Seminararbeit zum Thema „Netzwerksicherheit“

Vorgelegt von: Johannes Hellmich

Im Hauptseminar: Kryptologie

Leitung: Prof. Dr. Wätjen

---

5. Januar 2007

Inhaltsverzeichnis

1 IPSec

Das IPSec-Protokoll ermöglicht Nutzer- und Datenauthentizität, sowie Datenvertraulichkeit auf der Internetschicht. Damit können etwa zwei Rechner direkt ihre Kommunikation absichern, oder es können gesamte Netzwerke im VPN gekoppelt werden. Dieser Teil folgt im wesentlichen den Ausführungen in [9] und [8].

1.1 Security Association (SA)

Als übergeordnetes Ziel von IPSec kann der Aufbau sogenannter IPSec-Security Associations (im folgenden: IPSec-SA) erachtet werden. Aus virtueller Sicht handelt es sich hierbei um unidirektionale Kommunikationskanäle, die exklusiv einen der Services „Vertrauchlichkeit“ oder „Datenauthentizität“ sicherstellen. Die versendeten Daten werden innerhalb einer IPSec-SA also entweder verschlüsselt oder authentifiziert. In diesem Sinne könnte eine bidirektionale Kommunikationsverbindung, die Vertraulichkeit und Datenauthentizität gewährleistet, mit genau 4 IPSec-SAs realisiert werden.

Aus virtueller Sicht lassen sich IPSec-SAs also als Kommunikationskanäle interpretieren. Konkreter haben wir es hier mit Vereinbarungen zu tun, die auf beiden Seiten gewisse Parameter festlegen. So sind innerhalb einer IPSec-SA die verwendeten Identifikationsverfahren der Nutzer, der IPSec-Verschlüsselungsalgorithmus, die IP-Adressen der Parteien und die Gültigkeitsdauern von Authentisierung und Schlüsseln festgelegt. Erstere erfolgt dabei mittels Internet Key Exchange Protocol (IKE) und besteht aus einer der drei Alternativen PSK, Public-Key Encryption oder Signaturverfahren, die im entsprechenden Unterkapitel genau erklärt werden. Als Verschlüsselungsalgorithmus muss zwingenderweise in jeder IPSec-Implementierung die Blockchiffre DES zu Verfügung gestellt werden. Optional findet man meist den 3-DES (Triple-DES) und BLOWFISH.

Da bei jedem Kommunikationsteilnehmer offensichtlich auch mehrere parallele IPSec-SAs bestehen können müssen, ist hier eine Variable notwendig, um IPSec-SAs eindeutig zu identifizieren. Genauer identifiziert jeder Nutzer eine konkrete IPSec-SA durch das Tripel <destination IP, SPI, security protocol>.
Der erste Wert zeigt dabei die Empfängeradresse an und der letzte das verwendete Sicherheitsprotokoll (AH oder ESP), womit auch der Sicherheitsservice dieser IPSec-SA festegelegt ist (Vertraulichkeit oder Datenauthentizität). Da diese Werte aber nicht zwingenderweise eindeutig sind, existiert der Security Parameter Index (SPI), der jede IPSec-SA bei einem Nutzer eindeutig identifiziert. Mittels dieses Wertes können bei mehreren parallelen IPSec-SAs die Pakete also eindeutig einem Kommunikationskanal zugeordnet werden.

Von der IPSec-SA, die einen Service unidirektional realisiert, muss man allerdings die ISAKMP-SA abgrenzen. Diese dient als übergeordneter Kanal zur Aushandlung einer oder mehrerer IPSec-SAs und arbeitet bidirektional mit gleichzeitiger Sicherung von Vertraulichkeit und Authentizität. Diese ISAKMP-SA ist Teil des IKE und wird im nächsten Kapitel ausführlich erklärt.

Sehr kritisch muss die Reihenfolge der Anwendung der einzelnen Transformationen,

die durch IPSec vorgegeben ist, betrachtet werden. Seien SA_{AH} der authentizitätssichernde und SA_{ESP} der vertrauchlichkeitssichernde Kanal. Dann fordert das Protokoll für jedes Paket P, das verschlüsselt und authentifiziert werden soll, den Zustand $SA_{AH}(SA_{ESP}(P))$. Damit wird darauf abgezielt, dass zerstörte oder gefälschte Pakete bereits an der Authentifizierung scheitern und die Rechenzeit für die Entschlüsselung gespart werden kann. Hiermit sollen vermutlich (leider findet sich in den IP-Sec-RFCs keine Erklärung, warum diese Reihenfolge benutzt werden muss) auch DOS-Attacken vorgebeugt werden, da die eingesparte Entschlüsselung durchaus als wesentlich rechenintensiver als die Authentizitätsprüfung angesehen werden kann.

Das dadurch entstehende Problem ist aber leider als schwerwiegender einzuschätzen, als eine mögliche DOS-Attacke auf einen einzelnen Rechner. Es wird jetzt nämlich nicht mehr das authentifiziert, was wirklich ausgesagt werden soll, sondern ein Chiffretext, dessen Bedeutung zufällig erscheint. Ein Grundsatz der Authentifizierung von Nachrichten ist es, dass entweder der Klartext oder der Chiffretext inklusive Schlüssel authentifiziert werden. In jedem Fall hat man damit das, was ausgesagt werden soll gesichert.
Ein entsprechender Angriff bei Nichtbeachtung dieses Grundsatzes findet sich in [3] auf Seite 9. Hier erlangt ein Angreifer tatsächlich die Möglichkeit, einer Partei authentische Daten unterzuschieben, die aber mit dem falschen Schlüssel entschlüsselt werden. Damit werden Zufallsdaten, die allerdings auch der Angreifer nicht voraussagen kann und die als fehlerhaft angenommen werden können, an höhere Schichten weitergegeben.

1.2 Das Internet Key Exchange Protocol - IKE

1.2.1 Grundlagen und Ziele von IKE

IPSec schreibt zur automatischen Authentifizierung der Nutzer und zum automatischen Schlüsselaustausch die Benutzung des Internet Key Exchange Protocols (IKE, siehe [4]) vor. Alternativ dazu können diese Aufgaben natürlich auch manuell durch einen Administrator ausgeführt werden, was aber sicher nicht skaliert und nur in kleinen Netzwerken mit fester Teilnehmerzahl überhaupt in Erwägung gezogen werden sollte.

Genauer beschreibt IKE *wie* Schlüssel ausgetauscht werden. Es wird also ein bestimmtes Paketformat definiert und festgelegt, in welcher Reihenfolge die entsprechenden Nachrichten verschickt werden müssen. Die Festlegung der Inhalte der Pakete (also *was* ausgetauscht werden muss) erfolgt durch IKE nur in abstrakter Form. Für eine genaue Definition wird die Benutzung eines DOI-Dokuments (Domain Of Interpretation) vorgeschrieben. In einem solchen kann beispielsweise die Menge der möglichen Chiffriertransformationen codiert sein.

Ferner stellt IKE die Zusammenfassung zweier Protokolle dar. Das ISAKMP-Protokoll (Internet Security Association and Key Management Protocol, siehe [5]) dient der Definition der Paketformate und legt die einzuhaltende Sequenz bei der Versendung dieser fest. Das OAKLEY Protokoll bestimmt, wie genau ein symmetrischer Schlüssel aus den ausgetauschten Komponenten berechnet werden kann. Hier kann etwa das Diffie-Hellmann-Verfahren (beschrieben in [2], Seite 134) herangezogen werden.

IKE ist in zwei Phasen aufgeteilt. In der ersten Phase authentifizieren sich die Kommunikationspartner und bauen eine übergeornete SA, die sogenannte ISAKMP-SA, auf. Diese sichert Authentizität und Vertraulichkeit in beide Richtungen und wird dazu genutzt, eine oder mehrere IPSec-SAs auf sicherem Wege auszuhandeln. Ausserdem ist mit der ISAKMP-SA auch ein Masterschlüssel $SKEYID$ assoziiert. Aus diesem werden die Unterschlüssel $SKEYID_d, SKEYID_e$ und $SKEYID_a$ abgeleitet und diese sichern in Phase2 die Vertraulichkeit und Authentizität der ISAKMP-Nachrichten. Phase2 ist also komplett verschlüsselt ($SKEYID_e$) und alle Nachrichten sind authentifiziert ($SKEYID_a$). Neben der Berechnung von $SKEYID_e$ und $SKEYID_a$ wird $SKEYID_d$ auch zur Berechnung symmetrischer Schlüssel für IPSec-SAs herangezogen. Damit aber die Kompromittierung von $SKEYID_d$ nicht auch zur Kompromittierung aller Schlüssel, die für die IPSec-SAs hieraus abgeleitet werden, führt, kann optional ein weiterer Diffie-Hellmann Schlüssel bei der Ableitung der IPSec-Schlüssel einberechnet werden. In diesem Sinne folgt Perfect Forward Security (wie in [2] beschrieben) für eine IPSec-SA.

Für die genaue Berechnung der Unterschlüssel ist die Verwendung eines sogenannten *keyed pseudorandom bitgenerators* vorgeschrieben. Dabei wird allerdings nur das Interface $PBG(keyinput, messageinput)$ und die Tatsache, dass die Ausgabewerte als zufällig erscheinen sollen, festgelegt. Eine entsprechende Implementierungsmöglichkeit findet sich in [4].
Die konkreten Berechnungen folgen dann mit:

- $SKEYID_d = PBG(SKEYID, g^{x_i x_r} | CKY_i | CKY_r | 0)$
- $SKEYID_a = PBG(SKEYID, SKEYID_d | g^{x_i x_r} | CKY_i | CKY_r | 1)$
- $SKEYID_e = PBG(SKEYID, SKEYID_a | g^{x_i x_r} | CKY_i | CKY_r | 2)$

Die Indizes i und r zeigen an, ob es sich um den Wert vom „Initiator“ der Verbindung (i) oder vom „Responder“ (r) handelt. $g^{x_i x_r}$ ist ein Diffie-Hellmann Schlüssel, dessen Zustandekommen erst in den Unterkapiteln zu den verschiedenen Authentifizierungsmöglichkeiten von Phase1 erklärt werden kann. $CKY_z, z \in \{i, r\}$ beschreibt ein „Cookie“. Das sind die Verbindungsdaten für die Partei z, etwa IP Adresse und Port, über den kommuniziert wird. Die Werte $0, 1$ und 2 am Ende sind zu interpretieren als Oktette.

1.2.2 Das ISAKMP Paketformat

Auf ISAKMP (siehe [5]) als Teil des IKE wurde schon im vorherigen Kapitel eingegangen. Es definiert Paketformat und Sequenz der auszutauschenden Nachrichten, um zwei Nutzer zu authentifizieren und einen symmetrischen Schlüssel zwischen diesen zu vereinbaren. Wie Authentifizierung und Schlüsselaustausch in IKE mit ISAKMP erfolgen, wird in den folgenden Kapiteln erklärt. Hier soll zunächst das ISAKMP-Paketformat beschrieben werden. Nachrichten von dieser Form werden also im IKE ausgetauscht.

