Technische Universität München

Institut für Informatik

Sichere TCP/IP–basierte Kommunikation bei der BMW AG

Helmut Reiser

Technische Universität München

Institut für Informatik

Diplomarbeit

Sichere TCP/IP–basierte Kommunikation bei der BMW AG

Bearbeiter:	Helmut Reiser
Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Dr. Bernhard Neumair
	Stephen Heilbronner
	Norbert Jungfleisch (BMW AG)
Abgabedatum:	15. August 1997

Ich versichere, daß ich diese Diplomarbeit selbständig verfaßt und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 10. August 1997

Helmut Reiser

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Die BMW AG betreibt ein weltweites Unternehmensnetz auf der Basis des TCP/IP–Protokolls. Dieses Intranet wird vom Netz der Konzernzentrale in München, den Netzen der Produktions– und Entwicklungszentren und den Netzen der Tochter– und Beteiligungsgesellschaften gebildet. In zunehmendem Maße werden Telearbeitsplätze eingerichtet, die über Modem– bzw. ISDN–Verbindungen auf das Intranet zugreifen. Künftig wird es auch ein Extranet geben, bei dem zunächst alle deutschen Händler vernetzt und an das Intranet angeschlossen werden.

Auf das Netz der BMW AG haben unterschiedlichste Benutzergruppen Zugriff, und in zunehmendem Maße werden sensible Daten über leicht angreifbare Netzstrukturen übertragen.

1.1 Ziele der Arbeit

Im Rahmen dieser Arbeit wird versucht, den Begriff der sicheren Kommunikation näher zu fassen. Wegen der verschiedenen Gruppen, die das BMW–Netz benutzen, wird die Möglichkeit von IP Version 6 (IPv6) zur Bildung von geschlossenen Benutzergruppen bzw. von Tunneling untersucht. Es werden Protokolle betrachtet, die auf TCP/IP aufsetzen und in der Lage sind, höheren Schichten Dienste zur Durchsetzung von Sicherheitsanforderungen anzubieten. Bei der Absicherung auf Ebene der Anwendungsschicht werden exemplarisch Dienste, die über die Protokolle SMTP und HTTP abgewickelt werden, untersucht. Daneben müssen Wählverbindungen sowie Sitzungen an entfernten Rechnern, die mittels Telnet, rlogin, rsh o.ä. aufgebaut werden, vor Angriffen geschützt werden.

Dazu werden existierende Protokolle bzw. Produkte betrachtet und auf ihre Einsatzmöglichkeit bei der BMW AG hin beurteilt. Ein wichtiges Differenzierungsmerkmal ist dabei die Standardisierung der Protokolle sowie die Verfügbarkeit kommerzieller Produkte. Bei einem derart großen Netzwerk sind aber auch Managementaspekte, wie Installations–, Administrations–, Infrastruktur– und Betriebsaufwand neben Performance– und Kostenfragen zu betrachten. Nicht zuletzt ist auch die Frage nach der Benutzerakzeptanz zu stellen.

Nicht untersucht werden Fragen der Firewallproblematik sowie Fragen zur Sicherung und Administration von Endsystemen. Ein Sicherheitskonzept für den BMW Internet Zugang über einen Firewall wurde umfassend in einer früheren Arbeit [Hau95] untersucht.
Da die Endsysteme in lokaler Verantwortlichkeit administriert werden und die Arbeit sich mit TCP/IP–basierter Kommunikation beschäftigt, wird die physikalische und administrative Sicherung von Endsystemen nicht näher betrachtet.

1.2 Aufbau der Arbeit

In Kapitel 2 werden die verschiedenen Dimensionen der sicheren Kommunikation herausgearbeitet, die zur späteren Betrachtung und Bewertung von Protokollen benötigt werden. Der „Raum" der sicheren Kommunikation wird dabei in die vier Dimensionen Sicherheitsanforderungen, Sicherheitsbedrohungen, Dimension der Kommunikationspartner sowie architekturelle Dimension unterteilt.

Insbesondere für die Dimension der Sicherheitsanforderungen wurden bereits viele Realisierungen entwickelt, um die unterschiedlichsten Sicherheitsanforderungen in Rechen– und Kommunikationssystemen durchzusetzen. Diese grundlegenden Konzepte werden in Kapitel 3 vorgestellt. Es handelt sich dabei um Lösungen für einzelne, spezielle Anforderungen, die in den, in späteren Abschnitten zu betrachtenden Protokollen und Produkten in Kombination verwendet werden, um die Kommunikation zu sichern.

