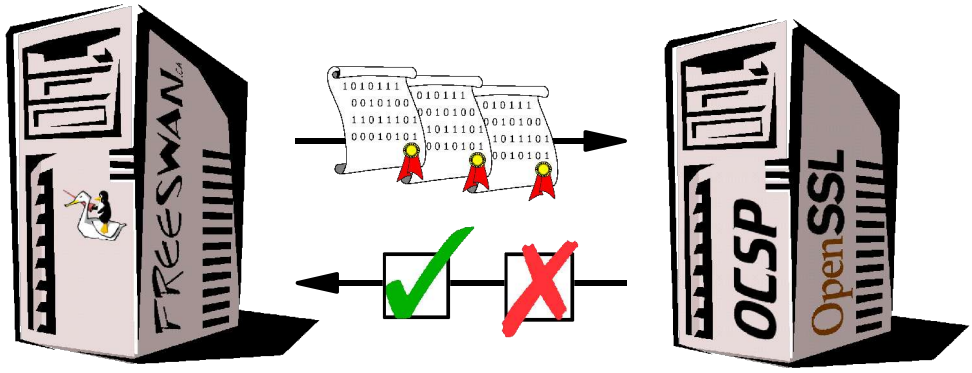
Zürcher
Hochschule
Winterthur

Mitglied
der Zürcher
Fachhochschule

Diplomarbeit
Oktober 2003



OCSP für Linux FreeS/WAN

Dozent:
Andreas Steffen

Autoren:
Christoph Gysin
Simon Zwahlen

Zusammenfassung

Mit FreeS/WAN existiert eine freie IPSec Implementierung für Linux, die es erlaubt, verschlüsselte Tunnel durch beliebige IP-Netzwerke aufzubauen. Durch die symmetrische Verschlüsselung benötigen jedoch alle Kommunikationspartner zuerst den geheimen Schlüssel des Gegenübers. Eine Public Key Infrastruktur (PKI) schafft hier Abhilfe. Eine Certificate Authority (CA) verpackt die Public Keys aller Kommunikationspartner in Zertifikate und signiert diese digital. Damit kann jeder den entsprechenden Schlüssel über einen unsicheren Kanal beziehen und überprüfen, ob ihm der authentische Public Key des Gegenübers vorliegt.

Mit einem Patch ist die Authentifikation mit X.509 Zertifikaten auch mit FreeS/WAN möglich. Damit sind Tür und Tor geöffnet für den Einsatz von FreeS/WAN in grösseren Umgebungen, wie zum Beispiel einer Grossfirma oder Universität, wo sonst unzählige Schlüssel ausgetauscht werden müssten. Falls ein Zertifikat einmal vor Ablauf seiner Gültigkeitsdauer gesperrt werden soll, gibt es mit den Certificate Revocation Lists (CRL) einen Mechanismus, der die aktuellen Stati der Zertifikate verwaltet. Gerade bei grossen PKI's kann eine CRL aber schnell einmal zu einer stattlichen Grösse heranwachsen.

Mit OCSP existiert nun ein Protokoll, um den Status einzelner X.509 Zertifikate abzufragen. Dadurch wird eine zusätzliche Belastung des Netzwerks verhindert und gleichzeitig die Statusinformation über die Zertifikate immer auf dem aktuellsten Stand gehalten.

Im Laufe der Diplomarbeit wurde der Aufbau des bestehenden FreeS/WAN-Projektes analysiert und ein standardkonformer OCSP Client implementiert. Während der Designphase zeigte sich schnell, dass ein Cache für gültige OCSP-Meldungen den Verbindungsaufbau stark optimiert. Auf der Basis dieses Realisierungskonzepts wurde die OCSP Client-Funktionalität in FreeS/WAN integriert.

Da die Implementierung weniger Zeit in Anspruch nahm als geplant, konnten zusätzliche Funktionen realisiert werden. Ausserdem konnte mehr Wert auf die Detailpflege und einen ausführlichen Funktionstest gelegt werden.

Um FreeS/WAN OCSP-fähig zu machen, braucht lediglich unser Patch angewandt und FreeS/WAN neu kompiliert werden. Erhält der FreeS/WAN Gateway nun ein Zertifikat, das die Adresse eines OCSP Servers enthält, erkundigt er sich bei Jenem nach dem aktuellen Status des Zertifikates. Anhand dessen kann der Gateway entscheiden, ob die Verbindung mit dem Client erlaubt werden soll.

Mit dem Resultat unserer Diplomarbeit existiert nun ein standardkonformer OCSP Client für FreeS/WAN, der zum Beispiel mit dem kürzlich erschienenen OpenSSL OCSP Server verwendet werden kann.

Winterthur, 24.10.2003 Christoph Gysin Simon Zwahlen

Abstract

FreeS/WAN is a free implementation for Linux of IPSec, which allows the building of encrypted tunnels through any IP based network. Because of symmetric encryption each side first, however, needs the secret key of the opposite side. A public key infrastructure (PKI) solves this problem. A certificate authority (CA) packages the public keys of all the communications partners in certificates and digitally signs them. With these anyone can get the corresponding key over an insecure network and verify whether it submits to the authentic public key of the opposite side.

Since the publication of the patch from Andreas Steffen, authentication by X.509 certificates is also possible with FreeS/WAN. This opens the doors to the use of FreeS/WAN in large environments such as universities or large companies, where otherwise countless keys would have to be exchanged. For cases where a certificate is revoked before it expires, certificate revocation lists (CRL) provide a mechanism for maintaining the current statuses of the certificates. But in large PKIs a CRL can quickly grow to a considerable size.

With OCSP there is now a protocol for querying the status of individual X.509 certificates. The use of OCSP avoids an additional load on the network and at the same time keeps the status information on the certificates up to date.

This diploma thesis includes analysing the structure of FreeS/WAN and the implementation of a standards conformant OCSP client. During the design, the need for a cache of valid responses became necessary. Based on this concept, OCSP client functionality was integrated into FreeS/WAN.

Since the implementation took less time than intended, more features were implemented. Additionally, more time was spent on details and a comprehensive test.

To make FreeS/WAN OCSP-capable one must merely apply our patch and recompile FreeS/WAN. If a FreeS/WAN gateway now receives a certificate containing the address of an OCSP server, it queries the server about the certificate's current status. Based on the response, the gateway can decide whether or not the connection with the client should be allowed. To minimize the delay caused by the OCSP request our implementation includes a cache to temporarily save received messages. The OCSP server updates the cache entries periodically.

With our diploma thesis there now exists a standards conformant OCSP client for FreeS/WAN, which can be used in conjunction with the recently released OpenSSL OCSP server.

Inhaltsverzeichnis

1 Einleitung 6

1.1 Vorwort 6

1.2 Aufgabenstellung 8

1.3 Pflichtenheft 10

2 Grundlagen 13

2.1 IPSec 13

2.2 RSA Public Key Verfahren 17

2.3 Public Key Infrastruktur (PKI) 19

2.4 ASN.1 26

2.5 ASN.1 encoding rules: BER / CER / DER 29

3 Analyse 35

3.1 FreeS/WAN 35

3.2 OpenSSL OCSP Server 42

3.3 CURL 42

4 Design 44

4.1 Verbindungsaufbau 44

4.2 OCSP Cache 46

4.3 Eingehende OCSP Antwort 48

5 Realisierung 51

5.1 Neues Modul OCSP 52

5.2 Änderungen Modul Fetch 54

5.3 Änderungen Modul X.509 55

5.4 ASN.1 Generierung 56

5.5 ASN.1 Parsing 57

5.6 Konfigurationsoptionen 60

6 Test 61

6.1 Anwendungstests 61

6.2 Anwendungstests mit mehreren Clients 71

6.3 Fehlermeldungen 72

7 Installation und Anwendung 75

7.1 Introduction 75

7.2 Requirements 75
7.3 Installation 75
7.4 Configuration 76
7.5 Usage 78

8 Schlusswort 80

9 Anhang 81
9.1 Projektplan 81
9.2 ASN.1 Syntax für OCSP-Meldungen 82
9.3 Quellen 88
9.4 Glossar 90
9.5 Index 91

1 Einleitung

1.1 Vorwort

Mit dem zunehmenden Ausbau des Internets und dem stetig wachsenden Datenverkehr steigt die Nachfrage nach Sicherheit. Sicherheit im Datenverkehr bedeutet Authentizität, Integrität und Vertraulichkeit zwischen einzelnen Kommunikationspartnern. Diese Anforderungen lassen sich mit der bisher verwendeten Technologie nur mühsam und nicht applikationsunabhängig realisieren, da dieser Aspekt bei der Entwicklung des bis heute gebräuchlichen Internetprotokolls IPv4 ausser acht gelassen wurde.

Diese Lücke schliesst der IETF Standard IPSec [14]. IPSec wurde ursprünglich für IPv6 entwickelt, den Nachfolger von IPv4. Da aber die Migration auf IPv6 seit Jahren auf sich warten lässt, wurde IPSec auch für IPv4 implementiert.

Mit FreeS/WAN [15] existiert nun eine freie Implementierung für Linux, veröffentlicht unter der *GNU General Public License* (GPL). Sie erlaubt das Aufbauen von sicheren Tunnels über unsichere Netze. So lassen sich zum Beispiel Notebooks von einem beliebigen Punkt aus über das Internet ins Firmennetzwerk verbinden (sogenannte Roadwarriors), ohne dass vertrauliche Daten an die Öffentlichkeit gelangen können.

Leider erfolgt im ursprünglichen FreeS/WAN Projekt die Authentifizierung über vorgängig ausgetauschte Schlüssel. Damit bleibt das Problem bestehen, dass der Schlüssel vor dem Aufbau einer Verbindung über einen sicheren Kanal übertragen werden muss. Abhilfe schafft hier eine *Public Key Infrastruktur* (PKI).

Dank dem X.509 Patch für FreeS/WAN [16] ist es möglich, sich mit einem X.509 Zertifikat zu authentifizieren. Damit vereinfacht sich das Management von Schlüsseln erheblich, da die Public Keys von einem öffentlichen Verzeichnis zur Verfügung gestellt werden können. Ein Sicherheitsrisiko besteht dadurch nicht, da die CA alle Zertifikate digital signiert. So kann jeder Client mit Hilfe des CA-Zertifikats überprüfen, ob ihm das unveränderte Zertifikat vorliegt.

Jedes Zertifikat ist nur für eine bestimmte Zeitdauer gültig. Wird diese überschritten, verliert das Zertifikat seine Gültigkeit. Nun kann es aber vorkommen, dass man ein Zertifikat vor Ablauf seiner Gültigkeit sperren möchte (etwa weil ein Mitarbeiter entlassen wurde, oder ein privater Schlüssel verloren ging). Dazu wird ein zusätzlicher Mechanismus benötigt, um den Status aller Zertifikate zu speichern. Alle Authentifizierungen über X.509 Zertifikate müssen dann zusätzlich zur Gültigkeit des Zertifikats auch noch den Status überprüfen. Nur dann wird ein Zertifikat auch akzeptiert.

Dieser Mechanismus wird mit *Certificate Revocation Lists* (CRL) realisiert. Eine CRL ist eine Liste aller Zertifikate einer CA, bei der jedem Zertifikat ein Status zugeordnet ist (GOOD oder REVOKED). Die CRL ist öffentlich verfügbar, z.B. auf einem HTTP Server. Damit sie auf dem Weg vom Server zum anfordernden Gateway nicht

modifiziert werden kann, ist sie von der CA digital signiert. Will nun ein Gateway ein Benutzer authentifizieren, überprüft er erst sein Zertifikat auf seine Gültigkeit, bezieht die aktuelle CRL und sucht den aktuellen Status des Zertifikats. Ist es nicht gesperrt, ist der Benutzer authentifiziert.

Betreibt man aber eine grosse PKI, erkennt man schnell das Problem dieser Methode: Da alle Zertifikate in der Liste eingetragen sind, wächst die CRL sehr schnell. Um den aktuellen Status eines Zertifikats in Erfahrung zu bringen, ist stets die komplette Liste zu beziehen. Das kostet Zeit und Bandbreite. Effizienter wäre ein Protokoll welches ermöglicht, nur die Stati der gewünschten Zertifikate abzufragen.

Genau das ist das Ziel des *Online Certificate Status Protocol* (OCSP). Ein OCSP Server verwaltet die CRL und empfängt (signierte) Anfragen von OCSP Clients. Er durchsucht die CRL nach den gewünschten Zertifikaten und sendet eine signierte Antwort an den Client zurück.

Das FreeS/WAN Projekt unterstützt zur Zeit lediglich die Überprüfung via CRL. Damit der Verwendung von FreeS/WAN auch in grossen Unternehmen nichts im Wege steht, haben wir uns die Implementierung von OCSP in FreeS/WAN zum Ziel gesetzt.

1.2 Aufgabenstellung

Kommunikation

Praktische Diplomarbeiten 2003 - Sna03/1

Online Certificate Status Protocol (OCSP) für Linux Free/SWAN

Studierende:

- Christoph Gysin, IT3a
- Simon Zwahlen, IT3a

Termine:

- Ausgabe: Donnerstag, 4.09.2002 9:00 im E523
- Abgabe: Montag, 27.10.2003 12:00

Beschreibung:

Als Alternative zu Certificate Revocation Lists (CRLs), die sehr umfangreich werden können, kann das Online Certificate Status Protocol (OCSP) verwendet werden, um in Erfahrung zu bringen, ob ein X.509 Zertifikat noch gültig ist. Das OpenSSL 0.9.7 Packet bietet seit kurzem eine Serverfunktion an, welche OCSP Anfragen beantworten kann.

Für den OpenSource FreeS/WAN Internet Key Exchange Daemon (IKE) soll im Rahmen der vorgeschlagenen Diplomarbeit die OCSP Clientfunktion auf der Basis der bestehenden FreeS/WAN Kryptofunktionen, sowie der GNU Multi-Precision Library realisiert werden. Wegen der US Exportrestriktionen für starke Kryptographie darf kein OpenSSL Source Code oder Libraryfunktionen in FreeS/WAN verwendet werden, da OpenSSL teilweise durch US Bürger oder in den USA erstellt wurde.

Ziele:

- Verstehen des OCSP Protokolls
- Festlegen der zu realisierenden OCSP Clientfunktionalität in einem Pflichtenheft
- Erstellen einer oder mehrerer Konzeptvarianten, Konzeptentscheid
- Implementierung der OCSP Clientfunktionalität auf der Basis von FreeS/WAN 2.01 und dem X.509 Patch 1.4.5.
- Testsuite im Zusammenspiel mit einem OCSP Server auf der Basis von OpenSSL 0.9.7b.
- Dokumentation der OCSP Funktionalität

- Kurzanleitung für eine typische FreeS/WAN Client - OpenSSL Server OCSP Konfiguration

Infrastruktur / Tools:

- Raum: **E523**
- Rechner: 2 PCs unter Windows 2000 / SuSE Linux
- SW-Tools: GNU C Compiler, Linux FreeS/WAN 2.01, OpenSSL 0.9.7b

Literatur / Links:

- IETF RFC 2459
  PKIX - Certificate and CRL Profile
  http://www.ietf.org/rfc/rfc2459.txt
- IETF RFC 2560
  PKIX - Online Certificate Status Protocol (OCSP)
  http://www.ietf.org/rfc/rfc2560.txt
- ISIS-MTT Part 4: Operational Protocols - Chapter 3 Directory Access via OCSP
  http://www.t7-isis.de/ISIS-MTT/ISIS-MTT-CoreSpec-V1_0_2.pdf
- FreeS/WAN und X.509 Patch
  http://www.freeswan.ca/download.php
- OpenSSL OCSP Funktionalität
  http://www.openssl.org/docs/apps/ocsp.html
- GNU Multi-Precision Library
  http://www.swox.com/gmp

Winterthur, 4. September 2003

Prof. Dr. Andreas Steffen

1.3 Pflichtenheft

1.3.1 Ziel

Für das Open-Source-Projekt FreeS/WAN [15] soll die OCSP-Clientfunktionalität gemäss RFC 2560 [9] implementiert werden.
Damit sollen die bestehenden Certificate Revocation Lists (CRL) ersetzt werden, welche es ermöglichen X.509-Zertifikate auf ihre Gültigkeit zu überprüfen.
Im Gegensatz zu den 'statischen' CRL's, welche in regelmässigen Abständen aktualisiert werden müssen, funktioniert OCSP mit dynamischen Anfragen auf einen zentralen Server. Dies ermöglicht Konsistenz der Zertifikats-Statusinformationen mit sehr geringem Netzwerkverkehr.

1.3.2 Anforderung an den OCSP-Client

Bei jedem eingehenden Verbindungsaufbau wird die Gültigkeit des Zertifikates mit OCSP überprüft, sofern im betreffenden Zertifikat eine entsprechende AuthorityInfoAccess-Extension mit einer URI vorhanden ist.

Features:

- Senden von Requests
  - ASN.1 Meldungstruktur gemäss RFC2560 , 4.1
  - Signieren von Requests mit SHA-1 Verfahren (RFC2560, 4.1.1)
  - Über HTTP Method POST (RFC2560, A.1)
  - Einbinden der Nonce-Extension zur Verhinderung von Replay-Attacken (RFC2560, 4.4.1)
  - Einbinden der Acceptable Response Types - Extension (garantieren der ausschliesslichen Verwendung von Basic-OCSP-Responses) (RFC2560, 4.4.3)
  - Cache für eingegangene Meldungen

- Empfangen von Responses
  - Nur Basic OCSP-Responses (RFC2560, 4.2.1)
  - Überprüfen der Response auf Gültigkeit (RFC2560, 3.2)
    - ☑ Response gehört zum Request
    - ☑ Gültige Signatur
    - ☑ Identität vom Signierer entspricht der Identität des Responders (gemäss Anfrage)

☑ Der Signierer ist entweder

- ➢ die CA des Zertifikats, oder
- ➢ direkt von der CA authorisiert

☑ Angegebener Zeitpunkt der Statusinformation (thisUpdate) ist genügend aktuell

☑ Falls der Zeitpunkt für die nächste Aktualisierung (nextUpdate) gegeben ist, muss er in der Zukunft liegen

☑ Falls die Nonce-Extension angegeben ist, muss die Nonce der Nonce des Requests entsprechen.

– Falls die strictCrlPolicy gesetzt ist, wird im Fehlerfall (Unerreichbarkeit des Servers, Fehlermeldungen) der Verbindungsaufbau verweigert

1.3.3 Voraussichtlich nicht unterstützte Features

- Senden von Requests
  - – Nur Transportprotokoll HTTP, nur Method POST, kein GET (RFC2560, A.1)
  - – Keine Hashes ausser SHA-1
  - – Keine Extensions ausser den explizit erwähnten (RFC2560, 4.4f)
  - – Keine lokale Konfiguration der OCSP-URI (RFC2560, 3.1)

- Empfangen von Responses
  - – Kein Auswerten von Extensions (RFC2560, 4.4f) ausser Nonces
  - – Keine lokale Konfiguration von Trusted Signers (RFC2560, 4.2.2.2)

1.3.4 Randbedingungen

- FreeS/WAN 2.0.2 mit X.509 Patch 1.4.7.
- Linux-Kernel 2.4.22.
- Das HTTP-Protokoll wird mit libcurl 7.10.5 implementiert.
- Als OCSP-Server steht OpenSSL 0.9.7b zur Verfügung.
- Es darf kein OpenSSL-Code verwendet werden.
- Coding-Conventions gemäss FreeS/WAN-Projekt.

1.3.5 Termine

Beginn: 4.9.2003
Abgabe: 27.10.2003

2 Grundlagen

2.1 IPSec

IPSec ist ein IETF-Standard [14] für Echtzeit-Kommunikationssicherheit. Es bietet eine sichere Kommunikation in IP-Netzwerken für alle Anwendungen. Im Gegensatz zu anderen Sicherheitsstandards, welche oberhalb der Transportschicht (z.B. SSL) oder der Anwendungsschicht (z.B. GPG) arbeiten, bietet IPSec Integrität, Authentizität und Vertraulichkeit schon auf Netzwerkschicht und ist somit viel universeller in der Anwendung.

Funktionsumfang:

- Integrität

  Die Daten können vor einer Modifikation bei der Übertragung geschützt werden, indem eine verschlüsselte Prüfsumme in das Datenpaket eingefügt wird.

- Authentizität

  Die Authentizität wird garantiert, indem die Integritätsbedingung auch auf die Absenderadresse zutrifft.

- Vertraulichkeit

  Die zu transportierenden Daten können mit Hilfe von kryptographischen Verfahren geschützt werden.

Die Hauptbestandteile von IPSec sind die zwei IP Header Extensions *Authentication Header* (AH) und *Encapsulating Security Payload* (ESP) und *Internet Key Exchange* (IKE), ein Protokoll zur Authentifizierung und Erstellung eines Session Keys.

2.1.1 SPI

IPSec bietet Geheimhaltung, Fälschungssicherheit oder beides zusammen. Für jede dieser Funktionen existiert ein eigener optionaler Header: AH und ESP. Beide Header enthalten den *Security Parameter Index* (SPI) anhand dessen der Rechner entscheidet, welche Chiffrierschlüssel und Verfahren er verwenden soll.

2.1.2 SA

Anhand des SPI und der Zieladresse wird eine sogenannte *Security Association* (SA) identifiziert und damit festgelegt, welche Parameter für die Authentisierung und Verschlüsselung verwendet werden (RFC 1825).

Zwei Rechner richten untereinander eine SA ein, welche das verwendete Verschlüsselungs- sowie das Authentifizierungsverfahren spezifiziert. Eine SA enthält dabei unter anderem folgende Werte:

- Zieladresse
- Verschlüsselungs-, Authentifizierungsverfahren
- Aktueller geheimer Schlüssel
- Zeitangabe für die Gültigkeit des Schlüssels

2.1.3 ISAKMP (RFC2408)

Das *Internet Security Association and Key Management Protocol* (ISAKMP) dient dazu, eine SA zwischen zwei Rechnern auszuhandeln. ISAKMP eignet sich auch um Schlüssel zwischen den beiden Rechnern auszutauschen. Dabei ist ISAKMP nicht auf bestimmte Verschlüsselungsverfahren beschränkt.

2.1.4 IKE (RFC2409)

Internet Key Exchange (IKE) ist ein konkretes Schlüsselaustauschverfahren, das auf die Benutzung mit ISAKMP abgestimmt ist. Eine IKE Implementierung bietet folgende Eigenschaften:

- Unterstützung des *Data Encryption Standards* (DES)
- Unterstützung der Hashfunktionen MD5 und SHA
- Authentifizierung durch zuvor ausgehandelte Schlüssel

IKE definiert zwei Phasen: Phase 1 erstellt Session Keys, mit welchen dann in Phase 2 Security Associations erstellt werden können.

IKE Phase 1

Für Phase 1 existieren zwei verschiedene Modi:

- *Aggressive Mode* authentifiziert zwei Stellen gegenseitig und erstellt Session Keys in drei Meldungen.
- *Main Mode* benützt sechs Meldungen und bietet zusätzliche Funktionalität (verstecken der Endpunkte, mehr Flexibilität beim Aushandeln der kryptographischen Algorithmen).