Der Header eines jeden ISAKMP-Paketes besteht aus:

- 64 Bit `Initiator-` und `Responder-Cookie`:
  Hier werden die Verbindungsdaten der beiden Parteien in einem String festgehalten. Diese dienen auch dazu, dass mit einem Kommunikationspartner nur genau eine `ISAKMP-SA` aufgebaut werden kann, denn die Pakete werden eindeutig als zugehörig zum Absender identifiziert. Damit wird einer `DOS`-Attacke vorgebeugt, bei der ein einzelner, feindlicher Initiator so lange `ISAKMP-SAs` erzeugen könnte, bis der Antwortende keine weiteren `ISAKMP-SAs` mehr mit anderen Parteien aufbauen kann.

- 8 Bit `NextPayload`:
  Dieses Feld findet sich auch im generischen Payload-Header eines jeden Payloadfeldes und es verleiht dem gesamten Paket damit eine dynamische Struktur im Sinne einer verketteten Liste. Es wird also so lang die Art der nächsten Nutzdaten angezeigt, bis durch ein entsprechendes Oktett aus Nullen das Ende folgt. Angegeben werden kann hier beispielweise der Wert `KE`, der ein `KeyEnvironment`, also die Diffie-Hellmann-Parameter, anzeigt.

- 8 Bit `Version`:
  Die verwendete Haupt- und Unterversion des `ISAKMP`-Protokolls.

- 8 Bit `Exchange`:
  Alle Authentifizierungsverfahren lassen sich im `main-` oder `aggressive-mode` betreiben, wie in den entpsrechenden Kapiteln gezeigt wird. Hier wird der verwendete Modus angegeben.

- 8 Bit `Flags`:
  Hiermit kann angegeben werden, ob die Nutzdaten verschlüsselt, oder gegebenenfalls nur authentifiziert sind.

- 32 Bit `Message-ID`:
  Dient der eindeutigen Zuordnung dieses Paketes zu einer `IPSec-SA`. (Wichtig in Phase2, wo auch mehrere `IPSec-SAs` mit `ISAKMP` parallel ausgehandelt werden können.)

- 32 Bit `Length`:
  Länge der gesamten Nachricht mit Header und allen Payloads in Byte.

Wie schon angedeutet, kann ein `ISAKMP`-Paket variabel viele Nutzdaten (`Payloads`) enthalten. Ein jedes dieser Felder ist dabei mit einem eigenen, generischen `Payload-Header` ausgestattet:

- 8 Bit `Next Payload`:
  Wie im `ISAKMP`-Header wird hier die Art der Nutzdaten im nächsten `Payload`-Feld angegeben.

- 8 Bit `Reserved`:
  Dieses Feld ist für Erweiterungen reserviert.

- 16 Bit `Payload-Length`:
  Hier steht die Länge der in diesem Abschnitt codierten Nutzdaten.

1.2.3 IKE-Phase1 mit PSK Authentifizierung

Im folgenden wird die Authentifizierung mit dem `Pre-Shared-Key`-Verfahren in `IKE` erklärt. Die anderen Auhentifizierungsverfahren lassen sich gut hieraus ableiten, weswegen die hier gemachten Definitionen auch für die beiden Folgekapitel gelten. Diese Kapitel sind entsprechend kurz gehalten, da die beschriebenen Verfahren als Spezialformen des `PSK`-Verfahrens betrachtet werden können.

Beim `PSK` wird jeder Teilnehmer als authentisch angenommen, der einen bestimmten, vorher verteilten Schlüssel (den `Pre-Shared-Key`) kennt. Es ist sofort klar, dass das Verfahren wegen der aktiven Schlüsselverteilung nicht skalieren kann und deswegen nur in kleinen Netzwerken mit fester Teilnehmerzahl praktikabel ist. Ein weiterer Nachteil ist, dass die Kompromittierung des `PSKs` dazu führen muss, dass dieser bei allen Teilnehmern angezweifelt und ausgetauscht werden muss.

Innerhalb des Verfahrens werden bestimmte `ISAKMP`-Nachrichten ausgetauscht, wobei die `NextPayload`-Felder zunächst genau erklärt werden müssen:

- $\mathbb{SA}$: Das „Proposal“, eine Menge von `Security Association`-Parametern, die der Initiator dem Responder vorschlägt. Darin enthalten sind alle Verschlüsselungs- und Hashfunktionen, die er unterstützt und als sicher erachtet.
- $SA!$: Eine konkrete Auswahl aus $\mathbb{SA}$, die der Responder trifft. Typischerweise wählt er die Funktionen aus, die er als sicher erachtet und unterstützt.
- KE: Das `Key-Exchange-Payload`. Hiermit werden die Diffie-Hellmann Komponenten, also Primzahl, Generator und öffentlicher Schlüssel, übermittelt.
  Nach Versendung dieser Pakete kennen also beide Nutzer g^{x_i} und g^{x_r}, was für die Berechnung von $H_z, z \in \{i, r\}$ wichtig sein wird.
- $N_z, z \in \{i, r\}$: Das „Nonce“ (Number used ONCE), des Nutzers z. Es handelt sich um einen genau einmal zu verwendenden Zufallswert.
- $ID_z, z \in \{i, r\}$: `Identification-Payload` von Initiator (i) oder Responder (r). Dies muss ein bestimmter Wert sein, mit dem sich der Nutzer identifizieren kann. `IKE` schreibt aber nicht konkret vor, was hier genau erwartet wird, sondern überlässt diese Aufgabe einem entsprechenden `DOI`-Dokument. Hier ist es vorstellbar, dass es sich um eine `IP`-Adresse, oder einen `IP`-Adressbereich handelt. Auf jeden Fall muss der Responder mit ID_z entscheiden können, wie die Daten für den entsprechenden Initiator zu behandeln sind. Ob beispielsweise eine Verschlüsselung genügt, oder ob diese auch authentifiziert werden müssen.
- $SKEYID$: Der Masterkey, aus dem die oben beschriebenen Unterschlüssel zur weiteren Schlüsselableitung, Authentifizierung und Verschlüsselung berechnet werden.

Die Berechnung von $SKEYID$ hängt vom verwendeten Authentifizierungsverfahren ab und bestimmt sich hier zu:
$SKEYID := PBG(k, Ni|Nr)$. PBG ist dabei wieder der nicht weiter spezifizierte Pseudozufallsgenerator und k ist der `PSK`.

- $H_z, z \in \{i, r\}$: Hash-Payload
  $Hi := PBG(SKEYID, g^{x_i}|g^{x_r}|CKY_i|CKY_r|SA!|ID_i)$
  $Hr := PBG(SKEYID, g^{x_r}|g^{x_i}|CKY_r|CKY_i|SA!|ID_r)$
  Mit diesen Werten wird der gesamte Nachrichtenaustausch authentifiziert.

Abbildung1 zeigt das Verfahren im Sequenzdiagramm. Vom Initiator wird also zunächst das „Proposal“ gesendet und der Antwortende wählt daraus genau eine Chiffrierfunktion und eine Hashfunktion aus, welche er in $SA!$ angibt. Dann werden die Diffie-Hellmann-Komponenten in KE, nebst Zufallswerten N_z ausgetauscht. Ab dieser Stelle kann der Masterkey $SKEYID$ beiderseits berechnet werden, denn der `PSK` k kann immer als bekannt angekommen werden und die Zufallswerte wurden nun auch ausgetauscht, womit die Parameterliste für den PBG komplett ist.

Im so genannten `aggressive-mode` kann die Anzahl der Pakete noch halbiert werden. Hier werden sämtliche Nachrichten des Antwortenden in einem einzigen Paket übertragen. Die Daten des Initiators werden auf zwei Pakete aufgeteilt, wobei das letzte nur den Hashwert H_i enthält, welcher erst berechnet werden kann, nachdem der Initiator alle Daten des Antwortenden erhalten hat. Dieses Verfahren ist schnell, denn es halbiert nicht nur die Anzahl der übertragenen Pakete, sondern verzichtet auch auf jegliche Verschlüsselung. Damit liegt aber auch der Nachteil sofort auf der Hand, denn die Identitäten der Teilnehmer können hier jederzeit mitgelesen werden.

1.2.4 IKE-Phase1 mit Public-Key-Encryption Authentifizierung

Dieses Verfahren ist der `PSK`-Authentifzierung sehr ähnlich, nur dass die Nutzerauthentizität hier nicht auf der Kenntnis eines `PSK`s beruht, sondern auf ein Public-Key-System verlagert wird. Die Voraussetzung ist also, dass eine entsprechende Infrastruktur, unabgängig von `IPSec` zur Verfügung steht, deren prinzipielle Funktionsweise hier als bekannt angenommen wird und in [11], Kapitel 5.2 nachgelesen werden kann.

Die Authentizität des `Responders` kann nun nachgewiesen werden, indem der `Initiator` einen bestimmten Wert (hier: den Zufallswert N_i) mit dem öffentlichen Schlüssel des `Responders` verschlüsselt (er bildet $E_r(N_i)$). Der `Responder` wird im folgenden als authentisch angenommen, wenn er N_i lesen konnte und damit $SKEYID$ und H_r korrekt berechnet. Diese Werte werden schliesslich an den `Initiator` gesandt, womit er sich von der Echtheit des Antwortenden überzeugt.

Der `Initiator` stellt also die Herausforderung $E_r(N_i)$ an den `Responder`, aus der sicher nur N_i berechnet werden kann, wenn die private Transformation D_r angewandt wird. Die Authentizität des `Responder`s folgt hier also aus der Kenntnis des privaten Schlüssels. Analog prüft der `Responder` die Authentizität des `Initiator`s.

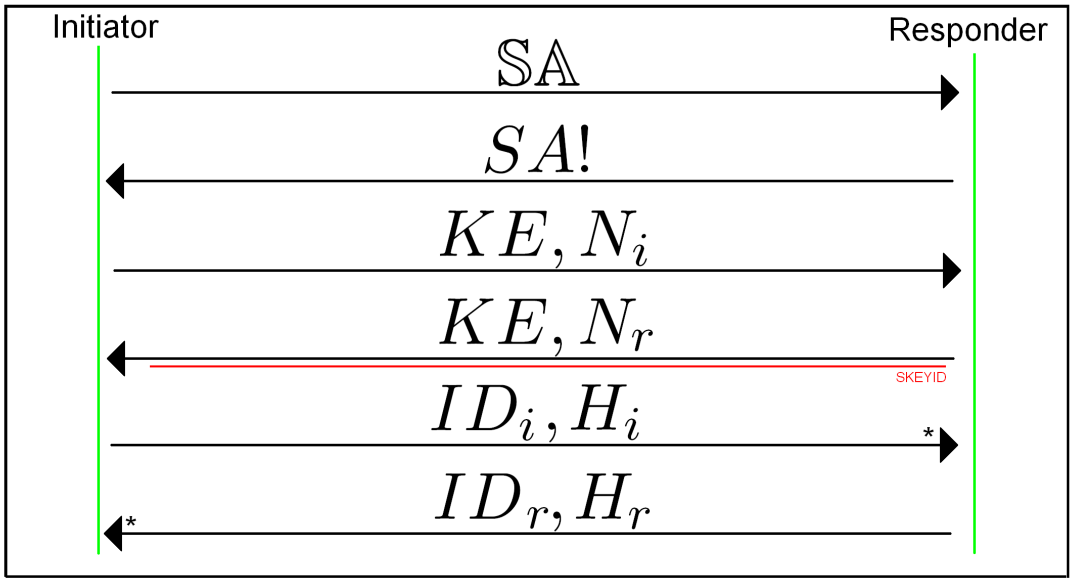

Abbildung 1: Sequenzdiagramm zur PSK-Authentifizierung in IKE.Phase1 (main-mode)

Das Sequenzdiagramm entspricht dem der `PSK`-Authentifizierung, nur dass im zweiten Paket der Zufallswert N_i, wie oben beschrieben, verschlüsselt übertragen wird. Zusätzlich können die Identitäten nun auch ein Paket früher versandt werden, wo sie ebenfalls durch das Public-Key-System geschützt werden. Hier muss nämlich nicht mehr auf die Berechnung von $SKEYID$ gewartet werden, da ja ein unabhängiges Verschlüsselungsverfahren zur Verfügung steht. In den letzten Paketen werden also nur noch die Hashwerte übertragen, die den Nachrichtenaustausch insgesamt authentifizieren.

