HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

VPN Endpunktauthentifizierung

DNS basierte VPN Endpunktauthentifizierung für strongSwan

Reto Guadagnini

22. Januar 2012

VPN Endpunktauthentifizierung

Dieser Projektarbeitsbericht wurde im Rahmen des Master of Science in Engineering Studienganges an der Hochschule für Technik Rapperswil erstellt.

Student

Reto Guadagnini
Via Dim Lej 5
CH-7500 St. Moritz

Advisor

Prof. Dr. Andreas Steffen
Hochschule für Technik Rapperswil
Oberseestrasse 10
CH-8640 Rapperswil

Abstract

Bei strongSwan handelt es sich um eine freie Implementierung des IPsec Protokolls für Linux. strongSwan implementiert das IKEv2 Protokoll und unterstützt die darin spezifizierte Authentifizierung von VPN Endpunkten über Pre-Shared Keys, das Extensible Authentication Protocol und über Zertifikate. Das vorliegende Dokument schlägt ein weiteres Authentifizierungsverfahren für VPN Endpunkte in strongSwan basierend auf Einträgen im Domain Name System (DNS) mit DNSSEC vor (bei DNSEC handelt es sich um eine Erweiterung des Domain Name Systems um Sicherheitsfunktionen, welche die Authentizität und Integrität der im DNS gespeicherten Einträge gewährleisten). Als DNS Einträge werden im vorgeschlagenen Verfahren die in RFC 4025 spezifizierten IPSECKEY Resource Records eingesetzt.

Im Rahmen eines Proof of Concept wurde gezeigt, dass die zur Realisierung des vorgeschlagenen Authentifizierungsverfahrens benötigten Technologien bereitstehen. Da für das vorgeschlagene Authentifizierungsverfahren ein DNS Resolver benötigt wird, wurden verschiedene DNS Resolver Bibliotheken für den Einsatz in strongSwan evaluiert und eine dieser Bibliothek wurde für den Einsatz in strongSwan ausgewählt. Weiter wurde ein Konzept zur Implementierung des vorgeschlagenen Authentifizierungsverfahrens in strongSwan erstellt und mit der Implementierung des darin vorgesehenen DNS Resolvers begonnen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1 Projektbeschreibung

1.1 Ausgangslage

Bei strongSwan [1] handelt es sich um eine freie Lösung zur Realisierung von VPNs basierend auf dem IPsec Protokoll [2] für Linux. strongSwan unterstützt momentan drei Methoden zur Authentifizierung von VPN Endpunkten via IKEv2 [1]:

- Pre-Shared Keys (PSK)
- Extensible Authentication Protocol (EAP)
- X.509 und PGP Zertifikate

Damit die Authentifizierung über Zertifikate sicher ist, bedarf es einer Public Key Infrastructure, welche die Echtheitsprüfung von Zertifikaten ermöglicht (oder die Kommunikationspartner haben sich beim Zertifikatsaustausch bereits authentifiziert, z.B. durch persönliche Übergabe des Zertifikats unter Vorlage des Reisepasses, wodurch sie die ausgetauschten Zertifikate als authentisch ansehen können). Der Unterhalt einer solchen Infrastruktur stellt einen zusätzlichen Aufwand dar. Daher wäre es wünschenswert, wenn man auf sie verzichten könnte und trotzdem nicht auf „Pre-Shared Keys“ oder das „Extensible Authentication Protocol“ zwecks Endpunktauthentifizierung zurückgreifen müsste.
Das Domain Name System mit DNSSEC [3] stellt eine mögliche Alternative zur Verwendung einer "Public Key Infrastructure" dar. DNSSEC erweitert das DNS System um Sicherheitsfunktionen, welche die Authentizität und Integrität der im DNS gespeicherten Einträge gewährleisten [3]. Der Public Key eines VPN Endpunkts könnte zusammen mit dem Endpunktnamen (Domain Name oder IP Adresse) im DNS System abgelegt werden. Da durch DNSSEC die Authentizität und Integrität der DNS Einträge gewährleistet ist, käme der DNS Eintrag einem Zertifikat gleich und die Authentifizierung eines VPN Endpunktes könnte dann über dessen DNS Eintrag statt über dessen Zertifikat erfolgen.

1.2 Zielsetzung

Möglichkeiten zur Speicherung der Public Keys von VPN Endpunkten im DNS System zwecks Authentifizierung der VPN Endpunkte sind zu suchen. Werden passende Möglichkeiten gefunden, so sind diese im Rahmen eines Proof of Concepts zu prüfen. Weiter ist ein Grobkonzept zur Implementierung eines auf diesen DNS Einträgen basierenden Authentifizierungsmechanismus in strongSwan zu erstellen.

1.3 Zu erwartende Ergebnisse

- Konzept zur Authentifizierung von VPN Endpunkten durch DNS Einträge.
- Proof of Concept (Konzeptnachweis) zum aufgestellten Konzept.
- Grobkonzept zur Implementierung eines entsprechenden Authentifizierungsverfahrens in strongSwan.

1.4 Projektplanung

Die Planung der Projektarbeit wurde in einem separaten Dokument (Planung_MSE_Projekt_1.xlsx) durchgeführt. Das Balkendiagramm aus diesem Dokument befindet sich auch im Anhang unter Kapitel 10.1.

In den folgenden Kapiteln wird erläutert, wie bei der Lösung dieser Aufgaben im Rahmen der Projektarbeit vorgegangen wurde.

2 Einleitung

Bei strongSwan [1] handelt es sich um eine freie Implementierung des IPsec Protokolls [2] für Linux. StrongSwan implementiert das IKEv2 Protokoll und unterstützt die Authentifizierung von VPN Endpunkten über Pre-Shared Keys, das Extensible Authentication Protocol und über Zertifikate [1]. Für die Authentifizierung der VPN Endpunkte über Zertifikate bedarf es einer Public Key Infrastructure über welche sich die Zertifikate validieren (auf ihre Echtheit überprüfen) lassen. In der vorliegenden Arbeit schlagen wir als erstes ein weiteres Authentifizierungsverfahren für die VPN Endpunkte in strongSwan vor das ohne Public Key Infrastructure auskommt. Das vorgeschlagene Authentifizierungsverfahren basiert auf im DNS abgelegten Public Keys der VPN Endpunkte, welche sich mittels DNSSEC validieren lassen. Es ersetzt also gewissermassen die Public Key Infrastructure des zertifikatbasierten Authentifizierungsverfahrens durch das DNS mit DNSSEC. Dann zeigen wir im Rahmen eines Proof of Concept, dass die für die Realisierung des vorgeschlagenen Authentifizierungsverfahrens notwendigen Technologien bereitstehen und entwickeln ein Konzept zur Implementierung des neuen Authentifizierungsverfahrens in strongSwan. Da das neue Authentifizierungsverfahren einen DNS Resolver benötigt, evaluieren wir DNS Resolver Bibliotheken für den Einsatz in strongSwan. Basierend auf dem Sieger dieser Evaluation beginnen wir schliesslich mit der Implementierung eines DNS Resolvers für strongSwan, welcher den ersten Bestandteil der Implementierung des neuen Authentifizierungsverfahrens in strongSwan bilden wird.

3 Konzept zur VPN Endpunktauthentifizierung per DNS

Ziel dieses Kapitels ist es aufzuzeigen, wie die Authentifizierung eines IPsec VPN Endpunktes per DNS und dessen Erweiterung DNSSEC erfolgen könnte, sowie die an der vorgeschlagenen Lösung beteiligten Technologien vorzustellen. Wir stellen in diesem Kapitel also ein neues auf DNS mit DNSSEC basierendes Authentifizierungsverfahren vor VPN Endpunkte vor.
Als erstes beschäftigen wir uns mit der Frage, wozu die Authentifizierung der VPN Endpunkte notwendig ist und erläutern dann kurz das auf Zertifikaten beruhende Authentifizierungs-verfahren, welches von strongSwan bereits unterstützt wird. Schlussendlich stellen wir das Konzept zur Authentifizierung der VPN Endpunkte per DNS mit DNSSEC vor.

3.1 Endpunktauthentifizierung: Weshalb?

Betrachten wir exemplarisch folgendes „Site to Site“ VPN:

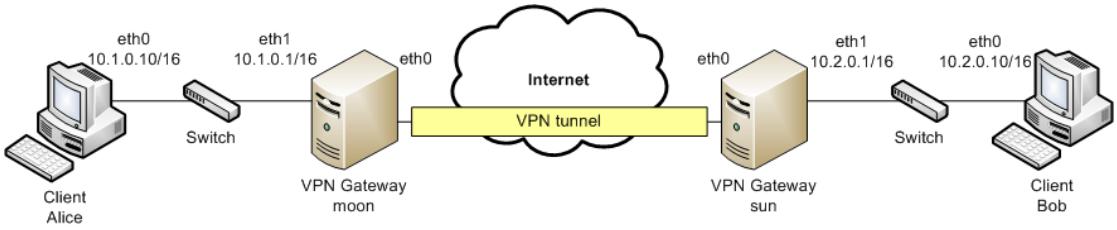


Abbildung 1: "Site-to-Site" VPN

Der VPN Tunnel in Abbildung 1 verbindet die beiden Subnetze 10.1.0.0/16 und 10.2.0.0/16 zu einem VPN und wird mithilfe des IPsec Protokolls realisiert. Die VPN Gateways "moon" und "sun" bilden dabei die VPN Endpunkte. IPsec verwendet die Protokolle AH und/oder ESP um die Integrität, Authentizität (genauer: „data origin authentication“) und Vertraulichkeit (nur mit ESP) der durch den Tunnel übertragenen Pakete zu gewährleisten. Die Eigenschaften des VPN Tunnels werden durch sogenannte SAs (Security Associations) beschrieben. Zum Aufbau dieser SAs (und damit zum Aufbau des Tunnels) wird das IKE Protokoll verwendet [2]. IKE [5] handelt die SAs für den VPN Tunnel aus und tauscht dabei die kryptographischen Schlüssel aus, welche zur Authentifizierung/Verschlüsselung der über den Tunnel übertragenen Pakete benötigt werden. Ein VPN Endpunkt (z.B. ein VPN Gateway) wird durch seine IP Adresse oder seinen FQDN (fully qualified domain name) identifiziert. Da ein Angreifer sowohl die IP Adresse (z.B. via IP-Spoofing) als auch den FQDN (z.B. via DNS Cache Poisoning) des VPN Gateways annehmen könnte, eignen sich diese nicht zur eindeutigen Identifizierung des VPN Endpunktes. Will beispielsweise in Abbildung 1 der VPN Gateway moon mit dem VPN Gateway sun eine Verbindung aufbauen, so könnte sich ein Angreifer als sun ausgeben und moon würde mit dem Angreifer eine VPN Verbindung herstellen. Der Tunnel dieser Verbindung wäre zwar sicher im Sinne der durch AH/ESP gewährleisteten Integrität, "data origin authentication" und Vertraulichkeit, aber er würde das Subnetz von moon mit dem Subnetz des Angreifers verbinden was es zu verhindern gilt. Damit dieses Szenario nicht eintreten kann, überprüft IKE beim Aushandeln der SAs die

Identität der Gegenstelle. Die VPN Endpunkte werden durch IKE also authentifiziert. Der IKEv2 Standard nennt drei Methoden zur VPN Endpunktauthentifizierung [5]:

- Shared Secret
- Digitale Signaturen (mit zugehörigen Zertifikaten)
- EAP

Um zu verstehen wie die Authentifizierung der VPN Endpunkte durch IKEv2 erfolgt, müssen wir das IKEv2 Protokoll etwas näher betrachten:

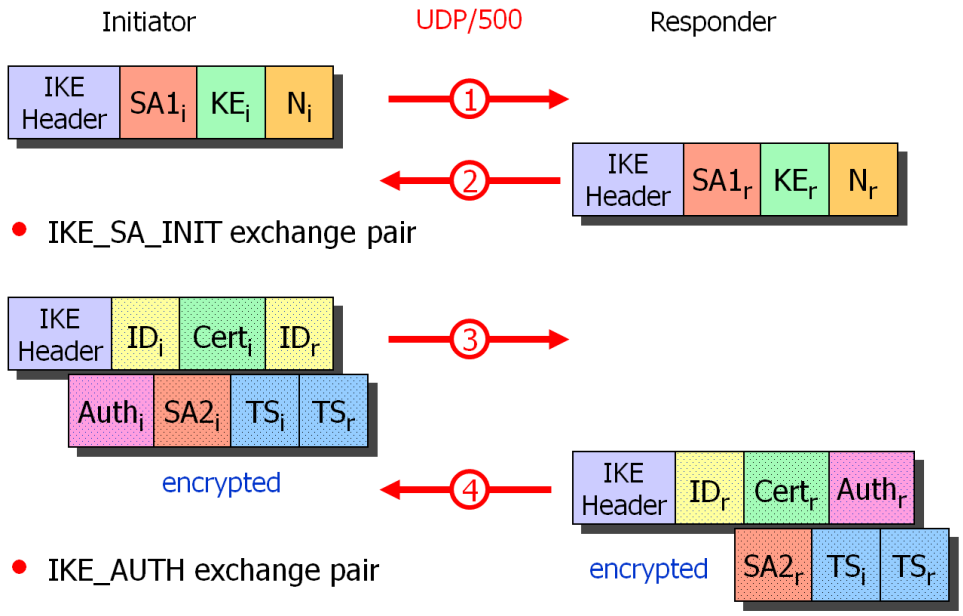


Abbildung 2: Aufbau einer IKE_SA und CHILD_SA mittels IKEv2 [6]

Die Aufgabe von IKE ist es die zur Realisierung eines VPNs nötigen SAs aufzubauen. IKE unterscheidet dabei zwischen IKE_SAs und CHILD_SAs. Bei der IKE_SA handelt es sich um eine SA welche von IKE selbst verwendet wird. Über die IKE_SA baut IKE bei Bedarf CHILD_SAs auf. Die CHILD_SAs realisieren dann die eigentlichen VPN Verbindungen für die Nutzdaten.

