strongManager

strongSwan Web Management Tool

Diplomarbeit

Andreas Eigenmann
Joël Stillhart

strongManager: strongSwan Web Management Tool: Diplomarbeit

von Andreas Eigenmann und Joël Stillhart
Betreuer: Prof. Dr. Andreas Steffen
Betreuer: Dipl. Ing. Martin Willi
Experte: Dr. Ralf Hauser

Veröffentlicht 15.12.2006

Inhaltsverzeichnis

Aufgabenstellung

Die Aufgabenstellung befindet sich auf der nächsten Seite.

Management Summary

Inhaltsverzeichnis

1. Einleitung

strongSwan ist eine IPsec-basierte VPN Lösung für Linux, die in Form eines Open Source Projekts durch das Institut für Internet Technologien und Anwendungen vorangetrieben wird. Die VPN Software kann auf Gateways mit mehreren hundert aktiven Tunnelverbindungen eingesetzt werden. Im Rahmen dieser Diplomarbeit wurde die Web-basierte Managementkonsole strongManager für strongSwan entwickelt, welche das übersichtliche und komfortable Management von mehreren VPN Gateways ermöglicht und dabei leicht zu bedienen ist.

2. Ausgangslage

2.1. Warum machen wir das Projekt?

Das Thema zur Erstellung eines Managementtools für strongSwan wurde gewählt, weil es sehr facettenreich ist. Nebst der Konzeption eines Management Protokolls sollte eine Webapplikation implementiert werden, durch welche das Protokoll verifiziert werden konnte. Es ist sehr interessant an einem grösseren Projekt mitzuwirken und wir glauben an eine grosse Zukunft der strongSwan Software.

2.2. Ziele

Die Ziele waren die Erstellung eines Konzepts für eine Kommunikationsschnittstelle, sowie einer Managementkonsole in Form einer Web-Applikation. Durch die Schnittstelle soll die Kommunikation zwischen der Managementkonsole und dem VPN Gateway ermöglicht wer-

den. Diese muss dabei flexibel, programmiersprachenunabhängig und einfach erweiterbar sein. Durch die Schnittstelle sollen Statusinformationen bezogen, VPN Verbindungen konfiguriert und gesteuert werden können. Um im Falle von unerwarteten Ereignissen auf Seiten des VPN Gateways reagieren zu können, soll in der Kommunikationsschnittstelle eine geeignete Komponente zur Notifikation vorhanden sein. Damit die Kommunikationsdaten vor unberechtigten Eingriffen geschützt sind müssen geeignete Massnahmen getroffen werden, welche keine Kompromisse zulassen. Aus Zeitgründen beschränkt sich die Implementation der Managementkonsole auf das Abfragen und Darstellen von Status bezogenen Informationen.

3. Vorgehen

3.1. Umfang

Im Rahmen dieser Diplomarbeit wurde das XML-basierende Strong Management Protocol (SMP) und eine dazugehörige Managementkonsole in Form einer Web-Applikation entwickelt. Der Umfang der erstellten Komponenten verteilt sich auf die drei Komponenten Strong Management Protocol, Managementkonsole und Emulator.

3.1.1. Strong Management Protocol

Das definierte Strong Management Protocol bietet die Möglichkeit für das Abfragen und Übertragen von Statusinformationen jeglicher Art. Dazu gehören Informationen über die bestehende Konfiguration von Verbindungen und Sicherheitsrichtlinien, die Performance, die aufgebauten Verbindungen und die dafür ausgehandelten Parameter. Des Weiteren unterstützt SMP die Signalisation, wodurch auf unerwartete Ereignisse auf Seiten eines VPN Gateways reagiert werden kann. Nebst dem Abfragen von Informationen können durch SMP auch Verbindungen und Sicherheitseinstellungen konfiguriert werden. Es besteht ausserdem die Möglichkeit VPN Gateways zu steuern. Dies umfasst zur Zeit die Steuerung von VPN Verbindungen und deren Sicherheitsbeziehungen. Damit die zu übertragenden Daten von unberechtigten Personen nicht eingesehen werden können, wurden XML-basierte Signatur und Verschlüsselung in das erstellte Konzept miteinbezogen. In Zukunft kann das Protokoll auch für andere Applikationen wie z.B. eine GUI-Applikation genutzt werden.

3.1.2. Managementkonsole

Die Managementkonsole besteht aus der Kommunikationsschnittstelle für SMP und einer Web-Applikation. In ihr wurde der vollständige Support für das Abfragen von Statusinformationen, wie dieser in SMP spezifiziert wurde, implementiert. Damit die Webapplikation direkt verwendet werden kann, beinhaltet die Managementkonsole einen integrierten Webserver, welcher nach der Installation des Managementtools sofort verfügbar ist.

3.1.3. Emulator

Die Kommunikationsschnittstelle auf Seiten des VPN Gateways existiert noch nicht. Damit das Managementtool getestet werden konnte, wurde ein Emulator erstellt, welcher den VPN Gateway simuliert. Zur Zeit erlaubt der Emulator die lokale, sowie die dezentrale Kommunikation. Für die Entwicklung eventueller weiterer Management-Applikationen und Erweiterungen kann der Emulator einfach wiederverwendet und erweitert werden.

3.2. Teilschritte

Das ganze Projekt verlief über acht Wochen. In den ersten drei Wochen wurde das Strong Management Protocol wie oben beschrieben spezifiziert und dokumentiert. Anschliessend konnten in den Wochen 4 - 7 die Kommunikationsschnittstelle, der Emulator und die Webapplikation implementiert werden. Die Entwicklung erfolgte dabei in inkrementeller und iterativer Vorgehensweise mit der Implementierung über alle Schichten (Model-View-Controller) in jeder Iteration. Die Dokumentation, welche parallel zur Implementation geführt wurde, konnte in der letzten Woche noch ergänzt, verbessert und überarbeitet werden.

3.3. Involvierte Personen

Bei der Durchführung und den Zwischenprüfungen waren die Betreuer Prof. Dr. Andreas Steffen und Dipl. Ing. Inf. Martin Willi und wir als Entwickler beteiligt. Da das Produkt für Prof. Dr. Andreas Steffen entwickelt wurde, mussten keine weiteren Personen mit einbezogen werden.

4. Ergebnisse

4.1. Nutzen des Projektes

Durch das erstellte Managementtool können mehrere VPN Gateways auf einfache und effiziente Weise von einem zentralen Punkt aus verwaltet werden. Da sich dadurch das explizite einloggen auf einem VPN Gateway erübrigt, können Informationen ohne viel Aufwand gesammelt werden. Durch dieses Tool kann sich strongSwan einen Vorteil gegenüber konkurrierenden Lösungen schaffen.

4.2. Eigene und Verwendete Komponenten

Grundsätzlich wurden alle Komponenten selber erstellt. Beim erstellen der Webapplikation wurden auf bekannte, bestehende Web-Bibliotheken zurückgegriffen.

4.3. Erreichte Ziele

Durch dieses Projekt und dem damit entstandenen strongManager und dem Strong Management Protocol konnten die oben aufgeführten Ziele erreicht werden. Die Qualität des Produktes wurde mit geeigneten Massnahmen sicher gestellt.

5. Ausblick

5.1. Was ist weiter zu tun

Da in der Managementkonsole nur die auf den Status bezogenen Informationen implementiert wurden, muss diese noch um die fehlenden Komponenten für Konfiguration, Steuerung und Notifikation erweitert werden. In einem weiteren Schritt ist die Implementierung der Sicherheitsmassnahmen (XML-Signatur und Verschlüsselung) noch durchzuführen, damit eine Kommunikation auch über unsichere Kanäle möglich wird. Dem Strong Management Protocol sel-

ber fehlt es noch an einer kompletten Spezifikation für alle Steuerungs- sowie Notifikationsmöglichkeiten und eventuelle andere asynchrone Messages.

Einleitung

Inhaltsverzeichnis

1. Ziele und erstellte Komponenten

strongSwan ist eine IPsec-basierte VPN Lösung für Linux, die in der Form eines Open Source Projekts durch das Institut für Internet Technologien und Anwendungen vorangetrieben wird. Eingesetzt werden kann die VPN Software auf Gateways mit mehreren hundert aktiven Tunnelverbindungen. Die Ziele dieser Diplomarbeit waren die Erstellung eines Konzepts für eine Kommunikationsschnittstelle, sowie einer Managementkonsole in Form einer Web-Applikation. Durch die zu erstellenden Komponenten sollen mehrere strongSwan VPN Gateways auf einfache Weise verwaltet werden können. Des Weiteren soll die Kommunikationsschnittstelle das Beziehen von Statusinformationen, die Konfiguration und Steuerung von VPN Verbindungen ermöglichen. Um im Fall von unerwarteten Ereignissen auf Seiten des VPN Gateways reagieren zu können, muss die Schnittstelle eine geignete Komponente enthalten um asynchrone Meldungen zu verarbeiten. Aus Zeitgründen beschränkt sich aber die Implementation der Managementkonsole auf das Abfragen und Darstellen von Status bezogenen Informationen.

Im Rahmen dieses Projekts wurde die Web-basierte Managementkonsole strongManager für strongSwan entwickelt, welche das übersichtliche und komfortable Management von mehreren VPN Gateways ermöglicht und dabei leicht zu bedienen ist. Für die Kommunikation zwischen Managementkonsole und VPN Gateway wurde das XML basierte Strong Management Protocol (SMP) entwickelt, welches die Abfrage von Informationen, die Konfiguration sowie die Steuerung von Verbindungen ermöglicht. SMP ist einfach erweiterbar und unabhängig von einer Programmiersprache und kann auch von anderen Applikationen wie z.B. einer GUI-Applikation genutzt werden. SMP funktioniert nach dem Request/Response-Prinzip, erlaubt aber auch asynchrone Meldungen zur Notifikation. Zur Validierung der Nachrichten wird ein Relax NG Schema (XML Schema Standard) verwendet, welches einfach verständlich und leicht erweiterbar ist. Zur Gewährleistung der Kommunikationssicherheit wurde SMP um XML Signature und XML Encryption ergänzt. Um die Web-Applikation zu testen, wurde ein Emulator entwickelt, der einen strongSwan Gateway emuliert. Die Kommunikation mit dem Emulator erfolgt dabei wahlweise über TCP oder Unix Socket. Des Weitern ist es möglich die erstellte Webapplikation zu Testzwecken direkt mit dem integrierten Webserver zu betreiben.

Im Rahmen dieser Dokumentation werden die erstellten Komponenten und die Arbeitsschritte, welche zu diesen geführt haben, im Detail beschrieben.

2. Der Aufbau dieser Dokumentation

Der Aufbau dieser Dokumentation gliedert sich in die folgenden Kapitel.

2.1. Grundlagen

Um dieses Projekt bewältigen zu können, musste IPsec, IKEv2 und strongSwan studiert werden. Das Kapitel Grundlagen gibt eine Einführung in diese Technologien, so dass die Inhalte der restlichen Kapitel verständlich sein sollten.

2.2. Analyse für Schnittstelle

In diesem Kapitel werden die Punkte behandelt, welche zur Definition des Strongmanagemenprotokolls geführt haben. Dazu gehören die Anforderungen, die Evaluation der Basistechnologie für die Kommunikation, das Sicherheitskonzept und die Analyse der derzeitigen Statusausgabe von strongSwan.

2.3. Strong Management Protocol

Das Kapitel Strong Management Protocol enthält die Spezifikation des Strong Management Protokoll (SMP). Es beinhaltet Informationen über die Architektur, die Nachrichtenstruktur und die XML Elemente von SMP.

2.4. Anforderungen strongManager

Wie die Anforderungen der Managemkonsole strongManager aussehen, werden im Kapitel Anforderungen strongManager in Form von User Stories beschrieben.

2.5. Design strongManager

In diesem Kapitel wird das Design der Managementkonsole erläutert. Das Kapitel beschreibt die Komponenten für die Webapplikation, die Kommunikationsschnittstelle und das Model, welches das SMP Protokoll abbildet.

2.6. Testen

Das Kapitel Testen befasst sich mit dem Testen der Managementkonsole strongManager und mit dem Design des Emulators, welcher zum Testen und Demonstrieren benötigt wurde.

2.7. Schlussfolgerungen

Im Kapitel Schlussfolgerungen wird nochmals verdeutlicht was erreicht wurde und wie wir die Resultate bewerten. Des Weiteren geht das Kapitel auf Arbeiten für die Zukunft ein.

2.8. Anhang

Im Anhang sind die Persönlichen Berichte, Code Statistiken, eine Installationsanleitung sowie das Glossar und Literaturverzeichniss enthalten.

3. Typografische Konventionen

Kursivschrift	Kennzeichnet XML Element- und Klassennamen, sowie Pythonmodule.
`Nichproportionalschrift`	Kennzeichnet Dateinamen, Bestandteile aus Sourcecode und Konsolenausgaben.
Normalschrift fett	Kennzeichnet Konsolenbefehle und wichtige Schlagwörter.
„Schrift mit Anführungszeichen“	Kennzeichnet spezielle Komponenten.

Wichtiger Hinweis

Gibt ergänzenden Hinweis mit mittlerer Priorität.

Achtungshinweis

Gibt ergänzenden Achtungshinweis mit hoher Priorität.

Bemerkung

Gibt Achtungshinweis mit geringer Priorität.

Kapitel 1. Grundlagen

Inhaltsverzeichnis

1.1. Einleitung

Ein grundlegendes Verständnis von IPsec, IKE und strongSwan gehört zum Bestandteil dieser Diplomarbeit. Um einen Einblick in diese Technologien zu erhalten, wird hier eine kurze Einführung gegeben. Für weitere Informationen können die entsprechenden RFC's, welche im Literaturverzeichnis aufgelistet sind, zu Hande gezogen werden.

1.2. IPsec

IPsec wurde entwickelt um kryptografisch basierte Sicherheit für IPv4 und IPv6 zur Verfügung zu stellen. Es beinhaltet folgende Sicherheitsdienste:

- Datenverschlüsselung
- Zugangskontrolle
- Datenintegrität
- Schutz gegen Replay Attacken
- Herkunft der Daten sicherstellen

Die Sicherheitsdienste werden durch die IPsec Protokolle AH (Authentication Header) und ESP (Encapsulating Security Payload), sowie die Verwendung eines Schlüsselverwaltungssystems (z.B. IKE) gewährleistet. Durch IPsec kann jeder Verkehr der über der Netzwerkschicht liegt geschützt werden. Es gilt als Basistechnologie für virtuelle private Netzwerke (VPN's) und besitzt zwei Modi in welchen es betrieben werden kann: den Transport Modus und den Tunnel Modus.

1.2.1. AH (Authentication Header)

Der Authentication Header stellt die Integrität und die Herkunft der Daten sicher und bietet (optional) Schutz gegen Replay Attacken. In AH werden alle invarianten Felder eines IP Pakets gesichert, in dem ein Hash darüber gebildet wird.

1.2.2. ESP (Encapsulation Security Payload)

ESP stellt nebst den Diensten des AH zusätzlich Datenverschlüsselung für die Nutzdaten zur Verfügung. Im Unterschied zu AH wird bei ESP die Integrität und Herkunft des IP Headers nicht gewährleistet.

1.2.3. Transport Modus

Im Transportmodus wird der bestehende IP Header um IPsec Informationen ergänzt, in dem diese zwischen dem IP Header und den Nutzdaten eingefügt werden. Der Transport Modus eignet sich für die Sicherung durch IPsec auf Host zu Host oder Host zu Gateway Basis.

1.2.4. Tunnel Modus

Beim Tunnel Modus wird das ursprüngliche IP Paket in einem neuen IP Paket gekapselt. Durch diese beiden IP Header ist ein Tunnel auf Gateway-Gateway Basis möglich. Es können also zwei Netzwerke über einen sicheren Tunnel verbunden werden.

1.2.5. Security Associations (SA's)

Als Security Associations (SA's) werden Verträge zwischen Endpunkten bezeichnet, welche den Umfang der Sicherheit einer Beziehung festlegen. In einer SA werden unter anderem folgende Punkte ausgehandelt:

- Algorithmus und Schlüssel für AH
- Algorithmus und Schlüssel für die ESP Verschlüsselung
- Algorithmus und Schlüssel für die ESP Authentisierung
- verwendeter Modus (Tunnel/Transport Modus)

Eine Beziehung zwischen zwei Endpunkten kann aus mehreren SA's bestehen (SA Bundle). Für die Anzahl der vorhandenen SA's gelten folgende Richtlinien:

- Wenn AH und ESP gemeinsam verwendet werden, wird für jeden Typ mindestens eine SA ausgehandelt.
- Für jede Verbindungsrichtung wird eine SA benötigt. Für eine Beziehung ist also mindestens ein SA Pair vorhanden (je eine SA für in- und outbound Traffic).

Eine SA wird eindeutig identifiziert über ein Tripel, das aus Security Parameter Index (SPI), IP Destination Address und den Security Protocol Identifier (AH oder ESP) besteht. Security Associations werden in der SAD (Security Association Database) im Kernel abgelegt und können über deren Identifizierung gefunden werden.

1.2.6. Security Policies

Security Policies, welche in der SPD abgelegt werden, beschreiben den Sicherheitsservice, welcher einem IP Paket zur Verfügung steht. Es sind die folgenden drei Verarbeitungsabläufe möglich:

- `discard` - IP Paket darf Host oder Security Gateway nicht verlassen.
- `bypass IPsec` - IP Paket darf Host oder Security Gateway verlassen. Es wird kein IPsec darauf angewendet.
- `apply IPsec` - Auf ein IP Paket wird IPsec angewendet (Verweis zu SA oder SA Bundle in Security Policy vorhanden).

Wie man erkennen kann, entscheidet IPsec für jedes IP Paket was mit diesem geschehen soll. Die Zuweisung zu einer Policy erfolgt dabei über Selektoren. Die Art des Selektors wird in der Security Policy selbst definiert. Unter anderem sind folgende Selektoren vorhanden:

- Ziel IP Adresse
- Quellen IP Adresse
- Name - User ID (DNS, X.500 DN), System Name (DNS, X.500 DN)
- Transport Layer Protokoll
- Quell- und Zielport

Durch diesen Entscheidungsprozess kann die Granularität der angebotenen Sicherheit festgelegt werden. Es ist z.B. möglich sämmtlichen IP Verkehr zwischen zwei Endpunkten über eine SA abzudecken, oder aber nur TCP Verkehr auf dem Port 80 mit IPSec zu behandeln. Security Policies weisen eine Ähnlichkeit mit **iptables** Firewall Regeln auf. Auch dort existieren eine Art Selektoren, welche entscheiden was mit dem Traffic passiert.

Die Security Policies in der SPD sind sortiert und weisen eine bestimmte Reihenfolge auf. Bei einem eintreffenden IP Paket wird immer diejenige Security Policy angewandt, dessen Selektor zuerst matched.

1.3. IKEv2

Ein wichtiger Punkt in IPsec stellt der Austausch der Schlüssel dar, welche für die Sicherheitsdienste benötigt werden. Die Art des Austausches entscheidet dabei über die Sicherheit der angebotenen Dienste. In IPsec kann der Austausch der Schlüssel manuell oder automatisch erfolgen. Bei der manuellen Schlüsselverwaltung werden diese fest in den Endpunkten konfiguriert. Im Gegensatz dazu werden bei der automatischen Schlüsselverwaltung die Schlüssel durch ein Protokoll ausgehandelt. Die automatische Schlüsselverwaltung stellt die sichere Methode dar und sollte deshalb bevorzugt verwendet werden, denn von Hand erzeugt Schlüssel können in der Regel einfacher erraten werden. In diesem Abschnitt wird IKE in der Version 2 kurz vorgestellt.

IKEv2 ist der Nachfolger von IKEv1. Dieser ist weniger komplex, da statt der neun IKE Meldungen nur noch vier für einen ersten Verbindungsaufbau benötigt werden. IKEv2 unterstützt des Weiteren eine flexiblere Authentisierung durch EAP und ermöglicht ein erweitertes Aushandeln von Policies. Die Nachrichten für das Aushandeln einer IKE SA treten im Paar auf als `IKE_SA_INIT` und `IKE_AUTH` . Durch `IKE_SA_INIT` wird ein Diffie Hellman Exchange durchgeführt, kryptografische Algorithmen ausgehandelt und Nonces ausgetauscht. Durch den

Diffie Hellman Exchange wird dabei sichergestellt das `IKE_AUTH` verschlüsselt übertragen werden kann. Die kryptografischen Algorithmen selbst werden für die Sicherstellung der Herkunft der Daten und für die Verschlüsselung der Nachrichten, durch welche weitere Child SA's ausgehandelt werden, benötigt. Durch das zweite Meldungspaar (`IKE_AUTH`) werden die vorher übertragenen Daten authentifiziert, die Identitäten und Zertifikate ausgetauscht, sowie die erste Child SA erstellt. Nach vier Meldungen existiert nun eine IKE SA und eine Child SA. Es besteht also ein Tunnel.

1.4. strongSwan

Die IKEv2 Unterstützung in strongSwan ist in Entstehung und wird im Daemon Charon von strongSwan 4.0 implementiert. Die Konfiguration von Charon verläuft über die Datei `ipsec.conf` . Als für die Managementschnittstelle wichtige Komponenten von strongSwan haben sich die Implementierung der Policy , Connection , IKE SA und Child SA heraus kristallisiert. Dabei ist zu beachten, dass eine Policy in strongSwan nur beschränkt das gleiche repräsentiert wie eine Policy in der SPD im Kernel. Unter IKE SA und Child SA versteht man die entsprechenden Komponenten aus dem vorherigen Abschnitt, wobei die Connection und Policy als Vorlage für diese dienen. Eine Connection beispielsweise legt fest, zwischen welchen beiden Kommunikationspartner eine IKE SA aufgebaut werden kann und welche kryptografischen Algorithmen für die Aushandlung von Child SA's zur Verfügung stehen. Eine Policy legt für eine Child SA fest, welche Algorithmen für die Verschlüsselung des Traffics eines Tunnels erlaubt sind, was für Algorithmen für die Sicherung der Herkunft der Daten verwendet werden können, welche Selektoren für den Traffic vorhanden sind, beschreibt die ID's der Kommunikationspartner und noch einiges mehr. Da die vier Komponenten (Policy, Connection, IKE SA und Child SA) die grundsätzlich wichtigen Informationen für die Konfiguration, das Management und die Auswertung beinhalten, liegt es nahe diese auf die Managementschnittstelle abzubilden. Mehr Informationen hierzu können dem Kapitel Analyse für Schnittstelle entnommen werden.

Kapitel 2. Analyse für Schnittstelle

Inhaltsverzeichnis

2.1. Einleitung

In diesem Kapitel werden die Punkte behandelt, welche zur eigentlichen Schnittstellendefinition (Kapitel Strong Management Protocol) geführt haben. Das sind zum einen die Anforderungen, die teilweise aus der Aufgabenstellung stammen, als auch die Evaluation der Basistechnologie für das Managementprotokoll. Im Weiteren werden unterschiedliche Sicherheitskonzepte betrachtet und auf ihre Tauglichkeit überprüft. Im Abschnitt Analyse ipsec statusall wird zum Schluss die Ausgabe von **ipsec statusall** analysiert, und anhand dessen die Elemente für das Managementprotokoll festgelegt.

2.2. Anforderungen

In der Aufgabestellung wurden bereits die wichtigsten Anforderungen für die Kommunikationsschnittstelle aufgeführt. Diese Anforderungen, welche noch um Weitere ergänzt wurden, sind:

- Unabhängigkeit von einer Programmiersprache.
- Flexibilität und einfache Erweiterbarkeit.
- Erweiterungen sollten einfach implementiert werden können.

- Die Definition der Schnittstelle sollte einfach zu verstehen sein. Nach dem Prinzip Keep It Simple.
- Unabhängigkeit von grösseren Fremdbibliotheken.
- Unabhängigkeit von grafischer Darstellung.
- Unterstützung von Abfrage, Konfiguration, Steuerung (z.B. Tunnel up/down) und Signalisierung von VPN Verbindungen (im Vordergrund steht die Statusüberwachung). Die Verwaltung und das Monitoring sollte für mehrere hundert Verbindungen skalieren.
- Die Möglichkeit besitzen von einem zentralen Punkt aus mit mehreren VPN Gateways zu kommunizieren.
- Kompromisslos sichere Kommunikation zwischen Management Server und Daemon.

2.3. Technologieevaluation

2.3.1. Einleitung

Gemäss den Anforderungen sollte die Schnittstelle unabhängig von der Programmiersprache, flexibel und erweiterbar sein. Um die Daten zu repräsentieren gibt es unterschiedliche Möglichkeiten. Darunter fallen z.B. ASN.1, XML und YAML. ASN.1 und XML gehören dabei sicher zu den bekannteren und mächtigeren Technologien, wobei YAML eher für die Datenserialisierung geeignet ist.

Im Falle dieser Diplomarbeit, haben wir uns für eine XML-basierte Lösung entschieden. Dafür sprechen einige Gründe:

- Weit verbreitet und grosse Akzeptanz in der Wirtschaft
- State of the Art
- Flexibel und gut erweiterbar
- Unabhängig von der Programmiersprache
- Sehr gute Unterstützung von vielen Programmiersprachen
- Grosse Vielfalt von XML-basierten Technologien
- Vorhandene Spezifikationen zur Verschlüsselung und Bestimmung der Datenherkunft.

Der Nachteil einer XML-basierten Lösung besteht in der grossen Menge der zu übertragenden Daten, was viel Netzwerkbandbreite benötigt. Des Weiteren wird relativ viel Rechenleistung für das Parsen und die Validation der Nachrichten verbraucht.

Im Folgenden werden nun einige XML-basierte Kommunikationstechnologien betrachtet und auf ihre Tauglichkeit hin überprüft.

2.3.2. SOAP

SOAP (Simple Object Access Protocol) ist ein auf XML-RPC basiertes Protokoll zum Datenaustausch und Aufruf von Remote Procedure Calls (RPC) zwischen Webservices. Es wird normalerweise über HTTP betrieben. Andere Transportprotokolle wie SMTP oder POP3 sind aber als Übertragungsmedium ebenfalls möglich. Eine SOAP Nachricht besteht aus einem Envelope XML Element, welches aus einem Body und einem optionalen Header zusammengesetzt ist. Im Body werden dabei die Daten, Meldungen oder Funktionsaufrufe transportiert. SOAP ist plattformunabhängig, flexibel und für viele Programmiersprachen verfügbar. Der grosse Nachteil ist der Overhead, unter welchem die Performance und die Bandbreite leidet.

2.3.3. XML-RPC

XML-RPC ist wie der Name schon sagt ein RPC-Protokoll, in welchem Methodenaufrufe, Parameter und Rückgabewerte mit Hilfe von XML übertragen werden. Zu diesem Zweck ist ein Set von grundlegenden Datentypen (int, float, etc.) und Strukturen (struct, array) als XML-Element definiert. XML-RPC läuft ausschliesslich über HTTP. Jede Message wird auf ein HTTP-Post Request abgebildet. XML-RPC eignet sich gut für die Verwendung zwischen Webservices, es ist aber nicht so mächtig wie SOAP. Im Vergleich zu SOAP ist der Overhead bei XML-RPC kleiner. Dieses Verhalten hat positive Auswirkungen auf die Performance.

2.3.4. Schlussfolgerung

Die oben genannten Technologien eignen sich nicht als Basis für die Schnittstelle zwischen einem VPN Gateway und der Managementkonsole. Es existieren dafür unterschiedliche Gründe.

- Die oben genannten Technologien unterstützen im Normalfall nur die synchrone, "one-way" Kommunikation. Die Verarbeitung von asynchronen Meldungen wäre nicht möglich.

- XML-RPC basiert auf HTTP. Für die Kommunikation würde ein Webserver auf dem Security Gateway benötigt. Da Charon (Daemon von strongSwan) aber kein Webservice ist, erscheint ein Webserver fehl am Platz. Zu dem wird durch einen zusätzlichen Webserver die Sicherheit verschlechtert.

- Für eine sichere Übertragung mittels HTTP würde TLS und so fast zwingendermassen openssl benötigt werden. strongSwan sollte jedoch von openssl und somit TLS unabhängig sein.

- Die Kommunikation mittels SOAP müsste nicht zwingend über HTTP erfolgen, obwohl es auf XML-RPC basiert. Mit der C-Library von SOAP ist aber nur die Kommunikation mittels HTTP möglich. Es treten also die gleichen Probleme wie bei XML-RPC auf.

- XML-RPC besitzt nur einige grundlegende Datentypen und Strukturen, was für den Aufruf von entfernten Funktionen durchaus ausreicht. Kurz gesagt XML-RPC ist funktionsorientiert. Die Kommunikation zwischen Charon und der Managementkonsole ist aber informationsorientiert. Die Übertragung von XML-Informationsobjekten wäre besser geeignet. Des Weiteren sollten die zu transportierenden Elemente einen bestimmten Typ aufweisen und so

eine strikte Typenprüfung erlauben. Die Datentypen von XML-RPC erweisen sich dafür als nicht ausreichend.

Da sich die oben genannten Technologien als nicht geeignet erweisen, muss eine eigene Form der Kommunikation auf Basis von XML entwickelt werden. Es gilt dabei aber bestehende Technologien zu berücksichtigen.

2.4. Sicherheitsevaluation

2.4.1. Anforderungen

Bezüglich der Sicherheit ergeben sich für die Kommunikationsschnittstelle folgende Anforderungen:

- Keine Verwendung von TLS/SSL, da strongSwan von openssl unabhängig sein soll.
- Mindestens so sicher wie IPSec/IKEv2 selber.
- Möglichst einfach verständlich und implementierbar.

2.4.2. Evaluation

Es existieren vier naheliegende Lösungen um die Sicherheit der Verbindung zwischen dem Managementtool und dem strongSwan Daemon zu garantieren. Da die Verwendung von SSL/TLS aus obigem Grund nicht in Frage kommt, sind noch die folgenden Drei übrig.

