werte übernehmen. Ein Client hat die Möglichkeit die benutzte Validation Policy samt aller Parameter anzufordern, der Server packt dann die policy und die zugehörigen Parametern in seiner Response ein, und leitet sie an den Client weiter.

SCVP benutzt das sogenannte `basic certification path processing algorithm` der im Abschnitt 6.1.1 vom RFC 3280 definiert ist.

²RFC 2616

³RFC 3379

Die Eingaben (inputs) zu diesem Algorithmus sind:

- certification path validated, by value or by reference
- Validation time
- The initial policy set
- Initial inhibit policy mapping setting
- Initial inhibit anyPolicy setting
- Initial require explicit policy setting

Der basic certification path processing algorithm, bcpp unterstützt die Eingabe eines oder mehrere Vertrauensankern (by value or reference), als Algorithmus-Eingabe. Dadurch entsteht für den Client die Möglichkeit, einen aus mehreren Vertrauensankern, die für ihn akzeptabel sind zu bestimmen. D.h wenn der client ein Pfad mit einem bestimmten CA-Ursprung verlangt, wird dementsprechend einen Vertrauensanker spezifiziert. Wenn er aber einen Pfad beginnend mit einem von mehreren CA bereit zu akzeptieren, dann wird eine Reihe von Vertrauenankern angegeben.

Folgende zwei Parameter werden auch vom bcpp-Algorithmus unterstützt⁴ :

- Der Benutzungszweck, der im Zertifikat enthaltenen Public Key (usage of key z.B. key encipherment, key agreement, signature)
- Andere applikationsspezifische Zwecke, wofür der zertifizierte public key Benutzt werden darf.

3.2.2 Validation Algorithmus

Der Validation Algorithmus wird durch eine OID Referenz zwischen Client und Server vereinbart, dieser definiert Überprüfungen und Bedingungen sog. Checks, die das Zertifikat bei der Validierung erfüllen muss. Der Validation Algorithmus ist selbst ein Parameter der Validation Policy.

⁴siehe rfc 3280 Abschnitt 4

SCVP baut auf den basic Validation Algorithmus (RFC 3280), der vom Path Validation Algorithm (RFC 3280) benötigt wird.

Neue Validation Algorithmen können definiert werden, um zusätzliche Einschränkungen, Kontrolle und Prüfungen zu implementieren. Dies bedeutet, dass es je nach Anforderung applikationsspezifische Validation Algorithmen entwickelt werden können.

3.2.3 Validation Requirements:

Damit ein Zertifikatspfad bzw. Zertifizierungskette unter einer bestimmten Validation Policy für gültig erklärt wird, muss dieser Pfad, wie von RFC 3280 definiert, gültig sein und alle Einschränkungen der Validation policy erfüllen. Ein Aspekt der in den RFC 3280 definierten Kontrollen ist die Sperrungsüberprüfung⁵, jedoch ist diese optional.

Clients geben in ihren Requests an, ob diese Prüfung stattfinden soll, und ob der SCVP Server die Revokationsinformation auch an die Clients weiterleiten soll. Der Server muss die Fähigkeit besitzen, die Revokation-Informationsquelle anzugeben.

Mögliche Revokation Informationsquelle sind:

- Full CRLs (or full Authority Revocation Lists)
- OCSP responses
- Delta CRLs
- Indirect CRLs

3.2.4 Validation Request

SCVP schließt zwei Request-Response Paare ein. Das erste Paar behandelt die Zertifikat Validation, wohingegen es sich bei dem zweiten Paar um sämtliche Informationen über Validation policies, die vom SCVP unterstützt werden, handelt.

Eine SCVP-Request hat als Single-CVRequest Element zu sein. Es gibt zwei Typen von SCVP-Anfragen: geschützt (protected) und ungeschützt (unprotected). Die geschützte

⁵Revocation Checking

Request wird benutzt um den Client gegenüber dem Server zu authentifizieren, oder um die Namenlose Client-Integrität zu gewährleisten. Der Schutz ist mittels digitaler Signatur oder Message Authentication Code (MAC) gegeben. Ein SCVP Server kann voraussetzen, dass alle Requests in der protected Form sein müssen, kann aber auch Unprotected Requests akzeptieren. In der in dieser Arbeit implementierten SCVP Server-Applikation werden sowohl protected als auch unprotected Requests weitergeleitet.

Die unprotected Request muss folgende Form aufweisen:

```
ContentInfo {
    contentType      id-ct-scvp-certValRequest
                    -- (1.2.840.113549.1.9.16.1.10) --
    content          CVRequest }
```

Die protected Request besteht aus einer CVRequest [Abb 3.3] gekapselt in einer SignedData oder AuthenticatedData, in der wiederum eine ContentInfo gekapselt ist. Das ist das sogenannte EncapsulatedContentInfo Feld von der jeweiligen Signed- oder Authenticated-Data, bestehend aus eine CVRequest in DER-Kodierung.

SignedData wird benutzt, wenn die Anfrage digital signiert wird, AuthenticatedData wird in Zusammenhang mit dem MAC ⁶ eingesetzt. Syntax und Semantik für Signed-, Authenticated-Data und ContentInfo sind in (CMS-rfc) ⁷ definiert.

⁶Message Authentication Code

⁷Cryptographic Message Syntax, RFC 3852

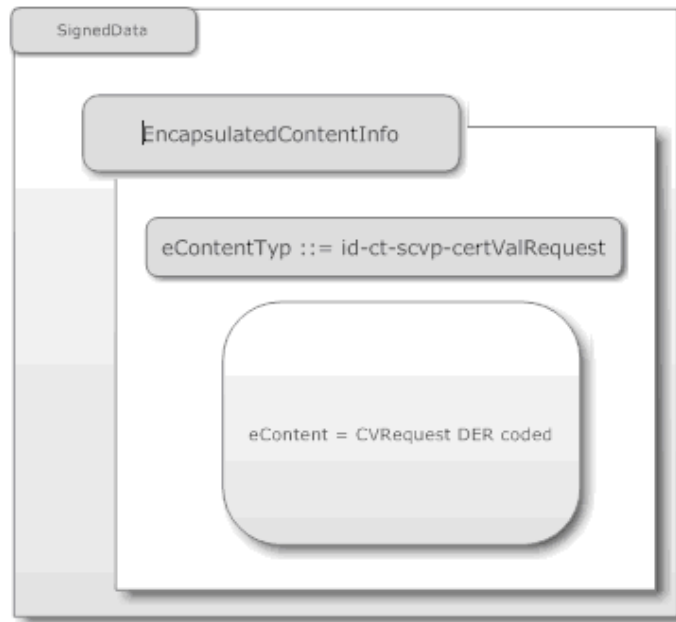


Abbildung 3.3: Signed Data gekapselter CVRequest

Die CVRequest hat folgende Syntax :

```

CVRequest ::= SEQUENCE {
    cvRequestVersion      INTEGER DEFAULT 1,
    query                 Query,
    requestorRef          [0] GeneralNames          OPTIONAL,
    requestNonce          [1] Octet String          OPTIONAL,
    requestorName        [2] GeneralName           OPTIONAL,
    responderName        [3] GeneralName           OPTIONAL,
    requestExtensions    [4] Extensions            OPTIONAL,
    signatureAlg         [5] AlgorithmIdentifier    OPTIONAL,
    hashAlg              [6] OBJECT IDENTIFIER      OPTIONAL,
    requestorText        [7] UTF8String (SIZE 1..256) OPTIONAL
}

```

Clients müssen in der Lage sein, CVRequest mit cvRequestVersion und query zu konstruieren. Dazu muss auch die CVRequest vom Clients DER kodiert wer-

den. Clients die darauf bestehen, neue Response zu erhalten z.B. gegen replay attack oder damit sie eine Uptodate-Response erhalten, sollen `requestNonce` in ihrem Request einbeziehen. Die übrigen Elemente sind optional.

`cvRequestVersion`: gibt die Version der CVRequest an. Spätere Response muss die selbe Version-Nummer haben. Aktuelle Version ist 1. Künftige Updates können die Version dementsprechend erhöhen, jedoch kann man Erweiterungen definieren ohne die Version zu ändern.

`Query`: spezifiziert ein oder mehrere Zertifikate, die den Subjekt der Request bilden. Die Zertifikate können entweder Public Key-⁸ oder Attribut-Zertifikate⁹ sein. Eine Query muss folgende drei Komponenten enthalten:

- `queriedCerts`
- `checks`
- `validationPolicy`,

und kann freier noch folgende Elemente beinhalten: `wantBack`, `responseFlags`, `serverContextInfo`, `validationTime`, `intermediateCerts`, `revInfos`, und `queryExtensions`.