Eine IKE_SA wird durch die in Abbildung 2 dargestellte Protokollsequenz aufgebaut. Zum Aufbau der IKE_SA erfolgt als erstes der Austausch der IKE_SA_INIT Meldungen (Meldung 1 und 2 in Abbildung 2). Mit den IKE_SA_INIT Meldungen einigen sich die VPN Endpunkte (Initiator und Responder) unter anderem über die Sicherheitsparameter für die IKE_SA und führen einen Diffie-Hellman Key Exchange durch [5]. Dann erfolgt der Austausch der IKE_AUTH Meldungen (Meldungen 3 und 4 in Abbildung 2). Mit den IKE_AUTH Meldungen Authentifizieren sich die beiden VPN Endpunkte gegenseitig und erzeugen die erste CHILD_SA [5]. Um sich zu authentifizieren erzeugt der Initiator eine Signatur (oder einen Shared Key Message Integrity Code) über Teile der ersten (und zweiten) Meldung und sendet diese im $Auth_i$ Feld der Meldung 3 an den Responder. Analog dazu erzeugt der Responder eine Signatur (oder einen Shared Key Message Integrity Code) über Teile der zweiten (und ersten) Meldung und sendet diese im $Auth_r$ Feld der Meldung 4 an den Initiator. Da nur der "echte" Initiator das Geheimnis (z.B. den Pre-Shared Key oder den Private Key) kennt mit

dem sich $Auth_i$ erzeugen lässt, kann der Responder sicher sein, dass er es mit dem gewünschten Initiator zu tun hat, d.h., dass der Initiator derjenige ist, für den er sich ausgibt. Analog kann der Initiator sicher sein, dass der Responder authentisch ist, da nur der "echte" Responder das zum Erstellen von $Auth_r$ notwendige Geheimnis kennt.
Um die in $Auth$ enthaltene Signatur zu überprüfen, benötigen die VPN Endpunkte das Zertifikat des zugehörigen Public Key. Dieses kann im $Cert$ Feld der IKE_AUTH Meldungen übertragen werden oder kann unter einer URL bereitgestellt werden [5]. Befindet sich im $Auth$ Feld statt einer Signatur ein Shared Key Integrity Code so benötigen die Endpunkte zu dessen Überprüfung den Pre-Shared Key [5].

3.2 Endpunktauthentifizierung mit Zertifikaten

Ein Zertifikat lässt sich vereinfacht wie folgt darstellen:

$$Zertifikatsinhalt = Aussteller + Gültigkeitszeitraum + Subjekt + Public\ Key$$

$$Zertifikat = Zertifikatsinhalt + Aussteller + Signatur(Zertifikatsinhalt)$$

Das Zertifikat ordnet einem Subjekt/Entität (z.B. einer Person oder einem VPN Gateway) einen Public Key zu und bestätigt diese Zuordnung durch eine digitale Signatur. Zertifikate werden von CAs (Certificate Autorities) ausgestellt, welche mit ihrem Private Key die Zertifikate signieren.
Um die zertifikatsbasierte Authentifizierung der VPN Endpunkte genauer zu erläutern betrachten wir nochmals die Protokollsequenz aus Abbildung 2. Wie kann der Initiator (Alice) die Identität des Responders (Bob) mithilfe eines Zertifikats überprüfen, d.h. den Responder authentifizieren? Der Responder hat dazu ein Schlüsselpaar bestehend aus einem Public Key und dem zugehörigen Private Key. Mit seinem Private Key signiert der Responder, wie in Kapitel 3.1 beschrieben, Teile der ersten beiden Meldungen und sendet diese Signatur im $Auth_r$ Feld dem Initiator. Der Initiator kann mit dem Public Key des Responders die Signatur entschlüsseln und weiss somit, dass er mit einem Responder kommuniziert, der den zu diesem Publik Key gehörenden Private Key besitzt. Er weiss allerdings nicht, ob dieser Public Key der Public Key von Bob ist, d.h. ob er es tatsächlich mit dem gewünschten Responder zu tun hat. Dazu verwendet der Initiator das zum Public Key gehörende Zertifikat, welches bestätigt wem der Schlüssel gehört. Die Signatur des Zertifikats ermöglicht es dem Initiator das Zertifikat auf seine Echtheit hin zu überprüfen. Dazu verwendet er den Public Key der CA welche das Zertifikat ausgestellt hat. Der Initiator muss dieser CA vertrauen. Er führt dazu eine Liste von Public Keys von vertrauenswürdigen CAs. Alternativ ist es auch möglich, dass der Initiator der CA indirekt über eine sogenannte „Chain of Trust“ vertraut. Der Responder authentifiziert den Initiator auf dieselbe Art und Weise.
Mithilfe von Zertifikaten und Signaturen können sich die VPN Endpunkte also gegenseitig authentifizieren. Da die Zertifikate selbst wiederum authentifiziert werden müssen, bedarf es einer Public Key Infrastructure, welche Zertifikate ausstellt und deren Authentifizierung erlaubt.

3.3 Endpunktauthentifizierung per DNS mit DNSSEC

Wie wir in Kapitel 3.2 gesehen haben, kann man einen VPN Endpunkt mithilfe eines Zertifikats authentifizieren. Das Zertifikat besteht im Wesentlichen aus dem Namen (FQDN oder IP-Adresse) und Public Key des VPN Endpunktes und einer Signatur, welche die Echtheit der im Zertifikat gemachten Angaben bestätigt.
Speichert man im DNS den Namen eines VPN Endpunktes und dessen „Public Key“, so liefert einem DNS mit DNSSEC die gleiche Information wie ein Zertifikat, nämlich:

- Name des VPN Endpunktes
- Public Key des VPN Endpunktes
- Bestätigung, dass diese Daten unverändert und authentisch sind (durch DNSSEC Signaturen)

Anstelle von Zertifikaten sollten sich also auch DNS Einträge zur Authentifizierung von VPN Endpunkten einsetzen lassen. Recherchen haben gezeigt, dass es bereits Bemühungen in diese Richtung gab. So definiert RFC 4025 [4] einen speziellen DNS Ressource Record zur Speicherung des Public Keys eines VPN Endpunktes im DNS.
Da der RFC 4025 im Standardisierungsprozess der IETF bereits weit fortgeschritten ist (aktuell "Proposed Standard" [7]), haben wir uns dazu entschlossen seinen Ansatz zur Authentifizierung von VPN Endpunkten per DNS weiterzuverfolgen. Nach einer kurzen Einführung in DNSSEC folgt unser Konzept zur Authentifizierung von VPN Endpunkten über DNS Einträge.

3.3.1 DNSSEC

Bei DNSSEC handelt es sich um eine Erweiterung des DNS um Sicherheitsfunktionen. Diese Sicherheitsfunktionen gewährleisten [3]:

- Data Origin Authentication
  Der DNS Resolver kann prüfen, ob Resource Records wirklich aus der angegebenen Zone stammen.
- Data Integrity
  Allfällige Manipulationen an den Ressource Records können vom DNS Resolver festgestellt werden.
- Authenticated Denial of Existence
  Eine vertrauenswürdige Aussage über die Existenz oder Nichtexistenz von RR durch den DNS Server ist möglich.

Zur Realisierung dieser Sicherheitsfunktionen definiert DNSSEC vier neue Resource Record Typen [3]:

- RRSIG
- DNSKEY
- DS
- NSEC (und die neuere Variante NSEC3)

Ausserdem erweitert es das DNS Protokoll. Da die RR von DNSSEC für klassische DNS Pakete zu gross sein können, müssen die DNS Server und Resolver für den Einsatz von DNSSEC das EDNS0 Protokoll unterstützen [3].
Um die Authentizität und Integrität der DNS Daten zu gewährleisten greift DNSSEC auf Verfahren der Public Key Kryptographie zurück und setzt zwei Arten von Schlüsselpaaren ein:

- KSK
  Der KSK (Key Signing Key) Private Key wird von DNSSEC zum Signieren des ZSK Public Key eingesetzt. Damit ein Resolver die mit einem KSK ausgestellten Signaturen überprüfen kann, wird der zum KSK gehörende Public Key in einem DNSKEY RR im DNS abgespeichert.
- ZSK
  Der ZSK (Zone Signing Key) Private Key wird zum Signieren der Zone eingesetzt. Mit ihm werden also die RR der Zone signiert. Damit ein Resolver die mit dem ZSK erstellten Signaturen überprüfen kann, wird der zum ZSK gehöhrende Public Key in einem DNSKEY RR im DNS gespeichert.

Nachdem wir nun die wichtigsten Komponenten von DNSSEC eingeführt haben, betrachten wir ihr Zusammenspiel an einem konkreten Beispiel:

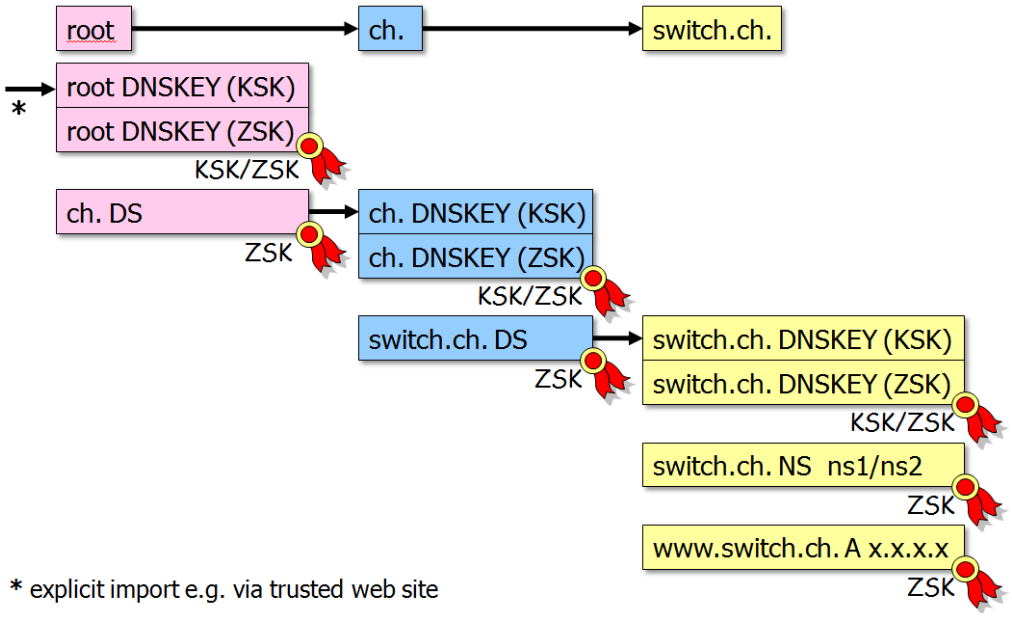


Abbildung 3: Zusammenspiel der DNSSEC Komponenten [6]

In Abbildung 3 ist ein DNSSEC Szenario mit drei Zonen „root“, „ch.“ und „switch.ch.“ dargestellt. Die Zone "switch.ch." ist dabei der Zone "ch." untergeordnet, welche wiederum der „root“ Zone untergeordnet ist. Jede Zone hat einen KSK und ZSK. Der Public Key des KSK einer Zone wird in der Zone selbst in Form eines DNSKEY RR abgelegt. Analog wird der Public Key des ZSK einer Zone in einem DNSKEY RR in der Zone abgelegt. Die zugehörigen Private Keys bleiben geheim und werden sicher (am besten offline) gespeichert. Die RRs einer Zone werden mit dem ZSK Private Key signiert und die resultierenden Signaturen als