Die Berechnung des Masterkeys ist für dieses Authentifizierungsverfahren mit $SKEYID := PBG(h(N_i|N_r), CKY_i|CKY_r)$ vorgeschrieben.

Der `aggressive-mode` fasst die Pakete wie bei der `PSK`-Authentifizierung zusammen. Hier ist dieser aber sicherheitstechnisch unproblematisch, da die Identitäten ja mit dem Public-Key System verschlüsselt werden. Diese müssen also nicht im Klartext übertragen werden, was beim `PSK.aggressive-mode` den entscheidenden Nachteil darstellte.

1.2.5 IKE-Phase1 mit Signatur Authentifizierung

Diese Art der Authentifizierung lässt sich noch leichter aus der `PSK`-Authentifizierung herleiten, als die Encryption-Authentifizierung. Im wesentlichen werden hier nur die Hashwerte im letzten Paket durch die entsprechend signierten Werte ersetzt. Optional kann in diesem Paket noch ein Zertifikat übertragen werden.

Die Authentizität des Nutzers $z, z \in \{i, r\}$ folgt also, wenn er die Signatur $Sig_z = D_z(H_z)$ korrekt gebildet hat. Dies kann jederzeit geprüft werden, indem seine öffentliche

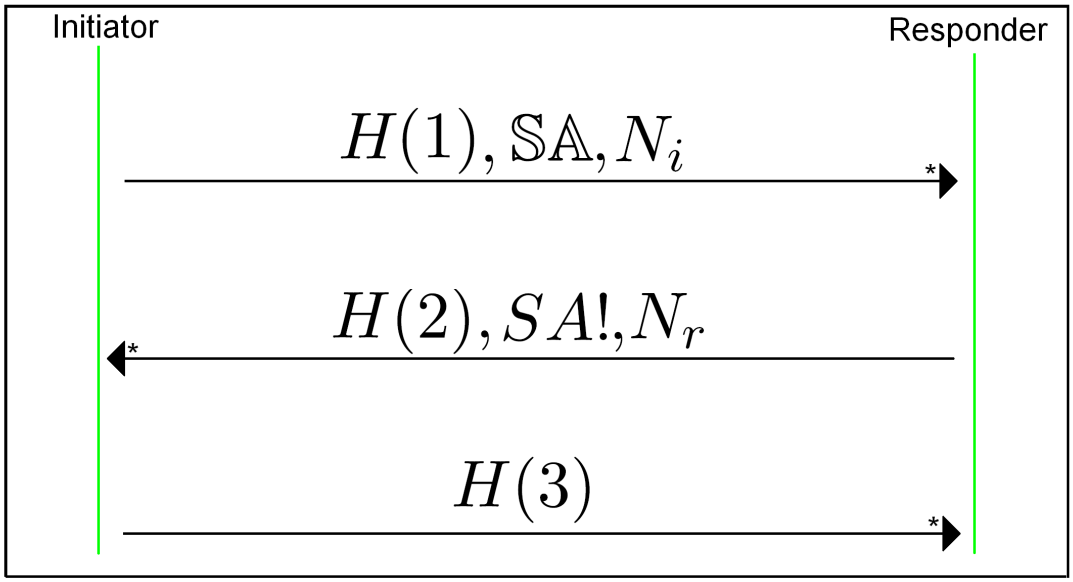

Abbildung 2: IKE.Phase2 (main-mode)

Transformation E_z auf diesen Wert angewandt und das Ergebnis mit dem Hashwert H_z verglichen wird. Voraussetzung für dieses Verfahren ist also auch wieder eine Public-Key Infrastruktur, oder aber das Vorhandensein von Zertifikaten, die Signaturtransformationen bereit halten und im letzten Paket mit übertragen werden.
Da die Hashwerte in `IKE` der Authentifizierung des Nachrichtenaustausches dienen, wird mit Sig_z gleichzeitig Nutzer- und Datenauthentizität realisiert.

Der Masterkey berechnet sich hier zu $SKEYID := PBG(N_i|N_r, g^{x_i x_r})$

1.2.6 IKE-Phase2 zum Aufbau von IPSec-SAs

Durch Phase1 sind nun die Nutzer authentifiziert, die Schlüssel $SKEYID, SKEYID_d$, $SKEYID_a$ und $SKEYID_e$ berechnet und Chiffrier- und Hashfunktionen für `ISAKMP` festgelegt. Damit kann nun jeglicher Verkehr in Phase2, auch bezeichnet als `quick-mode`, verschlüsselt und authentifiziert werden.

Die Aufgabe von Phase2 ist es, eine oder mehrere `ISAKMP-SA`s aufzubauen oder zu erneuern. Insbesondere kann bei der Erneuerung ein neuer Sessionschlüssel festgelegt werden. Da mehrere parallele `quick-modes` ausgeführt werden können, ist hier die `message-ID` wichtig, um die Pakete diesen eindeutig zuordnen zu können.

Abbildung2 zeigt Phase2 im Sequenzdiagramm. Hier werden also im wesentlichen wieder die zu verwendenden Chiffrier- und Hashfunktionen für die `IPSec-SA` ausgehandelt $(\mathbb{SA}, SA!)$ und Zufallswerte ausgetauscht (N_i, N_r). Alle Pakete sind mit $SKEYID_e$ verschlüsselt und in dem Hashwert eines jeden Paketes ist $SKEYID_a$ einberechnet, so dass die Nachrichtenauthentizität im Sinne eines `MAC`s folgt:

$H(1) := PBG(SKEYID_a, M_{ID}|SA|N_i)$
$H(2) := PBG(SKEYID_a, M_{ID}|SA|N_r)$
$H(3) := PBG(SKEYID_a, (0)^8|M_{ID}|SA|N_i|N_r)$

Hier werden also mit $H(1)$ und $H(2)$ die Nutzdaten der ersten beiden Pakete authentifiziert und $H(3)$ authentifiziert nochmals den gesamten Nachrichtenaustausch.

Mittels $SKEYID_d$ und den neuen Zufallswerten N_i und N_r wird nun für eine konkrete `IPSec-SA` ein symmetrischer Schlüssel berechnet. Dieser folgt aus:

$KEYMAT := PBG(SKEYID_d, protocol|SPI|N_i|N_r).$

Dieser Schlüssel kann aber sofort berechnet werden, wenn $SKEYID_d$ kompromittiert wurde. Soll dieses verhindert werden, kann mit den ersten beiden Paketen von Phase2 ein zusätzlicher Diffie-Hellmann Schlüsselaustausch realisiert werden. Dieser Schlüssel muss sich von dem aus Phase1 unterscheiden und liefert dann `Perfect-Forward-Security` (siehe[2]). Hier ist:

$KEYMAT := PBG(SKEYID_d, g^{x_i x_r}|protocol|SPI|N_i|N_r)$

In jedem Fall ist jetzt $KEYMAT$ bestimmt und aus diesem können gewisse Sequenzen für den symmetrischen Verschlüsselungsschlüssel und den `MAC`-Schlüssel herausgegriffen werden. Dabei kann es aber auch schnell vorkommen, dass $KEYMAT$ zu kurz ist und dann muss iterativ verlängert werden:

$KEYMAT := K_1|K_2|K_3|\ldots$
$K_1 := PBG(SKEYID_d, [g^{x_i x_r}|]protocol|SPI|N_i|N_r)$
$K_{i+1} := PBG(SKEYID_d, K_i|[g^{x_i x_r}|]protocol|SPI|N_i|N_r)$

1.3 Die Betriebsmodi von IPSec: Transport- und Tunnelmodus

Prinzipiell gibt es zwei Möglichkeiten, in denen `IPSec` arbeiten kann: Den Transport- und den Tunnelmodus. Damit wird festgelegt, wo im `IP`-Paket der `IPSec`-Header einzufügen ist und nach welcher Adresse das Paket zu routen ist. Der Betriebsmodus wird damit durch die Infrastruktur, in der `IPSec` betrieben werden soll, bestimmt.
Mit `IPSec`-Header ist im folgenden entweder ein `AH`- oder `ESP`-Header gemeint. Dies sind die Unterprotokolle von `IPSec`, mit denen Datenauthentizität oder Vertraulichkeit gewährleistet werden können. Sie werden in den nächsten Abschnitten genauer betrachtet und sind hier noch nicht wesentlich.

1.3.1 Der Transportmodus

Dieser `IPSec`-Betriebsmodus ist in erster Linie für die Direktverbindung zweier Rechner gedacht. Und zwar wird das zu versendende `IP`-Paket um einen `IPSec`-Header ergänzt,

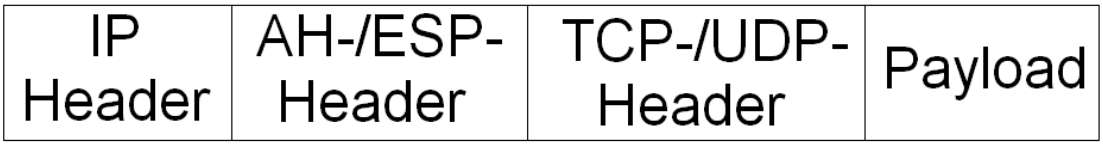

Abbildung 3: IPSec im Transportmodus

der hier zwischen originalem IP-Header und den Nutzdaten steht. Genauer liegt der IPSec-Header also zwischen IP- und TCP-/UDP-Header, je nachdem, was für ein Transportprotokoll verwendet wird.

Die Infrastruktur ist bei diesem Modus dadurch gekennzeichnet, dass „Kommunikationsendpunkte“ und „kryptographische Endpunkte“ zusammenfallen. Damit ist gemeint, dass die Rechner, die miteinander kommunizieren, auch Ver- und Entschlüsselung, sowie Datenauthentifizierung und Datenauthentizitätsprüfung durchführen müssen. Die Endgeräte müssen also IPSec im Protokollstack implementiert haben. Neben direkten Host-zu-Host Verbindungen ist der Transportmodus noch für Host-zu-Router Verbindungen (Netzwerkmanagement) wichtig. Hier kann ein Administrator beispielsweise einen IPSec-Gateway mit Absicherung durch IPSec konfigurieren.

1.3.2 Der Tunnelmodus

Dieser Betriebsmodus zielt auf die Verbindung von Netzwerken (etwa zur Bildung eines VPN) über Gateways, die IP auf IPSec umsetzen, ab. Im Gegensatz zum Transportmodus, mit dem man sicher keine Netzwerke koppeln kann, schliesst der Transportmodus aber nicht aus, dass Rechner direkt in diesem miteinander kommunizieren. In diesem Fall würde der äußere IP-Header gerade dem inneren entsprechen und es würde ein gewisser Overhead in Form von einem zusätzlichen IP-Header entstehen.