Die Query Syntax sieht wie folgt aus:

```
Query ::= SEQUENCE {
    queriedCerts          CertReferences,
    checks                CertChecks --tag [0] not used--
    wantBack              [1]  WantBack          OPTIONAL
    validationPolicy      ValidationPolicy
    responseFlags        ResponseFlags         OPTIONAL,
    serverContextInfo    [2]  OCTET STRING      OPTIONAL,
    validationTime       [3]  GeneralizedTime  OPTIONAL,
    intermediateCerts    [4]  CertBundle        OPTIONAL,
    revInfos             [5]  RevocationInfos   OPTIONAL,
    producedAt          [6]  GeneralizedTime  OPTIONAL,
    queryExtensions      [7]  Extensions       OPTIONAL,
}
```

⁸PKIX-1, RFC 3280

⁹PKIX-AC, RFC3281

queriedCerts: Enthält ein Zertifikat oder eine Liste von Zertifikaten, die jeweils Thema der Anfrage darstellen und entweder Public Key certificates oder attribute certificates sein können. Wenn mehr als ein Zertifikat vorhanden ist, dann müssen alle vom selben Typ sein. Jedes Zertifikat kann ganz in dem Request enthalten sein, oder nur referenziert werden.

```
CertReferences ::= CHOICE {  
    pkcRefs    [0] SEQUENCE SIZE (1..MAX) OF PKCReference,  
    acRefs     [1] SEQUENCE SIZE (1..MAX) OF AReference }
```

```
PKCReference ::= CHOICE {  
    cert       [0] Certificate,  
    pkcRef     [1] SCVPCertID }
```

```
ACReference ::= CHOICE {  
    attrCert   [2] AttributeCertificate,  
    acRef      [3] SCVPCertID }
```

```
SCVPCertID ::= SEQUENCE {  
    certHash      OCTET STRING,  
    issuerSerial   SCVPIssuerSerial,  
    hashAlgorithm AlgorithmIdentifier DEFAULT {algorithm sha-1}}
```

```
SCVPIssuerSerial ::= SEQUENCE {  
    issuer         GeneralNames,  
    serialNumber   CertificateSerialNumber  
}
```

Für den Fall, dass eine Referenz benutzt ist, muss der Hashwert von dem referenzierten Zertifikat im Request inbegriffen sein, damit beide; Client und Server sicherstellen können, dass es sich um denselben Zertifikat handelt. Der Hashwert in `SCVPCertID` wird

über das gesamte DER-Zertifikat berechnet, Signatur inklusive.

checks: Dieses Feld gibt an, welche Überprüfungen der Server im Auftrag des Clients erledigen soll. Er enthält eine Sequenz von OIDs. Jedes OID ist eine Referenz auf einen bestimmten Check. Der Server muss jeden referenzierten Check ausführen, oder einen Fehler (error) zurückgeben, wenn die Durchführung nicht möglich war.

`CertChecks ::= SEQUENCE SIZE (1..MAX) OF OBJECT IDENTIFIER`

In dem SCVP aktuellen RFC 5055, sind für public key certificates folgende Checks definiert:

- `id-stc-build-pkc-path`: Bilde ein Cert Pfad zu einem Trustanker wie es in [PKIX-1] definiert wurde
- `id-stc-build-valid-pkc-path`: Bilde ein validierten Cert Pfad zu einem Vertrauensanker (Revokation Prüfung nicht notwendig)
- `id-stc-build-status-checked-pkc-path`: Bilde ein validierten Cert Pfad zu einem Vertrauensanker und prüfe den Revokation-Status

Wie schon oben erwähnt wurde, können zusätzliche Checks bei Bedarf implementiert werden.

Wantback: Dieses optionale Feld beschreibt zusätzliche Informationen, die der Client vom Server für jedes in dem `queriedCerts` enthaltenen Zertifikat erfahren möchte. Diese Informationen werden als Checks-Zusatz definiert. Das Feld umfasst eine Sequenz von OIDs. Dabei teilt jedes OID mit, was der Client über die `queriedCerts` Elemente wissen will. Zum Beispiel kann eine Client-Anfrage ein `Checks` Feld haben, das das `certification path building (cpb)` sowie den `WantBack id-stc-build-pkc-path` definiert. In diesem Fall enthält die Antwort keinen `Validation-Status`. Clients die ihre eigene `Validation Policy` verwenden wollen, können dann so eine Anfrage benutzen. Andere Clients, die die `Validation Schritte` völlig delegieren, können in ihren `Requests` das `Wantback` Feld weglassen. Dafür soll ein `Checks` Feld mit der Anforderung zur Pfadbildung, inklusive `Validation- und Revokations-prüfung`, Teil ihres `Requestobjekts` sein.

`WantBack ::= SEQUENCE SIZE (1..MAX) OF OBJECT IDENTIFIER`

Alle SCVP Server Implementationen müssen die `id-swb-pkc-cert` und `id-swb-pkc-public-key-info` unterstützen.

Implementierungen die `delegated path discovery (DPD)` zur Verfügung stellen, müssen die zwei Wantbacks: `id-swb-pkc-best-cert-path` und `swb-pkc-revocation-info` unterstützen.

Validationpolicy: Definiert die Client- (Wunsch) Policy, die der Server bei der Zertifikat-Validation anwenden soll. Sollte die Policy für irgendwelche Gründe nicht angewendet werden können, dann muss der Server einen Fehler zurückmelden. Eine `Validationpolicy` muss für alle notwendigen Parameter Standardwerte definieren. Sie darf auch für alle Parameter neue Werte erlauben.

Eine Client-Anfrage kann einfach gestaltet werden, indem ein `OID` zur Bestimmung des Algorithmus und alle notwendigen Parameter benutzt wird. Die Anfrage wird aber hingegen komplexer, wenn die Policy den Clients es erlaubt, dass einige Parameter in deren `Request` definiert werden. Sollte es ein Konflikt zwischen der referenzierten Policy und einem übermittelten Parameterwert im `Request` bestehen dann wird vom Server ein Fehler zurückgeliefert.

```
ValidationPolicy ::= SEQUENCE {
    validationPolRef          ValidationPolRef,
    validationAlg             [0] ValidationAlg OPTIONAL,
    userPolicySet             [1] SEQUENCE SIZE (1..MAX) OF OBJECT
                               IDENTIFIER OPTIONAL,
    inhibitPolicyMapping      [2] BOOLEAN OPTIONAL,
    requireExplicitPolicy     [3] BOOLEAN OPTIONAL,
    inhibitAnyPolicy          [4] BOOLEAN OPTIONAL,
    trustAnchors              [5] TrustAnchors OPTIONAL,
    keyUsages                 [6] SEQUENCE OF KeyUsage OPTIONAL,
    extendedKeyUsages         [7] SEQUENCE OF KeyPurposeId OPTIONAL,
    specifiedKeyUsages        [8] SEQUENCE OF KeyPurposeId OPTIONAL }
```

```
ValidationPolRef ::= SEQUENCE {
    valPolId                  OBJECT IDENTIFIER,
    valPolParams              ANY DEFINED BY valPolId OPTIONAL }
```

Clients können den Server auffordern die Default Validation Policy, oder eine andere Policy zu benutzen. Die Default Validation Policy entspricht der Standard `certification path processing` wie sie in den RFC 3280 definiert wurde. Die Standardwerte können Out of Band oder durch ein Policy Request-Response Paar, veröffentlicht werden. Die OID für die Default Validation Policy hat den Wert:

```
id-svp OBJECT IDENTIFIER ::= { iso(1) identified-organization(3)
    od(6) internet(1) security(5) mechanisms(5) pkix(7) 19 }

id-svp-defaultValPolicy OBJECT IDENTIFIER ::= { id-svp 1 }
```

Die Default Validation Policy muss den basic Validation Algorithm als ihren default Validation Algorithmus verwenden. Server können so Konfiguriert werden, dass sie eine Error-Response für alle Requests mit Default Validation Policy zurückliefern, damit eine Lokale Policy Verwendung findet.

Das `responseFlags` Feld gibt den Clients die Möglichkeit, optionale Features für die Response anzufordern.

Die `serverContextInfo` weist den Server auf, dass der Client ergänzende Informationen von vorherigen Requests-Responses erwartet.

`ValidationTime` teilt dem Server mit, zu welchem Datum bzw. Zeitpunkt die Checks stattfinden sollen.

`intermediateCerts` (wenn vorhanden), enthält mindestens ein Zertifikat. Diese(s) Zertifikat(e) könnte(n) dann vom Server bei der Zertifikatspfadbildung benutzt werden.

```
CertBundle ::= SEQUENCE SIZE (1..MAX) OF Certificate.
```

`revInfos`: Stellt Revokationsinformationen wie CRLs, Delta CRLs¹⁰ oder OCSP Responses zur Verfügung. Auf diese Informationen kann dann beim Validieren zugegriffen werden. Das Ziel dieses Felds ist es, dem Server die Revokationsinformationen zu liefern, falls er selbst nicht in der Lage sein sollte, auf die Revokationsquellen zuzugreifen. Server können die `RevInf` außer Acht lassen, z.B. falls sie die Zertifikate, die mit den `RevInf` assoziiert worden sind, gar nicht in den Pfad gebrauchen.

¹⁰RFC3280

```
RevocationInfos ::= SEQUENCE SIZE (1..MAX) OF RevocationInfo
```

```
RevocationInfo ::= CHOICE {  
    crl                [0] CertificateList,  
    delta-crl         [1] CertificateList,  
    ocsp              [2] OCSPResponse,  
    other             [3] OtherRevInfo }
```

```
OtherRevInfo ::= SEQUENCE {  
    riType              OBJECT IDENTIFIER,  
    riValue            ANY DEFINED BY riType }
```

ProducedAt: Ein Client kann dem Server erlauben, eine vorige zwischengespeicherte SCVP-Response zu benutzen. In einem solchen Szenario, benutzt der Client dieses Feld um die Aktualität der zwischengespeicherten Responses zu bestimmen. Also wird durch dieses Feld der früheste Zeitpunkt bestimmt, an dem eine cached Response noch akzeptiert wird. Der Wert der `ProducedAt` ist unabhängig von `Validation time`.

3.2.5 Server Policy Request

Eine `ValPolRequest` wird von Clients eingestzt um Informationen über Policies und Konfigurationsinformationen, einschliesslich eine Liste vom `Validation policies`, die der Server unterschützt, zu erfragen.

