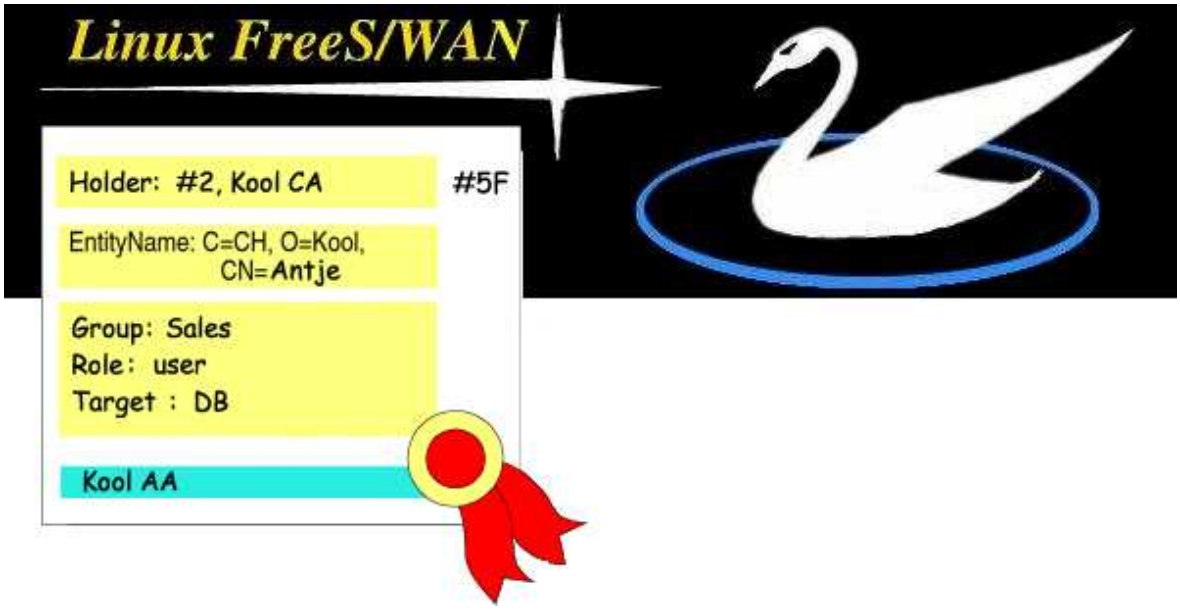
Zürcher
Hochschule
Winterthur

Mitglied
der Zürcher
Fachhochschule

Linux FreeS/WAN IPsec mit Attributszertifikaten



Projektarbeit
von
Martin Berner und Lukas Suter

Studiengang: Informationstechnolgie
Betreuender Dozent: Andreas Steffen

Winterthur, 4.7.2003

Inhaltsverzeichnis

1 Zusammenfassung

Viele Firmen besitzen bereits einen Breibandzugang ins Internet. Die teuren Mietleitungen, zur Verbindung von verschiedenen Standorten, werden immer mehr durch verschlüsselte Verbindungen (VPN's) via Internet ersetzt. Als nächster Schritt soll nun Mitarbeitern das Arbeiten via x-beliebigem Internetzugang (ADSL, Cablemodem, Mobiltelefon, ...) möglich gemacht werden, ohne dabei die Sicherheit des Firmen-Netzwerkes zu beeinträchtigen.

Um einer grossen Zahl von Remote-Benutzern den Zugang zu ermöglichen, ist der Einsatz von x.509 User-Zertifikaten nötig, womit die Benutzer eindeutig und sicher authentifiziert werden können. Das VPN-Gateway soll abhängig vom Benutzer nur den Zugriff auf einzelne Dienste und Server freischalten, wobei das Definieren der Regeln flexibel und benutzerfreundlich sein soll. Mit dem Einsatz von Attributs-Zertifikaten können diese Anforderungen erfüllt werden, da die Authentifizierung von der Autorisierung getrennt wird. In einem Attributs-Zertifikat sind die Rollen-, die Gruppenzugehörigkeit und bei Bedarf die Zielplatform individuell für jeden Benutzer definiert. Erst damit können komplexe Zugriffsregeln mit vernüftigem Aufwand realisiert werden. Die Attributs-Zertifikate werden auf einem LDAP-Server gespeichert und vom VPN-Gateway nach erfolgter Benutzer-Authentifizierung abgerufen und angewendet. Der LDAP-Server muss dabei nicht zur Festung ausgebaut werden, da jeweils vor der Verwendung eines Attribut-Zertifikates, die Signatur überprüft wird.

Mit dem Abschluss dieser Projektarbeit ist FreeS/WAN nun in Lage die jeweiligen Attributs-Zertifikate beim Verbindungsaufbau via LDAP anzufordern und zu analysieren. Die Gültigkeit wird aufgrund des im Zertifikat definierten Zeitrahmens festgestellt. Die Signaturüberprüfung wurde vorbereitet, kann aber aufgrund fehlender Funktionen zur Einbindung von Authorization-Authority-Zertifikaten noch nicht durchgeführt werden. Weiter müssten Teile der User-Zertifikate im Speicher verbleiben, um die Attributs-Zertifikate noch zweifelsfreier den entsprechenden Usern zuzuordnen.
Die Attributs-Zertifikate sind im Gegensatz zu den User-Zertifikaten in der Regel mit einer kurzen Gültigkeitsdauer von 1 bis 25 Stunden versehen. Dadurch kann auf das Führen einer Certificate Revokation List (CRL) verzichtet werden. Ein einmal in FreeS/WAN geladenes Attributs-Zertifikat, wird periodisch auf seine Gültigkeit überprüft und bei Bedarf neu vom LDAP-Server angefordert.
Die Auswahl der korrekten Verbindung, aufgrund der neu vorhandenen Attribute, ist noch nicht fertig gestellt.

2 Projektbeschreibung

2.1 Aufgabenstellung

Linux FreeS/WAN IPsec mit Attributszertifikaten

Sichere Netzwerkkommunikation (SNK)

Projektarbeiten im SS 2003 - PA2 Sna01

Studierende:
- Martin Berner, IT3
- Lukas Suter, IT3

Termine:
- Ausgabe: Dienstag, 20.05.2003 im E523
- Abgabe: Freitag, 4.07.2003

Beschreibung:
Seit drei Jahren trägt die ZHW einen beträchtlichen Teil zum populären Linux FreeS/WAN IPsec Stack bei. So wurde unter anderem die Benutzerauthentisierung auf der Basis von X.509 Zertifikaten realisiert. Mit der ausgeschriebenen Projektarbeit soll ein mächtiger Schritt Richtung Management von komplexen Virtual Private Networks mit Hunderten bis Tausenden von Remote-Access Benutzern getan werden. Dabei sollen die Zugriffsrechte auf Subnetze und Dienste im durch eine Firewall geschützten Intranet flexibel und benutzerfreundlich definiert werden können. Dazu sollen Attributszertifikate verwendet werden, welche die Rollen und Gruppenzugehörigkeit individuell für jeden Benutzer definieren.
Im Rahmen einer letztjährigen ZHW Projektarbeit ist ein Tool entstanden, das die Erstellung von Attributszertifikaten erlaubt. Zurzeit wird in einer weiteren Arbeit eine web-basierte Managementoberfläche für dieses Tool erstellt, welche die Erzeugung und Verteilung von Attributszertifikaten erleichtern, respektive automatisieren soll.

Mittels der ausgeschriebenen Arbeit soll es möglich werden, den VPN-Zugriff von einer Gruppen- zugehörigkeit oder einer Rolle abhängig zu machen. Um die Zugriffsrechte verifizieren zu können, muss das Attributszertifikat des Benutzers bei einem OpenLDAP-Server abgeholt, in einem internen Cache abgelegt und das Benutzerprofil analysiert werden.

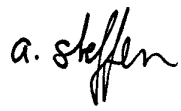
Aufgaben:
- Einarbeiten in den Linux FreeS/WAN 2.00 IPsec Stack mit OpenLDAP Anbindung
- Anwendung von Attributszertifikaten und Erzeugung mittels OpenAC..
- Erstellen eines Realisierungskonzepts.
- Implementation und Test der Einbindung von Attributszertifikaten in FreeS/WAN.

Infrastruktur / Tools:
- Raum: E523
- Rechner: 2 PCs mit Windows/Linux
- SW-Tools: Linux FreeS/WAN, OpenAC, OpenLDAP

Literatur / Links:
- An Internet Attribute Certificate Profile for Authorization
  http://www.ietf.org/rfc/rfc3281.txt
- ZHW Projektarbeit von Ueli Gallizzi und Ariane Seiler
  http://home.zhwin.ch/~sna/PA/PAK2_2002_Sna04.pdf
- OpenAC Projekt von Ueli Gallizzi und Ariane Seiler
  http://www.strongsec.com/openac/
- OpenLDAP Projekt
  http://www.openldap.org

- FreeS/WAN Inofficial Mirror Site
  http://www.freeswan.ca
- X.509 Patch für FreeS/WAN
  http://www.strongsec.com/freeswan/

Winterthur, 20. Mai 2003



Prof. Dr. Andreas Steffen

2.2 Resultate

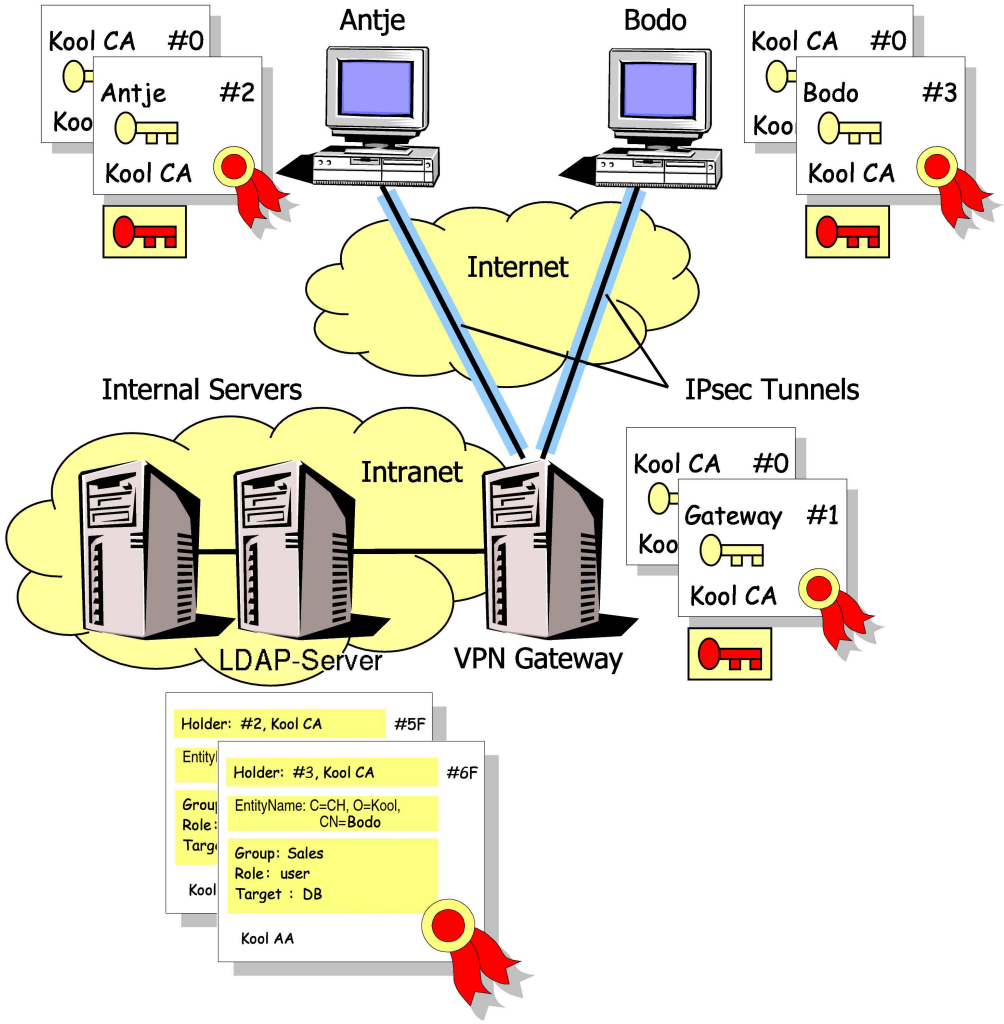
+ es wird eine Liste von AC's geführt
+ greift ein User auf den Server zu wird automatisch dessen AC vom LDAP-Server zu holen versucht
+ neu wird auch LDAP Version 3 unterstützt, falls Version-2-Bind fehlschlägt
+ Die Listen der AC-Fetchrequests und der AC's werden sauber gelöscht und der Speicher freigegeben
+ die Attribute group, role und target werden bei der Statusanzeige durch whack ausgegeben falls vorhanden
+ Die Liste der AC's und der AC-Fetchrequests sind mittels eines Mutexes geschützt
+ Die Liste der AC's und der AC-Fetchrequests können durch whack resp. auto angezeigt werden
+ Die URL des LDAP-Servers kann im Konfigurationsfile angegeben werden
- Fetchrequests für AC's werden auch gelöscht falls sie nicht erfolgreich waren
- Die Signatur der AC's wird noch nicht überprüft
  --> AA müsste vorhanden sein, sowie Teile des User-Zertifikats (u.a. Serial)
- Im Quick-Mode werden die Attribute group, role und target (Benutzerprofil) noch nicht berücksichtigt zur Auswahl einer Connection
- In der Implementation wurde mit "holderIssuer" gearbeitet, anstatt mit "entityName", da bis gegen Ende der Arbeit keine korrekten AC's mit entityName zur Verfügung standen

2.3 Erkenntnisse

- Nicht im vornherein erkennbare Hindernisse, machten eine Redimensionierung der Ziele nötig. Damit konnten nicht alle Punkte der Aufgabenstellung erfüllt werden. Authorization Authority fehlte, User-Zertifikate nicht im Speicher gehalten, LDAP-trouble, keine korrekten AC's vorhanden.
- Sich in den Code von FreeS/WAN einzudenken war schwieriger als erwartet. FreeS/WAN ist auf sehr viele Files verteilt und der gesammte Ablauf war oft schwierig nachzuvollziehen. Vor allem wie die Übergabe von Parametern aus dem Konfigurationsfile letztendlich funktioniert, ist uns bis heute nicht bis ins letzte Detail klar geworden. Unter Mithilfe von A. Steffen haben wir in den entsprechenden Files, analog zu anderen Parametern, eigene Einträge gemacht.