Main Mode

Der Main Mode besteht aus sechs Meldungen:

In den ersten zwei Meldungen werden Parameter ausgetauscht. Alice schickt ein Cookie und die gewünschten kryptographischen Algorithmen. Bob antwortet mit einem Cookie und wählt die Algorithmen mit denen er einverstanden ist.

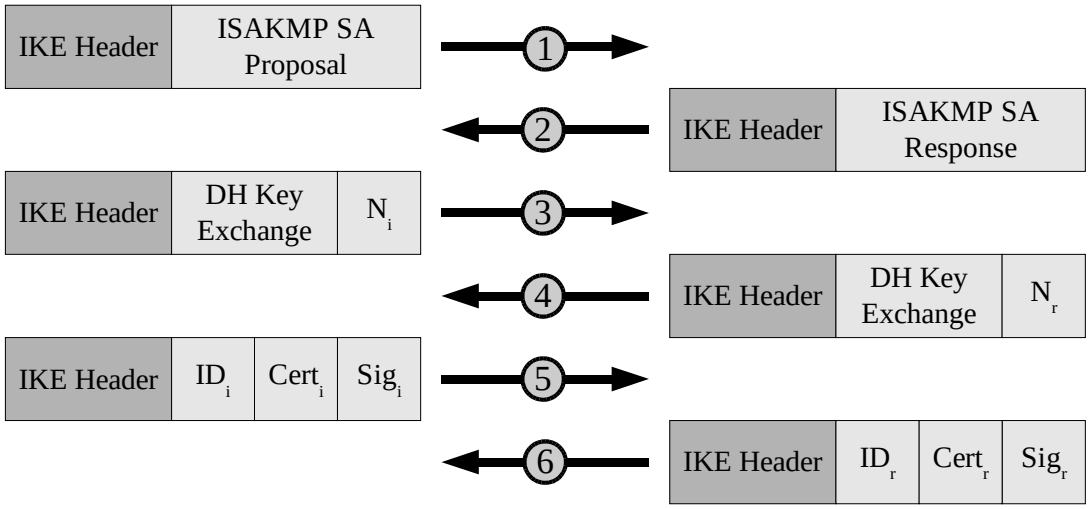


Abbildung 1 IKE Main Mode

In den Meldungen drei und vier wird ein mittels dem Diffie-Hellman-Verfahren ein Session Key generiert.

Meldungen fünf und sechs werden mit dem Session Key verschlüsselt. Darin werden die Identitäten und Zertifikate ausgetauscht und der Beweis erbracht, dass der entsprechende Private Key bekannt ist (Signatur).

Aggressive Mode

Der Aggressive Mode besteht aus nur drei Meldungen:

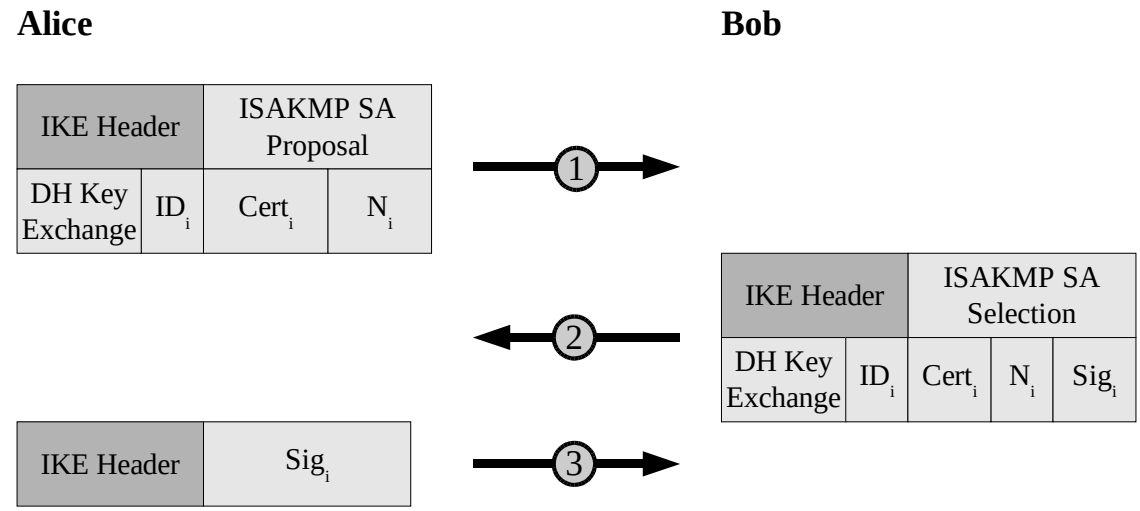


Abbildung 2 IKE Aggressive Mode

In den ersten beiden Meldungen findet ein Diffie-Hellman-Schlüsselaustausch statt, um einen Session Key zu erstellen. Gleichzeitig sendet Alice in der ersten Meldung ihre

Identität an Bob, zusammen mit einem Vorschlag für das kryptographische Verfahren. In der zweiten Meldung trifft Bob eine Auswahl aus den Verfahren und beweist seine Identität. In der letzten Meldung beweist schliesslich auch noch Alice ihre Identität.

IKE Phase 2

IKE Phase 2 ist bekannt als *Quick Mode*. In drei Meldungen werden dabei die Parameter für die Phase 2 SA ausgehandelt (Kryptographischer Algorithmus, SPI).

2.1.5 FreeS/WAN

FreeS/WAN [15] ist eine freie IPSec-Implementation, veröffentlicht unter der *GNU General Public License* (GPL).

Folgende Standards werden von FreeS/WAN unterstützt:

- IPSec
- IKE
- Manuelle Schlüsselverteilung
- Triple-DES als Standardverfahren
- MD5 und SHA
- AH und ESP

FreeS/WAN besteht aus zusätzlichem Kernelcode (KLIPS), einem Dienst der für die Schlüsselverwaltung zuständig ist (Pluto) und Tools welche zum Betrieb von FreeS/WAN benötigt werden.

Der Daemon Pluto ist die User-Level IKE-Implementierung von FreeS/WAN, welche für den Verbindungsaufbau zuständig ist.

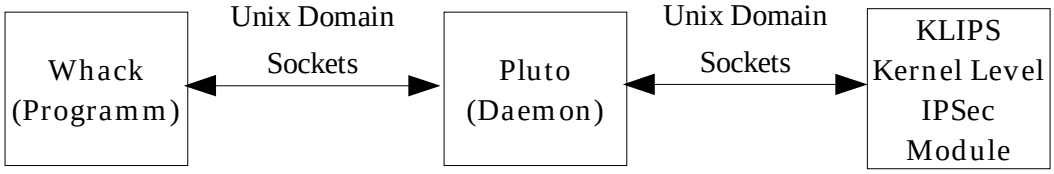


Abbildung 3 Aufbau FreeS/WAN

Abbildung 3 zeigt den logischen Aufbau des FreeS/WAN-Paketes im Überblick.

Pluto ist ein Daemon, der stets im Hintergrund läuft und in erster Linie für das IKE-Handling zuständig ist. Er kommuniziert mit dem Kernel-Modul über Unix Domain Sockets.

Um dem Daemon Befehle geben zu können, wurde ein Programm “whack” eingeführt, das über Unix Domain Sockets mit Pluto kommuniziert.

Die normale Benutzerinteraktion geschieht über das Shell-Script “ipsec”.

Beispiel:

```
ipsec auto --up pluto
```

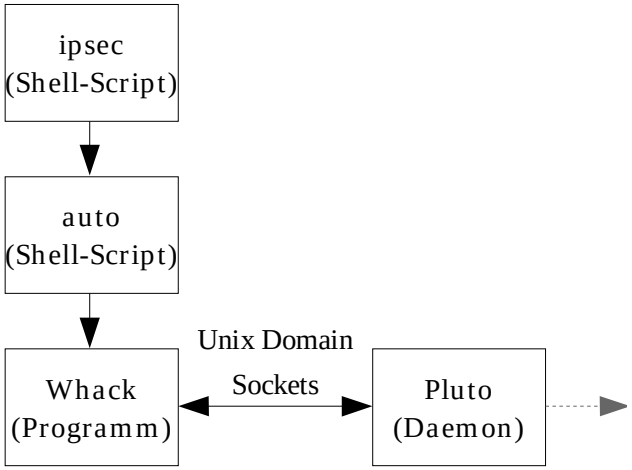


Abbildung 4 Kommunikation FreeS/WAN

Abbildung 4 zeigt den typischen Ablauf bei Aufruf eines Benutzer-Kommandos.

2.2 RSA Public Key Verfahren

RSA ist ein asymmetrisches Verschlüsselungsverfahren, benannt nach seinen Erfindern Rivest, Shamir und Adleman. Es basiert auf dem mathematischen Problem der Primfaktorzerlegung. Die Schlüssellänge ist frei wählbar. Es ist dem Benutzer überlassen lange Schlüssel für erhöhte Sicherheit oder kurze Schlüssel für bessere Effizienz zu verwenden. Es werden zurzeit jedoch meistens RSA-Schlüssel mit 1024 Bit verwendet.

2.2.1 Schlüsselgenerierung

Als erstes wird ein Public und ein Private Key generiert. Dazu werden zufällig zwei grosse Primzahlen p und q gewählt (beide mit etwa der halben Schlüssellänge). Da es jedoch zu aufwändig wäre, zu Überprüfen ob die Zahlen wirklich prim sind, beschränkt man sich auf Wahrscheinlichkeitstests. Der Bekannteste ist der Rabin-Miller Test. Passiert eine Zahl diesen Test, ist sie zu 99.9% prim. Nach 5 erfolgreichen Durchläufen dieses Tests kann man annehmen, die Zahl sei prim.

- Die beiden Zahlen werden nun multipliziert, das Ergebnis nennen wir n. Die Faktoren p und q bleiben geheim. Es ist praktisch unmöglich n in die beiden Faktoren p und q zu zerlegen.

- Nun wird eine zufällige Zahl $e<(p-1)(q-1)$ gewählt, wobei e und $(p-1)(q-1)$ keine gemeinsamen Primfaktoren enthalten sollten.

Als letztes berechnen wir $d=e^{-1} mod (p-1)(q-1)$. Die Gleichung $d \cdot e mod (p-1)(q-1)=1$ lässt sich mit dem Euklidischen Algorithmus lösen.

Der Public Key besteht nun aus dem Wert e und dem Modulus n, der Private Key aus dem Wert d und dem Modulus n.

2.2.2 Verschlüsselung/Entschlüsselung

Um den Klartext zu Verschlüsseln, wird der Public Key benötigt.

- Der Klartext wird in Blöcke fixer Grösse zerlegt, wobei die Blockgrösse kleiner als die Schlüssellänge sein muss.
- Jeder Klartextblock x wird mittels der Funktion $y=x^e mod n$ in den entsprechenden Ciphertextblock y transformiert.
- Die Ciphertextblöcke werden aneinandergehängt und können nun über einen unsicheren Kanal übertragen werden.

Der Ciphertext lässt sich nun nur noch mit dem Private Key entschlüsseln.

- Der Empfänger zerlegt den Ciphertext wieder in Blöcke der Länge des Schlüssels.
- Jeder Ciphertextblock y wird mittels der Funktion $x=y^d mod n$ in den entsprechenden Klartextblock x transformiert.
- Die Klartextblöcke werden auf ihre ursprüngliche Länge gekürzt und aneinandergehängt.

2.2.3 Digitale Signatur

Klartext zu Verschlüsseln ist aber nicht die einzige Anwendung des Public Key Verfahrens. Eine weitere, sehr wichtige Funktion ist das digitale Signieren von Daten. Damit lässt sich überprüfen, ob eine gewisse Meldung seit unterschreiben modifiziert wurde. Um dies zu erreichen, werden die beiden Schlüssel in umgekehrter Reihenfolge angewandt:

- Als erstes wird ein Hash über die zu signierenden Daten gebildet.
- Der Hash wird mit dem Private Key verschlüsselt.
- Der verschlüsselte Hash sowie die Information, welcher Hashalgorithmus verwendet wurde wird an die Daten angehängt.

Die Meldung kann nun über einen beliebigen Kanal übertragen werden. Der Empfänger kann mit folgenden Schritten überprüfen, ob die Daten unterwegs manipuliert worden sind:

- Die Signatur sowie die Information über den Hashalgorithmus werden von den Daten getrennt.
- Ein Hash wird mit dem gleichen Algorithmus über die Daten gebildet.
- Die Signatur wird mit dem Public Key des Absender entschlüsselt und mit dem berechneten Hash verglichen.

Wenn beide Werte übereinstimmen, sind auch die Daten mit grösster Wahrscheinlichkeit gleich.

2.3 Public Key Infrastruktur (PKI)

Da jedes asymmetrische Verschlüsselungsverfahren darauf basiert, dass der Sender im Besitz des Public Keys des Empfängers ist, wird ein Verfahren benötigt, das es ermöglicht, die Authentizität eines Public Keys zu überprüfen. Dies ist die Aufgabe einer Public Key Infrastruktur (PKI). Unter einer PKI versteht man die Komponenten, die nötig sind für eine sichere Verteilung der Public Keys. Sie besteht idealerweise aus:

- einer Certificate Authority (CA), welche Zertifikate ausstellt
- einem Dienst der die Zertifikate zur Verfügung stellt
- einer Möglichkeit, Zertifikate zu sperren
- einer Möglichkeit, Zertifikate auf ihre Gültigkeit zu überprüfen

Ein Zertifikat enthält den Public Key und die Identität des Besitzers (z.B. in Form einer Email Adresse) und ist von einer Certificate Authority digital signiert. Somit muss man nur noch im Besitz des CA-Zertifikats sein, um alle ausgestellten Zertifikate überprüfen zu können. Die entsprechenden Public Keys können von einem öffentlichen Server in Form von Zertifikaten geholt werden und auf ihre Authentizität überprüft werden.

2.3.1 Hierarchische Vertrauensketten

In einer hierarchischen Vertrauenskette bildet die RootCA die oberste Instanz. Alle Teilnehmer müssen ihr zwingend vertrauen, da alle Zertifikate von ihr signiert werden. Sie kann Benutzerzertifikate oder Intermediate CA's austellen, welche wiederum Benutzerzertifikate austellen können.

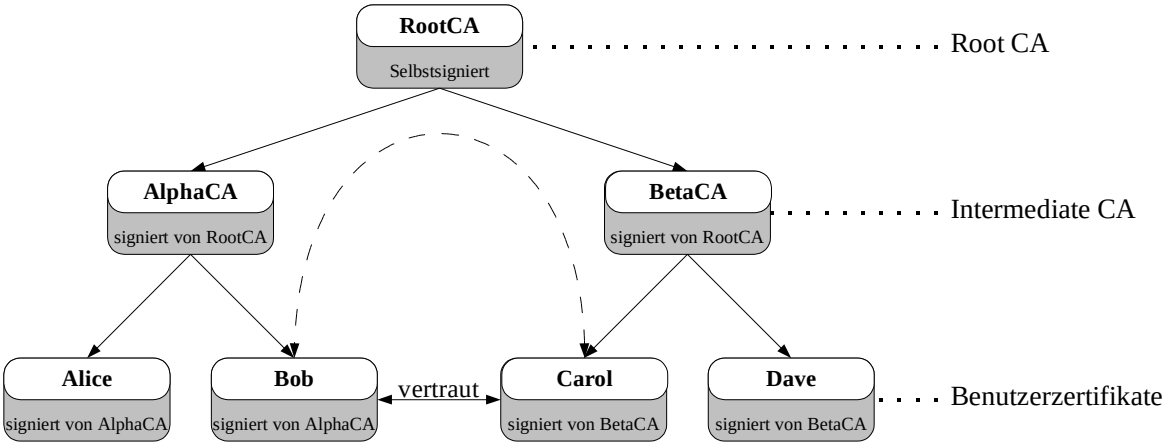


Abbildung 5 Hierarchische Vertrauenskette

Will nun Bob mit Carol kommunizieren, muss er Carols Public Key besitzen. Diesen bekommt er in Form eines Zertifikats über einen öffentlichen Server, oder direkt von Carol. Aus dem Zertifikat liest Bob, dass es von der BetaCA ausgestellt wurde. Er fordert das Zertifikat der BetaCA an, um Carols Signatur zu überprüfen. Wenn alles in Ordnung ist, geht Bob eine Stufe höher und überprüft die Signatur der BetaCA mit dem Zertifikat seines Ausstellers, der RootCA. Da Bob der RootCA vertraut, ist die Authentizität von Carols Public Key sichergestellt.

In hierarchischen Vertrauensketten werden meistens X.509 Zertifikate verwendet. Im folgenden wird kurz auf den Aufbau solcher Zertifikate eingegangen.

2.3.2 X.509 Zertifikate

Ein X.509 Zertifikat enthält folgende Informationen:

- **version** : Zur Zeit sind drei Versionen definiert. Aktuell ist Version 3.
- **serialNumber** : Eine Ganzzahl, die innerhalb einer CA jedes Zertifikat eindeutig identifiziert.
- **signature** : Enthält den Algorithmus welcher für die Signatur verwendet wurde und eventuell verwendete Parameter für den Algorithmus.
- **issuer** : Der X.500 Name der CA, welche das Zertifikat ausgestellt hat.
- **validity** : Enthält zwei Zeitstempel, welche Anfang und Ende der Gültigkeit des Zertifikats definieren.
- **subject** : Der X.500 Name des Inhabers des Zertifikats. Dieser darf aber auch ein Leerstring sein (NULL-Objekt in ASN.1) und stattdessen die Extension **subjectAltName** verwendet werden. Damit lässt sich der Name der Anwendung anpassen (z.B. DNS Name, Email Adresse).
- **subjectPublicKeyInfo** : Enthält den Public Key, sowie den Algorithmus und eventuell verwendete Parameter für den Algorithmus.

- **signatureAlgorithm** : Dieses Feld ist identisch mit dem **signature** Feld und dadurch absolut redundant.
- **signatureValue** : Enthält die digitale Signatur über alles, ausser dem Feld signatureAlgorithm .
- **extensions** : In X.509 Zertifikaten können ab der Version 3 folgende Extensions angehängt werden (Liste nicht vollständig):
  - **authorityKeyIdentifier** : Identifiziert den Schlüssel der CA, welche dieses Zertifikat untertzeichnet hat. Normalerweise enthält dieses Feld den Hash über den Public Key der CA.
  - **subjectKeyIdentifier** : Identifiziert eindeutig den Schlüsel zusammen mit Namen des Zertifikats. Normalerweise enthält dieses Feld den Hash über den Public Key.
  - **keyUsage** : Enthält einen Bit String, wovon jedes Bit für eine spezielle Verwendung des Schlüssel steht. Es sind neun Typen definiert, für Verwendungen wie Signaturen, Key-Verschlüsselung, Daten- Verschlüsselung, Signieren von X.509 Zertifikaten, Signieren von CRL's.
  - **subjectAltName** : Enthält eine Sequence von Namen. Damit lassen sich der Anwendung entsprechende Namen vergeben (z.B. DNS Name, Email Adresse).
  - **issuerAltName** : Gleich codiert wie **subjectAltName** , mit dem Namen der CA statt des Besitzers.
  - **subjectDirectoryAttributes** : Dieses Feld bietet die Möglichkeit, weitere Attribute des Besitzers zu definieren, wie z.B. das Geburtsdatum.
  - **basicConstraints** : Dieses Feld gibt dem Zertifikat die Erlaubnis andere Zertifikate auszustellen (identisch mit dem entsprechenden Bit im Feld **keyUsage**) und bestimmt die maximale Länge der folgenden Kette von Zertifikaten.
  - **nameConstraints** : Enthält die Namen für welche das Zertifikat die Erlaubnis hat, Zertifikate auszustellen.
  - **extendedKeyUsage** : Enthält weitere Verwendungszwecke des Zertifikats. Im Gegensatz zum Feld **keyUsage** sind die Verwendungszwecke durch eine OID definiert, was es vereinfacht, neue Verwendungszwecke einzuführen.
  - **cRLDisributionPoints** : Dieses Feld gibt an wo man die CRL findet und wer sie ausgestellt hat.
  - **authorityInfoAccess** : Beschreibt, wie man Informationen über den Aussteller dieses Zertifikats findet.
  - **subjectInfoAccess** : Beschreibt, wie man Informationen über den Besitzer des Zertifikats findet.

2.3.3 Certificate Revocation List (CRL)

In einer grösseren PKI besteht bald einmal die Situation, dass man ein ausgestelltes Zertifikat vor Ablauf seiner Gültigkeit sperren möchte. Sei es weil ein Firmen-Laptop mit einem Private Key gestohlen wurde, oder ein Mitarbeiter entlassen wird. Damit dies möglich ist, braucht es einen Dienst der es erlaubt, den Status eines Zertifikats abzufragen. Dazu wurden die Certificate Revocation Lists eingeführt.

Eine CRL enthält eine Liste der gesperrten Zertifikate sowie Zeitstempel, welche die Gültigkeit der CRL zeitlich beschränken. Damit niemand die Liste manipulieren oder eine gefälschte Liste einschleusen kann, ist auch die CRL digital signiert.

Eine CRL enthält folgende Informationen:

- **signature** : Identisch mit dem **signature** Feld in X.509 Zertifikaten.
- **issuer** : Der X.500 Name der CA, welche die CRL ausgestellt hat.
- **thisUpdate** : Zeitstempel, wann die CRL ausgestellt wurde.
- **nextUpdate** : Zeitstempel, wann die nächste CRL erstellt wird. Dieses Feld ist optional.
- für jedes gesperrte Zertifikat enthält die CRL folgende drei Felder:
  - **userCertificate** : Enthält die serialNumber des gesperrten Zertifikats
  - **revocationDate** : Zeitstempel, wann das Zertifikat gesperrt wurde.
  - **crlEntryExtensions** : Optionale Informationen, wie z.B. der Grund für die Sperrung des Zertifikats.
- **crlExtensions** : Optionale Informationen, wie z.B. **authorityKeyIdentifier** , **issuerAltName** .
- **algorithmIdentifier** : Dieses Feld ist wie bei den Zertifikaten identisch mit dem **signature** Feld.
- **encrypted** : Enthält die Signatur über alles ausser dem Feld **algorithmIdentifier** .

2.3.4 Online Certificate Status Protocol (OCSP)

Der grosse Nachteil von CRLs zeigt sich in einer PKI mit tausenden von Zertifikaten und mehreren Gateways. Auch um nur ein einziges Zertifikat auf seine Gültigkeit zu überprüfen, muss jeder Gateway die komplette CRL beziehen. Will man das Aktualisierungsintervall der CRL verkleinern, muss die CRL umso häufiger bezogen werden. Das alles führt zu einer (unnötigen) Belastung des Netzwerks.

Die Lösung ist ein Protokoll, welches erlaubt den Status von ein oder mehreren Zertifikaten abzufragen. Ein Server verwaltet eine Datenbank mit den Stati der

Zertifikate. Er wartet auf Anrfagen von Gateways und beantwortet diese aufgrund der Einträge in seiner Datenbank. Damit niemand diese Antworten fälschen oder verändern kann, werden die Antworten digital signiert. Um ein Zertifikat zu sperren, muss nun einfach der entsprechende Eintrag in der Datenbank geändert werden und alle Gateways sind auf dem aktuellen Stand.