Das zu versendende IP-Paket wird im Tunnelmodus also zunächst um den IPSec-Header und dann um einen neuen IP-Header ergänzt. Im Falle der Kommunikation über Gateways dient der äußere IP-Header nun der Adressierung eben dieser. Er spricht also die „kryptographischen Endpunkte“, die das Paket mit den bekannten Sicherheitsservices behandeln, an. Die „Kommunikationsendpunkte“ bekommen davon nichts mit und werden mit dem inneren IP-Header adressiert.

Ein Vorteil des Tunnelmodus ist es, dass IPSec hier nur in den Geräten, die mit dem äußeren Header adressiert werden, implementiert sein muss. Es reicht also, wenn die Gateways IPSec sprechen. Die Clients in damit verbundenen Netzwerken können dann transparent, also ohne IPSec im eigenen Protokollstack laufen zu haben, miteinandern kommunizieren. In diesem Fall besteht auch der Vorteil, dass ein potentieller Angreifer anhand abgefangener Pakete lediglich die IP-Adressen der Gateways rausfinden kann, aber nicht die der kommunizierenden Endgeräte.

Abbildung 4: IPSec im Tunnelmodus

1.4 Das Encapsulating Security Payload Protokoll (ESP)

Ziel des `Encapsulating Security Payload Protocols` (`ESP`, siehe [6]) ist es, die zu versendenden Daten zu verschlüsseln und optional zu authentifizieren. Was hier allerdings verschlüsselt oder authentifiziert wird, ist abhängig vom verwendeten `IPSec`-Betriebsmodus (Transport- oder Tunnelmodus).

Verschlüsselt werden im *Transportmodus* die Nutzdaten des `IP`-Paketes, ohne den `IP`-Header, also `TCP/UDP`-Segmente. Zur Erinnerung: Der `ESP`-Header folgt hier nach dem Original `IP`-Header, mit dem nach wie vor geroutet wird. Dieser darf also auch gar nicht verschlüsselt werden.

Im *Tunnelmodus* werden die gesamten `IP`-Pakete verschlüsselt. Hier wird schliesslich ein zusätzlicher `IP`-Header eingefügt, nach dem geroutet wird, so dass der originale `IP`-Header ohne Probleme mit verschlüsselt werden kann.

In jedem Fall werden die zu verschlüsselnden Daten um ein `Padding`-Feld auf ein Vielfaches von 32 Bit erweitert. Dementsprechend muss auch immer ein 8 Bit langes `PadLength` Feld mit verschlüsselt werden, mit dem der Empfänger dann Nutzdaten und `Pad` trennen kann. Die Art der Nutzdaten sind entweder `TCP/UDP`-Segmente oder `IP`-Pakete und dies wird mit einem weiteren 8 Bit Feld `NextHeader` angezeigt, das als letztes Feld zu verschlüsseln ist.
Im weiteren werden `ESP`-Pakete stets durch einen nicht verschlüsselten Header eingeleitet. Dieser beschreibt mit 32 Bit jeweils den `SPI` (`Security Parameter Index`) und die Sequenznummer. `SPI` dient der eindeutigen Zuordnung dieses Paketes zu einer `IPSec`-SA. Die Sequenznummer hat die Besonderheit, dass jegliche Zyklen verboten sind, so dass bei Wiederholung einer Nummer die Verbindung sofort geschlossen werden muss. Damit wird ein Replay-Angriff, bei dem ein Angreifer ein einzelnes Paket wiederholen würde, unmöglich gemacht.

Optional kann ein `ESP` Paket auch noch um ein 16 Bit Feld `AuthData` erweitert werden. Damit kann tatsächlich eine gewisse Authentizität der Nachrichten sichergestellt werden, wobei aber wieder der Betriebsmodus von `IPSec` bestimmt, was authentifiziert wird. Im Transportmodus sind es, wie oben erklärt, nur die `IP`-Nutzdaten (`TCP/UDP`-Segmente), die authentifiziert werden können. Der `IP`-Header wird hier nicht mit einbezogen. Im Tunnelmodus hingegen wird das gesamte „innere“ `IP`-Paket, also inklusive original `IP`-Header, authentifiziert. Der äußere `IP`-Header, der zum Tunneln genutzt wird, wird hier ebenfalls nicht berücksichtigt.

1.5 Das Authentication Header Protokoll (AH)

Das AuthenticationHeader-Protokoll (AH, siehe[7]) liefert eine Authentifizierung der versendeten IP-Pakete. Dabei wird im wesentlichen ein Integrity Check Value (ICV, korrektere Bezeichnung: MAC) gebildet, der neben den Nutzdaten auch gewisse unveränderliche oder vorhersagbare Felder des IP-Headers mit in die Berechnung einbezieht. Damit ist immer der „äußere“ IP-Header gemeint, nach dem geroutet wird und nicht der IP-Header der im Tunnelmodus in den Nutzdaten stehen kann. Dieser ist trivialerweise durch die Nutzdatenauthentifizierung geschützt. Damit ist aber auch klar, dass AH im Tunnelmodus keinerlei zusätzliche Sicherheit liefern kann, denn dort würde es die Tunnelendpunkt-IP-Adressen (die der Gateways) authentifizieren, was im allgemeinen nicht notwendig ist. Im Tunnelmodus ist das ESP Protokoll für die Sicherung der Authentizität vollkommen ausreichend.

Der AH-Header beinhaltet wieder die Felder SPI und die Sequenznummer, die schon beim ESP-Protokoll erklärt wurden. Vor diesen finden sich ein 8 Bit Feld NextHeader, 8 Bit PayloadLength und 16 reservierte Bit. NextHeader ist dabei wieder vom Betriebsmodus abhängig (IP oder TCP/UDP).

Die authentifizierten Felder des „äußeren“ IP-Headers sind dabei auch abhängig davon, ob IPv4 oder IPv6 genutzt wird. Allgemein kann man sagen, dass für die Felder zur Priorisierung der Pakete und zum Anzeigen der Dienstgüte angenommen werden muss, dass sie innerhalb des Routingprozesses geändert werden, so dass diese in jedem Fall von der Berechnung auszuschliessen sind. Gleiches gilt für TimeToLive bei IPv4 und HopCounter bei IPv6, die definitiv von jedem Router geändert werden. Da sich überhaupt Felder ändern können, ist auch die IPv4-Prüfsumme Header-Checksum von der Berechnung auszuschliessen.

1.6 Kritik an den Betriebsmodi und den IPSec-Unterprotokollen

In [3] findet sich eine sehr kritische Betrachtung von IPSec und seiner Vielzahl an Optionen und Modi, die an dieser Stelle nicht unerwähnt bleiben soll.

Oben wurde festgestellt, dass der Transportmodus in jedem Fall die original IP-Adresse im Header bestehen lässt. Nur in diesem Modus ist das AH-Protokoll sinnvoll, denn es authentifiziert (zumindest teilweise) den IP-Header, nach dem geroutet wird, was ESP nicht leisten kann. Wird der Tunnelmodus benutzt, in dem das gesamte IP-Paket in den Nutzdaten steht und damit auch authentifiziert werden kann, ist AH überflüssig, denn es würde lediglich die „äußere“ IP-Adresse, also die der Gateways authentifizieren und das liefert keine zusätzliche Sicherheit. Man kann also feststellen, dass AH nur im Transportmodus sinnvoll eingesetzt werden kann und dass ESP im Tunnelmodus sogar eine stärkere Authentizität der Pakete liefert, als AH im Transportmodus. Bei ESP im Tunnelmodus wird schliesslich der gesamte innere IP-Header authentifiziert und nicht nur dessen unveränderliche Felder.
Empfohlen wird daher, den Transportmodus und AH zu eliminieren und damit die Komplexität von IPSec zu reduzieren. Für Direktverbindungen zwischen zwei Rechnern müss-

ten dann zwar die Pakete auch getunnelt werden, was eine „Verschwendung“ von einem Header bedeuten würde, denn der innere und äußere IP-Header wären hier gleich. Dies ist aber ein akzeptabler Preis für die dadurch entstehende Komplexitätsreduktion.

1.7 IPSec im Einsatz: VPNs

Abschliessend soll auf die häufigste praktische Anwendung von IPSec eingegangen werden: Die Bildung virtueller privater Netzwerke.

Das Konzept sieht vor, dass zwei entfernte Netze, etwa ein Unternehmensnetzwerk und das Netzwerk einer zugehörigen Zweigstelle, auf sichere Art über das Internet gekoppelt werden können, so dass sie als ein großes Netzwerk erscheinen. In diesem Netzwerk sollen also alle Zugriffe möglich sein, die in einem lokal beschränkten LAN auch möglich sind. Dazu sind die Netzwerke jeweils über Security-Gateways an das Internet angeschlossen. Diese tunneln alle Pakete, die aus einem der beiden Netzwerke in das andere geschickt werden sollen. Dabei wird der Security-Gateway des entsprechenden Netzwerkes mit dem neuen, äußeren IP-Header adressiert, der IPSec-Header ist abhängig von eingesetzten IPSec-Unterprotokoll (AH oder ESP) und die IP-Adresse des Originalpakets bleibt typischerweise bestehen. Der adressierte Gateway entfernt dann den tunnelnden IP-Header, entschlüsselt die Daten gemäß Vorgabe durch den IPSec-Header und stellt das innere IP-Paket an den adressierten Rechner in seinem Netzwerk zu. Anzumerken ist dabei, dass es untypisch ist, das AH-Protokoll im Tunnel einzusetzen, da die Security-Gateways als authentisch angenommen werden können.

VPNs sind damit sicher konstengünstiger als eine funktional gleiche Realisierung über eine Standleitung. Im VPN ist es außerdem möglich, mobilen Mitarbeiter, die Zugriff aufs Internet haben, jederzeit auch Zugriff auf das Firmennetzwerk zu ermöglichen. Damit ist ein VPN auch flexibler, als eine Standleitung.

Es gibt drei Konfigurationsmöglichkeiten von VPNs, die prinzipiell unterschieden werden können. Bei Host-to-Host Verbindungen sind zwei Rechner, die IPSec sprechen, direkt miteinander verbunden. Beim Gateway-to-Gateway-VPN sind, wie oben beschrieben, zwei geographisch entfernte Netzwerke über Router, die IP auf IPSec umsetzen (also über Security-Gateways) miteinander gekoppelt, so dass sie als ein lokales Netzwerk erscheinen. In der „mehrfach verschachtelten Konfiguration“ sind diese beiden Möglichkeiten kombiniert. Sinnvollerweise ist dabei jeglicher Verkehr zwischen den Routern über das Internet mit dem ESP-Protokoll verschlüsselt. Optional können sich dann einzelne Rechner in den verbundenen Netzwerken mit dem AH-Protokoll authentifizieren. IPSec muss dann natürlich auch in den sich authentifizierenden Endsystemen implementiert sein. Genau genommen tunneln die Router in diesem Fall auch kein IP-Paket, sondern ein IPSec-Paket.