3 Lösungskonzept

3.1 Architektur



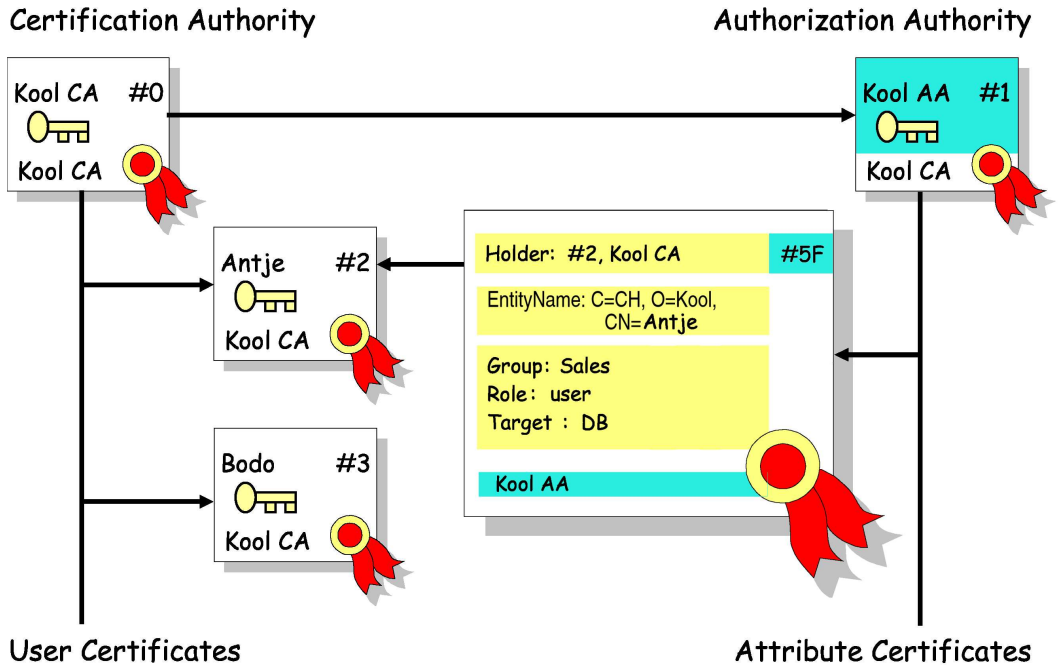
Attribute Certificates signed from the Authorization Authority (AA)

Vereinfachter Ablauf

1. Roadwarrior stellt Verbindung zum VPN-Gateway her und liefert sein User-Zertifikat mit
2. User-Zertifikat wird geprüft (Gültigkeit, Signatur)
3. Entsprechendes Attributs-Zertifikat wird via LDAP angefordert
   (Initiator-ID im Format DER_ASN1_DN muss dazu identisch wie das Subject des User-Zertifikates sein, dann erfolgt die Anfrage, ldap://ldapbaseurl/initiator-id?attributeCertificate)
4. Attributs-Zertifikat wird geprüft (Gültigkeit, Signatur und ob passend zu User-Zertifikat)
   (Signaturprüfung und exakte Zugehörigkeitsprüfung zu UC noch nicht implementiert)
5. Beste Verbindungsdefinition wird gesucht, Zugriff gemäss Role/Group/Target ermöglicht
   (noch nicht implementiert)

Die auf dem LDAP-Server gespeicherten Attributs-Zertifikate (AC) werden von einer Authorization Authority ausgestellt. In einer Firma kann es mehrere Authorization Authority geben, z.B. eine pro Abteilung, diese können autonom die AC's erstellen und entlasten damit den VPN-Gateway-Administrator.
Der LDAP-Server könnte auch in einem Extranet stehen, welches per statischem VPN-Tunnel verbunden ist, dem Wunsch grosser Firmen nach Zentralisierung könnte damit entsprochen werden.

3.2 Attribute Certificates Authorization scheme

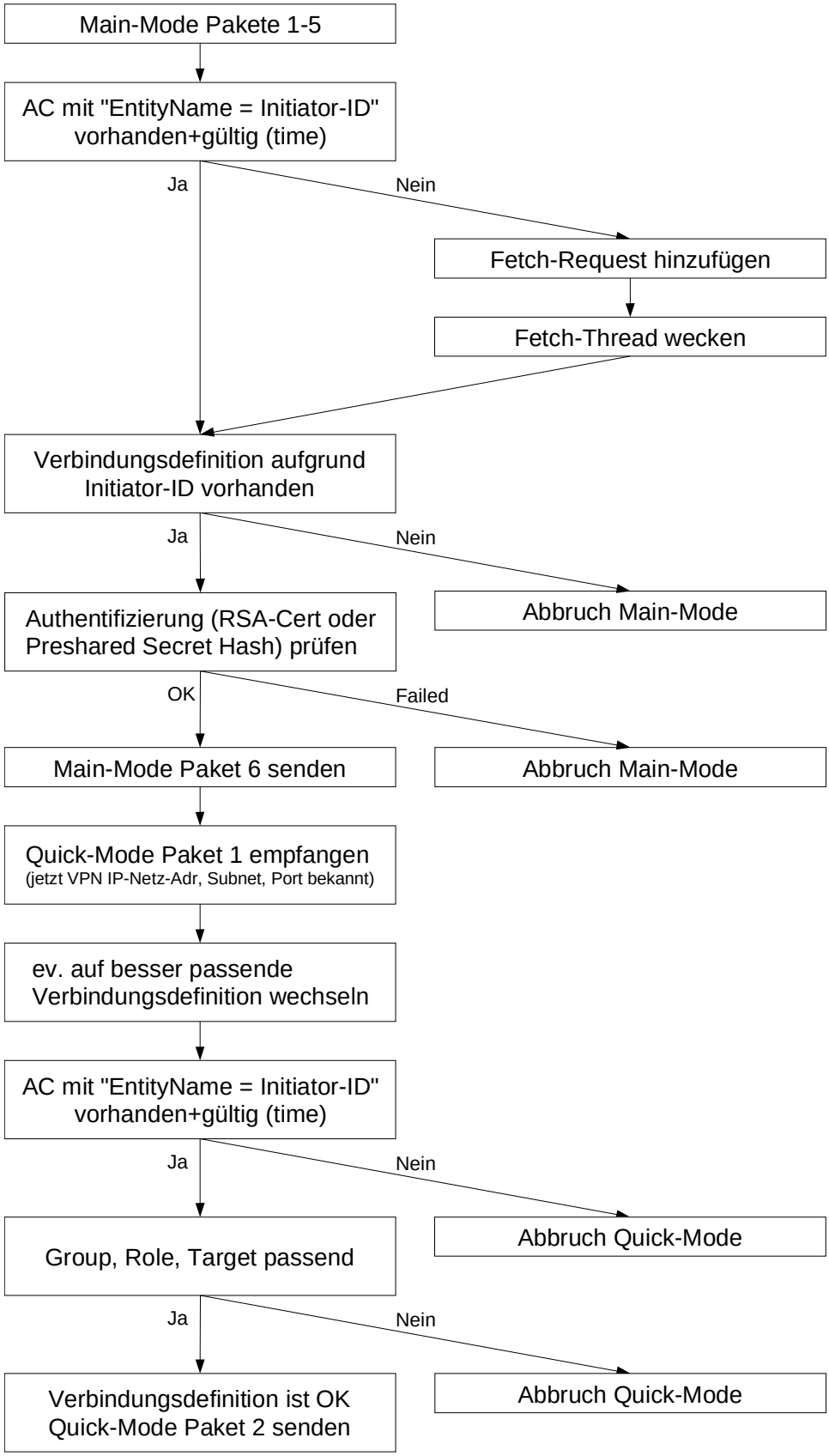


Überprüfung der Attributs-Zertifikate

Zur Überprüfung eines Attribut-Zertifikates muss die Authentizität der Authorization Authority (AA) mittels Kontrolle der CA-Signatur bzw. der CA mittels root-Cert überprüft werden. Anschliessend folgt die Kontrolle User-Zertifikat <--> Attribut-Zertifikat (Serialnumber, Subject == EntityName).