- Verwendung eines strongSwan VPN Tunnels für die Administration.
- Verwendung von SAML 1.1, SOAP Message Security 1.1 oder ähnlichem.
- Eigene XML-basierte Lösung mit XML-Signature und XML-Encryption (basierend auf RFC's), welche in die Kommunikationsschnittstelle integriert werden.

Diese drei Möglichkeiten werden nun im Genauen unter die Lupe genommen.

2.4.2.1. VPN Tunnel für die Administration

Die Verwendung eines VPN Tunnels für die Administration ist wohl die einfachste und naheliegendste Variante. Diese hat jedoch den Nachteil, dass man sich bei einer Fehlkonfiguration ausschliessen könnte, und so die Konfiguration nicht mehr Rückgängig zu machen wäre. In diesem Fall müsste man sich z.B. über einen SSH Zugang wieder Zugriff verschaffen. Als Devise gilt: Die Schnittstelle zur Konfiguration sollte von der zu konfigurierenden Schnittstelle unabhängig sein.

2.4.2.2. SAML 1.1, SOAP Message Security 1.1

Diese Standards werden in der Praxis verwendet und haben sich bewährt, auch wenn die SOAP Message Security Spezifikation noch ziemlich neu ist. Sie sind sehr umfangreich und berücksichtigen alle nur erdenklichen Möglichkeiten. Die Implementierung der Standards ist daher

eher komplex und wäre für das Strong Management Protokoll zu überdimensioniert. Denn die Kommunikationsschnittstelle sollte möglichst einfach gehalten werden.

2.4.2.3. Eigene XML-basierte Lösung

Eine eigene, auf XML-basierende Lösung könnte explizit auf die Anforderungen der Schnittstelle angepasst und möglichst einfach gehalten werden. Zur Gewährleistung einer gewissen Qualität würde diese auf bereits existierenden und bewährten Spezifikationen basieren (XML-Signature und XML-Encryption).

2.4.3. Schlussfolgerung

Eine eigene, auf XML-basierende Sicherheitslösung bietet sich als beste und flexibelste Lösung an. Bei der Implementierung sollte darauf geachtet werde, dass das Verfahren zur Verschlüsselung austauschbar ist, ohne dass das Management Protokoll angepasst werden müsste. So wäre es in Zukunft problemlos möglich die Management Schnittstelle über HTTPS oder andere relativ sichere Protokolle zu betreiben. Diese Lösung erlaubt auch in Zukunft eine eigene, unabhängige und rollenbasierte Autorisierung für die Administration.

2.5. Analyse ipsec statusall

2.5.1. Einleitung

Gemäss Aufgabenstellung der Diplomarbeit besitzt das Spezifizieren der Statusabfrage die höchste Priorität. Zur erfolgreichen Durchführung muss deshalb die Ausgabe von **ipsec statusall** analysiert werden.

2.5.2. Analyse

Zur Zeit sind bei der Ausgabe von **ipsec statusall** unter IKEv2 die folgenden Informationsgruppen vorhanden:

- `Performance` - Informationen, welche sich auf die Performance eines Security Gateways beziehen. Darunter fallen die Anzahl Worker Threads und ob diese in Verwendung sind oder nicht, die Auslastung der Jobqueue und die Anzahl der ausstehenden Events.
- `Listening` - IPv4 und IPv6 Adressen der Interfaces auf welchen der Security Gateway lauscht.
- `Connections` - Alle in strongSwan vorhandenen Connections.
- `Policies` - Alle in strongSwan vorhandenen Security Policies.
- `Security Associations` - Alle Sicherheitsbeziehungen, bestehend aus IKE SA und Child SA.

Anhand eines Beispiels werden nun die einzelnen Teile der **ipsec statusall** Ausgabe im Genaueren betrachtet. Bei der folgenden Konfiguration handelt es sich um den Test **ikev2/crl-strict** aus der Webseite von strongSwan [http://www.strongswan.org].

```
# /etc/ipsec.conf - strongSwan IPsec configuration file

config setup
        strictcrlpolicy=yes
        plutostart=no

conn %default
        ikelifetime=60m
        keylife=20m
        rekeymargin=3m
        keyingtries=1
        keyexchange=ikev2
        left=192.168.0.1
        leftnexthop=%direct
        leftcert=moonCert.pem
        leftid=@moon.strongswan.org

conn net-net
        leftsubnet=10.1.0.0/16
        right=192.168.0.2
        rightsubnet=10.2.0.0/16
        rightid=@sun.strongswan.org
        auto=add

conn host-host
        right=192.168.0.2
        rightid=@sun.strongswan.org
        auto=add

conn rw
        leftsubnet=10.1.0.0/16
        right=%any
        auto=add
```

Wie man oben sehen kann wurden auf dem Security Gateway Moon drei Verbindungen definiert. *net-net* für eine Netz zu Netz Verbindung, *host-host* für die Verbindung von zwei Security Gateways und *rw* für die Verbindung von einem Roadwarrior zum Security Gateway.

Nach einer erfolgreichen **ipsec statusall** Ausgabe auf Moon, erhält man folgenden Output.

```
Performance:
  worker threads: 4 idle of 4,  job queue load: 0,
                  scheduled events: 3

Listening on 8 IP addresses:
  192.168.0.1
  10.1.0.1
  127.0.0.1
  fec0::1
```

```
  fe80::fcfd:c0ff:fea8:1
  fec1::1
  fe80::fcfd:aff:fe01:1
  ::1

Connections:
    net-net:  192.168.0.1...192.168.0.2
  host-host:  192.168.0.1...192.168.0.2
         rw:  192.168.0.1...%any

Policies:
     net-net:  '@moon.strongswan.org'...'@sun.strongswan.org'
   host-host:  '@moon.strongswan.org'...'@sun.strongswan.org'
          rw:  '@moon.strongswan.org'...'%any'

Security Associations:
   rw{1}: ESTABLISHED, 192.168.0.1[@moon.strongswan.org]...
          192.168.0.100[carol@strongswan.org]
   rw{1}: IKE SPIs: 0x10b3fd172edaac34_i 0x3711a820c2dd28f2_r*
   rw[1]: INSTALLED, reqid: 2000000001,
          ESP SPIs: 0xcc716134_i 0xc812c80c_o
   rw[1]: AES_CBC-128/HMAC_SHA1_96, rekeying in 1013s
   rw[1]: 10.1.0.0/16===192.168.0.100/32,
          last use: no_in no_out no_fwd
```

Im obigen Auszug sind die bereits fünf bekannten Informationsgruppen ersichtlich. Interessant ist dabei, dass zur Zeit nur eine Verbindung aufgebaut ist (*rw{1}*) und zwar vom Roadwarrior `carol@strongswan.org` zum Security Gateway `@moon.strongswan.org`.

Nun sollten die drei Informationsgruppen *Connections, Policies* und *Security Associations* genauer betrachtet werden, da diese die Informationen, welche von Interesse sind, beinhalten.

2.5.2.1. Connections

Unter `Connections` werden alle in strongSwan konfigurierten Verbindungen aufgeführt. Es handelt sich dabei aber nicht eins zu eins um die in `ipsec.conf` konfigurierten Verbindungen, obwohl sie deren Namen tragen. Eine in `ipsec.conf` festgelegte Verbindung wird in strongSwan auf eine Connection und Policy verteilt. Wobei dieses Verfahren die Migrationsstrategie von strongSwan wiederspiegelt. Eine Connection in strongSwan dient als *Template* für das Aushandeln einer IKE SA. Bei den IP Adressen, neben den Verbindungsnamen, handelt es sich dabei um die IP Adressen der beteiligten Host, zwischen welchen eine IKE SA aufgebaut werden kann. In `ipsec.conf` werden diese IP Adressen in den Eigenschaften `left` und `right` festgelegt. Natürlich beinhaltet eine Connection mehr Informationen als unten aufgeführte.

```
Connections:
    net-net:  192.168.0.1...192.168.0.2
  host-host:  192.168.0.1...192.168.0.2
         rw:  192.168.0.1...%any
```

2.5.2.2. Policies

Eine Policy dient als *Template* für das Aushandeln einer Child SA. Dies geschieht indem einer IKE SA eine Policy übergeben wird, anhand welcher die IKE SA eine Child SA erzeugt und aufbaut. Bei der Ausgabe der Policies sind zur Zeit nur die ID's der Kommunikationspartner aufgelistet. Ein `%any` bedeutet, dass ein Kommunikationspartner mit einer beliebigen ID mit dem Security Gateway kommunizieren kann. In `ipsec.conf` wird die ID in den Eigenschaften `leftid` und `rightid` festgelegt.

```
 Policies:
  net-net:  '@moon.strongswan.org'...'@sun.strongswan.org'
host-host:  '@moon.strongswan.org'...'@sun.strongswan.org'
       rw:  '@moon.strongswan.org'...'%any'
```

Mögliche Varianten von ID's sind:

- IPv4 und IPv6 Adressen
- FQDN's und USER_FQDN's
- DER_ASN1_DN

Wie die oben aufgeführten ID's aussehen, kann der Webseite von strongSwan [http://www.strongswan.org] im Configuration Guide entnommen werden.

Nebst den ID's beinhalte eine Policy natürlich noch viele weitere Informationen, wie z.B. den ESP Verschlüsselungsalgorithmus welcher zur Sicherung des Traffics verwendet werden soll.

2.5.2.3. Security Associations

Durch eine Security Association wird eine Sicherheitsbeziehung zwischen zwei Kommunikationspartnern beschrieben. Eine SA ist immer dann vorhanden, wenn ein VPN Tunnel zwischen zwei Kommunikationspartnern besteht. In strongSwan wird eine Sicherheitsbeziehung (Child SA) mit Hilfe von IKEv2 durch eine IKE SA aufgebaut. In der Ausgaben von **ipsec statusall** sind diese beiden Komponenten zuunterst aufgeführt (IKE SA - *rw{1}* , Child SA - *rw[1]*).

Die IKE SA von Moon besteht gemäss Ausgabe aus folgenden Komponenten:

- `ESTABLISHED` - Der Status der aufgebauten Sicherheitsbeziehung.
- `192.168.0.1[@moon.strongswan.org]...` - Die IP Adressen und die IKE ID der Kommunikationspartner.
- `IKE SPIs: 0x10b3fd172edaac34_i 0x3711a820c2dd28f2_r*` - Die SPIs der Sicherheitsbeziehung, `_i` für Initiator und `_r` für Responder. Der Stern legt fest, ob es sich bei Moon um den Initiator oder Responder handelt.

Die zur IKE SA gehörende Child SA besitzt folgenden Aufbau:

- `INSTALLED` - Status der Child SA (`INSTALLED` bedeutet, dass die Child SA in Betrieb ist).

- `reqid: 2000000001` - Die ID, welche die Child SA identifiziert.
- `ESP SPIs: 0xcc716134_i 0xc812c80c_o` - SPI's der Child SA. Wobei diese aus einem Child SA Pair besteht, eine für inbound Traffic (`_i`) und eine für outbound Traffic (`_o`).
- `AES_CBC-128/HMAC_SHA1_96` - Der Algorithmus für die ESP Verschlüsselung und die ESP Authentication.
- `rekeying in 1013s` - Die Dauer bis die Sessionschlüssel neu ausgehandelt werden.
- `10.1.0.0/16===192.168.0.100/32` - Trafficselektoren, welcher für diese SA ausgehandelt wurden. Es handelt sich dabei um Addressranges, welche mit `leftsebnet` und `rightsubnet` in `ipsec.conf` konfiguriert wurden.
- `last use: no_in no_out no_fwd` - Die Dauer der letzten Benutzung der Child SA für in, out und forward Traffic.

```
Security Associations:
rw{1}: ESTABLISHED, 192.168.0.1[@moon.strongswan.org]
       ...192.168.0.100[carol@strongswan.org]
rw{1}: IKE SPIs: 0x10b3fd172edaac34_i 0x3711a820c2dd28f2_r*
rw[1]: INSTALLED, reqid: 2000000001,
       ESP SPIs: 0xcc716134_i 0xc812c80c_o
rw[1]: AES_CBC-128/HMAC_SHA1_96, rekeying in 1013s
rw[1]: 10.1.0.0/16===192.168.0.100/32,
       last use: no_in no_out no_fwd
```

2.5.3. Ergebnisse

Aus der Analyse wird ersichtlich, dass es Sinn macht in der zu definierenden Kommunikationsschnittstelle Elemente zu verwenden, welche den Komponenten der Informationsgruppen entsprechen. Das wären die folgenden Elemente.

- *Performance*
- *Interface*
- *Connection*
- *Policy*
- *IKESA*
- *ChildSA*

Wenn man strongSwan betrachtet, wird ersichtlich das ein Teil der oben aufgeführten Elemente mit Klassen in den strongSwan Sourcen konvergent sind (*Connection, Policy, ChildSA* und *IKESA*). Daneben wäre es sinnvoll die Elemente *Connection* und *Policy* nicht nur beim Transport für Informationen, sondern auch bei einer Konfiguration zu verwenden.

Kapitel 3. Strong Management Protocol

Inhaltsverzeichnis

3.1. Einleitung

Das Strong Management Protocol (SMP) ist ein Layer 7 (Application Layer) Protokoll um den IKEv2 Daemon (Charon) von strongSwan 4 zu managen. SMP basiert auf XML und sämtliche Nachrichten werden gemäss XML strukturiert, typisiert dargestellt und übertragen. Das Protokoll ist zustandslos und arbeitet nach dem Request/Response-Prinzip. Auf eine Anfrage folgt immer eine Antwort. Anfragen und Antworten werden nicht direkt durch ID's in Beziehung zueinander gesetzt und sind somit nicht unabhängig von dem zugrunde liegenden Transfermechanismus. In diesem Fall hätten auch alle charakteristischen Eigenheiten des Protokolls für den Transport, wie Wartezeit, Synchronisierung, etc. berücksichtigt werden müssen.

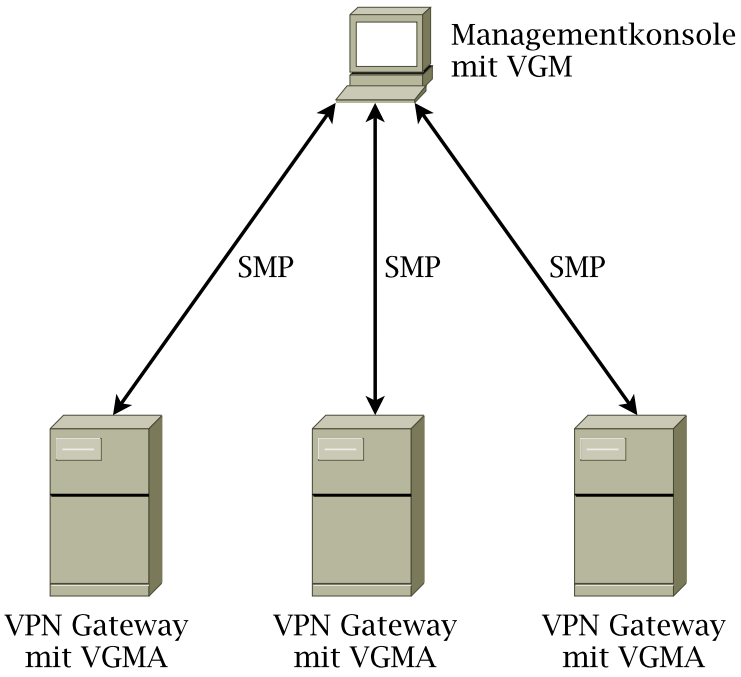
Zu den Aufgaben des Strong Management Protocol zählen:

- Abfragen von Informationen (z.B. Status von VPN Verbindungen, vorhandene CA Zertifikate etc.)
- Steuerung von VPN Verbindungen (z.B. up/down)
- Konfiguration von VPN Verbindungen (neue Konfiguration erstellen, bestehende abfragen und ändern)
- Signalisation (Fehlererkennungen und Benachrichtigungen -> asynchrone Meldungen)

Da SMP flexibel, plattform- und programmiersprachenunabhängig ist, ermöglicht es den IKEv2 Daemon mit unterschiedlichen Managementkonsolen zu warten. Das kann sowohl eine Web- als auch eine GUI-Applikation sein.

3.2. Architektur

Die SMP Architektur besteht aus den beiden Komponenten **VGMA** (VPN Gateway Management Agent) und **VGM** (VPN Gateway Manager). Der VGMA stellt dabei die Kommunikationsschnittstelle zum SMP dar, durch welche der IKEv2 Daemon von strongSwan mit dem VGM kommunizieren kann. Durch einen VGM können mehrere VGMA's verwalten werden. In welcher Programmiersprache ein VGMA implementiert wird, spielt dabei eben so wenig eine Rolle, wie die Applikationsart in welcher dieser verwendet wird.



Architektur von SMP

3.2.1. SMP-Message

Im Strong Management Protocol ist jede Anfrage, Antwort oder Notifikation eine XML strukturierte SMP-Message, welche aus einem optionalen Header und einem Body besteht. Die eigentlichen Request und Response Elemente werden im Body der SMP Message übertragen. Diese Form der Kapselung erlaubt eine flexible und komfortable Implementierung der nötigen Sicherheitskonzepte. Das Element *SMPMessage* repräsentiert eine SMP-Message. Die Spezifiaktion dieses Elements kann dem Abschnitt SMPMessage Element entnommen werden.

3.2.2. Security

Kompromisslose Sicherheit ist eine wichtige Anforderung an das Protokoll. Die Integrität der Daten, als auch die Senderidentifizierung und -authorisierung muss gewährleistet sein. Des Weiteren sollten die Daten vor unberechtigten Einblicken geschützt werden. Diesen Anforderungen wird Rechnung getragen durch Verwendung von XML Signaturen und einer XML basierten Verschlüsselung der Nachrichten. Die Spezifikationen der Elemente, welche diese Fuktionalität zur Verfügung stellen, können den Abschnitten XML-Signature und XML-Encryption entnommen werden.

3.2.3. Requests

Um Informationen über einen VPN Gateway zu erhalten und dessen Verbindungen zu steuern oder zu konfigurieren, sind folgende Request-Message PDU's definiert:

- *GetRequest* - Abfragen von Managementinformationen und Konfigurationen (z.B. Tunnelinformationen, verwendete Zertifikate etc.).
- *SetRequest* - Initiieren einer Konfiguration auf dem Security Gateway (z.B. definieren einer neuen Connection).

- *ControlRequest* - Request zur Initiierung einer Steuerungsanweisung (z.B. up/down der Connection).

Die oben aufgelisteten Requests werden zur Zeit ausschliesslich vom VPN Gateway Manager (VGM) ausgelöst. Später wäre jedoch auch ein *GetRequest* von Seiten des VGMA denkbar.

Die Spezifikationen der oben aufgeführten Elemente, befinden sich im Abschnitt Requests.

3.2.4. Responses

Auf eine Request-Message folgt immer eine Response-Message. Zu diesem Zweck sind folgende PDU's definiert:

- *GetResponse* - Antwort auf einen *GetRequest*, welche die gewünschten Managementinformationen beinhaltet.

- *SetResponse* - Antwort auf einen *SetRequest*.

- *ControlResponse* - Antwort auf einen *ControlRequest*.

In jeder Response-Message wird über Erfolg oder Nichterfolg des getätigten Requests informiert. Zu diesem Zweck beinhalten eine Response-Message einen Statuscode, ähnlich dem des HTTP Protokolls, sowie eine optionale Statusmeldung.

Die Spezifikationen der oben aufgeführten Elemente, befinden sich im Abschnitt Responses.

3.2.5. Asynchrone Meldungen

In den meisten Systemen können unerwartete Ereignisse, wie Fehler, Alarme, etc., auftreten. Damit auf solche Typen von Ereignissen entsprechend reagiert werden kann, ist in SMP die Notification-Message definiert. Durch sie kann ein VPN Gateway Manager Agent (VGMA), beim Auftreten der oben genannten Ereignisse, den VPN Gateway Manager informieren. Die Notification ist aber nicht auf das Mitteilen von Fehlern und Alarmen beschränkt, auch andere asynchrone Meldungen können versandt werden. Die Spezifikation des Elements, welches die Notification repräsentiert, kann dem Abschnitt Notification Element entnommen werden.

3.3. Kommunikation

3.3.1. Transportmedium

Das Strong Management Protocol verwendet als Transportmedium TCP auf dem **Port 4502**. TCP wurde gegenüber UDP bevorzugt, weil es einen zuverlässigen und garantierten Transport der Daten ermöglicht. SMP kann dadurch lokal über Unix Sockets wie auch in einer dezentralisierten, verteilten Umgebung über TCP Sockets eingesetzt werden. Die Verwendung von TCP erlaubt auch das Versenden von asynchronen Notification-Messages, ohne dass der entsprechende Empfänger sich zuvor registrieren muss. Solange eine TCP Verbindung besteht, können beide Seiten Messages austauschen.

Anwendung	SMP
Transport	TCP
Internet	IP (IPv4, IPv6)
Netzwerk	Ethernet

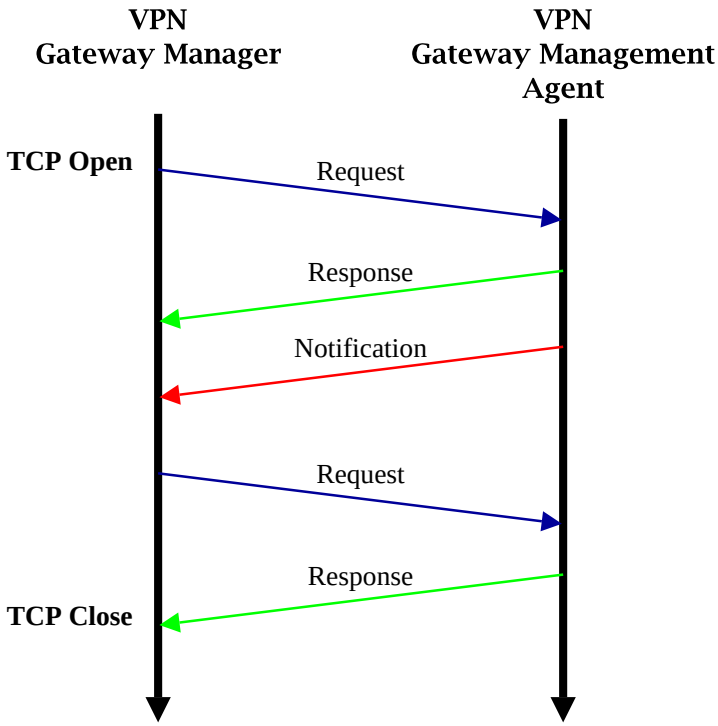
SMP Protokollstack

3.3.2. Struktur

Die als XML kodierte SMP-Message wird direkt über den TCP- oder Unix-Socket übermittelt. Gemäss XML-Spezifikation kann die Message in der ersten Zeile eine optionale XML-Deklaration enthalten, die über XML-Version und Zeichenkodierung informiert. Diese Deklaration ist nicht verpflichtend, wird jedoch empfohlen. Wenn die Kodierung fehlt, muss die XML-Message entweder UTF-8 oder UTF-16 formatiert sein. Empfohlen wird die Kodierung mittels UTF-8. Jede SMP-Message muss mit einer leeren Zeile abgeschlossen werden. Die Message selber darf **keine Leerzeilen** enthalten, da diese das Ende der Message signalisieren würden. Es wird Empfohlen die Message vor der Übertragung mittels Canonical XML (C14N) zu kanonisieren. So kann sichergestellt werden, dass die Message UTF-8 kodiert ist und keine Leerzeilen enthält.

3.3.3. Kommunikationsablauf

Der VGM stellt die TCP Verbindung zum VGMA her. Die Verbindung bleibt im Normalfall solange bestehen, bis der VGM diese beendet. Über diese Verbindung kann der Manager beliebig viele Requests absetzten. Auf eine Request-Message muss immer eine Response-Message folgen. Der VGMA behandelt eine Anfrage immer entweder vollständig oder gar nicht. Der Statuscode der Response informiert hierbei über Erfolg oder Misserfolg. Bei einem Fehlercode kann davon ausgegangen werden, dass der VGMA überhaupt keine Anfrage abgearbeitet hat. Der VGMA kann bei einer offenen Verbindung zu jedem Zeitpunkt eine Notification-Message versenden.



Kommunikation von SMP

3.3.4. Validation

Um den Aufbau eines Requests und Response auf seine Gültigkeit hin zu prüfen, werden diese anhand eines Relax NG Schemas validiert. Die Schemen wurden derart gestaltet, dass deren Inhalt gut erweitert werden kann. Zur Validierung der Message muss der XML-Namensraum **http://www.strongswan.org/smp/1.0** in der SMP-Message angegeben werden.

```
<?xml version='1.0' encoding='UTF-8'?>
<SMPMessage xmlns="http://www.strongswan.org/smp/1.0">
...
</SMPMessage>
```

Relax NG wurde als XML-Schemasprache bevorzugt, weil diese einfach zu verstehen, schnell zu erlernen und der Aufbau im Vergleich zu XML-Schema intuitiver ist, was zu einer einfacheren Lesbarkeit führt. Diese Vorteile erleichtern die Erweiterung von SMP durch unterschiedliche Personen sehr und das Protokoll kann deswegen auch von weniger XML versierten Anwendern verstanden werden. Relax NG verfügt dabei praktisch über einen gleichwertigen Funktionsumfang wie XML-Schema.

3.4. SMPMessage Element

Eine *SMPMessage* stellt die grundlegende Nachrichtenstruktur des Strong Management Protocol dar. Jede Request- und Response-Message ist eine *SMPMessage*. Eine *SMPMessage* besteht immer aus einem *Body* und einem optionalen *Header*, wobei im *Body* die eigentlichen Request oder Response Elemente gekapselt werden. Im Header sind die XML-Signatur und weitere sicherheitsbezogene Informationen abgelegt. Als Einführung wird im Folgenden nun eine *SMPMessage* anhand eines Beispiels erklärt.

```
 1 <SMPMessage>
 2   <Header>
 3     <Security>
 4       <Signature>
 5         <SignedInfo>
 6           <SignatureMethod Algorithm="RSA_SHA-256"/>
 7           <Digest>
 8             <DigestMethod Algorithm="SHA-256"/>
 9             <DigestValue>
10               213d61229b591e3b81e90f1c052f5f0ae9e0a157
11             </DigestValue>
12           </Digest>
13         </SignedInfo>
14         <SignatureValue>
15           481a59501150691f1cf602e41e64abd0b6d2c166
16         </SignatureValue>
17       </Signature>
18     <Security>
19   </Header>
20   <Body>
21     <InformationRequest>
22       ...
23     </InformationRequest>
24   </Body>
25 </SMPMessage>
```

Die oben aufgeführte *SMPMessage* stellt einen *InformationRequest* mit Signatur dar. Der optionale Header (2-19) beinhaltet diese Signatur (4-17), welche aus dem Digest des *Body* (7-12) durch das Signaturverfahren (6) berechnet wurde und im Element *SignatureValue* (14-16) abgelegt ist. Der *Body* (20-24) selbst kapselt den *InformationRequest*, durch welchen Informationen vom IKEv2 Daemon abgefragt werden können. Nebst den oben aufgeführten Komponenten besitzt die *SMPMessage* noch weitere. Alle Komponenten der *SMPMessage* können den nachfolgenden Schemen, sowie der komponenten-spezifischen Dokumentation entnommen werden.

3.4.1. Schema

Das Schema einer *SMPMessage* ist wie folgt definiert.

```
<define name ="SMPMessage">
  <element name="SMPMessage">
    <optional>
      <element name="Header">
        <ref name="Security"/>
      </element>
    </optional>
    <ref name="Body"/>
  </element>
</define>
```

Im Body können die eigentlichen Request oder Response Elemente entweder direkt aufgeführt, oder in einem *EncryptedData* Element verschlüsselt übertragen werden. Das Schema für den Body ist wie folgt definiert.

```
<define name="Body">
  <element name="Body">
    <choice>
      <ref name="MessageType"/>
      <element name="EncryptedData">
        <ref name="MessageType"/>
      </element>
    </choice>
  </element>
</define>
```

Das Strong Management Protocol besitzt sechs grundlegende Nachrichtentypen, welche in der Definition *MessageType* zusammengefasst sind. Weitere Informationen über die einzelnen Request und Response Elemente können den entsprechenden Kapiteln entnommen werden. In einer *SMPMessage* ist nur ein Nachrichtentyp auf einmal erlaubt.

```
<define name="MessageType">
  <choice>
    <ref name="GetRequest" />
    <ref name="SetRequest" />
    <ref name="ControlRequest" />
    <ref name="GetResponse" />
    <ref name="SetResponse" />
    <ref name="ControlResponse" />
    <ref name="Notification" />
  </choice>
</define>
```

Damit die Signatur nicht direkt im Header platziert werden muss, wurde das Element *Security* geschaffen. Dadurch wird im Header eine klare Trennung zwischen der Security und allfälligen späteren Komponenten geschaffen.

```
<define name="Security">
  <element name="Security">
    <optional>
      <ref name="Signature"/>
    </optional>
  </element>
</define>
```

3.5. Security

3.5.1. XML-Signature

3.5.1.1. Einleitung

Um die Integrität und Authentizität der Daten, als auch die Senderberechtigung und Identifikation zu gewährleisten, werden XML-Signaturen und -Zertifikate verwendet. Der Aufbau und die Verarbeitung von XML-Signaturen wird bereits im RFC 3275 (XML-Signature Syntax and Processing) spezifiziert. Diese Spezifikation weist aber gewisse Unzulänglichkeiten in den verwendeten Algorithmen auf und ist für unsere Zwecke zu überladen. Aus diesem Grund wurde, basierend auf dem RFC 3275, eine eigene Spezifikation entworfen, welche auf strongSwan abgestimmt ist.