RRSIG Resource Records im DNS abgelegt (rote ZSK Schleifen in Abbildung 3). Die DNSKEY RR einer Zone werden mit dem KSK Private Key der Zone signiert (rote KSK/ZSK Schleife in Abbildung 3). In der Zone "ch." ist der KSK Public Key der Zone "switch.ch." in Form eines DS RR gespeichert, welcher wie alle RR der Zone "ch." mit dem ZSK Private Key der "ch." Zone signiert ist.
Um aufzuzeigen, wie ein DNS Resolver einen RR mittels DNSSEC validiert (seine Authentizität und Integrität prüft), gehen wir nun davon aus, dass der Resolver den RR „www.switch.ch. A x.x.x.x“ von einem für die Zone „switch.ch.“ zuständigen DNS Server abfragt und iterativ validiert:

1. Der Resolver fragt beim DNS Server nach dem A RR für den FQDN „www.switch.ch.“.
2. Der DNS Server antwortet dem Resolver mit dem gewünschten RR "www.switch.ch. A x.x.x.x" und dem zugehörigen RRSIG RR mit der ZSK Signatur des abgefragten „www.switch.ch A x.x.x.x“ Resource Records.
3. Der Resolver prüft nun mithilfe des ZSK Public Keys der Zone "switch.ch" (welchen er durch Abfrage der DNSKEY RRs der Zone „switch.ch.“ erhält, sofern er ihn in Schritt 2 nicht schon erhalten hat), ob die Signatur (RRSIG) des RR „www.switch.ch. A x.x.x.x“ gültig ist.
4. Ist die Signatur gültig, so kann der Resolver davon ausgehen, dass der RR den er erhalten hat von einer Zone mit dem zur Validierung der Signatur eingesetzten ZSK stammt.
5. Um sicherzugehen, dass es sich bei dieser Zone tatsächlich um die Zone „switch.ch.“ handelt muss der Resolver nun noch prüfen, ob der verwendete ZSK auch tatsächlich zur Zone „switch.ch.“ gehört. Dazu lädt er als erstes mit einer DNSKEY Abfrage den KSK Public Key der Zone „switch.ch.“ und überprüft mit ihm die Signatur des verwendeten ZSK. Ist diese gültig, so weiss der Resolver, dass der ZSK und damit der mit ihm signierte „www.switch.ch A x.x.x.x“ RR aus einer Zone mit diesem KSK stammt.
6. Um zu prüfen, ob dieser KSK tatsächlich der KSK der Zone „switch.ch.“ ist, lädt der Resolver aus der übergeordneten Zone „ch.“ den DS RR „switch.ch. DS“. Dieser enthält einen SHA-1 und/oder SHA-256 Hash über den KSK Public Key der Zone „switch.ch.“ und der Resolver kann somit prüfen, ob der zur Validierung des ZSK der Zone „switch.ch.“ eingesetzte KSK tatsächlich zur Zone „switch.ch.“ gehört. Ist dies der Fall, so gehören alle mit dem ZSK Signierten RR und damit der abgefragte RR „www.switch.ch. A x.x.x.x“ zur Zone „switch.ch.“.

Wie kann der Resolver nun aber sicher sein, dass der DS RR, den er zur Überprüfung des KSK verwendet hat, tatsächlich aus der Zone „ch.“ stammt und nicht gefälscht ist?
Der DS RR ist dazu in der Zone „ch.“ mit dem ZSK Public Key der Zone „ch.“ signiert und seine Authentizität kann der Resolver somit analog zu den Schritten 2 bis 6 validieren. Der Resolver validiert auf diese Art alle Keys bis zum KSK der „root“ Zone. Der KSK Public Key der „root“ Zone ist dem Resolver bekannt. Er bildet einen sogenannten Trust Anchor, welchem der Resolver vertraut. Gelingt die Validierung aller Schlüssel bis zum KSK der „root“

Zone, so weiss der Resolver, dass der RR „www.switch.ch A x.x.x.x“, welchen er abgefragt hat, tatsachlich aus der Zone „switch.ch.“ stammt und nicht manipuliert wurde. Der Resolver kann sich also der Authentizität und Integrität des RR „www.switch.ch x.x.x.x“ sicher sein.

3.3.2 Konzept

RFC 4025 [4] definiert einen speziellen Resource Record (IPSECKEY RR) zur Speicherung von Public Keys von VPN Endpunkten im DNS. RFC 4025 entstand im Rahmen des FreeS/WAN Projekts dessen Ziel die Implementierung der sogenannten „Opportunistic Encryption“ (siehe RFC 4322 [25]) war. Die Idee hinter der „Opportunistic Encryption“ war, die gesamte Kommunikation im Internet per IPsec zu verschlüsseln und die dafür notwendigen Public Keys im DNS abzulegen. Die „Opportunistic Encryption“ konnte sich im Rahmen des FreeS/WAN Projekts allerdings nicht durchsetzen und FreeS/WAN wurde eingestellt [26]. Das hier vorgeschlagene Verfahren zur Authentifizierung der VPN Endpunkte ist somit nicht grundsätzlich neu, sondern entstammt den Ideen, die im Zusammenhang mit der „Opportunistic Encryption“ entwickelt wurden. Ziel des hier vorgestellten Verfahrens ist nicht die „Opportunistic Encryption“, sondern lediglich die Einführung einer neuen auf dem DNS mit DNSSEC basierenden Authentifizierungsmethode für VPN Endpunkte. Das vorgestellte Verfahren stellt aber einen Grundstein der „Opportunistic Encryption“ dar und liesse sich daher später allenfalls noch zu ihr ausbauen.
Nun kommen wir zum „neuen“ Authentifizierungsverfahren. Verwendet man DNS mit DNSSEC, so kann man die in [4] definierten IPSECKEY RR zur Authentifizierung von VPN Endpunkten einsetzen und zwar wie folgt:

Für jeden VPN Endpunkt speichert man unter seinem FQDN im DNS seinen Public Key in einem IPSECKEY RR.
Will der VPN Endpunkt Alice nun mit dem VPN Endpunkt Bob ein VPN aufbauen, so gehen die beiden gemäss dem IKEv2 Protokoll wie in Kapitel 3.1 und Abbildung 2 dargestellt vor. Sie tauschen gegenseitig Meldungen aus, die sie mit ihrem jeweiligen Private Key signiert haben. Um diese Signaturen zu verifizieren und sich damit gegenseitig zu authentifizieren greifen sie nun aber nicht auf Zertifikate, sondern auf die IPSECKEY RR aus dem DNS zurück. Dazu ruft Alice den IPSECKEY RR von Bob aus dem DNS ab, welcher unter dem FQDN von Bob im DNS abgelegt ist. Alice prüft dann mithilfe von DNSSEC, ob der abgerufene IPSECKEY RR authentisch und integer ist, d.h. vertrauenswürdig ist. Ist der IPSECKEY RR vertrauenswürdig, so verwendet Alice den in ihm enthaltenen Public Key um die von Bob stammende Signatur zu überprüfen. Ist diese Überprüfung erfolgreich, so weiss Alice, dass sie mit dem „echten“ Bob kommuniziert, da nur dieser Signaturen ausstellen kann die zum im IPSECKEY RR enthaltenen Schlüssel passen (durch DNSSEC kann Alice sicher sein, dass dieser Schlüssel auch tatsächlich Bob gehört, da der IPSECKEY RR mit DNSSEC die gleiche Funktionalität wie ein Zertifikat erfüllt). Alice kann also mithilfe der von Bob signierten Meldungen, Bobs IPSECKEY RR und DNSSEC Bob authentifizieren. Analog kann Bob Alice authentifizieren.
Das eben vorgestellte Verfahren ermöglicht es dem IKEv2 Protokoll also VPN Endpunkte mittels IPSECKEY RR zu authentifizieren.

Ein IPSECKEY RR ist wie folgt aufgebaut:

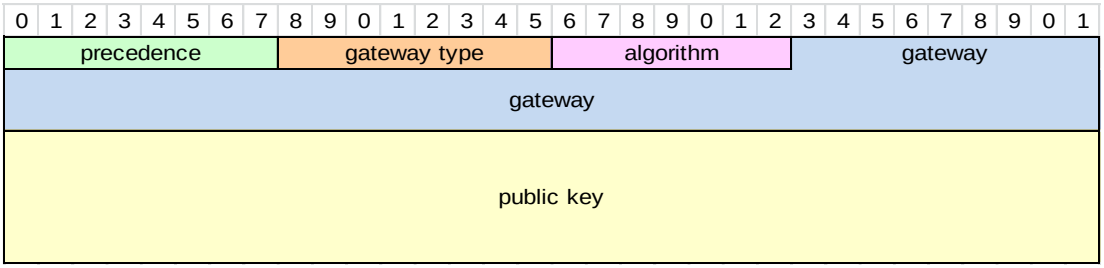


Abbildung 4: Aufbau eines IPSECKEY Resource Record

Die einzelnen Felder haben dabei folgende Bedeutung [4]:

- precedence
  Die „precedence“ gibt die Priorität an, mit der der IPSECKEY RR behandelt werden soll. Gibt es mehrere IPSECKEY Resource Records für einen FQDN, so soll der Gateway des IPSECKEY RR mit der niedrigsten „precedence“ als erster verwendet werden.
- gateway type
  Das Feld „gateway type“ gibt das Format der im „gateway“ Feld enthaltenen Information an.
- algorithm
  Das „algorithm“ Feld gibt an auf welchem kryptographischen Verfahren der im Feld „public key“ enthaltene Public Key basiert, und in welchem Format er gespeichert ist.
- gateway
  Im „gateway“ Feld ist die IP Adresse des IPsec VPN Gateways gespeichert zu welchem der ebenfalls im IPSECKEY RR gespeicherte „public key“ gehört.
- public key
  Das „public key“ Feld enthält den eigentlichen Public Key im in „algorithm” angegebenen Format.

Wie man anhand der Felder des IPSECKEY RR erkennen kann, ist es möglich zu einem FQDN mehrere IPSECKEY RR und damit mehrere IPsec Gateways (VPN Endpunkte) anzugeben. Diese können sich in der IP Adresse, dem für den Public Key verwendeten kryptographischen Verfahren und dem Public Key selbst voneinander unterscheiden.

4 Proof of Concept

Um sicher zu gehen, dass die in Kapitel 3.3 vorgeschlagene Methode zur Authentifizierung von VPN Endpunkten tatsächlich realisierbar ist, haben wir sie auf ihre Machbarkeit hin untersucht. Dabei ging es uns primär darum zu prüfen, inwiefern die für die Realisierung des Konzepts notwendigen Technologien (DNS mit DNSSEC, IPSECKEY RR etc.) bereits implementiert wurden und zusammenarbeiten und wo allfällige Probleme auftauchen könnten. Wir untersuchten dabei Primär drei Dinge:

1. Wird der in RFC 4025 [4] definierte IPSECKEY RR vom aktuellen BIND DNS Server unterstützt?
2. Lassen sich Resource Records aus dem BIND DNS Server abfragen und mittels DNSSEC validieren, d.h. funktioniert die DNSSEC Implementierung in BIND wie gewünscht?
3. Welche DNS Resolver Bibliotheken würde sich am besten für die Implementierung der IPSECKEY basierten Authentifizierung in strongSwan eignen?

Für den Proof of Concept wurde die 64 Bit Version von Ubuntu Linux 11.04 eingesetzt. Alle verwendeten Programmpakete stammen, sofern nicht anders vermerkt, aus dieser Distribution.

4.1 IPSECKEY Unterstützung durch BIND

Um zu prüfen, ob die in RFC 4025 [4] definierten IPSECKEY Resource Records vom BIND DNS Server [8] unterstützt werden, sind wir wie folgt vorgegangen:

Als erstes wurde BIND 9.7.3 installiert:

```
sudo apt-get install bind9
```

Zum Ausführen von DNS Abfragen aus der Shell heraus werden die „dnsutils“ benötigt welche wie folgt installiert wurden:

```
sudo apt-get install dnsutils
```

Nach der Installation von BIND wird der BIND Name Server Daemon “named” automatisch gestartet.

Mit dem Befehl

```
dig @127.0.0.1 localhost.
```

kann man die IP-Adresse des Rechners mit dem Namen „localhost.“ vom eben lokal installierten BIND DNS Server abfragen, und somit testen, ob der Nameserver korrekt installiert und gestartet wurde.