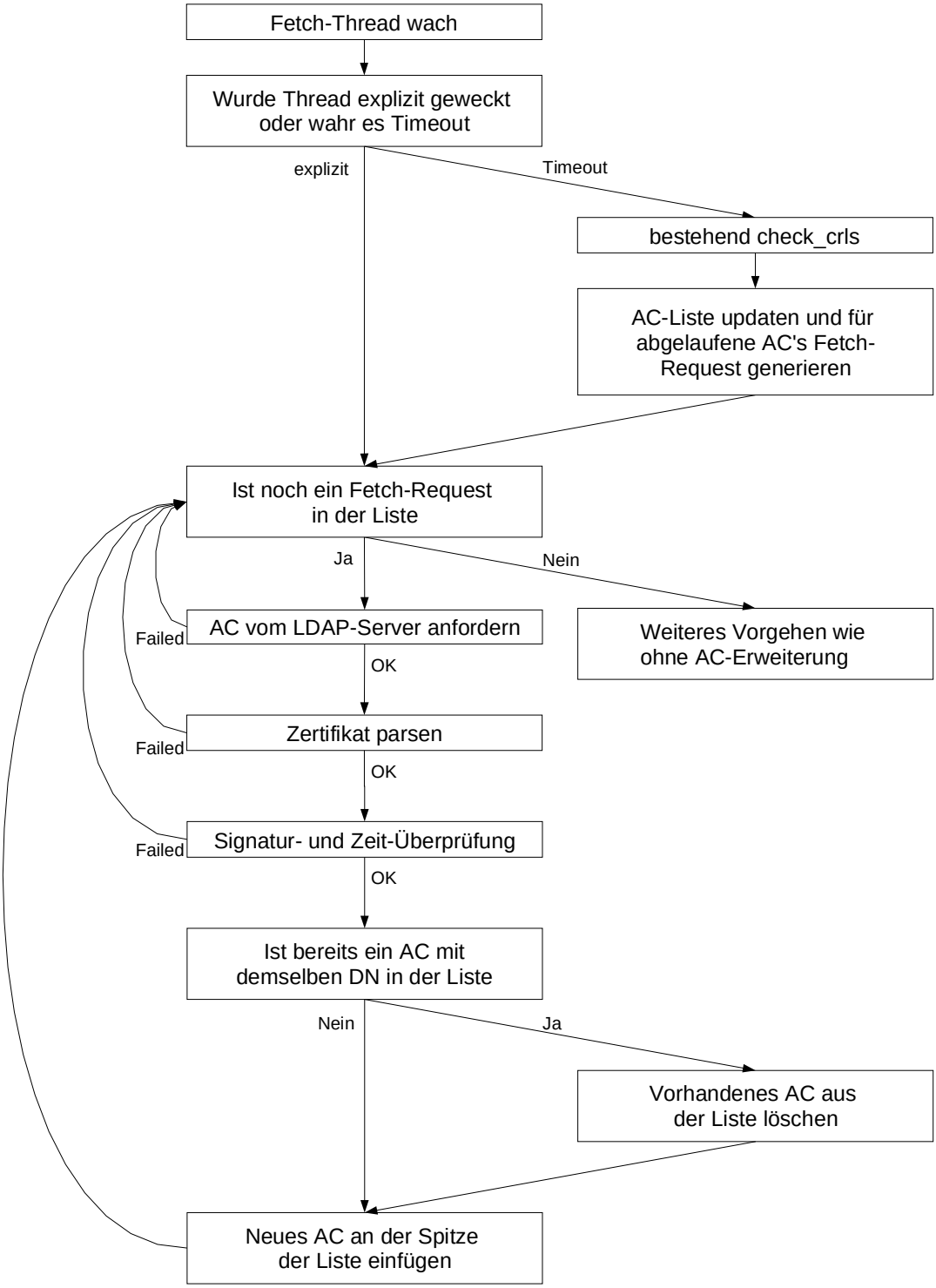
3.3 Ablauf

3.3.1 Übersicht



Details zu Main- und Quick-Mode siehe 3.3.3 und 3.3.4

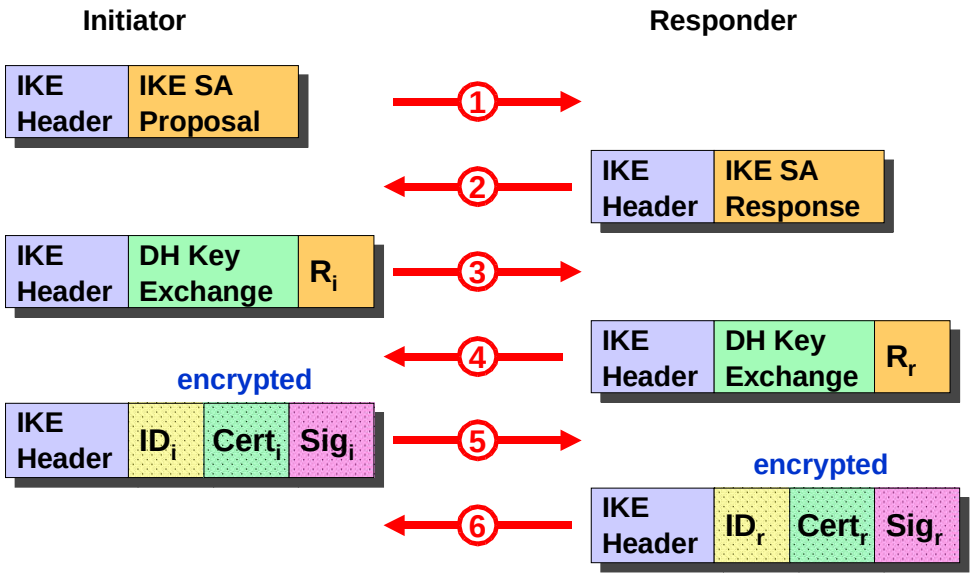
3.3.2 Fetch-Ablauf



Design-Überlegungen zum abholen der AC's (Variante 2 wurde implementiert)

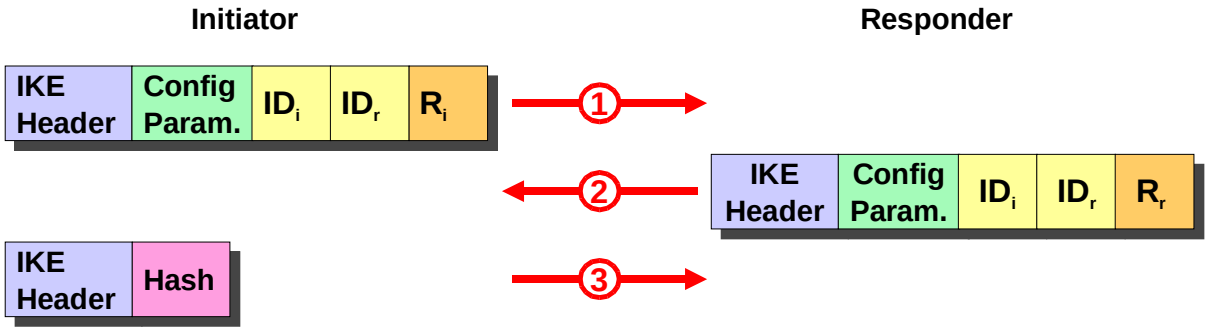
1. Aufgrund der ldapbaseurl werden in regelmässigen Abständen (crlcheckinterval) "alle" Attributszertifikate abgeholt
   VT: AC ist bei Verbindungsaufbau eines Clients sicher vorhanden
   NT: Speicherverbrauch (ca. 1kByte/AC), LDAP-Traffic
2. Nach Bedarf, d.h. nach dem Main-Mode, wird das AC des Verbindungswilligen Users angefordert. Dieses wird bei Ablauf automatisch neu angefordert.
   VT: Geringer Speicherverbrauch, weniger LDAP-Traffic, Skalierbarkeit noch machbar
   NT: Zeitbedarf für Abfrage, ev. ist AC beim Quick-Mode noch nicht verfügbar

3.3.3 Main Mode (IKE)



Quelle: SNK_VPN.ppt A.Steffen

3.3.4 Quick Mode



Config Param. enthält:

VPN IP, [Subnet, Port], AH, ESP, authentication / encryption methods and parameters

3.4 Funktionen der Module

3.4.1 Konfiguration

Durch die Einbindung von Attributszertifikaten kommen die Parameter **group**, **role** und **target** zusätzlich in die 'conn' - Definitionen. Das bedeutet, dass die Überprüfung des Konfigurationsfiles auf Korrektheit in der Datei _confread.in angepasst werden muss.

Damit die Parameter auch bis ins Hauptprogramm kommen, muss das Steuerprogramm auto.in respektive whack angepasst werden.

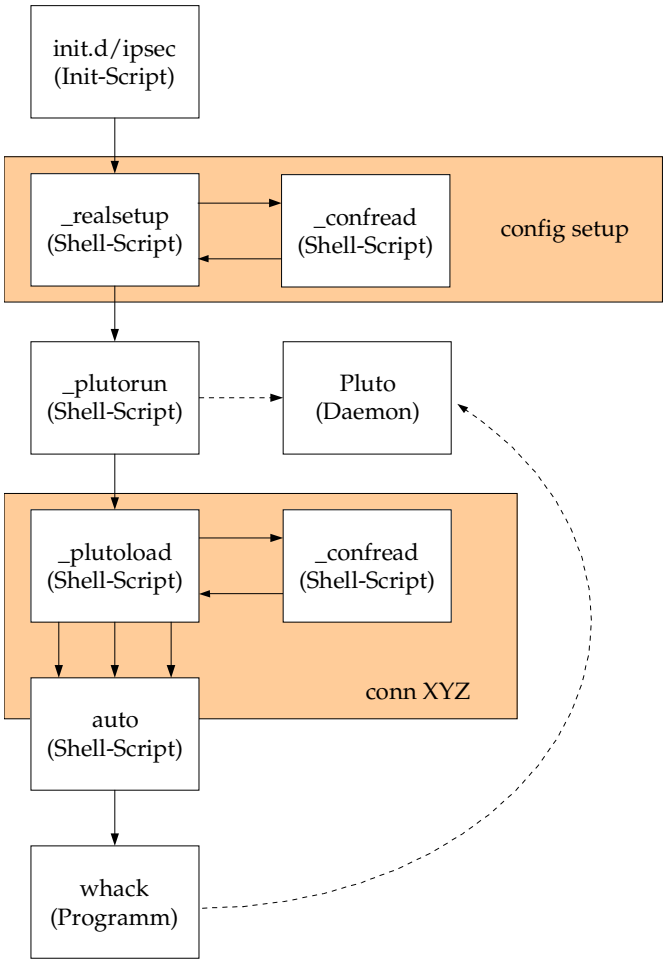
Die neuen Parameter gehören dann zu einer Connection und deshalb muss auch in der Definition einer Connection (connections.h und connections.c) dieser Parameter eingeführt werden.

Für die Abfrage eines LDAP-Servers muss dem Mainprogramm die URL des Servers bekannt sein. Diese wird im 'config setup'-Teil von ipsec.conf mit dem Parameter 'ldapbaseurl' angegeben. Die URL wird anders als die zusätzlichen Parameter in den 'conn' - Definitionen bereits beim Aufstarten respektive dem Neueinlesen der Configdatei übergeben.

Das Updaten und Fetchen der aktuell in der Liste vorhandenen AC's werden im bereits vorhandenen Fetch-Thread für die CRL's abgearbeitet. Deshalb gilt der unter "crlcheckinterval" angegebene Intervallwert auch für die das Updaten und Fetchen der AC's.

Files: _confread.in, plutorun.in, _realsetup.in, auto.in, connections.h, connections.c, plutomain.c, whack.h, whack.c, rcv_whack.c, ipsec.conf

Ablauf



3.4.2 AC's überprüfen

Jedes AC ist ausgestelt von einer Authorization Authority (AA), und zeigt auf das User-Zertifikat für das es gilt. Ebenfalls ist wie jedes Zertifikat auch das AC zeitlich beschränkt gültig (meist ca 25h). Die Signaturüberprüfung ist in dieser Arbeit noch nicht implementiert worden da verschiedene Informationen und zuletzt auch die Zeit dafür gefehlt haben. Für die Signaturüberprüfung müsste die AA verfügabr sein, siehe 3.2

Zur Überprüfung der zeitlichen Gültigkeit muss darauf geachtet werden, dass alle Zeiten in UTC angegeben werden. Ein AC ist gültig falls die aktuelle UTC zwischen den Werten 'notBefore' und 'notAfter' liegt.
Achtung besser "time + 2 * crlcheckinterval" <= notAfter, damit AC auch wirklich die ganze Zeit gültig (in der vorhandenen Implementierung wurde crlcheckinterval noch nicht berücksichtigt)

Files: ac.h, ac.c

3.4.3 AC in Liste einfügen

Bevor ein AC in die Liste der bekannten AC's kommt wird überprüft ob nicht bereits ein AC mit dem selben EntityName in der Liste ist. Danach wird die Gültigkeit des Zertifikates überprüft. Falls das einzufügende AC gültig ist wird es am Anfang der Liste eingehängt und falls es vorher schon in der Liste existierte, wird das alte aus der Liste genommen.

Files: ac.h, ac.c

3.4.4 Updaten der AC-Liste

Um sicherzustellen, dass alle AC's in der Liste noch gültige AC's sind wird vom Fetchthread periodisch alle AC's in der Liste auf ihre Gültigkeit überprüft. Ist ein AC abgelaufen so wird, falls nicht bereits ein Fetchrequest für dieses AC existiert, ein neuer generiert. Nachdem alle AC's geprüft worden sind wird die Liste der Fetchrequests abgearbeitet. Sollte aus irgend-einem Grund ein Fetchrequest nicht erfolgreich abgearbeitet werden können, so sollte er in der Liste bleiben und erst nach mehrmaligem Versuch aus der Liste entfernt werden. Das entfernen eines nicht erfolgreichen Fetchrequests sollte auch bewirken, dass das betreffende (ungültige) AC aus der Liste entnommen wird.

Aus Zeitlichen Gründen ist die Behandlung von nicht erfolgreichen Fetchrequests noch nicht implementiert worden.

Files: ac.h, ac.c, fetch.h, fetch.c

3.4.5 AC im Quick-Mode aus der Liste auslesen/anfordern

Erst Quick-Mode wird die endgültige Connection-Definition ermittelt. Um dies unter Berücksichtigung der zusätzlichen AC-Attribute zu tun, muss das AC des Clients angefordert werden. In der AC-Liste wird nach dem AC mit dem entsprechenden EntityName gesucht und falls eines gefunden wird, dieses zurückgegeben. Ist jedoch keines in der AC-Liste so wird ein Fetchrequest für dieses generiert.

Dieses Verhalten wird im Main-Mode ausgenutzt, um sicherzustellen dass das AC eines Users im Quick-Mode möglichst schon vorhanden ist. Ist ein gültiges AC in der Liste wird nichts gemacht, sonst falls es noch keines hat oder abgelaufen ist, wird zu diesem Zeitpunkt bereits das AC des betreffenden Users mittels eines Fetchrequests angefordert.

Files: ac.h, ac.c, ipsec_doi.c

3.4.6 Auflisten der AC's und AC-Fetchrequests

Das Steuerprogramm whack kann veranlassen dass die Liste der AC's und der AC-Fetchrequests ausgegeben wird. Dafür wird durch die jeweilige Liste gegangen und die wichtigsten Informationen durch den Logger auf dem Bildschirm ausgegeben.

AC-Informationen:

- Installations-Zeit und -Datum
- HolderIssuer
- HolderSerial
- entityName
- notBefore
- notAfter
- group
- role
- target

AC-Fetchrequest-Informationen:

- Installations-Zeit und -Datum
- trials
- entityName

Files: ac.h, ac.c, fetch.h, fetch.c, auto.in, whack.h, whack.c, rcv_whack.c

3.4.7 Statusauflistung durch whack

Das Steuerprogramm whack kann den Status der aktuellen Verbindungen ausgeben. Damit auch die neuen AC-Attribute einer Verbindung ausgegeben werden mussten noch Anpassungen in der Funktion 'show_connections' gemacht werden.