Analog zur `CVRequest`, wird eine `ValPolRequest` auch in eine MIME Nachricht gekapselt. Also besteht auch hier die Anfrage aus einer `ValPolRequest` gekapselt in ein `ContentInfo`.

```
ContentInfo {  
    contentType        id-ct-scvp-valPolRequest,  
    content            ValPolRequest  
}
```

Der Syntax vom `ValPolRequest` sieht wie folgt aus:

```
ValPolRequest ::= SEQUENCE {  
    vpRequestVersion      INTEGER DEFAULT 1,  
    requestNonce          OCTET STRING }
```

3.2.6 Validation Response

Eine SCVP Validation Response (`CVResponse`) hat analog zu `CVRequest` auch als Single Response sein. Wenn die Response als Mime Nachricht gekapselt ist, wird der `contentType: application/scvp-cv-response` benutzt. Es gibt verschiedene Formen einer Response (`AuthenticatedData` wird hier nicht betrachtet), nämlich:

1. Eine erfolgreiche bzw. gelungene Response zur einer `CVRequest`-Nachricht, die das `protectResponse` Element zu `false` gesetzt hat. Diese Response muss nicht vom Server geschützt (`protected`) werden. Anderenfalls müssen alle anderen erfolgreichen Responses geschützt werden. Sollte jedoch der Server nicht im Stande sein die erfolgreiche Response in `protected` Form zu liefern, dann muss er eine Fehler-Response zum Client zurückschicken.
2. Eine Fehler-Response soll auch in `protected` Form sein, ausser das `protectResponse` Element im Request zu `false` gesetzt ist.

Eine Response ist dann erfolgreich, wenn der Server den Zertifikatspfad unter Verwendung der referenzierten oder gelieferten Validation Policy erfolgreich bauen konnte, sowie alle in dem Request enthaltenen Parameter im Sinne des SCVP-Protokoll bearbeiten und benutzen konnte.

Der Server muss denselben Schutzmechanismus wie der Client verwenden. Also für `Authenticated-Request` eine `Authenticated-Response`, und für `Signed-(Request/Response)` liefern.

Eine unprotected CVResponse sieht wie folgt aus:

```
ContentInfo {
    contentType      id-ct-scvp-certValResponse,
                   -- (1.2.840.113549.1.9.16.1.11) --
    content          CVResponse }
```

Die Syntax einer CVResponse ist:

```
CVResponse ::= SEQUENCE {
    cvResponseVersion      INTEGER,
    serverConfigurationID  INTEGER,
    producedAt             GeneralizedTime,
    responseStatus         ResponseStatus,
    respValidationPolicy   [0] RespValidationPolicy OPTIONAL,
    requestRef             [1] RequestReference OPTIONAL,
    requestorRef           [2] GeneralNames OPTIONAL,
    requestorName          [3] GeneralNames OPTIONAL,
    replyObjects           [4] ReplyObjects OPTIONAL,
    respNonce              [5] OCTET STRING OPTIONAL,
    serverContextInfo      [6] OCTET STRING OPTIONAL,
    cvResponseExtensions  [7] Extensions OPTIONAL,
    requestorText          [8] UTF8String (SIZE 1..256 OPTIONAL)}
```

Es wird auf die Bedeutung von den optionalen Feldern nicht eingegangen¹¹

cvResponseVersion: Syntax und Semantik von der cvResponseVersion haben denen der cvRequestVersion zu entsprechen. Sollte der Server nicht in der Lage sein, seine Response für die Verlangte Version zu erstellen, dann muss der Server eine Fehler-Antwort mit der höchsten unterschützten Version zum Client zurückschicken

serverConfigurationID: Ein Integer-Wert, der die aktuelle Version der Serverkonfiguration, die dann durch validationPolicies, validationAlgs,

¹¹siehe RFC5055

`authPolicies`, `clockSkew` und `defaultPolicyValues` repräsentiert wird. Sollte einer dieser Werte geändert werden dann muss der Server eine neue `ValPolResponse` mit einer neuen `serverConfigurationID` erstellen

`producedAt`: Gibt das Datum und die Zeit an der der SCVP Server die Response generiert hat. Der `producedAt` muss als UTC¹² dargestellt werden. Der `producedAt` ist unabhängig von Validation-Zeit.

`responseStatus`: Gibt dem SCVP Client Status-Informationen über seine Request. Der `responseStatus` hat einen numerischen Status Code und ein Optionalen String¹³. Der String wird benutzt um den Status von Menschen lesbar zu machen.

```
ResponseStatus ::= SEQUENCE {
    statusCode          CVStatusCode DEFAULT okay,
    errorMessage       UTF8String OPTIONAL }
```

```
CVStatusCode ::= ENUMERATED {
    okay                (0),
    skipUnrecognizedItems (1),
    tooBusy             (10),
    invalidRequest      (11),
    internalError       (12),
    badStructure        (20),
    unsupportedVersion   (21),
    abortUnrecognizedItems (22),
    unrecognizedSigKey   (23),
    badSignatureOrMAC    (24),
    unableToDecode       (25),
    unsupportedChecks    (27),
    unsupportedWantBacks (28),
    unsupportedSignatureOrMAC (29),
    invalidSignatureOrMAC (30),
```

¹²Universal Time Coordinated

¹³ISO/IEC 10646-1 character set encoded with UTF-8

```
protectedResponseUnsupported      (31),
unrecognizedResponderName         (32),
relayingLoop                      (40),
unrecognizedValPol                (50),
unrecognizedValAlg                (51),
fullRequestInResponseUnsupported (52),
fullPolResponseUnsupported        (53),
inhibitPolicyMappingUnsupported   (54),
requireExplicitPolicyUnsupported   (55),
inhibitAnyPolicyUnsupported        (56),
validationTimeUnsupported         (57),
unrecognizedCritQueryExt          (63),
unrecognizedCritRequestExt        (64) }
```

Status Codes zwischen 0 und 9 sind reserviert für Codes, die anzeigen, dass die Request erfolgreich verarbeitet wurde und dementsprechend in der erfolgreichen Response gesendet wird. Die Codes 10 und höher bezeichnen einen Fehler.

4 Entwurf und Implementierung in Java

In diesem Kapitel wird eine SCVP-Server Implementierung auf Java Basis beschrieben, dabei wird auf die einzelnen Komponenten [Abb. 4.1] eingegangen. Bei diesem Prototyp handelt es sich um eine Webanwendung, die aus drei Hauptkomponenten besteht:

- einem WebServer, der als Applikation Server dient
- einem Datenbank-Server für die persistente Datenhaltung und
- einer SCVP-Kern-Anwendung, die dann als Controller oder Steuerungseinheit für die ankommenden Requests und deren Bearbeitung fungiert

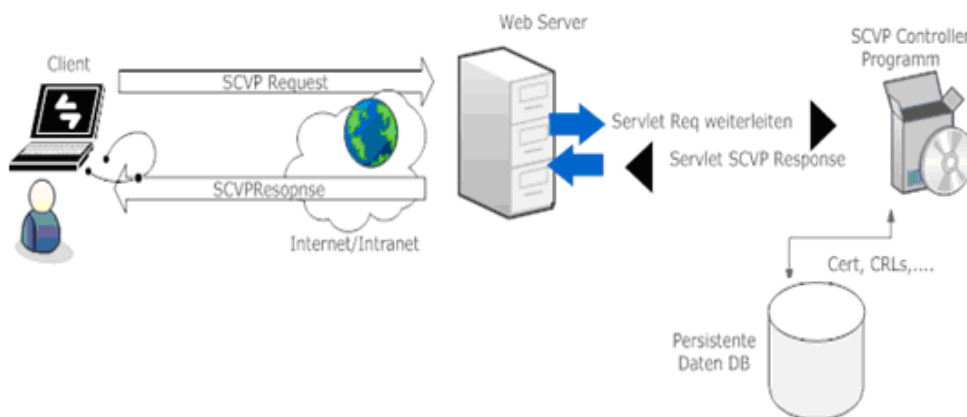


Abbildung 4.1: Komponenten des SCVP-Prototyps

4.1 Webserver

Für den WebServer wurde den Apache-Tomcat-6.0.20 eingesetzt. Dieser ist eine Referenzimplementierung der Servlet-Spezifikation. Dadurch bietet sich für die Anwendung die Möglichkeit Servlets zu verwenden. Für detaillierte Informationen über die Möglichkeiten des Tomcat-Containers sei auf seine Dokumentation¹ verwiesen.

Der apache-tomcat stellt im Bezug auf die Sicherheit eine Sicherheitsstrategie, die in zwei Teile unterteilt werden kann, nämlich in einen deklarativen und einen programmatischen Teil. Der Programmatische Teil ist Optional und kann als eine „second line of defense²“ angesehen werden. Im Verzeichnis `<CATALINA_HOME>/conf/` sind die folgenden vier Dateien, die in diesem Zusammenhang die deklarativen Teil (Konfiguration) definieren: `web.xml`, `server.xml`, `catalina.policy` und `catalina.properties`. In diesen Dateien werden die Sicherheitspolitik des Tomcats definiert und die Standardinstellungen, die für alle Webanwendungen gelten vorgenommen.

Die Zugriffskontrolle in Apache Tomcat ist rollenbasiert. Zuerst erfolgt die Übertragung der Authentisierungsdaten, dann finden eine Kontrolle dieser Daten und letztlich die Zuweisung einer Rolle oder die Ausgabe einer Fehlermeldung statt.

An dieser Stelle ist zu erwähnen, dass Sicherheitsuntersuchungen des Tomcat Servlet Containers, laut Bundesamt für Sicherheit in der Informationstechnik (BSI-2006), einige Sicherheitslücken ergab. Auf der Seite der Tomcat-Entwickler ist davon die Rede, dass diese mit dem Release 6.0.18 behoben worden sind.

Die Kommunikationssicherheit in dem hier eingesetzten SCVP-Webserver ist hart-codiert implementiert, d.h. es wurde auf die Sicherheitsfunktionen von Tomcat verzichtet und für benötigte Schritte eigene Programmierung bevorzugt. Es wird daher in diesem Dokument weder auf die Sicherheitsstrategie noch auf die Sicherheitslücken des Tomcats näher eingegangen.

¹<http://tomcat.apache.org/>

²zweite Abwehrfront

Das Prüfen der Signierten Requests (`SignedData`) sowie das Signieren der Response wird nicht vom WebServer sondern direkt von der Anwendung durchgeführt.