Genau dies ist die Aufgabe des Online Certificate Status Protocol (OCSP).

OCSP ist in RFC2560 [9] definiert. Es spezifiziert die Daten, welche zwischen der Applikation, welche den Status eines Zertifikats wissen möchte, und dem Server, welcher die Stati der Zertifikate kennt, ausgetauscht werden müssen.

Das Protokoll besteht aus einem OCSP Request und einer OCSP Response. Beide Meldungen sind als ASN.1 Strukturen definiert (siehe Anhang 9.2). Für die Übertragung werden die Meldungen mit den ASN.1 Distinguished Encoding Rules (DER) codiert.

Abbildung 6 zeigt ein mögliches Szenario mit einem OCSP Server und mehreren Gateways und Roadwarrior.

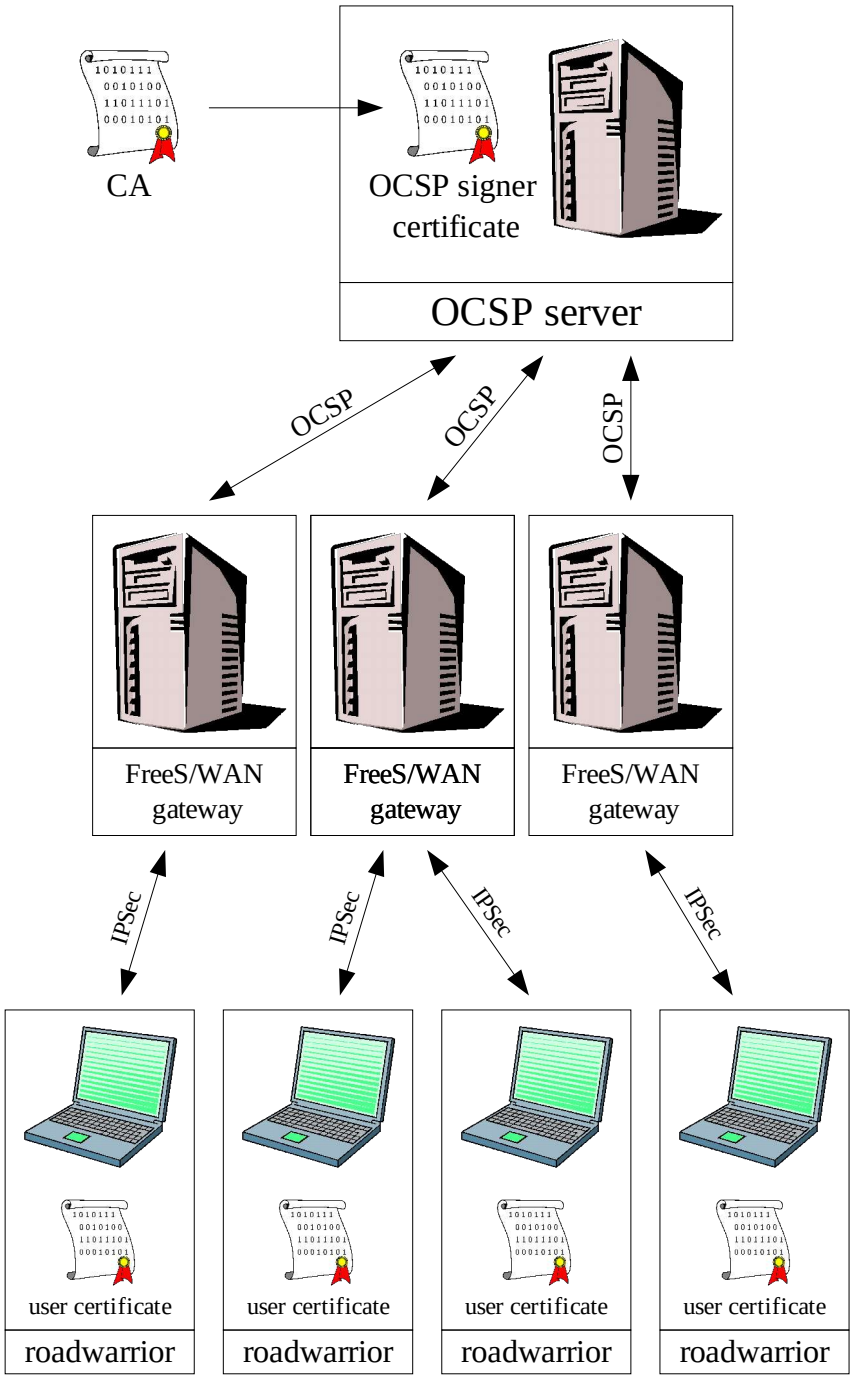


Abbildung 6 Mögliches OCSP Szenario

OCSP Request

Ein OCSP Request beinhaltet:

- Protokoll Version
- Eindeutige Kennung der Zertifikate für die der Status gewünscht wird
- Optionale Extensions

Beim Empfang eines Requests, prüft der OCSP Responder ob:

- die Meldung dem OCSP Syntax konform ist (gemäss RFC2560 [9])
- der Request alle Informationen enthält, die der OCSP Responder benötigt um eine Antwort zu generieren

Falls eine der obengenannten Bedingungen nicht erfüllt wird, erzeugt der OCSP Responder eine Fehlermeldung, ansonsten antwortet er mit einer OCSP Resonse.

OCSP Response

Es gibt verschiedene Typen von OCSP Responses. Eine OCSP Response besteht aus einer OID, welche den Typ der Response kennzeichnet und einem Octet String, welcher die eigentliche Response beinhaltet. Der Typ *id-pkix-ocsp-basic* muss von allen OCSP Applikationen unterstützt werden.

Alle Antworten, die der Server versendet, müssen digital signiert werden. Dies geschieht entweder durch die CA selbst, einem Responder dem explizit vertraut wird, oder einem von der CA authorisierten Responder, dessen Zertifikat die ExtendedKeyUsage "id-kp-OCSPSigning" enthält.

Eine Response besteht aus:

- der Version des Response Syntax
- dem Namen des Responders
- einer Response für jedes Zertifikat des Requests
- optionalen Extensions
- dem Algorithmusverfahren der Signatur
- einer Signatur über den Hash der Response

Die Responses für jedes einzelne Zertifikat bestehen aus:

- einer eindeutigen Kennung des Zertifikats
- dem Status des Zertifikats (GOOD, REVOKED, UNKOWN)
- dem Gültigkeitsintervall der Response
- optionalen Extensions

Falls ein Fehler auftritt, sendet der OCSP-Server eine der folgenden Fehlermeldungen:

- **malformedRequest** : Der Request ist nicht konform mit dem OCSP Request Syntax.
- **internalError** : Der Server erreichte einen ungültigen Zustand.
- **tryLater** : Der Server ist momentan unfähig, den Request zu beantworten.
- **sigRequired** : Der Server verlangt einen signierten Request.
- **unauthorized** : Der Client ist nicht berechtigt, die Anfrage zu machen.

2.4 ASN.1

(ITU-T Recommendation X.680 [11] / ISO/IEC 8824-1)

ASN.1 (Abstract Syntax Notation One) ist, wie der Name sagt, eine Notation für eine abstrakte Syntax. Sie ermöglicht das Beschreiben von Datenstrukturen unabhängig von Benutzer- und Maschinenspezifischen Codierungsverfahren. Wie diese Daten codiert werden um sie über einen physikalischen Übertragungskanal zu transportieren, wird durch entsprechende Encoding Rules spezifiziert (z.B. BER).

2.4.1 Einfache Datentypen

ASN.1 definiert eine Menge von einfachen Datentypen.

Eine Auswahl der gebräuchlichsten Typen:

```
BOOLEAN
INTEGER
BIT STRING
OCTET STRING
NULL
OID
ENUMERATED
CONSTRUCTED
SEQUENCE [OF]
SET [OF]
```

2.4.2 Typendefinition

Es lassen sich eigene Typen definieren, welche auf einfachen Datentypen basieren.

```
Syntax: <type name> ::= <type>
```

Beispiele:

```
Age ::= INTEGER
IpAddress ::= OCTET STRING
DayOfWeek ::= ENUMERATED {
      Monday      (0),
      Tuesday     (1),
      Wednesday   (2),
      Thursday    (3),
      Friday      (4),
      Saturday    (5),
      Sunday      (6)
}
```

Mit strukturierten Datentypen lassen sich einzelne Variablen gruppieren. Man unterscheidet zwischen Datenstrukturen geordneter (SEQUENCE) und ungeordneter (SET) Reihenfolge.

```
UserAccount ::= SEQUENCE {
      username VisibleString,
      password VisibleString,
      accountNr INTEGER
}
```

Mit den Typen SEQUENCE OF bzw. SET OF lassen sich mehrere Werte des gleichen Typs zusammenfassen (entsprechend einem Array).

```
Accounts ::= SEQUENCE OF UserAccount
AllowedHosts ::= SET OF IpAddress
```

OID Typendefinition

Object Identifier (OID) sind in einer hierarchischen Struktur organisiert, damit jedem Objekt eine eindeutige Bezeichnung zugeordnet werden kann.

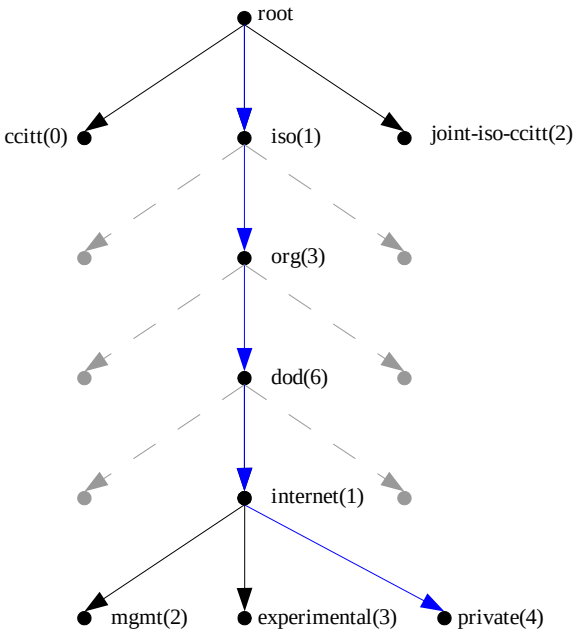


Abbildung 7 OID Hierarchie

In ASN.1 kann ein Objekt entweder von der Wurzel aus beschrieben werden, oder sich auf ein anderes, übergeordnetes Objekt beziehen.

```
internet ::= OBJECT IDENTIFIER { iso(1) org(3) dod(6) 1 }
private ::= OBJECT IDENTIFIER { internet 4 }
```

Subtypen Definition

Mit der Definition von Subtypen lassen sich gewisse Typen einschränken. Ein Subtyp enthält immer eine Untermenge des Typs auf dem er basiert.

```
Syntax: <subtype name> ::= <type> ( <constraint> )
```

Beispiele:

```
Age ::= INTEGER ( 0..128 )
IpAddress ::= OCTET STRING ( SIZE(4) )
Workday ::= DayOfWeek ( Monday | Tuesday | Wednesday | Thursday |
Friday }
Weekend ::= DayOfWeek ( Saturday | Sunday )
Hash ::= BIT STRING ( SIZE(25) )
```

Wertzuweisung

Jedem Typ lassen sich nun beliebige, seinem Gültigkeitsbereich entsprechende Werte zuweisen.

```
Syntax: <value name> <type> ::= <value>
```

Beispiele:

```
myAge Age ::= 52
serverAddress IpAddress ::= 'C055932A'H
today DayOfWeek ::= Tuesday
keyHash Hash ::= '01100101000101101010011O1'B

myAccount UserAccount ::= {
      username "lisa",
      password "(*4=ZgçFfB",
      accountNr 1234
}

localnet AllowedHosts ::= {
      serverAddress,
      gatewayAddress,
      hostAddress
}
```

2.5 ASN.1 encoding rules: BER / CER / DER

(ITU-T Recommendation X.690 [12] / ISO/IEC 8825-1)

Dieser Standard definiert eine Menge von Basic Encoding Rules (BER), welche auf ASN.1 spezifizierten Datentypen angewandt werden können. Die Anwendung dieser Regeln erzeugen eine Transportsyntax für ASN.1 Objekte. Weiter definiert der Standard die Canonical Encoding Rules (CER) und die Distinguished Encoding Rules (DER), beide definieren strengere Auflagen zur Codierung von Datentypen. Der Hauptunterschied zwischen den Beiden ist, dass DER in ein Format endlicher Länge codiert und CER in ein Format unendlicher Länge. DER ist somit besser geeignet für kleine Werte, CER besser für grosse Werte.

2.5.1 Struktur eines BER-codierten Datenobjekts

Struktur eines einfachen oder zusammengesetzten Datenobjekts:

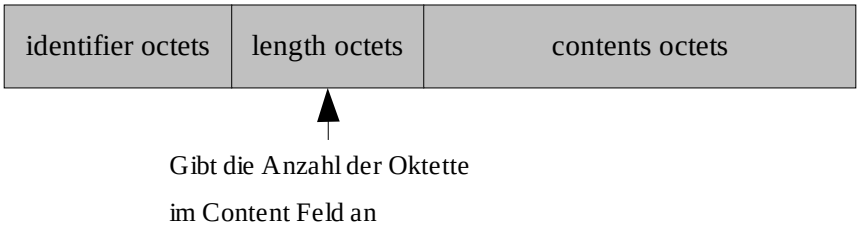


Abbildung 8 Einfaches BER Datenobjekt

Ein einfaches Datenobjekt besteht aus einem Identifier, einem Längenfeld und dem Inhalt. Der Identifier bestimmt den Datentyp des Inhalts (Boolean, Integer, Sequence, ...), das Längenfeld gibt die Länge des Inhalts in Byte an.

Struktur eines alternativen zusammengesetzten Datenobjekts:

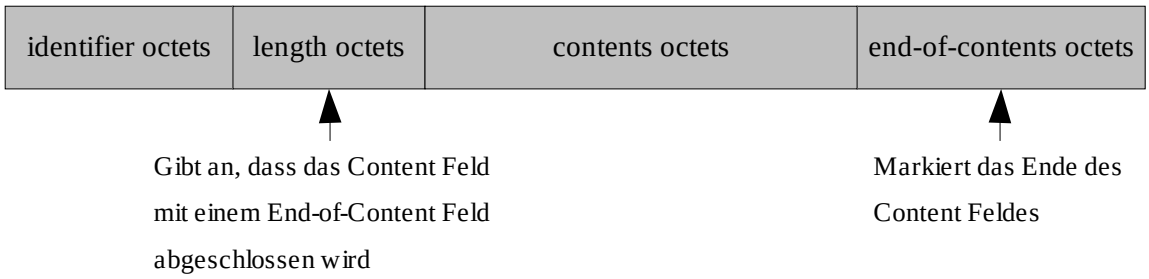


Abbildung 9 Zusammengesetztes BER Datenobjekt

Alternativ kann ein zusammengesetztes Datenobjekt auch durch ein End-of-Contents Feld terminiert werden.

Identifier

Das Identifier Feld beschreibt den Typ (Klasse und Nummer) des Datenobjekts.

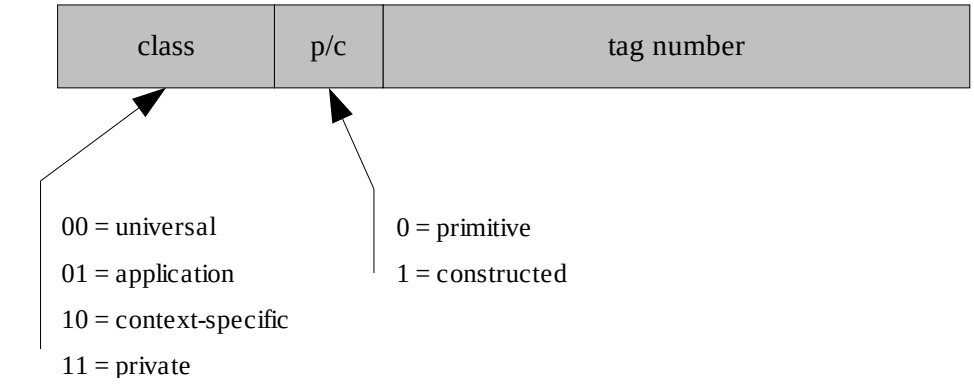


Abbildung 10 Identifier Feld

Die folgende Tabelle enthält einen Auszug der Tag-Nummern von Datentypen der Klasse Universal:

Datentyp	*Tag-Nummer*	*Constructed*	*Hex-Value*
Boolean	1	0	0x01
Integer	2	0	0x02
Bit String	3	0	0x03
Octet String	4	0	0x04
Null	5	0	0x05
Object ID	6	0	0x06
Object Descriptor	7	0	0x07
Real	9	0	0x09
Enumerated	10	0	0x0A
Sequence [of]	16	1	0x30
Set [of]	17	1	0x31

Length

Das Length Feld gibt die Länge des Content Feldes in Bytes an. Es sind zwei verschiedene Formen für die Codierung der Länge spezifiziert: die endliche und die unendliche Form.

- Bei der unendlichen Form gibt das Length Feld an, dass das Content Feld mit einem End-of-Content Feld abgeschlossen wird.
- Bei der endlichen Form wird Bit 8 auf 0 gesetzt und die Länge in den Bits 7 bis 1 codiert. Für Werte grösser als 127 wird Bit 8 auf 1 gesetzt. Bits 7 bis 1 beschreiben dann die Anzahl der folgenden Bytes, welche die Längeninformation enthalten.

Beispiele (endliche Form):

```
67    =     01000011₂
201   =     10000001 11001001₂
9874  =     10000010 00100110 10010010₂
```

2.5.2 Einfache Datentypen

Boolean

Ein Boolean ist immer primitiv und von der Länge 1. Die Tag-Nummer des Typs Boolean ist 01_{16}. Der Content ist 00_{16} für den Wert FALSE und ungleich 00_{16} für den Wert TRUE.

Beispiele:

```
FALSE =     01 01 00₁₆
TRUE  =     01 01 FF₁₆
```

Integer

Ein Integer wird als Zweierkomplement codiert. Dabei sollen Bit 8 bis 1 des ersten und Bit 8 des zweiten Oktetts des Contents nie aus lauter Nullen oder Einsen bestehen. Damit wird erreicht, dass ein Integer immer in der kleinstmöglichsten Form codiert wird.

Beispiele:

```
-129  =     11111111 01111111₂      =     02 02 FF 7F₁₆
-128  =     11111111 10000000₂      =     02 01 80₁₆
-1    =     11111111 11111111₂      =     02 01 FF₁₆
0     =     00000000 00000000₂      =     02 00₁₆
1     =     00000000 00000001₂      =     02 01 01₁₆
127   =     00000000 01111111₂      =     02 01 7F₁₆
128   =     00000000 10000000₂      =     02 02 00 80₁₆
```

Bit String

Beim Bit String gibt das erste Oktett an, wieviele Bits im letzten Oktett ungenutzt sind. Die Anzahl wird als unsigned Integer codiert und muss zwischen 0 und 7 liegen.

Beispiel:

```
                        11001101 010₂          Bit String
=           00000101 11001101 01000000₂        Padded
=    03 03     05       CD       40₁₆         ASN.1 Object
```

Octet String

Beim Octet String werden die Daten direkt in den Content geschrieben.

Beispiel:

```
                 04 F7 B8 3D 72 02₁₆      Octet String
=      04 06 04 F7 B8 3D 72 02₁₆      ASN.1 Object
```

Null

Ein Objekt vom Typ Null enthält keine Oktette im Content.

Beispiel:

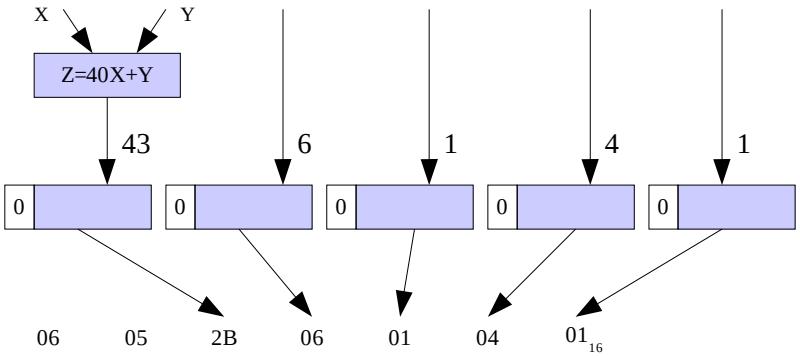
```
05 00₁₆
```

OID

Object Identifier werden als Liste von aneinandergereihten Subidentifiern codiert. Jeder Subidentifier wird als ein oder mehrere Oktette codiert, wobei jeweils Bit 8 angibt ob noch weitere Oktette zum aktuellen Subidentifier gehören. Bit 8 des letzten Oktetts ist 0. Bits 7 bis 1 der zusammengehörenden Oktette bilden zusammen einen unsigned Integer mit dem Wert des Subidentifiers.

Beispiel:

```
enterprise OBJECT IDENTIFIER ::=
{ iso(1) org(3) dod(6) internet(1) private(4) 1 }
```



Enumerated

Objekte vom Typ Enumerated werden als Integer des entsprechenden Wertes codiert.

Sequence

Eine Sequence ist ist immer constructed und besteht aus den codierten ASN.1 Objekten in der Reihenfolge ihrer Definition. Das Keyword OPTIONAL gibt an, dass ein Wert auch weggelassen werden kann. Das Keyword DEFAULT kennzeichnet einen Wert, welcher als Defaultwert angenommen wird wenn er nicht angegeben ist.

Beispiel:

```
Person ::= SEQUENCE {
      name VisibleString,
      age INTEGER
}

Person p ::= {
      name  "Fritz",
      age   28
}
```

Set

Ein Set ist, ähnlich wie die Sequence, ein zusammengesetzter Typ aus codierten ASN.1 Objekten. Anders als bei der Sequence spielt es aber keine Rolle, in welcher Reihenfolge die einzelnen Objekte definiert oder codiert wurden. Auch beim Typ Set gelten die Keywords OPTIONAL und DEFAULT.

3 Analyse

3.1 FreeS/WAN

Wie in den Grundlagen gezeigt, ist der Daemon Pluto für das Aufbauen und Managen der Verbindungen zuständig. Die OCSP Funktion soll ähnlich dem CRL-Fetching in Pluto eingefügt werden.

Um Änderungen am Quellcode von Pluto durchführen zu können muss zuerst der Aufbau von Pluto und die Abläufe beim Verbindungsaufbau verstanden werden.

Da während unserer Diplomarbeit FreeS/WAN 2.02 mit dem X.509 Patch 1.4.7 erschienen ist, beziehen sich alle Angaben auf die neuen Versionen.

3.1.1 Aufbau von Pluto

Die Software Pluto besteht aus 43 Modulen, wobei jedes Modul einen bestimmten Aufgabenbereich übernimmt. Auf diese Art und Weise erreicht man eine Kapselung, welche die Übersicht beim Programmieren fördert und das Einfügen von neuen Funktionen erleichtert.