Files: connections.c

4 Zeitplanung

Woche	Geplant	Effektiv
1	Freeswan Praktikas, Konzeption, Freeswan Struktur verstehen, LDAP-Server installieren	Freeswan+LDAP installation, Freeswan Praktikas, Freeswan Struktur verstehen
2	Freeswan Struktur verstehen, Implementation div Funktionen	Konzeption, Freeswan Struktur verstehen, Role/Group/Target-Parameter implementiert
3	Implementation div. Funktionen	Konzeption, LDAP-Versuche (ldif, add, search)
4	Implementation div. Funktionen	Quellcode Laroche studieren, openac-Versuche, LDAP Abfrageversuch in fetch.c
5	Implementation div. Funktionen, Dokumentation	Implementation div. Funktionen
6	Abschlussarbeiten an Implementation, Dokumentation	Implementation div. Funktionen, Abschlussarbeiten
7	Dokumentation, Diff-file zum patchen	Abschlussarbeiten, Dokumentation

5 Implementation

5.1 Neue Funktionen

bool parse_ac(chunk_t blob, ac_cert_t *ac)

Übergabewerte:

struct chunk_t blob	AC-Blob
struct ac_cert *ac	Pointer auf leeres AC-Zertifikat

Rückgabewert:

bool	bei Erfolg true, sonst false

Der übergebene AC-Blob wird geparst und in das übergebene AC-Cert abgefüllt.

void dn_rfc2253(char *str)

Übergabewerte:

char *str	DN-String

Rückgabewert:

void	

Der übergebene Destinguished Name wird umgekehrt und so "LDAP-Search konform" gemacht (rfc2253).

err_t check_ac_time_validity(const ac_cert_t *accert, time_t *until)

Übergabewerte:

struct ac_cert_t *accert	Pointer auf ein AC-Cert
time_t *until	Pointer auf eine Timevariable

Rückgabewert:

err_t	noch nicht gültig 'ac certificate is not valid yet', abgelaufen 'ac certificate has expired'

Das übergebene AC-Cert wird geprüft, ob es zeitlich noch gültig ist bis zum Zeitpunkt until. Falls nicht wird until auf die Zeit gesetzt bis wohin es noch gültig wäre.

bool check_ac_signature(chunk_t tbs, chunk_t sig, chunk_t algorithm, const x509cert_t *issuer_cert)

Übergabewerte:

struct chunk_t tbs	ACInfo
struct chunk_t sig	Signatur
struct chunk_t algorithm	Verschlüsselungsalgorithmus
struct x509cert_t *issuer_cert	AuthorizationAuthority

Rückgabewert:

bool	true falls gültig, sonst false

Es wird getestet ob die Signatur gültig ist.

bool insert_ac(chunk_t acblob, chunk_t dn)

Übergabewerte:

struct chunk_t acblob	Der AC-Blob
struct chunk_t dn	Der Distinguished Name des AC's

Rückgabewert:

bool	true falls es eingefügt wurde, sonst false

Der AC-Blob wird auf seine Gültigkeit überprüft und falls gültig in die Liste der AC's eingehängt. Ein eventuell schon vorhandenes, älteres AC mit demselben dn wird aus der Liste entfernt. Bei der zeitlichen Überprüfung des AC's wird noch nicht auf 2xCrlCheckInterval geprüft (ist noch zu implementieren).

ac_cert_t* get_ac(chunk_t dn)

Übergabewerte:

struct chunk_t dn	DN des gewünschten AC's

Rückgabewert:

struct ac_cert_t*	Pointer auf das gewünschte AC, sonst NULL

Es wird ein AC in der Liste gesucht mit entsprechendem DN. Falls keines vorhanden ist wird NULL zurückgegeben.

void update_ac_list(void)

Übergabewerte:

void	

Rückgabewert:

void	

Die Liste der AC's wird durchlaufen und kontrolliert ob die Zertifikate noch gültig sind. Ist ein Zertifikat abgelaufen, wird ein AC-Fetchrequest generiert. Bei der zeitlichen Überprüfung des AC's wird noch nicht auf 2xCrlCheckInterval geprüft (ist noch zu implementieren).

void free_ac_cert(ac_cert_t *cert)

Übergabewerte:

struct ac_cert_t *cert	Pointer auf ein AC

Rückgabewert:

void	

Das übergebene AC wird gelöscht und der Speicher dafür freigegeben.

void free_ac_certs(void)

Übergabewerte:

void	

Rückgabewert:

void	

Es wird durch die Liste der AC's gegangen und jedes AC mittels free_ac_cert gelöscht.

void list_acs(bool utc)

Übergabewerte:

bool utc	true übergeben, wenn Zeit in UTC

Rückgabewert:

void	

Gibt die Liste der AC's auf dem Bildschirm aus.

static void lock_ac_fetch_list(const char *who)

Übergabewerte:

char *who	von wem die Funktion aufgerufen wird

Rückgabewert:

void	

Die Liste der AC-Fetchrequests wird mittels Mutex geschützt.

static void unlock_ac_fetch_list(const char *who)

Übergabewerte:

char *who	von wem die Funktion aufgerufen wird

Rückgabewert:

void	

Die Liste der AC-Fetchrequests wird wieder freigegeben.

void lock_ac_list(const char *who)

Übergabewerte:

char *who	von wem die Funktion aufgerufen wird

Rückgabewert:

void	

Die Liste der AC's wird mittels Mutex geschützt.

void unlock_ac_list(const char *who)

Übergabewerte:

char *who	von wem die Funktion aufgerufen wird

Rückgabewert:

void	

Die Liste der AC's wird wieder freigegeben.

static void free_ac_fetch_request(ac_fetch_req_t *req)

Übergabewerte:

struct ac_fetch_req_t *req	Pointer auf ein AC-Fetchrequest

Rückgabewert:

void	

Der übergebene AC-Fetchrequest wird gelöscht und dessen Speicher freigegeben.

void free_ac_fetch_requests(void)

Übergabewerte:

void	

Rückgabewert:

void	

Es wird durch die Liste der AC-Fetchrequests gegangen und jeder AC-Fetchrequest mittels free_ac_fetch_reqest gelöscht.

static void update_acs(void)

Übergabewerte:

void	

Rückgabewert:

void	

Die Liste der AC-Fetchrequests wird abgearbeitet und anschliessend die Liste gelöscht.

void add_ac_fetch_req(chunk_t dn)

Übergabewerte:

struct chunk_t dn	DN des geforderten AC's

Rückgabewert:

void	

Falls nicht schon ein AC-Fetchrequest für diesen DN in der Liste existiert, wird ein solcher generiert.

void list_ac_fetch_requests(bool utc)

Übergabewerte:

bool utc	true übergeben, wenn Zeit in UTC

Rückgabewert:

void	

Gibt die Liste der AC-Fetchrequests auf dem Bildschirm aus.

5.2 Konfigfile ipsec.conf

```
# /etc/ipsec.conf - FreeS/WAN IPsec configuration file

# basic configuration

version 2.0

config setup
      # THIS SETTING MUST BE CORRECT or almost nothing will work;
      # %defaultroute is okay for most simple cases.
      interfaces=%defaultroute
      # Debug-logging controls:  "none" for (almost) none, "all" for lots.
      klipsdebug=none
      plutodebug=control
      # Use auto= parameters in conn descriptions to control startup actions.
      # plutoload=%search
      # plutostart=%search
      # Close down old connection when new one using same ID shows up.
      uniqueids=yes
      # Enforce updated CRLs
      strictcrlpolicy=no
      crlcheckinterval=180
      # ldapserver for Acs, host[:port] (default-port 389)
      # ldap.example.net:524
      ldapbaseurl=dskt6805.zhwin.ch


# defaults for subsequent connection descriptions

conn %default
      # Lifetime of ISAKMP and IPse SAs
      ikelifetime=3h
      keylife=1h
      keyingtries=0
      disablearrivalcheck=no
      # RSA authentication based on certificates
      authby=rsasig
      rightrsasigkey=%cert
      # Local side is left
      left=%defaultroute
      leftcert=dskt6680_cert.pem
      rightgroup=Sales
      lefttarget=DNS
      rightrole=Admin


# connection descriptions

conn dskt6803
      leftcert=hostCert.pem
      right=160.85.170.131
      rightid="C=CH, O=ZHW, OU=SNK, CN=dskt6693.zhwin.ch"
      rightsubnet=160.85.170.131/30
      auto=add
```

5.3 Fileübersicht

File	Zweck
_confread.in	Kontrolle des Konfigfiles
_plutorun.in	Übergabe von Parametern
_realsetup.in	Start von Pluto mit Parametern
auto.in	Hilfssteuerprogramm
Makefile	Kompilations- und Installationsfile
ac.c	AC relevante Funktionen
ac.h	AC relevante Funktionen
asn1.h	Erweiterungen von A.Steffen
connections.c	Structs und Funktionen für Connections
connections.h	Structs und Funktionen für Connections
fetch.c	Structs und Funktionen des Fetchthreads
fetch.h	Structs und Funktionen des Fetchthreads
ipsec_doi.c	Verbindungsaufbau (Main- und Quick-Mode)
oid.c	Parsing-relevante Daten
oid.h	Parsing-relevante Daten
oid.txt	Parsing-relevante Daten
plutomain.c	Mainprogramm von Pluto
rcv_whack.c	Empfänger des Steuerprogramms
whack.c	Structs und Funktionen des Steuerprogramms
whack.h	Structs und Funktionen des Steuerprogramms
x509.c	X509-Certifikat-relevante Funktionen und Structs
x509.h	X509-Certifikat-relevante Funktionen und Structs

5.4 Quellcodes

5.4.1 ac.h

```
/* access structure for an X.509 attribute certificate */

typedef struct ac_cert ac_cert_t;


struct ac_cert {
  ac_cert_t      *next;
  time_t    installed;
  chunk_t   certificate;
  chunk_t     certificateInfo;
  u_int             version;
          /*   holder */
          /*     baseCertificateID */
  chunk_t        holderIssuer;
  chunk_t        holderSerial;
  chunk_t        entityName;
                 /*   v2Form */
  chunk_t         issuerName;
                 /*   signature */
  chunk_t              sigAlg;
  chunk_t       serialNumber;
                 /*   attrCertValidityPeriod */
  time_t               notBefore;
  time_t               notAfter;
          /*   attributes */
  chunk_t              group;
          /*   extensions */
          /* signatureAlgorithm */
  chunk_t           algorithm;
  chunk_t          signature;
};

/* used for initialization */
extern const ac_cert_t empty_ac;

extern bool parse_ac(chunk_t blob, ac_cert_t *ac);

extern void dn_rfc2253(char *str);
extern bool insert_ac(chunk_t acblob, chunk_t dn);
extern ac_cert_t* get_ac(chunk_t dn);
extern void update_ac_list(void);
extern err_t check_ac_time_validity(const ac_cert_t *accert, time_t *until);
extern bool check_ac_signature(chunk_t tbs, chunk_t sig, chunk_t algorithm,const
x509cert_t *issuer_cert);
extern void free_ac_cert(ac_cert_t *cert);
extern void free_ac_certs(void);
extern void list_acs(bool utc);
```

