LUDWIG-MAXIMILIANS-UNIVERSITÄT
MÜNCHEN

Institut für Informatik

Projektarbeit

Keymanagement mit DNSSec

Alexander Schwinn

Aufgabensteller: Prof. Dr. H.-G. Hegering

Betreuer (LMU): Helmut Reiser
Norbert Wienold

Abgabetermin: Juni 2000

Inhaltsverzeichnis

1. EINFÜHRUNG 6

2. DOMAIN NAME SERVICE (DNS) 6

3. DNS SECURITY EXTENSIONS (DNSSEC) 7

3.1. SCHLÜSSELVERTEILUNG 7

3.1.1. Flags-Field 8

3.1.2. Protocol-Field 9

3.1.3. Algorithm-Field 9

3.1.4. Der SIG Resource Record (SIG RR) 10

3.2. DER CERT RESOURCE RECORD 12

3.3. AUTHENTIFIZIERUNG DER DATENHERKUNFT 13

3.4. DNS TRANSACTION AND REQUEST AUTHENTICATION 13

3.5. SICHERHEITSASPEKTE 14

3.6. PERFORMANCE-BETRACHTUNGEN 14

3.7. OPERATIONELLE BETRACHTUNGEN 15

3.7.1. Schlüsselgenerierung 15

3.7.2. Lebenszeit der Schlüssel 15

3.7.3. Schlüssellänge und Schlüsselarten 16

3.7.4. Aufbewahrung der privaten Schlüssel 16

4. VERSUCHSAUFBAU 17

4.1. INSTALLATION VON BIND 17

4.2. KONFIGURATION VON BIND 18

4.3. ZONENSCHLÜSSELERSTELLUNG 22

4.4. SCHLÜSSEL IN DATENBANK-FILE INTEGRIEREN 24

4.5. SIGNIEREN DER ZONE 25

4.6. ERSTELLUNG VON BENUTZER/HOST-SCHLÜSSELN 27

5. DNSSEC IN VERBINDUNG MIT IPSEC 33

5.1. IPSEC MIT MANUAL KEYING 33

5.2. IPSEC MIT AUTOMATIC KEYING 33

5.3. IPSEC MIT DNSSEC 34

5.4. AKTUELLE IMPLEMETIERUNGEN VON IPSEC/DNSSEC 34

6. ZUSAMMENFASSUNG 35

LITERATURVERZEICHNIS 36

Abbildungsverzeichnis

Abbildung 1: Aufbau des KEY Resource Records 8
Abbildung 2: Aufbau des SIG Resource Records. 11
Abbildung 3: Aufbau des CERT Resource Records..12

Tabellenverzeichnis

Tabelle 1: Derzeit unterstützte Protokolle im KEY Resource Record 9
Tabelle 2: Unterstützte Algorithmen im KEY Resource Record.............................10
Tabelle 3: Beispiel Lables Field 11
Tabelle 4: Zertifikatstypen..13
Tabelle 5: Rechner im Versuchsaufbau 17
Tabelle 6: Beispiel Key Resource Record 31
Tabelle 7: Beispiel SIG Resource Record 32

1. Einführung

Diese Arbeit beschäftigt sich mit der Schlüsselverteilung in TCP/IP basierten Netzen. Es stellt sich die Frage, wie man in kleinen und großen Organisationen öffentliche Schlüssel effektiv verteilen kann. Ein weiterer Gesichtspunkt, der bei der Schlüsselverteilung beachtet werden sollte, ist die zentrale Speicherung der öffentlichen Schlüssel, damit nicht jeder Client eine eigene Schlüsselverwaltung leisten muss. In dieser Arbeit soll untersucht werden, wie man öffentliche Schlüssel effektiv verteilen kann, um verschiedene Anwendungen sicherer zu machen, da das sehr weit verbreitete TCP/IP-Protokoll ohne Erweiterungen keine Sicherheitsmechanismen zur Verfügung stellt. Diese Schlüssel können für verschiedene Dienste, wie z.B. Email, Telnet, FTP, usw. verwendet werden, um die Kommunikationsverbindungen zu sichern. Die hier untersuchte Möglichkeit, den Domain Name Service (DNS) zu verwenden, bietet zudem den Vorteil, evtl. vorhandene Ressourcen zu nutzen. In größeren TCP/IP-basierten Netzten ist ein DNS-Server meist vorhanden, um IP-Adressen in Domainnamen aufzulösen und umgekehrt. Als prototypische Implementierung eines Key Distribution Centers wurde der Berkley Internet Name Daemon (BIND) verwendet. In der hier verwendeten Version (BIND 8.2.2) sind die DNS Security Extensions (DNSSec), die in [RFC 2535] spezifiziert sind, weitgehend implementiert. Im folgenden sollen nun zunächst die Funktionen der DNS Security Extensions beschrieben werden. Anschließend wird exemplarisch ein einfacher Versuchsaufbau aufgezeigt, der eine Schlüsselverteilung über einen DNS-Server ermöglicht.

Da bisher nur sehr wenig mit DNSSec experimentiert worden ist, stützt sich diese Arbeit hauptsächlich auf RFC-Dokumente und Internet-Drafts (*http://www.ietf.org*). In der BIND-Distribution sind zwar einige Tools zum Aufbau eines Schlüsselverteilungsservers enthalten, der Einsatz in der Praxis scheitert wohl momentan noch an dem Mangel von Applikationen, die diese Technik unterstützen. Da zukünftige Versionen von BIND DNSSec noch mehr unterstützen werden, ist es wohl eine Frage der Zeit, bis Anwendungen implementiert werden, die diese Technik nutzen, da in fast allen DNS-Servern BIND verwendet wird. Die neue Version von BIND (9.0), die volle DNSSec-Unterstützung bietet, soll laut ISC (Internet Software Consortium http://www.isc.org/bind) ab März 2000 verfügbar sein.

2. Domain Name Service (DNS)

Da Programme und Benutzer selten auf die binäre Netzadresse zugreifen, wurde ein Mechanismus entwickelt, der es ermöglicht den binären Netzadressen ASCII-Zeichenketten zuzuordnen. In Netzen mit wenigen Hosts geschieht dies über eine Datei (meist `/etc/hosts` bei UNIX-Betriebssystemen), in der in tabellarischer Form jeder IP-Adresse eine eindeutige ASCII-Zeichenkette zugeordnet ist. Das

Problem daran ist, dass diese Datei auf jedem Rechner vorhanden sein muss. Kommt also ein neuer Host zu dem Netz dazu, muss dieser in allen Dateien eingetragen werden, bzw. muss sich jeder Host in zyklischen Abständen eine neue Host-Datei vom Master besorgen. Betrachtet man nun z.B. das Internet, sieht man, dass dieser Mechanismus nicht funktioniert, da zum einen die Host-Datei viel zu groß werden würde und zum anderen eine zentrale Verwaltung mit einem Master nicht mehr möglich ist. Deshalb wurde das Domain Name System (DNS) eingeführt, das einen hierarchischen Aufbau hat. Um nun einen Namen in eine IP-Adresse umzuwandeln ruft ein Anwendungsprogramm eine Prozedur namens *Resolver* auf, an die es den gesuchten Namen als Parameter übergibt. Der Resolver sendet ein UDP-Paket an den lokalen DNS-Server, der nach dem entsprechenden Eintrag sucht und die passende IP-Adresse an den Resolver übergibt. In der Vergangenheit wurde DNS hauptsächlich zur Umwandlung von Hostnamen und E-Mailadressen in IP-Adressen benutzt. Alle Einträge in der DNS-Datenbank werden in sogenannten Ressource-Records (RR) gespeichert, auf die im folgenden noch näher eingegangen wird. DNS ist in [RFC 1034] und [RFC 1035] vollständig definiert. Zahlreiche Erweiterungen kamen in den letzten Jahren hinzu, u.a. die DNS Security Extensions (DNSSec), die diese Arbeit näher untersucht.

3. DNS Security Extensions (DNSSec)

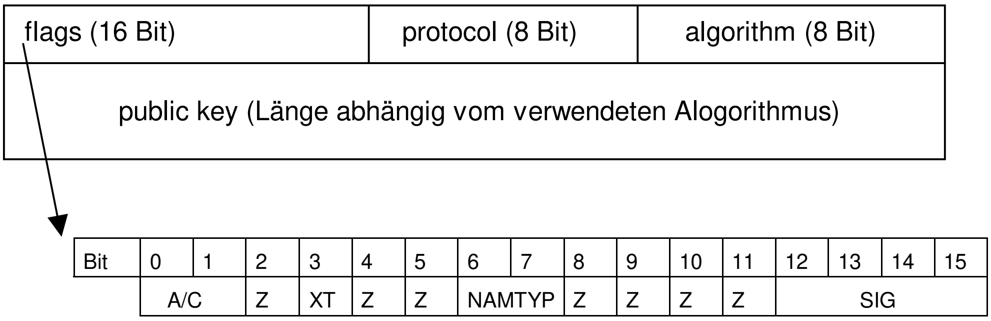
Im wesentlichen stellen die DNS Security Extensions (DNSSec), wie in [RFC 2535] spezifiziert, drei neue Funktionen zur Verfügung:

- Schlüsselverteilung
- Authentifizierung der Datenherkunft von einem DNS-Server
- Sicherung der Kommunikationsverbindung zwischen einem DNS-Server und einem Resolver, bzw. zwischen einem DNS-Server und einem weiteren DNS-Server

Aufgrund der Philosophie von DNS, dass Daten aus der DNS Datenbank für jeden öffentlich zugänglich sein sollen, wurden in DNSSec keine Access-Control-Listen, oder andere Einschränkungen für den Zugriff eingeführt.