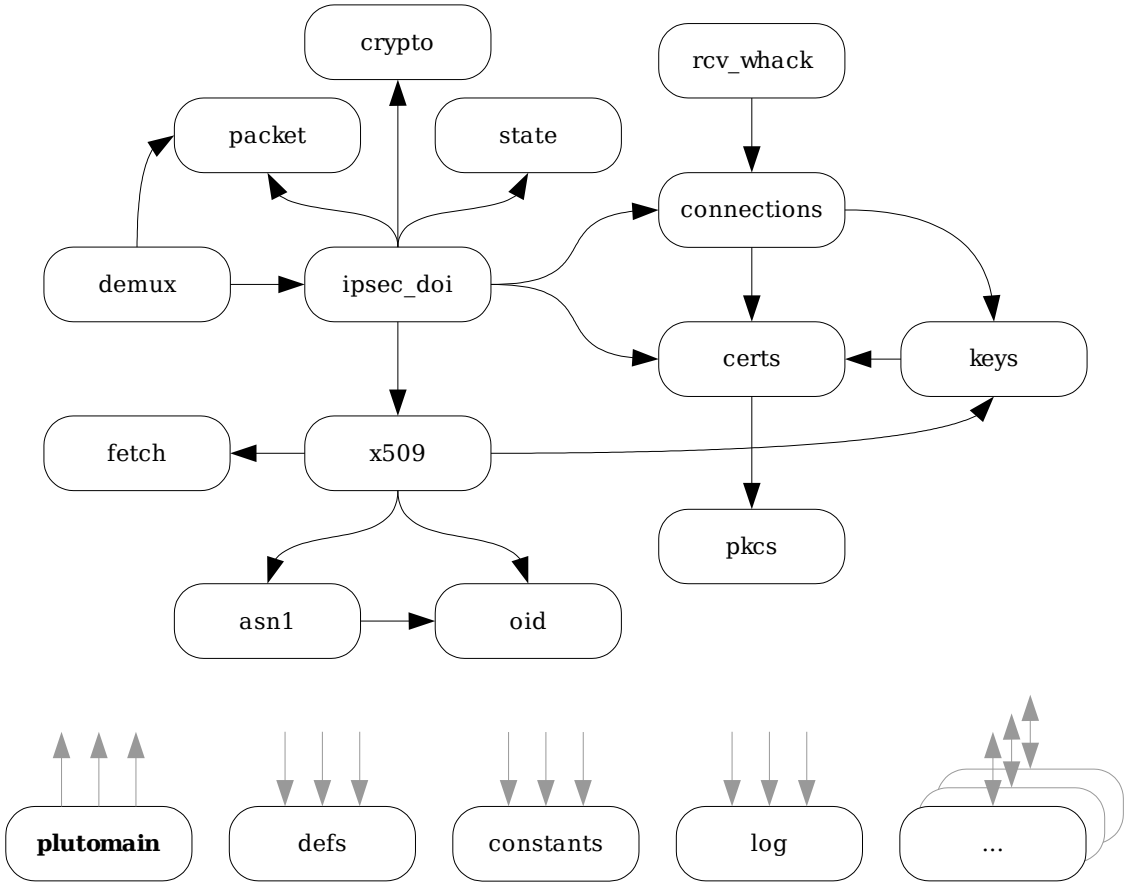


Abbildung 11 Aufbau Pluto

Abbildung 11 zeigt die wichtigsten Module von Pluto mit den jeweiligen Abhängigkeiten. Zur Verbesserung der Übersicht wurden nur die wichtigsten Module und Zusammenhänge aufgeführt, zum Beispiel wurden die Module im Zusammenhang mit der Opportunistic Encryption weggelassen, da sie für unsere Anwendung nicht von Bedeutung sind.

Das Hauptmodul von Pluto ist *plutomain*. Es enthält die Startfunktion *main* und startet, initialisiert und beendet das Programm. Mehrere Module stellen Hilfsfunktionen zur Verfügung, zum Beispiel die Module *defs*, *constants*, *log* und *timer*.

Modul	*Beschreibung*
plutomain	Hauptmodul. Beinhaltet die Hauptfunktion welche die Kommandozeilen-Optionen parst, alle Module initialisiert und startet und schliesslich wieder beendet.
demux	Empfängt Pakete, analysiert den Inhalt und gibt die Informationen an das Modul *ipsec_doi* weiter.
ipsec_doi	Enthält die eigentliche IKE-Logik. Reagiert auf empfangene Pakete und antwortet mit eigenen Paketen. Greift auf Verbindungsinformationen und Zertifikate aus den Modulen *connections*, *certs* und *x509* zu.
packet	Enthält Hilfsfunktionen zum Parsen und Erstellen von IKE Meldungen.
rcv_whack	Empfängt Befehle von Bedienungs-Tool Whack und führt die entsprechenden Aktionen aus.
connections	Verwaltet die Verbindungen und stellt entsprechende Funktionen zur Verfügung.
certs	Verwaltet die Zertifikate und stellt entsprechende Funktionen zur Verfügung.
keys	Verwaltet die Schlüssel und stellt entsprechende Funktionen zur Verfügung.
pkcs	Stellt verschiedene Hilfsfunktionen für die Standards PKCS zur Verfügung.
crypto	Stellt Hash-Funktionen zur Verfügung.
state	Verwaltet die Stati der SA's.
x509	Enthält die Funktionen und Daten im Zusammenhang mit X.509-Zertifikaten. Speichert verkettete Listen von Zertifikaten und CA-Zertifikaten.
asn1	Stellt Funktionen für das Parsen von ASN.1 Objekten zur Verfügung.
oid	Enthält die Object ID's von verschiedenen ASN.1 Objekten. Dieses Modul wird aus dem OID Index File oid.txt erstellt.

Modul	*Beschreibung*
fetch	Enthält den Fetch-Thread. Der Fetch-Thread enthält eine Liste von URI's, welche parallel zum Hauptthread heruntergeladen werden sollen.
defs	Stellt diverse Hilfsfunktionen zur Verfügung.
constants	Enthält Definitionen von Konstanten für die Verwendung in mehreren Modulen.
log	Stellt Funktionen für die Ausgabe von Fehlermeldungen, Warnungen und Informationen zur Verfügung.
timer	Enthält einen Scheduler. Es können Events gesetzt werden, welche zu einem bestimmten Zeitpunkt ausgelöst werden.

3.1.2 Abläufe beim Verbindungsaufbau

Uns interessierte der Aufbau von Pluto natürlich speziell in Bezug auf unsere Aufgabenstellung. Das Zertifikat des Roadwarriors sollte beim Verbindungsaufbau überprüft werden, der Ablauf ist dabei ähnlich dem CRL-Verfahren.

Der Verbindungsaufbau geschieht bei IPSec, wie in den Grundlagen beschrieben, mit IKE. Hier interessieren speziell die Schritte 5 und 6, bei denen die ausgehandelte SA und die Schlüssel mittels dem privaten Schlüssel des Roadwarrior authentisiert werden. Die Authentizität der Signatur wird mit dem übermittelten Zertifikat überprüft. Bei der Überprüfung des Zertifikates auf seine Gültigkeit soll unser OCSP Mechanismus eingreifen.

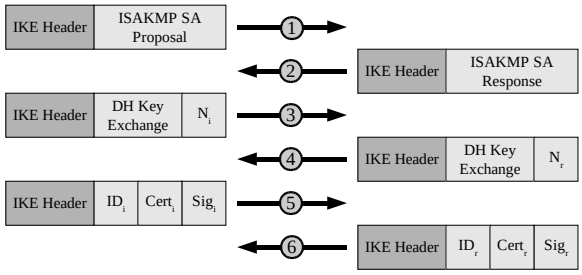


Abbildung 12 IKE

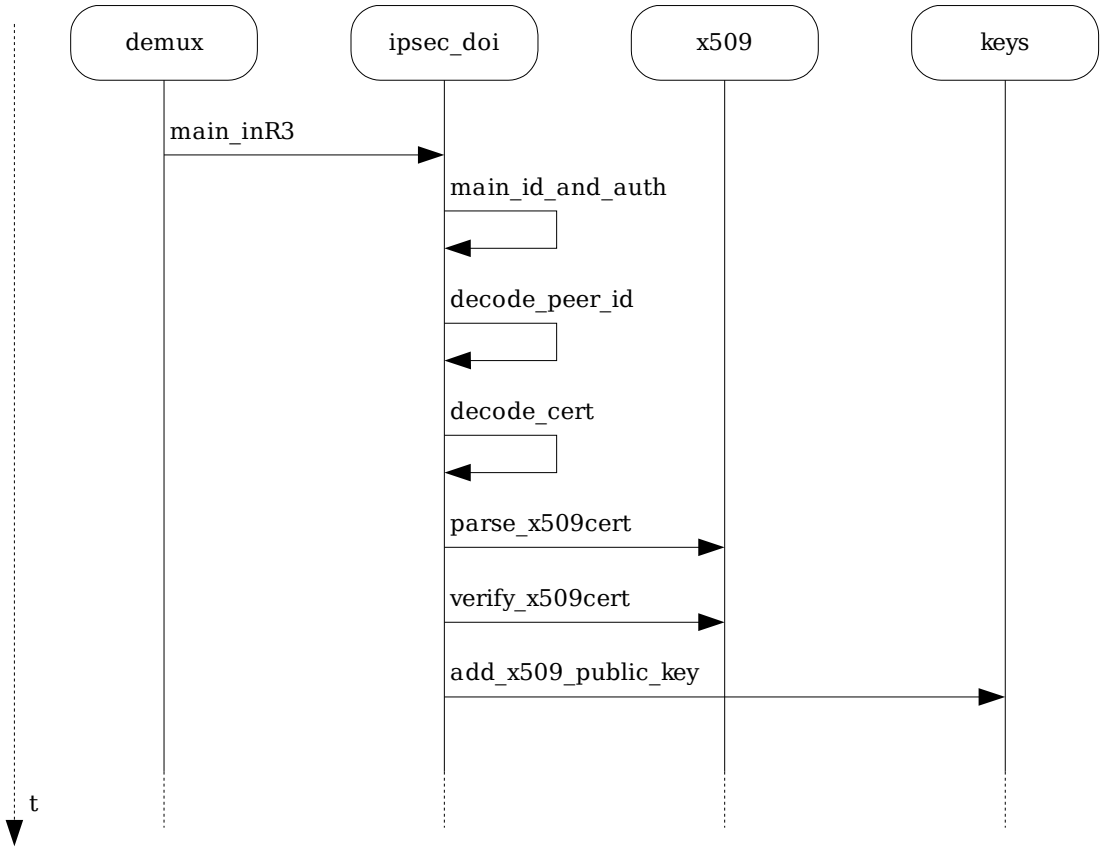


Abbildung 13 Sequenzdiagramm Verbindungsaufbau

Abbildung 13 zeigt als Beispiel die Funktionshierarchie über die verschiedenen Module, wenn die letzte Meldung von Pluto empfangen wird. Wiederum werden nur die wichtigsten Abläufe gezeigt.

Das Modul *demux* empfängt und parst das Paket, und ruft danach im Modul *ipsec_doi* die Funktion main_inR3 auf. Diese Funktion wertet den Inhalt des Pakets genauer aus, liest das angehängte Zertifikat (*decode_cert*) und ruft im *x509*-Modul die Funktion *parse_x509cert* auf. Diese Funktion parst das X.509-Zertifikat und speichert die benötigten Informationen in einer Struktur.

War das parsen des Zertifikats erfolgreich, wird die Gültigkeit des Zertifikats im Modul *x509* mit der Funktion *verify_x509cert* überprüft. Ist das Zertifikat gültig, wird der Public Key in eine verkettete Liste von Schlüsseln im Modul *keys* aufgenommen (*add_x509_public_key*).

In der Funktion *verify_x509cert* wird die ganze Zertifikatshierarchie bis zur CA überprüft. Jedes Zertifikat wird zudem mittels der CRL verifiziert. Genau an diesem Punkt soll nun statt der CRL-Überprüfung unsere OCSP Validierung stattfinden.

3.1.3 Automatisches CRL fetching

FreeS/WAN unterstützt den automatischen Download von CRL's, wahlweise über HTTP oder LDAP. Die URI der CRL wird im Zertifikat mit der Extension *cRLDistributionPoints* angegeben. Der eigentliche Download der CRL wird über einen eigenen Thread durchgeführt.

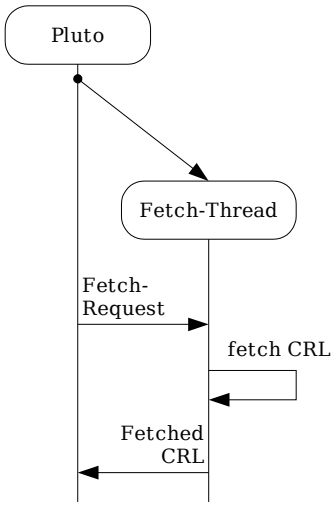


Abbildung 14 Sequenzdiagramm CRL-fetching

Wie Abbildung 14 zeigt, startet Pluto einen Fetch-Thread, welcher eine Liste von Fetch-Aufträgen führt. Pluto gibt die Fetch-Requests an den Fetch-Thread weiter, welcher die CRL's eigenständig herunterlädt. Die Threads kommunizieren über den gemeinsamen Speicher, deshalb wird als Synchronisierungsmechanismus Mutex verwendet.

Der Fetch-Thread wird nach dem Download für eine bestimmte Zeit (crlcheckinterval) schlafen gelegt. Nach dieser Zeit wird die Liste von CRL's überprüft und bald ablaufende CRL's neu heruntergeladen. Wird eine CRL angefordert, welche noch nicht heruntergeladen wurde, wird der Fetch-Thread von Pluto explixit aufgeweckt.

3.1.4 Aufstarten von Pluto / FreeS/WAN

Das Aufstarten von FreeS/WAN geschieht über eine recht komplizierte Verknüpfung von Shell-Scripts. Abbildung 15 zeigt die wichtigsten Abläufe.

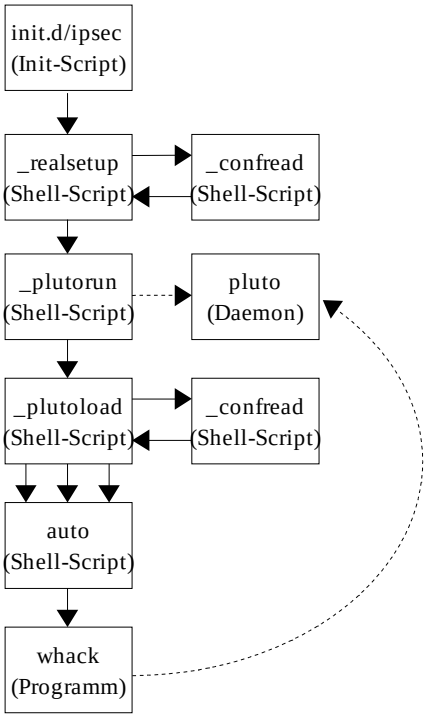


Abbildung 15 Ablauf Start von Pluto

Initialisiert wird das Aufstarten mit */etc/init.d/ipsec start* (nicht zu verwechseln mit dem gleichnamigen Shell-Script *ipsec*). Über *_realsetup* wird das Script *_plutorun* aufgerufen. Dieses liest die erste Konfigurations-Sektion über das Script *_confread* ein und startet den Daemon Pluto. Danach wird das Script *_plutoload* aufgerufen, welches die restliche Konfiguration von Pluto übernimmt. Wird Pluto vorzeitig beendet (durch einen Segmentation Fault oder ähnliches), so wird der Daemon von *_plutorun* automatisch wieder gestartet und danach wieder *_plutoload* aufgerufen.

Das Script *_plutoload* liest über das Script *_confread* die Konfigurations-Datei ipsec.conf und führt je nach Konfiguration mehrere Aufrufe von *ipsec auto* durch. So wird bei der Konfiguration einer Verbindung mit “auto=add” für diese Verbindung *ipsec auto --add ...* mit allen Verbindungsparametern aufgerufen und so Pluto eine Verbindungskonfiguration übergeben. Entsprechend wird bei der Konfiguration von “auto=start” danach auch *ipsec auto –up ...* aufgerufen.
Durch dieses Vorgehen ist es im Script *auto* nicht möglich zu erkennen, ob *ipsec auto --up ...* durch das Start-Script aufgerufen wurde oder durch den Benutzer.

Die gezeigten Abläufe wurden zur besseren Verständlichkeit und Übersichtlichkeit vereinfacht.

3.2 OpenSSL OCSP Server

OpenSSL [17] enthält ab Version 0.9.7 einen einfachen OCSP Server und Client. Somit steht erstmals eine freie Implementation des OCSP Protokolls zur Verfügung.

Der Server liest den Status der Zertifikate aus einer Index-Datei im OpenSSL CA-Format. Wurde OpenSSL bereits zur Zertifizierung (als CA) benutzt, kann der OCSP Server sehr einfach konfiguriert und benutzt werden.

Die Bedienung des OCSP Servers wird im Kapitel 7.4.3 ausführlich erklärt.

3.2.1 Funktionsumfang

Der OpenSSL OCSP Server stellt die wichtigsten OCSP Funktionalitäten zur Verfügung und ist konform zum Standard RFC 2560 [9]. Der Einsatz als produktiver Server wird allerdings nicht empfohlen, da die Implementierung einige Schwächen aufweist.

Einschränkungen des OpenSSL OCSP Server:

- Als HTTP Request wird nur der POST Mechanismus akzeptiert.
- Die Requests werden seriell abgearbeitet, das heisst während der Bearbeitung eines Requests kann kein weiterer empfangen werden.
- Die Index-Datei im Text-Format ist ineffizient für viele ausgestellte Zertifikate.
- Es können nur Requests für eine CA (eine Index-Datei) beantwortet werden.
- Im Fehlerfall (ungültige Abfrage, ...) wird der Server beendet und muss neu gestartet werden, was eine Denial of Service Attacke ermöglicht. In der Version 0.9.7c, die während der Diplomarbeit freigegeben wurde, ist dieses Problem allerdings behoben.

3.3 CURL

3.3.1 Zweck

CURL (Client for URLs, [18]) ist ein Projekt, das ein Kommandozeilen-Tool und eine Bibliothek für den Download und das Senden von Dateien via URL's zur Verfügung stellt. In unserem Projekt kam die Bibliothek *libcurl* zur Anwendung.

libcurl stellt eine einfache und flexible API zur Verfügung, welche verschiedenste Protokolle unterstützt. Zu diesen gehören unter anderen HTTP, HTTPS, FTP und LDAP.

Da Pluto für das CRL-Fetching bereits CURL benutzt, lag es nahe für unsere Anwendung ebenfalls die Curl-Bibliothek zu benutzen, welche die möglichen Fehler bei der Kommunikation mit dem Server selber behandelt.

3.3.2 API

libcurl stellt zwei Interfaces zur Verfügung: *easy* und *multi*. Das *easy* Interface ermöglicht den einfachen, synchronen (blockierenden) Download, das *multi* Interface ist die asynchrone Variante, mit dem mehrere Downloads gleichzeitig gestartet werden können. Für unsere Applikation war das *easy* Interface ausreichend, welches kurz beschrieben werden soll.

libcurl Interface *easy*

Funktion	*Beschreibung*
curl_global_init()	Initialisiert die CURL Bibliothek. Soll vor der Verwendung der Bibliothek genau einmal aufgerufen werden. Als Parameter wird angegeben, welches Modul initialisiert werden soll. Wird CURL_GLOBAL_SSL übergeben, wird OpenSSL für diese Applikation initialisiert, falls *libcurl* mit dem entsprechenden Support kompiliert wurde. Mit der OpenSSL Bibliothek wird der Zugriff auf HTTPS URL's möglich.
curl_global_cleanup()	Gibt die von *libcurl* benötigten Ressourcen wieder frei. Soll genau einmal aufgerufen werden.
curl_easy_init()	Erstellt den Kontext für einen Request. Gibt einen CURL *easy* handle zurück, mit welchen die anderen e*asy* Funktionen aufgerufen werden können.
curl_easy_cleanup()	Gibt den erstellten Kontext für einen Request wieder frei.
curl_easy_setopt()	Ermöglicht das Setzen von Optionen für den Request. Dazu gehören unter anderem die URL und die zu sendenden Header.
curl_easy_perform()	Führt den tatsächlichen Request aus. Die Funktion blockiert während des Requests.
curl_easy_getinfo()	Ermöglicht das Abfragen von Informationen zum erfolgten Request.
curl_slist_append()	Ermöglicht das Anhängen eines Headers an eine Headerliste.
curl_slist_free_all()	Gibt die Headerliste wieder frei.

4 Design

Da OCSP eine Alternative zu CRL's darstellt, setzt unser Code an den selben Stellen an, an denen auch CRL's verwendet werden.

4.1 Verbindungsaufbau

Wenn ein Host eine Verbindung aufbauen will, muss sein Zertifikat zuerst auf Gültigkeit überprüft werden.

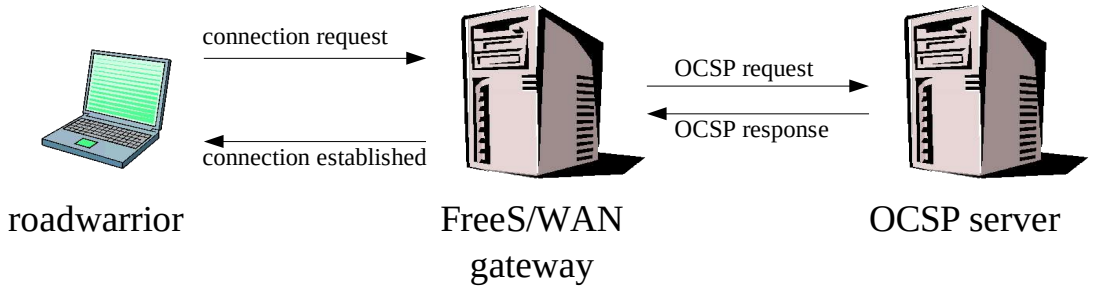


Abbildung 16 Ablauf OCSP Request

Dies führt jedoch zu einer markanten Verzögerung beim Verbindungsaufbau, da der Roadwarrior und der Gateway warten müssen, bis der OCSP Server die Anfrage verarbeitet hat und eine entsprechende Antwort retourniert. Besser wäre eine asynchrone Abfrage des OCSP Servers. Zusätzlich wird auch das Netzwerk unnötig belastet, da beim mehrmaligen Verbinden stets der aktuelle Status abgefragt wird, obwohl die letzte Anfrage eventuell noch gültig gewesen wäre. Wir führen deshalb einen Cache ein, welcher Antworten des OCSP Servers zwischenspeichert.