3.5.1.2. Signature Element

Eine Signatur einer Message wird durch das Element *Signature* repräsentiert. Das Element *Signature* besteht aus Informationen über die Signatur, der Signatur selbst und Informationen über das verwendete Zertifikat, welches zur Überprüfung der Signatur und Identifikation des Senders verwendet wird.

```
<define name="Signature">
  <element name="Signature">
    <ref name="SignedInfo" />
    <ref name="SignatureValue" />
    <ref name="KeyInfo" />
  </element>
</define>
```

Das Element *Signature* befindet sich optional in einer *SMPMessage* im Element *Security* , welches wiederum im *Header* Element platziert ist.

3.5.1.2.1. SignedInfo Element

Das Element *SignedInfo* beinhaltet die Elemente *SignatureMethod* und *Digest* und stellt somit Informationen für das Signieren zur Verfügung.

```
<define name="SignedInfo">
  <element name="SignedInfo">
    <ref name="SignatureMethod"/>
    <ref name="Digest"/>
  </element>
</define>
```

Das Element *SignatureMethod* legt das Verfahren fest, welches für das Erstellen und Validieren einer Signatur verwendet wird. In SMP sind diese Verfahren: Hashing mittels SHA und Signierung mittels RSA oder HMAC.

```
<define name="SignatureMethod">
  <element name="SignatureMethod">
```

```
    <attribute name="Algorithm">
      <choice>
        <value type="string">RSA_SHA_PSS</value>
        <value type="string">RSA_SHA_PKCS1-256</value>
        <value type="string">RSA_SHA_PKCS1-384</value>
        <value type="string">RSA_SHA_PKCS1-512</value>
        <value type="string">HMAC_128_SHA-256</value>
        <value type="string">HMAC_128_SHA-384</value>
        <value type="string">HMAC_128_SHA-512</value>
      </choice>
    </attribute>
    <optional>
      <element name="PSSParams">
        <data type="base64Binary" />
      </element>
    </optional>
    <optional>
      <ref name="EncryptedKey" />
    </optional>
  </element>
</define>
```

Wie oben ersichtlich ist, sind für das Erstellen der Signatur verschiedene Verfahren mit HMAC oder RSA zulässig. Im Falle von HMAC wird ein Secret benötigt. Dieses muss im optionalen Element *EncryptedKey* abgelegt werden. Der Aufbau von *EncryptedKey* wird im Abschnitt XML-Encryption in diesem Kapitel erläutert.

Für die oben aufgelisteten RSA-Verfahren wird RSA in Form des Algorithmus RSASSA-PSS oder RSASSA-PKCS1, beide im RFC 3447 beschrieben, verwendet. Der Signierungsvorgang sieht für PSS vereinfacht folgendermassen aus.

```
RSASSA-PSS-SIGN(PrivateKey, DATA)
```

Das Element *SignedInfo* (`DATA`) wird übergeben, gehashed und mit RSA signiert. Aus Sicherheitsgründen werden nur Algorithmen für sicheres Hashing verwendet (Wikipedia - Im Sommer 2006 sind wesentliche Schwächen von SHA-1 entdeckt worden). Als RSA Schlüssel sollten nur Schlüssel verwendet werden, welche mindestens eine Länge von 2'048 Bit aufweisen.

Um die Signatur zu überprüfen werden wiederum die Algorithmen RSASSA-PSS oder RSASSA-PKCS1 verwendet. Die Überprüfung sieht vereinfacht folgendermassen aus.

```
RSASSA-PSS-VERIFY((Modulus, Exponent), DATA, Signatur)
```

Das Element *SignedInfo* (`DATA`) wird mit der Signatur aus dem Element *SignatureValue* übergeben und mittels der Methode `RSASSA-PSS-VERIFY(...)` überprüft.

Bitte beachten!

Im RFC 3275 wird für die Signatur der Algorithmus RSASSA-PKCS1-v1_5 verwendet. Obwohl bis jetzt keine Sicherheitslücken für diesen Algorithmus bekannt

sind, empfiehlt es sich in neuen Applikationen den Nachfolgealgorithmus RSASSA-PSS einsetzen. Die Schlüssellänge und andere für RSASSA-PSS benötigte Parameter können durch das Element *PSSParams* übertragen werden.

3.5.1.2.2. Digest Element

Um die Integrität der übertragenen Daten im *Body* Element zu gewährleisten, wird über diese ein Digest berechnet. Das Element *Digest* kapselt diesen Hash und den zum Hashing verwendeten Algorithmus.

```
<define name="Digest">
  <element name="Digest">
    <ref name="DigestMethod"/>
    <element name="DigestValue">
      <data type="base64Binary"/>
    </element>
  </element>
</define>
```

DigestMethod legt den Hashalgorithmus fest, mit welchem der Digest des Elements *Body* berechnet wird. In SMP sind dafür die Algorithmen SHA-256, SHA-384 und SHA-512 zulässig. Der berechnete Digest selber wird im Element *Digest*, codiert im Base64 Format, abgelegt.

```
<define name="DigestMethod">
  <element name= "DigestMethod">
    <attribute name="Algorithm">
      <choice>
        <value type="string">SHA-256</value>
        <value type="string">SHA-384</value>
        <value type="string">SHA-512</value>
        </choice>
    </attribute>
  </element>
</define>
```

3.5.1.2.3. SignatureValue Element

Die Signatur, welche über dem Element *SignedInfo* berechnet wurde, wird im Element *SignatureValue* Base64 codiert abgelegt.

```
<define name="SignatureValue">
  <element name="SignatureValue">
    <data type="base64Binary"/>
  </element>
</define>
```

3.5.1.2.4. KeyInfo Element

KeyInfo beinhaltet den Key und Informationen über den Key, der für die Signierung verwendet wurde. Mit diesem mitgelieferten Key kann die Signatur und die Vertrauenswürdigkeit des Absenders überprüft werden (Identifikation und Ableitung der Berechtigung).

```
<define name="KeyInfo">
  <element name="KeyInfo">
    <choice>
      <ref name="PGPData"/>
      <ref name="X509Data"/>
    </choice>
  </element>
</define>
```

Zum Signieren müssen in SMP immer Schlüssel im RSA Format verwendet werden. Diese RSA Schlüssel sind in unterschiedlichen Formaten gekapselt vorhanden. In SMP werden folgende zwei Typen unterstützt.

- *PGPData* - PGP Public Key Packet (Certificate)
- *X509Data* - X.509 Certificate mit Public Key

Beim ersten Typ handelt es sich um eine „PGP Key ID“ und ein „Public Key Packet“ wie diese gemäss RFC 2440 (Abschnitt 11.2 / 5.5.1.1) spezifiziert wurden. Mit Hilfe des Keys kann der Empfänger die Signatur auf ihre Gültigkeit hin überprüfen und durch die Key ID feststellen, ob der Absender berechtigt ist auf Dienste des IKEv2 Daemons zuzugreifen.

```
<define name="PGPData">
  <element name="PGPData">
    <choice>
      <group>
        <element name="PGPKeyID">
          <data type="base64Binary" />
        </element>
        <optional>
          <element name="PGPKeyPacket">
            <data type="base64Binary" />
          </element>
        </optional>
      </group>
      <group>
        <element name="PGPKeyPacket">
          <data type="base64Binary" />
        </element>
      </group>
    </choice>
  </element>
</define>
```

Beim zweiten Typ handelt es sich um ein X.509 Zertifikat, welches zur Überprüfung der Signatur und der Berechtigung des Absenders verwendet werden kann. Falls kein Zertifikat mitgeliefert wurde, kann es auf dem Host mittels *X509SKI* oder *X509SHA1Digest* gefunden werden.

```
<define name="X509Data">
```

```
  <element name="X509Data">
    <choice>
      <element name="X509SKI">
        <data type="base64Binary" />
      </element>
      <element name="X509SHA1Digest">
        <data type="base64Binary" />
      </element>
    </choice>
    <optional>
      <element name="X509Certificate">
        <data type="base64Binary" />
      </element>
    </optional>
    <optional>
      <element name="X509CRL">
        <data type="base64Binary" />
      </element>
    </optional>
  </element>
</define>
```

3.5.1.3. Prozessablauf

Dieser Abschnitt beschreibt den Ablauf zur Signaturgenerierung und -validierung. Dieser Prozessablauf muss bei einer Implementierung von SMP zwingend eingehalten werden, damit ein korrektes Ergebnis erzielt werden kann.

3.5.1.3.1. Erstellung des Digest

1. Serialisieren der Elemente im Element *Body* mittels Canonical XML (XML-c14n).

2. Berechnen des Digest über die serialisierten Elemente mit der in *DigestMethod* festgelegten Hashing Methode.

3. Erstellen des *Digest* Element, in welchem die verwendete Hashing Methode im Element *DigestMethod* und der Digest im Element *DigestValue* im Base64 Format abgelegt werden.

3.5.1.3.2. Erstellen der Signatur

1. Erstellen des *SignedInfo* Elements mit den Elementen *SignatureMethod* und *Digest* .

2. Serialisieren des Elements *SignedInfo* mit Canonical XML (XML-c14n).

3. Anwenden des in *SignatureMethod* festgelegten Signaturverfahrens auf das serialisierte *SignedInfo* Element.

4. Erstellen des *SignatureValue* Elements, welches die berechnete Signatur im base64 Format beinhaltet.

3.5.1.3.3. Digestvalidierung

1. Serialisieren der Elemente im Element *Body* mittels Canonical XML (XML-c14n).

2. Berechnen des Digest über die serialisierten Elemente mit der in *DigestMethod* festgelegten Hashing Methode.

3. Vergleichen des berechneten Digest mit dem übertragenen Wert. Falls diese nicht übereinstimmen ist die Validierung fehlgeschlagen.

3.5.1.3.4. Signaturvalidierung

1. Beschaffen des Keys aus dem Element *KeyInformation* oder von einer externen Quelle.

2. Berechnen der serialisierten Form des Elements *SignedInfo* .

3. Überprüfen der Signatur nach dem verwendeten Verfahren.

Nicht vergessen:

Das Berechnen der Signatur erfolgt nach der Verschlüsselung der Daten. Das Überprüfen der Signatur muss hingegen vor der Entschlüsselung stattfinden.

3.5.2. XML-Encryption

3.5.2.1. Einleitung

Damit die zu übertragenden Informationen vor unberechtigten Einblicken geschützt sind, werden diese symmetrisch verschlüsselt. In SMP beschränkt sich die Verschlüsselung auf die Request und Response Elemente. Diese werden im Falle einer Verschlüsselung nicht direkt im *Body* , sondern in einem *EncryptedData* Element platziert, welches sich im Body befindet. Ein Beispiel sieht folgendermassen aus.

```
<EncryptedData>
  <EncryptionMethod Algorithm="AES-256-CBC"/>
  <KeyInfo>
    <EncryptedKey>
      <CipherData>
        <CipherValue>K8HBLT45H</CipherValue>
      </CipherData>
    </EncryptedKey>
  </KeyInfo>
  <CipherData>
    <CipherValue>A23B45C56</CipherValue>
  </CipherData>
</EncryptedData>
```

Das XML-Encryption orientiert sich grundsätzlich an der XML-Encryption Spezifikation des W3C.

3.5.2.2. EncryptedData Element

Wie man im Beispiel sehen kann, enthält das *EncryptedData* Element nebst den verschlüsselten Daten ein *KeyInfo* Element mit dem dazugehörigen asymmetrisch verschlüsselten Key. Im Strong Management Protocol kommt für diesen Zweck ein hybrides Verschlüsselungsverfahren zur Anwendung. Das *EncryptedData* Element sieht in Relax NG folgendermassen aus.

```
<define name="EncryptedData">
  <element name="EncryptedData">
    <ref name="EncryptionMethod/">
    <element name="KeyInfo">
      <ref name="EncryptedKey"/>
    </element>
    <ref name="CipherData"/>
  </element>
</define>
```

Zur Verschlüsselung wird ausschliesslich AES im Cipher Block Chaining (CBC) Modus mit folgenden Stärken unterstützt.

```
<define name="EncryptionMethod">
  <element name="EncryptionMethod">
      <attribute name="Algorithm">
        <choice>
          <value type="String">AES-128-CBC</value>
          <value type="String">AES-256-CBC</value>
          <value type="String">AES-512-CBC</value>
        </choice>
      </attribute>
    </element>
</define>
```

Der Schlüssel für die symmetrische Verschlüsselung wird asymmetrisch mit dem Public Key des Gegenübers verschlüsselt und im Element *CipherData* abgelegt. Falls der Kommunikationspartner mehrere Public Keys besitzt, kann der verwendete Schlüssel in einem *KeyInfo* Element übertragen werden, wobei es sich dabei um das gleiche Element handelt wie bei der XML-Signatur.

```
<define name="EncryptedKey">
  <optional>
    <ref name="KeyInfo"/>
  </optional>
  <ref name="CipherData"/>
</define>

<define name="CipherData">
  <element name="CipherData">
    <element name="CipherValue">
      <data type="base64Binary"/>
    </element>
```

```
  </element>
</define>
```

3.5.2.3. Prozessablauf

Diese Sektion beschreibt die Operationen, die ausgeführt werden müssen, um SMP XML-Encryption zu implementieren.

3.5.2.3.1. Verschlüsselung

1. Serialisieren der zu verschlüsselnden Daten (Elemente im Body Element) mittels Canonical XML (C14N), so dass diese als Octets vorliegen.

2. Verschlüsseln der Daten mittels eines ausgewählten Algorithmus und einem zufälligen Key.

3. Erstellen des *EncryptedData* Elements.

4. Abspeichern des verwendeten Verschlüsselungsalgorithmus im Element *EncryptionMethod*.

5. Erstellen des *EncryptedKey* Elements mit dem Public Key verschlüsselten, symmetrischen Key. Wenn der verwendete Public Key identifiziert werden muss, wird zusätzlich das entsprechende Element *KeyInfo* im *EncryptedKey* Element erstellet.

6. Abspeichern der verschlüsselten Daten im Element *CipherData* .

7. Ersetzen des Body Inhaltes mit dem generierten *EncryptedData* Element *CipherData* .

3.5.2.3.2. Entschlüsselung

1. Entschlüsseln des symmetrischen Keys anhand des optionalen *KeyInfo* Elements im *EncryptedKey* Element.

2. Entschlüsseln der Daten im Element *CipherData* mit dem Key aus Punkt 1 und dem Algorithmus im Element *EncryptionMethod*.

3. Die Entschlüsselten Daten sind als UTF-8 codierte Zeichen zu interpretieren. Wenn das XML-Dokument, in welchem das *EncryptedData* Element vorkommt, nicht UTF-8 codiert ist, müssen die entschlüsselten Daten entsprechend umcodiert werden.

4. Ersetzen des *EncryptedData* Elements mit den entschlüsselten Elementen.

3.6. Requests

3.6.1. GetRequest Element

Über einen *GetRequest* können Informationen vom IKEv2 Daemon abgefragt werden. Folgende Informationen werden zur Zeit von strongSwan zur Verfügung gestellt und sind über den Konsolenbefehl **ipsec** abrufbar.

- **ipsec listalgs** - Alle IKE und ESP Verschlüsselungsalgorithmen, welche bei strongSwan registriert wurden.

- **ipsec listpubkeys [--utc]** - Alle von `ipsec.conf` geladenen und vom Gegenüber oder Secure DNS erhaltenen public Keys.
- **ipsec listcerts [--utc]** - Das gleiche wie **listpubkeys**, anstatt der Schlüssel werden die Zertifikate aufgelistet.
- **ipsec listcacerts [--utc]** - Alle CA Zertifikate, welche von `/etc/ipsec.d/cacerts` geladen wurden.
- **ipsec listaacerts [--utc]** - Alle Authorization Authority Zertifikate, welche von `/etc/ipsec.d/aacerts/` geladen wurden.
- **ipsec listocspcerts [--utc]** - Alle OCSP Signer Zertifikate, welche von `/etc/ipsec.d/ocspcerts/` geladen und von einem OCSP Server erhalten wurden.
- **ipsec listacerts [--utc]** - Alle X.509 Attribut Zertifikate, welche von `/etc/ipsec.d/acerts/` geladen wurden.
- **ipsec listgroups [--utc]** - Alle Gruppen Attribute, welche in `right|leftgroups` Statements in `ipsec.conf` festgelegt wurden oder in geladenen X.509 Attributen enthalten sind.
- **ipsec listcainfos [--utc]** - Alle Eigenschaften welche in den CA Bereichen in `ipsec.conf` festgelegt wurden.
- **ipsec listcrls [--utc]** - Alle CRLs, welche von `/etc/ipsec.d/crls` geladen wurden.
- **ipsec listocsp [--utc]** - Der Inhalt des OCSP Response Cache.
- **ipsec listcards [--utc]** - Alle Smartcard Einträge, welche zur Zeit benutzt werden.
- **ipsec listall [--utc]** - Alle oben aufgelisteten Informationen auf einmal.
- **ipsec status connection** - Statusinformationen über einen bestimmten Tunnel.
- **ipsec statusall** - Statusinformationen über alle Tunnel.

Alle die Uhrzeit betreffenden Informationen können entweder in lokaler Uhrzeit oder in UTC (Flag **--utc**) ausgegeben werden. Weitere Informationen über die unterstützten Managementfunktionen können auf der Homepage von strongSwan [http://www.strongswan.org] im Configuration Guide entnommen werden.

Anhand dieser Informationen und dem Quelltext von strongSwan wurden für das Management Protokoll folgende Request Elemente definiert. Es handelt sich dabei nur um die Requests, welche Informationen beschaffen die gleich sind wie die Ausgabe des **ipsec statusall** Befehls. Die Spezifikation der anderen Befehle sind noch offen für weitere Arbeiten.

```
<define name="GetRequest">
  <element name="GetRequest">
    <oneOrMore>
      <choice>
        <ref name="PerformanceRequest" />
```

```
        <ref name="InterfacesRequest" />
        <ref name="ConnectionRequest" />
        <ref name="IKESARequest" />
        <ref name="PolicyRequest" />
      </choice>
    </oneOrMore>
  </element>
</define>
```

Ein *GetRequest* Element kann wie man oben sieht aus mehreren konkreten Request Elementen bestehen. Es ist also möglich mehrere Informationen auf einmal anzufordern.

Die einzelnen Request Elemente werden nun im weiteren Verlauf erläutert.

3.6.1.1. PerformanceRequest Element

Ein *PerformanceRequest* wird verwendet um Informationen über die Performance des Security Gateways zu erhalten.

```
<define name="PerformanceRequest">
  <element name="PerformanceRequest">
    <empty/>
  </element>
</define>
```

3.6.1.2. InterfacesRequest Element

Mittels eines *InterfaceRequest* können die IP Adressen aller Interfaces, auf denen ein Security Gateway lauscht, abgefragt werden.

```
<define name="InterfacesRequest">
  <element name="InterfaceRequest">
    <empty/>
  </element>
</define>
```

3.6.1.3. ConnectionRequest Element

Um Informationen über die definierten Connections zu bekommen wird ein *ConnectionRequest* verwendet. Für das Absetzen dieses Requests wird der Name einer Connection oder das Schlüsselwort `%all`, welches alle Connections liefert, benötigt.

```
<define name="ConnectionRequest">
  <element name="ConnectionRequest">
    <attribute name="ConnectionName">
      <data type="string"/>
    </attribute>
  </element>
</define>
```

Beispiel: Abfragen von allen definierten Connections.

```
<GetRequest>
  <ConnectionRequest ConnectionName="%all"/>
</GetRequest>
```

3.6.1.4. PolicyRequest Element

Mit Hilfe von *PolicyRequest* können Informationen über Policies angefordert werden. Als Parameter muss der Name einer Policy oder das Schlüsselwort `%all`, welches alle definierten Policies liefert, angegeben werden.

```
<define name="PolicyRequest">
  <element name="PolicyRequest">
    <attribute name="PolicyName">
      <data type="string"/>
    </attribute>
  </element>
</define>
```

3.6.1.5. IKESARequest Element

Die bestehenden Sicherheitsbeziehungen und deren Informationen können über das Request Element *IKESARequest* bezogen werden. Die einzelnen Attribute haben folgende Bedeutung:

- *ConnectionName* - Nur IKE SA's, welche für die Connection mit dem Namen in *ConnectionName* gelten, werden geholt. Der Wert `%all` steht für alle Connections.
- *PolicyName* - Nur IKE SA's, welche Child SA's mit der Policy in *PolicyName* besitzen, werden geholt.
- *WithChildSA* (`true` oder `false`) - Je nach Wert des Attributes werden die Child SA's mitgeliefert oder nicht. Beim Wert `false` darf in der Antwort im Element *IKESA* kein Element *ChildSAList* vorhanden sein. Wurde zusätzlich ein *PolicyName* angegeben, werden nur Child SA's der entsprechenden Policy aufgeführt.
- *SPIInitiator* - Es werden nur IKE SA's geliefert, bei welchen die SPI des Initiators übereinstimmt.
- *SPIResponder* - Es werden nur IKE SA's geliefert, bei welchen die SPI des Responders übereinstimmt.

Falls beide SPI's gesetzt sind, werden alle IKE SA's geliefert bei welchen beide SPI's gleichzeitig auftreten.

```
<define name="IKESARequest">
  <element name="IKESARequest">
    <attribute name="ConnectionName">
      <data type="string" />
    </attribute>
    <attribute name="PolicyName">
      <data type="string" />
```

```
    </attribute>
    <attribute name="WithChildSA">
      <data type="boolean" />
    </attribute>
   <optional>
      <attribute name="SPIInitiator">
        <data type="hexBinary" />
      </attribute>
    </optional>
    <optional>
      <attribute name="SPIResponder">
        <data type="hexBinary" />
      </attribute>
    </optional>
  </element>
</define>
```

3.6.2. SetRequest Element

Über ein *SetRequest* Element kann ein Security Gateway konfiguriert werden. Ein *SetRequest* Element und somit eine übermittelte Konfiguration besteht aus *Connection* und *Policy* Elementen. Bei diesen Elementen handelt es sich um die gleichen, welche auch in einem *GetResponse* Element vorkommen können. Dabei ist die Übermittlung einer beliebigen Anzahl dieser Elemente möglich. Um die Konfiguration einer bestehenden Policy oder Connection zu ändern, wird einfach die neue Konfiguration mit dem entsprechenden *ConnectionName* oder *PolicyName* neu übermittelt. Eine Konfigurationsanfrage wird gemäss SMP entweder ganz oder gar nicht behandelt. Das *SetRequest* Element selber ist nun wie folgt definiert.

```
<define name="SetRequest">
  <element name="setRequest">
    <oneOrMore>
      <choice>
        <ref name="Connection" />
        <ref name="Policy" />
      </choice>
    </oneOrMore>
  </element>
</define>
```

Die entsprechenden Schemen für *Connection* und *Policy* sind im Abschnitt Responses unter den Punkten Connection Element und Policy Element zur finden.

3.6.3. ControlRequest

Durch einen *ControlRequest* können Steuerungsanweisungen jeglicher Art abgesetzt werden. Im Umfang dieser Diplomarbeit wurde eine Steuerungsanweisung für den Auf- und Abbau von Verbindungen definiert (Element *ShutDownRequests*). In weiteren Versionen des Protokolls können beliebig weitere Elemente hinzugefügt werden. Folgende Anforderungen werden für die Kontrolle einer Verbindung an die Steuerungsanweisung gestellt:

- Aufbauen einer Verbindung vom Gateway zu einem anderen Kommunikationspartner über den Namen einer definierten Connection.
- Beenden einer Child SA über deren IPsec Protokoll und SPI's. Es müssen die Child SA's für In- und Outbound Traffic beendet werden.
- Beenden einer IKE SA über deren SPI's (beinhaltet beenden aller Child SA's).
- Beenden aller IKE SA's und Child SA's einer Connection über deren Namen.
- Beenden aller IKE SA's und Child SA's aller Verbindungen.

Das *ControlRequest* Element ist folgendermassen definiert.

```
<define name="ControlRequest">
  <element name="ControlRequest">
    <optional>
      <ref name="ShutDownRequest" />
    </optional>
    <optional>
      <ref name="GetUPRequest" />
    </optional>
  </element>
</define>
```

Ein *ControlRequest* Element kann eine beliebige Anzahl konkreter Kontrollelemente enthalten.

3.6.3.1. Verbindungen beenden

Für den Abbau von Verbindungen ist das Schema des *ShutDownRequest* Elements wie folgt definiert.

```
<define name="ShutDownRequest">
  <element name="ShutDownRequest">
    <zeroOrMore>
      <choice>
        <ref name="ShutDownConnection" />
        <ref name="ShutDownIKESA" />
        <ref name="ShutDownChildSA" />
      </choice>
    </zeroOrMore>
  </element>
</define>
```

Über *ShutDownConnection* können die IKE SA's und die dazugehörenden Child SA's, einer oder aller Connections, heruntergefahren werden. Um alle Connections herunterzufahren kann das Schlüsselwort `%all` verwendet werden.

```
<define name="ShutDownConnection">
  <element name="ShutDownConnection">
    <attribute name="ConnectionName">
```

```
      <data type="string"/>
    </attribute>
  </element>
</define>
```

Mit *ShutDownIKESA* kann eine IKE SA über deren SPIs beendet werden.

```
<define name="ShutDownIKESA">
  <element name="ShutDownIKESA">
    <attribute name="SPIInitiator">
      <data type="hexBinary" />
    </attribute>
    <attribute name="SPIResponder">
      <data type="hexBinary" />
    </attribute>
  </element>
</define>
```

Um eine einzelne Child SA zu beenden, kann *ShutDownChildSA* mit dem IPsec Protokoll und den SPI's der Child SA verwendet werden.

```
<define name="ShutDownChildSA">
  <element name="ShutDownChildSA">
    <attribute name="Protocol">
      <choice>
        <value type="string">AH</value>
        <value type="string">ESP</value>
      </choice>
    </attribute>
    <attribute name="SPIInbound">
      <data type="hexBinary" />
    </attribute>
    <attribute name="SPIOutbound">
      <data type="hexBinary" />
    </attribute>
  </element>
</define>
```

3.6.3.2. Verbindungen aufbauen

Um eine neue Verbindung aufzubauen, kann das Element *GetUpConnection* verwendet werden. Für den Aufbau wird der Name einer Connection benötigt.

```
<define name="GetUPRequest">
  <element name="GetUPRequest">
    <zeroOrMore>
      <ref name="GetUpConnection" />
    </zeroOrMore>
  </element>
```

```
</define>

<define name="GetUpConnection">
  <element name="GetUpConnection">
    <attribute name="ConnectionName">
      <data type="string"/>
    </attribute>
  </element>
</define>
```

3.7. Responses

3.7.1. GetResponse Element

Jede *GetRequest* Message wird mit einem *GetResponse* Element, welches auch in einer *SMP-Message* gekapselt ist, beantwortet. Im Abschnitt GetRequest Element wurden die einzelnen Requests festgelegt. In diesem Kapitel werden nun die entsprechenden Responses definiert.

Um das *GetResponse* Element für die Übermittlung von statusbezogenen Informationen spezifizieren zu können, wurden Informationen von unterschiedlichen Quellen verwendet. Es waren dies: RFC 4301 Security Architecture for the Internet Protocol, RFC 4306 Internet Key Exchange (IKEv2) Protocol und die Headerdateien `child_sa.h, ike_sa.h, connection.h und policy.h` von strongSwan. Gemäss der Analyse von **ipsec statusall** und der Anwendung der genannten Quellen, kann ein *GetResponse* folgende Elemente besitzen:

- *Performance* - Kapselt Informationen bezüglich der Performance des Security Gateways.
- *Interfaces* - Beinhaltet die IP Adressen der Interfaces, auf welchen ein Security Gateway lauscht.
- *IKESA* - Repräsentiert die Sicherheitsbeziehungen für eine Connection. Beinhaltet IKE SA's und Child SA's .
- *Connection* - Repräsentiert eine Connection im Sinne von strongSwan.
- *Policy* - Repräsentiert eine Policy im Sinne von strongSwan.
- *Status* - Beinhaltet Informationen über Erfolg oder Nichterfolg des vorausgegangenen Requests.

GetResponse selbst ist in Relax NG folgendermassen definiert:

```
<define name="GetResponse">
  <element name="GetResponse">
    <ref name="Status" />
    <optional>
      <ref name="Performance" />
    </optional>
    <optional>
      <ref name="Interfaces" />
```

```
    </optional>
    <optional>
      <element name="ConnectionList">
        <zeroOrMore>
          <ref name="Connection" />
        </zeroOrMore>
      </element>
    </optional>
    <optional>
      <element name="PolicyList">
        <zeroOrMore>
          <ref name="Policy" />
        </zeroOrMore>
      </element>
    </optional>
    <optional>
      <element name="IKESAList">
        <zeroOrMore>
          <ref name="IKESA" />
        </zeroOrMore>
      </element>
    </optional>
  </element>
</define>
```

3.7.1.1. Status Element

Das *Status* Element wird verwendet um über Erfolg oder Misserfolg eines getätigten Requests zu informieren. Zu diesem Zweck steht es in jedem Response (*Get-* , *Set-* , *ControlResponse*) an erster Stelle. Der Aufbau des Elements ist wie folgt definiert.

```
<define name="Status">
  <element name="Status">
    <ref name="Code" />
    <optional>
      <element name="Message">
        <data type="string" />
      </element>
    </optional>
  </element>
</define>

<define name="Code">
   <attribute name="Code">
    <data type="string">
      <param name="pattern">[0-9]{3}</param>
    </data>
  </attribute>
</define>
```

```
<define name="Message">
  <element name="Message">
    <data type="string" />
  </element>
</define>
```

Ein Status besteht aus einem dreistelligen Status-Code, sowie einer optionalen Message. Der Text dient dabei zur einfacheren Entschlüsselung der Zahlenfolge durch den Anwender. Er beschreibt einen Error oder einen erfolgreichen Request im Genaueren. Im Gegensatz zu dem festgelegten Status-Code, ist der Wortlaut der Message nicht eindeutig definiert. Die möglichen Status-Codes sind in der folgenden Tabelle aufgelistet.