5.4.2 ac.c

```
#include <stdlib.h>
#include <time.h>

#include <freeswan.h>

#include "constants.h"
#include "defs.h"
#include "asn1.h"
#include "oid.h"
#include "id.h"
#include "x509.h"
#include "log.h"
#include "ac.h"
#include "whack.h"
#include "fetch.h"

/* ASN.1 definition von ietfAttrSyntax */

static ac_cert_t *ac_certs  = NULL;     //acadd

static const asn1Object_t ietfAttrSyntaxObjects[] = {
  { 0, "policyAuthority",       ASN1_CONTEXT_C_0,     ASN1_OPT |
                                          ASN1_BODY }, /*  0 */
...
```

```
  { 1,   "signature",               ASN1_BIT_STRING,      ASN1_BODY }  /* 58 */
};

#define AC_OBJ_CERTIFICATE            0
#define AC_OBJ_CERTIFICATE_INFO       1
...
...
#define AC_OBJ_SIGNATURE        58
#define AC_OBJ_ROOF             59

const ac_cert_t empty_ac = {
      NULL     , /* *next */
            0  , /* installed */
    { NULL, 0 }, /* certificate */
    { NULL, 0 }, /*   certificateInfo */
            1  , /*     version */
           /*     holder */
           /*       baseCertificateID */
    { NULL, 0 }, /*         holderIssuer */
    { NULL, 0 }, /*         holderSerial */
           /*       entityName */
    { NULL, 0 }, /*         generalNames */
           /*     v2Form */
    { NULL, 0 }, /*       issuerName */
                 /*     signature */
    { NULL, 0 }, /*       sigAlg */
    { NULL, 0 }, /*     serialNumber */
                 /*     attrCertValidityPeriod */
            0  , /*       notBefore */
            0  , /*       notAfter */
           /*     attributes */
    { NULL, 0 }, /*       group */
           /*     extensions */
           /*   signatureAlgorithm */
    { NULL, 0 }, /*     algorithm */
    { NULL, 0 }, /*   signature */
};


/* Maximum length of ASN.1 distinquished name */

#define BUF_LEN       512

/*
 *  Parses an X.509 attribute certificate
 */
bool
parse_ac(chunk_t blob, ac_cert_t *ac)
{
    u_char  buf[BUF_LEN];
    asn1_ctx_t ctx;
    bool critical;
    chunk_t extnID;
    chunk_t object;
    int objectID = 0;

    asn1_init(&ctx, blob, 0, FALSE, DBG_RAW);

    while (objectID < AC_OBJ_ROOF) {

     if (!extract_object(acObjects, &objectID, &object, &ctx))
          return FALSE;

     switch (objectID)
     {
     case AC_OBJ_CERTIFICATE:
         ac->certificate = object;
         break;
     case AC_OBJ_CERTIFICATE_INFO:
         ac->certificateInfo = object;
         break;
     case AC_OBJ_VERSION:
         ac->version = 1 + (u_int)*object.ptr;
         DBG(DBG_PARSING,
          DBG_log("  v%d", ac->version);
         )
         break;
```

```
      case AC_OBJ_HOLDER_ISSUER:
          ac->holderIssuer = get_directoryName(object
            , acObjects[objectID].level, FALSE);
          break;
      case AC_OBJ_HOLDER_SERIAL:
          ac->holderSerial = object;
          break;
      case AC_OBJ_ENTITY_NAME:
          ac->entityName = get_directoryName(object
            , acObjects[objectID].level, TRUE);
          break;
      case AC_OBJ_ISSUER_NAME:
          ac->issuerName = get_directoryName(object
            , acObjects[objectID].level, FALSE);
      case AC_OBJ_SIG_ALG:
          ac->sigAlg = object;
          break;
      case AC_OBJ_SERIAL_NUMBER:
          ac->serialNumber = object;
          break;
      case AC_OBJ_NOT_BEFORE:
          ac->notBefore = asn1totime(&object, ASN1_GENERALIZEDTIME);
          break;
      case AC_OBJ_NOT_AFTER:
          ac->notAfter = asn1totime(&object, ASN1_GENERALIZEDTIME);
          break;
      case AC_OBJ_EXTN_ID:
          extnID = object;
          break;
      case AC_OBJ_CRITICAL:
          critical = object.len && *object.ptr;
          DBG(DBG_PARSING,
            DBG_log("  %s",(critical)?"TRUE":"FALSE");
          )
          break;
      case AC_OBJ_EXTN_VALUE:
          {
           u_int extn_oid = known_oid(extnID);
           u_int level = acObjects[objectID].level + 1;

           if (extn_oid == OID_TARGET_INFORMATION)
               DBG_log("  parse target");
          }
          break;
      case AC_OBJ_ALGORITHM:
          ac->algorithm = object;
          break;
      case AC_OBJ_SIGNATURE:
          ac->signature = object;
          break;

      default:
      }
      objectID++;
    }
    time(&ac->installed);
    return TRUE;
}


/* Reverse a DN string extracted from a X509 certificates to comply
*  * with RFC 2253, code from Stephane Laroche */
void
dn_rfc2253(char *str)
{
    char basedn[BUF_LEN];
    char *b;
    char *p;
    int first = 1;

    basedn[0] = '\0';
    b = basedn;
    while ((p = strrchr(str, ','))) {
        *p = 0;
        do { p++; } while (isspace(*p));
        if (!first) strcat(b, ",");
        strcat(b, p);
        b += strlen(b);
        first = 0;
    }
```

```
    if (!first) strcat(b, ",");
    strcat(b, str);
    strcpy(str, basedn);
}

/* verify the validity of a ac-certificate by
 * checking the notBefore and notAfter times
 */
err_t
check_ac_time_validity(const ac_cert_t *accert, time_t *until){
     time_t current_time;
     time(&current_time);

     DBG(DBG_PARSING,
          DBG_log("  ac not before  : %s", timetoa(&accert->notBefore, TRUE));
          DBG_log("  ac current time: %s", timetoa(&current_time, TRUE));
          DBG_log("  ac not after   : %s", timetoa(&accert->notAfter, TRUE));
     )

     if (accert->notAfter < *until) *until = accert->notAfter;

     if (current_time < accert->notBefore)
          return "ac certificate is not valid yet";
     if (current_time > accert->notAfter)
          return "ac certificate has expired";
     else
     return NULL;
};

/*
 *   Check if a signature over binary ac-blob is genuine
 *
 *   not implemented yet
 *   never tested so far, may it works
 */
bool
check_ac_signature(chunk_t tbs, chunk_t sig, chunk_t algorithm, const x509cert_t
*issuer_cert){
     // issuer_cert needs to be the cert of the authorization authority (AA)
     // tbs = acinfo
     // sig = signature
     // check (algorithm == sigAlg), after checking the signature

     return TRUE;

     return check_signature(tbs, sig, algorithm, issuer_cert);
};

/*   Inserts an attribute certificate into the List of ACs
 *   if it is valid and not already there
 */
bool
insert_ac(chunk_t acblob, chunk_t dn){
     ac_cert_t *ac_cert_list;
     ac_cert_t *tmp_ac_cert;
     ac_cert_t *prev_ac_cert;
     err_t ugh = NULL;
     time_t *until;

     chunk_t unused;

     lock_ac_list("insert_ac");

     ac_cert_list = ac_certs;
     prev_ac_cert = ac_certs;
     while (ac_cert_list != NULL){
          if (same_dn(ac_cert_list->holderIssuer, dn)) break;
          prev_ac_cert = ac_cert_list;
          ac_cert_list = ac_cert_list->next;
     }
     tmp_ac_cert = alloc_thing(ac_cert_t, "tmp ac cert");
     *tmp_ac_cert = empty_ac;

     if(parse_ac(acblob, tmp_ac_cert)){
          log("parsing ac successfull");
          if(check_ac_signature(unused, unused, unused, NULL)){
               log("signature of ac valid");
               until = &tmp_ac_cert->notAfter;
               ugh = check_ac_time_validity(tmp_ac_cert, until);
               if (ugh == NULL){
                    log("ac is valid (time)");
```

```
                else            // ac out of time validity
                {
                     log("%s", ugh);
                     unlock_ac_list("insert_ac");
                     free_ac_cert(tmp_ac_cert);
                     return FALSE;
                }
           }
           else                 // signature invalid
           {
                log("signature of ac invalid");
                unlock_ac_list("insert_ac");
                free_ac_cert(tmp_ac_cert);
                return FALSE;
           }
      }
      else
      {                         // parse-error
           log("could not parse ac");
           unlock_ac_list("insert_ac");
           free_ac_cert(tmp_ac_cert);
           return FALSE;
      }
      if (ac_cert_list != NULL){     // cert found in the list (while-statement)
           // replace ac
           if (prev_ac_cert == ac_cert_list){   // 1. certificate in the list needs
to be exchanged
                log("old ac at 1th position");
                ac_certs = ac_cert_list->next;
           }
           else
           {
                log("old ac at some other position");
                prev_ac_cert->next = ac_cert_list->next;
           }
           free_ac_cert(ac_cert_list);
      }
      log("instaling %s at 1th position", timetoa(&tmp_ac_cert->installed, TRUE));
      tmp_ac_cert->next = ac_certs;
      ac_certs = tmp_ac_cert;
      unlock_ac_list("insert_ac");
      return TRUE;
};

/* search the list for an ac with same dn
 * return it, if found
 * else add a fetch request into the fetch-list and return NULL */
ac_cert_t*
get_ac(chunk_t dn){
      ac_cert_t *ac_cert_list;
      err_t ugh = NULL;
      time_t *until;
      lock_ac_list("get_ac");
      ac_cert_list = ac_certs;
      while(ac_cert_list != NULL){
           if (same_dn(ac_cert_list->holderIssuer, dn)){
                log("found ac with %s in the list", dn.ptr);
                until = &ac_cert_list->notAfter;
                ugh = check_ac_time_validity(ac_cert_list, until);
                if (ugh == NULL){
                     log("ac is valid now");
                     unlock_ac_list("get_ac");
                     return ac_cert_list;
                } else {
                     log("%s",ugh);
                     add_ac_fetch_req(dn);
                     wake_fetch_thread("get_ac");  // wake up the fetch thread to
handle the fetch-request immediatly
                     unlock_ac_list("get_ac");
                     return NULL;
                }
           }
           ac_cert_list = ac_cert_list->next;
      }
      log("ac with %s was put into the ac-fetch-list",dn.ptr);
      // no ac cert with same "dn" was found in the list
      unlock_ac_list("get_ac");
      return NULL;
};
```

```
/*
 * check if some ACs in the list are out of time
 * and add fetch-requests of them into the fetch-list
 */
void
update_ac_list(void){
     ac_cert_t *ac_cert_list;
     err_t ugh = NULL;
     time_t *until;
     lock_ac_list("update_ac_list");
     ac_cert_list = ac_certs;
     while(ac_cert_list != NULL){
          until = &ac_cert_list->notAfter;
          ugh = check_ac_time_validity(ac_cert_list, until);
          if (ugh == NULL){
               log("ac from %s is still valid",ac_cert_list->holderIssuer.ptr);
          }
          else
          {
               log("ac from %s needs to be fetched new", ac_cert_list-
>holderIssuer.ptr);
               add_ac_fetch_req(ac_cert_list->holderIssuer);
          }
          ac_cert_list = ac_cert_list->next;
     }
     unlock_ac_list("update_ac_list");
};

/*
 *  free an ac certificate
 */
void
free_ac_cert(ac_cert_t *cert)
{
     if (cert != NULL)
     {
          if (cert->certificate.ptr != NULL)
               pfree(cert->certificate.ptr);
          pfree(cert);
          cert = NULL;
     }
}

/*
 *  free all ac certificates
 */
void
free_ac_certs(void)
{
     lock_ac_list("free_ac_certs");
     while(ac_certs != NULL){
          ac_cert_t *ac = ac_certs;
          ac_certs = ac_certs->next;
          free_ac_cert(ac);
     }
     unlock_ac_list("free_ac_certs");
}

/*
 * list all ac certs in the chained list
 */
void
list_acs(bool utc){
     ac_cert_t *acs;
     lock_ac_list("list_acs");
     acs = ac_certs;

     if (acs != NULL)
     {
          whack_log(RC_COMMENT, " ");
          whack_log(RC_COMMENT, "List of ACs:");
          whack_log(RC_COMMENT, " ");
     }

     while (acs != NULL)
     {
          u_char buf[BUF_LEN];

          /*
           * print out the following parameters:
           *    -installed
```

```
             *    -entityName
             *    -notBefore
             *    -notAfter
             *    -group
             *    (-role)
             *    (-target)
             */
            whack_log(RC_COMMENT, "installed:  %s", timetoa(&acs->installed, utc));
            dntoa_or_null(buf, BUF_LEN, acs->holderIssuer, "not set");
            whack_log(RC_COMMENT, "       holderIssuer:  '%s'", buf);
            dntoa_or_null(buf, BUF_LEN, acs->holderSerial, "not set");
            whack_log(RC_COMMENT, "       holderSerial:  '%s'", buf);
            dntoa_or_null(buf, BUF_LEN, acs->entityName, "not set");
            whack_log(RC_COMMENT, "       entityName:  '%s'", buf);
            whack_log(RC_COMMENT, "       notBefore:  %s", timetoa(&acs->notBefore,
utc));
            whack_log(RC_COMMENT, "       notAfter:  %s", timetoa(&acs->notAfter,
utc));
            whack_log(RC_COMMENT, "");

            acs = acs->next;
        }
        unlock_ac_list("list_acs");
}
```

5.4.3 _confread.in

Damit die neuen Parameter im ipsec.conf erkannt werden. Analog zu leftca

```
...
        left = left " leftespenckey leftespauthkey leftahkey"
        left = left " leftespspi leftahspi leftid leftrsasigkey leftrsasigkey2"
        left = left " leftcert leftca leftsubnetwithin leftprotoport"
        left = left " lefttarget leftgroup leftrole"
        mkey = mkey " ah ahkey ahreplay_window"
        right = left
        gsub(/left/, "right", right)
...
...
        good = good " fragicmp hidetos rp_filter uniqueids"
        good = good " overridemtu"
        good = good " nocrsend strictcrlpolicy crlcheckinterval"
        good = good " ldapbaseurl"
        n = split(good, g)
        for (i = 1; i <= n; i++)
                goodnames["config:" g[i]] = 1
...
```