Empfängt der Gateway ein Zertifikat mit unbekanntem Status, gibt er dem Cache den Auftrag, das Zertifikat zu überprüfen. Statt auf dessen Antwort zu warten, fährt er unverzüglich mit dem Aufbau der Verbindung fort. Diese Vorgehensweise erlaubt es einem gesperrten Zertifikat, welches sich noch nicht im Cache befindet, ein letztes Mal eine Verbindung aufzubauen. Will man auch diesen Fall ausschliessen, sollte man die *strictCrlPolicy* verwenden. Sie bedingt, analog zu den CRL's, die strikte Prüfung eines Zertifikats. Ist sie gesetzt, werden Verbindungsanfragen von unbekannten Zertifikaten verworfen, bis eine positive Antwort des OCSP Servers im Cache eingetroffen ist.

4.1.1 Ablauf

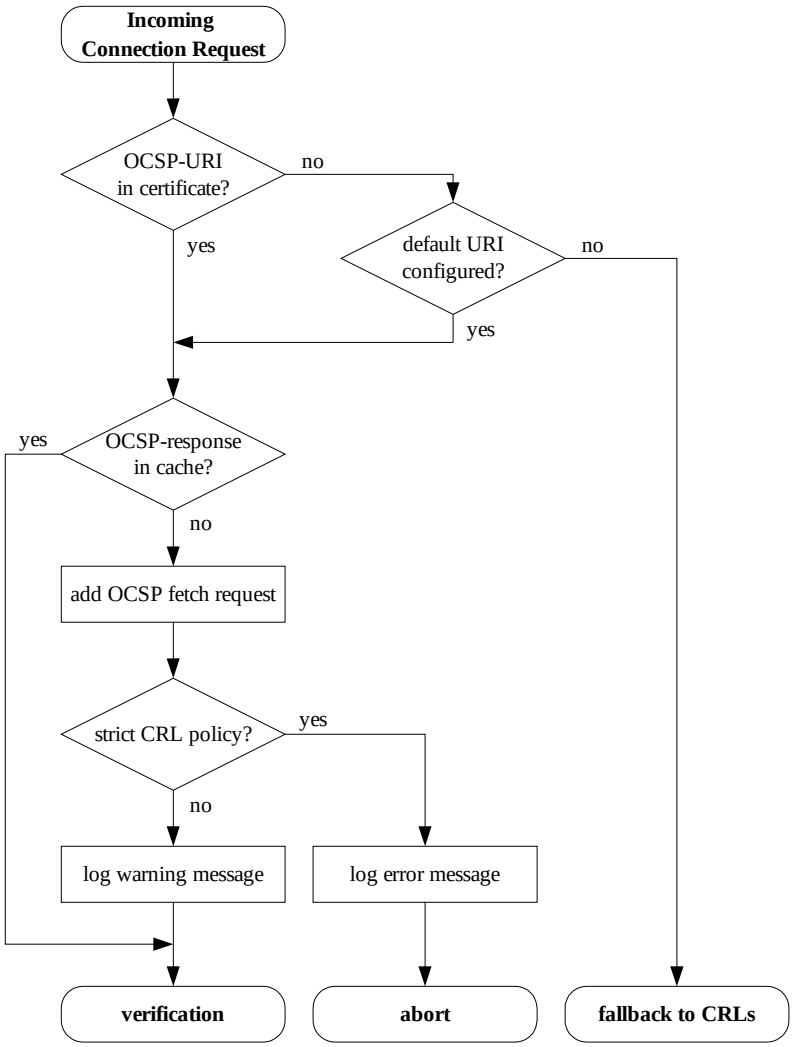


Abbildung 17 Flussdiagramm Verbindungsaufbau mit OCSP

Als Erstes muss überprüft werden, ob überhaupt OCSP verwendet wird. Dazu muss eine *AuthorityInfoAccess* Extension im zu überprüfenden Zertifikat vorhanden sein, oder eine OSCP URI im Konfigurationsfile von FreeS/WAN stehen. Ist dies der Fall, wird der Cache nach dem entsprechenden Zertifikat angefragt. Wenn der Status des Zertifikats noch nicht im Cache vorhanden ist, wird der Cache beauftragt, den Status beim OCSP Server anzufragen.

4.2 OCSP Cache

Der Cache hat die Aufgabe, gültige Antworten eines OCSP Servers zwischenzuspeichern und zu aktualisieren. Dazu ist es notwendig, dass eine Datenstruktur angelegt wird, welche alle relevanten Informationen im Speicher ablegt. Damit auch bei grossen PKI's der Speicherbedarf minimal bleibt, wurde darauf geachtet, dass nur die allernötigsten Informationen behalten werden.

Wenn ein Zertifikat angefragt werden soll, wird es zuerst im Main-Cache eingetragen. Dabei wird gleichzeitig die Gültigkeitsangabe auf 0 gesetzt. Danach wird der Fetch-Thread aus dem Sleep-Status geweckt, welcher anhand der Gültigkeitsangabe erkennt, dass der entsprechende OCSP Server angefragt werden muss.

Sobald die Antwort vom OCSP Server eintrifft, werden die Zertifikate mit Gültigkeitsangabe im Main-Cache abgelegt und regelmässig aktualisiert. Antworten ohne Gültigkeitsangabe werden in den One-Timer-Cache verschoben. Sie dürfen nur einmal und nur innerhalb kurzer Zeit (definiert zur Kompilierzeit als OCSP_DEFAULT_VALID_TIME) verwendet werden.

4.2.1 Main-Cache

Der Main-Cache besteht den Strukturen *ocsp_location* und *ocsp_certinfo*. Er enthält eine verkettete Liste von *ocsp_location*s, welche je für einen OCSP-Server stehen.

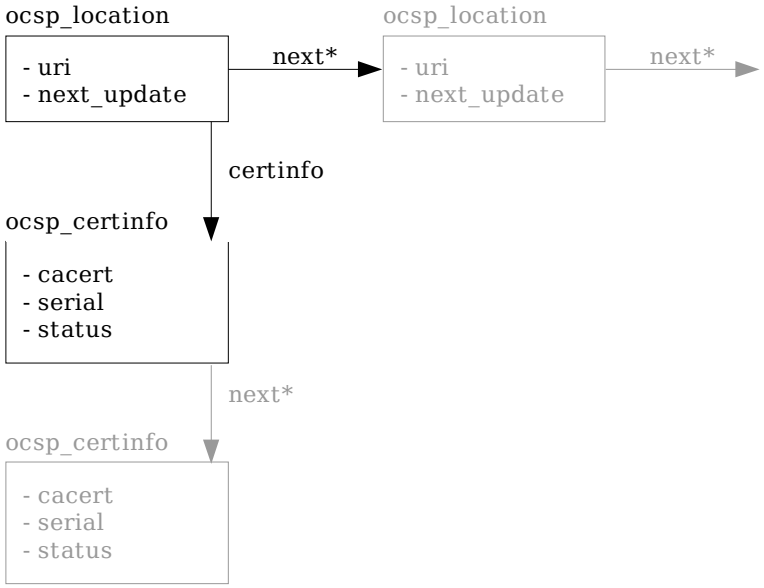


Abbildung 18 Datenstruktur Main-Cache

OCSP Location

Jede *ocsp_location* beinhaltet eine URI mit der Adresse des Servers, einen Zeitstempel, wann das nächste Update auf diesem Server verfügbar ist, sowie einen Pointer auf eine verkettete Liste von *ocsp_certinfo*s.

- **uri**: URI des OCSP Servers
- **next_update**: Gültigkeits-Zeitstempel
- **certinfo**: Pointer auf Liste von *ocsp_certinfos*
- **next**: Pointer auf nächste *ocsp_location*

OCSP Certinfo

Ein *ocsp_certinfo* beschreibt ein Zertifikat sowie dessen Status. Mittels des CA-Zertifikats und der Seriennummer wird das Zertifikat eindeutig identifiziert.

- **cacert**: Pointer auf CA des Zertifikats
- **serial**: Seriennummer des Zertifikats
- **status**: Status des Zertifikats
- **next**: Pointer auf nächsten *ocsp_certinfo*

4.2.2 One-Timer-Cache

Der One-Timer-Cache besteht aus der Struktur ocsp_onetimer. Er enthält eine verkette Liste von *ocsp_onetimer*.

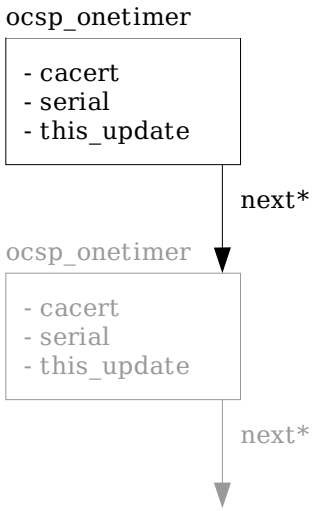


Abbildung 19 Datenstruktur One-Timer-Cache

OCSP One-Timer

Ein *ocsp_onetimer* beschreibt, ähnlich einem *ocsp_certinfo*, ein Zertifikat eindeutig. Er enthält zusätzlich einen Zeitstempel mit dem Zeitpunkt seiner Erstellung. Das Status Feld ist unnötig, da ein One-Timer immer vom Status GOOD ist bis seine Gültigkeit überschritten wurde.

- **cacert**: Pointer auf CA des Zertifikats
- **serial**: Seriennummer des Zertifikats
- **this_update**: Zeitpunkt der Erstellung
- **next**: Pointer auf nächsten *ocsp_onetimer*

4.3 Eingehende OCSP Antwort

Da die Antworten des OCSP Servers darüber entscheiden, ob ein Zertifikat akzeptiert wird oder nicht, ist die Kommunikation der beiden OCSP Partner kritisch: Jede Antwort muss genaustens auf ihre Echtheit überprüft werden. Dazu wird sie vom OCSP Server digital signiert. Trotzdem behebt diese Methode nicht alle Verwundbarkeiten dieses Protokolls. Sendet der OCSP Server auf eine Anfrage eine signierte Meldung das Zertifikat sei gültig, ohne Einschränkung der Gültigkeitsdauer dieser Aussage, kann die Meldung von irgendjemandem abgefangen und zu einem beliebigen späteren Zeitpunkt wieder eingeschleust werden (Replay-Attacke). Da die Meldung eine gültige Signatur trägt, würde dem Gateway nichts auffallen und er würde das (eventuell unterdessen gesperrte) Zertifikat akzeptieren.

Deshalb wird der Anfrage zusätzlich ein zufälliger Wert (eine sogenannte *Nonce*) mitgegeben. Der Server sollte diese auch in seiner Antwort einfügen. Dadurch wird jede Meldung individuell und Replay-Attacken können effektiv verhindert werden.

Leider ist es dem Server aber gemäss RFC2560 [9] freigestellt, ob er die Nonce in die Antwort einbindet. So wird lediglich eine Warnung ins Logfile eingetragen, das die Nonce in der Antwort nicht vorhanden war. Der Grund dafür findet sich in der Möglichkeit des OCSP Servers, vorgefertigte Antworten zu produzieren (Response Pre-Production). Damit ist eine sehr viel effizientere Arbeitsweise möglich, da der OCSP Server nur bei einer Änderung der Zertifikat-Stati die neuen Antworten generiert und dann pro Anfrage nur noch die vorgefertigten Meldungen retourniert.

4.3.1 Ablauf

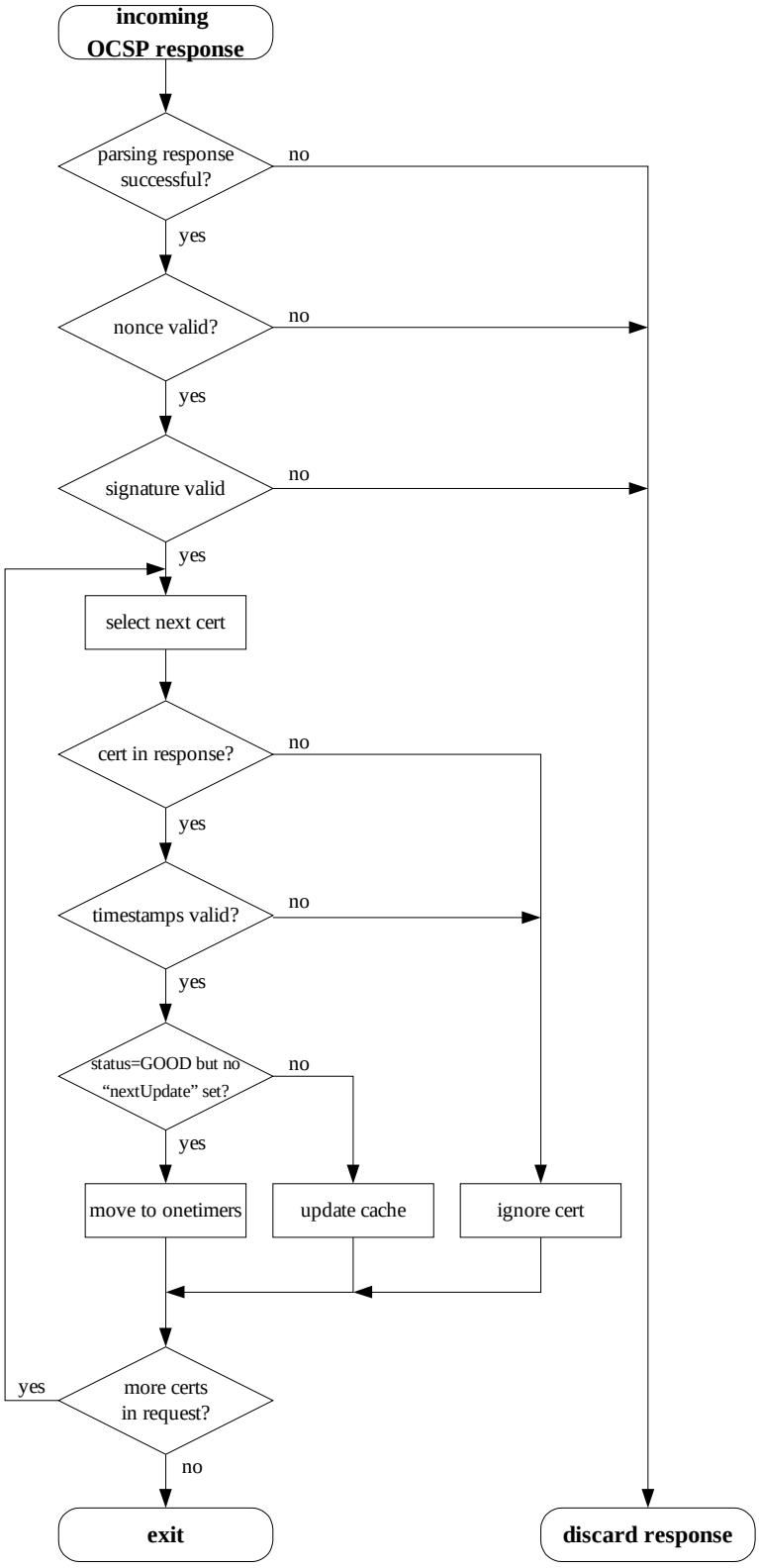


Abbildung 20 Flussdiagramm Eingang OCSP Response

Die Meldung vom OCSP Server kommt als ASN.1 Objekt an, wird als erstes geparst und in einer C Datenstruktur abgelegt. Wenn die Meldung einer gültigen ASN.1 OCSP Response entspricht, wird die Nonce mit der des Requests verglichen. Falls beide Nonces identisch sind und die Signatur dem erwarteten OCSP Signer entspricht, wird durch die Liste der Zertifikate iteriert. Für jedes Zertifikat wird überprüft ob:

- Das Zertifikat auch im Request angefordert wurde.
- Die Zeitstempel innerhalb des gültigen Bereichs liegen (z.B. nicht in der Zukunft).

Falls eine der Bedingungen nicht erfüllt wurde, wird das entsprechende Zertifikat einfach ignoriert.

Wenn ein nextUpdate Feld vorhanden ist wird der Status des Zertifikats in den Main-Cache eingetragen. Wird ein Zertifikat als gültig deklariert, aber das nextUpdate Feld fehlt, wird das Zertifikat dem One-Timer-Cache hinzugefügt.

Da Pluto einen eigenen Cache für Public Keys verwaltet, musste auch dieser manipuliert werden. So wird die Gültigkeitsdauer von überprüften Zertifikaten auch im entsprechenden Public Key aus dem Cache aufdatiert. Falls ein Zertifikat gar gesperrt wurde, wird der zugehörige Public Key gelöscht.

5 Realisierung

Für die OCSP-spezifischen Funktionen und Daten wird ein neues Modul mit dem Namen *ocsp* eingeführt. Dieses Modul enthält im Wesentlichen die OCSP Caches und Funktionen zum Generieren und Parsen von OCSP Meldungen.

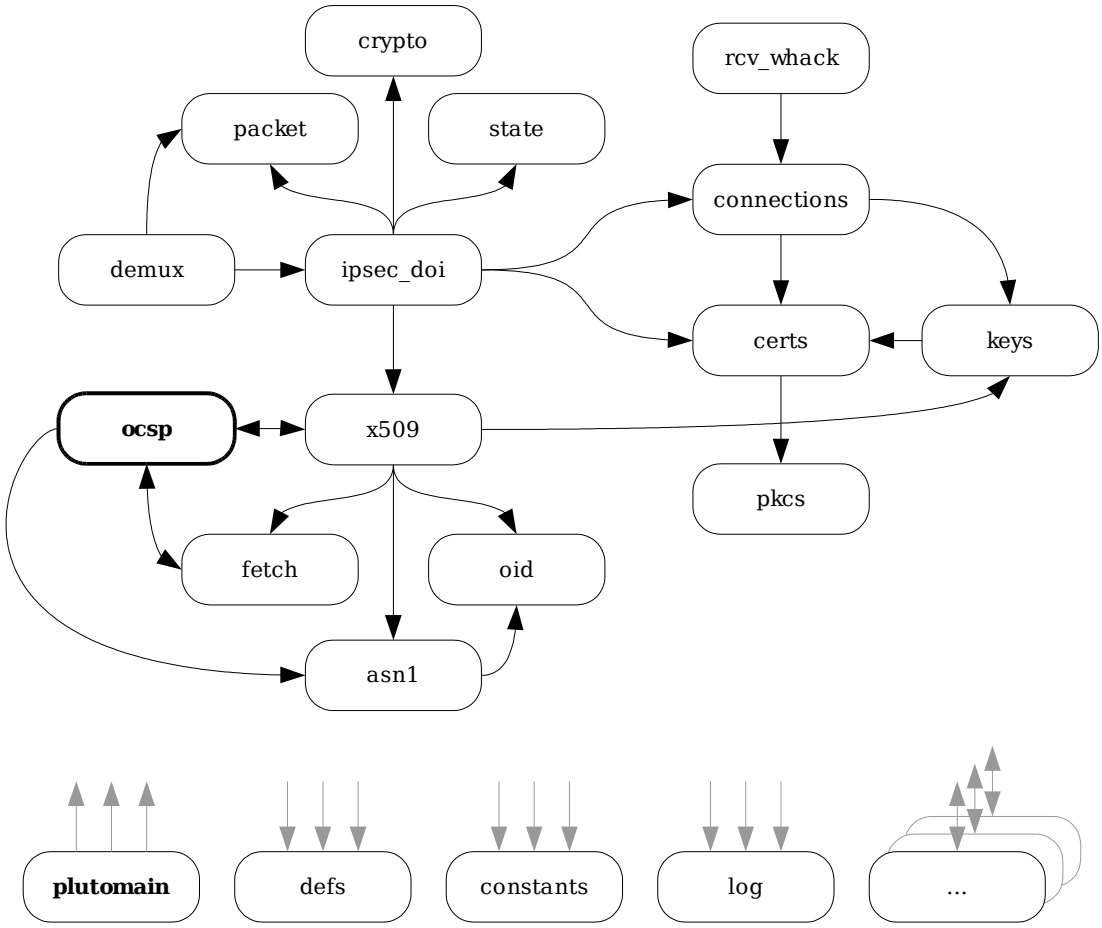


Abbildung 21 Aufbau Pluto mit OCSP Modul

Abbildung 21 zeigt das neue Modul *ocsp* mit den wichtigsten Zusammenhängen. Die OCSP Funktionalität wird im *x509*-Modul hinzugefügt, während das *ocsp*-Modul wiederum auf Funktionen (Zertifikate parsen, ...) im *x509*-Modul zugreift. Für das Parsen der OCSP Response wird auf den ASN.1-Parser im *asn1*-Modul zugegriffen.

Das *fetch*-Modul wird so angepasst, dass OCSP Anfragen ausgeführt werden können und der Cache automatisch aktualisiert wird. Für die Synchronisierung der OCSP Caches werden zwei neue Mutex im *fetch*-Modul eingefügt.

5.1 Neues Modul OCSP

Das Modul enthält alle OCSP-spezifischen Funktionen und Daten. Dazu gehören die OCSP Caches und Funktionen zum Generieren und Parsen von OCSP Meldungen.

5.1.1 Übersicht Funktionen

Das eingeführte Modul enthält eine Vielzahl von Funktionen. Zur besseren Übersicht wurden die Funktionen nach Themengebiet gruppiert.

Globale Funktionen tragen gemäss den FreeS/WAN-Konventionen den Prefix 'ocsp_'.

Modul-Konfiguration	
ocsp_set_request_cert	Setzt das Zertifikat, welches zum Signieren von Requests verwendet werden sollen.
ocsp_set_response_cert	Setzt das Zertifikat des OCSP Responders.
ocsp_set_default_uri	Setzt die Default URI für Zertifikate, die keine AccessLocationURI enthalten.
ocsp_has_default_uri	Gibt an, ob eine Default URI konfiguriert wurde.

OCSP Caches Hilfsfunktionen	
cert_access_location	Gibt die OCSP URI für ein Zertifikat zurück.
same_cert	Überprüft die Übereinstimmung einer Zertifikats-ID (certinfo) mit einem Zertifikat und dem CA-Zertifikat.
cacert_hashes	Erstellt den Hash über den Common Name (CN) der CA, falls das CA-Zertifikat diesen noch nicht enthält.

OCSP One-Timer-Cache	
add_onetimer	Fügt ein Zertifikat in die Liste der One-Timer ein.
free_onetimer	Löscht einen One-Timer aus der Liste und gibt den Speicher frei.
get_valid_onetimer	Sucht ein Zertifikat in der Liste der One-Timer und gibt den Status entsprechend dem Suchergebnis zurück.
ocsp_clean_onetimers	Löscht veraltete One-Timer aus der Liste.

OCSPCache	
cache_add_location	Fügt eine URI in die Liste der Locations ein.
cache_getlocation	Gibt die zur URI passende Location zurück.
ocsp_getcache	Gibt den Pointer auf den Beginn der Liste zurück.
ocsp_cache_add_cert	Fügt ein Zertifikat in den OCSP Cache ein und markiert die Location zum Download. Überprüft, ob das Zertifikat nicht bereits in der Liste vorhanden ist.
ocsp_cache_get_certstatus	Gibt den OCSP Status für ein Zertifikat zurück, überpüft auch den One-Timer Cache auf einen Eintrag.
ocsp_cache_get_nextupdate	Gibt die NextUpdate-Zeit für ein Zertifikat zurück.
free_allcertinfos	Gibt alle Certinfos einer verketteten Liste frei.
free_certinfo	Entfernt eine Certinfo aus einer Liste und gibt den Speicher frei.
free_location	Entfernt eine Location aus der Liste und gibt den Speicher frei.
free_cache	Gibt beide Caches frei.
ocsp_purge_cache	Gibt beide Caches frei, mit Synchronisation.
ocsp_free_all	Gibt beide Cache und alle globalen Variablen frei.
ocsp_list_cache	Gibt den Inhalt der Caches aus.