3.7.1.1.1. Status Codes

Code	Name	Erklärung
200	OK	Der Request wurde erfolgreich empfangen und behandelt.
400	Message Validation Error	Fehler beim Validieren der Nachricht.
401	XML Parsing Error	Fehler beim Parsen der XML-Nachricht.
402	Unauthorized	Der Request benötigt User-Authentifizierung mittels Signatur.
403	Unencrypted	Der Request muss zwingend verschlüsselt sein.
404	Forbidden	Es besteht keine Berechtigung für diesen Request.
405	Decryption Error	Beim Entschlüsseln der Nachricht ist ein Fehler aufgetreten.
410	Not Found	Ein im Request angefordertes Element konnte nicht gefunden werden.
411	Get Error	Beim Bearbeiten des *GetRequests* ist ein Fehler aufgetreten.
412	Set Error	Beim Bearbeiten des *SetRequests* ist ein Fehler aufgetreten.
413	Control Error	Beim Bearbeiten des *ControlRequests* ist ein Fehler aufgetreten.
500	Internal Server Error	Beim Bearbeiten des Requests ist ein interner Server-Fehler aufgetreten.
501	Not Implemented	Die gewünschte Funktion ist zur Zeit noch nicht implementiert.

3.7.1.2. Performance Element

Um die Auslastung eines VPN Gateways überwachen zu können, werden Informationen bezüglich dessen Performance benötigt. Durch das Element *Performance* werden diese Informatio-

nen dargestellt. Aus dem Element *Performance* können die folgenden Informationen gewonnen werden:

- *Threads* - Die Anzahl der vorhandenen und belegten Worker Threads.
- *QueueLoad* - Die Auslastung der Queue, welche die zu bearbeitenden Pakete beinhaltet.
- *ScheduledEvents* - Die Anzahl der geplanten Ereignisse.

```
<define name="Performance">
  <element name="Performance">
    <element name="Threads">
      <element name="NrOfThreads">
        <data type="integer" />
      </element>
      <element name="NrOfIdleThreads">
        <data type="integer" />
      </element>
    </element>
    <element name="QueueLoad">
      <data type="integer" />
    </element>
    <element name="ScheduledEvents">
      <data type="integer" />
    </element>
  </element>
</define>
```

3.7.1.3. Interfaces Element

Beim Management von VPN Gateways sind auch die IP Adressen der Interfaces, auf welchen gelauscht wird, von Interesse. Für diesen Zweck wurde das Element *Interfaces* definiert, welches die IP Adressen dieser Interfaces kapselt. Als Wert sind eine oder mehrere IPv4 oder IPv6 Adressen erlaubt. Das Relax NG Schema besitzt folgenden Aufbau.

```
<define name="Interfaces">
  <element name="Interfaces">
    <zeroOrMore>
      <ref name="IPv4Address" />
    </zeroOrMore>
    <zeroOrMore>
      <ref name="IPv6Address" />
    </zeroOrMore>
  </element>
</define>
```

3.7.1.4. IKESA Element

Eine IKE SA wird benötigt um eine Sicherheitsbeziehung (Child SA) zwischen zwei Kommunikationspartnern auszuhandeln. Ob dafür eine neue IKE SA verwendet werden muss ist davon

abhängig, ob die beteiligten Hosts bereits von einer IKE SA abgedeckt werden (dies ist abhängig von den IP Adressen der Hosts). Die Informationen, welche für das Aushandeln einer neuen IKE SA benötigt werden, stammen aus der einer IKE SA zugehörigen Connection (nicht Connection in `ipsec.conf`). Alle über eine IKE SA erzeugten Child SA's werden durch die IKE SA verwaltet. Dieser Zusammenhalt wird auch von SMP beachtet.

Durch das Element *IKESA* wird eine IKE SA gekapselt. In ihr sind grundsätzlich die Informationen vorhanden, welche auch aus der Ausgabe des Befehls **ipsec statusall** gewonnen werden können. Bei der Definition der einzelnen Typen (Algorithmen, ID's etc.) wurden die Sourcedateien von strongSwan als auch das RFC 4306 (IKEv2) zu Hande gezogen. In SMP ist eine IKE SA wie folgt definiert.

```
<define name="IKESA">
  <element name="IKESA">
    <ref name="ConnectionName" />
    <ref name="IKEStatus" />
    <ref name="IKERole" />
    <ref name="LocalID" />
    <ref name="RemoteID" />
    <ref name="LocalAndRemoteAddressType" />
    <ref name="LocalAndRemotePortType" />
    <ref name="IKESPIPairType" />
    <ref name="NATTraversal" />
    <ref name="IKEEncryptionAlgorithm" />
    <ref name="IKEIntegrityAlgorithm" />
    <ref name="DiffieHellmanGroup" />
    <element name="ChildSAList">
      <zeroOrMore>
        <ref name="ChildSA" />
      </zeroOrMore>
    </element>
  </element>
</define>
```

Da eine IKE SA anhand einer Connection erstellt wird, besitzt eine *IKESA* das Element *ConnectionName* welches diese Beziehung ausdrückt.

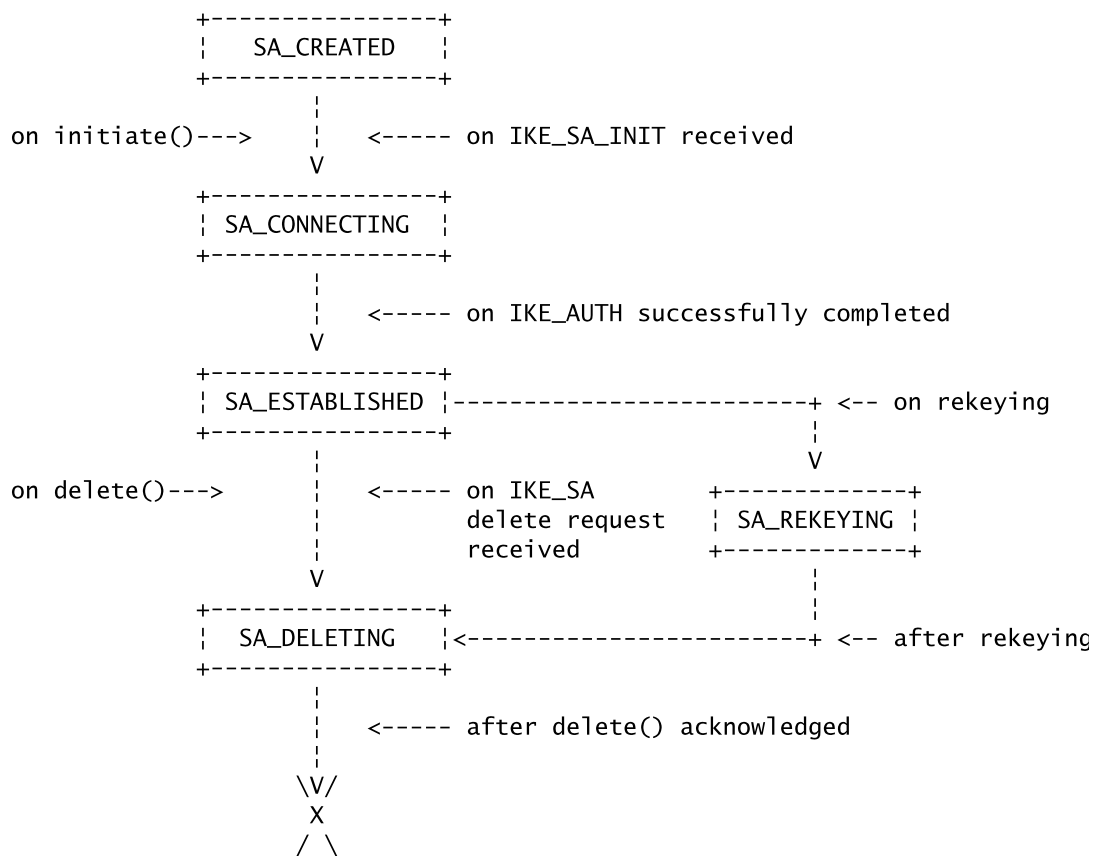
3.7.1.4.1. IKEStatus Element

Gemäss der Implementierung des IKEv2 Daemons von strongSwan besitzt eine vorhandene IKE SA einen Status, welcher durch das Element *IKEStatus* repräsentiert wird. Die dafür möglichen Werte stammen aus der Datei `ike_sa.h` der strongSwan Sourcen.

```
<define name="IKEStatus">
  <element name="IKEStatus">
    <choice>
      <value type="string">CREATED</value>
      <value type="string">CONNECTING</value>
      <value type="string">ESTABLISHED</value>
      <value type="string">REKEYING</value>
```

```
      <value type="string">DELETING</value>
    </choice>
  </element>
</define>
```

Wann welcher Status existieren kann, wird durch die folgende Grafik, welche ebenfalls aus der Datei `ike_sa.h` stammt, erläutert:



Statusübergänge einer IKE SA

3.7.1.4.2. IKERole Element

Beim Aushandeln einer IKE SA ist immer einer der Kommunikationspartner der Initiator - derjenige, der die Verbindung initiiert - und einer der Responder. Beim Management der IKE SA's auf einem Security Gateway ist es nun von Interesse, ob es sich beim Security Gateway um den Initiator oder Responder handelt. Diese Information kann über das Element *IKERole* repräsentiert werden, welches als möglichen Wert die zwei Zustände `INITIATOR` und `RESPONDER` erlaubt.

3.7.1.4.3. LocalID/RemoteID Element

Sowohl der Initiator als auch der Responder besitzen eine `IKE_ID` über welche sie eindeutig identifiziert werden können. Diese ID's, welche durch die Elemente *LocalID* und *RemoteID* repräsentiert werden, können gemäss RFC 4306 (Internet Key Exchange (IKEv2) Protocol) einen unterschiedlichen Typ aufweisen. Im Strong Management Protocol wurden sämtliche im RFC festgelegten Typen definiert.

- *IPv4Address* - Eine beliebige IPv4 Adresse.
- *IPv6Address* - Eine beliebige IPv6 Adresse.
- *FQDN* - Ein fully qualified Domain Name (z.B. www.strongswan.org).
- *EmailAddress* - Ein Benutzer spezifischer Domain Name (z.B. alice@strongswan.org).
- *ASN1DN* - Ein X.500 distinguished Name (z.B. cn=Jeff Michels, ou=People, o=strongswan, c=org).
- *ASN1GN* - Ein X.500 General Name.
- *PropTypeIdentification* - Irgendeine herstellerspezifische proprietäre ID.
- *AnyID* - ID für den Wert %any. Diese ID wird nur für Policies benötigt.

3.7.1.4.4. Local-/RemoteAddress Elemente

Die IP Adressen der Kommunikationspartner zwischen welchen eine Sicherheitsbeziehung besteht, werden durch die Elemente *LocalAddress* und *RemoteAddress* dargestellt. Als Wert sind sowohl IPv4 als auch IPv6 Adressen erlaubt.

```
<define name="LocalAndRemoteAddressType">
    <choice>
      <group>
        <element name="LocalAddress">
          <ref name="IPv6Address" />
        </element>
        <element name="RemoteAddress">
          <ref name="IPv6Address" />
        </element>
      </group>
      <group>
        <element name="LocalAddress">
          <ref name="IPv4Address" />
        </element>
        <element name="RemoteAddress">
          <ref name="IPv4Address" />
        </element>
      </group>
    </choice>
  </define>
```

3.7.1.4.5. Local-/Remote Port Elemente

Beinhaltet die Portnummern, welche zur Kommunikation verwendet werden. Normalerweise ist das der UDP Port 500 oder 4500 bei NAT-Traversal auf Seiten des Responders und irgend eine hohe Portnummer auf Seiten des Initiators.

```
<define name="LocalAndRemotePortType">
```

```
  <element name="LocalPort">
    <ref name="PortType" />
  </element>
  <element name="RemotePort">
    <ref name="PortType" />
  </element>
</define>

<define name="PortType">
  <data type="positiveInteger">
    <param name="maxInclusive">65535</param>
  </data>
</define>
```

3.7.1.4.6. SPIInitiator/SPIResponder Elemente

Die IKE SPI des Initiator und des Responder, welche verwendet werden um eine IKE SA zu identifizieren, werden durch das Element *IKESPIPair* repräsentiert. Die SPIs werden als Hex Werte übertragen.

```
<define name="IKESPIPairType">
  <element name="SPIInitiator">
    <data type="hexBinary" />
  </element>
  <element name="SPIResponder">
    <data type="hexBinary" />
  </element>
</define>
```

3.7.1.4.7. NATTraversal Element

Ob eine IKE SA NAT Traversal verwendet, wird durch das Element *NATTraversal* dargestellt. Die Werte sind als Boolean abgelegt.

3.7.1.4.8. DiffieHellmanGroup Element

Für den Aufbau einer IKE SA werden sowohl Algorithmen für die Verschlüsselung als auch für die Sicherung der Integrität benötigt. Um `IKE_AUTH` zu sichern wird das Diffie Hellman Verfahren verwendet. strongSwan unterstützt folgende Variationen.

```
<define name="DiffieHellmanGroupType">
  <choice>
    <value type="string">MODP_1024_BIT</value>
    <value type="string">MODP_1536_BIT</value>
    <value type="string">MODP_2048_BIT</value>
    <value type="string">MODP_4096_BIT</value>
    <value type="string">MODP_8192_BIT</value>
  </choice>
</define>
```

3.7.1.4.9. IKEEncryptionAlgorithm

Bei `IKE_SA_INIT` werden nebst den Diffie Hellman Komponenten Algorithmen für die Verschlüsselung von `CREATE_CHILD_SA` ausgehandelt. Diese Algorithmen sind:

```
<define name="IKEEncryptionAlgorithmType">
  <choice>
    <value type="string">3DES</value>
    <value type="string">AES_CBC-128</value>
  </choice>
</define>
```

3.7.1.4.10. IKEIntegrityAlgorithm

Beim Aushandeln einer IKE SA muss die Datenintegrität gewährleisten sein. Dafür werden in strongSwan die folgenden Algorithmen unterstützt.

```
<define name="IntegrityAlgorithmType">
  <choice>
    <value type="string">HMAC_MD5_96</value>
    <value type="string">HMAC_SHA1_96</value>
  </choice>
</define>
```

3.7.1.5. ChildSA Element

Durch eine IKE SA können mehrere Child SA's zwischen zwei Kommunikationspartnern verwaltet werden. Eine IKE SA besitzt also, wie bei **ipsec statusall** ersichtlich ist, eine oder mehrere Child SA. Eine Child SA beschreibt die eigentliche Sicherheitsbeziehung zwischen zwei Kommunikationspartnern. In ihr sind z.B. die verwendeten Algorithmen für das IPsec Protokoll festgelegt. Durch das Element *ChildSA* wird eine Child SA repräsentiert. Das Element selber wird dabei immer in einem *IKESA* Element platziert. Die Werte der in *ChildSA* definierten Elemente entsprechen weitgehend den Informationen der **ipsec statusall** Ausgabe. Es ergibt sich folgendes Relax NG Schema.

```
<define name="ChildSA">
  <element name="ChildSA">
    <ref name="PolicyName" />
    <ref name="ChildStatus" />
    <ref name="ReqID" />
    <ref name="ChildSPIPairType" />
    <ref name="RekeyingTime" />
    <ref name="DeletingTime" />
    <ref name="TrafficSelectorsLocalSite" />
    <ref name="TrafficSelectorsRemoteSite" />
    <ref name="LastUse" />
    <ref name="IPsecProtocolMode" />
    <choice>
      <ref name="AHIntegrityAlgorithm" />
```

```
      <group>
        <ref name="ESPIntegrityAlgorithm" />
        <ref name="ESPEncryptionAlgorithm" />
      </group>
    </choice>
  </element>
</define>
```

3.7.1.5.1. PolicyName Element

Eine Child SA wird durch eine Policy beschrieben. Diese Beziehung wird durch das Element *PolicyName* ausgedrückt.

3.7.1.5.2. ChildStatus Element

Genau wie die IKE SA besitzt auch die Child SA in strongSwan einen Status, welcher mit Hilfe des Elements *ChildStatus* übertragen werden kann. Die möglichen Werte des Elements *ChildStatus* entsprechen denjenigen aus der Sourcecodedatei `child_sa.h`.

- `CREATED` - Die Child SA wurde erstellt aber noch nicht in der SAD installiert.
- `ROUTED` - Eine Security Policy ist vorhanden, aber keine Child SA wurde in der SAD installiert.
- `INSTALLED` - Die Child SA wurde in die SAD installiert und ist in Verwendung.
- `REKEYING` - Das Rekeying für Child SA wird gerade durchgeführt.
- `DELETING` - Die Child SA wird gerade gelöscht.

3.7.1.5.3. ReqID Element

ReqID beinhaltet eine ID, welche eine Child SA im Kernel identifiziert.

```
<define name="ReqId">
  <element name="ReqId">
    <data type="integer" />
  </element>
</define>
```

3.7.1.5.4. SPIInbound/SPIOutbound Elemente

Die SPI identifiziert mit einer IP Adresse und einem IPsec Protokoll Typ eine Sicherheitsbeziehung (Child SA). Im Falle der **ipsec statusall** Ausgabe werden für die Child SA zwei SPI's dargestellt. Dies ist so, weil für In- und Outbound Traffic jeweils eine SA existiert. Bei der Ausgabe mittels **ipsec statusall** werden diese SA's aber nur als eine dargestellt.

```
<define name="ChildSPIPairType">
  <element name="SPIInbound">
    <data type="hexBinary" />
```

```
  </element>
  <element name="SPIOutbound">
    <data type="hexBinary" />
  </element>
</define>
```

3.7.1.5.5. RekeyingTime/DeletingTime Elemente

Den Elementen *RekeyingTime* und *DeletingTime* kann entnommen werden, wie lange es noch dauert (in Sekunden) bis für eine Child SA ein neues Rekeying stattfindet oder diese gelöscht wird.

3.7.1.5.6. TrafficselectorsLocalSite/TrafficselectorsRemoteSite Elemente

Für eine Sicherheitsbeziehung können die in einer Policy definierten Trafficselektoren ausgehandelt werden. Durch diese Trafficselektoren kann der Verkehr, welcher über einen Tunnel zugelassen wird, auf einfache Weise beschränkt werden. Gemäss RFC 4306 kann eine Child SA mehrere Trafficselektoren besitzen. Der Aufbau eines Trafficselektors ist in RFC 4306 definiert. Im Strong Management Protocol wurde dieser Aufbau übernommen. Ein Trafficselektor besteht somit aus folgenden Bestandteilen.

- *Protocol* - Legt das verwendete IP Protokoll anhand einer ID fest.
- *StartAddress/EndAddress* - Legt einen IP Adressenbereich fest, welcher von diesem Selektor erlaubt ist. Die angegebenen IP Adressen gehören ebenfalls zum erlaubten Bereich.
- *StartPort/EndPort* - Legt einen Portbereich fest, welcher von diesem Selektor erlaubt ist. Die angegebenen Portnummern gehören ebenfalls zum erlaubten Bereich.

```
<define name="TrafficSelector">
  <element name="TrafficSelector">
    <optional>
      <element name="Protocol">
        <data type="integer" />
      </element>
    </optional>
    <choice>
      <group>
        <element name="StartAddress">
          <ref name="IPv6Address" />
        </element>
        <element name="EndAddress">
          <ref name="IPv6Address" />
        </element>
      </group>
      <group>
        <element name="StartAddress">
          <ref name="IPv4Address" />
```

```
        </element>
        <element name="EndAddress">
          <ref name="IPv4Address" />
        </element>
      </group>
    </choice>
    <optional>
      <element name="StartPort">
        <ref name="PortType" />
      </element>
    </optional>
    <optional>
      <element name="EndPort">
        <ref name="PortType" />
      </element>
    </optional>
  </element>
</define>
```

3.7.1.5.7. LastUse Element

Das Element *LastUse* beinhaltet die Dauer der letzten Verwendung, sowie die Anzahl der von der Child SA bearbeiteten Bytes für In- und Outbound Traffic.

```
<define name="LastUse">
  <element name="LastUse">
    <element name="Inbound">
      <ref name="Duration" />
      <ref name="ByteCount" />
    </element>
    <element name="Outbound">
      <ref name="Duration" />
      <ref name="ByteCount" />
    </element>
  </element>
</define>
```

3.7.1.5.8. IPsec Protokoll, Modus und Algorithmen

Eine Child SA zwischen zwei Kommunikationspartnern legt immer fest, welches IPsec Protokoll (AH oder ESP) und welcher Mode (Tunnel, Transport) verwendet werden. Damit diese Informationen auch in SMP übertragen werden können, wurde die Element *IPSecProtocolMode* , *AHIntegrityAlgorithm* , *ESPIntegrityAlgorithm* und *ESPEncryptionAlgorithm* festgelegt.

```
<define name="IPsecProtocolMode">
  <element name="IPsecProtocolMode">
    <choice>
      <value type="string">TRANSPORT</value>
      <value type="string">TUNNEL</value>
```

```
    </choice>
  </element>
</define>
```

In strongSwan können für die Verschlüsselung im ESP Header folgende Algorithmen verwendet werden.

```
<define name="EncryptionAlgorithmType">
  <choice>
    <value type="string">3DES</value>
    <value type="string">AES_CBC-128</value>
    <value type="string">AES_CBC-192</value>
    <value type="string">AES_CBC-256</value>
    <value type="string">BLOWFISH-256</value>
  </choice>
</define>
```

Für die Sicherstellung der Integrität sind folgende Algorithmen erlaubt.

```
<define name="IntegrityAlgorithmType">
  <choice>
    <value type="string">HMAC_MD5_96</value>
    <value type="string">HMAC_SHA1_96</value>
  </choice>
</define>
```

3.7.1.6. Connection Element

Das *Connection* Element kapselt eine Connection im Sinne von strongSwan (nicht Connection in `ipsec.conf`). Eine Connection wird für das Aushandeln einer IKE SA benötigt. Sie besteht aus einem Namen, den Adressen der beteiligten Kommunikationspartner und einigen weiteren Elementen, welche im Verlaufe dieses Abschnitts noch erläutert werden.

Das Relax NG Schema einer Connection sieht folgendermassen aus.

```
<define name="Connection">
  <element name="Connection">
    <ref name="Name" />
    <ref name="LocalAndRemoteAddressType" />
    <ref name="LocalAndRemotePortType" />
    <ref name="RekeyingTime" />
    <ref name="DeletingTime" />
    <ref name="DPDDelay" />
    <ref name="RetransmissionSequences" />
    <ref name="CertificateRequestPolicy" />
    <ref name="CertificateResponsePolicy" />
    <ref name="IKEEncryptionAlgorithmList" />
    <ref name="IKEIntegrityAlgorithmList" />
    <ref name="DiffieHellmanGroupList" />
  </element>
```

```
</define>
```

Das Element *Connection* wurde anhand der Analyse von **ipsec statusall** und der Headerdatei `connection.h`, welche aus den Sourcen von Charon stammt, definiert. Grundsätzlich ist *Connection* so aufgebaut, dass es sowohl in einem *GetResponse* als auch in einem *SetResponse* verwendet werden kann. Es ist also möglich das *Connection* Element für den Bezug von Informationen und zur Konfiguration zu verwenden. Falls eine Connection erweitert werden muss, lässt sich dies einfach bewerkstelligen, indem ein zusätzliches Element dem *Connection* Schema hinzugefügt wird. Die zur Zeit vorhandenen Elemente besitzen folgende Bedeutung:

- *ConnectionName* - Attribut, welches den Namen der Connection beinhaltet.
- *LocalAndRemoteAddressType* - Beinhaltet die IP Adressen der Kommunikationspartner zwischen welchen eine IKE SA aufgebaut werden soll. Zulässig sind die beiden Typen IPv4 und IPv6.
- *LocalAndRemotePortType* - Beinhaltet die Portnummern, welche zur Kommunikation verwendet werden. Normalerweise ist das der UDP Port 500 oder 4500 bei NAT-Traversal.
- *RekeyingTime* - Die Dauer (in Sekunden) bis die Sessionschlüssel neu ausgehandelt werden.
- *DeletingTime* - Die Dauer (in Sekunden) bis die IKE SA gelöscht wird.
- *DPDDelay* - Während dem Betrieb kann überprüft werden ob der Remote Host noch erreichbar ist (Dead Peer Detection). Durch *DPDDelay* wird festgelegt nach wie vielen Sekunden ein Security Gateway dies tun soll.
- *RetransmissionSequences* - Falls auf die Anzahl in *RetransmissionSequences* festgelegten Meldungen vom Remote Host keine Antwort kommt, wird dieser als beendet angenommen.
- *CertificateRequestPolicy* - Legt die Policy für das Senden des Zertifikates vom Remote Host fest, sprich ob dieser sein Zertifikat übertragen muss.
- *CertificateResponsePolicy* - Legt die Policy für das Senden des Zertifikates durch den Local Host fest, sprich ob dieser sein Zertifikat senden muss. Für *CertificateResponsePolicy* als auch *CertificateRequestPolicy* werden die Policies `ALWAYS_SEND`, `SEND_IF_ASKED` und `NEVER_SEND` unterstützt.
- *IKEEncryptionAlgorithmList* - Beinhaltet die unterstützten Verschlüsselungsalgorithmen für den IKE SA Traffic. Es sind folgende möglich: `3DES`, `AES_CBC-128`.
- *IKEIntegrityAlgorithmList* - Stellt eine Auswahl von Algorithmen zur Verfügung, welche zur Sicherung der Integrität der Daten verwendet werden kann. Zur Auswahl stehen die beiden Algorithmen `HMAC_MD5_96` und `HMAC_SHA1_96`.
- *DiffieHellmanGroupList* - Stellt eine Auswahl von Diffie Hellman Gruppen bereit, welche für den Aufbau der IKE SA zur Verfügung stehen. Es werden die Algorithmen `MODP_1024_BIT` bis `MODP_8192_BIT` unterstützt.

Da die hier definierten Elemente grösstenteils auch im Element *IKESA* verwendet werden und im Abschnitt IKESA Element ausreichend dokumentiert wurden, wird hier auf eine detail-

lierte Auflistung verzichtet. Für weitere Informationen sei daher auf das Relax NG Schema `connection.xml` und den Abschnitt IKESA Element verwiesen.

3.7.1.7. Policy

Das Element *Policy* kapselt eine Policy im Sinne von strongSwan. Dies ist nicht zu verwechseln mit einer Policy wie sie in der SPD (Security Policy Database) des Kernels abgelegt ist. Eine Policy in der Kernel SPD basiert aber auf einer strongSwan Policy. Im Verlauf dieses Abschnitts wird unter einer Policy immer eine Policy im Sinne von strongSwan verstanden. Eine Policy besteht aus einem Namen, Trafficselektoren und einigen weiteren Elementen, welche für das Aushandeln einer Child SA benötigt werden. Die Policy dient dabei als Richtlinie. Das Relax NG Schema der Policy hat folgenden Aufbau.

```
<define name="Policy">
  <element name="Policy">
    <ref name="PolicyName" />
    <ref name="LocalID" />
    <ref name="RemoteID" />
    <ref name="AuthenticationMethod" />
    <ref name="RekeyingTime" />
    <ref name="DeletingTime" />
    <ref name="DPDAction" />
    <ref name="TrafficSelectorsLocalSite" />
    <ref name="TrafficSelectorsRemoteSite" />
    <ref name="LocalCertificateAuthority" />
    <ref name="RemoteCertificateAuthority" />
    <ref name="AHIntegrityAlgorithmList" />
    <ref name="ESPEncryptionAlgorithmList" />
    <ref name="ESPIntegrityAlgorithmList" />
  </element>
</define>
```

Das Element *Policy* wurde anhand der Analyse von **ipsec statusall**, dem RFC 4301 (Appendix C: ASN.1 for an SPD Entry) und der Headerdatei `policy.h`, welche aus den Sourcen von Charon stammt, definiert. Grundsätzlich ist eine Policy so aufgebaut, dass es sowohl in einem *GetResponse* als auch in einem *SetResponse* Element verwendet werden kann. Es ist also möglich das *Policy* Element für den Bezug von Informationen und für die Konfiguration zu verwenden. Falls eine Policy erweitert werden muss, lässt sich dies einfach bewerkstelligen indem ein zusätzliches Element dem Policy Schema (`policy.h`) hinzugefügt wird. Die zur Zeit vorhandenen Elemente besitzen folgende Bedeutung:

- *Name* - Attribut, welches den Namen der Policy beinhaltet.
- *LocalID* - ID des Local Hosts, welche zur Identification benötigt wird. Folgende ID Typen sind zulässig: *IPv4Address, IPv6Address, FQDN, EmailAddress, ASN1DN, ASN1GN, PropTypeIdentification* (Proprietäre ID) und *AnyID*.
- *RemoteID* - ID des Remote Hosts, welche zur Identification benötigt wird. Die unterstützten ID-Typen sind die gleichen wie bei *LocalID*.

- *AuthenticationMethod* - Beinhaltet die verwendete Methode für die Authentifizierung (`auth data`). Die möglichen Verfahren sind: `RSA_DIGITAL_SIGNATURE`, `SHARED_KEY_MESSAGE_INTEGRITY_CODE` und `DSS_DIGITAL_SIGNATURE`.

- *RekeyingTime* - Die Dauer (in Sekunden) bis die Sessionschlüssel der Child SA neu ausgehandelt werden müssen.

- *DeletingTime* - Die Dauer (in Sekunden) bis die Child SA gelöscht wird.

- *DPDAction* legt fest, was geschieht falls ein Remote Host als tot erklärt wird. Die möglichen Verfahren sind: `NONE` (DPD ausgeschaltet), `CLEAR` (Child SA wird gelöscht), `ROUTE` (Resetup der Child SA falls benötigt) und `RESTART` (erstellen einer neuen Child SA unter einer neuen IKE SA).

- *TrafficSelectorsLocalSite* - *TrafficSelectorsLocalSite* beinhaltet die definierten Trafficselektoren des Local Host.