Die Konfigurationsdateien von BIND liegen unter „/etc/bind“. In der Datei „named.conf“ befinden sich die globalen Einstellungen von BIND. Hier sind auch die Zonefiles, in welchen BIND nach den Informationen über die von ihm verwalteten Namensräume sucht, vermerkt. Die eigentlichen Zonendaten befinden sich dann in den Zonefiles „db.local“, „db.root“ etc.

Im Zonefile „db.local“ legten wir einen IPSECKEY RR an. Dazu öffneten wir es per

```
sudo gedit /etc/bin/db.local
```

und erweiterten es wie folgt (Adaption eines Beispiels aus [4]):

```
; BIND data file for local loopback interface
;
$TTL  604800
@     IN    SOA   localhost. root.localhost. (
                        8     ; Serial
                   604800     ; Refresh
                    86400     ; Retry
                  2419200     ; Expire
                   604800 )   ; Negative Cache TTL
;
@     IN    NS    localhost.
@     IN    A     127.0.0.1
@     IN    AAAA  ::1
;
; Eintrag eines IPSECKEY für localhost
;
@     7200  IN    IPSECKEY ( 10       ; Precedence
                             1          ; Gateway Type
                             2          ; Algorithm Type
                             127.0.0.1  ; Gateway

AQNRU3mG7TVTO2BkR47usntb102uFJtugbo6BSGvgqt4AQ== ) ; Public Key
```

Vor dem Speichern des Zonefiles haben wir den Eintrag „Serial“ im SOA RR inkrementiert, damit BIND die Änderungen am Zonefile erkennen kann.
Nach dem Speichern der Datei haben wir „named“ mit dem Befehl

```
sudo rndc -s127.0.0.1 reload
```

angewiesen seine Konfigurationsdateien und die Zonefiles neu zu laden. Auf dem lokalen DNS Server befindet sich nun für die Domain „localhost.“ ein IPSECKEY RR, welchen man mit dem Programm „dig“ abfragen kann:

```
dig @127.0.0.1 localhost IPSECKEY
```

liefert:

```
; <<>> DiG 9.7.3 <<>> @127.0.0.1 localhost IPSECKEY
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 18023
;;  flags:  qr  aa  rd  ra;  QUERY:  1,  ANSWER:  1,  AUTHORITY:  1,
ADDITIONAL: 2

;; QUESTION SECTION:
;localhost.             IN    IPSECKEY

;; ANSWER SECTION:
localhost.        7200  IN    IPSECKEY    10    1    2    127.0.0.1
AQNRU3mG7TVTO2BkR47usntb102uFJtugbo6BSGvgqt4AQ==

;; AUTHORITY SECTION:
localhost.        604800      IN    NS    localhost.

;; ADDITIONAL SECTION:
localhost.        604800      IN    A     127.0.0.1
localhost.        604800      IN    AAAA  ::1

;; Query time: 0 msec
```

```
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Thu Oct  6 15:25:32 2011
;; MSG SIZE  rcvd: 138
```

4.1.1 Schlussfolgerungen

BIND 9.7.3 (und dig) unterstützt also IPSECKEY Resource Records. Neben BIND gibt es noch NSD als zweite freie Implementierung eines DNS Servers. Gemäss dem Changelog von NSD [16] unterstützt NSD seit Version 3.0.0 IPSECKEY RR. Die IPSECKEY RR gemäss RFC 4025 werden also von den wichtigsten freien DNS Server Implementierungen unterstützt.

4.2 DNSSEC in BIND

Um die DNSSEC Funktionalität von BIND zu testen, haben wir folgendes Szenario in BIND konfiguriert:

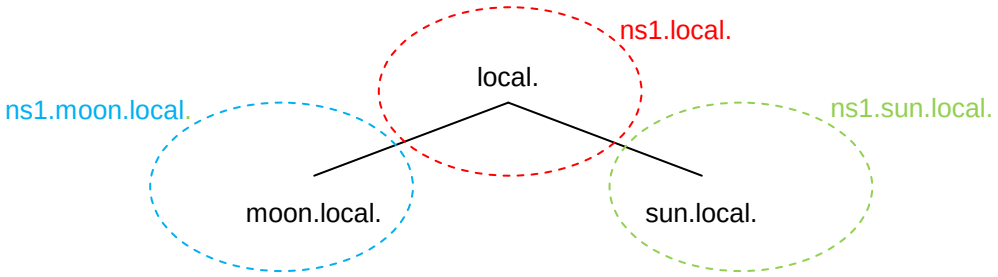


Abbildung 5: DNSSEC Testszenario für BIND

Wir definieren wie in Abbildung 5 dargestellt drei Zonen „local.“, „moon.local.“ und „sun.local.“ und schützen ihren Inhalt mittels DNSSEC. Für jede der drei Zonen ist ein DNS Server „ns1“ zuständig. Den KSK der Zone „local.“ verwenden wir als Trust Anchor für die Validierung der in unseren Zonen enthaltenen RR mittels DNSSEC. Das Szenario orientiert sich an den Testfällen für strongSwan, so, dass es später allenfalls zum Aufbau von Testfällen für die Endpunktauthentifizierung mittels DNS und DNSSEC wiederverwendet werden könnte.

4.2.1 DNSSEC Testszenario anlegen

Wir gehen an dieser Stelle davon aus, dass die in Kapitel 4.1 an der BIND Konfiguration vorgenommenen Änderungen rückgängig gemacht wurden.

Beim Aufbau des BIND DNSSEC Testszenarios sind wir wie folgt vorgegangen (näheres zur Konfiguration von DNSSEC findet man in [9] und [10]):

Als erstes haben wir Verzeichnisse für unsere Zonen angelegt:

```
sudo mkdir /etc/bind/strongSwanZones
sudo mkdir /etc/bind/strongSwanZones/local
sudo mkdir /etc/bind/strongSwanZones/moon.local
sudo mkdir /etc/bind/strongSwanZones/sun.local
```

Dann haben wir das Zonefile "db.moon.local" durch

```
sudo gedit /etc/bind/strongSwanZones/moon.local/db.moon.local
```

angelegt und mit folgendem Inhalt versehen:

```
; Zonefile for the moon zone
;
;
$TTL  604800
@     IN    SOA   ns1.moon.local. root.moon.local. (
                                1     ;Serial
                                604800      ;Refresh
                                86400 ;Retry
                                2419200     ;Expire
                                604800)     ;Negative Cache TTL
;
;
; Name servers of this zone
@     IN    NS    ns1.moon.local.
ns1   IN    A     127.0.0.1
ns1   IN    AAAA  ::1
;
;
; Entries for the VPN Gateway "gateway.moon.local"
gateway           IN    A     192.168.0.1
```

Um DNSSEC zu verwenden muss man die RR in den Zonefiles signieren. Dazu muss man als erstes die KSK und ZSK Schlüssel erzeugen, welche man zum Signieren verwenden will. Damit sich diese Schlüssel schnell Erzeugen lassen, muss auf dem Rechner genug Entropie vorhanden sein. Da dies auf dem von uns verwendeten Rechner nicht der Fall war, haben wir das Softwarepaket "randomsound" installiert, welches mithilfe der Soundkarte für genügend Entropie sorgt.
Dann haben wir mit dem Befehl "dnssec-keygen" die gewünschten Schlüssel für die Zone „moon.local.“ angelegt:

```
# Create KSK and ZSK for zone moon.local.
cd /etc/bind/strongSwanZones/moon.local
dnssec-keygen -a RSASHA1 -b 1024 -n ZONE moon.local.
dnssec-keygen -a RSASHA1 -b 2048 -n ZONE -f KSK moon.local.
```

Durch die obigen Befehle resultieren zwei *.key Dateien mit den Public Keys des ZSK / KSK und zwei *.private Dateien mit den zugehörigen Private Keys. Für die Public Keys muss man im Zonefile der Zone „moon.local.“ DNSKEY RRs anlegen, damit ein DNS Resolver die Public Keys bei Bedarf abrufen kann und mit ihnen RRSIG Signaturen, die aus der Zone stammen, überprüfen kann. Die *.key Dateien enthalten den passenden DNSKEY RR Eintrag für das Zonefile. Der Inhalt der beiden *.key Dateien wurde daher mit

```
sudo bash -c 'cat K.+005+xxxxx.key >> db.moon.local'
sudo bash -c 'cat K.+005+yyyyy.key >> db.moon.local'
```

an das Ende des Zonefiles "db.moon.local" angehängt.

Nun enthält das Zonefile der Zone „moon.local.“ die benötigten RR. Es wurde schliesslich wie folgt signiert:

```
cd /etc/bind/strongSwanZones/moon.local
sudo dnssec-signzone -o moon.local. -N increment db.moon.local
```

Dieser Befehl signiert die Zone und erzeugt das Zonefile “db.moon.local.signed” welches die signierte Version der Zone enthält. Die Keys, welche er zum Signieren benötigt, bestimmt er anhand der DNSKEY RR im zu signierenden Zonefile. Liegen die *.private Dateien mit den benötigten Schlüsseln im aktuellen Verzeichnis, so lädt er aus ihnen automatisch die fürs Signieren der Zone benötigten Private Keys und signiert mit ihnen die Zone. Neben dem signierten Zonefile legt der obige Befehl auch noch eine Datei „dsset-moon.local.“ an, welche die DS RRs für die eben signierte Zone enthält. Diese DS RRs muss man (wie wir später noch sehen werden) ins Zonefile der übergeordneten Zone einfügen, um die „chain of trust“ aufzubauen.

Nun haben wir die erste Zone angelegt und signiert. Bei den verbleibenden Zonen sind wir ähnlich vorgegangen.

Für die Zone „sun.local.“ haben wir das Zonefile "db.sun.local" mittels

```
sudo gedit /etc/bind/strongSwanZones/sun.local/db.sun.local
```

erstellt und mit folgenden RR versehen:

```
; Zonefile for the sun zone
;
;
$TTL  604800
@     IN    SOA   ns1.sun.local. root.sun.local. (
                              1     ;Serial
                              604800      ;Refresh
                              86400 ;Retry
                              2419200     ;Expire
                              604800)     ;Negative Cache TTL
;
;
; Name servers of this zone
@     IN    NS    ns1.sun.local.
ns1   IN    A     127.0.0.1
ns1   IN    AAAA  ::1
;
;
; Entries for the VPN Gateway "gateway.sun.local"
gateway           IN    A     192.168.0.1
```

Für diese Zone haben wir wiederum einen KSK und ZSK erstellt:

```
# Create KSK and ZSK for zone sun.local.
cd /etc/bind/strongSwanZones/sun.local
dnssec-keygen -a RSASHA1 -b 1024 -n ZONE sun.local.
dnssec-keygen -a RSASHA1 -b 2048 -n ZONE -f KSK sun.local.
```

die entsprechenden DNSKEY RR zum Zonefile hinzugefügt:

```
sudo bash -c 'cat K.+005+xxxxx.key >> db.sun.local'
sudo bash -c 'cat K.+005+yyyyy.key >> db.sun.local'
```

und schliesslich die Zone signiert:

```
cd /etc/bind/strongSwanZones/sun.local
sudo dnssec-signzone -o sun.local. -N increment db.sun.local
```

Das Vorgehen für die noch fehlende Zone „local.“ unterscheidet sich leicht vom bisherigen Vorgehen. Da diese Zone den beiden bereits erstellten Zonen übergeordnet ist, muss man sie mit den DS RR der ihr untergeordneten Zonen versehen. Diese DS RRs findet man in den Dateien „dsset-yyy.“ welche beim Signieren der Zonen „moon.local.“ und „sun.local.“ erzeugt wurden. Den Inhalt dieser Dateien muss man in das Zonefile der Zone „local.“ kopieren, bevor man die Zone „local.“ signiert.

Für die Zone „local.“ haben wir das Zonefile "db.local" mittels

```
sudo gedit /etc/bind/strongSwanZones/local/db.local
```

angelegt und mit folgenden RR versehen:

```
; Zonefile for the local zone
;
;
$TTL  604800
@     IN    SOA   ns1.local. root.local. (
```

```
                                1     ;Serial
                                604800      ;Refresh
                                86400 ;Retry
                                2419200     ;Expire
                                604800)     ;Negative Cache TTL
;
;
; Name servers of this zone
@     IN    NS    ns1.local.
ns1   IN    A     127.0.0.1
ns1   IN    AAAA  ::1
;
;
; Delegation of subdomain "moon"
moon  IN    NS    ns1.moon.local.
ns1.moon IN A     127.0.0.1
ns1.moon IN AAAA  ::1
;
;
; Delegation Signer RR for the subdomain "moon"
moon.local. IN DS 17476 5 1 F56C3074E8BC17CC74454F215A825A745B812D38
moon.local.          IN          DS          17476          5          2
336F3A564A75F7663616F5DC3ECA7D4406F73C899CAF3265032FE1F6 7F3225A9
;
;
; Delegation of subdomain "sun"
sun   IN    NS    ns1.sun.local.
ns1.sun     IN    A     127.0.0.1
ns1.sun     IN    AAAA  ::1
```

Die DS RRs stammen wie erwähnt aus den „dsset-yyy.“ Dateien.