Generieren von Requests, Hilfsfunktionen	
header_len	Berechnet die Länge des Headers (Object ID und Length) für ein ASN.1 Objekt mit bekannter Datenlänge.
add_header	Schreibt den Header eines ASN.1 Objektes.
allocate_chunk	Alloziert Speicher für einen Chunk mit bekannter Länge.
mv_chunk	Verschiebt den Inhalt eines Chunks an eine bestimmte Stelle im Speicher.

Generieren von Requests	
ocsp_getrequest	Generiert einen OCSP Request für eine Location.
get_tbs_request	Generiert einen TBSRequest.
get_requestor_name	Generiert den RequestorName, falls ein Request Zertifikat konfiguriert wurde.
get_request_list	Generiert eine RequestList.
get_request	Generiert einen Request für eine Certinfo.
get_req_cert	Generiert ein ReqCert (CertID) für eine Certinfo.
get_request_ext	Generiert die RequestExtensions.
get_nonce_extension	Generiert eine Nonce Extension.
get_accept_extension	Generiert eine AcceptableResponseTypes Extension.
get_signature	Generiert die Signatur, falls ein RequestCert konfiguriert wurde und ein passender Private Key vorhanden ist.

Parsen von Responses	
signature_valid	Überpüft eine Response auf eine gültige und authentisierte Signatur.
timestamps_valid	Überprüft eine Response auf gültige Zeitwerte.
parse_response	Parst eine OCSPResponse.
parse_basic_response	Parst eine BasicOCSPResponse.
free_response	Gibt die Datenstrukturen frei.
ocsp_parseresponse	Parst eine OCSP Response, überprüft die Angaben und aktualisiert den Cache.

Die Funktionen für das Generieren und Parsen von Meldungen werden in den Kapiteln 5.4 ASN.1 Generierung und 5.5 ASN.1 Parsing genauer erläutert.

5.2 Änderungen Modul Fetch

Das Fetch-Modul enthielt bisher den Fetch-Thread mit den Funktionen für das CRL-Fetching. Da auch die OCSP Funktionalität im Fetch-Thread ablaufen soll, werden am Fetch-Modul einige Änderungen und Erweiterung vorgenommen.

Bisher wurde CURL als eigenständiges Programm aufgerufen (popen). Gemäss Auftrag wurde stattdessen *libcurl* eingebunden und der entsprechende Code abgeändert.

5.2.1 Synchronisation

Für die Synchronisation der Threads wurden zwei neue Mutex eingeführt, für den OCSP Cache und für die One-Timer.

Entsprechend wurden neue Funktionen eingeführt, um diese Mutex zu setzen und freizugeben.

Mutex-Funktionen	
lock_ocsp_cache	Sperrt den OCSP Cache für den anderen Thread. Blockiert wenn der Cache bereits blockiert wurde.
unlock_ocsp_cache	Gibt den OCSP Cache wieder frei.
lock_ocsp_onetimer	Sperrt die OCSP One-Timer für den anderen Thread. Blockiert wenn der Cache bereits blockiert wurde.
unlock_ocsp_onetimer	Gibt die OCSP One-Timer wieder frei.

5.2.2 Neue Funktionen

OCSP Fetch-Funktionen	
fetch_ocsp	Aktualisiert den kompletten OCSP Cache.
fetch_ocspstatus	Aktualisiert einen OCSP Cache Eintrag.
write_buffer	Hilfsfunktion für die Verwendung von *libcurl*.

5.3 Änderungen Modul X.509

Im X.509-Modul wurde der OCSP Mechanismus in die Überprüfung der Zertifikate (Funktion *verify_x509cert*) eingebunden.

Das Parsing der Zertifikate wurde erweitert, so dass OCSP-spezifische Daten ebenfalls ausgewertet werden: die AuthInfoAcess Extension und die ExtendedKeyUsage OCSPSigner Extension.

5.4 ASN.1 Generierung

Wie in den Grundlagen gezeigt, ist die DER-Codierung von ASN.1 hierarchisch aufgebaut und verlangt die endliche Längenangabe. Beim Generieren von Meldungen stellt sich dadurch das Problem, dass die gesamte Meldung von innen heraus aufgebaut werden muss, da jeweils die Länge des unterliegenden Objektes bekannt sein muss.

Da für das Generieren von ASN.1-Meldungen in FreeS/WAN keine Hilfsfunktionen wie für das Parsen vorhanden waren, wurde eine einfache hierarchische Struktur von Funktionen gewählt, um die OCSP Requests zu erstellen.

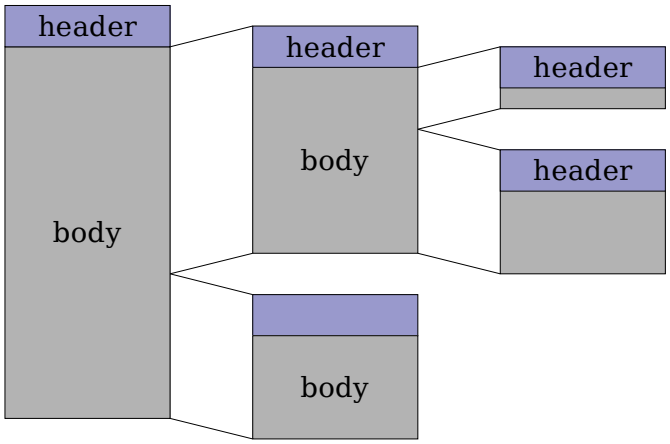


Abbildung 22 Aufbau ASN.1 Strukturen

Abbildung 22 zeigt den hierarchischen Aufbau von ASN.1 mit der Verschachtelung der Elemente. Für das Schreiben der Header ist es nötig, die gesamte Länge des Inhaltes zu kennen, ausserdem ist die Länge des Headers wiederum von der Länge des Inhalts abhängig.

Genau diese Struktur haben wir auch für unsere ASN.1 Generierung gewählt: Die kleinsten Elemente mit bekannter Grösse werden erstellt, danach Speicher entsprechend der Grösse des neuen Elementes alloziert und die kleineren Elemente in das gesamte Objekte verschoben.

5.4.1 Code-Beispiel für einen TBSRequest

Anhand des Programmcodes für die Generierung des TBSRequest-Objektes soll das Funktionsprinzip erläutert werden.

Die Unterelemente des TBSRequest werden als Chunks, also Binäre Strings, erstellt.

```
requestorName = get_requestor_name();
requestList = get_request_list();
requestExt = get_request_ext();
```

Die Datenlänge entspricht der Länge der Unterelemente. Aus der Datenlänge kann die Headerlänge bestimmt werden.

```
datalen = requestList.len + requestorName.len + requestExt.len;
headerlen = header_len(datalen);
```

Es wird Speicher für das neue Objekt angelegt, entsprechend der Datenlänge und der Headerlänge.

```
location = allocate_chunk(&tbsRequest, headerlen + datalen);
```

Der Header (ASN.1 Sequence) für das neue Objekt wird geschrieben.

```
add_header(&location, ASN1_SEQUENCE, headerlen, datalen);
```

Und schliesslich werden die Unterelemente in den neu allozierten Speicher verschoben.

```
mv_chunk(&location, requestorName);
mv_chunk(&location, requestList);
mv_chunk(&location, requestExt);
```

Das Resultat (wieder ein Unterelement eines ASN.1 Objektes) wird zurückgegeben und analog verarbeitet.

```
return tbsRequest;
```

5.5 ASN.1 Parsing

Um die ASN.1 codierten Meldungen in eine C Datenstruktur zu zerlegen, benötigen wir einen ASN.1 Parser. Da X.509 Zertifikate auch ASN.1 codiert sind, existiert bereits ein Parser im Modul *asn1*. Dieser muss nur minimal erweitert werden, um unsere Meldungen zu parsen.

5.5.1 Funktionsweise

Der Parser besteht aus zwei Funktionen:

- **asn1_init**

  Initialisiert den Parser und übergibt ihm den zu parsenden *blob* (block of bytes). Erzeugt einen ASN.1 Context, der beim Parsen laufend übergeben werden muss. Diese Funktion hat keinen Rückgabewert.

- **extract_object**

  Parst das nächste ASN.1-Objekt aus dem *blob*. Falls das gesuchte Objekt gefunden wurde, wird TRUE zurückgegeben, sonst FALSE. Der Parameter *objectID* enthält einen Pointer auf den Index des *asn1Object*-Arrays. Der Parameter *object* enthält einen Pointer auf das geparste Objekt.

Um dem Parser mitzuteilen wie die ASN.1 Meldung aufgebaut ist, muss erst ein asn1Object-Array erstellt werden. Ein asn1Object ist durch folgende Felder definiert:

```
/* definition of an ASN.1 object */
typedef struct {
    u_int   level;                  # Hierarchie Ebene
    const u_char  *name;            # Name des Objekts
    asn1_t  type;                   # Typ des ASN.1 Objekts
    u_char  flags;                  # Flags um den Parser zu kontrollieren
} asn1Object_t;
```

Gültige ASN.1 Objekttypen sind:

Typ	*Beschreibung*
ASN1_EOC	Platzhalter, bewirkt nichts (z.B. Ende einer Schlaufe)
ASN1_BOOLEAN	ASN.1 Typ: BOOLEAN
ASN1_INTEGER	ASN.1 Typ: INTEGER
ASN1_BIT_STRING	ASN.1 Typ: BIT STRING
ASN1_OCTET_STRING	ASN.1 Typ: OCTET STRING
ASN1_NULL	ASN.1 Typ: NULL
ASN1_OID	ASN.1 Typ: Object Identifier
ASN1_ENUMERATED	ASN.1 Typ: ENUMERATED
ASN1_UTF8STRING	ASN.1 Typ: UTF8STRING
ASN1_NUMERICSTRING	ASN.1 Typ: NUMERICSTRING
ASN1_PRINTABLESTRING	ASN.1 Typ: PRINTABLESTRING
ASN1_T61STRING	ASN.1 Typ: T61STRING
ASN1_VIDEOTEXSTRING	ASN.1 Typ: VIDEOTEXSTRING
ASN1_IA5STRING	ASN.1 Typ: IA5STRING
ASN1_UTCTIME	ASN.1 Typ: UTCTIME
ASN1_GENERALIZEDTIME	ASN.1 Typ: GENERALIZEDTIME
ASN1_GRAPHICSTRING	ASN.1 Typ: GRAPHICSTRING
ASN1_VISIBLESTRING	ASN.1 Typ: VISIBLESTRING
ASN1_GENERALSTRING	ASN.1 Typ: GENERALSTRING
ASN1_UNIVERSALSTRING	ASN.1 Typ: UNIVERSALSTRING
ASN1_BMPSTRING	ASN.1 Typ: BMPSTRING
ASN1_CONSTRUCTED	ASN.1 Typ: CONSTRUCTED
ASN1_SEQUENCE	ASN.1 Typ: SEQUENCE

Typ	*Beschreibung*
ASN1_SET	ASN.1 Typ: SET
ASN1_CONTEXT_S_*	Context-Objekt für "simple" ASN.1 Objekte mit EXPLICIT Tag
ASN1_CONTEXT_C_*	Context-Objekt für "constructed" ASN.1 Objekte mit EXPLICIT Tag

Mögliche Flags um den Parser zu kontrollieren:

Typ	*Funktion*
ASN1_NONE	Der Parser gibt nichts zurück
ASN1_DEF	Entspricht dem ASN.1 Default Tag
ASN1_OPT	Objekt ist optional
ASN1_LOOP	Beginn einer Schlaufe
ASN1_END	Ende einer Schlaufe
ASN1_OBJ	Parser gibt das ganze ASN.1 Objekt zurück
ASN1_BODY	Parser gibt nur den Inhalt des Objekts zurück
ASN1_RAW	Parser gibt nächstes Objekt ohne Typenprüfung zurück

Gegeben sei nun folgende ASN.1 Syntax:

```
BasicOCSPResponse       ::= SEQUENCE {
   tbsResponseData      ResponseData,
   ...............      ............,
}
ResponseData ::= SEQUENCE {
   version              [0] EXPLICIT Version DEFAULT v1,
   responderID              ResponderID,
   ...........              ...........,
}
ResponderID ::= CHOICE {
   byName   [1] Name,
   byKey    [2] KeyHash
}
```

Um diese mit unserem Parser zu verarbeiten, wird folgendes Objekt erstellt und bei jedem Aufruf der Methode *extract_object(...)* übergeben:

```
static const asn1Object_t basic_ocsp_response_object[] = {
  { 0, "BasicOCSPResponse",             ASN1_SEQUENCE,          ASN1_BODY },            /*  0 */
  { 1,   "tbsResponseData",             ASN1_SEQUENCE,          ASN1_OBJ  },            /*  1 */
  { 2,     "versionContext",            ASN1_CONTEXT_C_0,       ASN1_BODY | ASN1_DEF  },/*  2 */
  { 3,       "version",                 ASN1_INTEGER,           ASN1_BODY },            /*  3 */
  { 2,     "responderIdContext",        ASN1_CONTEXT_C_1,       ASN1_OPT  },            /*  4 */
  { 3,       "responderIdByName",       ASN1_SEQUENCE,          ASN1_OBJ  },            /*  5 */
  { 2,     "end choice",                ASN1_EOC,               ASN1_END  },            /*  6 */
  { 2,     "responderIdContext",        ASN1_CONTEXT_C_2,       ASN1_OPT  },            /*  7 */
  { 3,       "responderIdByKey",        ASN1_OCTET_STRING,      ASN1_BODY },            /*  8 */
  { 2,     "end choice",                ASN1_EOC,               ASN1_END  },            /*  9 */
  { .,       ".................",       ..............,         ........  },            /*  . */
};
```

Falls die Methode TRUE zurückgibt, wurde das nächste Objekt erfolgreich geparst und kann weiterverarbeitet werden.

5.6 Konfigurationsoptionen

Da unsere Implementation einige Konfigurationsoptionen erfordert, waren Änderungen an den Scripts notwendig, welche Pluto starten und konfigurieren. Da die meisten Scripts nur Konfigurationsparameter weitergeben, waren die Änderungen jedoch minimal.

Die geänderten Scripts sind in Abbildung 23 schattiert dargestellt:

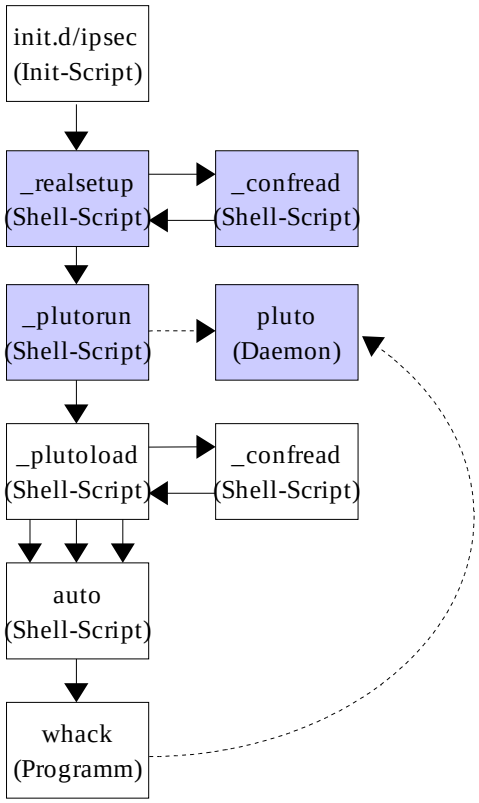


Abbildung 23 Änderungen Start-Scripts Pluto

- Das Init-Script **ipsec** startet das Script _realsetup.
- Das Script **_realsetup** ruft _confread auf.
- Das Script **_confread** parst die Konfigurationsdatei */etc/ipsec.conf* und setzt die Optionen als Shell-Variablen.
- Das Script _realsetup startet danach **_plutorun** , welches die Shell-Variablen auf ihre Werte überprüft und in Kommandozeilenparameter umwandelt. Pluto kann nun mit den vorbereiteten Parametern aufgerufen werden.
- Im Daemon **pluto** , in *plutomain.c* werden die Parameter geparst und im Modul *ocsp* in globalen Variablen abgelegt.

6 Test

Die erstellte Software wurde mit verschiedenen Tests verifiziert und geprüft, um die möglichen Anwendungs- und Fehlerfälle zu überprüfen.

6.1 Anwendungstests

6.1.1 Anwendungsfälle

Für die Anwendungstests definierten wir die wichtigsten Anwendungsfälle, bei denen die Software eine bestimmte Verhaltensweise zeigen sollte. Diese Anwendungsfälle wurden aus dem Pflichtenheft abgeleitet.

Requests

- Ist die AuthorityInfo Acess URI im Zertifikat angegeben, soll (sofern nicht im Cache) ein Request gestartet werden.
- Ist eine URI lokal konfiguriert, sollen für Zertifikate, die keine AuthorityInfoAccess-URI enthalten, diese URI statt CRL verwendet werden.
- Der Request soll alle Zertikate beinhalten, die für diese Location (URI) bekannt sind.
- Ist ein OCSP Request Signer Zertifikat konfiguriert, sollen alle OCSP-Requests signiert und das entsprechende Zertifikat angehängt werden.
- Die Signatur des Request muss gültig sein.

Allgemein

- Der OCSP-Cache soll aktualisiert werden.
- Das CRL-Fetching soll getestet werden (Umstellung auf *libcurl*).

Responses

- Eine Basic-Response soll ausgewertet werden können.
- Fehlermeldungen werden ignoriert und ein entsprechendes Log ausgegeben.
- Die Response soll nur akzeptiert werden, falls alle der folgenden Bedingungen erfüllt sind:
  - Die Signatur wurde von einer CA (aus der lokalen CA-Liste), einem OCSP-Signer (gemäss angehängtem Zertifikat) oder dem Trusted Responder (gemäss Konfiguration) erstellt.
  - Der OCSP-Signer entspricht dem Responder (Name).
  - Die Angabe thisUpdate ist nicht älter als 1 Minute.

  - Falls die Nonce in der Antwort enthalten ist, muss sie dem Nonce aus dem Request entsprechen.
  - Falls Nonce in der Antwort nicht enthalten ist, soll eine Warnung ausgegeben werden.
  - Die Signatur ist gültig.

- OCSP-Server mit nextUpdate:
  - Gültige Antworten sollen in einem Cache abgelegt werden.
  - Falls die StrictCRLPolicy gesetzt ist, soll das Ablaufdatum des Public Key auf das NextUpdate-Datum gesetzt werden.
  - Die Angabe nextUpdate muss in der Zukunft liegen.
  - Falls Zertifikat REVOKED, wird der Public Key gelöscht.

- OCSP-Server ohne nextUpdate:
  - Falls der Status GOOD ist, soll diese Angabe genau einmal verwendet werden, dies nur innerhalb von 2 Minuten.
  - Falls der Status REVOKED oder UNKOWN ist, soll die Antwort im Cache abgelegt werden, wo sie mindestens alle 24h aktualisiert werden soll.
  - Falls die StrictCRLPolicy gesetzt ist, soll die Gültigkeit des Public Key auf zwei Minuten gesetzt werden.
  - Falls Zertifikat REVOKED, wird der Public Key gelöscht.

Nicht testbare Funktionen

Mit dem zur Verfügung stehenden OpenSSL OCSP-Server war es nicht möglich, alle Funktionen unseres Clients zu testen.

- Mehrere Signatur- & Hashalgorithmen können verarbeitet werden.
- Der OCSP Server sendet eine falsche Nonce (Replay-Attacke).
- Der OCSP Server sendet keine Nonce (ermöglicht Replay-Attacke).
- Die Signatur des Requests ist gültig.

6.1.2 Testfälle

Um die definierten Anwendungsfälle zu testen, wurden Testfälle mit verschiedenen Konfigurationen von Roadwarrior, Gateway und OCSP-Server definiert.

		OCSP-Server			FSW-Client	FSW-Gateway
Nr	**Zu testen**	**Server-Zertifikat**	**Gültig (Min.)**	**Optionen**	**Client-Zertifikat**	**Optionen**
1	Herkömmliches CRL-Fetching	(nicht genutzt)			client-crl	
2	OCSP Standard-Anwendung, Roadwarrior ist GOOD	server-ca	8		client-good-uri	
3	Roadwarrior ist REVOKED	server-ca	8		client-revoked-uri	
4	Roadwarrior ist UNKNOWN	server-ca	8		client-unkown-uri	
5	Kein NextUpdate, GOOD	server-ca			client-good-uri	
6	Kein NextUpdate, REVOKED	server-ca			client-revoked-uri	
7	OCSP-Server: OCSPSigner – Zertifikat	server-ocspsigner	8		client-good-uri	
8	OCSP-Server: Unbekanntes Zertifikat	server-trusted	8		client-good-uri	
9	OCSP-Server: Konfiguriertes Zertifikat	server-trusted	8		client-good-uri	ocsprescert= server-trusted
10	Keine URI in Zertifikat	server-ca	8		client-good-nouri	ocspuri= dskt6817
11	Request-Signatur	server-ca	8		client-good-uri	ocspreqcert= gateway-cert
12	OCSP-Server liefert KeyID statt NameID	server-ca	8	resp_key_id	client-good-uri	
13	OCSP-Server hängt kein Zertifikat an, signiert als CA	server-ca	8	resp_no_certs	client-good-uri	
14	OCSP-Server hängt kein Zertifikat an, signiert als OCSPSigner	server-ocspsigner	8	resp_no_certs	client-good-uri	
15	Fehlermeldungen	(ocsp.openvalidation.org genutzt)			client-good-nouri	ocspuri= ov.org:8085-8089
16	Gateway Systemzeit falsch	server-ca	8		client-good-uri	Zeit-Offset= -1h 20 min
17	Gateway Systemzeit falsch	server-ca	8		client-good-uri	Zeit-Offset= +9 Tage
18	Gateway Systemzeit falsch	server-ca	1		client-good-uri	Zeit-Offset= +2 Minuten
19	Strict CRL Policy, Roadwarrior good	server-ca	8		client-good-uri	strictcrlpolicy=yes
20	Strict CRL Policy, Roadwarrior revoked	server-ca	8		client-revoked-uri	strictcrlpolicy=yes
21	Strict CRL Policy, Roadwarrior unknown	server-ca	8		client-unkown-uri	strictcrlpolicy=yes
22	Strict CRL Policy, kein NextUpdate, Roadwarrior good	server-ca			client-good-uri	strictcrlpolicy=yes

Die Tabelle zeigt die Testfälle mit den jeweiligen Konfigurationen. Die NextUpdate-Zeit des OCSP-Server wurde sehr klein gewählt, um den Test effizient durchführen zu können (Update des Cache testen etc.).

6.1.3 Benötigte Zertifikate

Um die wichtigsten Anwendungsfälle testen zu können, wurden eine Reihe von Zertifikaten erstellt. Alle Zertifikate wurden direkt von der RootCA ('daCA') ausgestellt.