Wie die Requests im einzelnen vom Server bzw. Servlet bearbeitet werden, wird weiter unten in diesem Kapitel diskutiert.

4.2 Die Datenbank-Komponente

4.2.1 DB und DB-Schema

Für die persistente Datenhaltung über ein Datenbanksystem wurde auf Derby³ zurückgegriffen. Derby ist ein Datenbanksystem, das rein in Java implementiert wurde. Dieses kann in zwei Varianten verwendet werden: im Client-Server oder im eingebetteten Modus. Derby kann auch von anderen, also nicht nur von Java basierten Frameworks angesprochen werden, daher umfasste es vom Anfang an eine Unterstützung der SQL-ANSI Standards⁴.

Derby ist mit Java 1.3 entwickelt worden. Derby läuft auf fast allen Versionen der Java-Plattformen J2ME⁵, J2SE⁶, J2EE⁷ und auch OSGi⁸-Plattformen.

Derby ist stark modular aufgebaut. Dadurch ergibt sich der Vorteil, je nach Anwendungsfall, nur die notwendigen Bibliotheken anzubinden.

Die wichtigsten Bibliotheken und Hilfsprogramme sind:

- derby: Datenbankkern
- derby-Tools: Werkzeuge ij, sysinfo und dblook
- derbynet: Netzwerkservers
- derbyclient: DRDA-Netzwerkclient und JDBC-Treiber

³<http://db.apache.org/derby/>

⁴<http://www.ansi.org>

⁵Java Platform, Micro Edition

⁶Java Platform, Standard Edition

⁷Java Platform, Enterprise Edition

⁸Open Services Gateway initiative

Dazu bietet Derby verschiedene Sicherheits- und Verschlüsselungsverfahren. Außerdem sticht Derby mit seiner einfachen Administration und dem geringen Ressourcenverbrauch (600 KB bis 2 MB auf der Platte).

Die für den hier entwickelten SCVP-Server verwendete Datenbank ist eine in der Client-Server Ausprägung eingestellten Derby-DB. Dies erhöht die Erreichbarkeit des Derby-Datenbanksystems durch die Unterstützung von den üblichen Client-Serverfunktionalitäten. Die Verbindung zum DB-Server wird also über TCP/IP mittels des Standardprotokolls (DRDA)⁹ aufgebaut.

Die Standalone Datenbank soll von anderen Applikationen und/oder Verantwortlichen Administriert und aktuell gehalten werden, zum Beispiel um CRLs auf den neusten Stand zu halten oder neue Zertifikate hinzuzufügen.

Es handelt sich bei dem benötigten DB-Schema um zwei Entitäten, nämlich `certificates` und `CRLs`. Diese sind Zwei verknüpfte Tabellen, für eine persistente Datenhaltung der Zertifikate und der Revokationsinformationen.

Die Tabelle `certificat` hat Folgende Spalten, dabei ist (°) als eine Konkatenation bzw. Verkettung zu verstehen:

Spalten Name	Daten Typ	Null	Wert
Key (primary key)	VARCHAR	unerlaubt	Issuer Name°serialNumber
IssuerName	VARCHAR	unerlaubt	Issuer Name des Zertifikates
SubjectName	VARCHAR	unerlaubt	Subject Name des Zertifikates
notBefore	Date	unerlaubt	Not Before Date in cert
notAfter	Date	unerlaubt	Not After Date in cert
CERT	BLOB	unerlaubt	cert
isCA	VARCHAR	unerlaubt	y/n als Boolean Kodierung
isAnchor	VARCHAR	unerlaubt	y/n als Boolean Kodierung

Ein Zertifikat mit dem Wert 'Y' in der Spalte `isCA`, kann einen Datensatz mit den eventuellen von diesen CA erzeugten Revokationsinformationen in der `CRLs` Tabelle haben. Diese Revokationsinformationen können dann in einem späteren Schritt beim Validieren verwendet werden.

⁹Distributed Relational Database Architecture

4.2.2 Die Datenbankverbindung

Jeder Zugriff bzw. Interaktion mit der Datenbank wird über eine Instanz der Singleton¹⁰ Klasse `DBconnection` verarbeitet.

Die Datenbank Connection wurde nicht standardmäßig in der `init()` Methode des Servlets registriert, sondern es wird erst beim Gebrauch einer solchen Connection instanziiert.

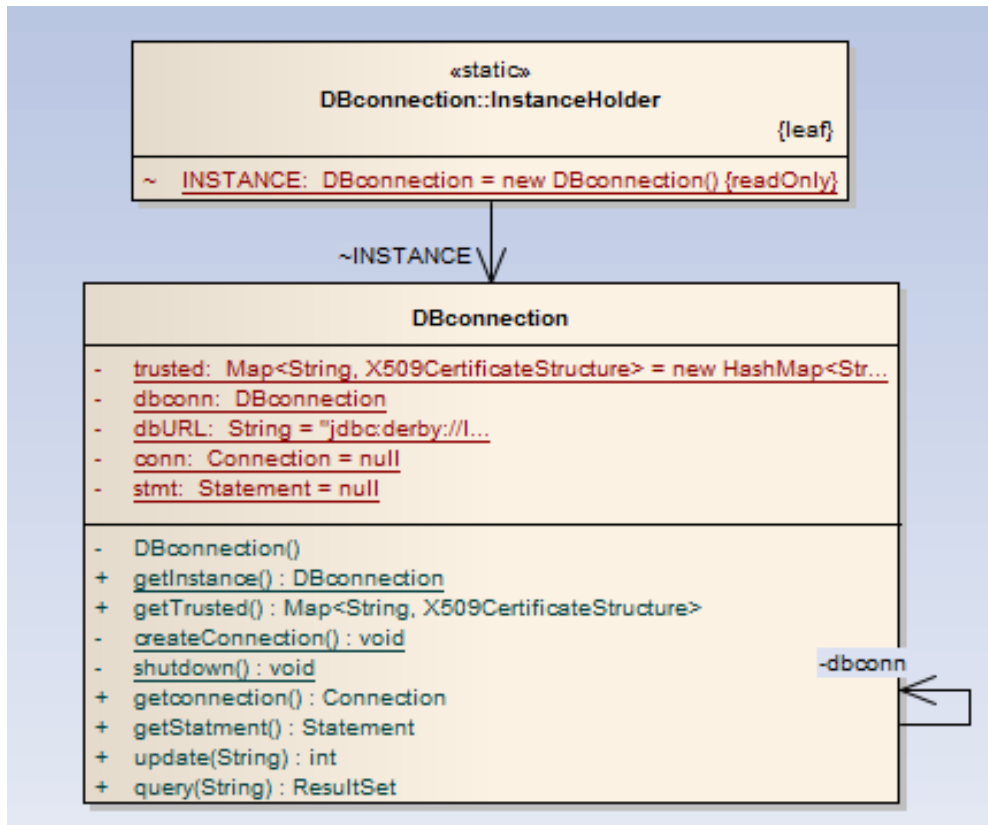


Abbildung 4.2: Die Klasse zur Verbindung mit der Derby Datenbank

Für eine bessere Übersicht der Implementierung wird im diesem Kapitel das Folgende Schema verwendet; eine erklärende Textbeschreibung gefolgt von dem passenden-Code

¹⁰Bei diesem Entwurfsmuster wird verhindert, dass von einer Klasse mehr als ein Objekt erzeugt werden kann

Fragment. Die Klasse `DBconnection` hat einen privaten Konstruktor, der durch eine „Lazy Creation“¹¹ einer Instanz dieser Klasse indirekt aufgerufen wird. Das eigentliche Instanzieren geschieht beim Aufruf der inneren Klasse `InstanceHolder`, die dann den privaten Konstruktor aufruft.

Dies hat den Vorteil, dass das Initialisieren der Klassenvariablen implizit vom `ClassLoader` synchronisiert wird. Diese „Initialize-On-Demand Holder“¹² Klasse ist Thread-sicher. Das ist auch eine Lösung gegen potentielle Performanz Defizite, die beim Einsatz von den alternativen Synchronisationsmechanismen `synchronized` entstehen kann.

```
private static Connection conn = null;
private static Statement stmt = null;
private static DBconnection dbconn;

private DBconnection() {}

private static final class InstanceHolder {
    static final DBconnection INSTANCE = new DBconnection();
}

public static DBconnection getInstance(){
    dbconn = InstanceHolder.INSTANCE;
    createConnection();
    return dbconn;
}
```

In der privaten statischen Methode `createConnection()` wird dann der `DBconnection` Objekt erzeugt und initialisiert.

```
Class.forName("org.apache.derby.jdbc.ClientDriver").newInstance();
conn = DriverManager.getConnection(dbURL);
```

Dieser kann dann später mittels einer getter-Methode abgefragt werden

```
public Connection getConnection(){
    return conn;
}
```

¹¹Das einzige Objekt der Singleton Klasse wird erst dann erzeugt, wenn es benötigt wird

¹²Eine Alternative, welche die Instanz erst beim ersten Aufruf von `getInstance()` erzeugt

```
}
```

Dasselbe gilt dann auch für das Statement-Objekt:

```
public Statement getStatement() throws SQLException{
if (stmt==null)
    stmt=conn.createStatement();
return stmt;
}
```

Für insert-, delete- und update-Operationen ist die Methode update() zuständig.

```
public synchronized int update(String expression)throws SQLException {
    getStatement();
    int i = stmt.executeUpdate(expression); // run the query
    if (i == -1) {
        throw new SQLException("db error with
                                + expression: "+expression);
    }
    stmt.close();
    return i;
}
```

Für die select Operationen wird die Methode query() benutzt. Dasselbe Statement Objekt kann hierbei mehrfach verwendet werden. Um Leseanomalien zu vermeiden wurde das Schlüsselwort synchronized benutzt .

```
public synchronized ResultSet query(String expression)throws
                                SQLException{
    Statement st = conn.createStatement();
    return st.executeQuery(expression);
}
```