3.1. Schlüsselverteilung

Um eine Schlüsselverteilung überhaupt zu ermöglichen wurde ein neuer Resource Record (RR) eingeführt, der sogenannte *KEY RR*. Jedem Eintrag in der DNS-Datenbank ist ein Resource Record zugeordnet, der angibt, um was für eine Art von Eintrag es sich handelt (z.B. IN (Internet Name), SOA (Start of Authority), MX (Mail Exchange), TXT (Text), ...). Der KEY Resource Record ermöglicht es nun, einen öffentlichen Schlüssel mit einem DNS-Namen in Verbindung zu bringen. Es werden in der DNS-Datenbank generell nur öffentliche Schlüssel gespeichert, wobei dieser öffentliche Schlüssel einem Host, einem User, oder einer Zone zugeordnet werden kann. Bei der Zuordnung zu einem User, wird die übliche BIND-Konvention verwendet (z.B. user.test.com entspricht user@test.com). Der KEY RR ist folgendermaßen aufgebaut:



Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	A/C		Z	XT	Z	Z	NAMTYP		Z	Z	Z	Z	SIG			

Abbildung 1: Aufbau des KEY Resource Records

3.1.1. Flags-Field

Die ersten 16 Bit können folgende Werte annehmen:

- **A/C:**
  Schlüsselart. Folgende Werte sind möglich
  - 10: Schlüssel darf nicht zum Authentifizieren verwendet werden.
  - 01: Schlüssel darf nicht zum Verschlüsseln verwendet werden.
  - 00: Schlüssel kann zum Authentifizieren und zum Verschlüsseln verwendet werden.
  - 11: Keine Schlüssel im Feld *public key* vorhanden.

- **Z (Zero)**
  Felder, die mit „Z“ gekennzeichnet sind, sind für späteren Gebrauch reserviert und müssen eine Null enthalten. Sie werden derzeit nicht verwendet.

- **XT (Extension bit)**
  Dieses Bit gibt an, ob ein weiteres Flag-Field existiert. Wenn es auf 1 gesetzt ist, folgt ein weiteres 16-Bit langes Flag-Field nach dem Algorithm-Field und vor dem Public-Key-Field. Dieses Bit dient sozusagen als Reserve, falls man

noch weitere Flags setzten möchte. Bisher wurden aber keine weiteren Flags definiert, die gesetzt werden könnten.

- **NAMTYP:**
  Typ des Eintrags. Folgende Werte kann das Feld annehmen:
    - 00: bedeute, dass der Schlüssel zu einem Benutzer oder Benutzer-Account zugeordnet ist.
    - 01: bedeutet, dass es sich um einen Zonen-Schlüssel handelt.
    - 10: bedeutet, dass es sich weder um einen Benutzerschlüssel noch um einen Zonenschlüssel handelt. Zum Beispiel könnte es sich um eine Telefonnummer [RFC 1530] handeln.

- **SIG (signatory field)**
  wenn diese Bits nicht auf Null gesetzt sind, ist der Schlüssel authorisiert, andere Einträge, wie in [RFC 2137 DNS Dynamic Update] spezifiziert, zu signieren. Es ist zu beachten, dass Zonen-Schlüssel (siehe Feld NAMTYP oben) immer authorisiert sind, jeden beliebigen RR zu signieren.

3.1.2. Protocol-Field

Da die Schlüssel, die im DNS gespeichert werden in Zukunft für viele Internet-Protokolle verwendet werden sollen, enthält der KEY Resource Record ein Protocol-Field. Damit kann man die Gültigkeit eines bestimmten Schlüssels auf einzelne Protokolle einschränken. Bisher sind folgende Werte reserviert:

Wert	Protokoll
0	Reserviert
1	TLS
2	Email
3	DNSSec
4	IPSec
5-254	noch verfügbar, Zuordnung erfolgt durch IANA
255	alle, d.h., der Schlüssel kann für jedes Protokoll verwendet werden.

Tabelle 1: Derzeit unterstützte Protokolle im KEY Resource Record

3.1.3. Algorithm-Field

Dieses Feld gibt an, welcher Algorithmus zur Schlüsselerzeugung verwendet wurde. Je nach dem, welcher Algorithmus und welche Schlüssellänge verwendet werden, richtet sich die Länge des „public-key“ Feldes. Folgende Werte wurden bisher für das Algorithmusfeld vergeben:

Wert	Algorithmus
0	Reserviert
1	RSA/MD5 [RFC 2527] (dieser Algorithmus wird empfohlen)
2	Diffie-Hellman [RFC 2539]
3	DSA [RFC 2536]
4	Reserviert
5-251	noch frei
252	Reserviert

Tabelle 2: Unterstützte Algorithmen im KEY Resource Record

3.1.4. Der SIG Resource Record (SIG RR)

Bisher wurde der KEY Resource Record betrachtet. Allein dieser RR kann keine sichere Schlüsselverteilung garantieren, da dieser, wie anderer RRs auch, durch bekannte Angriffe manipuliert werden kann. Eine solche Angriffsmöglichkeit wäre z.B. das DNS-Spoofing, bei dem dem Zielsystem falsche Daten eines DNS-Servers zugespielt werden. Dies kann entweder durch Manipulation des DNS selbst erfolgen oder durch Man-In-the-middle-Angriffe, bei denen sich der Rechner des Angreifers als legitimer DNS-Server ausgibt. Deshalb muss es zu jedem KEY Resource Record einen Signature Resource Record (SIG RR) geben, der zum Signieren des KEY Resource Records verwendet wird. Zum Signieren wird der private Schlüssels des „signers“ verwendet. Der „Signer“ ist immer der Eigentümer der Zone. Nur er kann die Resource Records signieren. Wenn man also der Zone vertrauen schenkt, so kann man auch den signierten Schlüsseln (KEY RR) trauen. Der SIG Resource Record ist das Herzstück von DNSSec. Durch ihn werden die drei oben genannten Erweiterungen in DNSSec gewährleistet. Er spielt nicht nur bei der Schlüsselverteilung eine Rolle, sondern genauso bei der Authentifizierung von Anfragen (z.B. MX Resource Record,...). Der SIG Resource Record hat folgenden Aufbau:

type covered (16 Bit)	algorithm (8 Bit)	labels (8 Bit)
original TTL		
signature expiration (32 Bit)		
signature inception (32 Bit)		
key tag (16 Bit)	signer’s name (48 Bit)	
signature (64 Bit)		

Abbildung 2: Aufbau des SIG Resource Records.

Die einzelnen Felder haben folgende Bedeutung:

- **type covered**
  gibt den RR-Typ an, der signiert worden ist (z.B. KEY, NS,....).

- **Algorithm Number Field**
  wie beim KEY-RR.

- **Lables Field**
  das Lables Field enthält einen Zähler, der angibt, wieviele Einträge der ursprüngliche owner SIG-RR enthält. Dieses Feld ermöglicht es dem Resolver die ursprüngliche Form des Eintrags leichter zu bestimmen (falls Wildcards bei der Anfrage verwendet worden sind), um die angefragten Einträge verifizieren zu können. Wenn der angeforderte RR einen längeren Namen aufweist, als in dem Labels-Feld angegeben, weiss der Resolver, dass Wildcard verwendet worden sind. Enthält der RR einen kürzeren Namen, als in dem Lables-Feld angegeben, wird der SIG RR ignoriert, da er fehlerhaft sein muss (*bad*). Folgendes Tabelle gibt ein Beispiel dafür:

lables=	0	1	2	3	4
.	.	bad	bad	bad	bad
d.	*.	d.	bad	bad	bad
c.d.	*.	*.d.	c.d.	bad	bad
b.c.d.	*.	*.d.	*.c.d.	b.c.d.	bad
a.b.c.d.	*.	*.d.	*.c.d.	*.b.c.d.	a.b.c.d.

Tabelle 3: Beispiel Lables Field

- **Original TTL (Time To Live) Field**

Im Gegensatz zum aktuelle TTL-Feld, kann dieses nicht verändert werden, z.B. beim „Caching“ oder durch Manipulation.

- **Signature Expiration and Inception Fields**
  Die Signatur (SIG) ist gültig von „Inception Field“ bis „Siganture Expiration“. In beiden Feldern steht die Anzahl der Sekunden, die seit 01.01.1970 vergangen sind.

- **Key Tag Field**
  Checksumme des zugehörigen KEY RR, damit der zugehörige Schlüssel leichter identifiziert werden kann, falls mehrere Schlüssel verwendet werden. Abhängig vom verwendeten Algorithmus.

- **Signer‘s Name Field**
  Domain Name des „Signer’s“, der die SIG RR erstellt hat.

- **Signature Field**
  Bindet den SIG-RR an den RR, der in „type covered“ angegeben ist, indem er den Namen und Typ des RR‘s speichert.

3.2. Der CERT Resource Record

Der SIG RR ist nicht dazu geeignet Zertifikate zu speichern. Hierfür wurde in [RFC 2538] ein eigener Certificate Resource Record (CERT RR) spezifiziert. Es ist möglich, bereits vorhandene Zertifikate (z.B. X.509 oder PGP-Zertifikate) zu integrieren. Der CERT RR hat folgenden Aufbau:

type (16 bit)	key tag (16 Bit)
algorithm (8 Bit)	certificate or CRL (=certificate revocation list) (56 Bit)

Abbildung 3: Aufbau des CERT Resource Records.

- **Type Field**
  folgenden Werte wurden bisher definiert:

Wert	**Typ**
0	reserviert
1	X.509 / PKIX
2	SPKI cert
3	PGP cert
4-252	noch verfügbar

253	URI private
254	OID private
255-65534	noch verfügbar
65535	reserviert

Tabelle 4: Zertifikatstypen

Das Key-Tag-Feld und das Algorithm-Feld sind wie bei dem SIG RR definiert. Das Certificate/CRL-Feld wird im Format Base 64 codiert [RFC 2538].