- *TrafficSelectorsRemoteSite* - *TrafficSelectorsRemoteSite* beinhaltet die definierten Trafficselektoren des Remote Host. Gemäss dessen Definition besteht ein Trafficselektor aus IP Adressen und Ports. Aus diesen kann eine Range gebildet werden, welche beispielsweise der `leftsubnet` Angabe in `ipsec.conf` entspricht.

- *LocalCertificateAuthority* - Beinhaltet die CA des Local Host. Ist üblicherweise der `Issuer` des mit `leftcert` angegebenen Zertifikats.

- *RemoteCertificateAuthority* - Beinhaltet die CA des Remote Host. Damit der Remote Host eine SA aufbauen darf, muss sein Zertifikat von der hier definierten CA signiert worden sein.

- *AHIntegrityAlgorithmList* - Stellt eine Auswahl von Algorithmen zur Verfügung, welche zur Authentifizierung im IPsec Protokoll AH verwendet werden können. Die unterstützten Algorithmen sind: `HMAC_MD5_96` und `HMAC_SHA1_96`.

- *ESPEncryptionAlgorithmList* - Stellt eine Auswahl von Algorithmen zur Verfügung, welche zur Verschlüsselung im IPse Protokoll ESP verwendet werden können. Die unterstützten Algorithmen sind: `3DES`, `AES_CBC-128`, `AES_CBC-192`, `AES_CBC-512` und `BLOWFISH-256`.

- *ESPIntegrityAlgorithmList* - Stellt eine Auswahl von Algorithmen zur Verfügung, welche zur Authentifizierung im IPsec Protokoll ESP verwendet werden können. Die unterstützten Algorithmen sind: `HMAC_MD5_96` und `HMAC_SHA1_96`.

Da die hier definierten Elemente grösstenteils auch im Element *ChildSA* verwendet werden und im Abschnitt IKESA Element ausreichend dokumentiert wurden, wird hier auf eine detaillierte Auflistung verzichtet. Für mehr Informationen sei auf das Relax NG Schema `policy.xml` und den Abschnitt IKESA Element verwiesen.

3.7.2. SetResponse Element

SetResponse ist die Antwort auf ein *SetRequest* . Zur Zeit besteht dieses Element nur aus einem *Status* Element über welches der Erfolg bzw. Nichterfolg des *SetRequests* mitgeteilt wird.

```
<define name="SetResponse">
  <element name="SetResponse">
    <ref name="Status"/>
  </element>
</define>
```

3.7.3. ControlResponse Element

ControlResponse ist die Antwort auf ein *ControlRequest* . Zur Zeit besteht dieses Element wie auch das *SetResponse* Element nur aus einem *Status* Element, über welches der Erfolg bzw. Nichterfolg des *ControlRequests* ersichtlich ist.

```
<define name="ControlResponse">
  <element name="ControlResponse">
    <ref name="Status"/>
  </element>
</define>
```

3.8. Notification Element

Das *Notification* Element wird verwendet um eine Managementkonsole über unerwartete Ereignisse wie z.B. Fehler zu informieren. Im Gegensatz zu den Request und Response Elementen folgt auf eine Notification keine Antwort. Eine Notification-Message kann nur von einem VGMA versand werden.

```
<define name="Notification">
  <element name="Notification">
    <ref name="Code"/>
    <element name="DateTime">
      <data type="dateTime" />
    </element>
    <optional>
      <ref name="Message"/>
    </optional>
  </element>
</define>
```

Ein *Notification* Element besteht aus einem dreistelligen Notification-Code sowie einer optionalen Message. Diese Elemente entsprechen den gleichnamigen Staus Elementen einer Response-Message. Die *Message* dient dabei der einfacheren Entschlüsselung der Zahlenfolge durch den Anwender. Sie beschreibt das aufgetretene Ereignis im Genaueren. Im Gegensatz zu dem festgelegten Notification-Code, ist der Wortlaut der Message nicht eindeutig definiert. Zur Zeit werden vom Strong Management Protocol keine Notification-Codes festgelegt.

Ein Beispiel einer Notification-Message könnte wie folgt aussehen.

```
<Notification Code="400">
  <DateTime>2006-11-30T07:03:20</DateTime>
  <Message>
```

```
    Connection net-net1 has terminated unexpectedly.
  </Message>
</Notification>
```

Kapitel 4. Anforderungen strongManager

Inhaltsverzeichnis

4.1. Einleitung

Um die Anforderungen für die Managementkonsole zu finden, wurden User Stories erstellt. Eine User Story beschreibt eine Funktionalität die ein System aus Sicht des Benutzers aufweisen muss und besteht üblicherweise aus ein bis zwei Sätzen.

Mehr über User Stories und dass von uns verwendete Template können den Webseiten http://c2.com/cgi/wiki?UserStory und http://c2.com/cgi/wiki?UserStoryTemplate entnommen werden.

4.2. User Stories

Für die Managementkonsole von strongSwan wurden die unten aufgeführten User Stories definiert. Der Standardbenutzer ist hierbei der **VPNGatewayAdministrator**. Dieser ist ein Benutzer mit guten IT und strongSwan Kenntnissen. Zur Verbesserung der Projektplanung wurde des Weiteren jeder User Story eine Priorität (hoch, mittel oder tief) zugewiesen. Die Höhe der Priorität bestimmt dabei den Zeitpunkt der Implementierung.

4.2.1. Auflistung aller Gateways

Priorität: mittel

Als ein VPNGatewayAdministrator **will ich** alle zu administrierenden strongSwan VPN Gateways angezeigt bekommen, **so dass** ich einen Überblick über die Anzahl der zu managenden VPN Gateways gewinnen kann.

4.2.2. Authentication

Priorität: mittel

Als ein VPNGatewayAdministrator **will ich** mich auf der Webapplikation authentifizieren können **so dass** keine unbefugte Person Zugriff auf die Managementapplikation erhält.

4.2.3. Auflistung Performance

Priorität: tief

Als ein VPNGatewayAdministrator **will ich** Informationen bezüglich der Performance eines Security Gateways, **so dass** ich mir ein Bild der Auslastung machen und allenfalls nötige Massnahmen ergreifen kann.

4.2.4. Auflistung Interfaces

Priorität: tief

Als ein VPNGatewayAdministrator **will ich** alle IP Adressen der Interfaces angezeigt bekommen auf welchen ein Security Gateway lauscht, **so dass** ich dadurch die Konfiguration der Interfaces überprüfen kann.

4.2.5. Auflistung Connections

Priorität: hoch

Als ein VPNGatewayAdministrator **will ich** die auf einem entsprechenden VPN Gateway konfigurierten Connections aufgelistet bekommen, **so dass** ich einen Überblick über die Anzahl vorhandenen Connections gewinnen kann.

4.2.6. Details zu Connection

Priorität: mittel

Als ein VPNGatewayAdministrator **will ich** die für eine Connection konfigurierten Details angezeigt bekommen, **so dass** ich sehen kann welche Hosts eine Connection aufbauen können.

4.2.7. Auflistung Policies

Priorität: mittel

Als ein VPNGatewayAdministrator **will ich** die auf einem entsprechenden VPN Gateway konfigurierten Policies aufgelistet bekommen, **so dass** ich einen Überblick über die Anzahl vorhandenen Policies gewinnen kann.

4.2.8. Details zu Policy

Priorität: mittel

Als ein VPNGatewayAdministrator **will ich** die für eine Connection konfigurierten Details angezeigt bekommen, **so dass** ich sehen kann welcher Traffic erlaubt ist.

4.2.9. IKE SA's zu Connections

Priorität: hoch

Als ein VPNGatewayAdministrator **will ich** alle für eine Verbindung bestehenden IKE SA's aufgelistet bekommen, **so dass** ich in die zur Zeit bestehenden Verbindungen Einblick gewinnen kann.

4.2.10. Details zu IKE SA

Priorität: hoch

Als ein VPNGatewayAdministrator **will ich** die Details von einer IKE SA angezeigt bekommen, **so dass** ich die Konfiguration einer Connection überprüfen kann.

4.2.11. CHILD SA's zu IKE SA

Priorität: hoch

Als ein VPNGatewayAdministrator **will ich** alle auf einer IKE SA ausgehandelten CHILD SA's aufgelistet bekommen, **so dass** ich einen Überblick über die zur Zeit ausgehandelten Tunnel erhalten kann.

4.2.12. Details zu CHILD SA

Priorität: hoch

Als ein VPNGatewayAdministrator **will ich** die Details von einer CHILD SA aufgelistet bekommen, **so dass** ich die Konfiguration einer Policy überprüfen kann.

Kapitel 5. Design strongManager

Inhaltsverzeichnis

5.1. Einleitung

strongManager ist die Implementation der Managementkonsole. Er umfasst die Kommunikationsschnittstelle, das Model und die Webapplikation. Nebst den Komponenten von strongManager werden in diesem Kapitel die verwendeten Technologien und das Design des VPN Gateway Emulators erläutert.

Bemerkung zu Codebeispielen

In den Codebebeispielen sind alle von Hand erstellten Zeilenumbrüche mit zwei Leerschlägen eingerückt (wegen Platzmanagel). Pythonfragemte sind hingegen mit vier Leerschlägen eingerückt.

5.2. Technologien

Für die Implementierung der Managementkonsole werden mehrere Komponenten benötigt.

- XML-Bibliothek für die Verarbeitung des Strong Management Protocols.
- Templateengine um Python Code in HTML Seiten auszuwerten.
- Bibliothek für die Verarbeitung mit FastCGI, welche Webserverunabhängigkeit schafft.

5.2.1. Verwendete Komponenten

5.2.1.1. XML Bibliothek

Für die Verarbeitung von SMP wird die XML-Bibliothek libxml2 von xmlsoft [http://xmlsoft.org] verwendet. libxml2 ist ein XML-C-Parser für welchen Bindings für viele Programmiersprachen existieren. libxml2 besitzt folgende für uns wichtigen Vorteile:

- libxml2 ist schnell
- unterstützt Relax NG Schema Validation
- unterstützt Canonical XML (XML-C14N wird für XML-Signature und XML-Encryption benötigt)

5.2.1.2. Python Binding

Das bei libxml2 mitgelieferte Python Binding ist schlecht dokumentiert und unterstützt nur DOM und SAX. Als Binding wurde aus diesem Grund das Python optimierte lxml von codespeak [http://codespeak.net/lxml] ausgesucht. lxml ist gut dokumentiert und bietet viel Komfort.

5.2.1.3. Templateengine

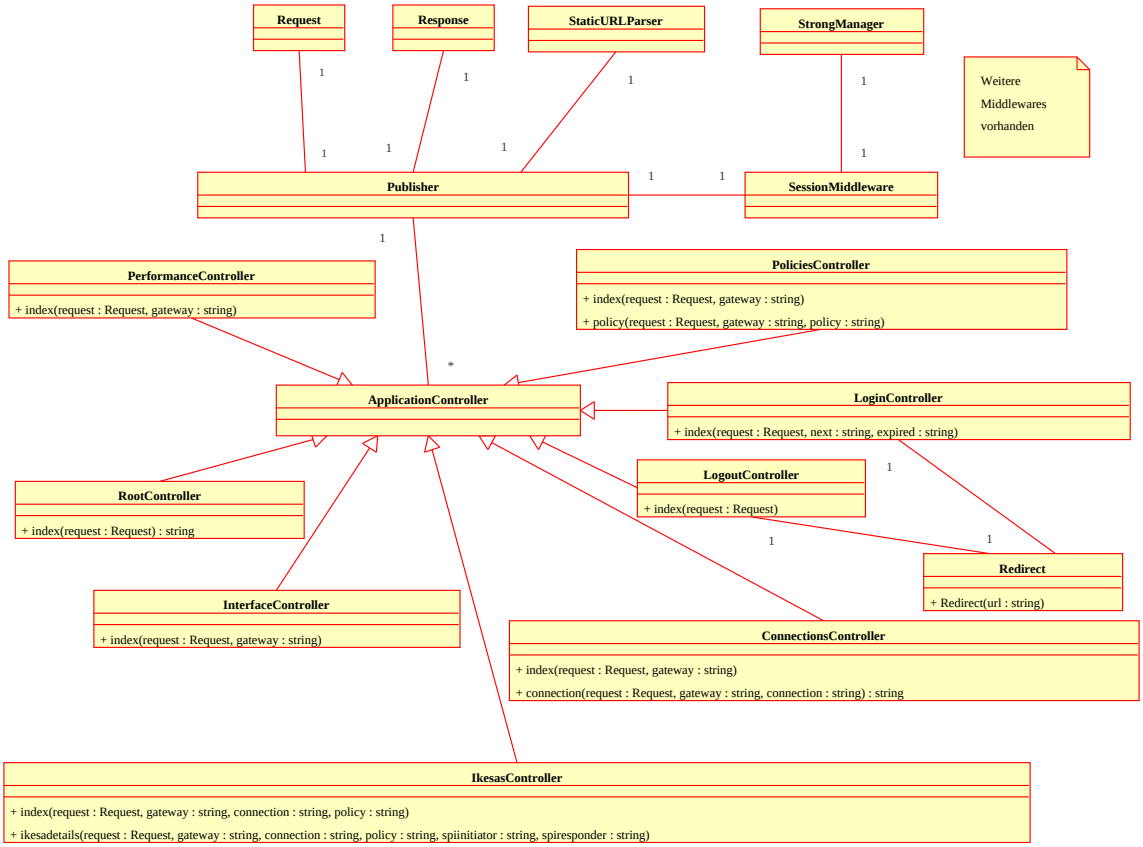
Als Templateengine kommt Genshi von edgewall [http://genshi.edgewall.org] zum Einsatz. Genshi ist eine Phyton Bibliothek, welche Komponenten für das Parsen, Erstellen und Verarbeiten von HTML, XML oder anderen Textdokumenten zur Verfügung stellt, mit deren Hilfe Webseiten generiert werden können. Genshi ist ausgezeichnet dokumentiert und besitzt Rückhalt. Hinter Genshi steht edgewall, die Firma welche Trac entwickelt.

5.2.1.4. FastCGI

Für die Kommunikation mit dem Webserver wird FastCGI verwendet. Durch FastCGI ist eine in Python geschriebene Webapplikation vom Webserver unabhängig. Wie eine FastCGI Schnittstelle in Python auszusehen hat, ist im WSGI Standard [http://wiki.python.de/WSGI] festgelegt. Für den WSGI Standard existieren unzählige Implementierungen. Die in der Webapplikation verwendeten Bibliotheken sind Flup [http://www.saddi.com/software/flup/] und Paste [http://pythonpaste.org/] Aus Paste werden nur einige WSGI Middlewares vorallem für das Error-Handling verwendet. Der FastCGI Server, auf welchem strongManager läuft, stammt aus der Flup Bibliothek.

5.3. Webapplikation

Die Webseitenverarbeitung erfolgt mittels FastCGI nach dem WSGI Standard von Python. Zu diesem Zweck wird ein FastCGI Server verwendet, welcher ein Objekt der Klasse *StrongManager* als WSGI Service (sogenannte Applikation) startet. Die Klasse *StrongManager* besteht aus einem Publisher und mehreren WSGI Middlewares, von welchen jede eine spezielle Aufgabe erfüllt. Der Publisher ist die zentrale Komponente der Webseitenverarbeitung. Er ist zuständig für das Abbilden der URL's (siehe Abschnitt URL Design) auf Controller und Actions und das Erstellen von HTTP-Request und -Response Objekten, welche für die Verarbeitung benötigt werden. Im Verlaufe dieses Abschnitts werden die einzelne Komponenten erläutert. Für den Überblick über die vorhanden Klassen ist unten das Klassendiagramm der Webapplikation aufgeführt.



Klassendiagramm Webhandling

5.3.1. Publisher

Der Publisher ist zuständig für die Verwaltung der *ApplicationController* (Oberklasse aller Controller) und die Verarbeitung von dynamischen und statischen URL's. Dynamische URL's sind URL's, welche auf Controller und Actions abgebildet werden. Unter einer statischen URL versteht man z.B. den Pfad zu einem CSS Stylesheet, welcher nicht über eine Action erreichbar ist.

Wenn ein HTTP-Request beim Publischer eintrifft, wird seine Methode `__call__()` aufgerufen.

```
1 def __call__(self, environ, start_response):
2     webinfo.request = Request(environ)
3     webinfo.response = Response()
4     path_info = environ.get('PATH_INFO', '').strip('/')
5     if path_info.startswith("static"):
6         return self._handle_static(environ, start_response)
7     return self._handle_action(environ, start_response)
```

In ihr wird ein Request Objekt anhand des HTTP-Request's, welcher sich im Parameter `environ` befindet, erstellt.

```
class Request(object):
    def __init__(self, environ):
        self.environ = environ
        self.postvars = parse_formvars(environ, \
          include_get_vars=False).mixed()
        self.getvars = dict(parse_querystring(environ))
        self.session_service =  environ['session']
```

Je nach URL (statisch oder dynamische), wird daraufhin der entsprechende Handler aufgerufen. Bei einer dynamischen URL wäre dies die Methode `_handle_action()` (Zeile 7), welche die zur URL gehörenden Action findet und aufruft. Bei einer statischen URL hingegen, würde der Request durch die Methode `_handle_static()` verarbeitet. Diese verwendet dafür ein Objekt der Klasse *StaticURLParser* von Paste. Der *StaticURLParser* ist nichts anderes als eine WSGI Middleware, welche beim Erstellen des Publishers an dessen Konstruktor übergeben wird.

Der Publisher wurde so konstruiert, dass er beim Hinzufügen einer neuen URL nicht erweitert werden muss. Der Controller und die Action, welche die neue URL verarbeiten, brauchen sich nur an die Namenskonvention des URL Designs zu halten. Beim Neustart des Publishers findet dieser alle neuen Controller und Actions. Wie das Einlesen der Controller verläuft, kann der unten aufgeführten Methode entnommen werden.

```
1 def get_controllers(gateway_manager):
2     controllers = {}
3     for var in globals().iteritems():
4         var_name = var[0]
5         if (var_name.endswith("Controller")):
6             controller = var[1](gateway_manager)
7             controllers[controller.NAME] = controller
8     return controllers
```

Es wird durch alle im Pythonmodule *controllers* vorhanden Definition iteriert (Zeile 3), und von allen Klassen, deren Namen auf **Controller** endet, eine Instanz erzeugt (Zeile 6) und der Liste hinzugefügt (Zeile 7).

5.3.2. Controller und Actions

Gemäss URL Design wird eine URL vom Publisher auf einen Controller und eine Action abgebildet. Beim Aufruf einer Action setzt diese eine *SMPMessage* (beinhaltet SMP-Request-Mes-

sage) über das Netzwerk ab, nimmt die erhaltene Antwort entgegen, rendert ein ihr zugewiesenes XHTML-Template mit den gewünschten Daten und returniert dieses als String. Dieser String wird vom Publisher entgegengenommen und als Response verpackt dem Browser zurück geschickt.

Beispiel: *ConnectionsController* .

```
 1 class ConnectionsController(ApplicationController):
 2
 3   TEMPLATE_DIR = "templates/connections"
 4   NAME = "connections"
 5
 6   ....
 7
 8   @authenticate
 9   def connection(self, request, gateway, connection):
10
11      TEMPLATE_FILE = "connection.html"
12      connection = smputils. \
13        make_connection_request(self.vpn_gateway_manager, \
14        gateway, connection)[0]
15      ikeencryptionalgorithms = \
16        connection.ikeencryptionalgorithmlist.element_list
17      ikeintegrityalgorithms = \
18        connection.ikeintegrityalgorithmlist.element_list
19      diffiehellmangroups = \
20        connection.diffiehellmangrouplist.element_list
21      context_data = {
22          "gateway":gateway,
23          "connection":connection,
24          "ikeencryptionalgorithms": \
25            ikeencryptionalgorithms,
26          "ikeintegrityalgorithms":ikeintegrityalgorithms,
27          "diffiehellmangroups":diffiehellmangroups
28                  }
29      return self._render_template(TEMPLATE_FILE,
30        **context_data)
```

Der *ConnectionsController* besitzt die beiden Actions `index()` (oben nicht aufgeführt) und `connection()`. Diese werden durch die zwei unten aufgeführten URL's abgedeckt.

- `index()`: http://strongmanager/connections?gateway=gateway_name (Action muss nicht explizit in URL angegeben werden)

- `connection()`: http://strongmanager/connections/connection?gateway=gateway_name &connection=connection_name

Die Action `connection()` erzeugt einen XHTML-String, welcher die Details einer strongSwan Connection anzeigt. Der Verarbeitungsablauf von `connection()` ist folgendermassen:

1. Zeile 12-14: Absetzen eines *ConnectionRequest* für den Gateway mit dem Namen `gateway` und einer Connection mit dem Namen `connection`.

2. Zeile 15-28: Die erhaltene *Connection* auswerten und daraus Daten für das Template bilden (`context_data`).

3. Zeile 29: Template (Zeile 11) anhand der erhaltenen Contextdaten rendern und als XHTML-String returnieren.

Bemerkung

Das verwendete Controller/Action-Prinzip ist ähnlich wie jenes von Ruby on Rails http://www.rubyonrails.org.

5.3.3. Templates

Jeder Action zugehörig ist ein Template mit dem Namen der Action. Bei der Action *connection* ist dies das Template `connection.html`, welches unten aufgeführt ist.

```
 1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 2     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
 3
 4 <html xmlns="http://www.w3.org/1999/xhtml"
 5   xmlns:py="http://genshi.edgewall.org/"
 6   xmlns:xi="http://www.w3.org/2001/XInclude"
 7   xml:lang="en" lang="en">
 8 <head>
 9 <meta http-equiv="content-type"
10   content="application/xhtml+xml; charset=UTF-8" />
11 <link rel="stylesheet"
12   type="text/css" href="/static/stylesheet.css" />
13 <title>Strongmanager - Connection Details</title>
14 </head>
15 <body>
16 <xi:include href="../common/menu.html" />
17 <div id="headermain">
18   <xi:include href="../common/header.html" />
19 <div id="main">
20 <h1>Connection Details</h1>
21 <ul>
22   <li><label>Connection Name: </label>
23   ${connection.connectionname.element_value}
24   </li>
25
26   <li><label>Local Address: </label>
27   ${connection.localaddress.ipaddress.element_value}
28   </li>
29
```

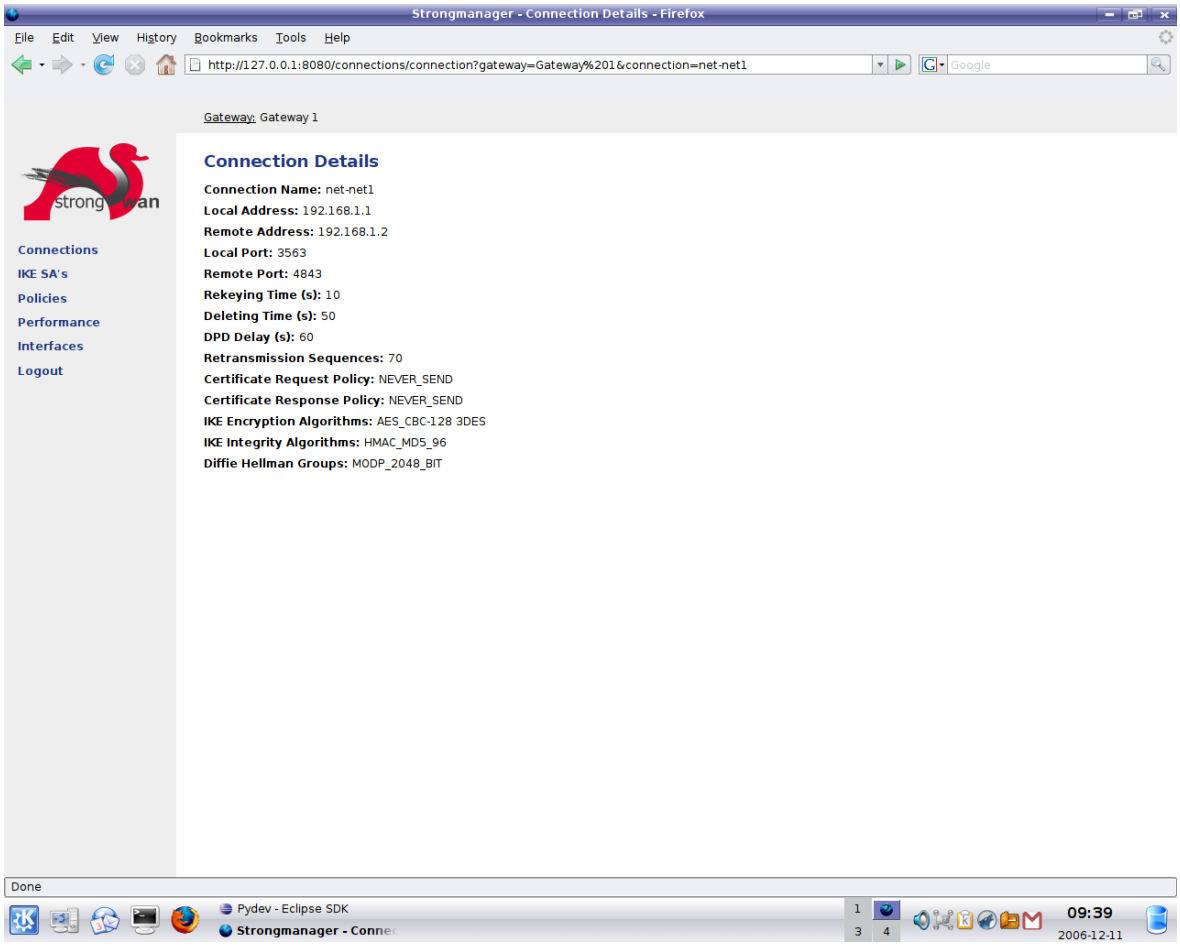
```
30   ....
31
32   <li><label>Certificate Response Policy: </label>
33   ${connection.certificateresponsepolicy.element_value}
34   </li>
35
36   <li><label>IKE Encryption Algorithms: </label>
37   <py:for each="algorithm in ikeencryptionalgorithms">
38   ${algorithm.element_value}
39   </py:for></li>
40
41   ....
42
43 </ul>
44 </div>
45 </div>
46 </body>
47 </html>
```

Ein Template besteht immer aus XHTML und Python Code in Form der Genshi XML-Template-Language. Die Genshi XML Template Language wird durch den Namespace `xmlns:py=" http://genshi.edgewall.org/` (Zeile 5) in das XHTML-Template eingebunden und kann daraufhin verwendet werden (alle unterstützten Fragmente sind im Genshi Guide unter der URL http://genshi.edgewall.org/wiki/Documentation/xml-templates.html beschrieben). Auf die dem Template als `context_data` übergebenen Objekten kann mittels `${myobject}` (Zeile 23) zugegriffen werden. Eine Iteration über eine List sieht gemäss Genshi Template Language folgendermassen aus:

```
<py:for each="algorithm in ikeintegrityalgorithms">
  ${algorithm.element_value}
</py:for>
```

Zum Rendern eines Templates wird die Methode `_render_template()` des *Application-Controller* aufgerufen.

```
def _render_template(self, template_name, **contextdata):
    tmpl = self.tmpl_loader.load(template_name)
    stream = tmpl.generate(**contextdata)
    return stream.render("xhtml")
```



Ansicht der Details einer Connection

5.3.4. WSGI und Middlewares

Für die Kommunikation mit dem Webserver wird FastCGI verwendet. Dadurch ist eine in Python geschriebene Webapplikation vom Webserver unabhängig. Wie eine FastCGI Schnittstelle in Python auszusehen hat, ist im WSGI Standard [http://www.python.org/dev/peps/pep-0333/] festgelegt. Für den WSGI Standard existieren unzählige Implementationen. In der Webapplikation wird der FastCGI Server von Flup benutzt. Der FastCGI Server wird im Pythonmodul *fastcgi* mit einem *StrongManager* Objekt als *WSGIService* gestartet. Beim Eintreffen eines HTTP-Request wird die unten aufgeführte Methode `__call__()` des *StrongManager* Objekts aufgerufen und der HTTP-Request vom Publisher und den anderen WSIG Middlewares der Reihe nach abgearbeitet, in dem ebenfalls die in den WSGI Middlewares enthaltene Methode `__call__()` aufgerufen wird.

```
class StrongManager(object):
    def __init__(self):
        self.gateway_manager = VPNGatewayManager()

        self.app = StaticURLParser(STATIC_FOLDER_LOCATION)
        self.app = Publisher(self.app, self.gateway_manager)
        self.app = HTTPExceptionHandler(self.app)
        self.app = errordocument.forward(self.app, codes={})
```

```
        self.app = SessionMiddleware(self.app)
        self.app = GzipMiddleware(self.app)
        self.app = ErrorMiddleware(self.app)

    def __call__(self, environ, start_response):
        return self.app(environ, start_response)
```

Der grosse Vorteil von WSGI ist, dass eine Applikation mit Hilfe des Decorator Patterns um beliebig viele WSGI Middlewares erweitert werden kann. Alle in strongManager verwendeten WSGI Middlewares sind in der oben aufgeführten Methode `__init__()` der Klasse *StrongManager* aufgelistet.

- *HTTPExceptionHandler* - Fängt HTTP-Error-Exceptions ab und erstellt entsprechende HTTP-Error-Response-Messages (Code 400, 500, etc.). HTTP Error Exceptions werden üblicherweise vom Publisher geworfen. Diese Middleware stammt aus dem Webframework von Paste [http://pythonpaste.org/module-paste.httpexceptions.html].

- `errordocument.forward()` - Weiterleiten der HTTP-Error-Response-Messages auf eine schön formatierte Webseite. Stammt ebenfalls aus Paste [http://pythonpaste.org/module-paste.errordocument.html]. Zur Zeit kommt diese Middleware jedoch nicht zum Einsatz.