FreeS/WAN-Gateway:

Zertifikatsname	*Beschreibung*
gateway-cert	Standard-Zertifikat

FreeS/WAN-Roadwarrior:

Zertifikatsname	*Beschreibung*
client-crl	Zertifikat ohne OCSP AccessLocation URI, mit CRLDistributionPoint
client-good-nouri	Zertifikat ohne OCSP AccessLocation URI
client-good-uri	Zertifikat mit OCSP AccessLocation URI, Status GOOD
client-revoked-uri	Zertifikat mit OCSP AccessLocation URI, Status REVOKED
client-unkown-uri	Zertifikat mit OCSP AccessLocation URI, Status UNKOWN (wurde aus CRL gelöscht)

OCSP-Server:

Zertifikatsname	*Beschreibung*
server-ca	RootCA-Zertifikat (OCSP-Response wird direkt von der CA geliefert)
server-ocspsigner	Zertifikat mit der ExtendedKeyUsage-Extension 'OCSPSigner'
server-trusted	Standard-Zertifikat

6.1.4 Testaufbau

Als Testaufbau wurde eine einfache Infrastruktur gewählt, bestehend aus einem FreeS/WAN Gateway, einem OCSP Server und einem Roadwarrior. Mit dieser Anordnung konnte die OCSP Funktionalität getestet werden.

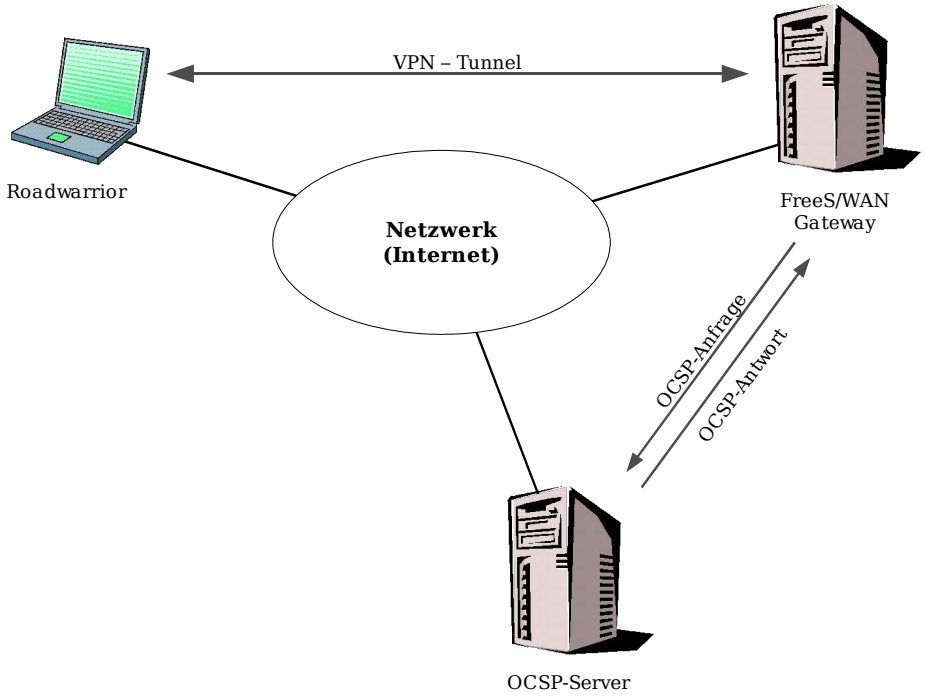


Abbildung 24 Testaufbau

6.1.5 Testablauf

Um den Test möglichst effizient zu gestalten, wurden alle Testfälle automatisiert und von mehreren Shell-Scripts gesteuert. Die Steuerung der Tests übernahm der Roadwarrior, der über SSH mit dem Gateway und dem OCSP-Server kommunizierte.

Während des Tests wurden die Log-Files aller beteiligten Rechner aufgezeichnet.

Ablauf für einen Testfall:

1. Test wird auf dem Roadwarrior gestartet.
2. Der OCSP-Server wird mit der gewünschten Konfiguration gestartet.
3. Der FreeS/WAN-Server wird mit der gewünschten Konfiguration gestartet.
4. Der Roadwarrior (FreeS/WAN Client) wird mit der gewünschten Konfiguration gestartet und versucht die VPN Verbindung aufzubauen.

5. Nach einer Minute wird der Status des FreeS/WAN Servers geloggt.
6. Nach zehn Minuten wird erneut der Status des FreeS/WAN Servers geloggt.
7. Die VPN-Verbindung wird durch den Roadwarrior gestoppt und neu gestartet. Der Verbindungsversuch wird aufgezeichnet.
8. Nach einer Minute wird der Status des FreeS/WAN Servers geloggt.
9. Nach zehn Minuten wird erneut der Status des FreeS/WAN Servers geloggt.
10. Roadwarrior, OCSP-Server und Gateway werden (in dieser Reihenfolge) gestoppt.
11. Die Log-Dateien des OCSP-Server und des Gateway werden zum Roadwarrior transferiert.

6.1.6 Scripts zur Teststeuerung

Die Tests können mit den auf CD beigelegten Shell-Scripts nachvollzogen werden. Die Host- und Benutzernamen sind einfach anpassbar. SSH muss für den Test so eingerichtet werden, dass für den Roadwarrior der Login auf dem OCSP Server und dem FreeS/WAN Gateway ohne Passwort erfolgen kann.

Script-Name	*Zweck*	*Beispiel*
automatic_test	Startet den kompletten Testablauf mit allen Tests. Sollen nur ausgewählte Tests gestartet werden, können diese auf der Kommandozeile angegeben werden.	./automatic_test (Führt alle 22 Tests durch) ./automatic_test 14 16 (Führt die Tests 14 und 16 durch)
starttest	Startet einen bestimmten Test. Der OCSP Server wird gestartet, danach der Gateway und schliesslich der Roadwarrior.	./starttest 3 (Startet den Test 3)
stoptest	Stoppt einen bestimmten Test (die Server und den Roadwarrior) und bezieht die Log-Dateien der Server.	./stoptest 3 (Stoppt den Test 3)
ocsp_server_test	Startet den OCSP Server mit der richtigen Konfiguration für den jeweiligen Test.	./ocsp_server_test 3 (Startet den OCSP Server für den Test 3)

Script-Name	*Zweck*	*Beispiel*
gateway_test	Startet den FreeS/WAN Gateway mit der richtigen Konfiguration für den jeweiligen Test.	./gateway_test 3 (Startet den FreeS/WAN Gateway für den Test 3)
roadwarrior_test	Startet den FreeS/WAN Roadwarrior mit der richtigen Konfiguration für den jeweiligen Test.	./roadwarrior_test 3 (Startet den FreeS/WAN Roadwarrior für den Test 3)
stopfreeswan	Stoppt den FreeS/WAN Server oder Roadwarrior.	./stopfreeswan
stopocsp	Stoppt den OCSP Server	./stopocsp
complete_log	Erstellt Log-Dateien (Status) für den Gateway.	./complete_log 03 01 (Loggt für den Test 3 den ersten Status)
config	Enthält die Konfiguration der Scripts (Hosts, Benutzernamen)	

6.1.7 Testauswertung

Nr	*Zu testen*	*Erwartetes Resultat*	*Tatsächliches Resultat*
1	Herkömmliches CRL-Fetching	Keine OCSP-Anfrage, 'Fallback' CRL verwenden.	OK.
2	OCSP Standard-Anwendung, Roadwarrior ist GOOD	Beim Verbinden des Client wird eine OCSP-Anfrage gestartet, deren Antwort als GOOD in den Cache abgelegt werden soll. Der Cache soll vor Ablauf geupdatet werden. Bei erneuter Verbindung wird der Status aus dem Cache genommen.	OK.

Nr	*Zu testen*	*Erwartetes Resultat*	*Tatsächliches Resultat*
3	Roadwarrior ist REVOKED	Die Anfrage soll in den Cache gelegt werden. Public Key soll gelöscht werden. Verbindung nicht mehr möglich.	OK. Public Key wird erst bei Reconnect (Rekeying) gelöscht.
4	Roadwarrior ist UNKNOWN	Die Anfrage soll in den Cache gelegt werden. Verbindungsaufbau weiterhin möglich.	OK.
5	Kein NextUpdate, GOOD	Die Anfrage soll nicht in den Cache abgelegt werden.	OK. Anfrage wird im Kurzzeit-Cache abgelegt und wird nach 2 Minuten gelöscht.
6	Kein NextUpdate, REVOKED	Die Anfrage soll in den Cache gelegt werden, mit NextUpdate von 24h; Public Key soll gelöscht werden. Verbindungsaufbau nicht mehr möglich.	OK. Public Key wird erst bei Reconnect (Rekeying) gelöscht.
7	OCSP-Server: OCSPSigner – Zertifikat	Anfrage soll akzeptiert werden.	OK.
8	OCSP-Server: Unbekanntes Zertifikat	Die Antwort soll ignoriert werden, Fehlermeldung.	OK. Fehlermeldung: “OCSP response: no matching OCSP-signer certificate found”.
9	OCSP-Server: Konfiguriertes Zertifikat	Anfrage soll akzeptiert werden.	OK.
10	Keine URI in Zertifikat	Anfrage soll gestartet werden.	OK.
11	Request-Signatur	Die Anfrage soll signiert werden, der RequestorName soll gesetzt werden, das entsprechende Zertifikat soll angehängt werden.	OK.

Nr	*Zu testen*	*Erwartetes Resultat*	*Tatsächliches Resultat*
12	OCSP-Server liefert KeyID statt NameID	Die Antwort soll akzeptiert werden.	OK.
13	OCSP-Server hängt kein Zertifikat an, signiert als CA	Die Antwort soll akzeptiert werden.	OK.
14	OCSP-Server hängt kein Zertifikat an, signiert als OCSPSigner	Die Antwort soll nicht akzeptiert werden; eine entsprechende Warnung soll ausgegeben werden.	OK. Fehlermeldung: "OCSP Response: no matching OCSP-signer certificate found".
15	Fehlermeldungen	Die Antwort soll ignoriert werden, eine Fehlermeldung soll geloggt werden.	OK. Fehlermeldung: "OCSP response: server said "internal error"".
16	Gateway Systemzeit falsch	Die Antwort soll verworfen werden, mit Warnung (thisUpdate ist in der Zukunft).	OK. Fehlermeldung: "OCSP response: this_update is in the future".
17	Gateway Systemzeit falsch	Die Antwort soll verworfen werden, mit Warnung (thisUpdate ist zu alt).	OK. Fehlermeldung: "OCSP response: this_update is too old".
18	Gateway Systemzeit falsch	Die Antwort soll verworfen werden, mit Warnung (es exisitert ein neueres Update).	OK. Fehlermeldung: "OCSP response: a newer update exists".

Nr	*Zu testen*	*Erwartetes Resultat*	*Tatsächliches Resultat*
19	Strict CRL Policy, Roadwarrior GOOD	Beim ersten Verbindungsversuch soll die Verbindung abgelehnt werden (kein Status verfügbar), bei einem weiteren Versuch nur wenn der Status vorhanden ist. Das Ablaufdatum des Public Key soll auf das NextUpdate Feld gesetzt werden	OK. Ablaufdatum des Public Key wird erst bei einem Reconnect neu gesetzt, wird dazwischen möglicherweise als 'fatal (expired)' ausgegeben.
20	Strict CRL Policy, Roadwarrior REVOKED	Der Verbindungsaufbau soll verweigert werden, der Public Key soll gelöscht werden.	OK.
21	Strict CRL Policy, Roadwarrior UNKOWN	Der Verbindungsaufbau soll verweigert werden, der Public Key soll gelöscht werden.	OK.
22	Strict CRL Policy, kein NextUpdate, Roadwarrior GOOD	Beim ersten Verbindungsversuch soll die Verbindung abgelehnt werden (kein Status verfügbar), bei einem weiteren Versuch nur wenn der Status vorhanden ist. Der Status soll nicht in den Cache aufgenommen werden; das Ablaufdatum des Public Key soll auf +1min gesetzt werden.	OK.

6.2 Anwendungstests mit mehreren Clients

Um zu kontrollieren, ob die Software auch mit mehreren Zertifikaten und mehreren OCSP-Servern umgehen kann, wurde zwei weitere Tests mit einem erweiterten Testaufbau durchgeführt.

6.2.1 Testfälle

Nr	*Beschreibung*
1	Roadwarrior 1: OCSP Server 1, GOOD Roadwarrior 2: OCSP Server 1, REVOKED
2	Roadwarrior 1: OCSP Server 1, REVOKED Roadwarrior 2: OCSP Server 2, GOOD

Die OCSP-Server und der Gateway wurden gemäss Testfall 2 aus dem Haupttest konfiguriert.

6.2.2 Testaufbau

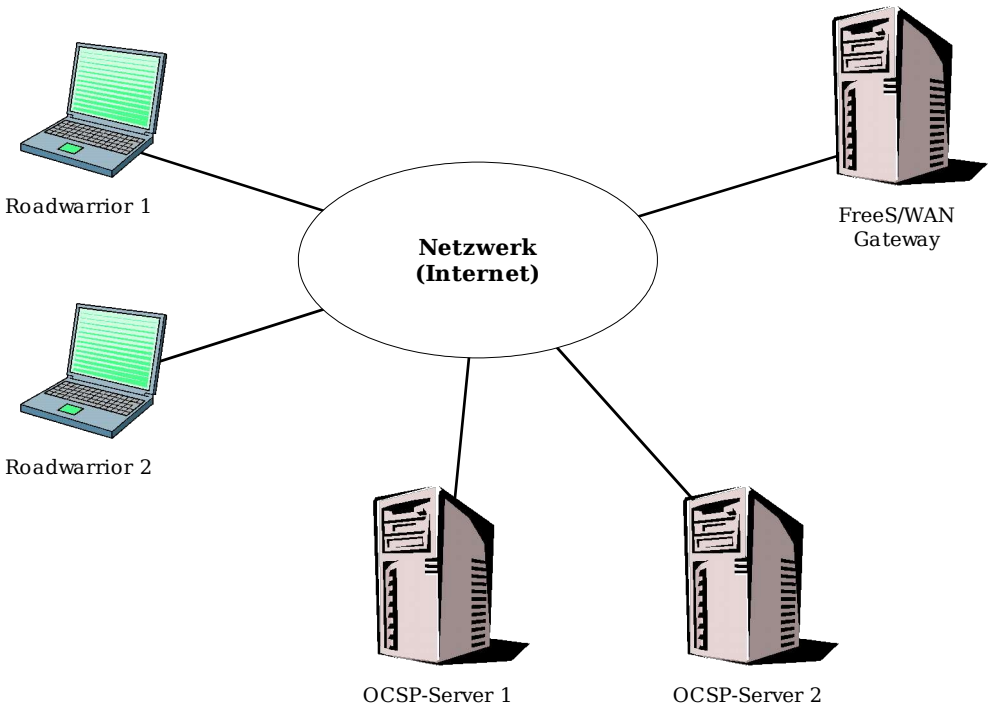


Abbildung 25 Testaufbau mit mehreren Clients

6.2.3 Testauswertung

Nr	*Zu testen*	*Erwartetes Resultat*	*Tatsächliches Resultat*
1	Roadwarrior 1: OCSP Server 1, GOOD Roadwarrior 2: OCSP Server 1, REVOKED	Beide Zertifikate werden in der selben Anfrage angefragt.	OK.
2	Roadwarrior 1: OCSP Server 1, REVOKED Roadwarrior 2: OCSP Server 2, GOOD	Zertifikat des Roadwarrior 1 wird bei OCSP Server 1 angefragt, das Zertifikat des Roadwarrior 2 beim OCSP Server 2. Getrenntes Updaten des Caches.	OK.

6.3 Fehlermeldungen

Bei diesem Test sollen die wahrscheinlichsten, in der Praxis auftretenden, Fehlerfälle simuliert werden. Damit soll geprüft werden, ob die Software im Fehlerfall die richtigen Fehlermeldungen ausgibt und somit bei der Fehlersuche Anhaltspunkte liefert.

Fehler	*Meldung*
FreeS/WAN wurde ohne *libcurl* kompiliert	OCSP error: pluto wasn't compiled with *libcurl* support
OCSP Server ist nicht erreichbar	Failed to fetch OCSP status: Connect failed
OCSP sendet ungültige Antwort (z.B. statt OCSP Server läuft Apache auf Port 80)	OCSP response: parse error
OCSP Server meldet ungültigen Request	OCSP response: server said "malformed OCSP request"
OCSP Server meldet internen Fehler	OCSP response: server said "internal error"
OCSP Server meldet temporäre Nichtverfügbarkeit	OCSP response: server said "try later"

Fehler	*Meldung*
OCSP Server meldet fehlende Authentisierung (Signatur nötig)	OCSP response: server said "signature required"
OCSP Server meldet ungültige Authentisierung	OCSP response: server said "unauthorized"
OCSP Server verwendet unbekanntes/ungültiges Zertifikat und ist kein OCSP Signer	OCSP response: no matching OCSP-signer certificate found
OCSP Server signiert als OCSP Signer, hängt aber das Zertifikat nicht an	OCSP response: no matching OCSP-signer certificate found
Signatur der Response ist ungültig/verfälscht	OCSP response: invalid signature
Zertifikat des OCSP Server (OCSP Signer) ist ungültig	OCSP response: invalid signature
FreeS/WAN Gateway und OCSP Server sind zeitlich nicht synchron	Mögliche Meldungen: OCSP response: this_update is in the future OCSP response: this_update is too old OCSP response: a newer update exists
OCSP Server sendet veraltete Response (bei serverseitigem Caching)	Mögliche Meldungen: OCSP response: a newer update exists OCSP response: this_update is too old
OCSP Server sendet den Nonce nicht zurück (ermöglicht Replay-Attacken)	OCSP warning: response contains no nonce, replay attack possible
OCSP Server sendet ungültigen Nonce zurück (Replay-Attacke)	OCSP response: invalid nonce
Konfiguriertes OCSP Request Signer Zertifikat: Datei kann nicht gelesen werden bzw. existiert nicht	warning: could not load OCSP request cert
Konfiguriertes OCSP Request Signer Zertifikat: Kein zugehöriger privater Schlüssel	OCSP warning: unable to sign request, no private key available

Fehler	*Meldung*
Konfiguriertes OCSP Response Signer Zertifikat: Datei kann nicht gelesen werden bzw. existiert nicht	warning: could not load OCSP response cert
Konfigurierte OCSP default AccessLocation URI: unbekanntes Protokoll (z.B. ldap)	warning: ignoring default OCSP uri with unkown protocol
Zertifikat enthält OCSP AccessLocation URI mit unbekanntem Protokoll (z.B. ldap)	warning: ignoring OCSP InfoAccessLocation with unkown protocol

7 Installation und Anwendung

Um unsere Arbeit einem breiteren Publikum zugänglich zu machen, wurde dieser Teil in Englisch verfasst.

7.1 Introduction

This document should help configuring FreeS/WAN in conjunction with an OCSP server.

7.1.1 Disclaimer

Use the information in this document at your own risk. We disavow any potential liability for the contents of this document. Use of the concepts, examples, and/or other content of this document is entirely at your own risk.

7.2 Requirements

The following packages are required to use an OCSP server with FreeS/WAN.

- CURL 7.10.5 (http://curl.haxx.se)
- FreeS/WAN 2.02 (http://www.freeswan.org)
- X.509 patch 1.4.7 (http://www.strongsec.com/freeswan)
- OCSP patch (should come with this documentation)

Of course you will also need an OCSP server, i.e.:

- OpenSSL 0.9.7c (http://www.openssl.org)

7.3 Installation

If you experience problems installing the following packages, it is possible that you do not met their requirements. Please refer to the *README* and *INSTALL* files in the appropriate tarballs.

7.3.1 CURL

CURL is used to retrieve messages from the OCSP server via HTTP.

```
tar xzf curl-7.10.5.tar.gz
cd curl-7.10.5
./configure
make
make test
make install
```

7.3.2 FreeS/WAN & X.509 patch

You should already have a working installation of FreeS/WAN on your system. If it isn't already installed please refer to the INSTALL file from the FreeS/WAN tarball.

Apply the X.509 patch, the OCSP patch and rebuild freeswan.

```
cd freeswan-x.xx
patch -p1 < ../x509patch-x.x.xx-freeswan-x.xx/freeswan.diff
patch -p1 < ../ocsppatch-x.x-freeswan-x.xx/ocsp.diff
make programs
make install
```

7.3.3 OpenSSL

As mentioned before, OpenSSL isn't needed for the OCSP patch. In our example, it is merely used as OCSP server for testing the patch.

```
tar xzf openssl-0.9.7c.tar.gz
cd openssl-0.9.7c
./config
make
make test
make install
```

7.4 Configuration

7.4.1 FreeS/WAN

There are some new configuration options in the FreeS/WAN config file */etc/ipsec.conf*:

- **ocspreqcert**: Path to certificate for signing OCSP requests. A matching private key must be included in */etc/ipsec.secrets*. If omitted, requests are not signed.
- **ocsprescert**: Path to certificate for validating OCSP responses. If omitted, responses must be signed either by the CA itself, or by a designated OCSP signer (and that certificate must be included in the response).
- **ocspuri**: URI of the OCSP server which will be used if no URI is included in the certificate.

The parameter *strictcrlpolicy* is used to enable strict certificate checking (similar to CRL).

A sample configuration could look like this (only section "config setup" is shown):

```
config setup
       crlcheckinterval=600                      # CRL/OCSP check intervall in seconds
       strictcrlpolicy=no                        # enable strict CRL/OCSP policy?
       ocspreqcert=ocsp-request-cert.pem         # OCSP request cert
       ocsprescert=ocsp-response-cert.pem        # OCSP response cert
       ocspuri=http://user:pass@123.45.67.89:2560 # OCSP default URI
...
```

7.4.2 Certificate generation

The usual way to use OCSP is to add an *authorityInfoAccess* to every certificate generated by the CA. If you would like to use OCSP with certificates without an URI included, you can use the *ocspuri* option as described above.

If you are using OpenSSL for generating certificates, you may add an OCSP URI to the certificates by adding an option (to x509_extensions) in your openssl.cnf.

```
authorityInfoAccess = OCSP;URI:http://my.ocspserver.org:2560/
```

The OCSP server needs to be either the CA itself, a normal certificate that needs to be trusted by the client, or a designated OCSP signer. We recommend to use a designated OCSP signer certificate, since it is the most simple and secure option.

A designated OCSP signer certificate contains a *extendedKeyUsage* extension. If you're using OpenSSL to generate your certificates, you may add the following option (to x509_extensions) to your *openssl.cnf* to generate such a certificate.

```
extendedKeyUsage=OCSPSigning
```

Please refer to the OpenSSL documentation for futher instructions.

7.4.3 OpenSSL OCSP server

The OpenSSL OCSP server takes his configuration from commandline options only. It is started by typing:

```
openssl ocsp <options>
```

Where options consist of the following:

- `-index <file>`
  The indexfile is a text index file in CA format containing certificate revocation information.
- `-CA <file>`
  CA certificate corresponding to the revocation information in indexfile.
- `-port <number>`
  The portnumber the server is listening on.
- `-rsigner <file>`
  The certificate to sign OCSP responses with.
- `-rkey <file>`
  The private key to sign OCSP responses with: if not present the file specified in the rsigner option is used.