3.3. Authentifizierung der Datenherkunft

Die zweite wesentliche Erweiterung von DNSSec ist die sogenannte „Data Origin Authentication and Integrity". Dabei wird in der DNS Datenbank jeder Eintrag signiert. In der Regel wird dazu ein einziger privater Schlüssel (im folgenden oft Zonenschlüssel genannt) verwendet. Mit dem öffentlichen Schlüssel der Zone, der ebenfalls in der DNS-Datenbank abgelegt ist, können dann die einzelnen Einträge anhand der Signatur verifiziert werden. Somit weiss ein Client, der eine Anfrage an den DNS-Server gestellt hat, dass die zurückgelieferte Information (z.B. eine IP-Adresse zu einem Hostnamen) korrekt ist. Der Client kann anhand des öffentlichen Zonenschlüssels den SIG-RR überprüfen. Der private Schlüssel der Zone, der dazu verwendet worden ist, alle RRs in der DNS-Datenbank zu signieren, sollte an einem sicheren Platz aufbewahrt werden. Er wird weiterhin benötigt, um die Zone neu zu signieren, falls neue Einträge in die DNS-Datenbank hinzukommen. Der Resolver liest den öffentliche Schlüssel entweder aus dem DNS aus, oder er wird statisch im Resolver konfiguriert.

Die Verwendung des SIG RR gewährleistet Authentifizierung von RR, die in der DNS-Datenbank vorhanden sind. Das Nichtvorhandensein von bestimmten Namen oder RR-Typen wird damit nicht authentifiziert. Deshalb wurde der NXT RR (Next Resource record) eingeführt. Somit bekommt ein Resolver immer einen signierten RR zurück, auch wenn zu seiner Anfrage kein entsprechender Eintrag existiert. Somit weiss der Resolver, dass der entsprechende Eintrag nicht in der DNS-Datenbank vorhanden ist.

3.4. DNS Transaction and Request Authentication

Authentifizierung der Datenherkunft bietet keinen Schutz bei DNS-Anfragen oder gegen Manipulation des message headers [RFC 1035]. Beispielsweise könnte ein Anfrage manipuliert werden, oder die Verbindung übernommen werden (session hijacking). Nur das Ergebnis der Anfrage kann authentifiziert werden. Damit ein Benutzer oder eine Anwendung weiss, dass die Antwort auch wirklich von dem Server kommt, an die er sie gestellt hat, bzw. dass genau der Server die Message weiterleitet, und sie nicht auf dem Weg manipuliert worden ist, wird jede Message, bzw. jede Antwort mit einem weiteren SIG RR signiert, der sich am Ende der Message befindet. Der private Schlüssel dazu gehört nicht der Zone, sondern ist in der Regel der Schlüssel von dem DNS-Server, der die Message/Antwort signiert. Zwei Probleme tauchen jedoch auf: zum ersten muss jede Message/Antwort signiert werden. Dies nimmt nicht unerhebliche Rechenzeit in Anspruch. Zweitens können die Messages/Antworten nicht schon vorher (vor der Anfrage) generiert werden, da die Inhalte zu unterschiedlich sind. Dies wäre jedoch evtl. wünschenswert, denn dann müßte der private Schlüssel von dem DNS-Server, der die Messages/Antworten signiert, nicht online verfügbar sein.

3.5. Sicherheitsaspekte

Ein zentrales Problem stellt die Tatsache dar, dass ein Resolver mindestens einen öffentlich Schlüssel benötigt, um die SIG-RR authentifizieren zu können. Hier gibt es zwei Möglichkeiten. Die erste Möglichkeit besteht darin, dass man den Schlüssel statisch im Resolver konfiguriert. Dies würde aber einen erheblichen Aufwand zur Folge haben, da alle Resolver auf den Clients ersetzt werden müßten. Dies wäre auch dann der Fall, wenn sich der öffentliche Schlüssel der Zone einmal ändern sollte. Die zweite Möglichkeit besteht darin, den öffentlich Schlüssel von dem lokalen DNS-Server der Organisation abzufragen. Geht man davon aus, dass man dem lokalen DNS-Server vertrauen schenkt, ist dies die bessere Alternative. Sollte der Resolver eine Anfrage stellen, die nicht die Zone innerhalb der Organisation betrifft, kann die Anfrage über den internen DNS-Server verarbeitet werden, der evtl. wieder öffentliche Schlüssel von anderen DNS-Servern statisch konfiguriert hat. Somit sind Man-In-The-Middle Attacken, bei denen ein Angreifer versucht, die Verbindung zwischen dem DNS-Server und dem Client zu übernehmen, und diesem dann fehlerhafte öffentlich Schlüssel zuspielt, bzw. Einträge mit diesem selbst erstellten Schlüssel signiert, nicht mehr durchführbar. Ebenso sind Versuche, den DNS-Cache mit fehlerhaften Einträgen zu manipulieren erfolglos, da es einem Angreifer nicht möglich ist einen entsprechenden SIG RR mitzuliefern.

3.6. Performance-Betrachtungen

Unter Umständen kann das Signieren von Zonen und die Generierung von Schlüsseln, je nach Größe (Anzahl der RR) einige Zeit in Anspruch nehmen. Dies ist natürlich auch immer vom gewählten Algorithmus und von der Schlüssellänge, sowie von der zur Verfügung gestellten Rechenkapazität abhängig [RFC 2065]. Bei Verwendung der Schlüsselverteilungsfunktion von DNSSec spielt dies allerdings keine große Rolle, da die Datenbankdateien von DNS-Servern nicht allzu oft neu signiert werden müssen. Dies ist nur der Fall, wenn sich Einträge ändern, oder neue Einträge hinzukommen. Performanceprobleme wird man eher, wie oben beschrieben, bei der „DNS Transaction and Request Authentication" bekommen, da für jede einzelne Anfrage, die gestellt wird, eine Signatur erstellt werden muss. Da das Domain Name System hierarchisch aufgebaut ist, kann es vorkommen, dass der lokale DNS-Server weitere Anfragen an den in der Hierarchie nächsten DNS-Server stellen muss, usw., d.h. jede Message muss von jedem DNS-Server, der in der Hierarchie durchlaufen wird, signiert werden.

3.7. Operationelle Betrachtungen

3.7.1. Schlüsselgenerierung

Es ist absolut wesentlich, dass die Schlüssel sorgfältig generiert werden. Es hilft wenig, eine große Schlüssellänge zu wählen, wenn ein Angreifer später genug Informationen bekommen kann, um den Schlüsselraum massiv zu verkleinern, um so einen Schlüssel knacken zu können. Vorschläge, wie man zufällige Schlüssel erzeugen kann, findet man in [RFC 1750]. Wenn man Schlüssel generieren will, die eine lange Lebenszeit haben, so wird dringend geraten, dass solche Schlüssel offline erzeugt werden, d.h., dass ein Angreifer, für den ja gerade langlebige Schlüssel interessant sind, keine Möglichkeit hat, eine Sicherheitslücke auszunutzen, um an die privaten Schlüssel zu gelangen.

3.7.2. Lebenszeit der Schlüssel

Schlüssel sollten aus Sicherheitsgründen immer ein „Verfallsdatum" haben, denn desto länger ein Schlüssel gültig ist, desto länger hat ein Angreifer Zeit, den Schlüssel zu knacken, bzw. wenn er ihn bereits geknackt hat, ihn zu verwenden. Auf der anderen Seite sollte die Lebenszeit eines Schlüssels auch nicht zu kurz sein, da eine Schlüsselgenerierung unter Umständen eine sehr aufwendige und zeitraubende Prozedur sein kann. Dies hängt natürlich auch immer von der Anzahl der Schlüssel ab, die verwaltet/neu erstellt werden müssen. In [RFC 2541] werden folgende Werte vorgeschlagen:

- Ein langlebiger Schlüssel sollte niemals länger als 4 Jahre gültig sein [RFC 2541].

- Ein guter Wert für einen langlebigen Schlüssel ist 1 Jahr (vorausgesetzt er wird offline gehalten).[RFC 2541]

- Schlüssel, die online verfügbar sind, und die für Transaktionen verwendet werden, sollten nicht länger als einen Monat gültig sein. In vielen sicherheitskritischen Fällen sollte der Schlüssel sogar nur einen Tag oder wenige Stunden gültig sein.

Auf keinen Fall sollte der Schlüssel, bei Anwendungen im Internet, weniger als 3 Minuten Gültigkeit haben, da in ungünstigen Fällen, IP Pakete mit bis zu 3 Minuten Verzögerung ankommen können [RFC 2541].

3.7.3. Schlüssellänge und Schlüsselarten

Bei der Wahl der Schlüssellänge gibt es immer einen „Trade-off": längere Schlüssel sind zwar sicherer, aber sie nehmen bei der Generierung und bei der Ver-/Entschlüsselung mehr Rechnerzeit in Anspruch. Hinzu kommt, dass bei der Verwendung von DNSSec die KEY RR´s und die SIG RR´s größer werden, was zur Folge hat, dass zum einen die DNS-Datenbank größer wird, und zum anderen besteht die Gefahr, dass es zu DNS UDP-packet-overflows kommt [RFC 2541]. Ein weiterer Nachteil ist, dass der TCP-Verkehr zunimmt, da bei längeren SIG RR und KEY RR mehr übertragen werden muss. [RFC 2541]. Bei der Wahl des MD5/RSA-Algorithmus ist zu beachten, dass die Schlüssellänge mindestens 704 Bits lang sein sollte [RFC 2541]. Dieser Wert wird momentan vorgeschlagen, jedoch wird sich dieser Wert in der Zukunft erhöhen. Außerdem ist anzumerken, dass die Verifikationszeit eines RSA-Schlüssels nicht proportional zum Schlüsselraum wächst. Erhöht man beispielsweise die Schlüssellänge eines RSA-Schlüssels von 640 Bits auf 1280 Bits (also eine Verdopplung der Schlüssellänge), steigt die Verifikationszeit um den Faktor 4, der Schlüsselraum wird aber um 2^{900} vergrößert. DSS Schlüssel sind ähnlich aufwendig zu knacken wie RSA Schlüssel (bei gleicher Schlüssellänge), die dazugehörigen Signaturen sind jedoch um einiges kleiner [RFC 2541].

3.7.4. Aufbewahrung der privaten Schlüssel

Es ist zu empfehlen, soweit möglich, zumindest den privaten Zonen-Schlüssel und eine Kopie des Master Zone Files offline, also auf einem Rechner ohne Netzanschluß aufzubewahren. Die Privat-Keys (z.B. von Users oder Hosts) können in der Regel nicht offline gehalten werden, da sie für verschiedene Anwendungen, wie z.B. DNS Transaction security, email, IPSec,.. verwendet werden.