- *SessionMiddleware* - Die *SessionMiddleware* wird benötigt um Sessions zu erstellen, welche für die Authentifizierung von Benutzern verwendet werden. Stammt aus Flup [http://trac.saddi.com/flup/browser/flup/trunk/flup/middleware/session.py].

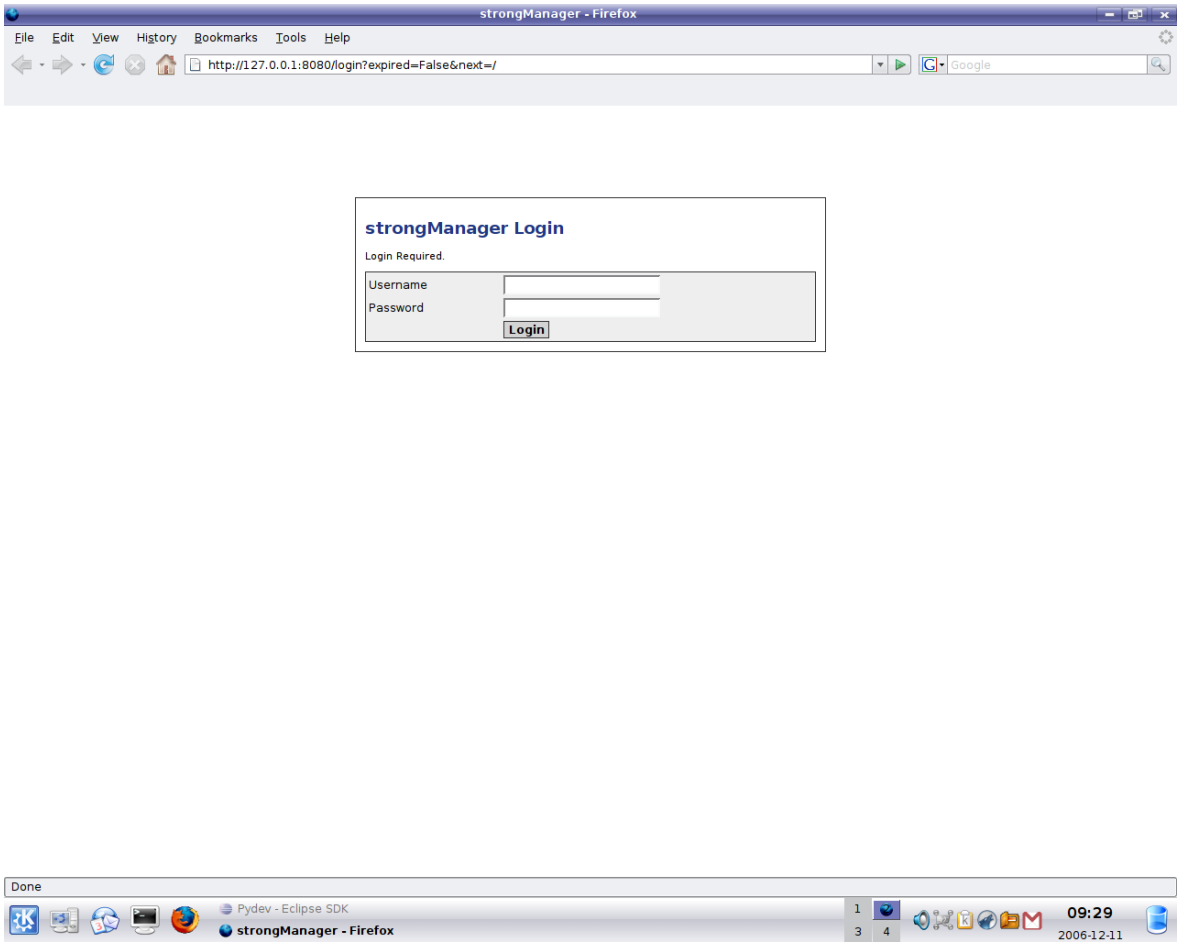
- *GzipMiddleware* - Middleware um die HTTP-Responses mit **gzip** zu verpacken. Stammt ebenfalls aus Flup [http://trac.saddi.com/flup/browser/flup/trunk/flup/middleware/gzip.py].

- *ErrorMiddleware* - Fängt Exceptions ab, welche ausserhalb des Webcontext (keine HTTP-Error-Exceptions) liegen. Stammt aus Paste [http://pythonpaste.org/module-paste.exceptions.errormiddleware.html].

Durch die WSGI Middlewares wird das Exception-Handling stark vereinfacht, ist es doch möglich eine beliebige Exception zu werfen, welche dann von der *ErrorMiddleware* abgefangen und als Webseite ausgegeben werden kann.

5.3.5. Authentifizierung

Um die Webapplikation von strongManager vor unbefugten Zugriffen zu schützen, wird eine Authentifizierung durchgeführt. Zu diesem Zweck existieren die beiden Controller *LoginController* und *LogoutController*, so wie die Methode `authenticate()`. Wenn ein Benutzer z.B. alle Connections abfragen möchte und dazu die entsprechende URL im Browser eingibt, für ihn aber noch keine Session existiert, wird er auf den *LoginController* umgeleitet. Dieser präsentiert eine Loginseite, auf welcher sich der Benutzer authentifizieren muss.



Loginseite zur Authentifizierung

Geschützt werden können die Webseiten mit diskreten Daten durch einen Python Decorator, welcher vor einer Action in einem Controller steht.

```
@authenticate
def connection(self, request, gateway, connection):
    ...
```

Ein Decorator ist nichts anderes als ein Platzhalter für eine Methode, welche jedes mal beim Aufruf einer Action zuerst abgearbeitet wird. In der unten aufgeführten Methode `authenticate()` wird die Action, welche nach dem Decorator *@authenticate* folgt, im Parameter `func` übergeben und im Fall das eine Session für den Benutzer existiert aufgerufen (Zeile 10). Dieses Verhalten ist möglich, da in Python Funktionen auch Objekte sind.

```
1 def authenticate(func):
2     def wrapper(self, **args):
3         request = args['request']
4         service = request.session_service
5         url = request.environ.get('PATH_INFO', '')
6         if request.environ.get('QUERY_STRING'):
7           url += '?' + request.environ['QUERY_STRING']
8         expired = str(service.hasSessionExpired)
```

```
 9         if service.hasSession:
10             return func(self, **args)
11         else:
12             raise Redirect("login?expired=" + expired \
13               + "&next=" + url)
14     return wrapper
```

Falls keine Session existiert, wird der Benutzer auf die Loginseite umgeleitet (Zeile 12).

Der Sessiontimeout beträgt **10 Minuten** . Der voreingestellt Benutzer ist **strongSwan** mit dem Passwort **strongSwan** . Der Benutzername und Passwort-Hash (SHA-1) können im Pythonmodule *defaultconfig* geändert werden, oder besser mit einer eigenen *config* Datei im Arbeitsverzeichnis überschrieben werden.

5.3.6. URL Design

Die URL's sind nach dem Controller und Action Prinzip aufgebaut. Dass heisst eine URL wird auf einen Controller und eine Methode abgebildet. Der Aufbau einer URL sieht folgendermassen aus:

http://host/controller_name/action_name?attribute1=value1&attribute2=value2

Im obigen Fall würde eine Action mit dem Namen `action_name` in der *ApplicationController* Klasse *Controller_name* aufgerufen. Die Methode `action_name` besitzt dabei die Parameter `attribute1` und `attribute2` welche die entsprechenden Werte beinhalten.

Die Parameter der zur Zeit vorhandenen URL's, werden per HTTP-GET-Request übertragen. Im Fall der Webapplikation ergeben sich folgende URL's:

- http://strongmanager/connections?gateway=gateway_name - Auflisten aller Connections eines Gateways mit dem Namen `gateway_name`.
- http://strongmanager/connections/connection?gateway=gateway_name&connection=connection_name - Auflisten der konfigurierten Details einer Connection mit dem Namen `connection_name`.
- http://strongmanager/policies?gateway=gateway_name - Auflisten aller Policies eines Gateways mit dem Namen `gateway_name`.
- http://strongmanager/policies/policy?gateway=gateway_name&policy=policy_name - Auflisten der konfigurierten Details einer Policy mit dem Namen `policy_name`.
- http://strongmanager/ikesas?gateway=gateway_name - Auflisten aller IKE SA's eines Gateways mit dem Namen `gateway_name`.
- http://strongmanager/ikesas?gateway=gateway_name&connection=connection_name - Auflisten aller IKE SA's einer Connection mit dem Namen `connection_name`.
- http://strongmanager/ikesas/details?gateway=gateway_name&connection=connection_name&spiinitiator=spi_initiator&spiresponder=spi_responder -

Auflisten der Details einer IKE SA mit den SPI's `spi_initiator` und `spi_responder`. Die Child SA's sind im View ebenfalls darin enthalten.

- http://strongmanager/performance?gateway=gateway_name - Auflisten von performancebezogenen Informationen eines Gateways mit dem Namen `gateway_name`.
- http://strongmanager/interface?gateway=gateway_name - Auflisten der IP Adressen auf denen ein Gateway mit dem Namen `gateway_name` lauscht.

Bemerkung

Die Action *index()* muss bei einer URL nicht explizit aufgerufen werden.

5.3.7. Integrierter Webserver

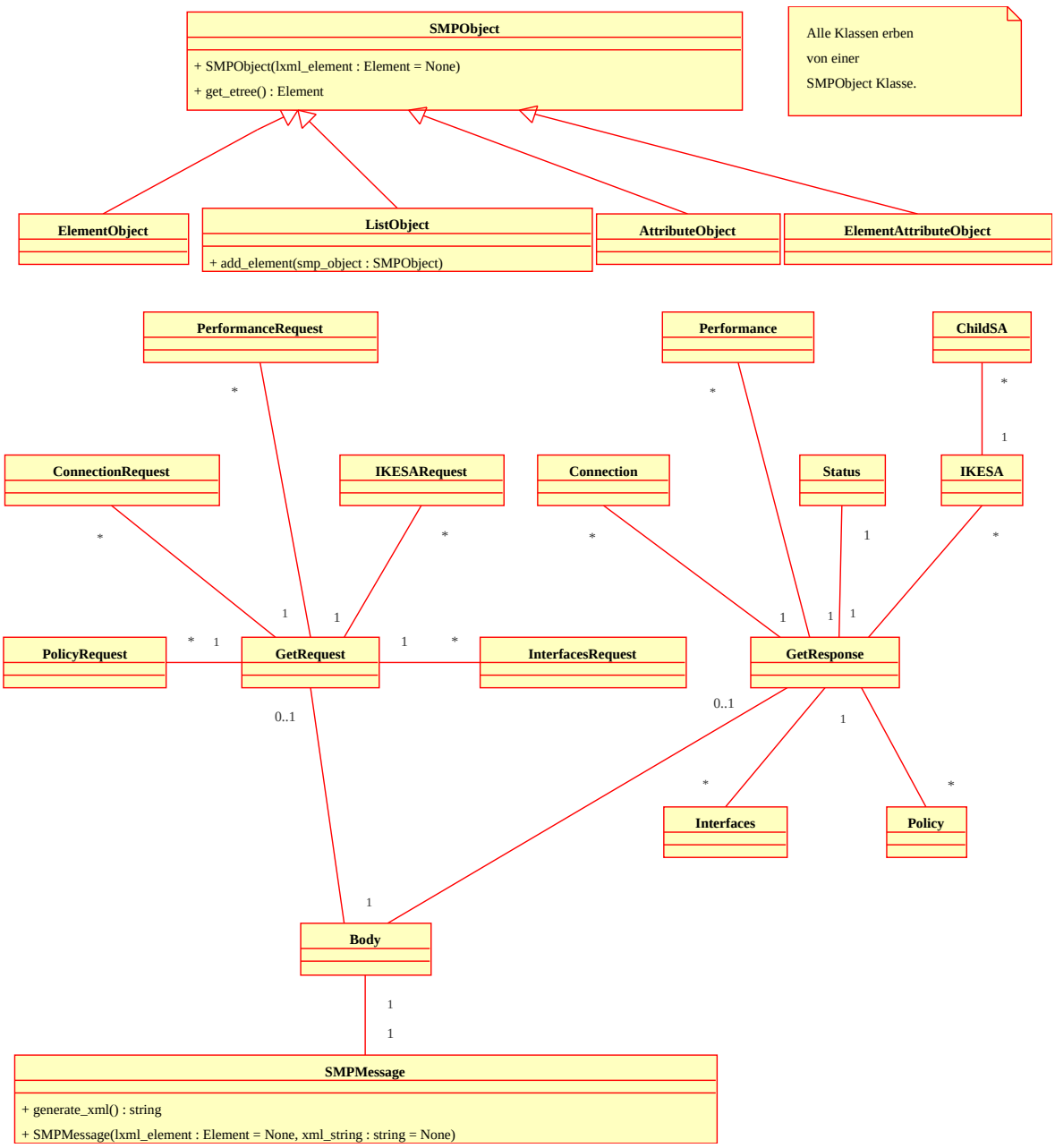
In der Webapplikation wurde ein Webserver integriert, damit sie On-the-Fly getestet werden kann. Standardmässig läuft dieser auf dem Port **8080**.

5.4. Model

Das Model repräsentiert die XML-Elemente aus der Spezifikation des Strong Management Protocols. Ein Objekt vom Typ *SMPMessage* beinhaltet beispielsweise ein Objekt vom Typ *Body*, welches wiederum ein Objekt vom Typ *GetRequest* beinhalten kann. Jedes XML-Element aus SMP wird auf eine eigene Klasse abgebildet (Zur Zeit sind nur Klassen für statusbezogene Informationen vorhanden).

Um aus einem *SMPMessage* Objekt ein XML-Stream zu generieren, kann die Methode `generate_xml()` von *SMPMessage* verwendet werden. Der XML-Stream wird dabei rekursiv generiert, indem die `get_etree()` Methode jedes Objekts aufgerufen wird. Beim Erstellen eines *SMPObjekt* verhält es sich gleich. Der XML-Stream wird dem Konstruktor von *SMPMessage* übergeben, wodurch rekursiv der Objektbaum mit Messages Objekten gebildet wird.

Das UML Diagramm der Modelklassen sieht folgendermassen aus:



Klassendiagramm Messages

5.4.1. Design Entscheidungen

Die Modelklassen, welche wie bereits erwähnt die XML-Elemente von SMP kapseln, sind so gestaltet, dass bei einer Erweiterung diese gar nicht oder um ein Minimum angepasst werden müssen. Eine Erweiterung ist auch durch Personen möglich, welche sich nicht intensiv mit der Materie beschäftigt haben.

Jedes Objekt einer Messagesklasse erbt von einer der folgenden Oberklassen:

- *SMPObject*
- *ListObject*

- *ElementAttributeObject*
- *ElementObject*
- *AttributeObject*

Die oben aufgeführten Klassen werden nun im Verlauf dieses Abschnitts im Genaueren betrachtet.

5.4.2. SMPObject

SMPObject stellt die Oberklasse von allen Modelklassen dar. Sie wird von Klassen direkt verwendet, welche ein XML-Element repräsentieren, welches ausschliesslich aus weiteren Elementen besteht, die nicht von der gleichen Sorte sind. Ein Beispiel dafür wäre das XML-Element *Connection*. Das XML-Element *ConnectionList* hingegen besteht aus mehreren *Connection* Elementen. Die Klasse *ConnectionList* erbt in diesem Fall nicht direkt von *SMPObject* sondern von *ListObject*.

Als Beispiel einer Implementierung von *SMPObject* wollen wir die Klasse *Connection* betrachten.

```
 1 class Connection(SMPObject):
 2
 3     ELEMENT_NAME = "Connection"
 4
 5     def __init__(self, lxml_element=None):
 6         attr_name_list = ["connectionname",
 7             "localaddress",
 8             "remoteaddress",
 9             "localport",
10             "remoteport",
11             "rekeyingtime",
12             "deletingtime",
13             "dpddelay",
14             "retransmissionsequences",
15             "certificaterequestpolicy",
16             "certificateresponsepolicy",
17             "ikeencryptionalgorithmlist",
18             "ikeintegrityalgorithmlist",
19             "diffiehellmangrouplist",
20                         ]
21         SMPObject.__init__(self, \
22           lxml_element, attr_name_list)
23
24     def _read_etree(self, lxml_element):
25         for child in lxml_element.getchildren():
26             tag = self._get_name_from_lxml_tag(child.tag)
27             setattr(self, tag.lower(), (globals()[tag]) \
```

```
28                  (lxml_element=child))
```

Jedes Objekt einer Modelklassse besitzt ein Element- oder Attributnamen, welcher dem XML-Element-/Attributnamen des entsprechenden SMP Elements entspricht. Dieser Elementname ist im Objektattribut `ELEMENT_NAME` (Zeile 3) abgelegt. Es wird benötigt um aus der Objekt Repräsentation die XML-Repräsentation zu erzeugen. Zu beachten ist, dass der Wert von `ELEMENT_NAME` dem Namen der Klasse entspricht, welche `ELEMENT_NAME` beinhaltet. Diese Konvention muss zwingend eingehalten werden, da sonst das Erstellen der XML-Repräsentation, bzw. das Einlesen von XML nicht funktioniert.

Hinweis

Alle angesprochenen Namenskonventionen gelten nicht für Klassen, welche von *AttributeObject* erben.

Die Namen der Objekte aus welchen ein *SMPObject* zusammengesetzt ist, sind in der Liste `attr_name_list` (Zeile 6) abgelegt. Die Objektnamen entsprechen dabei den entsprechenden Klassennamen in Lower Case. Am Beispiel von Connection würde dem Attribut `connectionname` ein Objekt vom Typ *ConnectionName* zugewiesen (Zeile 6). Auch diese Konvention muss wiederum zwingend eingehalten werden, damit das Erstellen der XML-Repräsentation und das Einlesen von XML funktioniert.

Um die beiden Abläufe: Erstellen der Objektrepräsentation und XML-Repräsentation zu verdeutlichen, betrachten wir nun zuerst die Methode `get_etree()` und danach `_read_etree()`. `get_etree()`, welche *Connection* von der Klasse *SMPObject* erbt, erstellt die XML-Repräsentation eines Objekt in Form von lxml etree Elementen. Durch `etree.Element(self.ELEMENT_NAME)` wird ein neues etree Element mit dem Wert von `ELEMENT_NAME` erzeugt. Im Fall von *Connection* wäre dies `Connection`. Als nächstes werden von allen in `attr_name_list` festgelegten Attributen die zugewiesenen Objekte geholt, davon etree Elemente erzeugt und dem etree Element von *Connection* hinzugefügt. Das Hinzufügen verläuft dabei rekursiv in dem die `get_etree()` Methode der zugewiesenen Objekte aufgerufen wird. Die Reihenfolge der Erzeugung der etree Elemente entspricht derjenigen der Attribute in `attr_name_list`, welche wiederum der Reihenfolge im Relax NG Schema von SMP entspricht.

```
def get_etree(self):
    element = etree.Element(self.ELEMENT_NAME)
    for smp_object in self._get_sorted_attribute_list():
        self._add_smp_object_to_element(element, smp_object)
    return element
```

Um ein *Connection* Objekt aus einem etree Element zu erstellen, kann das etree Element dem Konstruktor von *Connection* übergeben werden. Der Erstellungsvorgang, der auch die Kindobjekte erzeugt, erfolgt rekursiv über die private Hilfsmethode `_read_etree()` (siehe Codebeispiel Klasse *Connection*). In ihr wird der Tagname der etree Kindelemente verwendet um direkt Objekte von Modelklassen zu erzeugen (Zeile 32-33). Die erzeugten Objekte werden den entsprechenden Attributen zugewiesen (`setattr()` - Zeile 31). Bei einem etree Element mit dem Tagnamen *ConnectionName* würde ein Objekt der Klasse *ConnectionName* erzeugt und dieses dem Attribut `connectionname` zugewiesen. Nun stellt sich die Frage, aus welchem

Grund die Methode `_read_etree()` in der Klasse *Connection* überschrieben werden muss, obwohl sie eigentlich bei allen Klassen, welche von *SMPObject* direkt erben, gleich ist? Das Problem liegt in der Verwendung von `globals()` (Zeile 32). `globals()` liefert ein Dictionary aller in einem Pythonmodul festgelegten Namen als Key/Value Paar. Man kann z.B. über den Key *ConnectionName* das Klassenobjekt dieser Klasse holen und mit Hilfe des Konstruktors ein Objekt davon erzeugen (`(globals()[tag])(lxml_element=child)`) (Zeile 32). Da sich die Klasse *SMPObject* aber in einem eigenständigen Pythonmodul befindet, kennt sie die Klasse *ConnectionName* nicht. Bei einem direkten Aufruf von `globals()` in *SMPObject* würde dadurch die Klasse *ConnectionName* nicht gefunden werden. Als Lösung könnte man natürlich das Modul, welches *ConnectionName* beinhaltet, im Modul *smpobject* (beinhaltet Klasse *SMPObject*) importieren. Dadurch würden aber zyklische Abhängigkeiten entstehen, welche nicht sehr erstrebenswert sind. In der Lösung, welche hier verwendet wurde, kopiert man einfach die Methode `_read_etree()` der Modelklasse, von welcher man direkt erbt.

Um die Modelklassen um eine weitere Klasse zu ergänzen, muss man nur eine Klasse mit dem Namen des SMP XML-Elements erstellen, dem Attribut `ELEMENT_NAME` den Namen des SMP XML-Elements zuweisen, die Objektattribute in der List `attr_name_list` definieren (Klassennamen in Lower Case) und die Methode `_read_etree()` von der *SMPObject* Klasse kopieren, von welcher man direkt erbt. Als Resultat kann nun von Objekten der neu erstellten Klasse die XML-Repräsentation und anhand der XML-Repräsentation Objekte davon erstellt werden. Der Aufwand für die Erweiterung beschränkt sich dadurch auf ein paar Minuten und die API von lxml (beinhaltet etree) muss man nicht kennen.

5.4.3. ElementObject

ElementObject repräsentiert ein einfaches XML-Element, welches aus einem Namen (`ELEMENT_NAME`) und einem Wert (`element_value`) besteht. Am Beispiel von *IPv4Address* sieht dies folgendermassen aus:

```
class IPv4Address(SimpleObject):

    ELEMENT_NAME = "IPv4Address"
```

Ein Objekt kann wie folgt erstellt werden:

```
address = IPv4Address("192.168.1.2")
```

Mittels `address.element_value` wird auf den dazugehörigen Wert zugegriffen.

Natürlich kann vom Objekt `address` über die Methode `get_etree()` die XML-Repräsentation und über den Konstruktor ein *IPv4Address* Objekt, anhand eines etree Elements, erzeugt werden.

5.4.4. ListObject

ListObject ist ein erweitertes *ElementObject*, welches eine Liste mit Objekten beinhaltet. Als Beispiel dient hierzu die Klasse *ConnectionList*. Einem Objekt von dieser Klasse können meh-

rere *Connection* Objekte mit Hilfe der Methode `add_element()` hinzugefügt werden. Auf die *Connection* Objekte kann über das Attribut `element_list` zugegriffen werden. Das Erstellen der XML-Repräsentation und eines *ListObject* Objekts anhand der XML-Repräsentation erfolgt ähnlich wie bei *SMPObject*. Mehr Informationen dazu sind im Sourcecode zu finden.

5.4.5. AttributeObject

Ein *AttributeObject* ist ähnlich wie ein *ElementObject*, nur das es nicht ein Element sondern ein XML-Attribut repräsentiert. Die Namenskonvention von Attributobjekten ist folgendermassen:

- Der Name der Klasse ist immer der XML-Attributname mit dem Zusatz **Attr** (Bsp. *ConnectionNameAttr*).

- Ein Element, welches ein Attributobjekt beinhaltet, besitzt ein Attribut mit dem Namen der entsprechenden Klasse in Lower Case und der Endung **attr** (Bsp. `connectionnameattr`).

- Der Name, welcher im Attribut `ATTRIBUTE_NAME` abgelegt ist, entspricht dem Namen des XML-Attributs in SMP.

Der Zugriff auf den Attributwert erfolgt über *attribute_value*.

5.4.6. ElementAttributeObject

ElementAttributeObject ist ein erweitertes *SMPObject*, welches nebst Elementen auch Attribute beinhalten kann. Der Unterschied besteht nur in den Methoden `get_etree()` und `_read_etree()`, welche nebst den Elementen die Attribute separiert behandeln muss. Für mehr Informationen sei wiederum auf den Sourcecode verwiesen.

5.4.7. Beispiel eines Requests

Unten aufgeführt ist ein Beispiel für das Erzeugen eines *ConnectionRequests* (Abfragen aller strongSwan Connections), aus welchem anschliessend ein XML-String erstellt wird:

```
smp_message = SMPMessage()
smp_message.body = Body()
smp_message.body.message = GetRequest()
connection_request = ConnectionRequest()
connection_request.connectionnameattr = \
  ConnectionNameAttr("%all")
smp_message.body.message. \
  add_element_to_list(connection_request)

smp_message.get_xml()
```

Die XML-Repräsentation sieht folgendermassen aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<SMPMessage>
  <Body>
    <GetRequest>
      <ConnectionRequest ConnectionName="%all"/>
    </GetRequest>
  </Body>
</SMPMessage>
```

Bemerkung

Die Methode *get_xml()* existiert nur in der Klasse *SMPMessage*, weil von Objekten der anderen Klassen nicht direkt ein XML-String erzeugt wird.

5.4.8. Validation

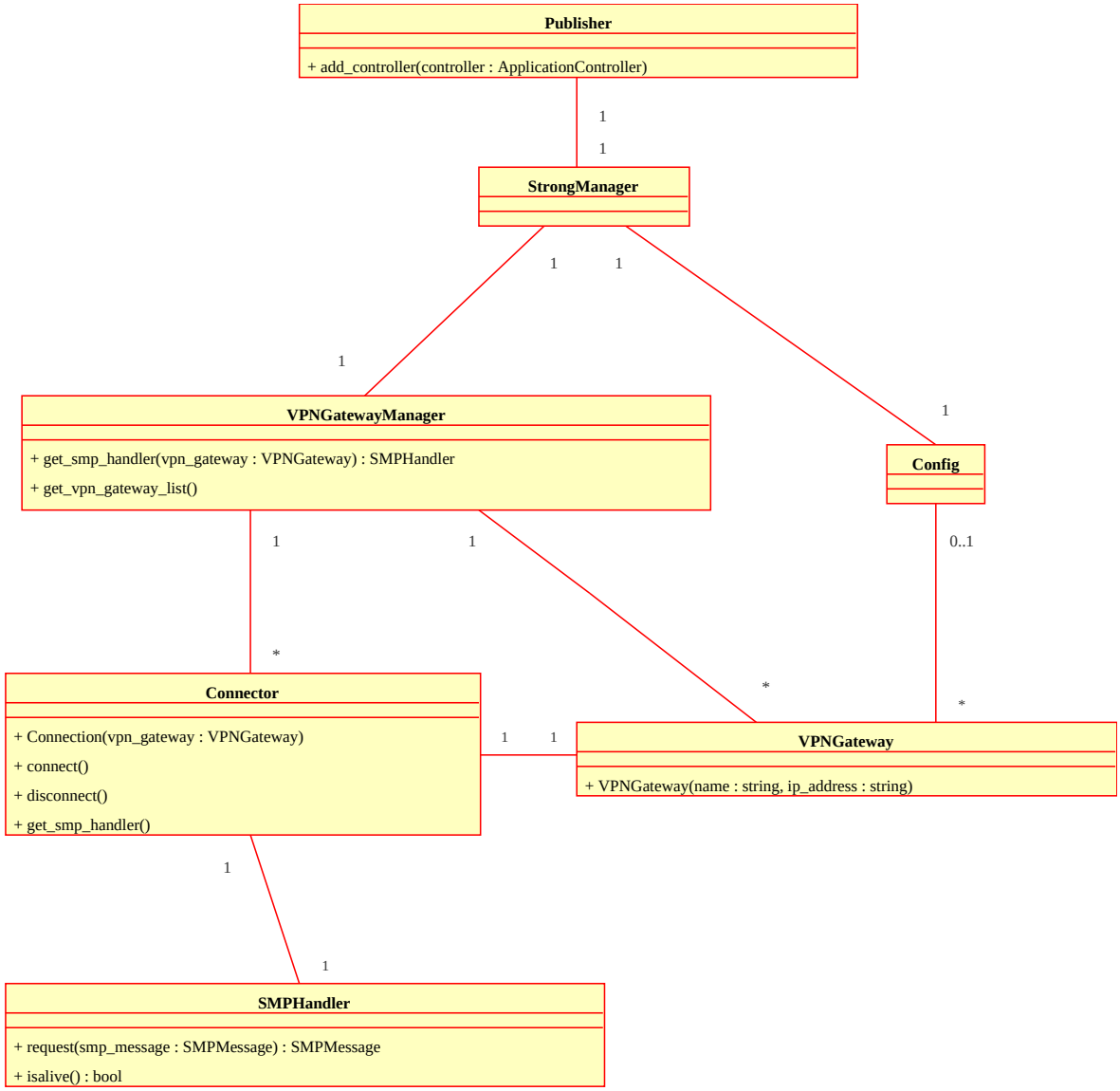
Um den Aufbau eines Requests und Response auf seine Gültigkeit hin zu prüfen, werden diese anhand eines Relax NG Schemas validiert. Die Validation erfolgt dabei immer im *SMPHandler* (liest Stream vom Socket - Abschnitt Kommunikationsschnittstelle) über die Methode `isvalid()`.