5.4.4 _plutorun.in

Damit die URL für den LDAP-Server übergeben werden kann wird analog zu crlcheckinterval verfahren

5.4.5 _realsetup.in

Damit die URL für den LDAP-Sserver übergeben werden kann wird analog zu crlcheckinterval verfahren

5.4.6 auto.in

Damit via whack die Listen der AC's und der AC-Fetchrequests ausgegeben werden können wird analog zu listcrls verfahren

Damit via whack die neuen Attribute einer Connection verändert werden können

```
...
            lkod = ""
            rkod = ""
            ltarget = ""
            if ("lefttarget" in s)
                ltarget = "--target " qs("lefttarget")
            rtarget = ""
            if ("righttarget" in s)
                rtarget = "--target " qs("righttarget")
            lgroup = ""
            if ("leftgroup" in s)
                lgroup = "--group " qs("leftgroup")
            rgroup = ""
```

```
            if ("rightgroup" in s)
                 rgroup = "--group " qs("rightgroup")
            lrole = ""
            if ("leftrole" in s)
                 lrole = "--role " qs("leftrole")
            rrole = ""
            if ("rightrole" in s)
                 rrole = "--role " qs("rightrole")
            if (authtype != "--psk") {
                 kod = ""
                 whackkey("left", "rsasigkey", "")
...
```

Der Übergabestring an whack wird zusammengestellt

```
...
            print "ipsec whack --name", name, settings, "\\"
            print "\t--host", qs("left"), lc, lp, "--nexthop",
                 qs("leftnexthop"), lud, lid, lkod, lcert, lca, ltarget, lgroup,
lrole, "\\"
            print "\t--to", "--host", qs("right"), rc, rp, "--nexthop",
+                qs("rightnexthop"), rud, rid, rkod, rcert, rca, rtarget, rgroup,
rrole, "\\"
            print "\t--ipseclifetime", qs("keylife"),
                 "--rekeymargin", qs("rekeymargin"), "\\"
            print "\t--keyingtries", qs("keyingtries"), fuzz, rk, pd, "\\"
...
```

5.4.7 Makefile

Damit auch unsere neuen Files ac.h und ac.c mitkompiliert werden

```
...
     x509.c x509.h \
     $(DISTGCRYPT) \
     adns.c adns.h \
     whack.c whack.h \
     ac.c ac.h
 DIST = $(DISTMISC) $(DISTSRC)
...
...
 OBJSPLUTO = asn1.o connections.o constants.o cookie.o crypto.o defs.o foodgroups.o
\
     log.o state.o plutomain.o server.o timer.o oid.o pem.o pgp.o pkcs.o x509.o \
     certs.o id.o ipsec_doi.o kernel.o rcv_whack.o demux.o packet.o lex.o keys.o \
     dnskey.o fetch.o rnd.o spdb.o sha1.o md5.o md2.o ac.o

 OBJSADNS = adns.o
...
```

5.4.8 connections.h

Im Struct eines Endpunktes einer Connection braucht es neu auch Platz für die AC-relevanten Parameter group, role und target

```
...
struct end {
    struct id id;
    ip_address
     host_addr,
     host_nexthop;
    ip_subnet client;

    bool key_from_DNS_on_demand;
    bool has_client;
    bool has_client_wildcard;
    bool has_id_wildcards;
    char *updown;
    u_int16_t host_port;  /* host order */
    u_int16_t port;       /* host order */
    u_int8_t protocol;
    cert_t cert;          /* end certificate */
    chunk_t ca;           /* CA distinguished name */
    char *group;          /* group name */
    char *role;           /* role name */
    char *target;         /* target name */
};
...
```

5.4.9 connections.c

Damit keine Memoryleaks entstehen müssen die neuen Parameter auch wieder gelöscht werden, wenn eine Connection gelöscht wird

```
void
delete_connection(struct connection *c)
{
    struct connection *old_cur_connection
...
...
    pfreeany(c->name);
    free_id_content(&c->this.id);
    pfreeany(c->this.updown);
    freeanychunk(c->this.ca);
    release_cert(c->this.cert);
    pfreeany(c->this.group);
    pfreeany(c->this.role);
    pfreeany(c->this.target);
    free_id_content(&c->that.id);
    pfreeany(c->that.updown);
    freeanychunk(c->that.ca);
    release_cert(c->that.cert);
    pfreeany(c->that.group);
    pfreeany(c->that.role);
    pfreeany(c->that.target);
    gw_delref(&c->gw_info);
    pfree(c);
}
...
```

Damit die neuen Parameter Speicherplatz zur Verfügung haben, muss neuer Speicher alloziert werden und dann abgefüllt

```
...
static void
unshare_connection_strings(struct connection *c)
{
    c->name = clone_str(c->name, "connection name");

    unshare_id_content(&c->this.id);
    c->this.updown = clone_str(c->this.updown, "updown");
    share_cert(c->this.cert);
    if (c->this.ca.ptr != NULL)
      clonetochunk(c->this.ca, c->this.ca.ptr, c->this.ca.len, "ca string");
    c->this.target = clone_str(c->this.target, "target string");
    c->this.group = clone_str(c->this.group, "group string");
    c->this.role = clone_str(c->this.role, "role string");
    unshare_id_content(&c->that.id);
    c->that.updown = clone_str(c->that.updown, "updown");
    share_cert(c->that.cert);
    if (c->that.ca.ptr != NULL)
      clonetochunk(c->that.ca, c->that.ca.ptr, c->that.ca.len, "ca string");
    c->that.target = clone_str(c->that.target, "target string");
    c->that.group = clone_str(c->that.group, "group string");
    c->that.role = clone_str(c->that.role, "role string");
}
...
...
static bool
extract_end(struct end *dst, const struct whack_end *src, const char *which)
{
    bool same_ca = FALSE;
...
...
    /* does id has wildcards? */
    dst->has_id_wildcards = id_count_wildcards(&dst->id) > 0;

    /* the rest is simple copying of corresponding fields */
    dst->host_addr = src->host_addr;
    dst->host_nexthop = src->host_nexthop;
    dst->client = src->client;
    dst->port = src->port;
    dst->protocol = src->protocol;
    dst->key_from_DNS_on_demand = src->key_from_DNS_on_demand;
    dst->has_client = src->has_client;
    dst->has_client_wildcard = src->has_client_wildcard;
    dst->updown = src->updown;
    dst->host_port = src->host_port;
    dst->target = src->target;
    dst->group = src->group;
    dst->role = src->role;
```

```
...
Falls in einer Connection einer die Parameter group, role, und target gesetzt sind
werden sie bei Aufruf der Funktion 'show_connections' ausgegeben
...
show_connections_status(void)
{
    struct connection *c;
...
...
     /* show group, role and target */
     {
         char GRT[200];
         if(c->this.group
           || c->this.role
           || c->this.target
           || c->that.group
           || c->that.role
           || c->that.target)
         {
             if(c->this.group) sprintf(GRT, "%slgroup: %s  ",GRT ,c->this.group);
             if(c->this.role) sprintf(GRT, "%slrole: %s  ",GRT ,c->this.role);
             if(c->this.target) sprintf(GRT, "%sltarget: %s  ",GRT ,c-
>this.target);
             if(c->that.group) sprintf(GRT, "%srgroup: %s  ",GRT ,c->that.group);
             if(c->that.role) sprintf(GRT, "%srrole: %s  ",GRT ,c->that.role);
             if(c->that.target) sprintf(GRT, "%srtarget: %s  ",GRT ,c-
>that.target);
             whack_log(RC_COMMENT
             , "\"%s\"%s:   %s"
             , c->name
             , instance
             , GRT);
         }
         sprintf(GRT,"");
     }
...
```

5.4.10 fetch.h

```
...
extern void lock_ac_list(const char *who);          //acadd
extern void unlock_ac_list(const char *who);        //acadd
extern void lock_crl_list(const char *who);
extern void unlock_crl_list(const char *who);
extern void lock_cacert_list(const char *who);
    , generalName_t **distributionPoints);
extern void add_fetch_request(chunk_t issuer, const generalName_t *gn);
extern void free_fetch_requests(void);
extern void free_ac_fetch_requests(void);               //acadd
extern void list_distribution_points(const generalName_t *gn);
extern void list_fetch_requests(bool utc);
extern void add_ac_fetch_req(chunk_t dn);           //acadd
extern void list_ac_fetch_requests(bool utc);            //acadd
```

5.4.11 fetch.c

```
Für die AC-Fetchrequests wird eine eigene Liste mit einem eigenen Typ geführt und
bleibt somit getrennt von den CRL-Fetchrequests
...
typedef struct ac_fetch_req ac_fetch_req_t;

struct ac_fetch_req {
     ac_fetch_req_t  *next;
     time_t          installed;      //adding time
     int             trials;         //number of failures, not limitted yet
     chunk_t         dn;             //asn1
};

/* chained list of ac fetch requests */
static ac_fetch_req_t *ac_fetch_reqs  = NULL;  //acadd

ac_fetch_req_t empty_ac_fetch_req = {
   NULL     , /* next */
         0 , /* installed */
         0 , /* trials */
  {NULL, 0}, /* issuer */
};
```

```
typedef struct fetch_req fetch_req_t;

struct fetch_req {
...
```
Für einen kontrollierten Zugriff auf die Listen der AC's und AC-Fetchrequests werden Mutexes benötigt
```
...
static pthread_t thread;
static pthread_mutex_t cacert_list_mutex =   PTHREAD_MUTEX_INITIALIZER;
static pthread_mutex_t crl_list_mutex =      PTHREAD_MUTEX_INITIALIZER;
static pthread_mutex_t fetch_list_mutex =    PTHREAD_MUTEX_INITIALIZER;
static pthread_mutex_t ac_list_mutex =       PTHREAD_MUTEX_INITIALIZER;   //acadd
static pthread_mutex_t ac_fetch_list_mutex = PTHREAD_MUTEX_INITIALIZER;   //acadd
static pthread_mutex_t fetch_wake_mutex =    PTHREAD_MUTEX_INITIALIZER;
static pthread_cond_t fetch_wake_cond =      PTHREAD_COND_INITIALIZER;

 #define BUF_LEN          512

/*
 * lock access to the ac fetch list
 */
static void
lock_ac_fetch_list(const char *who)             //acadd
{
    pthread_mutex_lock(&ac_fetch_list_mutex);
    DBG(DBG_CONTROL,
      DBG_log("ac fetch list locked by '%s'", who)
    )
}

/*
 * unlock access to the ac fetch list
 */
static void
unlock_ac_fetch_list(const char *who)               //acadd
{
    DBG(DBG_CONTROL,
      DBG_log("ac fetch list unlocked by '%s'", who)
    )
    pthread_mutex_unlock(&ac_fetch_list_mutex);
}

/*
 * lock access to the ac list
 */
void
lock_ac_list(const char *who)             //acadd
{
    pthread_mutex_lock(&ac_list_mutex);
    DBG(DBG_CONTROL,
      DBG_log("ac list locked by '%s'", who)
    )
}

/*
 * unlock access to the ac list
 */
void
unlock_ac_list(const char *who)           //acadd
{
    DBG(DBG_CONTROL,
      DBG_log("ac list unlocked by '%s'", who)
    )
    pthread_mutex_unlock(&ac_list_mutex);
}

/*
 * lock access to the chained cacert list
 */
void
...
```
Das allozierte Memory für die AC-Fetchrequests muss wieder freigegeben werden können
```
...
/*
 *  free the dynamic memory used to store ac fetch requests
 */
static void
free_ac_fetch_request(ac_fetch_req_t *req)                          //acadd
{
     log("freeing ac fetch req: %s", req->dn.ptr);
```

```
}

void
free_ac_fetch_requests(void)                                    //acadd
{
    lock_ac_fetch_list("free_ac_fetch_requests");

    while (ac_fetch_reqs != NULL)
    {
     ac_fetch_req_t *req = ac_fetch_reqs;
     ac_fetch_reqs = req->next;
     free_ac_fetch_request(req);
    }

    unlock_ac_fetch_list("free_ac_fetch_requests");
}

/*
 *  free the dynamic memory used to store fetch requests
 */
static void
...
Damit LDAPserver die nur v3 Protokoll akzeptieren, ebenfalls funktionieren
...
static err_t
fetch_ldap_url(chunk_t url, chunk_t *blob)
{
    LDAPURLDesc *lurl;
    err_t ugh = NULL;
    int rc;

    int ldap_protocol_version = LDAP_VERSION3;

    char *ldap_url = alloc_bytes(url.len + 1, "ldap query");

    sprintf(ldap_url,"%.*s", (int)url.len, url.ptr);

    DBG(DBG_CONTROL,
     DBG_log("Trying LDAP URL '%s'", ldap_url)
    )
...
...
          rc = ldap_simple_bind_s(ldap, NULL, NULL);

          if (rc != LDAP_SUCCESS)
          {
           /* Retry binding with version 3 */
           ldap_set_option(ldap, LDAP_OPT_PROTOCOL_VERSION, &ldap_protocol_version);
           rc = ldap_simple_bind_s(ldap, NULL, NULL);
          }


...
...
     ugh = ldap_err2string(rc);
    }
    return ugh;
}
...
...
/* processes all fetch-requests in the list
 *
 * To do:
 *         - remove the fetch requests immediatly after a successfull fetch
 *         - up-count the trials-field of broken fetch-requests
 *         - remove fetch-request when trials-field reaches a high number (broken)
 */
static void
update_acs(void)
{
    err_t acerr = NULL;
    chunk_t acblob;
    chunk_t acsearchurl;
     ac_fetch_req_t *req;
    char myholder[BUF_LEN];
     char ldapsearchstring[BUF_LEN];
    int myholder_alength;

     lock_ac_fetch_list("update_acs");
     req = ac_fetch_reqs;

     while (req != NULL)
     {
     myholder_alength = dntoa(myholder, BUF_LEN, req->dn);
```

```
            dn_rfc2253(myholder);
            sprintf(ldapsearchstring, "ldap://%s/%s?attributeCertificate",
dap_base_url, myholder);
            DBG(DBG_CONTROL,
            DBG_log("****----**** ac ldapsearchstring = '%s'", ldapsearchstring)
            );

            acsearchurl.len = strlen(ldapsearchstring)+1;
            acsearchurl.ptr = alloc_bytes(acsearchurl.len, "name acsearchurl");
            sprintf(acsearchurl.ptr, "%s", ldapsearchstring);

            acerr = fetch_ldap_url(acsearchurl, &acblob);

            log("fetch_ldap_url returnvalue: %s",acerr);

            if (acerr == NULL) insert_ac(acblob, acsearchurl);

            DBG(DBG_CONTROL,
            DBG_log("acsearchurl err code = '%s'", acblob.ptr)
            );

            pfree(acsearchurl.ptr);

            req = req->next;
      }
      unlock_ac_fetch_list("update_acs");
      free_ac_fetch_requests();
}
...
```