4. Versuchsaufbau

Für den Versuchsaufbau wurden zwei Rechner verwendet, auf denen jeweils Linux (Suse 6.2) installiert wurde:

Rechnername	**IP-Adresse**
linux1 (bzw. linux1.fopra.edu)	192.168.0.1
linux2 (bzw. linux2.fopra.edu)	192.168.0.2

Tabelle 5: Rechner im Versuchsaufbau

Standardmäßig ist bei dieser Linux-Distribution nur Bind 4.9 enthalten. Die in dem Versuchsaufbau verwendete BIND Version ist 8.2.2. Zunächst wird kurz auf die Installation von BIND eingegangen, und später wird gezeigt, wie man die Erweiterungen von DNSSec nutzen kann, um öffentliche Schlüssel sicher zu verteilen.

4.1. Installation von BIND

Die aktuelle Quellen von Bind findet man unter *ftp://ftp.isc.org/bind*. Die hier verwendete Version ist 8.2.2. Folgende Pakete werden benötigt:

```
bind-contrib_tar.tar (ca. 850KB)
bind-doc_tar.tar (ca. 1.2MB)
bind-src_tar.tar (ca. 1.8MB)
```

Nachdem man die Quellen heruntergeladen und entpackt hat, kann man BIND durch Eingabe von

```
make stdlinks
make clean
make depend
make
```

die Quellen kompilieren. Anschließend wird BIND durch Eingabe von

```
make install
```

installiert. Die Installation von BIND unter Suse Linux 6.2 verlief ohne Probleme. Es ist jedoch zu beachten, dass `flex` oder `lex` installiert sein muss (wird bei der Standardinstallation von SuSe Linux nicht automatisch installiert).

4.2. Konfiguration von Bind

Um den Versuchsaufbau möglichst einfach zu halten, wurde auf einem Rechner ein Primary DNS-Master (linux2.fopra.edu) installiert, d.h. es gibt genau einen DNS-Server in dem Versuchsaufbau. Problematiken wie Zone-Transfers, Loadbalancing, Suchreihenfolgen bei DNS-Anfragen,... wurden hier nicht behandelt. Hier sei auf das Buch „DNS and BIND“ von Paul Albitz und Cricket Liu [DnsB] verwiesen. Das zentrale Konfigurations-File von BIND findet man unter `/etc/named.conf`. Bei BIND Versionen unter 8 heißt dieser File `named.boot`. Die Konfigurationssyntax hat sich aber in den neuen Versionen von BIND (ab Version 8.0) grundlegend geändert. Unsere Zone heißt *fopra.edu*. Der DNS-Server soll nicht an andere DNS-Server angeschlossen werden. Er soll nur Anfragen von Rechnern innerhalb des Versuchsaufbaus beantworten. Für die Timeouts, TTL's,... wurden die Standardwerte verwendet:

```
//Bind Configuration File (/etc/named.conf)

options {
      directory "/usr/local/bind/conf";
};

zone "fopra.edu" in {
      type master;
      file "db.fopra";
};

zone "0.168.192.in-addr.arpa" in {
      type master;
      file "db.192.168.0";
};

zone "0.0.127.in-addr.arpa" in {
      type master;
      file "db.127.0.0";
};
```

Die .in-addr.arpa Einträge sind nicht unbedingt erforderlich. Sie ermöglichen nur, zu einem gegebenen Hostnamen (z.B. linux1.fopra.edu) die entsprechende IP-Adresse (z.B. 192.168.0.1) zu erfragen, bzw. vervollständigen die Einträge um den `localhost` (IP-Adresse 127.0.0.1), der standardmäßig auf jedem Rechner existiert. Die Datei `/etc/named.conf` wird von dem Namedaemon (`named`) als erstes aufgerufen. In ihr wird auf die eigentlichen DNS-Datenbankdateien verwiesen, die der Nameserver laden muss, um Anfragen erfolgreich beantworten zu können. In unserem Fall sind dies die Dateien `db.fopra`, `db.192.168.0` `und` `db.127.0.0` im Verzeichnis `usr/local/bind/conf`:

```
//BIND DB-File /usr/local/bind/conf/db.fopra

fopra.edu. IN SOA linux2.fopra.edu. root.linux2.fopra.edu. (
            5           ; Serial
            10800       ; Refresh after 3 hours
            3600        ; Retry after 1 hour
            604800      ; Expire after 1 week
            86400 )     ; Minimum TTL of 1 day

fopra.edu.  IN NS       linux2.fopra.edu.

; Addresses for the canonical names
localhost.fopra.edu.    IN A  127.0.0.1
linux2.fopra.edu.       IN A  192.168.0.2
linux1.fopra.edu.       IN A  192.168.0.1


//BIND DB-File /usr/local/bind/conf/db.192.168.0

0.168.192.in-addr.arpa. IN SOA linux2.fopra.edu.
root.linux2.fopra.edu. (
            1           ; Serial
            10800       ; Refresh after 3 hours
            3600        ; Retry after 1 hour
            604800      ; Expire after 1 week
            86400 )     ; Minimum TTL of 1 day

; Name server
0.168.192.in-addr.arpa.      IN NS linux2.fopra.edu.

; Addresses for the canonical names
1.0.168.192.in-addr.arpa.    IN PTR linux1.fopra.edu.
2.0.168.192.in-addr.arpa.    IN PTR linux2.fopra.edu.
```

```
//BIND DB-File /usr/local/bind/conf/db.127.0.0

0.0.127.in-addr.arpa. IN SOA linux2.fopra.edu.
root.linux2.fopra.edu. (
            1           ; Serial
            10800       ; Refresh after 3 hours
            3600        ; Retry after 1 hour
            604800      ; Expire after 1 week
            86400 )     ; Minimum TTL of 1 day

; Name server
0.0.127.in-addr.arpa.   IN NS linux2.fopra.edu.

; Addresses for the canonical names
1.0.0.127.in-addr.arpa. IN PTR localhost.
```

Die einzelnen Einträge haben folgende Bedeutung:

- **SOA Record:**
  Der erste Eintrag in einem Datenbankfile ist immer ein SOA-Resource-Record (Start Of Authority). Jeder Datenbankfile enthält genau einen SOA RR. Er gibt an, dass für die angegebene Zone (in unserem Fall z.B. fopra.edu.) dieser DNS-Server verantwortlich ist. IN steht für Internet. Der erste Name nach SOA (in unserem Beispiel linux2.fopra.edu) gibt den Primary Master Server an, d.h. den DNS-Server, der diese Zone pflegt. Der zweite name gibt die Emailadresse des Administrators an (in unserem Beispiel root.linux2.fopra.edu für `root@linux2.fopra.edu)`. Diese Emailadresse wird von dem Nameserver nicht verwendet, sondern dient Benutzern, die bei Problemen den Administrator kontaktieren können.

- **NS Record:**
  Der NS Resource Record gibt einen oder mehrere Nameserver für die Zone an. In unserem Beispiel gibt es einen Nameserver: linux2.fopra.edu.

- **Adress Records**
  Diese Einträge dienen zur Umsetzung von Namen in IP-Adressen. Im Gegensatz dazu dienen PTR Resource records (Pointer) zur Umsetzung von IP-Adressen in Namen.

- **Serial**
  Serial ist ein ganzzahliger Wert, der beliebig gesetzt werden kann. Sollte sich

der Inhalt dieser Datenbankdatei ändern, so muss man diese Nummer erhöhen. Sie dient dazu, dass andere Nameserver (Slave Server) (z.B. Secondary Name Server) erkennen, dass Daten verändert worden sind. Ist dies der Fall, kopiert sich der andere Nameserver die aktuellen Daten.

- **Refresh**
  Gibt an, wie oft ein Slave-Nameserver überprüfen soll, ob er die aktuellen Daten vom Master-Server besitzt.

- **Retry**
  Gibt die Zeit an, nach der es ein Slave Server erneut versuch einen ‚refresh' zu machen, falls dieser nach abgelaufener ‚refresh'-Zeit gescheitert ist.

- **Expire**
  Falls diese Zeit abgelaufen ist, gibt ein Slave-Server keine Antworten mehr, da die vorhandenen Daten zu alt sind, weil ein ‚refresh' über längere Zeit hinweg gescheitert ist.

- **TTL (Time To Live)**
  Diese Zeitangabe gilt für alle Resource Records. Sie gibt an, wie lange andere Nameserver einen Eintrag ‚cachen' dürfen bevor er verfällt. TTL wird bei Anfragen mit übertragen.

Wenn man diese Konfigurationsfiles erstellt hat, kann man testen, ob der DNS-Server läuft. Gestartet wird der Namedaemon (DNS-Server) durch Eingabe von `named`. Standardmäßig werden (Fehler-)Meldungen in die Datei `/var/log/messages` geschrieben. Am besten läßt man sich diese Datei mit `less -f /var/log/messages` anzeigen, damit die Ansicht bei neuen Meldungen aktualisiert wird. Wenn alles funktioniert hat sollte in dieser Datei folgende Meldung erscheinen:

```
Oct 29 15:29:25 linux2 named[477]: starting.
Oct 29 15:29:25 linux2 named[477]: Ready to answer queries.
```

Falls aus irgendwelchen Gründen der Nameserver nicht gestartet werden konnte, findet man hier entsprechende Fehlermeldungen. Durch `kill -HUP <pid>` kann man den `named` erneut laden (wenn man beispielsweise die Konfigurationsdateien geändert hat). `<pid>` ist dabei die Prozeß-ID, die man beispielsweise mit `ps ax |grep named` bekommt. Ob der DNS-Server nun auch wirklich funktioniert, kann man durch eine Anfrage (z.B. mit `nslookup` oder `dig`) überprüfen. Das Ergebnis sollte folgendermaßen aussehen:

```
root@linux2:/ >nslookup 192.168.0.1
Server:  linux2.fopra.edu
```

```
Address: 192.168.0.2

Name:    linux1.fopra.edu
Address: 192.168.0.1
```

Der DNS-Server sollte nun richtig installiert und konfiguriert sein. Im folgenden wird nun beschrieben, wie man Schlüssel erstellt und die Einträge in der DNS-Datenbank signiert.