```
def isvalid(xml_msg):
    relaxng_doc = etree.parse("../smp/schema/smp.xml")
    msg = etree.XML(xml_msg)
    relaxng = etree.RelaxNG(relaxng_doc)
    relaxng.assertValid(msg)
    return True
```

Falls eine aus dem Netzwerk erhaltene *SMPMessage* nicht valid ist, wird ein Fehler auf einer Webseite ausgegeben.

5.5. Kommunikationsschnittstelle

Die Kernkomponenten der Kommunikation sind die Klassen *VPNGatewayManager*, *Connector* und SMPHandler, wobei die eigentliche Kommunikation durch Objekte der Klassen *Connector* und *SMPHandler* verläuft. Ein Objekt der Klasse *Connector* erstellt dazu eine Verbindung zu einem VPN Gateway, welche vom *SMPHandler* beim Versand durch die Methode `request()` verwendet wird. Ein *SMPHandler* kann über die Methode `get_smp_handler()` des *VPNGatewayManagers* gefunden werden. Falls noch keiner existiert, wird vom *Connector* ein *SMPHandler* erzeugt und returniert. Um ein Überblick über die Klassen der Kommunikationsschnittstelle zu erhalten, ist unten das entsprechende UML Diagramm aufgeführt. Im weiteren Verlauf dieses Abschnitts wird auf den *VPNGatewayManager* und die Konfiguration von *VPNGateways* eingegangen. Für weitergehende Informationen sei auf den Sourcecode verwiesen.



Klassendiagramm Kommunikation

5.5.1. VPNGatewayManager

Der *VPNGatewayManager* verwaltet alle konfigurierten *VPNGateway* Objekte und Connectoren. Die *VPNGateway* Objekte sind über die Methode `get_vpn_gateway()` verfügbar. Für jeden konfigurierten *VPNGateway* existiert ein *Connector*, welcher anhand des *VPNGateway* Objekts initialisiert wird. Ein *Connector* selbst kapselt die Verbindung zu einem VPN Gateway. Über dessen Methoden `connect()` und `disconnect()` kann eine Verbindung mit Hilfe von TCP oder Unix Sockets zum Gateway auf-, bzw. abgebaut werden. Wenn man für einen VPN Gateway einen *SMPHandler* erhalten möchte, aber noch keiner existiert, wird dieser anhand der Methode `create_smp_handler()` des Connectors erzeugt.

5.5.2. Konfiguration der Gateways

Alle *VPNGateway* Objekte können im Pythonmodul *defaultconfig* nebst den Benutzernamen und Passwörtern für den integrierten Webserver konfiguriert werden. Ein *VPNGateway* Objekt

kapselt den Namen, die IP Adresse und den Port oder den Socket Pfad eines VPN Gateways. Alle in *defaultconfig* konfigurierten *VPNGateway* Objekte werden beim Start des *VPNGatewayManagers* von diesem eingelesen und zur Verfügung gestellt. Diese Konfiguration kann durch ein selbst erstelltes *config* Modul überschrieben werden.

```
gateways = ({
        'name' : "Gateway 1",
        'socket' : "/tmp/strongmanager.sock"
    },
    {
        'name' :"Gateway 2",
        'host' : "127.0.0.1",
        'port' : 4502
    })
```

Kapitel 6. Testen

Inhaltsverzeichnis

6.1. Einleitung

In diesem Kapitel wird beschrieben wie das Strong Management Protocol und strongManager getestet wurden.

Bemerkung zu Codebeispielen

In den Codebebeispielen sind alle von Hand erstellten Zeilenumbrüche mit zwei Leerschlägen eingerückt (wegen Platzmanagel). Pythonfragemte sind hingegen mit vier Leerschlägen eingerückt.

6.2. Model

Das Model, welches implementiert wurde, stellt eine zentrale Komponente von strongManager dar. Um die Funktionstüchtigkeit des Models zu testen, wurden Unit-Tests erstellt. Diese umfassen folgende zwei Bereiche:

- Funktionalitätstest des Models.
- Testen des Models anhand des Relax NG Schemas von SMP.

6.2.1. Funktionstest

Die für den Funktionstest des Models durchgeführten Unit-Tests umfassen Tests Cases für einzelne (z.B. *Connection*), als auch zusammengesetze Komponenten (z.B. *GetResponse* mit Connections). Ergo, das Model wird im Grossen und Kleinen getestet.

Die erstellten Unit-Tests sind im Pythonmodul `testmessages` abgelegt. Der Aufruf dieser Tests erzeugt folgende Konsolenausgabe.

```
testGetEtree (testmessages.ChildSATestCase) ... ok
testReadFromEtree (testmessages.ChildSATestCase) ... ok
testGetEtree (testmessages.ConnectionTestCase) ... ok
testReadFromEtree (testmessages.ConnectionTestCase) ... ok
testGetEtree (testmessages.
    GetResponsePerformanceTestCase) ... ok
testReadFromEtree (testmessages.
    GetResponsePerformanceTestCase) ... ok
testGetEtree (testmessages.
    GetResponseWithConnectionListTestCase) ... ok
testReadFromEtree (testmessages.
    GetResponseWithConnectionListTestCase) ... ok
testGetEtree (testmessages.
    GetResponseWithIKESAListTestCase) ... ok
testReadFromEtree (testmessages.
    GetResponseWithIKESAListTestCase) ... ok
testGetEtree (testmessages.
    GetResponseWithInterfacesTestCase) ... ok
testReadFromEtree (testmessages.
    GetResponseWithInterfacesTestCase) ... ok
testGetEtree (testmessages.
    GetResponseWithPolicyListTestCase) ... ok
testReadFromEtree (testmessages.
    GetResponseWithPolicyListTestCase) ... ok
testGetEtree (testmessages.IKESATestCase) ... ok
testReadFromEtree (testmessages.IKESATestCase) ... ok
testGetEtree (testmessages.InterfacesRequestTestCase) ... ok
testReadFromEtree (testmessages.
    InterfacesRequestTestCase) ... ok
testGetEtree (testmessages.InterfacesTestCase) ... ok
testReadFromEtree (testmessages.InterfacesTestCase) ... ok
testGetEtree (testmessages.LastUseTestCase) ... ok
testReadFromEtree (testmessages.LastUseTestCase) ... ok
testGetEtree (testmessages.LocalIDTestCase) ... ok
testReadFromEtree (testmessages.LocalIDTestCase) ... ok
testGetEtree (testmessages.PerformanceRequestTestCase) ... ok
testReadFromEtree (testmessages.
    PerformanceRequestTestCase) ... ok
testGetEtree (testmessages.PerformanceTestCase) ... ok
testReadFromEtree (testmessages.PerformanceTestCase) ... ok
testGetEtree (testmessages.PolicyRequestTestCase) ... ok
testReadFromEtree (testmessages.PolicyRequestTestCase) ... ok
testGetEtree (testmessages.PolicyTestCase) ... ok
testReadFromEtree (testmessages.PolicyTestCase) ... ok
testGetEtree (testmessages.RemoteIDTestCase) ... ok
```

```
testReadFromEtree (testmessages.RemoteIDTestCase) ... ok
testGetXML (testmessages.SMPMessageTestCase) ... ok
testReadFromXML (testmessages.SMPMessageTestCase) ... ok
testGetEtree (testmessages.
    SMPMessageWithConnectionRequestTestCase) ... ok
testReadFromEtree (testmessages.
    SMPMessageWithConnectionRequestTestCase) ... ok
testGetEtree (testmessages.
    SMPMessageWithIkeSARequestTestCase) ... ok
testReadFromEtree (testmessages.
    SMPMessageWithIkeSARequestTestCase) ... ok
testGetEtree (testmessages.StatusTestCase) ... ok
testReadFromEtree (testmessages.StatusTestCase) ... ok
testGetEtree (testmessages.TrafficSelectorTestCase) ... ok
testReadFromEtree (testmessages.
    TrafficSelectorTestCase) ... ok

----------------------------------------------------------------
Ran 44 tests in 0.225s

OK
```

Wie man der Konsolenausgabe entnehmen kann, umfassen die Funktionstests Test Cases für das Erstellen der XML-Repräsentation (`testReadFromEtree()` und `testReadFromXML()`) und der Objektrepräsentation (`testGetEtree()` und `testGetXML()`).

6.2.2. Schematest

Um das Model auf die korrekte Implementation des Strong Management Protocols hin zu überprüfen, wurden ebenfalls Unit-Tests erstellt. Diese befinden sich im Pythonmodul `testschemas` und überprüfen die Korrektheit von SMP Nachrichten, welche durch das Model erstellt wurden, anhand des Relax NG Schemas von SMP. Nebst dieser Implementationsüberprüfung kann durch diese Tests das Relax NG Schema von SMP verifiziert werden.

Das Ausführen der Unit Tests erzeugt folgende Konsolenausgabe:

```
testConnectionRequest (testschema.ConnectionRequestTestCase)
... None ok
testConnectionResponse (testschema.ConnectionResponseTestCase)
... None ok
testPolicyName (testschema.IKESARequestTestCase)
... None ok
testWihoutSPIResp (testschema.IKESARequestTestCase)
... None ok
testWithAllParams (testschema.IKESARequestTestCase)
... None ok
testWithoutSPIInit (testschema.IKESARequestTestCase)
... None ok
```

```
testWithChildSA (testschema.IKESAResponseTestCase)
... None ok
testWithChildSAAH (testschema.IKESAResponseTestCase)
... None ok
testWithoutChildSA (testschema.IKESAResponseTestCase)
... None ok
testConnectionResponse (testschema.InterfacesRequestTestCase)
... None ok
testConnectionResponse (testschema.InterfacesResponseTestCase)
... None ok
testConnectionResponse (testschema.PerformanceRequestTestCase)
... None ok
testConnectionResponse (testschema.
    PerformanceResponseTestCase) ... None ok
testConnectionResponse (testschema.PolicyRequestTestCase)
... None ok
testWithOnePolicy (testschema.PolicyResponseTestCase)
... None ok
testWithTwoPolicies (testschema.PolicyResponseTestCase)
... None ok

----------------------------------------------------------------
Ran 16 tests in 0.269s

OK
```

6.3. Webapplikation

Die Tests für die Webapplikation bestehen einerseits aus Unit Tests für die Controller, Actions und deren Hilfsmethoden, sowie aus manuellen GUI-Tests.

6.3.1. Controller und Actions

Die Unit-Tests für die Controller, Actions und deren Hilfsmethoden befinden sich in den beiden Pythonmodulen `testcontroller` und `testsmputils`. Diese umfassen Test Cases für alle Controller und Actions ausser der Login- und Logoutcontroller. Die Tests verwenden nebst den XHTML-Templates ein *VPNGatewayManager* Mock Objekt, welches den *VPNGatewayManager* simuliert. Das Mock Objekt wird zwingend benötigt, damit keine Netzwerkverbindung zum Emulator existieren muss. Durch dieses Vorgehen konnten die Unit-Tests isoliert durchgeführt werden.

Bemerkung

Nach dem Hinzufügen der Authentifizierung (Login- und Logoutcontroller) wurden die Controllertests fallen gelassen, da der Aufwand für die Wiederinstandsetzung zu gross gewesen wäre. Da die Controller aber nachträglich nicht mehr geändert wurden, fällt dies nicht ins Gewicht.

6.3.2. GUI

Das Testen des GUI erfolgte manuell. Nebst der Überprüfung aller Links und der Inhalte der Views, wurde versucht die Verhaltensvorgänge eines normalen Benutzers nachzuspielen.

Bemerkung

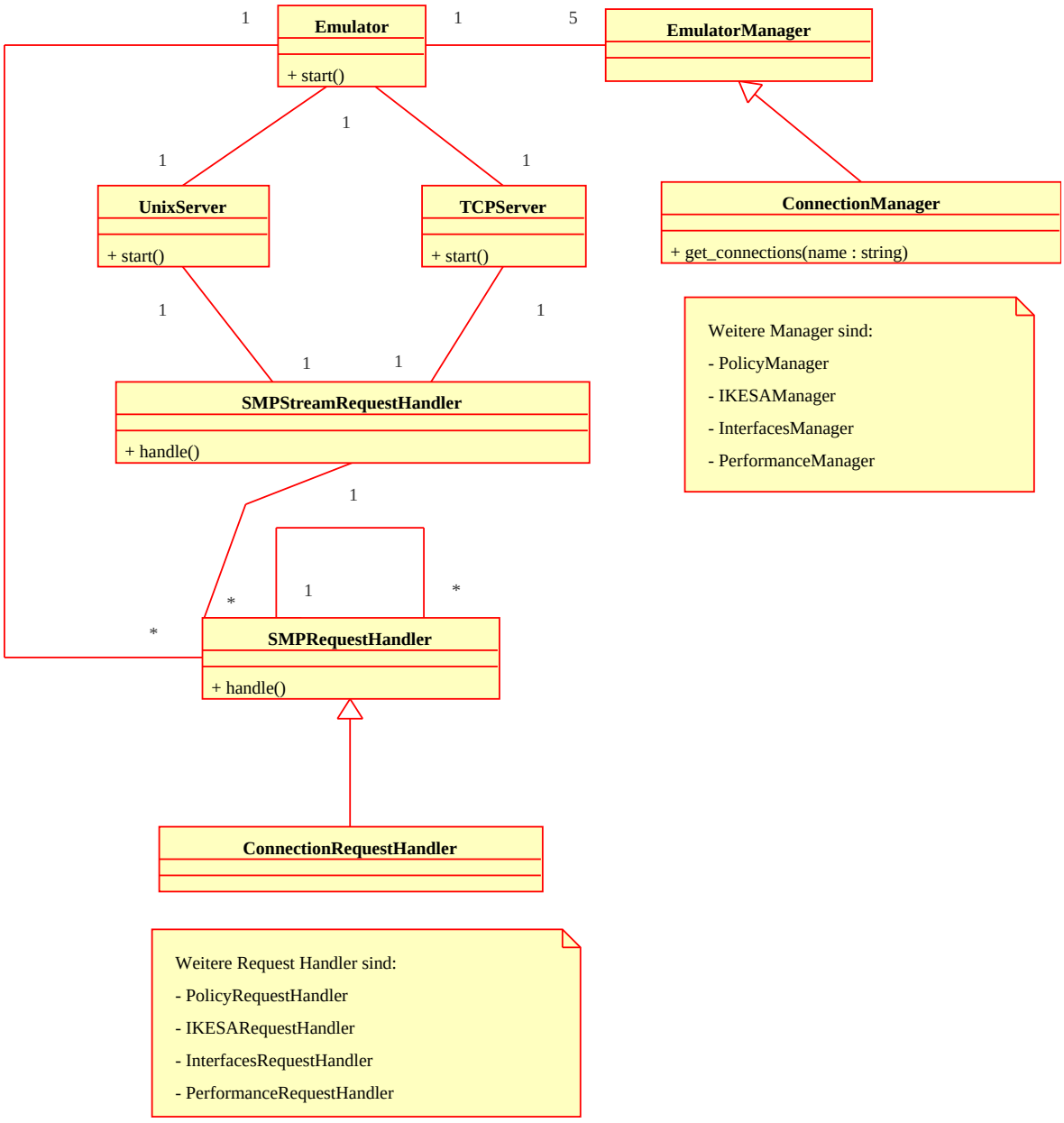
Durch die Verwendung von WSGI Exception Middlewares, können alle in strongManager auftretenden Exceptions abgefangen werden. Die Managementkonsole kann sich dadurch nicht, in Folge einer User-Interaktion, blockieren.

6.4. strongManager

Um die Managementkonsole strongManager als ganzes zu testen, wurde ein Emulator entwickelt, welcher einen VPN Gateway simuliert. Der Aufbau und die Funktionsweise des Emulators wird im nächsten Abschnitt erläutert.

6.5. VPN Gateway Emulator

Der *Emulator* beinhaltet einen multi threaded Server, welcher wahlweise ein *TCPServer* mit TCP Socket oder ein *UnixServer* mit Unix Socket sein kann. Dem gewählten Server wird bei dessen Erstellung ein *SMPStreamRequestHandler* Objekt übergeben, welches die SMP Message Requests entgegen nimmt und an den richtigen *SMPRequestHandler* weiterleitet. Der entsprechende *SMPRequestHandler* verarbeitet den Request und erstellt die Antwort, welche dann wieder vom *SMPStreamRequestHandler* an die Managementkonsole zurück gesendet wird. Um einen Überblick über das Design des Emulators zu erhalten, ist unten das Klassendiagramm aufgeführt. Die verwendeten *SMPObjects* sind im Klassendiagramm zu Gunsten der Übersichtlichkeit nicht aufgelistet. Im weiteren Verlauf dieses Abschnitts werden ausgewählte Komponenten des Emulators kurz beschrieben. Für mehr Informationen sei auf den Sourcecode des Emulators verwiesen.



Klassendiagramm Emulator

6.5.1. Request Handler

Die Request Handlers werden anhand des Beispiels *ConnectionRequestHandler* erläutert. Der *ConnectionRequestHandler* ist zuständig für die Verarbeitung von *ConnectionRequests* . Zu diesem Zweck besitzt er die Methode `handle()` (Zeile 5-10), welche die entsprechenden Connections für die Antwort vom *ConnectionManager* holt (8-9) und anschliessend die SMP Message Response (Zeile 10) returniert. Als Datenquelle verwendet der *ConnectionManager* die im Pythonmodul *connections* konfigurierten Connections.

```
1 class ConnectionRequestHandler(SMPRequestHandler):
2
3     REQUEST_NAME = ConnectionRequest.ELEMENT_NAME
```

```
 4
 5     def handle(self, request):
 6         connection_name = request.connectionnameattr. \
 7           attribute_value
 8         connections = self.emulator.connection_manager. \
 9           get_connections(connection_name)
10         return self._response(connections)
11
12     def _response(self, connections):
13         response = msgutils.GetResponseOK()
14         connectionlist = ConnectionList()
15         map(connectionlist.add_element, connections)
16         response.body.message.connectionlist = \
17           connectionlist
18         return response
```

6.5.2. StreamRequestHandler

Verwaltet werden die Request Handlers von der Klasse *StreamRequestHandler*. Die Request Handlers werden beim Erstellen des *Emulator* Objekts initialisiert und der *StreamRequest-Handler* Klasse übergeben. Die Instanzierung Erfolgt in der Function *get_handlers* (Zeile 4-11) des Modul *requesthandlers*, in dem alle im Pythonmodul enthaltenen Klassen instanziert werden (Zeile 9), welche die Endung *RequestHandler* aufweisen. Ein instanzierter Request Handler wird darauf hin anhand des Element-Namens des SMP Message Requests, welcher zum Request Handler gehört, in ein Python Dictionary abgefüllt (Zeile 10).

```
 1 def get_handlers(emulator):
 2     return handlerutils.get_handlers(emulator, globals())
 3
 4 def get_handlers(emulator, vars):
 5     handlers = {}
 6     for var in vars.iteritems():
 7         var_name = var[0]
 8         if (var_name.endswith("RequestHandler")):
 9             handler = var[1](emulator)
10             handlers[handler.REQUEST_NAME] = handler
11     return handlers
```

Der *ConnectionsRequestHandler* würde Beispielsweise mit dem Element-Namen *Connection-Request* registriert.

Der *StreamRequestHandler* ist so aufgebaut, dass er nicht geändert werden muss, falls ein neuer Request Handler dem Pythonmodul *requesthandlers* hinzugefügt wird.

Die eigentliche Abhandlung eines Requests erfolgt über die unten aufgeführten Methoden `handle()` und `_handle_request()`. `handle()` nimmt einen Request entgegen und leitet diesen zur Verarbeitung an `_handle_request()` weiter (Zeile 3). Diese Methode sucht den zum Request gehörigen Request Handler (Zeile 13), ruft diesen auf (Zeile 14) und sendet die Antwort zurück (Zeile 15).

```
 1 def handle(self):
 2     while 1:
 3         response = self._handle_request()
 4         if not response:
 5             continue
 6         self._send_response(response)
 7
 8 def _handle_request(self):
 9     request_msg = self._read_request()
10     if not request_msg:
11         return None
12     request = request_msg.body.message
13     handler = self.handlers[request.ELEMENT_NAME]
14     response = handler.handle(request)
15     return response
```

6.5.3. Konfigurationsobjekte

Der Inhalt der SMP Messages, welche der Emulator generiert, kann über Python Dictionaries in den Pythonmodulen *connections*, *ikesas*, *interfaces*, *performance* und *policies* konfiguriert werden. Diese Module enthalten die dem Namen entsprechenden Komponenten und sind im Ordner `defaultconfig` des Emulators abgelegt. Als Beispiel wollen wir die Konfiguration einer Connection betrachten.

```
connection1 = {
    'name' : "net-net1",
    'local_ipv4' : "192.168.1.1",
    'remote_ipv4' : "192.168.1.2",
    'local_port' : 3563,
    'remote_port' : 4843,
    'rekeying_time' : 10,
    'del_time' : 50,
    'dpd_delay' : 60,
    'req_policy' : PolicyTypes.CERT_NEVER_SEND,
    'retrans_seq' : 70,
    'resp_policy' : PolicyTypes.CERT_NEVER_SEND,
    'ike_enc_algs' : ( \
      IKEEncryptionAlgorithmTypes.AES_CBC_128, \
      IKEEncryptionAlgorithmTypes.DES), \
    'ike_int_algs' : (IntegrityAlgorithmTypes.HMAC_MD5_96,),
    'diffie_hellman_groups' : \
      (DiffieHellmanGroupTypes.MODP_2048_BIT,)
}
```

`connection1` ist ein Pythondictionary, welcher Key/Value Tupel beinhaltet. In ihm können alle für das *SMPObject Connection* gültigen Attribute konfiguriert werden, in dem der Value des entsprechenden Keys angepasst wird.

6.6. Performance

Um die Performance der Managementkonsole stronManager zu testen, wurden qualitative Performancetests durchgeführt. Da die Höhe der Last eines strongSwan VPN Gateways von desssen aufgebauten Tunnel abhängig ist, beschränken sich die Performancetests auf die Übertragung von mehreren tausend IKE SA's.

Resultat: Auch bei dieser hohen Last ist strongManager noch vernünftig einsetzbar.

Kapitel 7. Schlussfolgerungen

Inhaltsverzeichnis

7.1. Einleitung

In diesem Kapitel werden die Ergebnisse nochmals Punkt für Punkt erläutert und anschliessend in Bezug auf die Aufgabenstellung bewertet. Des Weiteren wird auf die Weiterführbarkeit der erstellten Komponenten eingegangen.

7.2. Bewertung der Ergebnisse

In der vorliegenden Arbeit wurden die beiden Komponenten Strong Management Protocol (SMP) und strongManager (Managementkonsole) erstellt. Die Bestandteile dieser beiden Komponenten werden nun im folgenden betrachtet.

7.2.1. Strong Management Protocol (SMP)

Das definierte Strong Management Protocol bietet die Möglichkeit für das Abfragen und Übertragen von Statusinformationen jeglicher Art. Dazu gehören Informationen über die bestehende Konfiguration von Verbindungen und Sicherheitsrichtlinien, die Performance, die aufgebauten Verbindungen und die dafür ausgehandelten Parameter. Des Weiteren unterstützt SMP die Signalisation, wodurch auf unerwartete Ereignisse auf Seiten eines VPN Gateways reagiert werden kann. Nebst dem Abfragen von Informationen können durch SMP auch Verbindungen und Sicherheitseinstellungen konfiguriert werden. Es besteht ausserdem die Möglichkeit VPN Gateways zu steuern. Diese umfasst zur Zeit die Steuerung von VPN Verbindungen und deren Sicherheitsbeziehungen. Damit die zu übertragenden Daten von unberechtigten Personen nicht eingesehen werden können, wurden XML-basierte Signatur und Verschlüsselung in das erstellte Konzept miteinbezogen. In Zukunft kann das Protokoll auch für andere Applikationen wie z.B. eine GUI-Applikation genutzt werden.

Gemäss Aufgabenstellung sollte die Kommunikationsschnittstelle flexibel, erweiterbar und programmiersprachenunabhängig sein. Das erstellte Management Protokoll verwendet XML-basierte Nachrichten. Die Programmiersprachenunabhängigkeit, Flexibilität und Erweiterbarkeit ist dadurch gewährleistet. Die bestehenden Nachrichten können einfach erweitert werden, in dem das Relax NG Schema (XML Schema Standard) angepasst wird. Des Weitern ist, durch die Verwendung von XML, SMP unabhängig vom Steuerungsprogramm.

SMP bietet Nachrichtentypen für die Abfrage, Konfiguration, Steuerung sowie Signalisierung. Bezüglich des Umfangs des Kommunikationsprotokolls konnte desshalb die Aufgabenstellung erfüllt werden.

Kompromisslose Sicherheit stellt gemäss Aufgabenstellung ein wichtiges Kriterium dar. In das Konzept von SMP wurden die Grundzügen der beiden W3C Standards XML-Signature und XML-Encryption mit einbezogen. Das erstellte Relax NG Schema, welches auf diesen Standards beruht, wurde zusätzlich bezüglich der Sicherheit gehärtet und für SMP optimiert. Dies wurde erreicht, in dem nicht relevante Funktionen entfernt wurden und nur Algorithmen, die nach heutigen Kenntnissen als sicher eingestuft sind, verwendet werden.

7.2.2. strongManager

Die Managementkonsole strongManager besteht aus der Kommunikationsschnittstelle für SMP und einer Web-Applikation. In ihr wurde der vollständige Support für das Abfragen von Statusinformationen, wie dieser in SMP spezifiziert sind, implementiert. Damit die Webapplikation direkt verwendet werden kann, beinhaltet die Managementkonsole einen integrierten Webserver, welcher nach der Installation des Managementtools sofort verfügbar ist.

Bezüglich der Kommunikationsschnittstelle für SMP, konnte mit der Implementation von Abfragemöglichkeiten für Statusinformationen das Grundziel erreicht werden. Die erstellte Kommunikationsschnittstelle erlaubt aber nebst der lokalen (über Unix Sockets) auch die dezentrale Kommunikation (TCP Socket) über das Internet. Unterstützt werden mehrere Gateways.

Die Implementation der Kommunikationsschnittstelle auf Seiten des Daemons ist optional. Weil diese in C geschrieben werden muss, wurde dieser Punkt aus Zeitgründen weggelassen. Zum Testen der Kommunikationsschnittstelle wurde aber ein Emulator erstellt, welcher ein VPN Gateway simuliert.

Die zu erstellende Web-Applikation sollte die Statusabfrage ermöglichen. Die Web-Applikation von strongManager unterstützt nebst der vollständigen Statusabfrage die Authentifizierung von Benutzern und die Möglichkeit für das Management von mehreren VPN Gateways. Die Aufgabenstellung konnte also bezüglich dieses Punktes erfüllt werden.

Die Möglichkeit zur Steuerung und Konfiguration des Daemons wurde aus Zeitgründen nicht mehr in die Webapplikation integriert. Da diese Punkte optional sind, fällt dies nicht ins Gewicht.

Als Zusatz wurde in der Webapplikation ein Weberver integriert durch welchen das On-the-Fly-Testen nach der Installation möglich ist.

7.3. Weitere Arbeiten

Da in der Managementkonsole strongManager nur die auf den Status bezogenen Informationen implementiert wurden, muss diese noch um die fehlenden Komponenten für Konfiguration, Steuerung und Notifikation erweitert werden. In einem weiteren Schritt ist die Implementierung der Sicherheitsmassnahmen (XML-Signatur und Verschlüsselung) noch durchzuführen, damit eine Kommunikation auch über unsichere Kanäle möglich wird. Dem Strong Management Protocol selber fehlt es noch an zusätzlichen Spezifikationen für Steuerungsanweisungen, sowie eine komplette Definition von Notifikationsmöglichkeiten und eventuell anderen asynchronen Nachrichten.

Anhang A. Installationsanleitung

Inhaltsverzeichnis

A.1. Systemvoraussetzungen

Um strongManager und den Emulator verwenden zu können, werden folgende Komponenten benötigt:

- Python [http://www.python.org/], Version >= 2.4.
- setuptools [http://peak.telecommunity.com/DevCenter/setuptools], Version >= 0.6
- Genshi [http://genshi.edgewall.org/wiki/Download], Version >= 0.3.3
- libxml2 [http://xmlsoft.org/downloads.html], Version >= 2.6.26
- lxml [http://codespeak.net/lxml/], Version >= 1.0.3
- Python Paste [http://pythonpaste.org/download/], Version >= 0.9.8
- flup [http://trac.saddi.com/flup], Version >= 0.5-r2106

Für SSL Unterstützung im embedded Webserver wird zusätzlich benötigt:

- pyOpenSSL [http://pyopenssl.sourceforge.net/] , Version >= 0.6

Im Folgenden werden für eine einfache Installation die entsprechenden Packages in den verschiedenen Linux Distributionen aufgeführt.

A.1.1. (K)Ubuntu 6.10 (Edgy Eft) Packages

- python2.4
- python-setuptools
- libxml2

- python-lxml
- python-paste
- (python-pyopenssl)

Genshi und Flup sind zur Zeit nicht im Ubuntu Package Repository vorhanden.

A.1.2. Gentoo Packages

- python
- setuptools
- libxml2
- lxml
- (pyopenssl)

Für Genshi, Paste und Flup existieren zur Zeit keine Gentoo Packages.

A.2. Installation strongManager

Um strongManager zu installieren, sollten folgende Schritte durchgeführt werden:

1. Python, libxml2, setuptools und falls benötigt pyopenssl mit dem Packagesystem der verwendeten Distribution installieren.
2. **tar xzvf strongManager-0.1.tar.gz**
3. **cd strongManager-0.1**
4. **sudo python setup.py install**

Nebst den Komponenten von strongManager sind im Archiv `strongManager-0.1.tar.gz` die Bibliotheken von Paste, Flup und Genshi vorhanden. Diese müssen aber separat installiert werden. Am Beispiel von Flup sieht das folgendermassen aus.