Ziel dieser Arbeit ist die Betrachtung einer sicheren TCP/IP–basierten Kommunikation im Umfeld eines Unternehmens. Dazu muß die Netzstruktur des Unternehmens bekannt sein. Im Abschnitt 4 wird daher die Struktur des Corporate Network der BMW AG vorgestellt.

Die Grundlage jeder Aktion und jedes Projektes zur Verbesserung oder Bewertung der Sicherheit ist eine unternehmensweite Sicherheitspolitik. In Kapitel 5 wird der Begriff der Sicherheitspolitik erläutert. Drüberhinaus wird ein Vorgehensmodell zur Erstellung sowie ein Beispiel einer Sicherheitspolitik vorgestellt.

Bei nahezu allen Verfahren, die zur sicheren Kommunikation eingesetzt werden, sind kryptologische Protokolle oder diverse Verschlüsselungsverfahren implementiert, die Schlüssel oder Zertifikate benötigen. In Kapitel 6 wird die Problematik des Schlüssel– und Zertifikatsmanagements erläutert. Hierunter fällt die Generierung, Speicherung, Verteilung und der Widerruf von Schlüsseln bzw. Zertifikaten.

In Kapitel 7 wird eine Möglichkeit zur Bildung von geschlossenen Benutzergruppen vorgestellt. Im Rahmen der Einführung von IPv6 wurden die beiden Protokolle IP Authentication Header und IP Encapsulation Security Payload spezifiziert, die Tunneling und damit geschlossene Benutzergruppen ermöglichen.

In den folgenden beiden Kapiteln werden Protokolle dargestellt, die architekturell unterhalb der Anwendungsschicht und oberhalb von TCP liegen und Anwendungen eine Schnittstelle bieten, um bestimmte Sicherheitsanforderungen durchzusetzen. Kapitel 8 beschäftigt sich mit Secure Socket Layer (SSL), das ursprünglich zur Sicherung des WWW–Verkehrs entwickelt wurde.

Im Gegensatz dazu, wurde das in Kapitel 9 betrachtete Secure Shell (SSH) primär zur Sicherung des Zugriffs auf entfernte Rechner konzipiert. Es soll insbesondere die sog. „r–Dienste" mit deren Sicherheitsrisiken ersetzen. In diesem Kapitel wird daher auch die Sicherung des Log on auf einem entfernten System untersucht. Da sowohl SSH als auch SSL als allgemeinere Protokolle spezifiziert sind, wird auch der Einsatz von SSH zur Sicherung des HTTP–Verkehrs untersucht. Abgeschlossen wird die Betrachtung mit einer Bewertung und dem Vergleich von SSH und SSL.

Als Beispiele für Kommunikationsprotokolle, die auf TCP/IP aufsetzen, wurde HTTP und damit der WWW–Verkehr sowie die elektronische Post (E-Mail) mittels SMTP und MIME ausgewählt. Die Sicherung der HTTP–Kommunikation wird in Kapitel 10 betrachtet. Untersucht wird dabei der Einsatz des Secure Hypertext Transfer Protocols (S-HTTP) und des Hypertext Transfer Protocols over SSL (HTTPS), das auf das im Kapitel 8 betrachtete Secure Socket Layer aufsetzt.

In Kapitel 11 wird die Sicherung elektronischer Post untersucht. Dabei werden die Protokolle Privacy Enhancement for Internet Electronic Mail (PEM) und Security Services for MIME (S/MIME) und die Anwendung Pretty Good Privacy (PGP) vorgestellt, bewertet und verglichen.

In einem Unternehmen, wie der BMW AG, steigt die Zahl von Telearbeitsplätzen, d.h. von Mitarbeitern, die ganz oder teilweise zu Hause arbeiten. Diese Mitarbeiter müssen über Wähl– bzw. ISDN–Verbindungen an das Unternehmensnetz angeschlossen werden. Kapitel 12 beschäftigt sich mit Remote Access Servern bzw. untersucht Protokolle, um diese Wählverbindungen zu sichern. Charakterisiert werden die Protokolle TACACS+ und RADIUS und die Anwendung RAS.

Um untersuchte Protokolle und Verfahren auf die Einsatzmöglichkeit bei der BMW AG hin evaluieren zu können, müssen diese getestet werden. Bei der BMW AG wurde dafür eine Testplattform aufgebaut, auf der im Rahmen dieser Arbeit die Protokolle SSL, S/MIME und SSH sowie die Anwendung RAS getestet wurden. Die Testplattform und die durchgeführten Tests werden in Kapitel 13 beschrieben.