2 Secure Socket Layer (SSL)

Bei SSL handelt es sich um ein Protokoll der Transportschicht. Als solches arbeitet es also über der Internetschicht (IP,IPX) und unter der Anwendungsschicht. Genauer arbeitet SSL auf der Transportschicht über TCP oder UDP, das die eigentlichen Transportaufgaben übernimmt. Das bedeutet aber, dass Anwendungen nicht mehr direkt auf das jeweilige Transportprotokoll zugreifen können, sondern SSL nutzen müssen. Deshalb muss SSL auch die Aufgabe der Segmentierung und Reassemblierung der Anwedungsdaten übernehmen. Anwendungen können also nach wie vor transparent über SSL kommunizieren, indem sie diesem einen kontinuierlichen Anwendungsdatenstrom übergeben und im Empfangsfall einen ebensolchen erhalten. SSL zerlegt diesen Datenstrom in Segmente, behandelt diese mit Transformationen zur Sicherung von Vertrauchlichkeit und Datenauthentizität und reicht diese Segmente dann an TCP oder UDP weiter. Zu diesem Zweck ist SSL in fünf Unterprotokolle aufgeteilt, wobei aus kryptographischer Sicht insbesondere das Handshake- und das Record-Protokoll ausführlich betrachtet werden müssen.

Praktisch wird SSL in der Regel eingesetzt, um Client-Server-Verbindungen aufzubauen und abzusichern. Dies wird beim Client dann beispielweise durch Benutzung des https-Befehls initialisiert. In den meisten Szenarien findet auch eine Authentifizierung des Servers durch Zertifikate statt und auf eine clientseitige wird aus Kostengründen oft verzichtet. Die günstigsten Zertifikate werden ab 200$ jährlich ausgestellt, einer Summe, die nicht jedem, der Onlinebanking oder -shopping betreiben möchte zugemutet werden kann.
Der folgende Abschnitt richtet sich im wesentlichen nach [9].

2.1 Sessions und Connections in SSL

In SSL müssen die Zustände Session und Connection prinzipiell unterschieden werden. Eine Session ist eine übergeordnete und langlebige Verbindungen, die zu einer Connection aktiviert, oder wiederaufgenommen, werden kann. In diesem Zusammenhang wird der Session-Status als pending und der Connection-Status als current bezeichnet.

Sessions beschreiben langlebige Verbindungen zwischen Client und Server im Sinne einer Vereinbarung. Hier sind die übergeordneten Parameter vereinbart:

- SessionID: Dient in erster Linie der Unterscheidung mehrerer Connections beim Server, der diesen Wert bestimmen kann.
- peer_certificate: Aktuell vorgeschrieben ist die Benutzung von X509v3-Zertifikaten. Diese liefern also die Nutzerauthentizität. Von Seiten des Clients kann dabei auch null angegeben werden, was von den meisten Servern akzeptiert werden wird.
- cipher_spec: Hier sind ein bestimmter Verschlüsselungs- und Hashalgorithmus festgelegt. Zweiterer dient der Authentifizierung der Nachrichten.
- compression_method: SSL komprimiert optional die Anwendungsdaten und hiermit ist ein entsprechender Algorithmus festgelegt.

- `MASTER_SECRET`: Die Schlüsselgrundlage für Verschlüsselung, `MAC`-Berechnung und Initialisierungsvektoren. Aus dieser werden also konkrete Schlüssel berechnet, wenn diese `Session` zur `Connection` aktiviert wird. Wird diese `Session` wiederaufgenommen, sind jeweils neue Zufallswerte mit einzuberechnen.

- `is_resumable`: Ein Flag, das anzeigt, ob diese `Session` nach Abbruch einer `Connection` erneut zu einer solchen aktiviert werden kann.

Connections sind konkrete Transportverbindungen, über die Daten übertragen werden können und die als aktivierte `Sessions` aufgefasst werden können. Dabei werden die oben genannten langlebigen Parameter der `Session` übernommen und es werden Zufallswerte ausgetauscht, mit denen aus dem `MASTER_SECRET` der `Session` ein neuer Schlüssel für die `Connection` berechnet werden kann. Genauer ist eine `Connection` charakterisiert durch:

- `Server-/Client-Random`: 32 Bit Zufallszahl, die zur Berechnung des `MASTER_SECRET` aus dem `pre_master_secret` genutzt wird. Für den Fall, dass eine `Connection` aus einer `Session` wieder aktiviert wird, ist dieser Wert neu zu bestimmen.

- `Server-/Client.Write.MAC.secret`: Geheimer Wert, zur Parametrisierung der Hashfunktion, die bei der Berechnung des `MAC` genutzt wird.

- `Server-/Client.Write.Key`: Symmetrischer Schlüssel für die Sicherung der Vertraulichkeit.

- `Initialisation.Vector`: Diesen Wert benutzten beide Parteien als 0. Block, wenn Blockchiffren im `CBC`-Modus verwendet werden.

- `Sequence.Numbers`: Tatsächlich ist für jede `Connection` eine bestimmte Menge von Sequenznummern charakteristisch. Jede dieser ist nämlich eindeutig für ein bestimmtes Segment, was bedeutet, dass die `Connection` sofort geschlossen wird, wenn Sequenznummern sich wiederholen. Da es sich um ein 64 Bit Feld handelt, ist die maximale Zyklenlänge aber mit 2^{64} Segmenten hinreichend groß. Replay-Angriffe werden damit effizient verhindert.

2.2 Das SSL-Handshake Protokoll

2.2.1 Grundlagen des SSL-Handshakes

Im `SSL-Handshake` geht es darum, dass sich Client und Server

- 1. mit Zertifikaten authentifzieren

- 2. die zu benutzenden Algorithmen für die Sicherung von Vertraulichkeit und Datenauthentizität aushandeln

- 3. sich auf ein `MASTER_SECRET` einigen.

Bei der Authentifizierung von Server und Client sind nur drei von vier möglichen Kombinationen erlaubt. Typischerweise kann ganz auf Authentifizierungen verzichtet werden, beide Parteien können sich authentifizieren, oder nur der Server authentifiziert sich. Es ist nicht zulässig, dass ein anonymer Server eine Clientauthentifizierung verlangt und dies führt sofort zum Abbruch der Verbindung.

Als symmetrische Blockchiffren können im wesentlichen der `DES`, `3-DES`, `IDEA` und `RC2` ausgehandelt werden. Alternativ kann die Stromchiffre `RC4` genutzt werden. Für die Berechnung der `MAC`s können die Hashfunktionen `MD5` oder `SHA1` ausgewählt werden.

Das `MASTER_SECRET` wird hier immer unter Benutzung des `pre_master_secret` berechnet. Dessen Form ist wiederum davon abhängig, was für einen Schlüssel die Zertifikate von Client und Server enthalten. Die formale Definition der Berechnung lautet:

```
M_S := MD5[p_m_s||SHA1('A'||p_m_s||C.h.r||S.h.r)]||
MD5[p_m_s||SHA1('BB'||p_m_s||C.h.r||S.h.r)]||
MD5[p_m_s||SHA1('CCC'||p_m_s||C.h.r||S.h.r)]
```

Dieser Wert gilt für die gesamte `Session`. Für eine `Connection` wird daraus auf ähnliche Art, aber mit neuen Zufallswerten konkretes Schlüsselmaterial `key_block` berechnet:

```
key_block := MD5[m_s||SHA1('A'||m_s||S.h.r||C.h.r)]||
MD5[m_s||SHA1('BB'||m_s||S.h.r||C.h.r)]||...
```

Dies wird so lange fortgeführt, bis Sequenzen für den symmetrischen Vertraulichkeitsschlüssel, für die Parametrisierung beim `MAC` und für die Initialisierungsvektoren von Blockchiffren im `CBC`-Modus herausgegriffen werden können.

2.2.2 Nachrichtenaustausch zum Handshake

Um eine Verbindung aufzubauen, werden gewissen Nachrichten ausgetauscht. Deren Format ist dabei denkbar einfach und besteht aus einem 8 Bit Feld `type`, einem 24 Bit Feld `length` und einem Nutzdatenfeld variabler Länge. Das `type`-Feld gibt die Art der Nutzdaten an und das `length`-Feld deren Länge in Byte. Insgesamt existieren zehn Nachrichten, die hier betrachtet werden müssen:

- `client_hello`: Der Client beginnt den Verbindungsaufbau und übermittelt seinen Zufallswert, die maximale `SSL`-Version, die er unterstützt, eine bestimmte `Session-ID` und die Menge `cipher-suite`, in der seine unterstützten Verschlüsselungs- und Hashfunktionen eingetragen sind.

- `server_hello`: Diese Nachricht hat die gleichen Parameter wie `client_hello`, aber diese haben eine andere Bedeutung. Der Zufallswert muss sich von dem des Clients unterscheiden, die `SSL-Version` wird so gewählt, dass sie kompatibel zu der des Clients ist, also nicht zwingenderweise maximal. Die `Session-ID` des Clients wird vom Server übernommen, wenn er sie noch nicht für einen anderen Client

vergeben hat und aus der `cipher-suite` des Clients wählt der Server die jeweils sichersten Funktionen aus.

- `certificate`: Das `X509v3`-Zertifikat. Kann auch `null` sein.
- `certificate_request`: Damit kann ein nicht-anonymer Server eine Clientauthentifizierung verlangen.
- `server_hello_done`: Da einige Nachrichten des Servers optional sind, zeigt der Server hiermit an, dass die Sequenz seiner Nachrichten abgeschlossen hat. Sonst wüSSte der Client nicht, ob er beispielsweise noch auf ein verzögertes `certificate_request` warten muss, oder ob der Server eine solche Nachricht gar nicht gesendet hat.
- `change_cipher_spec`: Damit wird die `Session` zur `Connection` aktiviert und also angezeigt, dass die ausgehandelten Funktionen und Parameter nun verwendet werden.
- `finished`: Dies zeigt das Ende des `Handshakes` an und authentifziert alle ausgetauschten Nutzdaten.

Die drei wichtigsten Nachrichten sind in dieser Aufzählung noch nicht erwähnt und sollen etwas genauer betrachtet werden. Im vorherigen Anschnitt wurde schon erwähnt, dass die Bestimmung des `pre_master_secret` von der Art der Schlüssel in den Zertifikaten abhängt. Gemäß dieser werden zur Bestimmung des `pre_master_secret` folgende Nachrichten verschickt:

- `server_key_exchange`: Diese Nachricht muss nur versandt werden, wenn das Serverzertifikat keinen festen Diffie-Hellmann- oder RSA-Schlüssel bereit stellt. Etwa kann es sich um `anonymous-DH` handeln, bei dem der Schlüssel nur nicht durch das Zertifikat öffentlich gemacht wird, oder es kann sich um `ephemeral-DH` handeln, bei dem der Schlüssel regelmäßig erneuert werden muss. Alternativ kann der Server auch temporäre RSA-Schlüssel generieren und deren öffentliche Komponente dem Client hiermit bekannt machen. Dies kann auch dann nötig werden, wenn das Serverzertifikat zwar einen RSA-Schlüssel bereitstellt, dieser aber nur zur Signierung verwendet werden darf.
- `client_key_exchange`: Diese Nachricht ist, im Gegensatz zur entsprechenden Servernachricht, immer zu versenden. Wenn bereits das Zertifikat des Clients einen festen Schlüssel beinhaltet, ist die Nachricht leer. In den Fällen `anonymous-DH` oder `ephemeral-DH` ist die Nachricht analog zur Servernachricht mit der öffentlichen Schlüsselkomponente versehen. Hat der Server dem Client einen RSA-Schlüssel mitgeteilt, generiert der Client das `pre_master_secret` als Zufallswert, verschlüsselt es mit dem öffentlichen RSA-Schlüssel des Servers und sendet es diesem mit `client_key_exchange`.
- `certificate_verify`: Mit dieser Nachricht signiert der Client sein Zertifikat, falls dieses keinen festen Schlüssel enthält.