```
cd lib/flup-r2172

sudo python setup.py install
```

Nach der Installation ist strongManager und der Emulator im Ordner `site-packages` der installierten Pythonversion abgelegt (Bsp. Python 2.4 - `/usr/lib/python2.4/site-packages`).

A.2.1. Konfiguration

Die Konfiguration des strongManager und des Emulators erfolgt über Pythondateien. Die im Archiv mitgelieferten Kofigurationsdateien sind:

- `/path/to/site-packages/strongmanager/manager/defaultconfig.py` - Konfigurationsdatei, in welcher die Benutzernamen und die sha1 codierten Passwörter für die Webpplikation, sowie die VPN Gateways konfiguriert werden können.
- `/path/to/site-packages/strongmanager/emulator/defaultconfig/` - Beinhaltet alle Konfigurationsdateien für den Emulator. Diese Dateien verwendet der Emulator als Informationsquelle für die aktuelle VPN Gateway Konfiguration.

Falls man Änderungen an der Standardkonfiguration durchführen möchte, sollte man die mitgelieferten Konfigurationsdateien an einen gewünschten Ort kopieren und von `defaultconfig` in `config` umbenennen. Der Emulators oder strongManager sollte nun in dem entsprechenden Verzeichnis gestartet werden, oder es können die entsprechenden Parameter verwendet werden (`daemonize=true` / `workdir=/folder/with/conf/files`). Es wird immer zuerst im aktuellen Arbeitsverzeichnis nach dem Ordner oder der Datei `config` gesucht. Können diese nicht gefunden werden, wird auf die Standardkonfiguration zurückgegriffen.

A.2.2. Betrieb Emulator

Um den Emulator zu starten wird die Pythodatei `/usr/bin/smp-emulator.py` verwendet. Wie diese aufgerufen werden muss und welche Parameter unterstütz werden, ist aus der Ausgabe des Konsolenbefehls **python smp-emulator.py runemulator help** ersichtlich.

```
Usage:
  smp-emulator.py runemulator [emulator settings]

Optional Emulator settings: (setting=value)
  host=HOSTNAME            Hostname to listen on.

  port=PORTNUM             Port to listen on. (default 4502)

  socket=FILE              UNIX socket to listen on.
                           (default /tmp/strongmanager.sock)

  mode=MODENUM             Change the mode of socket to the
                           numeric mode.

  daemonize=BOOL           Whether to detach from terminal.

  pidfile=FILE             Write the spawned process-id to
                           this file.

  workdir=DIRECTORY        Change to this directory when
                           daemonizing.

  loglevel=LEVELNAME       The emulator log level. One of
                           DEBUG, INFO, WARNING, ERROR or
                           CRITICAL. (default INFO)

Example:
```

```
    $ smp-emulator.py runemulator \
        socket=/tmp/strongmanager.sock mode=0777 \
        daemonize=true pidfile=/var/run/smp-emulator.pid
```

Alle `emulator settings` sind optional und werden für den normalen Betrieb nicht benötigt (Standardmässig verläuft die Kommunikation über ein Unix Socket).

Falls man die Kommunikation über einen TCP Port verlaufen soll, ist es zu empfehlen den Parameter `host=0.0.0.0` zu verwenden. Dadurch lauscht der Emulator auf allen IP Adressen.

Hinweis

Zur Zeit wird nur vom Loglevel `INFO` Gebrauch gemacht.

A.2.3. Betrieb strongManager HTTP

Um strongManager mit dem integrierten Webserver (ohne Apache) zu starten, wird die Pythodatei `/usr/bin/smp-manager.py` verwendet. Wie diese aufgerufen werden muss und welche Parameter unterstütz werden, ist aus der Ausgabe des Konsolenbefehls **python smp-manager.py runhttpd help** ersichtlich.

```
Usage:
  smp-manager.py runhttpd [httpd settings]

Optional httpd settings: (setting=value)
  host=HOSTNAME          Hostname to listen on.
                         (default 172.0.0.1)

  port=PORTNUM           Port to listen on. (default 8080)

  daemonize=BOOL         Whether to detach from terminal.

  pidfile=FILE           Write the spawned process-id to
                         this file.

  workdir=DIRECTORY      Change to this directory when
                         daemonizing.

  sslpem=FILE            This an optional SSL certificate
                         file.

  workerthreads=NUMBER   Number of worker threads to create.
                         (default 3)

  debug=BOOL             If true, then tracebacks will be
                         shown in the browser.

Example:
    $ smp-manager.py runhttpd host=127.0.0.1 port=80 \
```

```
        daemonize=true pidfile=/var/run/smp-manager.pid \
```

Alle `httpd settings` sind optional und werden für den normalen Betrieb nicht benötigt. Wird mit dem Parameter `sslpem"` ein Zertifikat angegeben, verwendet der Webserver HTTPS für die Kommunikation.

Nach dem Start des strongManagers und des Emulator ist die Webapplikation unter der URL http://127.0.0.1:8080 verfügbar. Für das Login kann der Benutzername **strongSwan** und das Passwort **strongSwan** verwendet werden (Diese Werte gelten nur für die Standardkonfiguration).

A.2.4. Betrieb strongManager FastCGI

Um den FastCGI Server von strongManager zu starten, wird die Pythodatei `/usr/bin/smp-manager.py` verwendet. Wie diese aufgerufen werden muss und welche Parameter unterstütz werden, ist aus der Ausgabe des Konsolenbefehls **python smp-manager.py runfcgi -r** ersichtlich.

```
Usage:
  smp-manager.py runfcgi [fcgi settings]

Optional fcgi settings: (setting=value)
  host=HOSTNAME        Hostname to listen on.

  port=PORTNUM         Port to listen on. (default 8025)

  socket=FILE          UNIX socket to listen on.

  method=IMPL          prefork or threaded (default
                       prefork)

  maxrequests=NUMBER   Number of requests a child handles
                       before it is killed and a new child
                       is forked (default 0 = no limit).

  maxspare=NUMBER      Max number of spare processes /
                       threads. (default 5)

  minspare=NUMBER      Min number of spare processes /
                       threads. (default 2)

  maxchildren=NUMBER   Hard limit number of processes /
                       threads. (default 50)

  daemonize=BOOL       Whether to detach from terminal.

  pidfile=FILE         Write the spawned process-id to
                       this file.
```

```
  workdir=DIRECTORY    Change to this directory when
                       daemonizing.

  debug=BOOL           If true, then tracebacks will
                       be shown in the browser.

Examples:
  Run a "standard" fastcgi process on a file-descriptor
  (for webservers which spawn your processes for you).
    $ smp-manager.py runfcgi method=threaded

  Run a fastcgi server on a TCP host/port.
    $ smp-manager.py runfcgi method=prefork \
        host=127.0.0.1 port=8025

  Run a fastcgi server on a UNIX domain socket
  (posix platforms only).
    $ smp-manager.py runfcgi method=prefork \
        socket=/tmp/fcgi.sock

  Run a fastCGI as a daemon and write the spawned
  PID in a file.
    $ smp-manager.py runfcgi socket=/tmp/fcgi.sock \
        method=prefork daemonize=true \
        pidfile=/var/run/django-fcgi.pid
```

Alle `fcgi settings` sind optional und werden für den normalen Betrieb nicht benötigt.

Grundsätzlich sollte der FastCGI Server nicht selber vom Benutzer gestartet werden, da dieser nur mit einem externen Webserver wie Apache zum Einsatz kommt.

A.3. Apache 2

A.3.1. FastCGI

Für den Betrieb von strongManager muss in Apache das Modul `mod_fastcgi` über das Package System der verwendeten Distribution installiert werden. Falls benötigt, sind hierzu mehr Informationen auf http://www.fastcgi.com/mod_fastcgi/docs/mod_fastcgi.html verfügbar.

A.3.2. Virtual Host

Für die Konfiguration des Virtual Host, kann das unten aufgeführte Template verwendet werden.

```
NameVirtualHost *:80
<VirtualHost *:80>
        DocumentRoot /var/www/myfiles
```

```
        AddHandler fastcgi-script .fcgi
        RewriteEngine On
        RewriteRule ^/static(.*)$ /templates/static$1 [QSA,L]
        RewriteCond %{REQUEST_FILENAME} !-f
        RewriteRule ^/(.*)$ /strongmanager.fcgi/$1 [QSA,L]
</VirtualHost>
```

Damit die Web-Applikation funktioniert, muss die Datei `smp-manager.fcgi`, der Ordner mit den statischen Daten (Bilder, Stylesheets) und die Templates des installierten strongManagers in den Webordner kopiert werden.

```
cd /path/to/site-packages/strongmanager
cp manager/servers/strongmanager.fcgi /var/www/myfiles
cp manager/defaultconfig.py /var/www/myfiles/config.py
cp manager/templates /var/www/myfiles/
```

Nach der Konfiguration von Apache, muss nur noch die Datei `strongmanager.fcgi` ausführbar gemacht (**chmod 755**) und der Emulator gestartet werden, damit die Webapplikation einsatzbereit ist.

Bemerkung

Die statischen Dateien sollten aus Performancegründen wie oben konfiguriert immer über den Webserver ausgeliefert werden.

Hinweis

Falls beim Neustart des Apache Webservers Fehler auftreten, wurde möglicherweise das *rewrite* Modul nicht eingeschaltet. In diesem Fall kann das Modul unter Debian mit **sudo a2enmod rewrite** aktiviert werden.

Anhang B. Projektmanagement

Inhaltsverzeichnis

B.1. Vorgehensweise

Das Softwareprojekt wurde mit einer agilen und pragmatischen Vorgehensweise durchgeführt. Diese beinhaltete die iterative und inkrementelle Entwicklung und nur so viel Analyse und Design wie benötigt.

B.2. Zeitplanung

Im Durchschnitt wurde pro Person 46.2 Stunden pro Woche gearbeitet. Die detaillierte Zeitplanung ist unten aufgeführt.

Woche	Datum	Zeit Soll	Zeit Ist
1	23.10.06 - 27.10.06	80	82
2	30.10.06 - 03.11.06	80	80.5
3	06.11.06 - 10.11.06	80	82.5
4	13.10.06 - 17.10.06	80	92.5
5	20.10.06 - 24.10.06	80	99.5
6	27.10.06 - 01.11.06	80	103.5
7	04.11.06 - 08.11.06	80	125.5
8	11.11.06 - 15.11.06	64	73
Total		**624**	**739**

B.2.1. Tasks Woche 1

Task	Zeit Soll	Zeit ist

Sitzung	3	3
Dokumentation/Administratives	12	12.5
Backup Wiki	0.5	0.5
PC's / Wiki aufsetzen	16.5	18.5
strongSwan 4 Paket erstellen	2	4
Technologiestudium	22	22
Brainstorming	4	4
Schnittstellen Definition Grundkonzept	16	17.5
Total	**80**	**82**

B.2.2. Tasks Woche 2

Task	Zeit Soll	Zeit ist
Sitzung	2	3
Dokumentation/Administratives	8	6
Backup Wiki	0.5	0.5
Technologiestudium	8	12
Schnittstellen Definition Grundkonzept	23.5	24
Schnittstellen Definition Statusabfrage	24	18
Dokumenttyp Definition	14	16
Total	**80**	**80.5**

B.2.3. Tasks Woche 3

Task	Zeit Soll	Zeit ist
Sitzung	2	2
Dokumentation/Administratives	8	16
Backup Wiki	0.5	0.5
Schnittstellen Definition Steuerung	8	6
Überarbeitung Grundkonzept	16	4
Schnittstellen Definition Statusabfrage	20	28
Schnittstellen Definition Konfiguration	10	8
Schnittstellen Definition Signalisation	10	8
Dokumenttyp Definition	5.5	10
Total	**80**	**82.5**

Schnittstellen Definition abgeschlossen

B.2.4. Tasks Woche 4

Task	Zeit Soll	Zeit ist
Sitzung	2	1
Dokumentation/Administratives	8	9
Backup Wiki	0.5	0.5
Schnittstellen Definition überarbeiten	11.5	14
Technologiestudium	4	8
Anforderungsspezifikation Managementkonsole	8	8
Design Managementkonsole	16	12
Implementierung Managementkonsole, Auflistung Connections	30	40
Total	**80**	**92.5**

B.2.5. Tasks Woche 5

Task	Zeit Soll	Zeit ist
Sitzung	2	1
Dokumentation/Administratives	8	10
Backup Wiki	0.5	0.5
Design Managementkonsole	8	7
Implementierung Managementkonsole, IKE SA's zu Connections, CHILD SA's zu IKE SA, Details zu IKE SA	61.5	81
Total	**80**	**99.5**

B.2.6. Tasks Woche 6

Task	Zeit Soll	Zeit ist
Sitzung	2	1
Dokumentation/Administratives	8	36
Backup Wiki	0.5	0.5
Design Managementkonsole	8	6
Implementierung Managementkonsole, Details zu CHILD SA, Auflistung Policies, Details Connection, Detail Policy	61.5	60
Total	**80**	**103.5**

B.2.7. Tasks Woche 7

Task	Zeit Soll	Zeit ist
Sitzung	2	1

Dokumentation/Administratives	16	60.5
Backup Wiki	0.5	0.5
Implementierung Managemenkonsole, Authentication, Auflistung aller Gateways, Auflistung Performance, Auflistung Interfaces	61.5	64
Total	**80**	**125.5**

B.2.8. Tasks Woche 8

Task	Zeit Soll	Zeit ist
Sitzung	2	1
Dokumentation/Administratives	26	34
Refactoring	16	18
Drucken und binden	3	3
Erstellen Präsentationsplakat	5	5
Präsentation	12	12
Total	**64**	**73**

Projekt abgeschlossen

Anhang C. Codestatistik

Unten aufgeführt sind die Anzahl Codezeilen in den entsprechenden Komponenten:

- **Webapplikation und Kommunikation:** 531
- **Emulator:** 659
- **Model:** 292
- **Unit-Tests:** 689
- **Total:** 2171

Nicht mit einberechnet wurde das Relax NG Schema von SMP, die Templates, die CSS Datei und die XML-Dateien, welche für die Unit Tests verwendet wurden.

Anhang D. Persönliche Berichte

Inhaltsverzeichnis

D.1. Andreas Eigenmann

Bei Herrn Steffen haben wir schon unsere erste Arbeit durchgeführt und seine Prioritäten und Vorgehensweisen haben uns sehr angesprochen. Nach dem wir in unserer zweiten Studienarbeit ein reines Softwareprojekt durchgeführt haben, merkten wir schnell dass unsere Stärken eher in einem konzeptionellen und web-basierten Projekt liegt. Entsprechend konnte uns Herr Steffen auch eine geeignete Arbeit vorschlagen.

Der erste Teil unserer Arbeit bestand in der Konzeptionierung des Managmentprotokolls. Diese Aufgabe bedurfte einer grossen Einarbeitung in die Technologien IPsec, IKE und strongSwan selber, und stellte deshalb eine grosse und zeitaufwendige Herausforderung für uns dar. Einige Implementationsabhängige Themen blieben dann aber doch Unklar und mussten mit den Betreuern geklärt werden. Anfangs haben wir uns zu sehr an die RFC Standards gehalten, weicht jedoch die Implementation von strongSwan ein Wenig davon ab. Die Schnittstellenspezifikation musste daher einige Male umdesignt werden. Auch später bei der Implementierung konnten noch einige Ungereihmtheiten ausgemerzt werden, die beim Entwerfen zu unserem Erstaunen unentdeckt blieben. Positiv überrascht war ich von der XML Schema Sprache Relax NG, die sich als einfache und elegante, aber dennoch mächtige Beschreibungssprache entpuppte. In direktem Vergleich zu der komplizierten und mächtigen W3C XML-Schema Spezifikation steht sie dieser in praktisch nichts nach.

Für die Implementierung setzten wir auf viele neu Technologien und es war wieder eine entsprechende Einarbeitungszeit von Nöten. Obwohl ich schon einige Erfahrung in Python hatte, musste ich mich in einige Gebiet vertieft einarbeiten. Die Herausforderung war also auch hier nicht zu unterschätzen. Die bereits vorhanden Kenntnisse in den Webframeworks Django und Ruby on Rails kamen uns sehr zu gute. Und nicht selten waren bereits bewährten Konzepte aus diesen Frameworks die Grundlage für unsere Implementation.

Ganz in der Manier von strongSwan wurde für diese Diplomarbeit nur Open Source Software eingesetzt. Dies war schon in den Studienarbeiten immer unser Ziel, haben wir es dort jedoch nie ganz erreicht. Wir wurden bestätigt, dass freie Software professionell und gut einsetzbar ist und ein Ersatz für proprietäre Software darstellt. In Zukunft werde ich weiterhin solche Software einsetzen.

Obwohl ich heute vor allem bei der Implementierung wenige Sachen anders gelöst hätte, sehe ich das Projekt als gelungen an und erachte vor allem die eingesetzten Technologien als die richtigen. Ich konnte wieder sehr viel lernen und neue Technologien und Konzepte kennen lernen. Vor allem aber war die Diplomarbeit durch die zweiteilige Gliederung sehr abwechslungsreich und interessant. Uns wurde sehr viel Freiheit bei der Implementierung und den verwendeten Technologien gelassen, was ich sehr positiv bewerte.

Da ich mit Joël Stillhart schon die zwei Studienarbeiten erfolgreich abgeschlossen habe, waren wir bereits ein gut eingespieltes Team. Ich möchte ihm darum auch hier für seine erneute gute Zusammenarbeit und die vielen aufgewendeten Stunden danken. Des Weiteren möchte ich hiermit unseren Betreuern Prof. Dr. Andreas Steffen und Dipl. Ing. Martin Willi für ihre kompetente und freundliche Unterstützung danken.

D.2. Joël Stillhart

Bereits einige Monate vor Beginn dieser Diplomarbeit haben wir Herrn Steffen nach einem Thema für die kommenden Arbeiten gefragt. Da wir bereits sehr gute Erfahrungen aus der ersten Studienarbeit mit ihm hatten, lag dieser Entschluss nahe. Herr Steffen empfahl uns das Thema „Management Tool für Linux strongSwan“ , da er dort unsere Stärken sah - wobei er nicht unrecht hatte. Nach dem wir den Zuschlag für das Thema erhalten haben, sind wir mit viel Elan an die bevorstehende Arbeit herangegangen.

Zu Beginn sollte ein Konzept für eine Kommunikationsschnittstelle erarbeitet werde. Diese Aufgabe, stellte hohe Anforderungen an den eigenen Intellekt - sollte doch das Konzept flexibel und einfach zugleich sein. Um eine gute Basis zu schaffen, mussten wir uns intensiv mit den unterschiedlichsten Technologien beschäftigen. Nebst den Details von strongSwan, IPsec und IKE war auch die verwendete Schemasprache Relax NG Neuland. Ich denke in dieser ersten Phase konnten ich viel lernen und meinen Horizont erweitern.

In der zweiten Phase der Diplomarbeit sollte das erstellte Konzept anhand einer Managementkonsole verifiziert werden. Auch hier war die Herausforderung wiederum gross, umfasste doch auch diese Phase viele neue Themenbereiche. Praktisch ein Novum für mich war die Programmiersprache Python, welche zur Entwicklung derauch Managementkonsole verwendet wurde. Ich kann sagen, ich habe die Vorzüge der Sprache lieben gelernt.

Zum Schluss möchte ich betonen, dass mir vor allem die Abwechslung zwischen Konzeption und Programmierarbeit, besonders zugesprochen hat. Des Weiteren möchte ich Andreas Eigenmann für seine geleistete Arbeit danken. Für mich war es immer erfreulich mit ihm zu arbeiten. An diesem Punkt möchte ich ausserdem unseren beiden Betreuern Prof. Dr. Andreas Steffen und Dipl. Ing. Martin Willi für ihre Unterstützung und aufgeopferte Zeit danken.

Literaturverzeichnis

Webseiten IPsec/IKE

[1] *RFC 4306*. Internet Key Exchange (IKEv2) Protocol. http://tools.ietf.org/html/rfc4306 .

[2] *RFC 4301*. Security Architecture for the Internet Protocol . http://tools.ietf.org/html/rfc4301 .

[3] *Einführung zu IPsec, AH und EPS*. http://de.wikipedia.org/wiki/IPsec .

[4] *strongSwan Official Page*. http://www.strongswan.org .

Webseiten Schnittstellendefinition

[5] *RFC 3275*. XML-Signature Syntax and Processing. http://tools.ietf.org/html/rfc3275 .

[6] *RFC 3447*. PKCS #1: RSA Cryptography Specifications Version 2.1 . http://tools.ietf.org/html/rfc3447 .

[7] *RFC 2440*. OpenPGP Message Format. http://tools.ietf.org/html/rfc2440 .

[8] *RFC 2459*. Internet X.509 Public Key Infrastructure Certificate and CRL Profile . http://tools.ietf.org/html/rfc2459 .

[9] *W3C Canonical XML Version 1.0*. http://www.w3.org/TR/xml-c14n .

[10] *XML Encryption Syntax and Processing*. http://www.w3.org/TR/xmlenc-core .

[11] *Base 64 Codierung*. http://de.wikipedia.org/wiki/Base64 .

Webseiten Technologien

[12] *Offizielle Webseite von RELAX NG*. http://relaxng.org .

[13] *Relax NG, Compared*. http://www.xml.com/lpt/a/2002/01/23/relaxng.html .

[14] *OASIS RELAX NG TC*. http://www.oasis-open.org/committees/relax-ng .

[15] *Offizielle Webseite von Python*. http://www.python.org .

[16] *Offizielle Webseite von lxml2*. http://xmlsoft.org .

[17] *Offizielle Webseite von lxml*. http://codespeak.net/lxml .

[18] *Offizielle Webseite von Flup*. http://www.saddi.com/software/flup .

[19] *Offizielle Webseite von Python Paste*. http://pythonpaste.org .

[20] *Offizielle Webseite von Genshi*. http://genshi.edgewall.org .

[21] *Offizielle Webseite von Django*. http://www.djangoproject.com/ .

[22] *Offizielle Webseite des WSGI Standards*. http://www.python.org/dev/peps/pep-0333/ .

[23] *WSGI auf Python Wiki*. http://wiki.python.de/WSGI .

Bücher

[24] *DocBook-XML*. Einführung und Anwendung. Lars Trieloff. 3-8266-1519-0. 2005. mitp.

[25] *XML kurz & gut*. Simon St.Laurent. Michael Fitzgerald. 3-89721-516-0. 2006. O'Reilly.

[26] *XML in a Nutshell*. Elliotte Rusty Harold. W. Scott Means. 3-89721-339-7. 2005. O'Reilly.

[27] *Python Ge-Packt*. Michael Weigend. 3-8266-1659-6. 2006. mitp.

[28] *Python in a Nutshell*. Alex Martelli. 0-596-10046-9. 2006. O'Reilly.

[29] *Python Cookbook*. Alex Martelli. Anna Martelli Ravenscroft. David Ascher. 0-596-00797-3. 2005. O'Reilly.

Glossar

Advanced Encryption Standard (AES)
: Ein symmetrisches Kryptosystem, welches der Nachfolger von DES bzw. 3DES ist.

Authentication Header (AH)
: IPsec Protokoll, welches die Integrität der übertragenen IP Pakete sicher stellt und den Sender authentifiziert.

Base64
: Ein Verfahren zur Kodierung von 8-Bit-Binärdaten in eine Zeichenfolge, die nur aus wenigen Codepage-unabhängigen ASCII-Zeichen besteht.

Canonical XML (C14N)
: W3C Standard um von XML Daten eine spezielle serialisierte Form zu erhalten, welche von XML Signature und XML Encryption verwendet werden kann.

Encapsulation Security Payload (ESP)
: IPsec Protokoll, welches die Authentifizierung, Integrität und Vertraulichkeit von IP Paketen sicher stellt.

Document Object Model (DOM)
: Programmierschnittstelle für den Zugriff auf XML-Dokumente auf Basis eines Objektbaumes.

Flup
: Sammlung von WSGI Modulen.

FastCGI (Fast Common Gateway Interface)
: Standard für die Einbindung externer Software zur Generierung dynamischer Webseiten in einem Webserver.

Genshi
: Template Engine um Python Code in HTML Seiten auszuwerten.

Keyed-Hash Message Authentication Code (HMAC)
: HMAC ist eine Art Message Authentication Code (MAC), der basierend auf einer kryptografischen Hash-Funktion berechnet wird.

IKE SA
: Sicherheitsbeziehung, welche von IKE für das Aushandeln von Child SA's verwendet wird.

IPsec
: Stellt Sicherheit auf dem IP Layer zur Verfügung. Darunter fällt z.B. die Verschlüsselung von IP Paketen und die Sicherstellung derer Herkunft.

Internet Key Exchange Protokoll (IKE)
: Ist unter IPsec verantwortlich für den automatischen Austausch der Session Schlüssel mittels dem Diffie Hellmann Algorithmus.

libxml2
: XML C Parser um XML Dokumente zu verarbeiten.

lxml
: Python Binding für libxml2.

OpenPGP
: Standard für Verschlüsselungs-Software.

Public Key Cryptography Standards (PKCS)
: Eine Reihe von kryptografischen Spezifikationen von den RSA Laboratorien.

Python
: Eine dynamische objektorientierte Programmiersprache.

Python Enhancement Proposal (PEP)
: Ein Design Dokument, welches der Python Cummunity Informationen zur Verfügung stellt oder eine neue Komponente von Python beschreibt.

Python Paste
: Python Web-Framework welches WSGI verwendet.

Request for Comments (RFC)
: RFC sind eine Reihe von technischen und organisatorischen Dokumenten des RFC-Editor zum Internet.

Remote Procedure Call (RPC)
: Aufruf einer Methode welche sich auf einer anderen Datenverarbeitungsmaschine befindet.

Relax NG
: Einfache Schemasprache für XML zur Spezifikation von Muster für die Struktur und den Inhalt eines XML-Dokuments.

RSA
: Asymmetrisches Verfahren oder Algorithmus zur Verschlüsselung diskreter Daten, der verschiedene Schlüssel zum ver- und entschlüsseln verwendet.

Simple Api for XML (SAX)
: Programmierschnittstelle für XML-Dokumente auf Basis von Rückruffunktionen.

Simple Object Access Protocol (SOAP)
: Auf XML-RPC basiertes Protokoll um Daten zwischen Webservices auszutauschen und RPC's durchzuführen.

Security Assertion Markup Language (SAML)
: XML-basierte Sprache für Sicherheitsbestätigungen.

Security Parameter Index (SPI)
: Identifiziert mit einer IP Adresse und einem IPsec Protokoll Typ eine Security Association.

Security Policy Database (SPD)
: Kernel Datenbank in welcher Security Policies abgelegt werden.

Security Association Database (SAD)
: Kernel Datenbank welche die Security Associations beinhaltet. Jeder Eintrag wird eindeutig identifiziert anhand einer IP Adresse, einem IPsec Protokoll Typ und einer SPI

strongSwan
: Open Source IPsec VPN Lösung.

strongSwan Policy
: Beinhaltet Informationen, welche für das Aushandeln einer Child SA benötigt werden. Dies sind Algorithmen für die Verschlüsselung und Integritätssicherung, ID's, CA's und Trafficselektoren. Die strongSwan Policy sollte nicht mit einem Policyeintrag in der SPD verwechselt werden. Eine Policyeintrag in der SPD basiert aber auf einer strongSwan Policy.

strongSwan Connection
: Beinhaltet Informationen, welche für das Aushandeln einer IKE SA benötigt werden. Dies sind Algorithmen für die Verschlüsselung (Diffie Hellman Gruppe) und Integritätssicherung, IP's der beteiligten Kommunikationspartner, etc..

Strong Management Protocol (SMP)
: XML basiertes Protokoll für das Management des IKEv2 Daemons und Bestandteil dieser Diplomarbeit.

Secure Socket Layer (SSL)
: Verschlüsselungsprotokoll für die Datenübertragung im Internet.

Secure Hash Algorithm (SHA)
: SHA bezeichnet eine Gruppe standardisierter kryptographischer Hash-Funktionen.

Security Association (SA/Child SA)
: Eine SA/Child SA beschreibt den Schutz einer Kommunikation zwischen zwei Host's oder Security Gateways. In ihr werden unter anderem die Algorithmen und Schlüssel für die AH Authentisierung und die ESP Verschlüsselung festgelegt.

Transport Layer Security (TLS)
: Verschlüsselungsprotokoll für die Datenübertragung im Internet. TLS ist die standardisierte Weiterentwicklung von SSL.

VPN (Virtual Private Network)
: Über Tunnel verbundenes Netzwerk. VPN's verlaufen üblicherweise über das Internet und verwenden kryptografisch geschützte Verbindungen.

VPN Gateway Manager (VGM)
: Kommunikationsschnittstelle für SMP auf Seiten einer Managementkonsole.

VPN Gateway Manager Agent (VGMA)
: Kommunikationsschnittstelle für SMP auf Seiten des VPN Gateways.

WSGI (Web Server Gateway Interface)
: Standard für ein Interface zwischen einem Webserver und einer Python Webapplikation.

The World Wide Web Consortium (W3C)
: Gremium, welches sich mit der Standardisierung das Web betreffender Techniken befasst.

Extensible Markup Language (XML)
: Standard zur Modellierung von halb-strukturierten Daten in Form einer Baumstruktur.

XML-Remote Procedure Call (XML-RPC)
: XML basiertes RPC Protokoll, dessen Kommunikation meist mittels HTTP verläuft. Es eignet sich besonders für die Kommunikation zwischen Webservices.

XML Signature
: Definiert die Vorgehensweise beim Signieren von XML Dokumenten. XML Signature ist ein W3C Standard.

XML Encryption
: Definiert eine Reihe von Möglichkeiten, wie XML Dokumente ver- und entschlüsselt werden können. XML Encryption ist ein W3C Standard.

X.509
: X.509 ist ein ITU-T-Standard für eine Public- Key-Infrastruktur und derzeit der wichtigste Standard für digitale Zertifikate.

Kolophon

Für die Erstellung der Diplomarbeit wurde ausschliesslich Open Source Software eingesetzt.

Dieses Dokument wurde mit DocBook in der Version 4.4 geschrieben. Die FO Dateien wurden mit Xalan-J in der Version 2.7 und die PDF Dateien mit Apache FOP in der Version 0.93pre generiert. Die ganze Transformation erfolgte dabei durch das Kalliope Eclipse Plugin in der Version 0.1.0.