4.3. Zonenschlüsselerstellung

Zur Erstellung eines Schlüsselpaares (private key/public key) wurde das Programm `dnskeygen` verwendet, das in der BIND-Distribution enthalten ist. Dieses Programm ist speziell zur Schlüsselerstellung für DNS geeignet, da es eine Datei erzeugt, die für den DNS-Server das passende Format hat. Zunächst wollen wir einen Schlüssel für die Zone erstellen, der dazu verwendet wird, später die einzelnen Einträge, in unserem Fall die KEY Resource Records, zu verifizieren und signieren. Durch Aufruf des Programms mit den folgenden Parametern, wird ein 768 Bit DSS/DSA-Zonenschlüssel für die Zone *fopra.edu* erzeugt. Welche Schlüsselarten sonst noch erstellt werden können, kann der Tabelle 2 entnommen werden.

```
dnskeygen –D 768 –z –c –n fopra.edu.
```

Durch Aufruf von `dnskeygen` ohne Parameter erhält man einen Hilfebildschirm, in dem die einzelnen Parameter beschrieben werden.

`–D 768`	DSS/DSA-Key wird erstellt (Schlüssellänge: 768 Bytes). Alternativ hierzu könnten folgende Parameter angegeben werden: -H generiert einen HMAC-MD5 Schlüssel (Länge zwischen [1..512]) -R generiert einen RSA Schlüssel (Länge zwischen [512..4096])
`–z`	ein Zonen-Schlüssel soll erzeugt werden
`–c`	Schlüssel kann nicht zum Verschlüsseln verwendet werden (er soll nur andere Einträge signieren können)
`–n fopra.edu.`	Name des Eigentümers des Schlüssels (in unserem Fall die Zone fopra.edu)

Je nach Algorithmus und angegebener Schlüssellänge sollten sich im aktuellen Verzeichnis nun zwei neue Dateien befinden. Die eine enthält den privaten Schlüssel (`Kfopra.edu.+003+03740.private`) und die andere den öffentlichen Schlüssel (`Kfopra.edu.+003+03740.key`). Die Zugriffsrechte werden von `dnskeygen` automatisch gesetzt. Danach haben die beiden Schlüssel folgende Zugriffsrechte:

```
-rw-r--r--1 root root 441 Nov  3 20:14 Kfopra.edu.+003+03740.key
-rw-------1 root root 556 Nov  3 20:14 Kfopra.edu.+003+03740.private
```

Öffentliche Schlüssel haben immer den Suffix `.key`, private Schlüssel immer `.privat`. Die zwei Zahlen in dem Dateinamen haben folgenden Bedeutung: die erste gibt den verwendeten Schlüssel an (in unserem Fall 003 für DSA/DSS, vgl. Tabelle 2), die zweite Zahl ist ein eindeutiger Identifier, der nur einmal vergeben wird, damit ein Schlüsselpaar auf Dateiebene eindeutig identifiziert werden kann. Dieser Identifier wird von `dnskeygen` und später von `dnssigner` (siehe „Signieren der Zone") benötigt (in unserem Beispiel 03740).

Die erstellten Dateien haben folgenden Inhalt (der Schlüssel wird hier, wie in [RFC 2535] gefordert im Base 64 Format dargestellt)

```
>more Kfopra.edu.+003+03740.key
fopra.edu. IN KEY 16641 3 3
BK7IkZWgYXmdncj0w5KnUsVOdTlLmXo8ilmOmkiOEIci+gCDN9A+9GseeYTE0
a/6uEfWlzIvItQTxQHEx0cN3FSwalfRnq/dpj9anehihMu1erakGE3ij7ao/K
RFNQmaMwx5gKqlfzRvrlhxwycnvSNbyCQ7A66AmvSb6OSULIdqIVpoPJSupLx
8dA3vLCTk1JwVzAaZW1ZRZMhLMmNG0YtMsVsMuXi0h1I+iYRsZGl11dt1pTPe
LqpFvFQwKsV3KTlQATOJB3nAUJN3RsuutJt3RouXIgM5UMW+Q9cDijMQjnLne
PpC5wmyDkLgeSP+31QqEAFqwldazbBLYdRxmqX8R0krLyuG9d70ilvNNGfW3O
fbluW/RIUWm+vSCZu8s//QpOZlGhMQwQLINSSMVKPXOaXj
```

öffentlicher Zonenschlüssel

```
>more Kfopra.edu.+003+03740.private
Private-key-format: v1.2
Algorithm: 3 (DSA)
Prime(p):
mXo8ilmOmkiOEIci+gCDN9A+9GseeYTE0a/6uEfWlzIvItQTxQHEx0cN3FSwa
lfRnq/dpj9anehihMu1erakGE3ij7ao/KRFNQmaMwx5gKqlfzRvrlhxwycnvS
NbyCQ7
Subprime(q): rsiRlaBheZ2dyPTDkqdSxU51OUs=
Base(g):
A66AmvSb6OSULIdqIVpoPJSupLx8dA3vLCTk1JwVzAaZW1ZRZMhLMmNG0YtMs
```

```
VsMuXi0h1I+iYRsZGl11dt1pTPeLqpFvFQwKsV3KTlQATOJB3nAUJN3RsuutJ
t3RouX
Private_value(x): CMxaZPaZcPz58EdfeCL0Kpur/ts=
Public_value(y):
IgM5UMW+Q9cDijMQjnLnePpC5wmyDkLgeSP+31QqEAFqwldazbBLYdRxmqX8R
0krLyuG9d70ilvNNGfW3OfbluW/RIUWm+vSCZu8s//QpOZlGhMQwQLINSSMVK
PXOaXj
```

In der Datei, in der der private Schlüssel abgelegt ist, sind ebenfalls die zur Schlüsselerzeugung verwendeten Primzahlen (Prime/Subprime) abgespeichert.

Will man bereits vorhandene Schlüssel in die DNS-Datenbank integrieren, muss der Schlüssel in das entsprechende Format gebracht werden (siehe oben). Dabei ist zu beachten, dass man das Algorithmus-Flag (vgl. Tabelle 2), das Schlüsselart-Flag (z.B. Userschlüssel, Zonenschlüssel,...(vgl. Abbildung 1)) und die Berechtigungen (z.B. darf signieren,...) richtig setzt. S/MIME Version 2/3 Zertifikate sowie PGP Zertifikate können über den CERT Resource Record in den Domain Name Service integriert werden. Siehe dazu CERT Resource Record.

4.4. Schlüssel in Datenbank-File integrieren

Der Private Key wird nun verwendet, um die Einträge in dem Datenbankfile der Zone *fopra.edu* (`db.fopra`) zu signieren. Danach wird dieser Schlüssel nicht mehr verwendet, und sollte dann auch nicht weiter auf einem Rechner mit Netzanschluß aufbewahrt werden. Die Public Key-Datei wurde schon in dem richtigen Format erstellt, um sie in das Master-Zone-File (`db.fopra`) einzubinden. Dazu muss man Ihn einfach an den Datenbankfile anhängen:

```
>cat Kfopra.edu.+003+03740.key >>db.fopra
```

Danach hat die Datei `db.fopra` folgenden Inhalt:

```
fopra.edu. IN SOA linux2.fopra.edu. root.linux2.fopra.edu. (
            5           ; Serial
            10800       ; Refresh after 3 hours
            3600        ; Retry after 1 hour
            604800      ; Expire after 1 week
            86400 )     ; Minimum TTL of 1 day

fopra.edu.              IN NS linux2.fopra.edu.
```

```
; Addresses for the canonical names
localhost.fopra.edu.    IN A  127.0.0.1
linux2.fopra.edu.       IN A  192.168.0.2
linux1.fopra.edu.       IN A  192.168.0.1

fopra.edu.              IN KEY 16641 3 3
BO8eP/4PqSsqqx4ZL8/yLlxhBf7ribX3SS1KXLY9Y7p1KiLwJ+QS0vv6yqeFc
VBPGr63/NFhGMlaTMKeZWD1pUpajq3xi6SJHqQb9CARQ6l6APIn+q6k58x/KW
i4fahR2t5dyC9Pz6sfxvsabGbJI3deRj0/DihfdfegbW2p4OTDrTEyy8abk3d
BACKNRMyBMT69qWVgIQt6SozXVJ+nmbf7fPZ1PHMzJefSfIyc8tBj3usuBrMn
jY9UklNqaLXOw42p0uLPZ77CaTimlNVyDC2F3zZXMpnnFQFPQIv2KvtvL6pg0
RrBo4J2QuKhbBIJkdliPD3W9tDW7eAt8Q8SoWm5oTmwbYn1eb9Km9iRVV0F7o
8bR55g56phVM6b5Z5T7dPjsttdvK9RAWl9xsHJKstIqitU
```

Der öffentliche Schlüssel muss deshalb im Datenbankfile vorhanden sein, damit später die signierten Einträge (z.B. linux1.fopra.edu), sofern sie nicht statisch im Resolver konfiguriert sind, verifiziert werden können. In der momentanen Konfiguration können aber noch keine Schlüssel verteilt werden, da noch keine weiteren Schlüsseleinträge (KEY RR) in der Datenbank vorhanden sind. Ein Zonenschlüssel muss jedoch immer vorhanden sein (entweder in der DNS-Datenbank oder im Resolver), da später die Signaturen der KEY RR‘s durch den Zonenschlüssel verifiziert werden müssen. Im nächsten Schritt werden wir nun die Zone signieren.