Dann haben wir den KSK und ZSK für die Zone erzeugt:

```
# Create KSK and ZSK for zone local.
cd /etc/bind/strongSwanZones/local
dnssec-keygen -a RSASHA1 -b 1024 -n ZONE local.
dnssec-keygen -a RSASHA1 -b 2048 -n ZONE -f KSK local.
```

die entsprechenden DNSKEY RR zum Zonefile hinzugefügt:

```
sudo bash -c 'cat K.+005+xxxxx.key >> db.local'
sudo bash -c 'cat K.+005+yyyyy.key >> db.local'
```

und schliesslich die Zone signiert:

```
cd /etc/bind/strongSwanZones/local
sudo dnssec-signzone -o local. -N increment db.local
```

Nun haben wir alle Zonefiles für unser Testszenario erstellt und signiert. Die signierten Zonefiles müssen nun noch BIND bekannt gemacht werden. Dazu wurde die Datei "zones.strongSwan" durch

```
sudo mkdir /etc/bind/zones.strongSwan
```

angelegt und mit folgenden Inhalt versehen:

```
// Zones for the strongSwan UML tests
zone "local." {
      type master;
      file "/etc/bind/strongSwanZones/local/db.local.signed";
};

zone "moon.local." {
      type master;
      file
```

```
"/etc/bind/strongSwanZones/moon.local/db.moon.local.signed";
};

zone "sun.local." {
      type master;
      file
"/etc/bind/strongSwanZones/sun.local/db.sun.local.signed";
};
```

Diese Datei wurde schliesslich mit folgendem Eintrag in die Datei "/etc/bind/named.conf.local" eingebunden:

```
include "/etc/bind/zones.strongSwan";
```

Somit haben wir sämtliche Zonen für unser DNSSEC Testszenario angelegt, signiert, die „chain of trust“ aufgebaut und die Zonen BIND bekannt gemacht. Da unsere Zonen nicht teil der öffentlichen DNS Hierarchie sind müssen wir den KSK unserer Zone „local.“ noch als „Trusted Anchor“ definieren. Ausserdem müssen wir noch den DNSSEC Support in BIND aktivieren. Dazu editierten wir die Datei „/etc/named.conf.options“:

```
sudo gedit /etc/named.conf.options
```

und haben sie mit folgendem Inhalt versehen:

```
options {
      directory "/var/cache/bind";

      // If there is a firewall between you and nameservers you want
      // to talk to, you may need to fix the firewall to allow
      // multiple ports to talk.
      // See http://www.kb.cert.org/vuls/id/800113

      // If your ISP provided one or more IP addresses for stable
      // nameservers, you probably want to use them as forwarders.
      // Uncomment the following block, and insert the addresses
      // replacing the all-0's placeholder.

      // forwarders {
      //    0.0.0.0;
      // };

      auth-nxdomain no;    # conform to RFC1035
      listen-on-v6 { any; };

      // Turn DNSSEC on
      dnssec-enable yes;
      dnssec-validation yes;
};
trusted-keys {
"local."                  257                     3                   5
"AwEAAcH3CCtFrKwr569N+37R1rXzCJ5QFMW2+x+JhdeFUj/cgv2i8gMH
xwlwsR2VcWnCCTLGXw4aDtjndzhiIBJ+TkBRST8roaxDJ0dviWhJVu/x
2JIvmwN92zpYxJUoCBEKo0zdneOWsYStfDV3IKpLt+TKMC4HoGyKsZ4J
gKCNZwXjOYLf5wsQWUrUJte5sN3eJzB4dEuqEKwf8akBojcc7F5+7X2J
H6r7p94sa7E3Phn1oNoamwdZHkdQaK58fVnC6pXC6eTYVsJZGnLfiJ+h
IQwW3uTN7RGADgYP8IeMa7m6VetmCXcSHfqoP4xtanDOzOLf1EX8fwmr
Xk4jPIQ7zck=";
};
```

Damit BIND mit den erstellten Zonen arbeitet, müssen wir es nun noch anweisen seine Zonefiles neu zu laden. Dies haben wir mit dem Befehl

```
sudo rndc -s127.0.0.1 reload
```

getan.

Damit war die Konfiguration unseres DNSSEC Testszenarios abgeschlossen und wir prüften noch mithilfe des Programmes „dig“, ob es sich wie gewünscht verhält.
„dig“ kann signierte DNS Einträge iterativ validieren. Dazu muss man „dig“ mitteilen, welche Schlüssel es als „Trusted Anchor“ verwenden soll. Um zu prüfen, ob die Zonen korrekt erzeugt und DNSSEC korrekt eingerichtet wurde, sind wir wie folgt vorgegangen:

Als erstes haben wir mit dem Befehl

```
dig @127.0.0.1 +nocomments +nostats +nocmd +noquestion -t DNSKEY
local. > trustedkey.key
```

den ZSK/KSK unserer Zone „local.“ in einer Datei gespeichert um ihn als „Trusted Anchor“ für unsere Abfrage zu verwenden. Dann haben wir mit

```
dig @127.0.0.1 +topdown +sigchase +multiline +trusted-key=./trusted-
key.key -t A gateway.moon.local.
```

den A RR für die Domain „gateyway.moon.local“ abgefragt.

„dig“ liefert den A Resource Record zur Domain „gateway.moon.local.“ und alle Schlüssel der „chain of trust“, welche es verwendet hat um die Signatur des A Resource Records zu validieren:

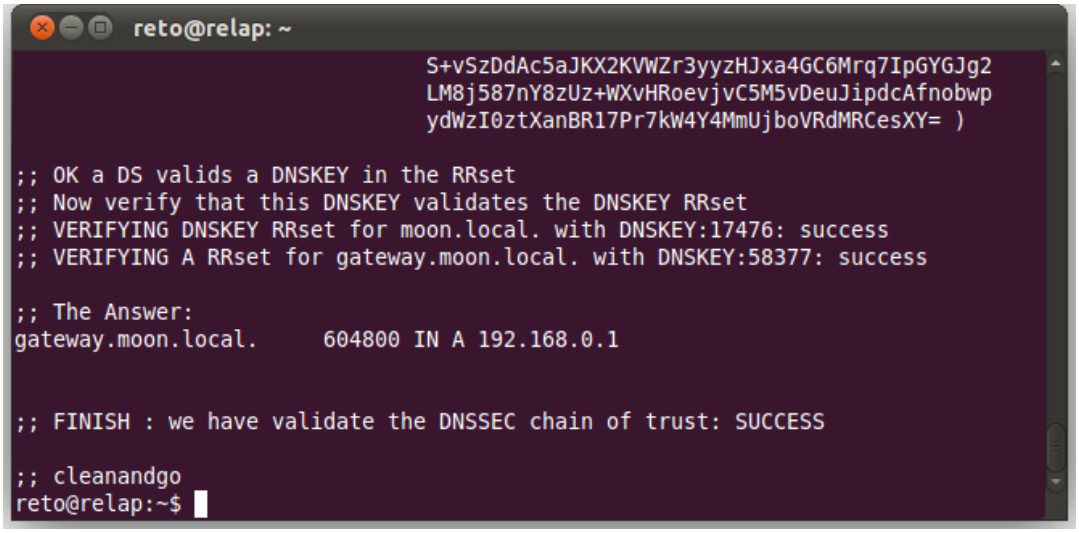


Abbildung 6: Erfolgreiche Validierung eines DNS RR durch „dig“ mittels DNSSEC

Aus Abbildung 6 geht hervor, dass „dig“ den abgerufenen A RR erfolgreich validieren konnte, d.h. dass der abgerufene RR authentisch im Sinne der von DNSSEC gewährleisteten „Data Integrity“ und „Data Origin Authentication“ ist.

4.2.2 Schlussfolgerungen

Mit BIND lässt sich ein lokaler DNS Dienst mit eigenen Namensräumen aufbauen. BIND unterstützt DNSSEC und Tools wie „dig“ sind in der Lage DNS RR abzufragen und mittels DNSSEC zu validieren.

4.3 Evaluation der DNS Resolver Libraries

Unser Ziel ist es das in Kapitel 3.3 vorgeschlagene Authentifizierungsverfahren in strongSwan zu implementieren. Damit strongSwan die IPSECKEY RR gemäss [4] verwenden kann, muss es dazu in der Lage sein sie abzurufen und zu validieren. StrongSwan benötigt also einen DNS Resolver. Bei unseren Recherchen sind wir auf eine Reihe von freien DNS Resolver Bibliotheken für Linux gestossen, die in der Programmiersprache C implementiert sind:

- UDNS [17]
- adns [18]
- c-ares [19]
- libsres [20]
- ldns [11]
- LibUnbound [21]
- libresolv (glibc) [13]

Diese haben wir mit dem Ziel untersucht, herauszufinden, welche für eine allfällige Implementierung des DNS basierten Authentifizierungsverfahren in strongSwan am besten geeignet wäre. Folgende Kriterien erachteten wir dabei als essentiell, d.h. muss eine DNS Resolver Bibliothek aus unserer Sicht zwingend erfüllen:

- Aktive Weiterentwicklung (keine „toten“ Projekte)
- Keine kleinen Ein-Mann Projekte
- Bibliothek ist als Paket in Ubuntu vorhanden

4.3.1 UDNS

Bei UDNS handelt es sich um eine sogenannte „Stub Resolver Library“, welche synchrone und asynchrone DNS Abfragen erlaubt [17]. Da es sich bei der UDNS Bibliothek lediglich um ein kleines Ein-Mann Projekt handelt, haben sie nicht weiter untersucht.

4.3.2 adns

Das adns Projekt scheint tot zu sein (letztes Update 2006) und wir haben die adns Bibliothek daher nicht näher untersucht [18].

4.3.3 c-ares

Bei c-ares [19] handelt es sich um eine asynchrone DNS Resolver Bibliothek. Alle DNS Abfragen werden von der Bibliothek asynchron ausgeführt und sobald sie beendet wurden, wird eine Callback Funktion aufgerufen. Die Bibliothek weist eine klassische prozedurale Architektur auf. Sie beschränkt sich auf die wesentlichen Funktionen eines DNS Resolvers und bietet beispielsweise keine spezielle DNSSEC Unterstützung. Die Dokumentation liegt in Form von manpages vor. Ubuntu enthält ein Paket mit der c-ares Bibliothek.

4.3.4 libsres

Die libsres ist Teil des DNSSEC-Tools Paket das grösstenteils von der Firma SPARTA mit der Förderung des „U.S. Department of Homeland Security/Science & Technology (S&T)“ entwickelt wurde [20]. Ubuntu enthält das DNSSEC-Tools Paket mit der libsres. Dokumentiert ist die libsres durch eine manpage. Der Funktionsumfang der libsres ist sehr spartanisch, so definiert die ganze Bibliothek laut der manpage lediglich acht Funktionen. Abgesehen von einer „print“ Funktion sind keine weiteren Funktionen zum Verarbeiten der Antworten auf DNS Abfragen vorhanden.