In jedem Fall ist mit diesen Nachrichten das `pre_master_secret` bestimmt. Wird das Diffie-Hellmann Verfahren eingesetzt, entspricht das `pre_master_secret` gerade dem symmetrische Schlüssel aus diesem Verfahren und bei RSA ist es ein vom Client generierter Zufallswert.

In Abbildung5 ist der genaue Ablauf des `SSL-Handshakes` dargestellt. Gestrichelte Pfeile deuten dabei an, dass die entsprechende Nachricht optional ist.
In der ersten Phase werden Zufallswerte ausgetauscht und verwendete Version, sowie Algorithmen ausgehandelt. In der zweiten Phase geht es darum, dass Client und Server sich gegenseitig authentifizieren und dabei auch das `pre_master_secret` austauschen. Der Server sendet zunächst sein Zertifikat und/oder Schlüssel in den entsprechenden Nachrichten und verlangt optional, dass sich der Client authentifiziert. Das Ende dieser Nachrichten zeigt der Server dann mit `server_hello_done` an. Wurde die Authentifizierung verlangt, schickt der Client auch sein Zertifikat, oder eine entsprechende Warnmeldung. Dies kann der Server akzeptieren und der Verbindungsaufbau geht ohne Probleme weiter. Mit der `client_key_exchange`-Nachricht beendet der Client den Schlüsselaustausch und ab hier können beide das `pre_master_secret` berechnen. Optional signiert der Client nun noch sein Zertifikat. Da der Server anhand der `certificate` Nachricht des Clients schon erkennen kann, ob die `certificate_verify`-Nachricht noch folgen wird, ist eine Abschlussnachricht des Clients hier nicht erforderlich. Im letzten Schritt zeigen beide an, dass sie die aktuell ausgehandelten Parameter übernehmen und authentifizieren alle Nutzdaten, die innerhalb des `Handshakes` ausgetauscht wurden.

2.3 Das SSL-Record Protokoll

Aufgabe dieses Protokolls ist es, die Anwendungsdaten mit allen notwendigen Diensten zur Sicherung von Vertraulichkeit und Datenauthentizität zu versorgen und auch eine optionale Komprimierung dieser Daten. Ferner müssen die Anwendungsdaten fragmentiert werden.

Auf die fragmentierten Daten kann also optional eine Komprimierungsfunktion angewandt werden. Diese ist zwingenderweise verlustfrei und muss auch die Eigenschaft erfüllen, Daten keinesfalls auf über 1024 Byte zu vergrößern. Eine Vergrößerung ist allgemein denkbar, wenn Daten komprimiert werden sollen, die von vorn herein schon sehr klein sind.

Die optional komprimierten Daten werden dann um einen `MAC`-Wert ergänzt. Dessen Berechnung ist definiert durch:

```
h(MAC_write_secret||pad2||h(MAC_write_secret||pad1||seq_num||length||content))
```

Im inneren Hashwert wird also der Schlüssel, der nur den kommunizierenden Parteien bekannt ist, konkateniert mit Sequenznummer, dem Längenfeld und den eigentlichen Nutzdaten. Das ganze wird mit einem Padding-Feld (`0x36`) auf die Eingabelänge der Hashfunktion aufgefüllt. Die Berechnung des äußeren Hashwertes erfolgt dann ganz analog, nur dass das Padding-Feld anders definiert ist (`0x5c`). Insgesamt werden damit also das eventuell komprimierte Segment, seine Größe im unkomprimierten Zustand und seine

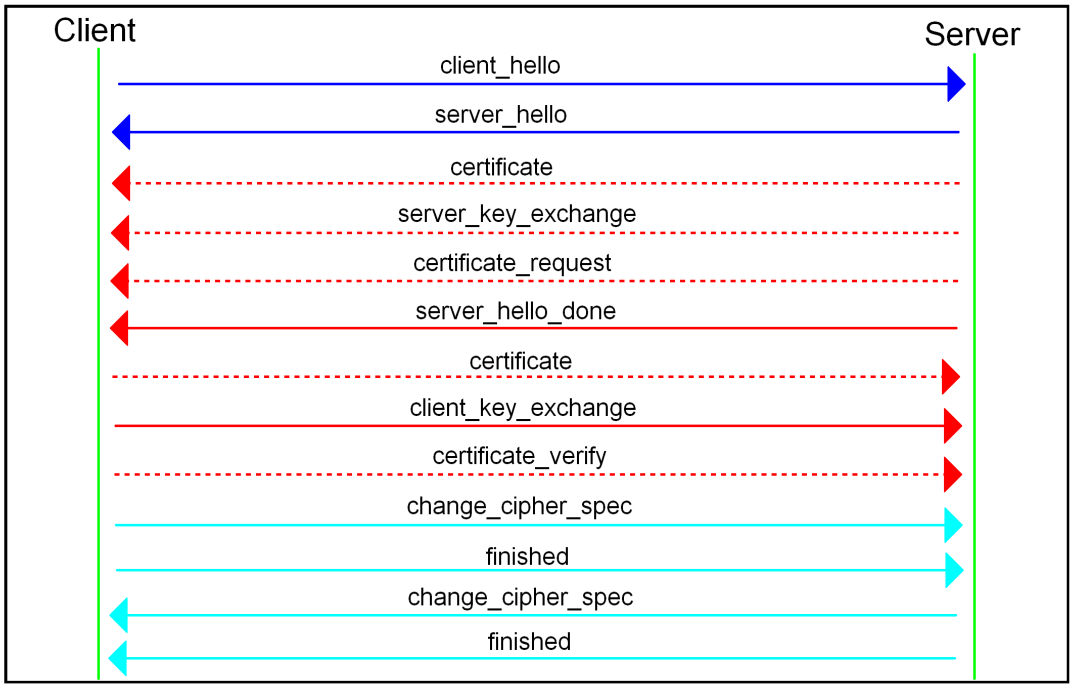

Abbildung 5: Sequenzen des SSL Handshakes

Sequenznummer authentifiziert.

Nach der Authentifizierung der Daten erfolgt die Verschlüsselung mit dem in `SSL-Handshake` ausgehandelten Algorithmus.

Schliesslich wird das Segment noch um den `Record-Header` ergänzt. Dieser besteht aus:

- 8 Bit `ContentType`: Dieses Feld gibt das nächsthöhere Protokoll an. Also `SSL-Handshake`, `Alert`, `ChangeCipherSpec`, oder `Application`. Letzteres dient dabei dem Versenden von Anwendungsdaten.
- 16 Bit `Major/MinorVersion`: Hier werden Haupt- und Unterversion angegeben. Aktuell: `3.0`.
- 8 Bit `CompressedLength`: Die Länge des Segmentes im unkomprimierten Zustand.

2.4 Das SSL-ChangeCipherSpec- und das SSL-Alert-Protokoll

2.4.1 Das SSL-ChangeCipherSpec-Protokoll

Bei diesem Protokoll handelt es sich um ein „triviales“ Protokoll. Seine Funktion wurde schon in `SSL-Handshake` genutzt und besteht lediglich daraus, eine einen Byte große Nachricht zu verschicken. Diese Nachricht zeigt dem Empfänger an, dass der Absender

die beiderseits ausgehandelten Optionen übernimmt. Damit wird also der Wechsel vom `Session`-Status zum `Connection`-Status angezeigt.

2.4.2 Das SSL-Alert-Protokoll

Mit dem `alert-protocol` werden 2 Byte große Alarmmeldungen verschickt. Dabei spezifiziert das erste Byte den Schweregrad des Alarms (`warning` oder `fatal`) und das zweite die genaue Art der Meldung. Bei `warning`-Meldungen handelt es sich um Sicherheitshinweise, die nicht zum Abbruch der `Connection` führen müssen. Ihre Interpretation ist von der jeweiligen Implementierung des Protokolls abhängig, darf also auch zum Verbindungsabbruch führen. Das Protokoll gibt nur vor, dass für entsprechenden Ereignisse ein Alarm generiert werden muss. `fatal`-Meldungen müssen hingegen in jeder `SSL`-Implementierung zum Abbruch der entsprechenden `Connection` führen.

Die `warning`-Meldungen sind im einzelnen:

- `close_notify`: Über den Kommunikationsport werden ab jetzt keine Daten mehr gesendet und empfangen.
- `no_cert`: Der Client kann damit anzeigen, dass er kein Zertifikat besitzt, obwohl der Server dies angefordert hat.
- `bad_cert`: Die Signaturprüfung des Zertifikats ist negativ ausgegangen.
- `unsopported_cert`: Das Zertifikatsformat wird nicht unterstützt.
- `cert_revoked`: Das Zertifikat befindet sich auf einer zentralen Sperrliste.
- `cert_expired`: Das Zertifikat ist abgelaufen.
- `cert_unknown`: Das Format oder der Aussteller ist unbekannt.

Die `fatal`-Meldungen können sein:

- `unexcepted_message`: Es wurde gegen die im Protokoll definierte Sequenz der Nachrichten verstoßen. Der Zustand, in dem ein anonymer Server eine Clientauthentifizierung verlangt, muss beispielsweise dazu führen, dass der Client diese Meldung verschickt.
- `bad_record_mac`: Die Nachricht ist nicht authentisch oder nicht unangetastet.
- `decompression_failure`: Die Daten ließen sich nicht korrekt dekomprimieren.
- `handshake_failure`: Muss gesendet werden, wenn sich Client und Server nicht auf die gleichen Parameter einigen konnten. Beispielsweise wenn der Server keine der Verschlüsselungsfunktionen des Clients unterstützt.
- `illegal_parameter`: Wert eines Feldes außerhalb des gültigen Bereichs

Es wird also klar, dass Probleme im Zusammenhang mit Zertifikaten nicht zwangsläufig einen schwerwiegenden Fehler darstellen und zum Verbindungsabbruch führen müssen. Dies lässt sich nachvollziehen, wenn man beachtet, dass Zertifikate hohe laufende Kosten verursachen und deswegen nicht bei jedem Client vorausgesetzt werden können. Verstöße gegen das Protokoll oder Fehler bezüglich Nachrichtenauthentizität und Kompression müssen immer zum Abbruch der `Connection` führen, da ein Angriff angenommen werden muss.

3 Computerviren

In diesem Abschnitt geht es um die Betrachtung von Computerviren für verschiedener Systemarchitekturen. Dabei werden jeweils die verschiedenen Infektions- und Verbergungsmöglichkeiten betrachtet. Danach wird auf den Virenschutz eingegangen. Grundlage ist [9].

3.1 Einführung

3.1.1 Def.: Virus

An dieser Stelle muss zunächst der Begriff „Virus“ definiert und abgegrenzt werden. Zu verstehen ist darunter ein Computerprogramm, das andere Programme derart infizieren kann, dass sie eine, möglicherweise weiterentwickelte, Kopie des Virus enthalten.
Typisch für Viren ist also insbesondere die Fähigkeit zur Infektion anderer Programme. Nicht zwingenderweise erhalten sie eine Schadroutine, die etwa Daten zerstören oder CPU-Zeit stehlen kann. Oft wird auch die Funktion des befallenen Programms beeinflußt. Selten kommt es auch vor, dass Viren das infizierte Programm nicht direkt befallen, sondern seine Umgebung beeinflussen. Diese Alternative wird hier mit den Companionviren betrachtet.
Die Tatsache, ob ein Virus Schaden verursacht oder nicht, hängt dabei sicher von der Motivation des Autors ab. Häufig werden Viren dazu genutzt, um Schwachstellen in Systemen aufzuzeigen, oder um zu beweisen, dass für eine bestimmte Architektur überhaupt Viren entwickelt werden können. Eine Schadroutine ist dann eher untypisch. Geht das Virus bei der Programminfektion allerdings ungeschickt vor, kann es passieren, dass das Originalprogramm zerstört wird. Dies ist allerdings eher als ungewollte Schadroutine zu bewerten.