4.5. Signieren der Zone

Um Zonen zu signieren wird das Programm `dnssigner` verwendet, das in dem Paket `bind–contrib_tar.tar` enthalten ist. Dazu rufen wir das Programm mit folgenden Parametern auf:

```
>dnssigner –zi db.fopra –k1 fopra.edu 3 3740 –zo zone.2
```

wobei die einzelnen Parameter folgende Bedeutung haben (Informationen zu allen Parametern erhält man durch `dnssigner –h`):

`–zi db.fopra`	Input Zonen-File
`–k1`	Schlüssel zum initialisieren (Algorithmus, Identifier)
`–zo`	Output Zonen-File

Das Ergebnis wurde in der Datei zone.2 gespeichert und sieht folgendermaßen aus:

```
>cat zone.2

; Generated by dns_signer dated April 8, 1999
$ORIGIN fopra.edu.
fopra.edu.      86400 IN    SOA   linux2.fopra.edu. root.linux2.fopra.edu. (
                               5  ; serial
                               3H  ; refresh
                               1H  ; retry
                               1W  ; expiry
                               1D )  ; minimum
                86400 IN    SIG   SOA 3 2 86400 19991204200319
                            19991103200319 3740 fopra.edu. (
                   BFJgMysFRKKEgn2caiKNaPST8LOAPJqcwRGH9zAutlO4+cFl
                   Av7QaF0= )
fopra.edu.      86400 IN    KEY   0x4101 3 3 (
                   BO8eP/4PqSsqqx4ZL8/yLlxhBf7ribX3SS1KXLY9Y7p1KiLw
                   J+QS0vv6yqeFcVBPGr63/NFhGMlaTMKeZWD1pUpajq3xi6SJ
                   HqQb9CARQ6l6APIn+q6k58x/KWi4fahR2t5dyC9Pz6sfxvsa
                   bGbJI3deRj0/DihfdfegbW2p4OTDrTEyy8abk3dBACKNRMyB
                   MT69qWVgIQt6SozXVJ+nmbf7fPZ1PHMzJefSfIyc8tBj3usu
                   BrMnjY9UklNqaLXOw42p0uLPZ77CaTimlNVyDC2F3zZXMpnn
                   FQFPQIv2KvtvL6pg0RrBo4J2QuKhbBIJkdliPD3W9tDW7eAt
                   8Q8SoWm5oTmwbYn1eb9Km9iRVV0F7o8bR55g56phVM6b5Z5T
                   7dPjsttdvK9RAWl9xsHJKstIqitU )
fopra.edu.      86400 IN    NS    linux2.fopra.edu.
                86400 IN    SIG   NS 3 2 86400 19991204200319 19991103200319
3740 fopra.edu. (
                   BCmTTHBOUN24Gzn8XDSyR3wA6xqk4c49tKvf6hSy/7hES9dY
                   DVG0LhQ= )
fopra.edu.      86400 IN    NXT   linux1.fopra.edu. NS SOA SIG KEY NXT
                86400 IN    SIG   NXT 3 2 86400 19991204200319
19991103200319 3740 fopra.edu. (
                   BBfhpiGUWxRHXnKHyY5W3MEVR1LnR/Kxv9hntYuHeLxFZ6yX
                   YK68oaE= )
linux1          86400 IN    A     192.168.0.1
                86400 IN    SIG   A 3 3 86400 19991204200319 19991103200319
3740 fopra.edu. (
                   BLPljMm023TCrS5kMGt0xesX+Gw1nq61mZk06CpV07t438ml
                   C54lHqc= )
linux1          86400 IN    NXT   linux2.fopra.edu. A SIG NXT
                86400 IN    SIG   NXT 3 3 86400 19991204200319
19991103200319 3740 fopra.edu. (
                   BCqhpUNjHAvHh5ORPJTIdODBjj7vlMEIjPZNpUE+8wJZ6c+Z
                   C+jh16w= )
```

```
linux2          86400 IN    A     192.168.0.2
                86400 IN    SIG   A 3 3 86400 19991204200319 19991103200319
3740 fopra.edu. (
                   BI/BMqFjxcZ0c3InQmRQRwAR/D/sO5B/cpOnEoN4/amcx3Ax
                   De6RWHQ= )
linux2          86400 IN    NXT   localhost.fopra.edu. A SIG NXT
                86400 IN    SIG   NXT 3 3 86400 19991204200319
19991103200319 3740 fopra.edu. (
                   BLSJ82A2kjk6PIWYe5x6X3DcKsRxhKfYybUWoR7nyZOanP8b
                   4jwHbtQ= )
localhost       86400 IN    A     127.0.0.1
                86400 IN    SIG   A 3 3 86400 19991204200319 19991103200319
3740 fopra.edu. (
                   BKXgQvYcRK+JUgOM0FnB4mZm/ZT7Mtg1FNBMPQxElKWuDm9a
                   00GGQK4= )
localhost       86400 IN    NXT   fopra.edu. A SIG NXT
                86400 IN    SIG   NXT 3 3 86400 19991204200319
19991103200319 3740 fopra.edu. (
                   BCZeEUUTLGpl5tiY7dlXnNwkEXWVAtFHujIAmrv1wA5PKmjq
                   GaOq9Uo= )
```

Das Programm `dnssigner` fügt jedem Eintrag automatisch ein SIG RR hinzu, der die entsprechenden Einträge signiert. Des weiteren werden automatisch die NXT RRs eingefügt (vgl. Authentifizierung der Datenherkunft). Anzumerken ist, dass der `dnssigner` die Kurzschreibweise verwendet. Diese kann in Bind Versionen ab 8.0, durch Angabe von `$origin` in der ersten Zeile, verwendet werden, um nicht immer die Domäne mit angeben zu müssen (z.B. linux1 statt linux1.fopra.edu).
Im nächsten Schritt werden wir nun zeigen, wie man Schlüssel für Benutzer oder Hosts in die DNS-Datenbank einfügt und sie abfragen kann.

4.6. Erstellung von Benutzer/Host-Schlüsseln

Im Prinzip funktioniert das Erstellen von Schlüsseln für Benutzer genauso wie das Erstellen des Zonen-Schlüssels. Es ist zu beachten, dass nach Einfügen der Public-Keys das Zonen-File wieder mit `dnssigner` signiert wird. Beim Signieren wird natürlich wieder der private Schlüssel der Zone verwendet, den wir im vorhergehenden Schritt erzeugt haben. Zu jedem KEY RR wird dann wieder ein SIG RR erstellt. Wenn ein Client/Resolver einen KEY RR anfordert, bekommt er auch den dazugehörigen SIG RR, den er anhand des Zonenschlüssels, der ebenfalls in der DNS-Datenbank abgelegt oder statisch konfiguriert ist, verifiziert. Der verwendetet Algorithmus bei der Erstellung von Benutzerschlüsseln kann sich von dem des Zoneneschlüssels unterscheiden. Im folgenden erstellen wir für 2

Benutzer (User1 und User2) einen 512 Bit RSA/MD5 Schlüssel. In der Praxis sollte zur Zeit die Schlüssellänge bei RSA/MD5 mindestens 768 Bit betragen [RFC 2541].

```
>dnskeygen –R 512 –u –n user1.fopra.edu
Generating 512 bit RSA Key for user1.fopra.edu.

Generated 512 bit Key for user1.fopra.edu. id=43835 alg=1
flags=0

>dnskeygen –R 512 –u –n user2.fopra.edu
Generating 512 bit RSA Key for user2.fopra.edu.

Generated 512 bit Key for user2.fopra.edu. id=39715 alg=1
flags=0
```

Das Ergebnis sind jetzt wieder jeweils zwei Dateien, die den privaten und den öffentlichen Schlüssel enthalten:

```
>cat Kuser1.fopra.edu.+001+43835.key:
user1.fopra.edu.  IN    KEY   0     2     1     (AQPgeN....)

Kuser1.fopra.edu.+001+43835.private

>cat Kuser2.fopra.edu.+001+46066.key
user1.fopra.edu.  IN    KEY   0     2     1     (AQPFcZ...)

Kuser2.fopra.edu.+001+46066.private
```

Anschließend hängen wir die zwei neuen öffentlichen Schlüssel an unseren Zonenfile an:

```
>cat Kuser1.fopra.edu.+001+43835.key
      Kuser2.fopra.edu.+001+46066.key >>db.fopra
```

Zuletzt müssen wir das komplette File signieren, damit auch die neu dazugekommenen Schlüssel signiert werden, damit einen Manipulation nicht mehr möglich ist. Es ist darauf zu achten, dass bei der Signierung der Zonenschlüssel verwendet werden muss, den wir davor erzeugt haben. Eine Zone (in unserem Fall fopra.edu) muss immer mit dem gleichen privaten Schlüssel der Zone signiert werden, denn mit dem öffentlichen Zonenschlüssel werden die SIG-Resource Records verifiziert. Der private Schlüssel der Benutzer User1 und User2 wird vom DNS nicht mehr benötigt. Dieser Schlüssel muss den Benutzern ausgehändigt werden. Oft ist es leider nicht zu vermeiden, dass die Benutzer den privaten Schlüssel lokal auf ihrem Rechner halten müssen, damit eine transparente Ver-/Entschlüsselung stattfinden kann. Wie weiter oben bereits erwähnt könnte dieser Schlüssel jetzt zum Beispiel zum Versenden von Email verwendet werden.

Die Emailadressen wären dann laut DNSSec-Spezifikation user1@fopra.edu bzw. user2@fopra.edu. Die Zone wird nun folgendermaßen neu signiert.