4.3.5 ldns

„ldns“ ist eine freie DNS Resolver Bibliothek, welche von den niederländischen LNet Labs entwickelt wird [11]. Bei den LNet Labs handelt es sich um ein Forschungs- und Entwicklungslabor, das unter anderem auch für die Entwicklung des NSD Nameservers verantwortlich ist, der auf 3 der insgesamt 13 root DNS Servern eingesetzt wird [12]. Ubuntu enthält bereits Pakete mit ldns. Das Developer ldns Paket lässt sich wie folgt installieren:

```
sudo apt-get install libldns-dev
```

Es existieren bereits eine Reihe von Programmen, welche auf die ldns Bibliothek zurückgreifen. So setzt beispielsweise das Programm „drill“, welches das Absetzen von DNS Abfragen von der Shell aus erlaubt, auf ldns auf. „drill“ und eine Reihe von weiteren Programmen, die auf ldns basieren, sind im Paket „ldnsutils“ enthalten. ldns wurde wie strongSwan mittels Doxygen dokumentiert. Es ist unter anderem dazu in der Lage DNS RR mittels DNSSEC zu validieren.
Da die ldns Bibliothek einen vielversprechenden Eindruck macht, haben wir, um sie näher kennen zu lernen, ein kleines Programm geschrieben, das mit ihrer Hilfe den A RR der Domain „www.google.ch“ abfragt und ausgibt:

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

#include <ldns/ldns.h>

int main(int argc, char *argv[])
{

	/* URL to query */
	char *dname = "www.google.ch";

	ldns rdf *domain = NULL;
	ldns resolver *resolver = NULL;
	ldns pkt *paket = NULL;
	ldns rr list *rr list = NULL;
	ldns status status;

	/* create a rdf for the domain which we want to query */
	domain = ldns dname new frm str(dname);
	if (!domain) {
		printf("Could not create rdf for domainname %s", dname);
		exit(EXIT FAILURE);
	}
```

```
	/* create a new resolver which queries the name servers
	 * mentioned in /etc/resolv.conf
	 */
	status = ldns resolver new frm file(&resolver, NULL);

	/* check if the resolver was created successfully */
	if (status != LDNS STATUS OK) {
		printf("Could not create resolver\n");
		ldns rdf deep free(domain);
		ldns resolver deep free(resolver);
		exit(EXIT FAILURE);
	}

	/* disable recursive DNS querying */
	/* ldns resolver set recursive(resolver, 0); */

	/* perform the query using the resolver */
	paket = ldns resolver query(resolver,
				domain,
				LDNS RR TYPE A,
				LDNS RR CLASS IN,
				LDNS QR);

	/* handle the results of the query */
	if (!paket)  {
		printf("Query failed: Did not receive answer paket.\n");
		ldns rdf deep free(domain);
		ldns resolver deep free(resolver);
		exit(EXIT FAILURE);
	} else {
		/* read the A records from the answer section of the
answer packet */
		rr list = ldns pkt rr list by type(paket,
						LDNS RR TYPE A,
						LDNS SECTION ANSWER);

		/* check if the query was successful */
		if (!rr list) {
			printf("invalid answer to the query!\n");
			ldns rdf deep free(domain);
			ldns resolver deep free(resolver);
			ldns pkt free(paket);
			exit(EXIT FAILURE);
		} else {
			ldns rr list sort(rr list);
			ldns rr list print(stdout, rr list);
		}
	}

	/* clean up */
	ldns rdf deep free(domain);
	ldns resolver deep free(resolver);
	ldns pkt free(paket);
	ldns rr list deep free(rr list);

	exit(EXIT SUCCESS);
}
```

Listing 1: Beispiel zur Nutzung der ldns Bibliothek

Bei der Implementierung unseres kleinen Beispielprogrammes stach uns vor allem die gute Dokumentation und moderne Architektur der ldns Bibliothek ins Auge. So definiert ldns wie man in Listing 1 sehen kann beispielsweise Datentypen für DNS Pakete („ldns_pkt“), für RR Listen („ldns_rr_list“) und für den Resolver („ldns_resolver“) und Funktionen zur Bearbeitung von Instanzen dieser Datentypen. Dies erinnert an ein objektorientiertes Design, auch wenn ldns rein prozedural in C geschrieben ist.

4.3.6 LibUnbound

Die LibUnbound stammt wie die ldns Bibliothek auch von den niederländischen LNet Labs und ist Teil des Unbound genannten DNS Servers, der ebenfalls von den LNet Labs entwickelt wird [21]. Ubuntu enthält ein Paket mit der LibUnbound. Wie aus dem Tutorial [21] zur LibUnbound hervorgeht, muss man die ldns Bibliothek ebenfalls installieren, wenn man die LibUnbound verwenden will. Die LibUnbound scheint also zumindest teilweise auf die ldns Bibliothek zurückzugreifen. Da die ldns Bibliothek schon die von uns gewünschten Funktionen bereitstellt, macht es keinen Sinn eine Bibliothek wie die LibUnbound zu verwenden die auf ldns aufsetzt.

4.3.7 libresolv (glibc)

Bei der libresolv handelt es sich um eine weitere DNS Resolver Bibliothek. Sie ist Bestandteil der glibc und basiert auf Code aus BIND. Version 2.2 der glibc enthält eine libresolv, die auf BIND 8 basiert (gemäss „resolv/README“ aus [13]). Leider ist die Dokumentation zur libresolv eher dürftig gehalten. Im offiziellen Manual der glibc [14] wird die libresolv beispielsweise nicht erwähnt. Die einzigen Quellen, die wir finden konnten, beschränken sich auf eine manpage („man 3 resolver“) und einen Auszug aus dem Buch „DNS and BIND“ von Cricket Liu und Paul Albitz der auf der Webseite [15] verfügbar ist. Da die libresolv fester Bestandteil der glibc ist, stellt sie trotzdem einen interessanten Kandidaten für den Einsatz in strongSwan dar. Um die Funktionalität der libresolv besser einschätzen zu können, haben wir daher ein kleines Beispielprogramm verfasst, dass mit ihrer Hilfe die A RRs der Domain „www.google.ch“ abfragt und ausgibt:

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

#include <errno.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/nameser.h>
#include <resolv.h>

#define PKTSIZE 4096

int main(int argc, char *argv[])
{
      unsigned char msg[PKTSIZE];
      ns msg handle;
      ns rr rr;
      int msglen;
      int rrnum;
      char * dn = "www.google.ch";
      res init();

      /* Perform a DNS query for RRs of type A and store the answer
packet
         in msg */
      printf("Performing a query for the A RR of the domain %s\n",
dn);
      msglen = res search(dn, ns c in, ns t a, msg, sizeof(msg));

      /* Try to parse the answer packet */
```

```
      if (msglen >= 0){
            printf("Answer received.\n");

            printf("Parsing answer...\n");
            if (ns initparse(msg, msglen, &handle) < 0) {
            printf("Prasing     failed:     ns initparse:     %s\n",
strerror(errno));
            exit(EXIT FAILURE);
            }

            /* Extract all RR from the answer section of the packet
               and print their name */
            printf("Names of all RR in the answer:\n");

            for   (rrnum   =   0;   rrnum   <   ns msg count(handle,
ns s an);rrnum++)
            {
                  ns parserr(&handle, ns s an, rrnum, &rr);
                  printf("Name of RR number %i: ", rrnum);
                  printf(ns rr name(rr));
                  printf("\n");
            }
      }

      exit(EXIT SUCCESS);
}
```

Listing 2: Beispiel zur Nutzung der libresolv Bibliothek

Aufgrund der spärlichen Dokumentation benötigten wir relativ viel Zeit zur Implementierung unseres kleinen Beispielprogrammes aus Listing 2. Ausserdem basiert die libresolv auf dem veralteten BIND 8 und damit ist nicht klar inwiefern sie die aktuellen DNS Standards unterstützt. Aus diesen Gründen ist die libresolv aus unserer Sicht nicht die erste Wahl für den Einsatz in strongSwan.

4.3.8 Entscheidung

Aus unserer Sicht eignet sich die ldns Bibliothek am besten zur Realisierung des DNS basierten Authentifizierungsverfahren in strongSwan. Sie erfüllt alle Anforderungen, die wir als zwingend definiert haben. Vor allem die gute Dokumentation und moderne Architektur sprechen für die ldns Bibliothek. Die LNet Labs, welche hinter der ldns stehen, entwickelten auch die beiden DNS Server NSD und Unbound und bringen daher viel Erfahrung im DNS Bereich mit, was natürlich auch für die Qualität der Bibliothek spricht.

4.4 Schlussfolgerungen aus dem Proof of Concept

Im Rahmen des Proof of Concept konnten wir zeigen, dass die IPSECKEY RR von BIND unterstützt werden und die DNSSEC Unterstützung in BIND funktioniert. Die Erfahrungen, die wir beim Einrichten von BIND mit DNSSEC im Rahmen des Proof of Concepts gemacht haben, werden sich beim Anlegen von Testszenarien für das neue Authentifizierungsverfahren sicher noch als hilfreich erweisen. Weiter konnten wir zeigen, dass es eine Reihe von DNS Resolver Bibliotheken gibt, die sich in strongSwan einsetzen liessen. Wir haben für den Einsatz in strongSwan schliesslich die ldns DNS Resolver Bibliothek ausgewählt, da diese uns am geeignetsten erscheint.
Wir konnten also zeigen, dass die Technologien, auf denen das in Kapitel 3.3 beschriebene Authentifizierungsverfahren aufsetzt, bereitstehen und aus dieser Sicht der Implementierung dieses Authentifizierungsverfahrens in strongSwan nichts mehr im Wege stehen sollte.

5 Grobkonzept zur Implementierung in strongSwan

In diesem Kapitel skizzieren wir, wie wir das in Kapitel 3.3 vorgestellte Authentifizierungsverfahren in strongSwan implementieren wollen.
strongSwan ist in der Programmiersprache C geschrieben wobei ein objektorientierter Programmierstiel verwendet wird, der den Einsatz objektorientierter Konzepte erlaubt [23]. Daher orientiert sich das hier vorgestellte Design zur Implementierung des DNS basierten Authentifizierungsverfahren in strongSwan an Konzepten der Objektorientierung.

Da die IPSECKEY RR zusammen mit DNSSEC die gleiche Funktion wie klassische Zertifikate übernehmen (vgl. Kapitel 3), können wir das neue Authentifizierungsverfahren auf dem bereits implementierten zertifikatbasierten Verfahren aufbauen. Die Zertifikate von VPN Endpunkten werden in strongSwan durch den „credential-manager“ verwaltet [22]. Der „credential-manager“ unterstützt unter anderem die Validierung von Zertifikaten (trust-chain-validation). StrongSwan setzt „authenticator“ Objekte ein, um beim Aufbau der IKE_SA die VPN Endpunkte zu authentifizieren. Diese „authenticator“ Objekte greifen auf den „credential-manager“ zurück, der die Zertifikate der VPN Endpunkte verwaltet mit denen sich die VPN Endpunkte authentifizieren lassen:

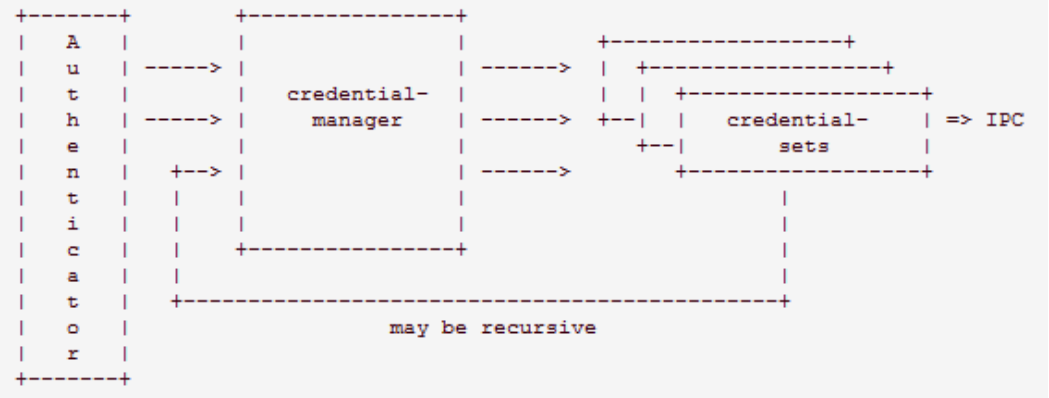


Abbildung 7: Zertifikatsbasierte Authentifizierung in strongSwan [22]

Damit wir die IPSECKEYs wie die klassischen Zertifikate zur Authentifizierung der VPN Endpunkte einsetzen können, bilden wir sie zusammen mit ihrem RRSIG RR auf einen neuen Zertifikatstyp ab. Weiter müssen wir auch die ZSK, KSK und DS Resource Records (inkl. ihrem jeweiligen RRSIG RR) aus dem DNS auf neue Zertifikatstypen abbilden, so dass der „credential-manager“ für die neuen IPSECKEY Zertifikate anhand der DNSSEC RRs eine trust-chain-validation durchführen und so die IPSECKEY Zertifikate validieren kann. Um die IPSECKEY und DNSSEC RR aus dem DNS abzufragen benötigen wir ausserdem einen DNS Resolver und eine Art Fetcher, der das Abrufen der IPSECKEYs und DNSSEC RRs und deren Konvertierung in Zertifikate kontrolliert. Für die Implementierung des neuen Authentifizierungsverfahrens müssen wir strongSwan also um folgende Komponenten erweitern:

- Neue Zertifikatstypen für IPSECKEYs und DNSSEC RRs
- DNS Resolver
- IPSECKEY Fetcher
- Konverter IPSECKEY/DNSSEC RR → Zertifikate

5.1 Design

Die eben gemachten Überlegungen führten uns zu folgenden Design:

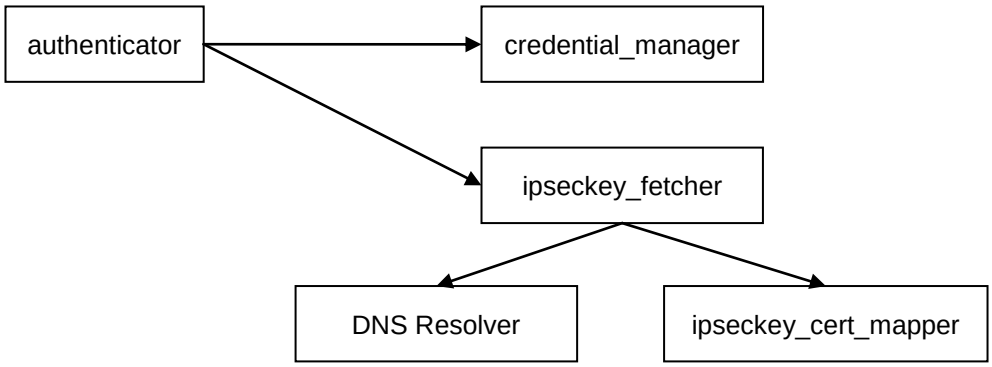


Abbildung 8: Design zur Implementierung der neuen Authentifizierung in strongSwan

Die Aufgabe des authenticators in unserem Design ist es VPN Endpunkte anhand der IKE Nachrichten zu authentifizieren. Dazu greift er auf die im credential_manager abgespeicherten Zertifikate zurück. Für die neue IPSECKEY basierte Authentifizierung benötigen wir einen speziellen ipseckey_authenticator, welcher das bestehende authenticator Interface implementiert. Dieser muss die VPN Endpunkte anhand der IPSECKEY RRs authentifizieren. Dazu verwendet er die IPSECKEY und DNSSEC Zertifikate, die bereits im credential_manager abgelegt sind oder holt sich die Zertifikate bei Bedarf mithilfe des IPSECKEY Fetchers aus dem DNS, legt sie im credential_manager ab und verwendet sie anschliessend.
Wir beschreiben nun die Aufgaben der einzelnen Komponenten aus Abbildung 8 etwas detaillierter, so, dass das Konzept klarer wird.

5.1.1 authenticator

Die Hauptaufgabe des authenticators ist es VPN Endpunkte anhand der IKE Nachrichten zu authentifizieren. Unser ipseckey_authenticator greift dabei auf die neu zu definierenden IPSECKEY und DNSSEC Zertifikate zurück, welche vom credential_manager verwaltet werden. Hält der credential_manager keine passenden Zertifikate bereit so muss der ipseckey_authenticator diese mithilfe des ipseckey_fetcher holen und im credential_manager ablegen bevor er sie zur Authentifizierung verwenden kann. Unser authenticator wird mithilfe des credential_managers die trust-chain-validation selbst durchführen. DNSSEC würde es zwar erlauben, dass die IPSECKEY RR durch einen DNS Server validiert werden. Dies würde aber dazu führen, dass wir neben dem trust anchor auch noch einem weiteren DNS Server vertrauen müssten, was aus Sicherheitsgründen nicht optimal ist. Indem wir die ganze trust-chain-validation selbst durchführen, müssen wir einzig dem trust anchor des DNS Systems vertrauen.
strongSwan definiert bereits ein authenticator Interface und enthält verschiedene authenticator Implementierungen. Der geplante ipseckey_authenticator wird also die Reihe der bereits vorhandenen authenticators lediglich erweitern.

5.1.2 credential_manager

Der credential_manager speichert und verwaltet sogenannte credentials (dt.: Berechtigungs-nachweise). Zertifikate sind ein Beispiel für solche credentials. Der credential_manager kümmert sich unter anderem auch um die Validierung der gespeicherten Zertifikate.
Für die Implementierung des neuen Authentifizierungsverfahrens sind voraussichtlich kaum Änderungen am bestehenden credential_manager notwendig.

5.1.3 DNS Resolver

Der DNS Resolver erlaubt es DNS Abfragen durchzuführen und die Antworten auszuwerten. Wir benötigen ihn zum Abfragen der IPSECKEY und DNSSEC RRs, welche wir im Rahmen des DNS basierten Authentifizierungsverfahrens benötigen.
Momentan enthält strongSwan keinen DNS Resolver. Wir werden die libstrongswan daher um ein DNS Resolver Interface erweitern und ein Plugin entwickeln, welches einen entsprechenden Resolver implementiert. Dieses Plugin wird einen Wrapper um den DNS Resolver der ldns Bibliothek darstellen, welche wir in Kapitel 4.3.8 für den Einsatz in strongSwan ausgewählt haben.

5.1.4 ipseckey_cert_mapper

Aufgabe des ipseckey_cert_mapper ist es die IPSECKEY und DNSSEC Resource Records (KSK, ZSK und DS inkl. RRSIG) in entsprechende Zertifikate abzubilden, welche im credential_manager abgelegt und vom authenticator zur Authentifizierung der VPN Endpunkte verwendet werden können.

5.1.5 ipseckey_fetcher

Der ipseckey_fetcher liefert auf Wunsch die IPSECKEY und DNSSEC Zertifikate, die für die Authentifizierung eines VPN Endpunktes gebraucht werden. Er nutzt den DNS Resolver um die entsprechenden Resource Records abzufragen und wandelt diese dann mithilfe des ipseckey_cert_mapper in entsprechende Zertifikate um und liefert diese.

5.1.6 Offene Probleme

Prof. Andreas Steffen wies uns darauf hin, dass es beim vorgeschlagenen Design Probleme mit der Abbildung von Resource Records, die Bestandteil eines RRset (Menge von Resource Records) sind, auf Zertifikate gibt.
Ein VPN Endpunkt kann z.B. mehrere IPSECKEY Resource Records haben, welche zusammen ein RRset bilden. Für dieses RRset ist dann nur eine RRSIG Signatur über das gesamte RRset (d.h. über alle darin enthaltenen RRs) vorhanden. Für die einzelnen RRs die Bestandteil dieses RRsets sind gibt es also keine eigenständigen RRSIG Signaturen. Zur Abbildung der RRs auf Zertifikate und die anschliessende Validierung dieser Zertifikate benötigt man aber zu jedem RR eine RRSIG Signatur, die nur über den betreffenden RR geht. Daher ist es nicht möglich, wie vorgeschlagen, die RRs mit ihren RRSIGs auf Zertifikate abzubilden und diese anschliessend zu validieren, wenn die RRs Bestandteil eines RRsets sind.
Eine mögliche Lösung des Problems könnte darin bestehen, dass der ipseckey_fetcher die Validierung der IPSECKEY RRs (inkl. trust-chain-validation) durchführt. Korrekt validierte IPSECKEY RR könnte man dann auf als vertrauenswürdig markierte Zertifikate abbilden und im credential_manager ablegen. Die im credential_manager gespeicherten IPSECKEY Zertifikate könnten direkt zur Authentifizierung der VPN Endpunkte eingesetzt werden. Auf die Abbildung der DNSSEC RRs auf Zertifikate könnte man in diesem Fall ganz verzichten.
Wie wir dieses Problem genau lösen werden, ist noch offen. Im Rahmen der zweiten Projektarbeit werden wir eine Lösung erarbeiten.

5.2 Geplantes Vorgehen bei der Implementierung

Wir planen die einzelnen Bestandteile unseres Designs in folgender Reihenfolge zu implementieren:

1. DNS Resolver
2. ipseckey_cert_mapper (gemäss der noch zu definierenden neuen Zertifikatstypen)
3. ipseckey_fetcher und ipseckey_authenticator

Der ipseckey_authenticator arbeitet eng mit dem ipsseckey_fetcher zusammen. Daher macht es Sinn, diese parallel zu implementieren und zu testen.

5.2.1 Tests

Jede der zu implementierenden Komponenten werden wir im Rahmen der Implementierung separat testen. Um das Zusammenspiel der Komponenten zu testen, werden wir analog zu den bereits in strongSwan existierenden UML Testszenarien eigene UML Testszenarien anlegen, welche später auch als Regressionstests dienen können. Mögliche Testszenarien sind beispielsweise:

- Host to Host VPN mit IPv4
- Host to Host VPN mit IPv6
- Gateway to Gateway VPN mit IPv4
- Gateway to Gateway VPN mit IPv6

6 DNS Resolver für strongSwan

Der DNS Resolver wird ein Bestandteil der libstrongswan Bibliothek. Wir werden den DNS Resolver analog zum bereits in der libstrongswan enthaltenen fetcher in die libstrongswan einbinden:

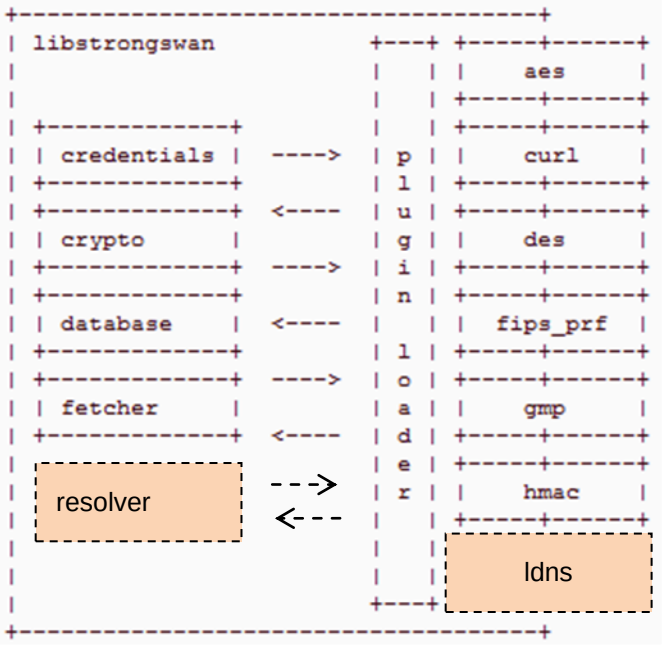


Abbildung 9: Integration des DNS Resolvers in die libstrongswan (Adaption von [24])

Beim fetcher wird ein fetcher Interface definiert, welches dann durch das „curl“ Plugin implementiert wird. Analog definieren wir für den Resolver ein Resolver Interface und erstellen ein Resolver Plugin, das dieses Interface implementiert (siehe Abbildung 9). Unser Plugin wird auf der ldns DNS Resolver Bibliothek basieren (vgl. Kapitel 4.3.8). Die Plugin basierte Architektur erlaubt es später alternative Resolver Plugins zu implementieren und einzusetzen.

6.1 Implementierung des Resolvers

Im Rahmen der vorliegenden Arbeit haben wir bereits damit begonnen erste Teile des DNS Resolvers in strongSwan zu implementieren. In einem separaten Branch (Brachname: „dnssec“) des strongSwan Projekts haben wir mit der Unterstützung von Prof. Andreas Steffen als erstes Gerüste der Klassen für den Resolver und das Resolver Plugin erstellt, die den Resolver in die libstrongswan einbinden. Zur Einbindung des Resolvers in die libstrongswan haben wir folgende Komponenten erstellt:

- Resolver Interface („resolver.h“)
- Resolver Manager („resolver_manager.h“ und „resolver_manager.c“)
- ldns Plugin („ldns_plugin.h“ und „ldns_plugin.c“)
- ldns Resolver welcher das Resolver Interface implementiert („ldns_resolver.h“ und „ldns_resolver.c“)

Wir wollen nun die einzelnen Komponenten und ihr Zusammenspiel näher betrachten.

6.1.1 Resolver Interface

Das Resolver Interface legt fest, welche Methoden und Attribute ein Resolver Objekt aufweist. Die anderen Bestandteile von strongSwan (z.B. der ipseckey_fetcher aus Kapitel 5.1) werden den Resolver durch dieses Interface nutzen. Dadurch kann die Implementierung des Resolvers bei Bedarf geändert werden, ohne dass man die anderen Komponenten von strongSwan anpassen muss.

6.1.2 Resolver Manager

Die libstrongswan wird durch ein library Objekt der Klasse library_t repräsentiert. Mithilfe dieses library Objekts kann man auf die einzelnen Bestandteile der libstrongswan zugreifen. Für den Resolver haben wir in der Klasse library_t ein Feld „resolver“ definiert, über welches man auf den Resolver Manager zugreifen kann. Der Resolver Manager verwaltet die registrierten Resolver Plugins und liefert auf Wunsch ein Resolver Objekt.

6.1.3 ldns Plugin

Die Aufgabe des ldns Plugins ist es den Konstruktor des ldns Resolvers beim Resolver Manager zu registrieren und den ldns Resolver damit in die libstrongswan einzubinden.