- `-req_text, -resp_text, -text`
  Print out the text form of the OCSP request, response or both respectively. This is useful for logging/debugging.
- `-nmin <minutes>, -ndays <days>`
  Number of minutes or days when fresh revocation information is available: used in the nextUpdate field. If neither option is present then the nextUpdate field is omitted meaning fresh revocation information is immediately available.
- `-resp_no_certs`
  Don't include any certificates in the OCSP response.
- `-resp_key_id`
  Identify the signer certificate using the key ID, default is to use the subject name.

Example:

```
openssl ocsp -index index.txt -CA cacert.pem -port 2560 -rsigner signer.pem
```

This starts an OCSP server on port 2560, with the revocation information from the file index.txt. All responses are signed with the key from signer.pem, and the certificate from signer.pem is included in the response.

A complete list of commandline options is avaible from http://www.openssl.org/docs/apps/ocsp.html or from the ocsp manpage.

7.5 Usage

To manage the OCSP cache, two new whack options, *--listocsp* and *--purgeocsp* have been introduced.

7.5.1 List cache

To simply print out the contents of the cache, type:

```
ipsec auto –-listocsp
```

This will output the contents of the one-timer-cache (referred to as "short-term cache"):

```
000 OCSP short-term cache:
000
000 this_update Oct 10 12:17:09 2003
000    Certificate:
000        CA subject: 'C=CH, ST=Switzerland, L=Winterthur, O=ZHW, OU=IT3, CN=daCA, E=student@zhwin.ch'
000        CA subjkey:  41:9d:b7:e7:d0:37:9b:e7:20:37:bf:41:83:df:92:8a:0b:42:fe:33
000        serial:   C=CH, ST=Switzerland, L=Winterthur, O=ZHW, OU=IT3, CN=daCA, E=student@zhwin.ch
```

Also the contents of the main cache will be displayed:

```
000 OCSP cache:
000
000 Location http://dskt6817.zhwin.ch/
000   - next_update Oct 11 12:29:08 2003
000   - Certificates:
000        CA subject: 'C=CH, ST=Switzerland, L=Winterthur, O=ZHW, OU=IT3, CN=daCA, E=student@zhwin.ch'
000        CA subjkey:  41:9d:b7:e7:d0:37:9b:e7:20:37:bf:41:83:df:92:8a:0b:42:fe:33
000        serial:   09
000          Certificate status: revoked
```

7.5.2 Purge cache

If you would like to clear the cache, type:

```
ipsec auto --purgeocsp
```

8 Schlusswort

Da wir bei der Implementierung schnell vorangekommen sind, hatten wir die im Pflichtenheft beschriebenen Features vor Ablauf der geplanten Zeit fertiggestellt. Dies erlaubte uns, in Absprache mit Herrn Steffen zusätzliche Features zu erarbeiten. So konnte viel Zeit für die Detailpflege und den Test aufgebracht werden.

Mit dem Resultat unserer Diplomarbeit steht ein standardkonformer OCSP Client für FreeS/WAN zur Verfügung. Damit steht der Verwendung von FreeS/WAN in grossen Umgebungen nichts mehr im Wege.

Zum Schluss möchten wir uns bei Herrn Steffen für die kompetetenten und prompten Auskünfte sowie die freundliche Betreuung ganz herzlich bedanken.

9 Anhang

9.1 Projektplan

Zu Beginn unserer Diplomarbeit erstellten wir eine Grobplanung für den Ablauf unseres Projektes.

Die ersten Tage waren vor allem mit der Projektplanung und dem Einrichten der Arbeitsplätze ausgefüllt. Schon bald begannen wir damit, uns in die Thematik OCSP einzuarbeiten. Mit dem erarbeiteten Wissen erstellten wir auf Basis des RFC ein Pflichtenheft, welches den Umfang unserer Arbeit definierte.

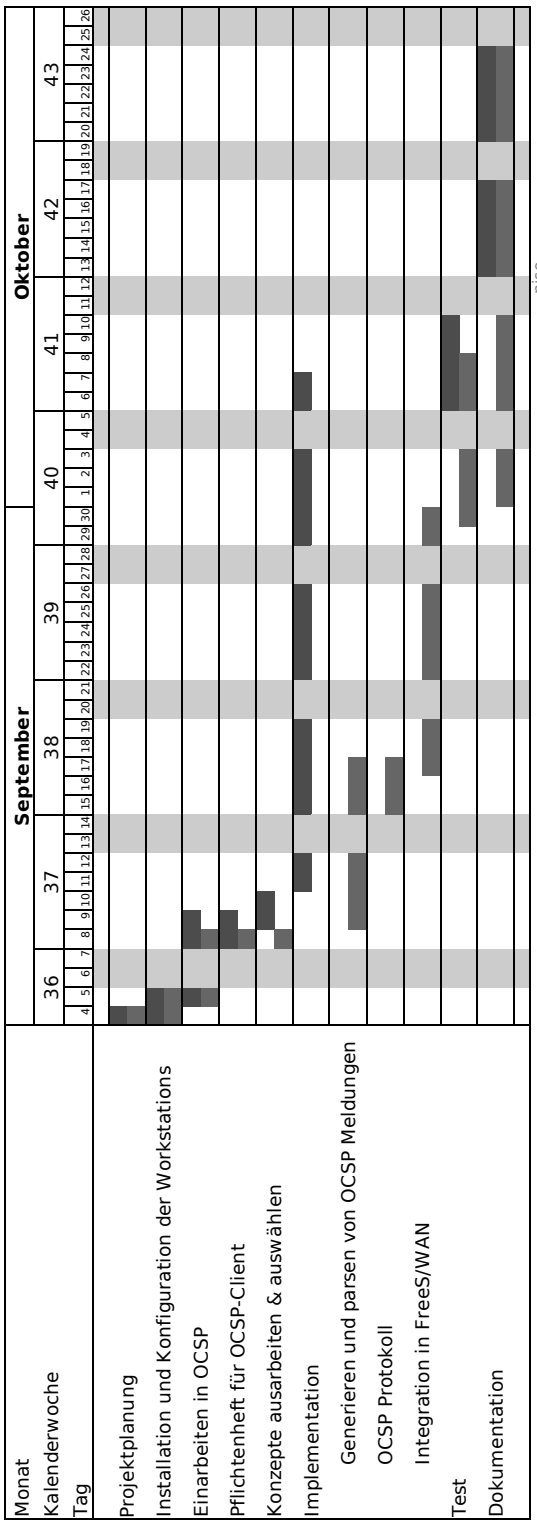
Auf Basis des Pflichtenheftes erarbeiteten wir ein Realisierungskonzept, womit wir bereits mit der Implementation beginnen konnten. Wir begannen mit dem Parsen und Generieren von OCSP Meldungen, woran ohne Probleme parallel gearbeitet werden konnte.

Als nächstes wurde eine kleine Demo-Applikation zur Verwendung von *libcurl* implementiert, deren Funktionen dann einfach in den bestehenden FreeS/WAN-Code übernommen werden konnte.

Den grössten Anteil des Implementierungs-Aufwandes war schliesslich, die OCSP-Funktionen komplett in FreeS/WAN integrieren. Hierzu gehörte auch die Implementierung der Caches und der dazugehörigen Funktionen, sowie die neuen Konfigurations- und Bedienungsparameter.

Mit dem frühzeitigen Abschluss der Realisierungsphase konnte genügend Zeit für einen ausführlichen Test und Detailpflege der Implementation verwendet werden.

Die letzten Wochen konnten schliesslich noch für das Fertigstellen der Dokumentation verwendet werden.



9.2 ASN.1 Syntax für OCSP-Meldungen

```
OCSP DEFINITIONS EXPLICIT TAGS::=
BEGIN
IMPORTS
```

9.2.1 Imports aus RFC2459

(Internet X.509 Public Key Infrastructure Certificate and CRL Profile)

```
        -- Directory Authentication Framework (X.509)
               Certificate, AlgorithmIdentifier, CRLReason
               FROM AuthenticationFramework { joint-iso-itu-t ds(5)
                        module(1) authenticationFramework(7) 3 }

Certificate  ::=  SEQUENCE  {
     tbsCertificate       TBSCertificate,
     signatureAlgorithm   AlgorithmIdentifier,
     signature            BIT STRING  }

TBSCertificate  ::=  SEQUENCE  {
     version         [0]  Version DEFAULT v1,
     serialNumber         CertificateSerialNumber,
     signature            AlgorithmIdentifier,
     issuer               Name,
     validity             Validity,
     subject              Name,
     subjectPublicKeyInfo SubjectPublicKeyInfo,
     issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                          -- If present, version shall be v2 or v3
     subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                          -- If present, version shall be v2 or v3
     extensions      [3]  Extensions OPTIONAL
                          -- If present, version shall be v3 --  }

Version  ::=  INTEGER  {  v1(0), v2(1), v3(2)  }

AlgorithmIdentifier  ::=  SEQUENCE  {
     algorithm               OBJECT IDENTIFIER,
     parameters              ANY DEFINED BY algorithm OPTIONAL  }

CRLReason ::= ENUMERATED {
     unspecified             (0),
     keyCompromise           (1),
     cACompromise            (2),
     affiliationChanged      (3),
     superseded              (4),
     cessationOfOperation    (5),
     certificateHold         (6),
     removeFromCRL           (8) }
```

```
    -- PKIX Certificate Extensions
                AuthorityInfoAccessSyntax
              FROM PKIX1Implicit88 {iso(1) identified-organization(3)
                      dod(6) internet(1) security(5) mechanisms(5) pkix(7)
                      id-mod(0) id-pkix1-implicit-88(2)}

    AuthorityInfoAccessSyntax  ::=
            SEQUENCE SIZE (1..MAX) OF AccessDescription

    AccessDescription  ::=  SEQUENCE {
            accessMethod          OBJECT IDENTIFIER,
            accessLocation        GeneralName  }


              Name, GeneralName, CertificateSerialNumber, Extensions,
               id-kp, id-ad-ocsp
                 FROM PKIX1Explicit88 {iso(1) identified-organization(3)
                      dod(6) internet(1) security(5) mechanisms(5) pkix(7)
                      id-mod(0) id-pkix1-explicit-88(1)};

Name ::= CHOICE {
  RDNSequence }

RDNSequence ::= SEQUENCE OF RelativeDistinguishedName

RelativeDistinguishedName ::=
  SET OF AttributeTypeAndValue

AttributeTypeAndValue ::= SEQUENCE {
  type     AttributeType,
  value    AttributeValue }

AttributeType ::= OBJECT IDENTIFIER

AttributeValue ::= ANY DEFINED BY AttributeType

    GeneralName ::= CHOICE {
         otherName                       [0]     AnotherName,
         rfc822Name                      [1]     IA5String,
         dNSName                         [2]     IA5String,
         x400Address                     [3]     ORAddress,
         directoryName                   [4]     Name,
         ediPartyName                    [5]     EDIPartyName,
         uniformResourceIdentifier       [6]     IA5String,
         iPAddress                       [7]     OCTET STRING,
         registeredID                    [8]     OBJECT IDENTIFIER }

    CertificateSerialNumber  ::=  INTEGER

Extensions  ::=  SEQUENCE SIZE (1..MAX) OF Extension
```

```
Extension  ::=  SEQUENCE  {
     extnID      OBJECT IDENTIFIER,
     critical    BOOLEAN DEFAULT FALSE,
     extnValue   OCTET STRING  }
```

9.2.2 ASN.1 Syntax Definitionen aus RFC2560

(X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP)

```
    OCSPRequest     ::=     SEQUENCE {
        tbsRequest                  TBSRequest,
        optionalSignature   [0]     EXPLICIT Signature OPTIONAL }

    TBSRequest      ::=     SEQUENCE {
        version             [0] EXPLICIT Version DEFAULT v1,
        requestorName       [1] EXPLICIT GeneralName OPTIONAL,
        requestList             SEQUENCE OF Request,
        requestExtensions   [2] EXPLICIT Extensions OPTIONAL }

    Signature       ::=     SEQUENCE {
        signatureAlgorithm   AlgorithmIdentifier,
        signature            BIT STRING,
        certs                [0] EXPLICIT SEQUENCE OF Certificate
    OPTIONAL }

    Version  ::=  INTEGER  {  v1(0) }

    Request ::=     SEQUENCE {
        reqCert                     CertID,
        singleRequestExtensions     [0] EXPLICIT Extensions OPTIONAL }

    CertID ::= SEQUENCE {
        hashAlgorithm           AlgorithmIdentifier,
        issuerNameHash     OCTET STRING, -- Hash of Issuer's DN
        issuerKeyHash      OCTET STRING, -- Hash of Issuers public key
        serialNumber       CertificateSerialNumber }

    OCSPResponse ::= SEQUENCE {
       responseStatus         OCSPResponseStatus,
       responseBytes          [0] EXPLICIT ResponseBytes OPTIONAL }

    OCSPResponseStatus ::= ENUMERATED {
        successful            (0),      --Response has valid confirmations
        malformedRequest      (1),      --Illegal confirmation request
        internalError         (2),      --Internal error in issuer
        tryLater              (3),      --Try again later
                                        --(4) is not used
        sigRequired           (5),      --Must sign the request
        unauthorized          (6)       --Request unauthorized
    }
```

```
ResponseBytes ::=       SEQUENCE {
    responseType   OBJECT IDENTIFIER,
    response       OCTET STRING }

BasicOCSPResponse       ::= SEQUENCE {
   tbsResponseData      ResponseData,
   signatureAlgorithm   AlgorithmIdentifier,
   signature            BIT STRING,
   certs                [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL }

ResponseData ::= SEQUENCE {
   version              [0] EXPLICIT Version DEFAULT v1,
   responderID              ResponderID,
   producedAt               GeneralizedTime,
   responses                SEQUENCE OF SingleResponse,
   responseExtensions   [1] EXPLICIT Extensions OPTIONAL }

ResponderID ::= CHOICE {
   byName   [1] Name,
   byKey    [2] KeyHash }

KeyHash ::= OCTET STRING --SHA-1 hash of responder's public key
                         --(excluding the tag and length fields)

SingleResponse ::= SEQUENCE {
   certID                       CertID,
   certStatus                   CertStatus,
   thisUpdate                   GeneralizedTime,
   nextUpdate           [0]     EXPLICIT GeneralizedTime OPTIONAL,
   singleExtensions     [1]     EXPLICIT Extensions OPTIONAL }

CertStatus ::= CHOICE {
    good                [0]     IMPLICIT NULL,
    revoked             [1]     IMPLICIT RevokedInfo,
    unknown             [2]     IMPLICIT UnknownInfo }

RevokedInfo ::= SEQUENCE {
    revocationTime              GeneralizedTime,
    revocationReason    [0]     EXPLICIT CRLReason OPTIONAL }

UnknownInfo ::= NULL -- this can be replaced with an enumeration

ArchiveCutoff ::= GeneralizedTime

AcceptableResponses ::= SEQUENCE OF OBJECT IDENTIFIER

ServiceLocator ::= SEQUENCE {
    issuer    Name,
    locator   AuthorityInfoAccessSyntax }
```

9.2.3 OCSP Request

```
OCSPRequest ::= SEQUENCE
    tbsRequest
      TBSRequest ::= SEQUENCE
        version              EXPLICIT Version DEFAULT v1
        requestorName        EXPLICIT GeneralName OPTIONAL
        requestList          SEQUENCE OF Request
          Request ::= SEQUENCE
            reqCert
              CertID ::= SEQUENCE
                hashAlgorithm
                  AlgorithmIdentifier ::= SEQUENCE {
                    algorithm          OBJECT IDENTIFIER
                    parameter          ANY DEFINED BY algorithm OPTIONAL
                issuerNameHash     OCTET STRING
                issuerKeyHash      OCTET STRING
                serialNumber
                  CertificateSerialNumber ::= INTEGER
            singleRequestExtensions          EXPLICIT Extensions OPTIONAL
    optionalSignature
      Signature ::= SEQUENCE {
        signatureAlgorithm
          AlgorithmIdentifier ::= SEQUENCE {
            algorithm          OBJECT IDENTIFIER
            parameter          ANY DEFINED BY algorithm OPTIONAL
        signature            BIT STRING
        certs                EXPLICIT SEQUENCE OF Certificate OPTIONAL
```

9.2.4 OCSP Response

```
OCSPResponse ::= SEQUENCE
    responseStatus
      OCSPResponseStatus ::= ENUMERATED
        successful              (0),    --Response has valid confirmations
        malformedRequest        (1),    --Illegal confirmation request
        internalError           (2),    --Internal error in issuer
        tryLater                (3),    --Try again later
                                (4),    --unused
        sigRequired             (5),    --Must sign the request
        unauthorized            (6),    --Request unauthorized
    responseBytes
      ResponseBytes ::= SEQUENCE {
        responseType       OBJECT IDENTIFIER
        response           OCTET STRING
```

9.2.5 OCSP Basic Response

```
BasicOCSPResponse ::= SEQUENCE
    tbsResponseData
      ResponseData ::= SEQUENCE
        version              EXPLICIT Version DEFAULT v1
        responderID
          ResponderID ::= CHOICE
            byName           Name
            byKey
              KeyHash ::= OCTET STRING
        producedAt           GeneralizedTime
        responses    SEQUENCE OF
          SingleResponse ::= SEQUENCE
            certID
              CertID ::= SEQUENCE
                hashAlgorithm
                  AlgorithmIdentifier ::= SEQUENCE {
                    algorithm        OBJECT IDENTIFIER
                    parameter        ANY DEFINED BY algorithm OPTIONAL
                issuerNameHash   OCTET STRING
                issuerKeyHash    OCTET STRING
                serialNumber
                  CertificateSerialNumber ::= INTEGER
            certStatus
              CertStatus ::= CHOICE
                good             IMPLICIT NULL
                revoked          IMPLICIT
                  RevokedInfo ::= SEQUENCE
                    revocationTime     GeneralizedTime
                    revocationReason   EXPLICIT OPTIONAL
                      CRLReason ::= ENUMERATED
                unknown          IMPLICIT
                  UnknownInfo ::= NULL
            thisUpdate           GeneralizedTime
            nextUpdate           EXPLICIT GeneralizedTime OPTIONAL
            singleExtensions     EXPLICIT Extensions OPTIONAL
        responseExtensions       EXPLICIT Extensions OPTIONAL
    signatureAlgorithm
      AlgorithmIdentifier ::= SEQUENCE {
        algorithm            OBJECT IDENTIFIER
        parameter            ANY DEFINED BY algorithm OPTIONAL
    signature            BIT STRING
    certs                EXPLICIT SEQUENCE OF Certificate OPTIONAL
```

9.3 Quellen

9.3.1 Literatur

[1] Charlie Kaufman, Radia Perlman, Mike Spencer
Network Security, 2nd Edition
2002, Prentice Hall
ISBN 0-13-046019-2

[2] Tobias Klein
Linux-Sicherheit
2001, dpunkt
ISBN 3-932588-04-5

[3] Andrew Tanenbaum
Computer Networks, 3rd Edition
1996, Prentice Hall
ISBN 0-13-394248-1

[4] RFC 1319 - **The MD2 Message-Digest Algorithm**
http://www.ietf.org/rfc/rfc1319.txt

[5] RFC 1321 - **The MD5 Message-Digest Algorithm**
http://www.ietf.org/rfc/rfc1321.txt

[6] RFC 2119 - **Key words for use in RFCs to Indicate Requirement Levels**
http://www.ietf.org/rfc/rfc2119.txt

[7] RFC 2313 - **PKCS #1: RSA Encryption**
http://www.ietf.org/rfc/rfc2313.txt

[8] RFC 2459 - **Internet X.509 Public Key Infrastructure Certificate and CRL Profile**
http://www.ietf.org/rfc/rfc2459.txt

[9] RFC 2560 - **X.509 Internet Public Key Infrastructure Online Certificate Status Protocol**
http://www.ietf.org/rfc/rfc2560.txt

[10] RFC 3370 - **Cryptographic Message Syntax (CMS) Algorithms**
http://www.ietf.org/rfc/rfc3370.txt

[11] ITU-T X.680 - **Abstract Syntax Notation One (ASN.1): Specification of basic notation**

[12] ITU-T X.690 - **ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)**

[13] RSA Laboratories - **PKCS #1 v2.1: RSA Cryptography Standard**
ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf

9.3.2 Links

[14] **IPSec** http://www.ietf.org/html.charters/ipsec-charter.htm

[15] **FreeS/WAN** http://freeswan.org
http://freeswan.ca

[16] **X.509 Patch** http://www.strongsec.com/freeswan

[17] **OpenSSL** http://www.openssl.org

[18] **CURL** http://curl.haxx.se

9.4 Glossar

AH	Authentication Header
ASN.1	Abstract Syntax Notation One
BER	Basic Encoding Rules
CA	Certificate Authority
CER	Canonical Encoding Rules
CRL	Certificate Revocation Lists
DER	Distinguished Encoding Rules
DES	Data Encryption Standard
DNS	Domain Name System
DH	Diffie-Hellman
ESP	Encapsulating Security Payload
GNU	GNU is Not Unix
GPG	GNU Privacy Gard
GPL	GNU General Public License
HTTP	Hypertext Transport Protocol
IEC	International Electrotechnical Commission
IETF	Internet Engineering Task Force
IKE	Internet Key Exchange
IP	Internet Protocol
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
IPSec	IP Security
ISAKMP	Internet Security Association and Key Management Protocol
ISO	International Organization for Standardization
KLIPS	Kernel Level IPSec
MD5	Message Digest 5
OCSP	Online Certificate Status Protocol
OID	Object Identifier
PKI	Public Key Infrastructure
RFC	Request For Comment
RSA	Rivest, Shamir, Adleman
SHA-1	Secure Hash Algorithm
SPI	Security Parameter Index
SSH	Secure Shell
SSL	Secure Socket Layer
S/WAN	Secure Wide Area Network
URI	Unified Ressource Identifier
VPN	Virtual Private Networks

9.5 Index

A

Aggressive Mode ..... 14p.
Anwendungstests 61, 71
ASN.1 26
ASN.1 Generierung 56
ASN.1 Parsing 57
Authentizität 13

B

BER 29

C

CER 29
Configuration 76
CRL 22, 40
CURL 42

D

DER 29
Digitale Signatur 18

E

Entschlüsselung 18

F

Fehlermeldungen 72
FreeS/WAN 16, 35

I

IKE 14
IKE Phase 1 14
IKE Phase 2 16
Installation 75
Integrität 13
IPSec 13
ISAKMP 14

K

Konfiguration 60

M

Main Mode 14

O

OCSP 22
OCSP Cache 46
OCSP Server 42
OpenSSL OCSP server 77

P

Pflichtenheft 10
PKI 19
Pluto 16, 35, 41
Public Key Verfahren 17

Q

Quick Mode 16

R

RSA 17

S

SA 13
SPI 13

V

Verschlüsselung 18
Vertraulichkeit 13

W

Whack 16

X

X.509 Zertifikate 20