Vom Virusbegriff müssen die „Trojanischen Pferde“ und die „Würmer“ prinzipiell abgegrenzt werden.
Bei *Trojanischen Pferden* handelt es sich um eigenständige Programme, die für den Nutzer eine augenscheinlich nützliche Funktion ausführen. Zusätzlich führen sie aber oft eine geheime Aktion im Hintergrund aus. Hier kann es sich um das Löschen oder Sammeln gewisser Dateien handeln. Denkbar sind hier in jedem Fall auch private Schlüssel, die auf der Festplatte des Nutzers abgelegt sind und damit an den Autors des Trojanischen Pferdes weitergeleitet werden. Die Verbreitung erfolgt meist als Anhang an e-Mails und es existiert keine Automatisierung im eigentlichen Programm hierfür.
Bei *Würmern* handelt es sich ebenfalls um ein eigenständiges Programm, für das aber die autonome Weiterverbreitung auf andere Rechner über das Internet charakteristisch ist. Die Motivation ist hier oft mit der von Viren vergleichbar, so dass eine Schadroutine meist nicht vorhanden ist, aber Schwachstellen in bestimmten Systemen aufgezeigt werden. Bei Würmern sind dies meist Schwachstellen im Zusammenhang mit netzwerktechnischen Aspekten in bestimmten Betriebssystemen. 1988 zeigte der „Morris-Wurm“ beispielsweise, dass man in `UNIX`-Systemen ohne Authentifizierung Programme auch übers

Netzwerk installieren kann. Eine Schadroutine widersprach glücklicherweise der Motivation des Autors.

3.1.2 Verbergungsstrategien von Viren

Auf allen Systemen kann man gewisse einheitliche Verbergungsstrategien von Viren identifizieren, da diese im wesentlichen unabhängig von der Art der Infektion sind.

Eine Verbergungsmöglichkeit realisieren die sogenannten Stealthviren. Diese fangen gewisse Systemaufrufe ab. Beispielsweise geben sie bei Abfrage der Größe der befallenen Datei die Größe vor der Infektion an. Dieser Wert entspricht natürlich nicht mehr dem korrekten, der durch die Infektion oft ansteigt.
Virenscanner suchen aber meist nach konkreten Virensignaturen in potentiell befallenen Programmen und prüfen nicht nur die Dateigröße. Virensignaturen sind Codeabschnitte, die für einen Virus charakteristisch sind und die in nicht-befallenen Programmen ausgeschlossen werden können.

Viren versuchen daher meist die Anzahl der möglichen Virensignaturen im befallenen Programm zu minimieren. Ein häufiger Ansatz ist es dabei, dass der Hauptteil des Virus, der auch mögliche Schadroutinen enthält, durch eine Verschlüsselungsroutine verschlüsselt wird. Dabei kann bei der Weiterverbreitung der Schlüssel geändert werden, so dass Virenscanner keine Chance mehr haben, Virensignaturen zu finden. Typischerweise können die Virenscanner nun aber einfach nach der Verschlüsselunsroutine suchen, die sicher immer unverschlüsselt vorliegen muss. Alleinige Verschlüsselung des Hauptteils bietet also keine gute Verbergungsmöglichkeit für Viren.

Ein weiterentwickelter Ansatz findet sich aber mit dem Polymorphismus. Hier ist der verschlüsselte Hauptteil um eine weitere Routine ergänzt, die die Verschlüsselungsroutine immer neu schreibt, wenn der Hauptteil aktiviert wird. Damit kann ein Virenscanner also weder Signaturen, noch eine einheitliche Verschlüsselungsroutine finden.

Eine Spezialform des Polymorphismus findet sich mit dem Metamorphismus. Hier wird die Verschlüsselungsroutine durch das eigentliche Virenprogramm disassembliert und es werden nicht-eindeutige Befehle im Code ersetzt oder es werden Befehle ohne Seiteneffekt eingefügt. So kann sicherlich jederzeit jedes Register mit sich selbst verodert werden. Es wird also der Code verändert, ohne die logische Funktion zu beeinflussen.

3.2 IBM-PC Viren

Beim IBM-PC lassen sich drei Gruppen von Viren klassifizieren, die sich jeweils in der Strategie bei der Infektion unterscheiden. Die wichtigste Gruppe nehmen sicher die dateiinfizierenden Viren ein und diese stellen auch aktuell noch eine große Bedrohnung dar. Bootsektorinfizierende Viren und Companionviren geben einen Aufschluss darüber, wie ein Virenautor Schwachstellen in einem bestimmten System aufdecken und ausnutzen kann, sind aber aktuell keine Bedrohung mehr. Gegen erstere bietet das `BIOS` heutzutage einen zuverlässigen Schutz und zweitere sind seit dem Verschwinden von `DOS` aus

`Windows` ebenfalls nicht mehr aufgetaucht.

3.2.1 Dateiinfizierende Viren

Bei dieser Art der Infektion bringt das Virus seinen Code in den Programmcode des infizierten Programms ein. Idealerweise so, dass der Virencode immer zunächst beim Programmstart ausgeführt wird und dann das Programm regulär startet. Bei geschickter Programmierung merkt der Anwender dann nur eine kurze Verzögerung beim Programmstart und kann sonst ohne Beeinträchtigung arbeiten. Umgekehrt kann natürlich auch ein entsprechend schlecht programmiertes Virus das infizierte Programm irreversibel schädigen, obwohl eigentlich kein schadhaftes Verhalten vom Autor beabsichtigt war.

Die einfachste Möglichkeit der Dateiinfektion stellen sicher die *Prependerviren* dar. Wie der Name schon sagt, wird der virale Code am Dateianfang des befallenen Programms eingefügt und der Originalcode geshiftet. Es ist auch klar, dass das Virus immer beim Programmstart ausgeführt wird, denn dann werden ja gerade die ersten Byte der befallenen Datei geladen. Ist das Virus dann aktiv, muss es das Originalprogramm wiederherstellen und starten, damit die Infektion nicht bemerkt wird. Virenscanner können Prependerviren aber sicher sehr schnell erkennen, da nur die ersten Byte aller Programme abgesucht werden müssen.

Den nächsten Entwicklungsschritt bei der Dateiinfektion stellte der *Appendervirus* dar. Dieser ist etwas komplexer, als ein Prependervirus, da der virale Code am Ende steht und nicht automatisch beim Programmstart geladen wird. Das Virus muss also die Startsequenz durch einen Sprungbefehl zum Virencode ersetzen und die Originalsequenz speichern. Damit führt ein Programmstart also zum Sprung an das Dateiende und damit zur Ausführung des Virus. Hiernach würde das Virus die Originalstartsequenz wiederherstellen und das Programm damit regulären starten lassen.
Für Virenscanner ist es ebenfalls kein Problem, auch einen Blick an das Dateiende zu werfen. Für diese steigt die Komplexität nur um einen konstanten Faktor, bleibt also gleich.

Die anspruchsvollste Variante, sowohl in der Programmierung als auch beim Auffinden, bilden sicher die *Cavityviren*. Diese fügen sich mitten in den Programmcode ein, ohne dabei die Dateigröße zu ändern. Einfachsterweise werden hierzu unbenutzte Programmabschnitte ausgenutzt. Eine etwas anspruchsvollere Version der Cavityviren stellt das *Kompressionsvirus* dar. Dieses sucht nach konstanten Abschnitten in der zu infizierenden Datei, komprimiert diese und nutzt den entstehenden Freiraum für den eigenen Code. Etwa könnte eine Lauflängenkomprimierung angewandt werden, indem ein hinreichend langer Abschnitt, der konstant auf `1` oder `0` ist, gesucht wird. Hier müssen dann nur das entsprechende Zeichen und die Anzahl der Wiederholungen dieses Zeichend festgehalten werden und der Freiraum könnte überschrieben werden. Zur Laufzeit erfolgt dann die Dekomprimierung.
Es ist sofort klar, dass ein Virenscanner den gesamten Programmcode absuchen muss,

um Cavityviren zu finde. Diese begründen damit den größten Komplexitätszuwachs beim Scanvorgang.

3.2.2 Bootsektorinfizierende Viren

An dieser Stelle sollen kurz die *Bootsektorviren* Erwähnung finden. Diese sind heutzutage nur noch historisch interessant, da aktuelle BIOS-Implementierungen einen hinreichenden Schutz bieten. Bootsektorviren nisten sich in den ersten Sektoren von Disketten oder Festplatten ein und werden aktiv, wenn ein Betriebssystem (im folgenden: OS) von diesen gestartet werden soll.

Im Falle von Disketten erfolgt der Systemstart, indem im ROM-Bootstrap direkt auf den DOS-Bootsektor der Diskette verwiesen wird. Es wird also direkt von der gestarteten Hardware an den ersten physikalischen Sektor der Diskette weitergeleitet, von wo aus ein OS gestartet werden kann. Bei Festplatten ist der Bootvorgang nur wenig komplexer. Hier wird im Bootstrap vom BIOS aus auch zum ersten physikalischen Sektor der Festplatte gesprungen, aber hier ist der MasterBootRecord (MBR) lokalisiert, der Informationen über startbare Partitionen bereithält und direkt zum ersten logischen Sektor der exklusiv aktivierten Startpartition springt. Genauer wird dazu ein kleines Programm, der Bootloader, genutzt.

Viren könnten in diesem Zusammenhang den DOS-Bootsektor von Disketten oder den MBR von Festplatten infizieren. Dies sind sehr attraktive Ziele für Virenautoren, da das OS noch nicht geladen ist und damit manipuliert oder ganz umgangen werden kann.
Dabei ist aber zu beachten, dass Bootsektorviren unter Berücksichtigung sehr strenger technischer Limitierungen programmiert werden müssen. So ist die Größe der Startsektoren und damit die der Viren auf 444 Byte beschränkt. Innerhalb des Virus müssen dann typischerweise auch die Funktionen des Bootloaders mit implementiert werden, wenn das OS noch geladen werden soll. Hinzu kommt, da das OS noch nicht geladen ist, dass keine Routinen zum Auffinden oder Manipulieren von Dateien zur Verfügung stehen.

Zusammenfassend lässt sich feststellen, dass Bootsektorviren sehr viel technisches Wissen in der Programmierung erfordern und dann aber auch zuverlässig vom BIOS erkannt werden können. Es ist also hinreichend unwahrscheinlich, dass Virenautoren ihre Zeit künftig mit Bootsektorviren für Festplatten und Disketten aufwenden werden. Eine aktuellere Bedrohung stellt die Infektion des CD-ROM-Bootsektores durch manipulierte Images dar.

3.2.3 Companionviren

Companionviren stellen eine Besonderheit dar, da sie keine Dateien direkt infizieren oder manipulieren. Stattdessen sorgen sie dafür, dass sie alternativ zu einem bestimmten Programm gestartet werden.