Das resultierende ResultSet ist die Rückgabe der Methode. Da beim Schließen des Statement Objekts das zugehörige ResultSet Objekt auch geschlossen wird, sollte

die `closeStatement` Methode erst nach der kompletten Analyse des resultierenden `ResultSets` **aufgerufen**.

```
public void closeStatement() {
    if (stmt != null)
        try {
            stmt.close();
        }
        catch (SQLException e) {
            e.printStackTrace(); } } }
```

Der Vollständigkeit halber wurden die Methoden `closeConnection`, `closeStatement` und `shutdown` implementiert, und zwar um die Attributsvariablen `Connection` und `Statement` aufzuräumen.

```
public void closeConnection() {
    if (conn != null)
        try {
            DriverManager.getConnection(dbURL + ";shutdown=true");
            conn.close();
        }
        catch (SQLException e) {
            e.printStackTrace(); }
    }
```

```
public void closeStatement() {
    if (stmt != null)
        try {
            stmt.close();
        }
        catch (SQLException e) {
            e.printStackTrace(); }
    }
```

```
public void shutdown() {
    closeStatement();
    closeConnection(); }
```

4.3 SCVP-ASN1-Modul: implementierte Klassen

4.3.1 BouncyCastle

Bouncy Castle¹³ ist eine Opensource Implementierung der Java Cryptography Extension (JCE), sie unterstützt Unter anderem S/MIME, CMS, PKCS7, TEA, XTEA, SHA224, X.509 Zertifikate. Dazu enthält sie eine key-API, die nicht von der JCE abhängig ist. Ferner stellt Bouncycastle einige andere Bibliotheken zur Verfügung.

Für einen kurzen Überblick werden die Komponenten der Crypto-API von Bouncy Castle aufgelistet:

- A lightweight cryptography API for Java and C#.
- A provider for the Java Cryptography Extension and the Java Cryptography Architecture.
- A clean room implementation of the JCE 1.2.1.
- A library for reading and writing encoded ASN.1 objects.
- A light weight client-side TLS API.
- Generators/Processors for S/MIME and CMS¹⁴.
- Generators/Processors for OCSP¹⁵.
- Generators/Processors for TSP¹⁶.
- Generators for Version 2 X.509 attribute certificates.
- A signed jar version suitable for JDK 1.4-1.6 and the Sun JCE.
- Generators/Processors for OpenPGP¹⁷.
- Generators for Version 1 and Version 3 X.509 certificates, Version 2 CRLs, and PKCS12 files.

¹³<http://www.bouncycastle.org/>

¹⁴PKCS7/RFC 3852

¹⁵RFC 2560

¹⁶RFC 3161

¹⁷RFC 2440

4.3.2 Die Basis-Klasse ASN1Encodable:

Die Bouncycastle-Bibliothek zum Lesen und Schreiben von ASN1-Objekten stellt eine abstrakte Basis Klasse: `ASN1Encodable` für Objekte zur Verfügung, die direkt von/in ASN.1 Output- bzw. Input-Streams geschrieben und/oder gelesen werden können.

`ASN1Encodable` implementiert wiederum die Schnittstelle `DEREncodable`.

Von dieser Klasse erben, entweder direkt oder indirekt, alle für den Entwurf implementierten ASN1-Modul Klassen. Die wichtigsten Methoden von `ASN1Encodable` werden im Folgenden kurz erläutert:

- `public byte[] getEncoded()`: gibt entweder die DER- oder Ber-Kodierung von dem aufrufenden Objekt.
- `public int hashCode()`: gibt den hash code der ASN.1 Struktur des aufrufenden Objekts.
- `public boolean equals()`: es ist true wenn die verglichenen Objekte die gleiche ASN.1 Struktur und Werte haben.
- `public byte[] getEncoded(String encoding)`: gibt die als String angegebene Kodierung des aufrufendes Objekts.
- `public byte[] getDEREncoded()`: gibt die DER-Kodierung oder, wenn die DER Kodierung nicht erfolgreich durchgeführt werden konnte, null zurück.
- `public abstract DERObject toASN1Object()`: muss von den Sub-Klassen überschrieben werden, um die richtige Rückgabe des jeweiligen Objekts in der spezifizierten ASN.1 Struktur zu ermöglichen.

Der package `org.bouncycastle.asn1` beinhaltet zusätzlich auch Hilfsklassen zum Generieren, Parsen, und Tokenizen von ASN1-Objekte, ferner beinhaltet er weitere Klassen, die beim Bearbeiten von ASN.1 Objekte eine Hilfe sein können. Ein Beispiel dieser Klassen sind: `ASN1TaggedObject`, `ASN1Sequence`, `DERSequence`, `DERObject`, `DERBitString`, `DERBoolean`, `DERGeneralizedTime`,

`DERObjectIdentifier`, `BERSequenceParser`, ..etc.

Der package `org.bouncycastle.asn1` enthält andere Packages, die eine Implementierung von anderen - für Kryptographische Zwecke wichtigen - ASN.1 Module und Schnittstellen zur Verfügung stellen. Z.B. enthält der package `org.bouncycastle.asn1.cms` die Bouncycastle Implementierung von den von Cryptographic Message Syntax¹⁸ definierten ASN.1 Modul.

Das CMS beschreibt eine Kapselung Syntax für den Datenschutz. Es unterstützt digitale Signaturen und Verschlüsselung. Die Syntax erlaubt mehrere encapsulations; ein Kapselung Umschlag kann verschachtelt werden in einen anderen. Ebenso kann eine Partei einige zuvor gekapselten Daten digital signieren. Darüber hinaus können beliebige Attribute, wie die Unterzeichnungszeit, mit dem Inhalt der Nachricht unterzeichnet und zusammen gepackt werden. Einige Klassen dieses Moduls, die in diesem Arbeit Verwendung finden sind: `ContentInfo`, `SignedAttrs`, `EncapsulatedContentInfo`, `SignerInfo`, etc.

4.3.3 SCVP-ASN1-Modulklassen:

Das in dem SCVP RFC 5055 definierte SCVP ASN1-Modul spezifiziert den Aufbau von SCVP-Datenstrukturen. Dieses Modul umfasst insgesamt 58 Daten Typen bzw. Strukturen, die bei der Implementierung des Prototyps in Java-Klassen umgesetzt wurden. Einige dieser Strukturen wie `CVRequest`, `query`, `requestorRef`, etc, wurden im Abschnitt 3.2.4 schon vorgestellt.

Das Package `asn1.cdc.tu.darmstadt.SCVP` umfasst die 58 Klassen, deren jeweilige Erläuterung den Rahmen dieses Dokument sicherlich sprengen würde. Es sei aber darauf hingewiesen, dass sie alle folgende Merkmale gemeinsam haben:

- sie erben alle direkt von der Klasse `ASN1Encodable`
- statische Methode `getInstance` für die „lazy creation“ der Instanz
- getter Methoden zur Rückgabe der privaten Attributvariablen

¹⁸CMS RFC 3852 ist von PKCS7-Version 1.5 RFC 2315 abgeleitet.

- Methode `toASN1Object` zur Rückgabe der jeweiligen Instanz ASN1-Struktur

4.3.3.1 Beispielklasse „Query“

Eine der aufwändigsten unter den 58 Klassen ist die Klasse `Query` [Abb 4.3], da sie eine etwas komplizierte Struktur hat.

Sie beinhaltet zwei nicht Optionale Objekte gefolgt von einem Tagged optionalen Objekt, dann von einem nicht Optionalen, das wiederum von einem optionalen nicht Tagged Objekt gefolgt ist. Die restlichen sind dann alle optional und vom Typ Tagged. Hier sind mehrere Fallunterscheidungen erforderlich sind. Es wird auf mehr Details dieser Klasse stellvertretend eingegangen.

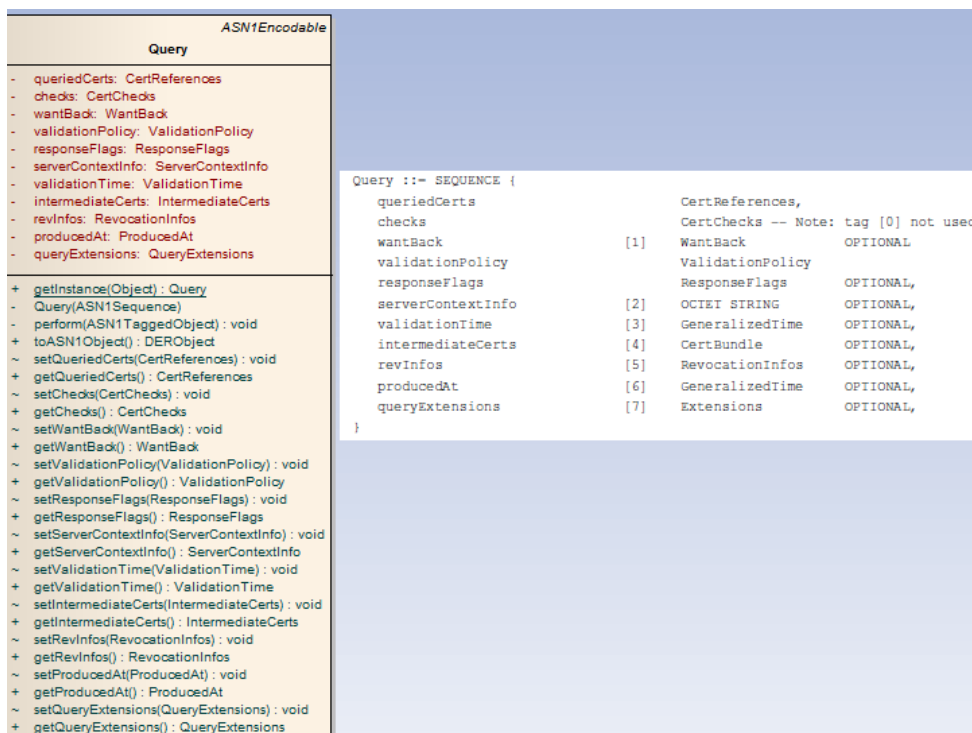


Abbildung 4.3: UML und ASN1 Struktur der Klasse „Query“

In Falle der Query gibt die Methode `toASN1Object` eine Sequenz zurück. Dafür wird ein `ASN1EncodableVector` mit den Komponenten der Query gefüllt, und dann als Konstruktor Parameter für eine `DERSequence`, die die Query als Sequenz darstellt.