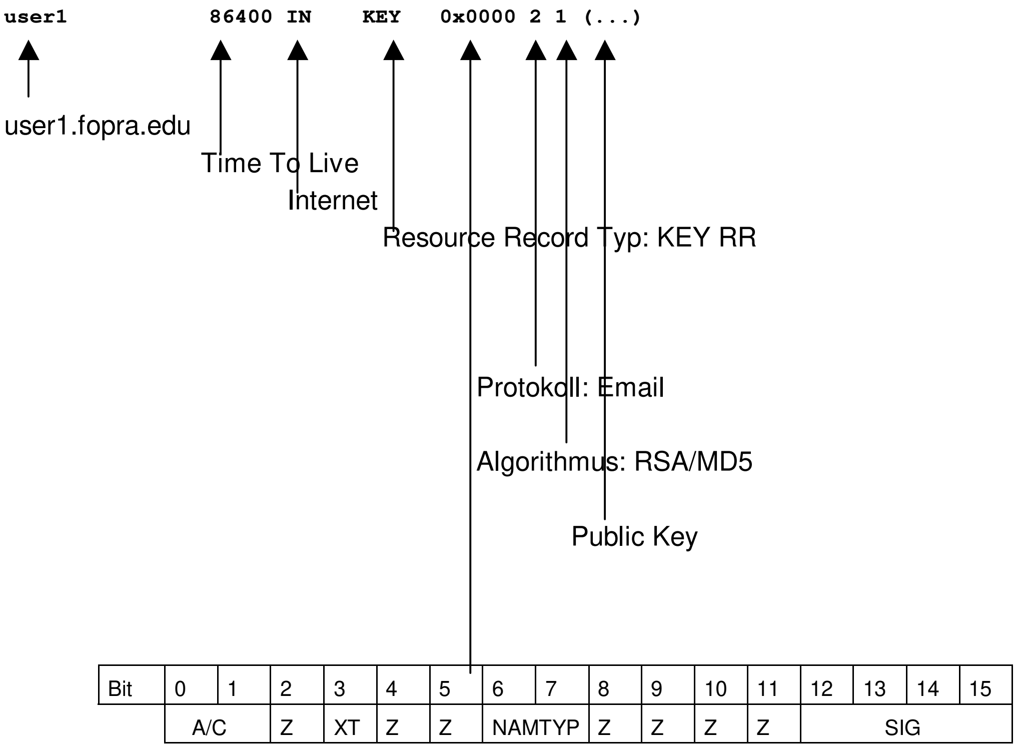
```
>dnssigner –zi db.fopra –k1 fopra.edu 3 3740 –zo db.fopra
```

Hier wird also wieder der gleiche Zonenschlüssel verwendet, wie bei der Signierung der Zone selbst. Es können natürlich auch beide Schritte in einem gemacht werden, d.h. man kann sowohl den Zonenschlüssel als auch die Benutzerschlüssel an die Zonendatei anhängen, und anschließend die komplette Datei signieren. In unserem Beispiel sollte die Zonendatei folgendermaßen aussehen (die Schlüsselwerte, bzw. die Signaturen wurden aus Gründen der Übersichtlichkeit durch „...“ ersetzt):

```
; Generated by dns_signer dated April 8, 1999
$ORIGIN fopra.edu.
fopra.edu.      86400 IN    SOA   linux2.fopra.edu. root.linux2.fopra.edu. (
                               5  ; serial
                               3H  ; refresh
                               1H  ; retry
                               1W  ; expiry
                               1D )  ; minimum
                86400 IN    SIG   SOA 1 2 86400 19991222152431
19991121152431 12406 fopra.edu. (...)

fopra.edu.      86400 IN    KEY   0x4101 3 3 (...)

fopra.edu.      86400 IN    NS    linux2.fopra.edu.
                86400 IN    SIG   NS 3 2 86400 19991222152431 19991121152431
                                                    12406 fopra.edu. (...)

fopra.edu.      86400 IN    NXT   linux1.fopra.edu. NS SOA SIG KEY NXT
                86400 IN    SIG   NXT 3 2 86400 19991222152431
                                        19991121152431 12406 fopra.edu. (...)

linux1          86400 IN    A     192.168.0.1
                86400 IN    SIG   A 3 3 86400 19991222152431 19991121152431
                                                    12406 fopra.edu. (...)
linux1          86400 IN    NXT   linux2.fopra.edu. A SIG NXT
                86400 IN    SIG   NXT 3 3 86400 19991222152431
                                     19991121152431 12406 fopra.edu. (...)
```

```
linux2          86400 IN    A     192.168.0.2
                86400 IN    SIG   A 3 3 86400 19991222152431 19991121152431
                                            12406 fopra.edu. (...)
linux2          86400 IN    NXT   localhost.fopra.edu. A SIG NXT
                86400 IN    SIG   NXT 3 3 86400 19991222152431
                                     19991121152431 12406 fopra.edu. (...)

localhost       86400 IN    A     127.0.0.1
                86400 IN    SIG   A 1 3 86400 19991222152431 19991121152431
                                            12406 fopra.edu. (...)

localhost       86400 IN    NXT   user1.fopra.edu. A SIG NXT
                86400 IN    SIG   NXT 3 3 86400 19991222152431
                                     19991121152431 12406 fopra.edu. (...)

user1           86400 IN    KEY   0x0000 2 1 (...)
                86400 IN    SIG   KEY 3 3 86400 19991222152431
                              19991121152431 12406 fopra.edu. (...)
user1           86400 IN    NXT   user2.fopra.edu. SIG KEY NXT
                86400 IN    SIG   NXT 3 3 86400 19991222152431
                              19991121152431 12406 fopra.edu. (...)

user2           86400 IN    KEY   0x0000 2 1 (...)
                86400 IN    SIG   KEY 3 3 86400 19991222152431
                                     19991121152431 12406 fopra.edu. (...)
user2           86400 IN    NXT   fopra.edu. SIG KEY NXT
                86400 IN    SIG   NXT 3 3 86400 19991222152431
                                     19991121152431 12406 fopra.edu. (...)
```

Erklärung eines KEY Resource Records:


user1 86400 IN KEY 0x0000 2 1 (...)
user1.fopra.edu
Time To Live
Internet
Resource Record Typ: KEY RR
Protokoll: Email
Algorithmus: RSA/MD5
Public Key

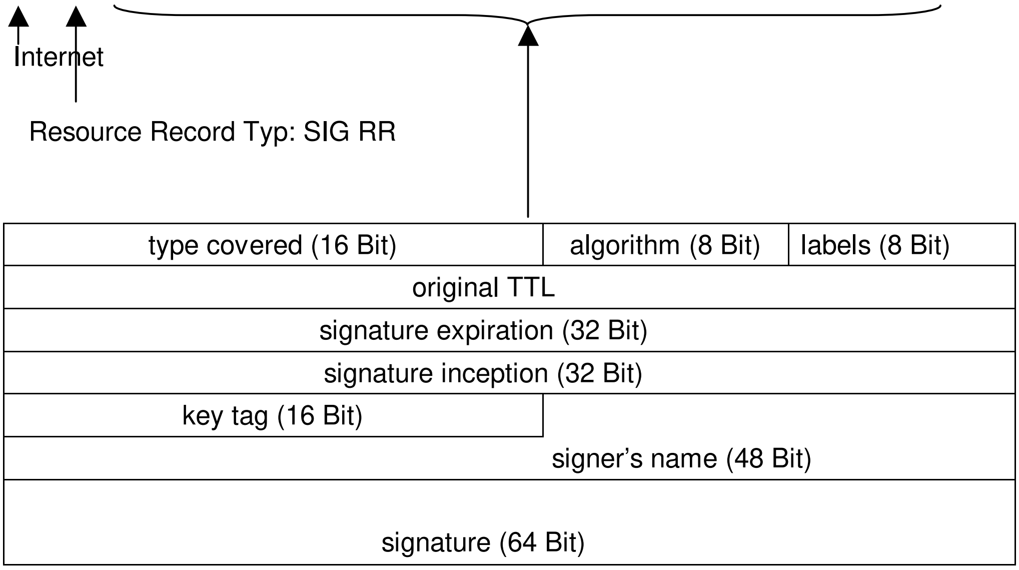
Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	A/C		Z	XT	Z	Z	NAMTYP		Z	Z	Z	Z	SIG			

Bit(s)	Wert im Beispiel	Bedeutung
A/C	00	Schlüssel kann zum Authentifizieren und Verschlüsseln verwendet werden.
XT	0	Es folgt kein weiteres Flag-Feld
NAMTYP	00	Schlüssel ist einem Benutzer oder Benutzer-Account zugeordnet.
SIG	0000	Schlüssel darf nicht verwendet werden, um andere Einträge zu signieren.
Z	0	Zero (immer Null)

Tabelle 6: Beispiel Key Resource Record

Erklärung eines SIG Resource Records:

```
IN SIG KEY 3 3 86400 19991222152431 19991121152431 12406 fopra.edu. (...)
```


Internet
Resource Record Typ: SIG RR
type covered (16 Bit)
algorithm (8 Bit)
labels (8 Bit)
original TTL
signature expiration (32 Bit)
signature inception (32 Bit)
key tag (16 Bit)
signer's name (48 Bit)
signature (64 Bit)

Bit(s)	Wert im Beispiel	Bedeutung
type covered	KEY	KEY RR wurde signiert.
algorithm	3	DSA (Typ Zonenschlüssel)
lables	3	user1 hat 3 Einträge (KEY, SIG, NXT)
original TTL	86400 (in s)	TTL = 1 Tag
signature expiration	19991222152431	Signatur gültig bis zum 22.12.1999 15:24:31 Uhr
signature inception	19991121152431	Signatur wurde am 21.11.1999 um 15:24:31 Uhr erstellt.
key tag	12406	Checksumme
signer's name	fopra.edu	Name des Signers (Zonenschlüssel).
signature	(...)	Signatur

Tabelle 7: Beispiel SIG Resource Record

Somit sind in der Datei nun alle Einträge signiert, und die Benutzerschlüssel können abgefragt werden. Um einen Benutzerschlüssel abzufragen, geht man analog zu ganz normalen DNS-Anfragen vor. Leider sind derzeit keine Implementierungen auf dem Markt, die diese Art der Schlüsselverteilung unterstützen. Abfragen mit `nslookup` sind zwar möglich (mit `type=key`), die SIG Resource Records werden allerdings nicht interpretiert, und somit können Schlüssel oder andere Einträge nicht verifiziert werden. Im Bind-Contribution Paket ist ein kleines C-Programm (`getkeyby`) enthalten, mit dem die Schlüssel, samt Algorithmus, Länge und weiteren Parametern aus dem DNS gelesen werden können.