Wird der Thread aufgeweckt wird nur die Fetchliste abgearbeitet, wenn der Timer abläuft wird zuerst die AC-Liste auf abgelaufene AC's überprüft und anschliessend die Fetchliste abgearbeitet

```
...
static void*
fetch_thread(void *arg)
{
    struct timespec wait_interval;

    DBG(DBG_CONTROL,
     DBG_log("fetch thread started")
    )

    pthread_mutex_lock(&fetch_wake_mutex);

    while(1)
    {
     int status;

     wait_interval.tv_nsec = 0;
     wait_interval.tv_sec = time(NULL) + crl_check_interval;

     DBG(DBG_CONTROL,
         DBG_log("next regular crl check in %ld seconds", crl_check_interval)
     )
     status = pthread_cond_timedwait(&fetch_wake_cond, &fetch_wake_mutex
                                     , &wait_interval);
     if (status == ETIMEDOUT)
     {
         DBG(DBG_CONTROL,
          DBG_log(" ");
          DBG_log("*time to check crls")
         )
         check_crls();
         update_ac_list();
     }
     else
     {
         DBG(DBG_CONTROL,
          DBG_log("fetch thread was woken up")
         )
     }
     update_acs();
     fetch_crls();
    }
}
...
```

Falls bereits ein Fetchrequest mit dem gleichen DN in der Liste forhanden ist wird kein neuer dazugefügt und auch nichts mit dem alten unternommen

```
...
void
add_ac_fetch_req(chunk_t dn){
      // check if the dn isn't alredy in the list
      // add the new dn tho the fetch-list
```

```
        ac_fetch_req_t *req;

        lock_ac_fetch_list("add_ac_fetch_req");
        req = ac_fetch_reqs;

        while (req != NULL)
        {
             if (same_dn(dn, req->dn))
             {
                  /* there is already a fetch request */
                  DBG(DBG_CONTROL,
                       DBG_log("ac fetch request already exists")
                  )

                  unlock_ac_fetch_list("add_ac_fetch_req");
                  return;
             }
             req = req->next;
        }
        req = alloc_thing(ac_fetch_req_t, "ac fetch request");
        *req = empty_ac_fetch_req;
        /* note current time */
        req->installed = time(NULL);
        clonetochunk(req->dn, dn.ptr, dn.len, "ac fetch request dn");
        req->next = ac_fetch_reqs;
        ac_fetch_reqs = req;

        DBG(DBG_CONTROL,
             DBG_log("ac fetch request added")
        )
        unlock_ac_fetch_list("add_ac_fetch_req");
}


/*
 *  list all ac fetch requests in the chained list
 */
void
list_ac_fetch_requests(bool utc)
{
    ac_fetch_req_t *req;

    lock_ac_fetch_list("list_ac_fetch_requests");
    req = ac_fetch_reqs;

    if (req != NULL)
    {
     whack_log(RC_COMMENT, " ");
     whack_log(RC_COMMENT, "List of ac fetch requests:");
     whack_log(RC_COMMENT, " ");
    }

    while (req != NULL)
    {
     u_char buf[BUF_LEN];

     whack_log(RC_COMMENT, "%s, trials: %d"
         , timetoa(&req->installed, utc), req->trials);
     dntoa_or_null(buf, BUF_LEN, req->dn, "not set");
     whack_log(RC_COMMENT, "       entityName:  '%s'", buf);
     req = req->next;
    }
    unlock_ac_fetch_list("list_ac_fetch_requests");
}
```

5.4.12 ipsec_doi.c

```
static bool
decode_peer_id(struct msg_digest *md, bool initiator)
{
    struct state *const st = md->st;
...
...
     /* check for certificates */
     decode_cert(md);


// acadd ob schon vorhanden+gueltig bzw fetch_request, nur wenn ldapacserver-
variable vorhanden
// acadd thread wecken

     if (peer.kind == ID_DER_ASN1_DN)
     {
     // i expect peer.name in ASN1 is something like "C=CH, O=ZHW, OU=SNK,
CN=Simpson Bart"
     // are there other IDs which i don't need ??? so may i need another filter ???

        if (strcmp(ldap_base_url, "noldapacserver"))
        {  // server is defined (ipsec.conf)



                if (get_ac(peer.name) != NULL)
                { //nothing to do
                     log("a valid ac for this dn is alredy in the list");
                }
                else
                {
                     add_ac_fetch_req(peer.name);
                     wake_fetch_thread("decode_peer_id");   // wake up the fetch
thread to handle the fetch-request immediatly
                }
        }
        else
        {
           DBG(DBG_CONTROL,
              DBG_log("****----**** ac ldapacserver not defined")
           );
        }
     } // peer.kind


    /* Now that we've decoded the ID payload, let's see if we
     * need to switch connections.
     * We must not switch horses if we initiated:
     * - if the initiation was explicit, we'd be ignoring user's intent
     * - if opportunistic, we'll lose our HOLD info
     */
...
...
    return TRUE;
}
...
...
static stf_status
quick_inI1_outR1_tail(struct verify_oppo_bundle *b
, struct adns_continuation *ac)
{
    struct msg_digest *md = b->md;
    struct state *const p1st = md->st;
    struct connection *c = p1st->st_connection;
    struct payload_digest *const id_pd = md->chain[ISAKMP_NEXT_ID];
    ip_subnet *our_net = &b->my.net
     , *his_net = &b->his.net;
     ac_cert_t *accert;   //acadd

    u_char /* set by START_HASH_PAYLOAD: */
     *r_hashval,     /* where in reply to jam hash value */
     *r_hash_start;  /* from where to start hashing */

    /* Now that we have identities of client subnets, we must look for
     * a suitable connection (our current one only matches for hosts).
     */
...
...
```

```
            p->that.client = *his_net;
            p->that.has_client_wildcard = FALSE;
        }


// try to get the ac
// if the role-group-target-policy matches with the chosen connection continue
// else refuse the connection (cancel quick mode)
// to do:
//   1. check if the connection needs role-group-target-policy matches
//      otherwise we can skip the "acadd"-code
//      or do we need to check for another connection?
//      may we need to change find_client_connection that for...
//   2. check the way of cancelling quick mode
     accert = get_ac(p->that.id.name);
     if (accert != NULL)
     {
          log("has an ac for this peer-name");
//        if rolegrouptarget_matches(p->that.id, accert)
//        {
//             log("ac  rolegrouptarget matches");
//        }
//        else
//        {
//             log("no connection matches to the role,group,target of ac");
//             return STF_FAIL + AUTHENTICATION_FAILED;
//              -> may there's a better way to exit...
//        }


     }
//  else
//  {
//  log("no ac ready to check policy in quick mode");
//   return STF_FAIL + AUTHENTICATION_FAILED;
//    -> may there's a better way to exit...
//  }

...
...
     return STF_OK;
    }
}
...
```

5.4.13 plutomain.c

Damit die URL für den LDAP-Server übergeben werden kann wird analog zu crlcheckinterval verfahren

Damit beim Beenden des Programmes die Listen der AC's und der AC-Fetchrequests gelöscht und dessen Speicher wieder freigegeben wird

```
void
exit_pluto(int status)
{
    reset_globals(); /* needed because we may be called in odd state */
    free_preshared_secrets();
    free_remembered_public_keys();
    delete_every_connection();
    free_fetch_requests();  /* free chain of fetch requests */
    free_cacerts();      /* free chain of X.509 CA certificates */
    free_crls();         /* free chain of X.509 CRLs */
    free_ac_fetch_requests();
    free_ac_certs();
    free_ifaces();
    stop_adns();
    free_md_pool();
    delete_lock();
#ifdef LEAK_DETECTIVE
    report_leaks();
#endif /* LEAK_DETECTIVE */
    close_log();
    exit(status);
}
```

5.4.14 whack.h

Die übergebenen Parameter group, role und target werden ebenfalls in die Message verpackt die an rcv_whack geht. Dabei ist darauf zu achten, dass sie in derselben Reihenfolge abgefüllt und wieder ausgelesen werden müssen.

```
...
```

```
struct whack_end {
    char *id;   /* id string (if any) -- decoded by pluto */
    char *cert; /* path string (if any) -- loaded by pluto  */
    char *ca;   /* distinguished name string (if any) -- parsed by pluto */
    ip_address
     host_addr,
     host_nexthop;
    ip_subnet client;

    bool key_from_DNS_on_demand;
    bool has_client;
    bool has_client_wildcard;
    char *updown;    /* string */
    u_int16_t host_port;  /* host order */
    u_int16_t port;       /* host order */
    u_int8_t protocol;
    char *group;     /* group string (if any) -- decoded by pluto */
    char *role;      /* role string (if any) -- decoded by pluto */
    char *target;    /* target string (if any) -- decoded by pluto */
 };

struct whack_message {
    unsigned int magic;

    /* name is used in connection and initiate */
    size_t name_len; /* string 1 */
    char *name;
...
...
    /* for WHACK_SHUTDOWN */
    bool whack_shutdown;

    /* space for strings (hope there is enough room):
     * Note that pointers don't travel on wire.
     *  1 connection name [name_len]
     *  2 left's name [left.host.name.len]
     *  3 left's cert
     *  4 left's ca
     *  5 left's updown
     *  6 left's group
     *  7 left's role
     *  8 left's target
     *  9 right's name [left.host.name.len]
     * 10 right's cert
     * 11 right's ca
     * 12 right's updown
     * 13 right's group
     * 14 right's role
     * 15 right's target
     * 16 keyid
     * plus keyval (limit: 8K bits + overhead), a chunk.
     */
    size_t str_size;
    char string[2048];
};
...
Damit verglichen werden kann ob alle Listen ausgegeben werden sollen oder nur
diejenige der AC's resp. der AC-Fetchrequests
...
/* options of whack --list*** command */

#define LIST_NONE    0x00 /* don't list anything */
#define LIST_PUBKEYS 0x01 /* list all public keys */
#define LIST_CERTS   0x02 /* list all host/user certs */
#define LIST_CACERTS 0x04 /* list all ca certs */
#define LIST_CRLS    0x08 /* list all crls */
#define LIST_ACS     0x16 /* list all acs */

#define LIST_ALL     LRANGES(LIST_PUBKEYS, LIST_CRLS)  /* all list options */
...
```