Von besonderem Interesse ist bei einer derartigen Untersuchung die Frage, welche Protokolle und Produkte nun tatsächlich im Umfeld der BMW AG eingesetzt werden sollen. Diese Fragestellung wird in Kapitel 14 behandelt. Betrachtet werden hier die konkreten Anwendungsszenarien für die entfernten Log on's, für WWW– und E-Mail Kommunikation und die Sicherung von Wählverbindungen. Für diese Problemkreise werden konkrete Protokolle empfohlen, es werden Installations– und Migrationsstrategien vorgeschlagen und Betriebsaspekte erörtert. In diesem Abschnitt wird die Frage des Schlüssel– und Zertifikatsmanagements, im Hinblick auf die empfohlenen Protokolle, nochmals aufgegriffen.

Das abschließende Kapitel 15 enthält eine Zusammenfassung der wichtigsten Ergebnisse und einen Ausblick auf mögliche weiterführende Arbeiten.

Im Anhang werden verwendete Begriffe aus der Kryptologie erläutert und mit IDEA und RSA je ein Vertreter eines asymmetrischen bzw. symmetrischen Verschlüsselungsverfahrens vorgestellt. Als Vertreter eines Message Authentication Code wird SHA erläutert.

Kapitel 2

Dimensionen der sicheren Kommunikation

Der Begriff der *sicheren Kommunikation* läßt sich nicht allgemeingültig definieren, sondern wird von einer Vielzahl von zu betrachtenden Kriterien bestimmt. Diese Kriterien lassen sich selbst wieder zu Dimensionen zusammenfassen, die einen mehrdimensionalen Raum aufspannen. Ein Unterraum dieses Raumes bestimmt für den jeweils zu betrachtenden Einzelfall den Begriff der sicheren Kommunikation und determiniert die für diesen Fall anzuwendenden Maßnahmen (vgl. Abb. 2.1). Auch die Sicherheitspolitik (vgl. Abschn. 5) als Grundlage für alle Maßnahmen, die zur Verbesserung der Kommunikationssicherheit ergriffen werden, muß sich mit jeder dieser Dimensionen und mit den Kriterien innerhalb der Dimensionen auseinandersetzen.

Jede Kommunikationsbeziehung stellt bestimmte Anforderungen bezüglich der Sicherheit (Dimension der Sicherheitsanforderungen) und muß vor Angriffen (Dimension der Bedrohungen) geschützt werden. Diese beiden Dimensionen bestimmen die Sicherheitsklasse der Kommunikationsbeziehung. Die Maßnahmen zur Sicherung hängen von den Kommunikationspartnern, die die Verbindung unterhalten (Dimension der Kommunikationspartner), ab. Auch die Schicht, in der die Maßnahmen greifen (Architekturelle Dimension), ist zu betrachten und zu bewerten.

2.1 Dimension der Sicherheitsanforderungen

Der Funktionsbereich, der die *Sicherheitsanforderungen* untersucht, bestimmt die technischen und kryptographischen Maßnahmen, die ergriffen werden müssen, um eine Verbindung abzusichern.

Als Sicherheitsanforderungen werden im Kontext dieser Arbeit die Begriffe *Authentication* (Authentisierung), *Access Control* (Zugriffskontrolle), *Confidentiality* (Vertraulichkeit), *Integrity* (Integrität), *Non–Repudiation* (Verbindlichkeit) und *Auditing* betrachtet. Abhängig von der jewei-