5. DNSSec in Verbindung mit IPSec

Im folgenden soll untersucht werden, in wieweit DNSSec dazu geeignet ist, Schlüssel für IPSec zur Verfügung zu stellen. IPSec löst das Problem der Verwaltung und Verteilung der Schlüssel nicht selber, sondern benötigt ein weiteres Protokoll zum Schlüsselaustausch und zur Schlüsselverwaltung.

5.1. IPSec mit manual keying

Die einfachste Möglichkeit ist das ‚manual keying', bei dem auf jedem beteiligten Rechner der gleiche geheime Schlüssel in einer Datei abgelegt wird. Dies hat jedoch den großen Nachteil, dass ein Angreifer, der es schafft root-Rechte zu erlangen, die verschlüsselte IPSec-Verbindung abhören und entschlüsseln kann. Zudem ist es aufwendig Schlüssel zu erneuern, da auf jedem einzelnen Host ein neuer Schlüssel konfiguriert werden müsste.

5.2. IPSec mit automatic keying

Die zweite Möglichkeit ist das sogenannte ‚automatic keying', bei dem zwar nicht der geheime Schlüssel in einer Datei abgelegt wird, aber es werden Informationen gespeichert, die eine Aushandlung eines Sitzungsschlüssels bewerkstelligen (z.B. mit dem Diffie-Hellman Algorithmus). Ein Angreifer hat somit nicht den Schlüssel, der von IPSec zum Verschlüsseln der Verbindung verwendet wird, in der Hand. Ein Man-In-The-Middle-Angriff wäre aber bei dieser Lösung immer noch möglich. Zudem kann beim ‚automatic keying' nach einem vorgegebenen Intervall ein neuer Sitzungsschlüssel ausgehandelt werden, ohne dass die IPSec-Verbindung unterbrochen werden muß. Dies ist bei der ersten Lösungsmöglichkeit, dem ‚manual keying' nicht möglich.

Für das ‚automatic keying' gibt es unterschiedliche Protokolle von unterschiedlichen Herstellern. Zum Beispiel wurde von Sun Microsystems SKIP (Simple Key Interchange Protocol) entwickelt, das zur Aushandlung von Schlüsseln in VPNs (Virtual Privat Networks) verwendet wird. Ein weiteres bekanntes Protokoll, das derzeit von der IETF (Internet Engineering Task Force) empfohlen wird, ist ISAKMP (Internet Security Association and Key Management Protocol). Dieses dient zur Verwaltung von Sicherheitsassoziationen und zum Schlüsselaustausch. [Fama]

5.3. IPSec mit DNSSec

Die sinnvollste Lösungsmöglichkeit ist, vorhandene Public-Key Infrastrukturen zu verwenden, um einen Sitzungsschlüssel für IPSec auszuhandeln. Dies hat den Vorteil, dass keine gemeinsamen Geheimnisse (shared secrets) erforderlich sind. Wenn ein Host eine IPSec-Verbindung mit einem anderen Host aufbauen will, so erfragt er den öffentlichen Schlüssel des anderen Hosts beim DNS-Server und verwendet ihn beispielsweise dazu eine zufällige Zahl zu verschlüsseln. Nur der Zielhost kann mit seinem privaten Schlüssel die vorher bestimmte zufällige Zahl entschlüsseln. Somit ist nur der Quell- und der Ziel-Host im Besitz dieser Zahl, die als Grundlage zur Aushandlung eines Sitzungsschlüssels verwendet werden kann. Selbst einem Angreifer, der Root-Rechte auf dem Quell- oder auf dem Ziel-Host erlangen kann, ist es nicht möglich den geheimen Schlüssel, oder Informationen darüber, wie dieser geheime Schlüssel erzeugt worden ist, zu bekommen, da zu keinem Zeitpunkt Informationen über den Schlüssel, bzw. der Schlüssel selbst in einer Datei abgespeichert werden müssen. Zudem treten keine Probleme auf, falls ein Host seine IP-Adresse von einem DHCP-Server (Dynamic Host Configuration Protocol) bezieht, da in DNSSec auch Benutzereinträge vorhanden sein können (siehe Erstellen von Benutzer/Host-Schlüsseln). Probleme würden auftreten wenn man Sicherheitsassoziationen auf Basis von IP-Adressen bilden möchte.

5.4. Aktuelle Implemetierungen von IPSec/DNSSec

Eine bekannte Implementierung für UNIX Systeme von IPSec ist FreeS/WAN. In der aktuellen Version (1.3) wird das ‚manual keying‘ und das ‚automatic keying‘ mit Hilfe des Pluto Daemon und IKE (Internet Key Exchange) unterstützt. [Fama] FreeS/WAN kündigt zwar eine DNSSec-Unterstützung an, bislang wurde sie aber noch nicht implementiert. Auch andere kommerzielle Hersteller wie z.B. Cisco Systems bieten bei ihren IPSec-Produkten noch keine DNSSec-Unterstützung an. Oft verwenden kommerzielle Hersteller Ihre eigenen Protokolle.

6. Zusammenfassung

Wie im obigen Beispiel-Szenario beschrieben, ist es relativ einfach, ein Key Distribution Center mit BIND zu implementieren, vor allem, wenn bereits notwendige Infrastrukturen, also ein DNS-Server, bereits vorhanden sind. Abgesehen von der Verwendung von DNS als Key Distribution Center, ist auf jeden Fall in Erwägung zu ziehen, andere Erweiterungen von DNSSec zu nutzen, um den Domain Service, der von jedem genutzt wird, der mit Internet verbunden ist, sicherer zu gestalten. Bisher ist er ein beliebtes Angriffsziel von vielen Hackern. Inwieweit sich DNS/Bind als Key Distribution Center durchsetzen kann, bleibt abzuwarten, da es sicherlich noch andere Alternativen, wie z.B. OCSP (Online Certificate Status Protocol), LDAP Lightweight Directory Access Protocol),... gibt. Momentan werden die DNS Security Extensions auch nur von BIND unterstützt. Andere DNS-Server, wie z.B. Microsoft DNS-Server, unterstützen derartige Funktionen noch nicht, obwohl die meisten DNS-Server auf BIND aufsetzen. Es bleibt also abzuwarten in welche Richtung die Entwicklung geht, und welche Standards gesetzt werden. Mangelnde Implementierungen von Anwendungen verhindern zur Zeit den produktiven Einsatz von DNSSec/Bind in der Praxis, wenn man den Aufwand, eigene Applikationen zu implementieren, vermeiden will.

Literaturverzeichnis

[RFC 1034] Domain Names - Concepts And Facilities
[RFC 1035] Domain Names- Implementation And Specification
[RFC 1101] DNS encoding of network names and other types
[RFC 1170] Public key standards and licenses
[RFC 1183] New DNS RR Definitions
[RFC 1530] Principles of Operation for the TPC.INT Subdomain: General Principles and Policy
[RFC 1535] A Security Problem and Proposed Correction With Widely Deployed DNS Software
[RFC 1537] Common DNS Data File Configuration Errors
[RFC 1713] Tools for DNS debugging
[RFC 1750] Randomness Recommendations for Security
[RFC 1886] DNS Extensions to support IP version 6
[RFC 2065] Domain Name System Security Extensions
[RFC 2137] Secure Domain Name System Dynamic Update
[RFC 2181] Clarifications to the DNS Specification
[RFC 2230] Key Exchange Delegation Record for the DNS
[RFC 2268] A Description of the RC2(r) Encryption Algorithm
[RFC 2312] S/MIME Version 2 Certificate Handling
[RFC 2401] Security Architecture for the Internet Protocol
[RFC 2407] The Internet IP Security Domain of Interpretation for ISAKMP
[RFC 2408] Internet Security Association and Key Management Protocol (ISAKMP)
[RFC 2409] The Internet Key Exchange (IKE)
[RFC 2527] Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework
[RFC 2535] Domain Name System Security Extensions
[RFC 2536] DSA KEYs and SIGs in the Domain Name System (DNS)
[RFC 2537] RSA/MD5 KEYs and SIGs in the Domain Name System (DNS)
[RFC 2538] Storing Certificates in the Domain Name System (DNS)
[RFC 2539] Storage of Diffie-Hellman Keys in the Domain Name System (DNS)
[RFC 2541] DNS Security Operational Considerations
[RFC 2559] Operational Protocols - LDAPv2
[RFC 2560] PKIX Online Certificate Status Protocol (OCSP)
[RFC 2632] S/MIME Version 3 Certificate Handling

[draft-ietf-dnsind-verify-00.txt] Verifying Resource Records Without Knowing Their Contents
[draft-ietf-dnssec-indirect-key-01.txt] Indirect KEY RRs in the Domain Name System

[draft-ietf-dnssec-externalkeys-01.txt]	Inter-operability of Secure Domain Name System with Other Key Distribution Services
[draft-ietf-dnsind-keyreferral-00.txt]	The Zone Key Referral
[draft-ietf-dnsop-keyhand-01.txt]	Handling of DNS zone signing keys
[draft-ietf-dnsext-tkey-01.txt]	Secret Key Establishment for DNS (TKEY RR)
[draft-ietf-dnsext-tkey-01.txt]	Secret Key Transaction Authentication for DNS (TSIG)
[draft-ietf-dnsop-dnsseccairn-00.txt]	DNSSEC Implementation in the CAIRN Testbed

[draft-ietf-dnnsec-key-handling-00.txt]	Zone KEY RRSet Signing Procedure

[MartK] DNS gegen Mißbrauch schützen, Martius, Kai in iX2/1999 (S.108-113)

[DnsB] DNS and Bind, Albitz Paul, Cricket Liu, O'REILLY 1998 3rd edition

[LinSA] Linux, Systemadminitration, Hein Jochen, Addisson-Wesley 1999

[LinUR] Linux, Die User-Referenz, Kester Grelck et. al., mitp 1999

[Suse] SuSe Linux 6.1, Installation, Konfiguration, Erste Schritte. SuSe GmbH, 14. Auflage 1999

[Fama] IPSec Integration in einer Linux-Umgebung, Fortgeschrittenenpraktikum, Dietmar Fackelmann, Dezember 1999