6.1.4 ldns Resolver

Beim ldns Resolver handelt es sich um einen auf der ldns Bibliothek basierenden DNS Resolver, der das in der Datei „resolver.h“ definierte Interface implementiert. In der Klasse „ldns_resolver_t“ ist also der eigentliche DNS Resolver, welchen wir einsetzen werden, implementiert.

6.2 Stand der Implementierung

Momentan sind nur diejenigen Komponenten des Resolvers implementiert, welche ihn in die libstrongswan einbinden. Das Resolver Interface definiert momentan lediglich eine Methode und der ldns Resolver liefert beim Aufruf dieser Methode zu Demonstrationszwecken den A RR der Domain „www.google.ch“. Ein Grossteil der Implementierung des DNS Resolvers ist also noch auszuführen.

7 Schlusswort

Im Rahmen der vorliegenden Arbeit konnten wir ein auf DNS Einträgen basierendes Authentifizierungsverfahren für VPN Endpunkte entwickeln. Dieses Verfahren zeichnet sich dadurch aus, dass es die Authentifizierung von VPN Endpunkten durch DNS Einträge erlaubt und somit den Betrieb von IPsec VPNs erleichtert, da man sich nicht mehr wie bei den bestehenden Authentifizierungsverfahren um die Verwaltung von Pre-Shared Keys oder den Unterhalt einer Public Key Infrastructure kümmern muss. Das vorgeschlagene Verfahren ist nicht gänzlich neu, sondern basiert im Wesentlichen auf Ideen, die im Zusammenhang mit der sogenannten „Opportunistic Encryption“ [25] entstanden. So nutzt das von uns vorgeschlagene Authentifizierungsverfahren die bereits standardisierten IPSECKEY Resource Records [4], die im Zusammenhang mit der „Opportunistic Encryption“ entwickelt wurden. Durch den von uns durchgeführten Proof of Concept konnten wir zeigen, dass die wesentlichen Technologien zur Realisierung des von uns vorgeschlagenen Authentifizierungsverfahrens bereitstehen und seiner Implementierung daher nichts im Wege stehen dürfte.

Wir entwickelten daher ein Konzept zur Implementierung des neuen Authentifizierungsverfahrens in strongSwan. Bei der Entwicklung dieses Konzepts haben wir darauf geachtet, dass es das neue Authentifizierungsverfahren möglichst gut in die bestehende strongSwan Architektur einbindet. Dazu haben wir uns an der Implementierung der in strongSwan bereits realisierten Authentifizierungsverfahren orientiert. Es zeigte sich, dass das vorgeschlagene Konzept noch ein Problem im Umgang mit RRsets hat, welches wir im Rahmen der nächsten Projektarbeit noch lösen müssen. Unser Konzept sieht den Einsatz einer DNS Resolver Bibliothek in strongSwan vor. Daher haben wir verschiedene DNS Resolver Bibliotheken für den Einsatz in strongSwan evaluiert und uns schliesslich für die ldns Bibliothek entschieden. Basierend auf der ldns Bibliothek haben wir dann mit der Implementierung eines DNS Resolvers in strongSwan begonnen.

Bei der vorliegenden Arbeit ging es vor allem um das Entwickeln und Prüfen von Konzepten. Da das vorgeschlagene Authentifizierungsverfahren auf bestehenden Technologien (DNS, DNSSEC, IPSECKEY, …) aufbaut war ein grosser Bestandteil der Arbeit das Lesen (und verstehen) bestehender Standards. Da wir kaum Erfahrungen mit IPsec und strongSwan hatten, mussten wir uns auch in dieses Gebiet einarbeiten. Wir haben im Rahmen der vorliegenden Projektarbeit daher in den genannten Gebieten viel gelernt.

In der nächsten Projektarbeit werden wir das in der vorliegenden Arbeit vorgestellte Authentifizierungsverfahren in strongSwan implementieren.

8 Quellenverzeichnis

[1] http://www.strongswan.org, zuletzt abgerufen am 07.01.2012.

[2] S. Kent, K. Seo: Security Architecture for the Internet Protocol. RFC 4301, December 2005.

[3] R. Arends, R. Austein, M. Larson, D. Massey, S. Rose: DNS Security Introduction and Requirements. RFC 4033, March 2005.

[4] M. Richardson: A Method for Storing IPsec Keying Material in DNS. RFC 4025, February 2005.

[5] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen: Internet Key Exchange Protocol Version 2 (IKEv2). RFC 5996, September 2010.

[6] A. Steffen: Vorlesungsunterlagen zur Vorlesung „Internet Security 1“. Herbstsemester 2010, Hochschule für Technik Rapperswil.

[7] RFC Editor: Internet Official Protocol Standards. RFC 5000, May 2008.

[8] http://www.isc.org, zuletzt abgerufen am 03.11.2011.

[9] Evi Nemeth, Trent Hain, Garth Snyder, Ben Whaley, et. al.: Unix and Linux System Administration Handbook. Prentice Hall International, 4th edition, 2011.

[10] Alan Clegg: DNSSEC in 6 minutes. Internet Systems Consortium, Version 1.5, 2008. http://www.isc.org/files/DNSSEC_in_6_minutes.pdf, zuletzt abgerufen am 21.11.2011.

[11] http://www.nlnetlabs.nl/, zuletzt abgerufen am 12.12.2011.

[12] http://en.wikipedia.org/wiki/Root_nameserver, zuletzt abgerufen am 12.12.2011.

[13] glibc 2.14. Verfügbar unter: http://ftp.gnu.org/gnu/glibc/, zuletzt abgerufen am 12.12.2011.

[14] http://www.gnu.org/software/libc/manual/, zuletzt abgerufen am 12.12.2011

[15] http://www.c3.hu/docs/oreilly/tcpip/dnsbind/ch14_02.htm, zuletzt abgerufen am 12.12.2011

[16] http://www.nlnetlabs.nl/svn/nsd/trunk/doc/ChangeLog, zuletzt abgerufen am 03.01.2012

[17] http://www.corpit.ru/mjt/udns.html, zuletzt abgerufen am 03.01.2012

[18] http://www.chiark.greenend.org.uk/~ian/adns/, zuletzt abgerufen am 03.01.2012

[19] http://c-ares.haxx.se/, zuletzt abgerufen am 03.01.2012

[20] https://www.dnssec-tools.org, zuletzt abgerufen am 03.01.2012

[21] http://unbound.net/documentation/libunbound-tutorial-1.html, zuletzt abgerufen am 03.01.2012

[22] http://wiki.strongswan.org/embedded/strongswan/structcredential__manager__t.html, zuletzt abgerufen am 06.01.2012

[23] http://wiki.strongswan.org/projects/strongswan/wiki/ObjectOrientedC, zuletzt abgerufen am 07.01.2012

[24] http://wiki.strongswan.org/projects/strongswan/wiki/Libstrongswan, zuletzt abgerufen am 07.01.2012

[25] M. Richardson, D.H. Redelmeier: Opportunistic Encryption using the Internet Key Exchange (IKE). RFC 4322, December 2005.

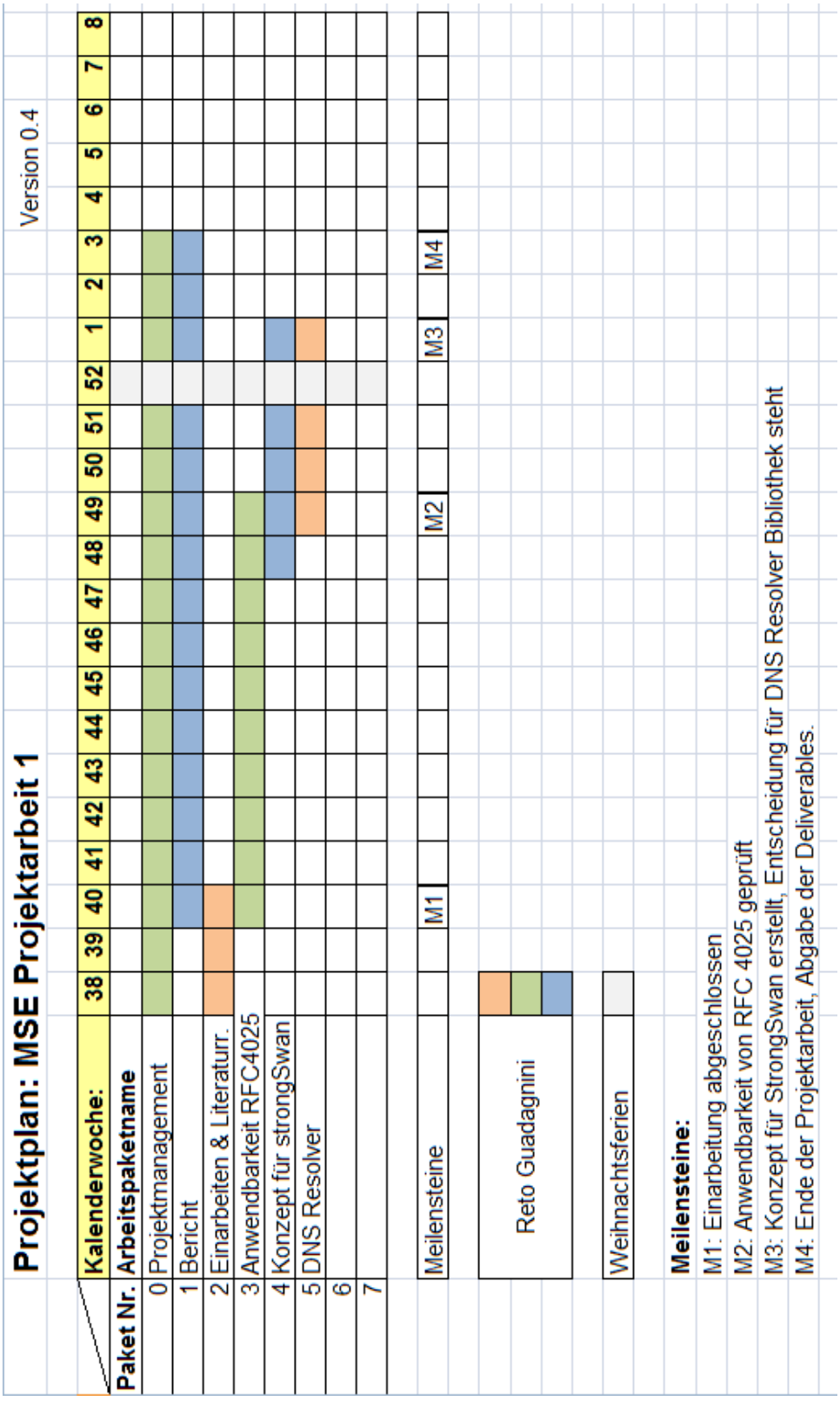
[26] http://www.freeswan.org/ending_letter.html, zuletzt abgerufen am 12.01.2012

9 Glossar

Akronym	Bedeutung
AH	Authentication Header
CA	Certificate Authority
DNS	Domain Name System
DNSSEC	Domain Name System Security Extensions
DS	Delegation Signer
EAP	Extensible Authentication Protocol
EDNS	Extended DNS
ESP	Encapsulation Security Payload
IETF	Internet Engineering Task Force
IKE	Internet Key Exchange
KSK	Key Signing Key
PGP	Pretty Good Privacy
RFC	Request for Comments
RR	Resource Record
RSA	Rivest Shamir Adleman
SA	Security Association
SOA	Start Of Authority
UML	User Mode Linux
VPN	Virtual Private Network
ZSK	Zone Signing Key

10 Anhang

10.1 Projektplanung



10.1.1 Abweichungen zur ursprünglichen Projektplanung

Ursprünglich war das Erreichen des Meilensteins M2 für die Kalenderwoche 47 geplant. Am 10. November haben wir diesen Meilenstein in Absprache mit Prof. A. Steffen auf Kalenderwoche 49 verschoben, da sich das Arbeitspaket „Anwendbarkeit RFC 4025“ als aufwendiger erwies als ursprünglich angenommen.
In der Kalenderwoche 49 haben wir uns dazu entschlossen im Rahmen der Projektarbeit mit der Implementierung eines DNS Resolvers für strongSwan zu beginnen. Daher haben wir am 5. Dezember ein entsprechendes Arbeitspaket „DNS Resolver“ definiert an welchem wir parallel zum Paket „Konzept für strongSwan“ gearbeitet haben. Dementsprechend haben wir den geplanten Arbeitsaufwand für das „Konzept für strongSwan“ Paket reduziert.
Im Dezember hatten wir neben der Projektarbeit relativ viel Arbeit (Seminar). Dies führte dazu, dass wir in der unterrichtsfreien Zeit (ab 23. Dezember) noch mehr an der Projektarbeit arbeiten mussten als ursprünglich geplant war.