5.4.15 whack.c

Analog zu den Flags OPT_LISTALL rsp. END_UPDOWN benötigen wir Flags für listacs und die Parameter group, role und target. Dasselbe gilt auch für die Behandlung der übergebenen Werte.

```
...
    OPT_LISTCRLS,
    OPT_LISTACS,
    OPT_LISTALL,
...
...
    END_UPDOWN,
    END_GROUP,
    END_ROLE,
    END_TARGET,
...
...
    { "listacs", no_argument, NULL, OPT_LISTACS + OO },
    { "listall", no_argument, NULL, OPT_LISTALL + OO },
...
...
    { "updown", required_argument, NULL, END_UPDOWN + OO },
    { "group", required_argument, NULL, END_GROUP + OO },
    { "role", required_argument, NULL, END_ROLE + OO },
    { "target", required_argument, NULL, END_TARGET + OO },
...
...
    e->updown = NULL;
    e->host_port = IKE_UDP_PORT;
    e->group = NULL;
    e->role = NULL;
    e->target = NULL;
...
...
     case OPT_LISTCERTS:  /* --listcerts */
     case OPT_LISTCACERTS:     /* --listcacerts */
     case OPT_LISTCRLS:   /* --listcrls */
     case OPT_LISTACS:    /* --listacs */
         msg.whack_list |= LELEM(c-OPT_LISTPUBKEYS);
         continue;
...
...
         msg.right.updown = optarg;
         continue;

     case END_GROUP:           /* --group <group name> */
         msg.right.group = optarg;   /* decoded by Pluto */
         continue;

     case END_ROLE:            /* --role <role name> */
         msg.right.role = optarg;    /* decoded by Pluto */
         continue;

     case END_TARGET:          /* --target <target name> */
         msg.right.target = optarg;  /* decoded by Pluto */
         continue;


     case CD_TO:          /* --to */
         /* process right end, move it to left, reset it */
...
```

Die Message, die an rcv_whack gesendet wird, wird zusammengesetzt, Reihenfolge muss beachtet werden

```
...
   /* pack strings for inclusion in message */
    next_str = msg.string;
    str_roof = &msg.string[sizeof(msg.string)];

    if (!pack_str(&msg.name)         /* string  1 */
    || !pack_str(&msg.left.id)       /* string  2 */
    || !pack_str(&msg.left.cert)     /* string  3 */
    || !pack_str(&msg.left.ca)       /* string  4 */
    || !pack_str(&msg.left.updown)   /* string  5 */
    || !pack_str(&msg.left.group)    /* string  6 */
    || !pack_str(&msg.left.role)     /* string  7 */
    || !pack_str(&msg.left.target)   /* string  8 */
    || !pack_str(&msg.right.id)      /* string  9 */
    || !pack_str(&msg.right.cert)    /* string 10 */
    || !pack_str(&msg.right.ca)      /* string 11 */
    || !pack_str(&msg.right.updown)  /* string 12 */
    || !pack_str(&msg.right.group)   /* string 13 */
```

```
    || !pack_str(&msg.keyid)         /* string 16 */
    || str_roof - next_str < (ptrdiff_t)msg.keyval.len)    /* chunk (sort of string
5) */
...
```

5.4.16 rcv_whack.c

Die Message von whack wird entpackt
Reihenfolge muss beachtet werden

```
...
     }
     else if (!unpack_str(&msg.name)      /* string  1 */
     || !unpack_str(&msg.left.id)         /* string  2 */
     || !unpack_str(&msg.left.cert)       /* string  3 */
     || !unpack_str(&msg.left.ca)         /* string  4 */
     || !unpack_str(&msg.left.updown)     /* string  5 */
     || !unpack_str(&msg.left.group)      /* string  6 */
     || !unpack_str(&msg.left.role)       /* string  7 */
     || !unpack_str(&msg.left.target)     /* string  8 */
     || !unpack_str(&msg.right.id)        /* string  9 */
     || !unpack_str(&msg.right.cert)      /* string 10 */
     || !unpack_str(&msg.right.ca)        /* string 11 */
     || !unpack_str(&msg.right.updown)    /* string 12 */
     || !unpack_str(&msg.right.group)     /* string 13 */
     || !unpack_str(&msg.right.role)      /* string 14 */
     || !unpack_str(&msg.right.target)    /* string 15 */
     || !unpack_str(&msg.keyid)           /* string 16 */
     || str_roof - next_str != (ptrdiff_t)msg.keyval.len) /* check chunk */
...
```

Falls listacs übergeben wurde werden die Funktionen zur Auflistung der Listen der AC's und der AC-Fetchrequests aufgerufen

```
...
    if (msg.whack_list & LIST_CERTS)
    {
      list_certs(msg.whack_utc);
    }

    if (msg.whack_list & LIST_CACERTS)
    {
      list_cacerts(msg.whack_utc);
    }

    if (msg.whack_list & LIST_CRLS)
    {
      list_crls(msg.whack_utc, strict_crl_policy);
      list_fetch_requests(msg.whack_utc);
    }

    if (msg.whack_list & LIST_ACS)
    {
      list_acs(msg.whack_utc);
      list_ac_fetch_requests(msg.whack_utc);
    }

    if (msg.whack_key)
...
```

5.5 LDAP

LDAP wurde geschaffen, um vereinfacht auf x.500-Directories zuzugreifen, also als eine Art Zwischenschicht. Mittlerweile gibt es aber auch eigenständige LDAP-Datenbank-Server, als Beispiel sei hier eine Open-Source-Implementation Namens slapd (Stand-alone LDAP Daemon) erwähnt.

5.5.1 LDAP-Utilities

Das OpenLDAP-Projekt stellt neben dem eigentlichen Server slapd auch Command-Line-Tools zur Verfügung.
Z.B. unter Debian als "ldap-utils"-Paket zu installieren. Siehe auch www.openldap.org
Es empfiehlt sich damit kurz einige Testabfragen auf dem LDAP-Server vorzunehmen, um die korrekte Funktion zu prüfen. Diese verwenden die gleichen Libraries wie FreeS/WAN.

5.5.2 LDAP-Voraussetzung FreeS/WAN

FreeS/WAN benötigt LDAP-Komponenten erst, wenn der x.509-Patch eingespielt wird bzw. die CRL via LDAP abgeholt wird.
Es werden die Header-Files sowie die "OpenLDAP libraries" benötigt.
Z.B. unter Debian als "libldap2-dev" bzw. "libldap2" zu installieren.

5.5.3 LDAP-Server installieren

Ein LDAP-Server ist nicht zwingend extra zu installieren, ev. ist bereits einer auf einem Unix- oder Windows-Server im Netzwerk installiert.
FreeS/WAN sendet ohne unsere Erweiterung den LDAP-Bind-Request nur als LDAP-Version 2, was mit Servern die nur Version 3 unsterstützen zu Problemen führt. Wir haben den Code so modifiziert, dass zuerst mit der LDAP-Version 2 und im Fehlerfall mit Version 3 versucht wird.

Achtung: "anonymous" muss read(only) Zugriff haben, da FreeS/WAN sich nicht anmeldet

- LDAP-Paket der jeweiligen Distribution bzw. Code von www.openldap.org installieren, wir verwendeten "slapd"
  Optionen in slapd.conf (meist /etc/ldap/), damit diese Example-Files funktionieren

```
include         /etc/ldap/schema/core.schema
include         /etc/ldap/schema/freeswan_attr_cert.schema
...
...
#######################################################################
# ldbm database definitions
#######################################################################

database        ldbm
suffix          "dc=my-domain,dc=com"
rootdn          "cn=Manager,dc=my-domain,dc=com"
# only if you d'like to save the password in plainText for testing!
rootpw          chooseapassword
```

- LDAP-Schema-Erweiterung für AC installieren
  wir verwendeten als Beta-Tester das File "freeswan_attr_cert.schema", siehe u.U. "PAKI2 Sna 03/5 - Management von Attributszertifikaten (Hollenstein/Morina)"

5.5.4 LDAP-Eintrag hinzufügen

```
>  ldapadd -x -W -D "cn=Manager,dc=my-domain,dc=com" -f add_pmuster.ldif
```

example file: add_pmuster.ldif

```
dn: cn=Muster Peter,dc=my-domain,dc=com
sn: Peter
cn: Muster Peter
userCertificate;binary:<file:///home/user1/myUC.der
telephoneNumber: 12345679
objectClass: strongAuthenticationUser
objectClass: top
objectClass: person
objectClass: attributeCertificateUser
objectClass: certAttributes
group: Research
target: DB
attributeCertificate;binary:<file:///home/user1/myAC.der
```

5.5.5 LDAP-Eintrag ändern

```
>  ldapmodify -x -W -D "cn=Manager,dc=my-domain,dc=com" -f modify_pmuster.ldif
```

example file: add_pmuster.ldif

```
dn: cn=Muster Peter,dc=my-domain,dc=com
telephoneNumber: 75845215
```

--> nur bereits vorhandene Attribute können geändert werden, ansonsten ldapadd verwenden

5.5.6 LDAP-Search

- Attributs-Zertifikat eines bestimmten DN (nur Objektklasse attributeCertificate ausgeben)

  ```
  >  ldapsearch -t -h dskt6805 -x -b "cn=Krauer Christoph,dc=my-domain,dc=com"
     attributeCertificate

     --> Pfad des Binary-Teils (AC) wird angezeigt
     attributeCertificate;binary:< file:///tmp/ldapsearch-attributeCertificate_xy
  ```

- Filtern, alle Objektklassen ausgeben die mit cn=Kr beginnen

  ```
  >  ldapsearch -h dskt6805 -x -b "dc=my-domain,dc=com" "cn=Kr*"
  ```

- Filtern, nur Objektklasse cn+sn ausgeben die mit cn=Kr beginnen

  ```
  >  ldapsearch -h dskt6805 -x -b "dc=my-domain,dc=com" "cn=Kr*" cn sn
  ```

5.5.7 Anzeigen mittels openac-parse

```
>  openac-parse -p plainText -o noout /tmp/ldapsearch-attributeCertificate_xy
```

6 Ausblick

- "holderIssuer" durch "entityName" ersetzen, um an den korrekten DN zu kommen
- update_acs ist noch abzuändern, damit der Fetchrequest bei nicht erfolgreicher LDAP-Abfrage in der AC-Fetchliste verbleibt
- Die Signatur der AC's ist zu prüfen, dazu muss das Zertifikat der Authorization Authority (AA) vorhanden sein. Weiter muss der Bezug zum Userzertifikat verifiziert werden, dazu müssen die Userzertifikate ganz oder teilweise gespeichert werden.
- Bei der Auswahl der Connection (quick_inI1_outR1_tail bzw. find_client_connection) müssen die neuen Parameter role/group/target ebenfalls berücksichtigt werden
- Die alternative Eingabe der ID in ipsec.conf (im DER_ASN1_DN-Format) mit Slashes, muss mutmasslich umgewandelt werden, damit jeder LDAP-Server die Abfrage versteht. Dies wäre am einfachsten in der Funktion "dn_rfc2253" zu implementieren.
  ID bsp: /C=CH/O=strongSec GmbH/CN=wroclaw.strongsec.com
  Von Zertifikat anschauen: `openssl x509 -in usercert_name.pem -noout -subject`
- Allenfalls eine Erweiterung um mehrere Parameterwerte mitgeben zu können (roles/groups/targets und ldapbaseurls).
  Bei ldapbaseurls müssten die Server dann inkl. des base-DN angegeben werden, da sonst unklar wäre welche AC's auf welchem Server zu finden sind.

7 Installation

- FreeS/WAN gemäss Anleitung auf www.freeswan.org/
- x.509-patch gemäss Anleitung auf www.strongsec.com/freeswan/
- Zur Kompilation müssen die Developerfiles von OpenLDAP installiert sein (z.B. unter Debian "libldap2-dev"-Package)
- _confread arbeitet mit awk. Allerdings mit Befehlen die nur von gawk unterstützt werden, mawk führt zu Fehlern (ggf. mawk durch gawk ersetzen)

8 Anhang

8.1 Lizenzbestimmungen

Das FreeS/WAN-Projekt unterliegt der GNU GENERAL PUBLIC LICENSE (GPL).

http://www.gnu.org/licenses/gpl.txt

8.2 Quellen

- C/C++, Galileo Verlag, ISBN 3-934358-03-9
- PA1 Sna 03/6 - Smartcard-Unterstützung für Linux FreeS/WAN (Gysin/Zwahlen)
- PAKI2 Sna 02/4 - Attributszertifikate (Gallizzi/Seiler) und deren Tools openac/openac-parse www.strongsec.com/openac/
- ZHW Fach SNK "VPN and Security Policies" (Kapitel 7-9) und "Lab 5 - Virtual Private Networks under Linux" www-t.zhwin.ch/it/snk/
- Main- und Quick-Mode Details http://www.tdt.de/switched/akt/ipsec_teil3.pdf
- LDAP RFC-Übersicht http://www.networksorcery.com/enp/protocol/ldap.htm
- PAKI2 Sna 03/5 - Management von Attributszertifikaten (Hollenstein/Morina)
- www.openssl.org/
- www.openldap.org/
- rfc2408 --> IPsec Main- und Quick-Mode

8.3 CDROM

- CD: Source + Doku