Reguläre Companionviren geben sich den Namen irgendeines Zielprogramms, aber

mit veränderter Dateiendung. Unter MS-DOS kann dies etwa die .com-Endung zu einem Zielprogramm mit .exe-Endung sein. Bei Aufruf des Programms, ohne genaue Angabe der Endung, würde DOS dann immer zunächst das Programm mit der .com-Endung, also das potentielle reguläre Companionvirus, starten.

Pfad-Companionviren arbeiten auf ähnliche Weise, schreiben sich aber unter dem exakten Namen des Zielprogramms in einen Pfad, der auf dem Suchpfad dieses Zielprogramms liegt und vor diesem gefunden wird. Unter anderem kann dazu auch der DOS-PATH-Eintrag in Autoexec.bat genutzt werden. Hier sucht DOS nach Programmpfaden, wenn diese nicht im aktuellen Verzeichnis gefunden werden. Das Pfad-Companionvirus müsste hier einfach einen Eintrag zu seinem Pfad machen, der dann vor dem zu dem Zielprogrammsteht, dessen Namen es trägt.

Die einfachste Form stellen die *Surrogate-Companionviren* dar, die einfach das Zielprogramm umbenennen und sich selbst den entsprechenden Originalnamen geben.

3.3 Macintosh-Viren

3.3.1 Hintergrund: Die MAC-OS Architekturen

Beim Macintosh sind zwei grundlegende Architekturen zu unterscheiden: Der 68K Macintosh und der PowerPC Macintosh. Dabei ist der 68K ein klassischer CISC-Mikroprozessor, der auf 68K-Code arbeitet. Der PPC hingegen ist ein RISC-Mikroprozessor und ist auf eigenen PPC-Code angewiesen. Dementsprechend ist 68K-Code nicht mehr kompatibel zu aktuellen PPC-Rechnern. Hier stellt das Betriebssystem aber den Emulator 68LC040 zur Verfügung, damit der Maschinencode alter 68K-Anwendungen wenigstens softwareseitig ausgeführt werden kann.

68K-Code besteht aus der Aneinanderreihung sogenannter Resourcen, die den gesamten Programmcode abspeichern. Dabei wird zwischen Anwenderprogrammen, Metadaten (zugeordnetes ausführendes Programm, Icon, etc.) und Betriebssystemresourcen (Prozeduren zum Zeichnen von Fenstern, Menues usw.) nur durch das type-Feld der Resource unterschieden. Alle stehen in der resource-fork. In dieser steht auch an der Stelle #0 die jump-table, die Sprünge zwischen einzelnen Routinen und Segmenten realisiert. Zur jump-table wird am Ende jeder Routine zurückgesprungen, so dass zur nächsten Routine verzweigt werden kann.

Der PPC-Code ist hingegen fragmentierter Code. Hier steht der eigentliche Programmcode im sogenannten data-fork. Für Metadaten und Betriebssystemressourcen wird ein resource-fork hinzugebunden und auch shared-libraries sind nutzbar. Die Indizierung dieser verschiedenen Fragmente geschieht ähnlich der jump-table mit der cfrg (CodeFragmentResource).

Viren existieren direkt nur für 68K-Code. Allerdings kann dieser auch im resource-fork von PPC-Code Anwendungen mit eingebunden sein, wodurch der PPC indirekt immer mitbetroffen ist. Ferner beinhaltet jede PPC-Anwendung wenigstens einen kleinen 68K-Codeabschnitt, der auf 68K-Rechnern die Meldung liefert, dass das Programm nicht ausgeführt werden kann. Damit ist zumindest eine weitere theoretische Angriffsstelle des

PPC-Codes identifiziert. Ebenso ist es denkbar, dass die shared-libraries viralen Code enthalten. Derartige Viren sind aber noch nicht veröffentlich worden.

Die *Infektionsmöglichkeiten* sind ähnlich denen des IBM-PC. Sicherlich kann ein Virus einfach eine neue Ressource zum Code hinzufügen und die jump-table derart modifizieren, dass beim Programmstart zu dieser gesprungen wird. Wurde der Originaleintrag der jump-table gesichert, kann nach der Ausführung des Virus das Programm regulär ablaufen. Dieses Vorgehen ist vergleichbar mit dem von Prependerviren beim IBM-PC. Alternativ kann ein Virus seinen Code auch an das Ende einer bereits eingebundenen Ressource anfügen. Die jump-table kann dann unverändert bleiben, aber die ersten Bytes der infizierten Ressource müssen gesichert und überschrieben werden. Hier kann dann bei Ausführung der Ressource ein Sprung zum viralen Code am Ende realisiert werden. Das Virus kann anschliessend die Originalstartsequenz wiederherstellen und auch hier das Programm unbeeinflusst laufen lassen. Das entspricht dem Konzept eines Appendervirus beim IBM-PC.

Die dritte Möglichkeit kombiniert die ersten beiden Alternativen. Es wird dabei die jump-table derart modifiziert, dass innerhalb einer infizierten Ressource an die Stelle des Virencodes gesprungen wird. Das hat den Vorteil, dass zur regulären Programmausführung nicht die Startsequenz der infizierten Ressource gespeichert werden muss, sondern nur der erste Eintrag der jump-table.

3.4 Virenschutz

Im Rahmen des *Virenschutzes* sollen die Virenscanner und die Integritätsprüfer genau betrachtet werden. Es wird in beiden Kapitel klar, dass die jeweiligen Konzepte für sich genommen keinen umfassenden Schutz bieten können. Maximale Absicherung kann erst durch ein System mit Virenscanner, mit Signatursuche und generischer Entschlüsselung, und Integritätsprüfer bereitsgestellt werden.

3.4.1 Virenscanner

Virenscanner verfolgen das Konzept der Signatursuche in potentiell befallenen Dateien. Signaturen sind Codeabschnitte, die für ein bestimmtes Virus charakteristisch sind und in regulärem Code ausgeschlossen werden können. Dabei muss spätestens seit Aufkommen der Cavityviren, die ihren Schadcode an einer beliebigen Stelle im Programm unterbringen können, immer die gesamte Datei abgesucht werden.

Virenscanner können und sollten in zwei Betriebsmodi angewandt werden. Im on-demand-Modus durchsucht der Scanner jede ausführbare Datei auf der Festplatte nach Virensignaturen. Da dieser Vorgang viel Zeit benötigt, sollten Virenscanner auch immer im on-access-Modus im Hintergrund laufen und jede Datei prüfen, die ausgeführt und in den Speicher geladen wird.

Da Virenscanner typischerweise auf eine Datenbank mit Virensignaturen angewiesen sind, können unbekannte Viren nicht ohne weiteres erkannt werden, wenn für sie noch keine Signatur bereit steht. An dieser Stelle setzen Heuristiken an, die versuchen, Viren anhand ihres Verhaltens zu erkennen. Aktuell arbeiten Heuristiken aber noch sehr unzu-

verlässig, mit Erkennungsraten von 10-60%. Da Viren ihren Code aber zukünftig immer schneller verändern können werden, wird der Einsatz von Heuristiken zur Erkennung zunehmend wichtiger.

Das Problem nicht erkennbarer Signaturen besteht bei polymorphen Viren, die ihren Hauptteil verschlüsseln, so dass auch bekannte Signaturen verborgen werden. Ein interessanter Ansatz für die Lösung dieses Problems findet sich in [10] mit der *generischen Entschlüsselung*. Hier ist die Grundidee, dass verschlüsselte Programme sicher eine Entschlüsselungsroutine besitzen, die aktiv wird und den Virencode entschlüsselt, wenn das befallene Programm gestartet wird. Der Ansatz sieht nun also vor, zu prüfende Programme in einer `sandbox` zu starten, in der sie keinen Schaden anrichten können. Hier werden sie auch nicht durch die `CPU` ausgeführt, sondern durch einen `CPU`-Emulator, der mit einem entsprechenden Emulationskontrollmodul zusammen arbeitet. Das Kontrollmodul unterbricht den Emulator regelmäßig und der Programmcode kann dann nach entstandenen Virensignaturen abgesucht werden. Es ist aber klar, dass dieser Ansatz nicht performant für alle ausfühbaren Dateien durchgeführt werden kann. Ein Einsatz im Zusammenhang mit Heuristiken eines Virenscanners ist hier empfehlenswert, da nur „verdächtige“ Dateien geprüft werden müssen.

3.4.2 Integritätsprüfer

Integritätsprüfer basieren auf der Annahme, dass Viren die Dateien, die sie infizieren, auch verändern. Im Sinne einer kryptographischen Hashfunktion wird für jede ausfühbare Datei, bei Installation des Integritätsprüfers oder bei Insallation neuer Programme, ein charakteristischer Wert gebildet. Dieser Wert muss sich ändern, sobald auch nur ein einziges Bit der zugehörigen Datei verändert wird. Im Rahmen einer Integritätsprüfung kann nun festgestellt werden, ob Dateien verändert wurden, ob also eine potentielle Infektion stattgefunden hat. Der große Vorteil im Vergleich zu Virenscannern ist, dass Integritätsprüfer nicht auf die Kenntnis aktueller Virensignaturen angewiesen sind. Der Nachteil ist aber, dass Dateien auch schon vor der Installation des Integritätsprüfers infiziert sein konnten, also trotz dieser als integer angenommen werden. Außerdem besteht natürlich auch die Möglichkeit, dass ein Anwender Dateien ändert, wenn er etwa ein Update einspielt. Dies wird zwangsläufig dazu führen, dass der Integritätsprüfer eine Infektion vermutet, die der Nutzer aber im Bewußtsein seiner Änderung, ignorieren wird. Zum gleichen Zeitpunkt ist es nun aber denkbar, dass ein Virus aktiv wird und die entsprechende Datei auch infiziert. Ferner kann beispielweise ein Update auch viralen Code enthalten. In beide Szenarien werden die meisten Nutzer jegliche Warnmeldungen ignorieren. Integritätsprüfer allein können also auch keinen umfassenden Virenschutz bieten.

Literatur

[1] Blum, M. und Micali, S., *How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits*, SIAM J. Computing, Kapitel 13(4), Seiten 850-864; Siam, 1984.

[2] Doraswamy und Harkins; *IPSec, The New Security Standard for the Internet, Intranets, and Virtual Private Networks*; Prentice Hall, 2003

[3] Ferguson, N. und Schneier,B., *A Cryptographic Evaluation of IPsec*; http://www.schneier.com/paper-ipsec.html, 1998

[4] IETF; *RFC2409: The Internet Key Exchange (IKE)*; http://tools.ietf.org/html/rfc2409; 1998

[5] IETF; *RFC2408: Internet Security Association and Key Management Protocol (ISAKMP)*; http://tools.ietf.org/html/rfc2408; 1998

[6] IETF; *RFC4303: IP Encapsulating Security Payload (ESP)*; http://tools.ietf.org/html/rfc4303; 2005

[7] IETF; *RFC4302: Authentication Header*; http://tools.ietf.org/html/rfc4302; 2005

[8] IETF; *RFC2401:* ; http://tools.ietf.org/html/rfc2401; 1998

[9] Pieprzyk, Hardjono und Seberry; *Fundamentals of Computer Security*; Kapitel 18; Springer-Verlag, 2003

[10] Stallings, W.; *Cryptography and Network Security Principles and Practices*; Kapitel 4, Abschnitt 19.2; Prentice Hall, 2003

[11] Wätjen, D.; *Kryptographie*; Spektrum, 2004