Da es sich hier um eine Sequenz handelt, muss die Reihenfolge der Query Komponenten beachtet werden. Ferner muss sowohl beim Extrahieren aus einer Sequenz als auch beim Erzeugen der darstellenden Sequenz, die Präsenz der optionalen Bestandteile geprüft werden.

```
public DERObject toASN1Object() {
    ASN1EncodableVector v = new ASN1EncodableVector();
    v.add(queriedCerts);
    v.add(checks);
    if (wantBack!=null)
        v.add(wantBack);
    v.add(validationPolicy);
    if (responseFlags!=null)
        v.add(responseFlags);
    if (serverContextInfo!=null)
        v.add(serverContextInfo);
    if (validationTime!=null)
        v.add(validationTime);
    if (intermediateCerts!=null)
        v.add(intermediateCerts);
    if (revInfos!=null)
        v.add(revInfos);
    if (producedAt!=null)
        v.add(producedAt);
    if (queryExtensions!=null)
        v.add(queryExtensions);
    return new DERSequence(v);
}
```

Die `getInstance`-Methode `public static Query getInstance(Object obj)` bekommt als Argument ein Objekt vom Typ `java.lang.Object`. Die Zugehörigkeit dieses Objekts wird mit Hilfe von `instanceof` getestet. Ist das getestete Objekt eine Instanz der Klasse `Query` oder weist es den Wert `null` auf, dann ist die zu erzeugende `Query` genau dieses Objekt gecastet `Query`.

Ist das Objekt hingegen ein Exemplar einer Sequenz dann wird der private Konstruktor der Klasse `Query` aufgerufen zur weiteren Bearbeitung dieser Sequenz.

Ist aber der Wert des übergebenen Objekts ein anderer als `null` und weder vom Typ `Query` noch vom Typ `ASN1Sequence` dann wird ein `Exception` Objekt geworfen.

```
public static Query getInstance(Object obj){
```

```

    if (obj instanceof Query || obj == null) {
        return (Query) obj;
    }
    else if (obj instanceof ASN1Sequence) {
        return new Query((ASN1Sequence) obj);
    }
    else
        throw new IllegalArgumentException("unknown Object in factory"
            + obj.getClass().getName());
}

```

Im Minimalfall, wo die Query lediglich aus nicht optionalen Objekte besteht, muss sie drei Elemente besitzen.

An erster Stelle ist ein CertReferences Objekt danach zweite Objekt in der Sequenz muss es vom Typ CertChecks geben.

An der dritten Stelle muss getestet werden, ob es sich um ein Tagged Objekt handelt: wenn ja dann wird noch der Tag geprüft. Wenn es den erwarteten Wert nämlich 1 hat, dann wird der Inhalt dieses Tagged Objekts als Übergabe Parameter der getInstance-Methode der WantBack-Klasse in einem entsprechenden Aufruf übergeben. Wird ein anderer Tag Wert als 1 gefunden dann ist mit einem Error zu rechnen. Diese Schritte lassen sich besser mit Hilfe des Codes erklären:

```

private Query(ASN1Sequence seq) {

    if (seq.size() > 2) {
        setQueriedCerts(CertReferences.getInstance(seq.getObjectAt(0)));
        setChecks(CertChecks.getInstance(seq.getObjectAt(1)));
        if (seq.getObjectAt(2) instanceof ASN1TaggedObject) {
            ASN1TaggedObject obj = (ASN1TaggedObject) seq.getObjectAt(2);
            if (obj.getTagNo() == 1)
                setWantBack(WantBack.getInstance(obj.getObject()));
        }
        // Vierte Stelle ist kein optionaler Wert. Keine Tests notwendig.
        setValidationPolicy(ValidationPolicy.getInstance(seq.getObjectAt(3)));
        // weiter mit den 5 Stellen, wenn diese existiert.
        if (seq.size() > 3) {
            // Wenn Ja und das ist da kein tagged Objekt
            if (!(seq.getObjectAt(4) instanceof ASN1TaggedObject)) {
                // dann muss das an der 5 Stelle ein ResponseFlags sein
                setResponseFlags(ResponseFlags.getInstance(seq.getObjectAt(4)));
            }
            //und dann die Elemente ab der 6 Stelle bearbeiten mit perform(tagged o)
            //da alle restlichen Elemente Taged Obj sein müssen.
        }
    }
}

```

```

        for (int i = 5; i < seq.size(); i++)
            perform((ASN1TaggedObject) seq.getObjectAt(i));
    }
    //Nein, der 5 Element ist doch ein Tagged Objekt, dann fehlt der
    //ResponseFlags Obj aus. Und weiter mitbearbeiten mit perform(tagged o)
    else{
        for (int i = 4; i < seq.size(); i++)
            perform((ASN1TaggedObject) seq.getObjectAt(i));
        }
    }
}
//Nein, an der dritte Stelle ist kein tagged (Optional Wantback fehlt)
//dann ist ein ValidationPolicy (nicht Optional)
else {

    setValidationPolicy(ValidationPolicy.getInstance(seq.getObjectAt(2)));
    if (seq.size() > 3) {
        if (!(seq.getObjectAt(3) instanceof ASN1TaggedObject)) {
            setResponseFlags(ResponseFlags.getInstance(seq.getObjectAt(3)));
        }
        //Und dann weiter mit den Elemente ab 5 Stelle, wenn welche existieren.
        for (int i = 4; i < seq.size(); i++)
            perform((ASN1TaggedObject) seq.getObjectAt(i));
        }
    else {
        for (int i = 3; i < seq.size(); i++)
            perform((ASN1TaggedObject) seq.getObjectAt(i));
    }
}
}
}
}
else
    throw new IllegalArgumentException("bad Sequenz size: "+ seq.size());
}

```

Die `perform` Methode `private void perform(ASN1TaggedObject objectAt)` ist für die Bearbeitung der restlichen sechs Elemente der Query zuständig. Sie bekommt ein Tagged Objekt, prüft sein Tag Wert, und anhand dessen initialisiert sie dann die zugehörige Optionale Instance Variable mit dem Inhaltsobjekt diesen übergebenen Tagged Objekt. Die Prüfung des Tag Wertes bzw. die Fallunterscheidungen ist durch switch case Anweisungen implementiert.

4.3.3.2 Beispiel einer CVRequest

Es werden signierte und unsignierte requests generiert, ausserdem werden Requests mit Zertifikate und Requests mit Referenzen auf Zertifikate erzeugt. Es ist bei dieser Beispielanfrage von einem Zertifikat auszugehen, das aus einer Datei gelesen wird.

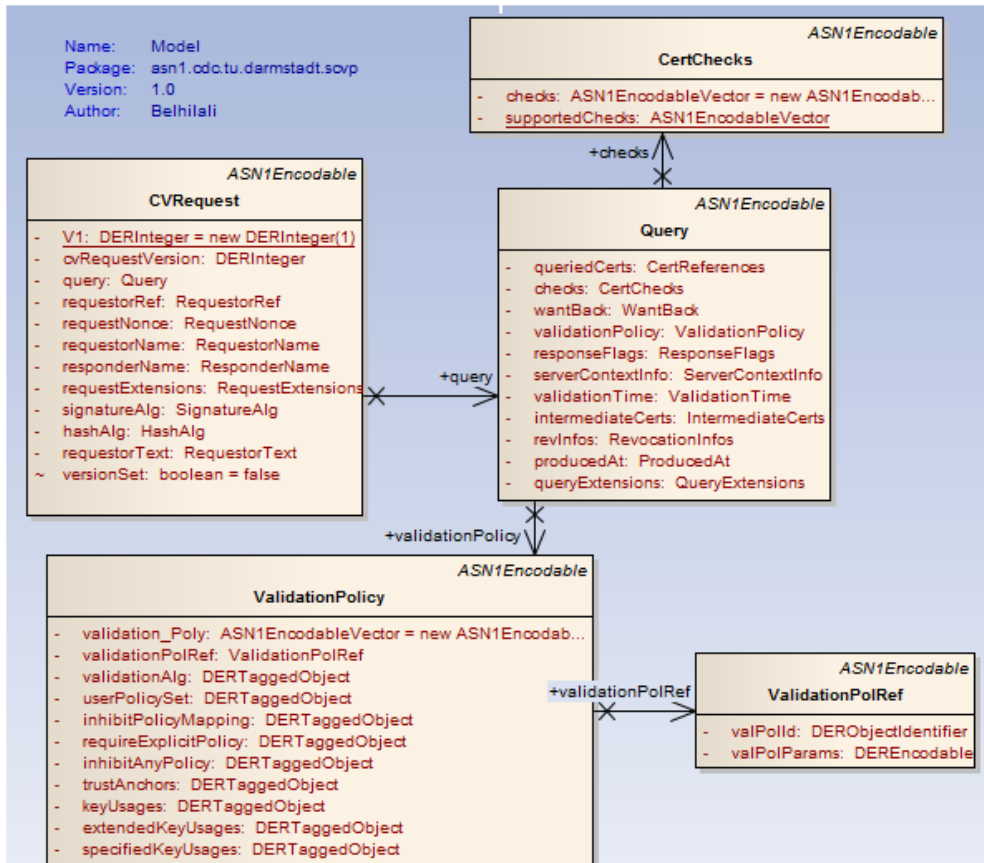


Abbildung 4.4: CVRequest und einige Abhängigkeiten (UML)

Es wird ein SCVPCertID-Objekt mit der statischen Methode generateFromX509Cert generiert. Diese Methode wurde implementiert, um für Testzwecke eine Client-Anfrage zu erzeugen.