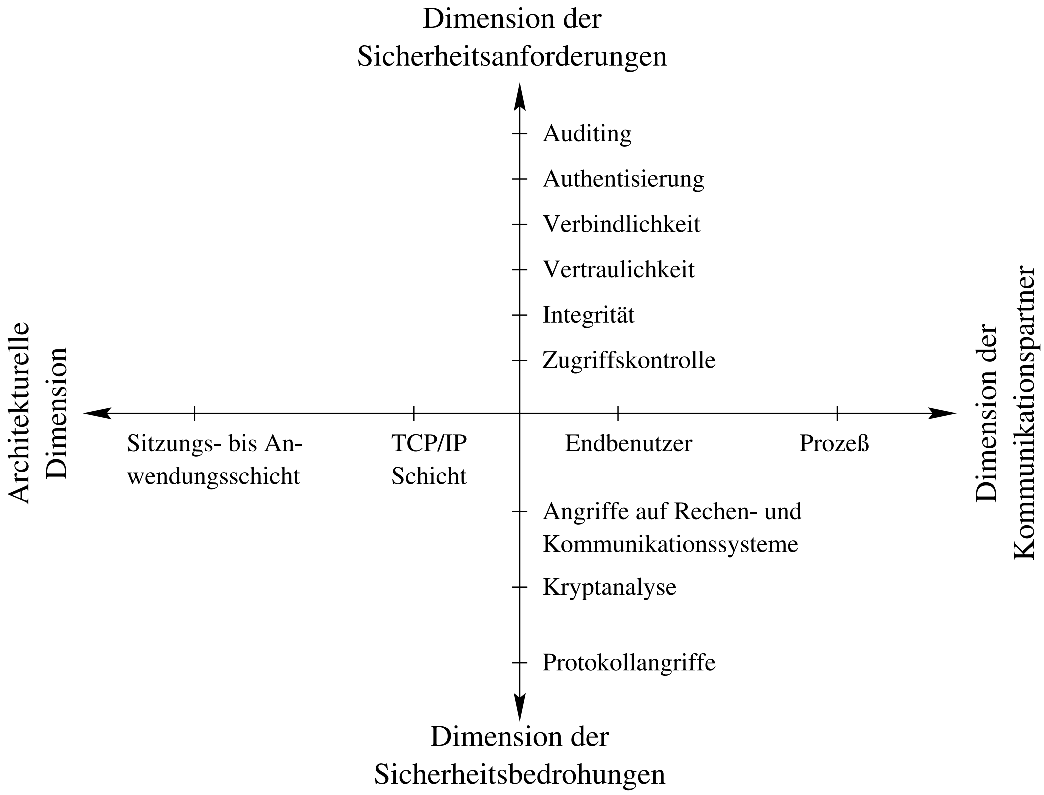

Abbildung 2.1: Dimensionen der sicheren Kommunikation

ligen Sicherheitsklasse, innerhalb derer die Kommunikation stattfindet, müssen eine Teilmenge oder alle diese Sicherheitsanforderungen erfüllt werden.

2.1.1 Authentication (Authentisierung)

Die *Authentisierung* liefert die Gewißheit über die Identität des Kommunikationspartners und damit die Sicherheit, daß die empfangenen Daten auch vom angegebenen Sender stammen. Wenn beide Kommunikationspartner sich gegenseitig authentisieren, so spricht man von zweiseitiger, andernfalls von einseitiger Authentisierung.

Zur Durchsetzung benötigt man sowohl Maßnahmen zur Identifikation als auch zur eigentlichen Authentisierung. Für die Identifikation werden wohldefinierte Eigenschaften benötigt, so daß ein Objekt, insbesondere ein Kommunikationspartner über zweifelsfreie Merkmale (z.B. Benutzer–ID) eindeutig identifiziert werden kann. Bei der Authentisierung muß ein Objekt die Korrektheit der von ihm behaupteten Identität nachweisen. Nur wenn die Identität eines Benutzers zweifelsfrei feststeht, können an den Benutzer gekoppelte Berechtigungsprofile (vgl. Abschnitt 3.2) greifen.

2.1.2 Access Control (Zugriffskontrolle)

Die *Zugriffskontrolle* verhindert die unberechtigte Nutzung von Ressourcen. Sie schließt auch die (berechtigte) Nutzung einer Ressource in unberechtigter Art und Weise aus. Hat sich ein Partner erfolgreich authentisiert, so bestimmt die Zugriffskontrolle den Umfang und die Art seiner Zugriffsrechte.

2.1.3 Confidentiality (Vertraulichkeit)

Vertraulichkeit ist gewährleistet, wenn Daten vor unberechtigtem Ausspähen geschützt sind und die übertragenen Informationen nur vom rechtmäßigen Empfänger verwertet werden können. Für einen Dritten ist es nicht möglich an die Informationen zu gelangen.

2.1.4 Integrity (Integrität)

Eine Nachricht erfüllt die Forderung nach *Integrität*, falls es nicht möglich ist, die Nachricht auf dem Weg vom Sender zum Empfänger zu verändern, ohne daß der Empfänger dies erkennen kann. Außerdem muß es für den Empfänger möglich sein Replay Angriffe (vgl. Abschnitt 2.2.1) zu erkennen.