Dafür wird als erstes ein `SCVPIssuerSerial` wie folgt erzeugt:

```
ASN1EncodableVector v = new ASN1EncodableVector();
v.add(cert.getIssuer()); //cert ist das ausgelesene Zertifikat
v.add(cert.getSerialNumber());
SCVPIssuerSerial issuerSerial = SCVPIssuerSerial.getInstance(
    new DERSequence(v));
```

danach wird den SHA-1¹⁹ Wert des Zertifikates berechnet und in einem vorher erzeugten Vektor addiert.

```
MessageDigest md;
try {
md = MessageDigest.getInstance("SHA-1");// Default sha-1
md.update( cert.getDEREncoded() );
}
v1.add(new DEROctetString(md.digest()));
```

als letztes wird das `AlgorithmIdentifier` mit dem oid (1.3.14.3.2.26) von SHA-1 instanziiert.

Aus dem erzeugten `SCVPCertID` wird eine `PKCReference` generiert. Es handelt es sich eine `CVRequest`, die eine Referenz auf das Zertifikat enthält und nicht das Zertifikat an sich.

```
PKCReference ::= CHOICE {
    cert [0] Certificate,
    pkcRef [1] SCVPCertID
}
```

Die `Alternative Request` kann eine `PKCReference` direkt mit dem Zertifikat erzeugen. In diesem Fall reicht es wenn ein `Tagged-Objekt` mit `Tag-Wert` null und das Zertifikat als Objekt generiert wird.

```
PKCReference.getInstance(new DERTaggedObject(0, cert));
```

¹⁹secure hash algorithm

dann wird ein `CertReferences` Element mit einer Sequenz erzeugt, die in diesem Fall nur aus einem `PKCReference` (ein Zertifikat) item besteht.

Um ein zulässiges `Query` Objekt konstruieren zu können bedarf es ein `CertReferences` Objekt dazu noch ein `CertChecks` Objekt und ein `ValidationPolicy` Objekt, und zwar in diese Reihenfolge.

Das `CertChecks` Objekt wird so erzeugt:

```
ASN1EncodableVector checks1 = new ASN1EncodableVector();
checks1.add(new DERObjectIdentifier("1.3.6.1.5.5.7.17.1"));
checks1.add(new DERObjectIdentifier("1.3.6.1.5.5.7.17.2"));
CertChecks checks = CertChecks.getInstance(new DERSequence(checks1));
```

die `Validation Policy` in dieser `CVRequest` ist die `Standard Policy` `id-svp-defaultValPolicy` mit dem `Oid` (1.3.6.1.5.5.7.19.1)

```
public static ValidationPolRef getDefaultValValidPolRef () {
return new ValidationPolRef(new DERObjectIdentifier("1.3.6.1.5.5.7.19.1"));
}
```

dementsprechend ist der verwendete `Validation-Algorithmus` der „basic validation algorithm“ `id-svp-basicValAlg`

```
public static ValidationAlg getDefaultValidationAlg() {
return new ValidationAlg(new DERObjectIdentifier("1.3.6.1.5.5.7.19.3"));
}
```

Im Falle einer `protected Request` wird die `CVRequest` in einer `CMSSignedData` gepackt und mit dem privaten Schlüssel des Clients signiert.

Bei einer `unprotected Request` bedarf es keines Signierens der `CVRequest`. Diese wird in einer `Mime` Nachricht gepackt, und dann mittel der `HTTP-POST` Methode zum Server geschickt.

4.4 Bearbeitungslogik für ankommende Anfragen

Kommt die `Request` beim Server bzw. `Servlet` an, wird ihrer content in einem `Byte-Array` gelesen. Aus diesem `Array` wird dann eine Sequenz konstruiert. Diese wird dafür

verwendet, um verschiedene interne Abfragen durchzuführen, damit der Typ der Anfrage ermittelt.

```
Seq = ASN1Sequence.getInstance(ASN1Object.fromByteArray(request));
```

Wenn sie vom Typ `protected` ist, dann wird eine Methode `controlSignature` mit der `Request` als Argument aufgerufen. Diese Methode prüft die Signatur der `Request`.

```
public boolean controlSignature(byte[] cms) throws Exception
```

Hierfür wird ein `org.bouncycastle.cms.CMSSignedData` Objekt generiert. Es werden dann mit Hilfe seiner Methoden alle für den Prüfvorgang signierten relevanten Informationen ermittelt (Certs und CRLs).

```
CMSSignedData s = new CMSSignedData(cms);
CertStore certs = s.getCertificatesAndCRLs("Collection", "BC");
SignerInformationStore signers = s.getSignerInfos();
Collection c = signers.getSigners();
Iterator it = c.iterator();
```

Danach wird die Signatur sukzessiv für jeden Signierer mit der Methode `verify()` aus dem Package `org.bouncycastle.cms.SignerInformation`

```
while (it.hasNext())
{
    SignerInformation signer = (SignerInformation)it.next();
    Collection certCollection = certs.getCertificates(signer.getSID());

    Iterator certIt = certCollection.iterator();
    X509Certificate cert = (X509Certificate)certIt.next();

    if (signer.verify(cert.getPublicKey()))
    {
        verified++;
    }
}
}
```

Sollte die Signatur akzeptiert worden sein, dann wird noch mal geprüft, um was für eine Request es sich hierbei handelt (`CVRequest` oder `ValPolRequest`) und die entsprechende Handler-Klasse mit ihr instanziiert. Anderenfalls wird ein Fehler Response zum Klient zurück geschickt.

Im Falle, dass das Ergebnis des ersten Tests eine `unprotected Request` liefert, dann wird die handelnde Klasse ohne weiteres damit beauftragt, die Request zu bearbeiten und die gemäß SCVP Spezifikation entsprechende Response zu generieren und anschließend zu liefern.

```
CVResponse cvRes = new CVRequestHandler(cvreq).handle();
```

Die `CVRequestHandler` Klasse hat in diesem Zusammenhang die Aufgabe, alle nötigen Informationen bzw. Parameter für den PKIX certification path validation algorithm²⁰ bereitzuhalten und weitere erforderliche Prüfungen der Werte im Request-Objekt durchzuführen. Zu ihren Aufgaben gehört aber auch die `CVResponse` Konstrukte zu generieren und mit den entsprechenden Werten zu versehen.

Dabei wird als erstes der Kern der Request, also das bzw. die Zertifikat(e), die der Server validieren soll, ermittelt. Das sieht folgender Maßen aus:

```
CertReferences certref =  
    CertReferences.getInstance(cvRequest.getQuery().getQueriedCerts());
```

Durch die Methode `toASN1Object()` von `CertReferences`, wird ein Tagged Objekt erzeugt. Mit dessen Hilfe kann herausgefunden werden, um welchen Typ es sich bei dem zu validierenden Zertifikat handelt (Attribut- oder PK-Zertifikat).

```
CertReferences ::= CHOICE {  
    pkcRefs      [0] SEQUENCE SIZE (1..MAX) OF PKCReference,  
    acRefs       [1] SEQUENCE SIZE (1..MAX) OF ACRreference }
```

Wir fokussieren in diesem Dokument auf Public-Key-Certificates. Für jedes Element in der `PKCReferences`-Sequenz wird die `toASN1Object()`-Methode angewendet für die bevorstehenden Prüfungen; um den Typ der `PKCReference` festzustellen.

²⁰[RFC 3280]

```

PKCReference ::= CHOICE {
    cert    [0] Certificate,
    pkcRef  [1] SCVPCertID
}

```

Wenn der Tag Wert Null ist dann ist das komplette Zertifikat in der Request enthalten und kann Problemlos generiert werden.

```
cert[i] = X509CertificateStructure.getInstance(derTagged.getObject());
```

Wenn dieser Wert gleich eins ist, dann wird ein SCVPCertID erzeugt und an die Methode `getRequestedCert` weitergereicht, diese holt das durch SCVPCertID referenzierte Zertifikat aus der DB.

Es wird für jedes in der Request enthaltene Zertifikat die Methode `getPfad(cert[i])` aufgerufen. Diese Methode ist rekursiv und soll für jedes ihr übergebene Zertifikat einen Zertifizierungsvektor liefern, worin alle Zertifikate sind, die mindestens einen Zertifizierungspfad bilden. Dabei geht sie folgendermaßen vor; sie lädt aus der DB für ihren Argumentparameter sein Issuer-Zertifikat und fügt dieses in den Vektor ein, dann ruft sie sich selbst mit dem Issuer-Zertifikat als aktuellen Parameter auf.

Die Bedingung die diese Rekursion beenden soll ist eine oder-Verknüpfung von 2 Fällen und Zwar:

- das Aktuelle Zertifikat ist ein Trusted Anchor
- IssuerName gleich der SubjectName (selbst issued Zertifikat)

```

private void getPfad(X509CertificateStructure cert)throws
SQLException, InstantiationException, IllegalAccessException,
ClassNotFoundException, CertificateException, IOException {

String      key      = cert.getIssuer().toString();
DBconnection dbcon   = DBconnection.getInstance();
ResultSet   resultSet = dbcon.query("SELECT CERT FROM
                                CERTIFICATES where subjectName ='"'+key+"'");
Blob        blob     = null;
X509CertificateStructure cert2 = null;

```

```

while (resultSet.next()) {

    blob      =   resultSet.getBlob("CERT");
    byte[]  b =   new byte[blob.getBinaryStream().available()];
    blob.getBinaryStream().read(b);
    cert2    =   X509CertificateStructure.getInstance(
                  ASN1Sequence.getInstance(ASN1Object.fromByteArray(
                  blob.getBytes(1, (int) blob.length()))));
    pfad.add(cert2);

    if (!(dbcon.getTrusted().containsKey(cert2.getSubject().toString())
        || cert2.getIssuer().toString().equals(cert2.getSubject().toString())))
        getpfad( cert2);
    }
}

```

Am Ende von diesem Prozess steht für jedes Zertifikat im CVRequest alle möglichen Pfade.

```
Vector <X509CertificateStructure> pfad;
```

Die Klasse CVRequestHandler stellt zur weiteren Vorbereitung und Behandlung der Struktur Komponenten der CVResponse und deren Werte noch weitere Methoden zur Verfügung. Diese ASN.1 Komponenten werden dann in der richtigen Reihenfolge in einen ASN1EncodableVector cvResponseAsVector eingefügt. Aus diesem Vektor wird dann später eine Sequenz erzeugt, aus der dann die eigentliche CVResponse generiert wird.

Solche Methoden sind:

```

private void performversion(){
    if (cvRequest.getcvRequestVersion().equals(new DERInteger(1).
                                                getPositiveValue()))
        cvStatusCode= new CVStatusCode (21);
    else
        cvResponseAsVector.add(new DERInteger(1));
}

```

und

```

private void performPreoduzedat (){
    Date      time      = new Date();

```

```

ProducedAt producedAt = ProducedAt.getInstance(new
    DERGeneralizedTime(time));

cVResponseAsVector.add(producedAt);}

```

Die Methode `ArrayList<X509Certificate> getcertlist ()` wandelt den Typ der Zertifikate von `X509CertificateStructure` zu `java.security.cert.X509Certificate` für die spätere Validation mit `java.security.cert.CertPathBuilder`. Da alle Klassen des Packages `java.security.cert` nur Zertifikate als Instanz der Klasse `X509Certificate` erkennen.

```

public ArrayList<X509Certificate> getcertlist()throws
    CertificateException, IOException{

    CertificateFactory    cf    = CertificateFactory.
                                getInstance("X.509");
    InputStream          inStream = null;

    for (Iterator iterator = pfad.iterator(); iterator.hasNext();) {

        X509CertificateStructure cert = (X509CertificateStructure)
                                        iterator.next();

        inStream                    = new ByteArrayInputStream(cert.getEncoded());

        certList.add((X509Certificate) cf.generateCertificate(inStream));
    }

    return certList;
}

```

Für das Validieren wird jeder konstruierter Pfad an der Methode `BasicAlgvalidate (ArrayList<X509Certificate> certlist)` der Klasse `BasicAlg` übergeben.

Im Folgenden werden die Hauptschritte der in der `BasicAlg`-Klasse Prozesse vorgestellt.

Das Package `java.security.cert` ist sofern wichtig, da er eine Menge von Klassen und Schnittstellen zur Analyse und Verwaltung von Zertifikaten, Sperrlisten (CRL), und Zertifizierung zur Verfügung stellt.

Für das Validieren des Zertifizierungspfads, mit dem von Java implementierte Basic Validation Algorithmus sind folgenden Klassen unentbehrlich:

```
java.security.cert.TrustAnchor
java.security.cert.CollectionCertStoreParameters
java.security.cert.X509CertSelector
java.security.cert.PKIXParameters
java.security.cert.CertPathBuilder
java.security.cert.PKIXBuilderParameters
java.security.cert.CertPathValidator
java.security.cert.CertPathValidatorResult.
```

Für die komplette Bereitstellung aller notwendigen Parameter für den Basic Validation Algorithmus wird noch die Methode `getTrusted` von `DBconnection` aufgerufen.

```
Map<String, X509CertificateStructure> trusted = dbconn.getTrusted();
```

Sie liefert ein Objekt der Klasse `java.util.Map`, in ihm sind die von der internen Validation Policy bzw. in der Datenbak als vertrauten Zertifikat-Aussteller gekennzeichneten CAs. Dann wird in ein weiteren Schritt über das Map-Objekt in einer for-Schleife iteriert und ein vorher dafür erstellte Set-Objekt `java.util.Set` `<TrustAnchor>` mit den vertrauten Zertifikaten ausgefüllt.

```
for (Iterator iterator = v.iterator(); iterator.hasNext();) {
    X509CertificateStructure certificateStructure =
        (X509CertificateStructure)iterator.next();

    inStream = new ByteArrayInputStream(certificateStructure.getEncoded());

    set.add(new TrustAnchor(
        (X509Certificate) cf.generateCertificate(inStream), null));
}
```

Als nächstes wird eine Instance der Klasse `java.security.cert.PKIXParameters` erstellt, so eine Instanz kann auf zwei Wege generiert werden. Der Konstruktor bekommt entweder ein Set-Objekt oder ein KeyStore-Objekt die jeweils mit den sog. Most trusted CAs. Hier ist auf die erste Variante zugegriffen.

```
PKIXParameters pKIXParameters = new PKIXParameters(set);
```

Ein `CertPathValidator` benutzt diese Parameter (`pKIXParameters`) für das Validieren eines Zertifizierungspfads gemäß den PKIX certification path validation algorithm. Davor wird eine Instanz der Klasse `java.security.cert.CertPathBuilder` erzeugt, dieser soll ein Pfad erzeugen.

```
CertPathBuilder cpbTest = CertPathBuilder.getInstance("PKIX", "BC");
```

Das `CertPathBuilder`-Objekt ist die Provider Algorithmus Implementierung, d.h. das erzeugte Objekt kapselt die `CertPathBuilderSpi`-Implementierung von dem Provider (hier Bouncycastle Provider), der beim Start des Servers - Initialisierung des Servlets - in der security provider Liste registriert wurde.

Ein `java.security.cert.X509CertSelector` Objekt selektiert `X509Certificate` Instanzen, die die spezifizierten Kriterien der `PKIXParameters` für den Aufbau eines passenden PKIX certification path entspricht.

```
X509CertSelector selector = new X509CertSelector();
PKIXBuilderParameters pKIXBuilderParameters =
    new PKIXBuilderParameters(set, selector);
```

Beispiel einige Validation Kriterien :

```
pKIXBuilderParameters.setRevocationEnabled(false);
pKIXBuilderParameters.addCertStore(CAS);
pKIXBuilderParameters.getInitialPolicies();
```

Jetzt kann die Validation mit einem `CertPathValidator` stattfinden. Dies geschieht mit dem Aufruf seiner Methode `validate(CertPath certPath, CertPathParameters params)` wobei `certPath` der zu validierende Zertifizierungspfad ist. Diese Methode liefert ein `CertPathValidatorResult` oder wirft dann einen Fehler aus, wenn die Validation misslungen ist.

```
CertPathValidator cpvTest = CertPathValidator.getInstance("PKIX", "BC");
```

```
CertPathValidatorResult cpvrTest =
    cpvTest.validate(cpbTest.getCertPath(), pKIXParameters );
```

Liegt das Ergebnis der Validation einmal vor, wird dementsprechend eine `CVResponse` zusammengestellt, mit dem privaten Serverschlüssel des Servers signiert, und an das Servlet zurückgeschickt und danach an den Client weitergeleitet.

5 Nachwort

Diese Arbeit hat den Grundstein dafür gelegt, eine SCVP-Webanwendung zu realisieren. Dies erfolgte durch die Realisierung der SCVP-RFC-Anforderungen für die ASN1-Datenstrukturen.

Der während dieser Arbeit implementierte Prototyp könnte mit zukünftigen Arbeiten, die dieses Thema weiter behandeln noch mit weiteren Schnittstellen erweitert werden, z.B. für die Kommunikation mit OCSP Servern. Eine weitere Erweiterung könnte der Einsatz von einem Hardware Token (smartcard) zum Beispiel für den Schlüssel des Server sein.

Die dem Prototyp zugrunde liegenden Komponenten, wie Tomcat-Server und die Derby-Datenbank waren nicht auf optimierte Performanz ausgerichtet. Ziel war mehr, eine SCVP-Bibliothek mit den entsprechenden Datenstrukturen zu liefern und den Anforderungen zu genügen.

Die Komponenten einer auf einen marktreifen Einsatz ausgerichteten Lösung sollten möglichst mit einer großen Anzahl von Requests, Zertifikaten und CRLs zwecks Skalierung getestet werden. Hier gilt es, die Datenbankverwendung neu zu bedenken und gegebenenfalls die Tabellen neu zu indizieren. Auch der Server kann durch einen Cache optimiert werden.

6 Literaturverzeichnis

- Pattern-orientierte Software-Architektur: ein Pattern-System
Autor : Frank Buschmann
Verlag : ADDISON Wesley

- Software Architecture Design Patterns in Java
Autor : Partha Kuchana
Verlag : Auerbach Publications

- Einführung in die Kryptographie
Autor : Johannes Buchmann
Verlag : Springer

- IT-Sicherheit. Konzepte - Verfahren - Protokolle
Autor : Claudia Eckert
Verlag : Oldenbourg

- Public Key Infrastructure: First European Pki Workshop: Research and Applications
Autoren : Sokratis K. Katsikas, Stefanos Gritzalis, Javier Lopez
Verlag : Springer

- Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile RFC 3280, April 2002

- Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile April 2002

- Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Certificate Handling, RFC 3850, July 2004.

- ASN.1 und OID Project Dokumentation:
<http://www.itu.int/ITU-T/asn1/>
<http://telecom.htwm.de/asn1/index.htm>

- ASN.1 communication between heterogeneous systems
 Autor : Olivier Dubuisson
 Herausgeber : 2001-OSSNokalva.

- Server-Based Certificate Validation Protocol (SCVP), RFC 5055, December 2007

- Derby-DBMS; <http://db.apache.org/derby/manuals/index.html>

- Tomcat-Webserver; <http://tomcat.apache.org/tomcat-6.0-doc/index.html>

- OpenSSL; <http://www.openssl.org/docs/apps/openssl.html>