2.1.5 Non–Repudiation (Verbindlichkeit)

Die *Verbindlichkeit* ist vergleichbar mit einer Unterschrift auf einem Dokument. Sie erfüllt die folgenden Funktionen [HT97].

1. Eine Unterschrift kann nicht gefälscht werden und die Unterschrift ist dauerhaft (*Perpetuierungsfunktion*).
2. Die Unterschrift ist authentisch (*Echtheitsfunktion*).
3. Die Unterschrift kann nicht wiederverwendet werden.
4. Das unterschriebene Dokument kann später nicht verändert werden (*Abschlußfunktion*).
5. Der Unterzeichner kann die Unterschrift nicht leugnen (*Beweisfunktion*).[1]

Das Äquivalent zur Unterschrift auf Papier wird in der digitalen Informationsverarbeitung als *Digital Signature* bezeichnet.

[1] In der Realität ist keine dieser Funktionen vollständig erfüllt. Eine Unterschrift auf Papier kann gefälscht und von einem Stück Papier auf ein anderes übertragen werden. Das unterschriebene Dokument kann nachher verändert werden und der Unterzeichner kann seine Unterschrift später leugnen. Trotzdem wird die Unterschrift im Rechtsverkehr akzeptiert, da ihre Funktion durch Rahmenbedingungen gesichert wird. Dabei wird angenommen, daß Betrügen schwierig, und das Risiko der Entdeckung und Bestrafung zu hoch ist.

2.1.6 Auditing und Logging

Auditing und *Logging* liefern Daten, um den Netzverkehr zu überwachen, die Sicherheit zu kontrollieren und evtl. Einbrüche bzw. Angriffe auf das System zu erkennen. Dabei werden Verbindungsdaten wie z.B. welche Aktionen der Nutzer ausführt, von wo er auf das System zugreift u.ä. protokolliert und gespeichert.

2.2 Dimension der Sicherheitsbedrohungen

Um die konkrete Gefahr, der ein System ausgesetzt ist, bestimmen zu können, sind die potentiellen *Sicherheitsbedrohungen* zu betrachten. Die Sicherheit von Rechenanlagen, Protokollen, Kommunikations- und Kryptosystemen kann durch Angriffe (*Attack*) bedroht werden.
Unter einem *Angriff* versteht man einen nicht autorisierten Zugriff, bzw. bereits den Versuch eines solchen, auf ein System. Die Angriffe lassen sich wie in Abb. 2.2 klassifizieren.

Grundsätzlich sind Angriffe von „innen“, d.h. von Benutzern, die Rechte auf Systemen innerhalb des Unternehmens haben und versuchen diese unberechtigterweise zu erweitern, von Angriffen von „außen“, d.h. von Personen die nicht zum Unternehmen gehören oder versuchen über Netzzugangspunkte unberechtigterweise Zugriff auf das Unternehmensnetz zu erhalten, zu unterscheiden. Dabei werden Angriffe auf Rechen– bzw. Kommunikationssysteme von Angriffen auf Kryptosysteme unterschieden. Innerhalb dieser Gruppen wird in passive und aktive Angriffe unterteilt.

2.2.1 Angriffe auf Rechen– und Kommunikationssysteme

Bei Angriffen auf Endsysteme, Netzeinrichtungen oder die Kommunikation zwischen zwei (oder mehreren) Partnern, unterscheidet man zwischen passiven und aktiven Angriffen.

- **Passiver Angriff**

  Unter den Begriff des *passiven Angriffs* fallen alle Verfahren, mittels derer der Angreifer versucht, unrechtmäßig in den Besitz von Daten zu gelangen, ohne diese verändern zu können. Passive Angriffe lassen sich wie folgt weiter unterteilen.

  - **Wiretapping (Abhören des Netzverkehrs)**: Der Angreifer kann den privaten Nachrichtenaustausch von Nutzern abhören.
  - **Lesen von Daten auf Speichermedien:** Der Angreifer besitzt die Möglichkeit, unautorisiert auf gespeicherte Daten bzw. ganze Speichermedien lesend zuzugreifen.
  - **Traffic Analysis (Verkehrsflußanalyse)**: Eine nicht autorisierte Person erlangt durch die Beobachtung anderer Nutzer Kenntnisse über deren Kommunikationsgewohnheiten und kann damit u.U. *Benutzerprofile* erstellen, d.h. der Angreifer weiß, wer mit wem, wie und wann kommuniziert.

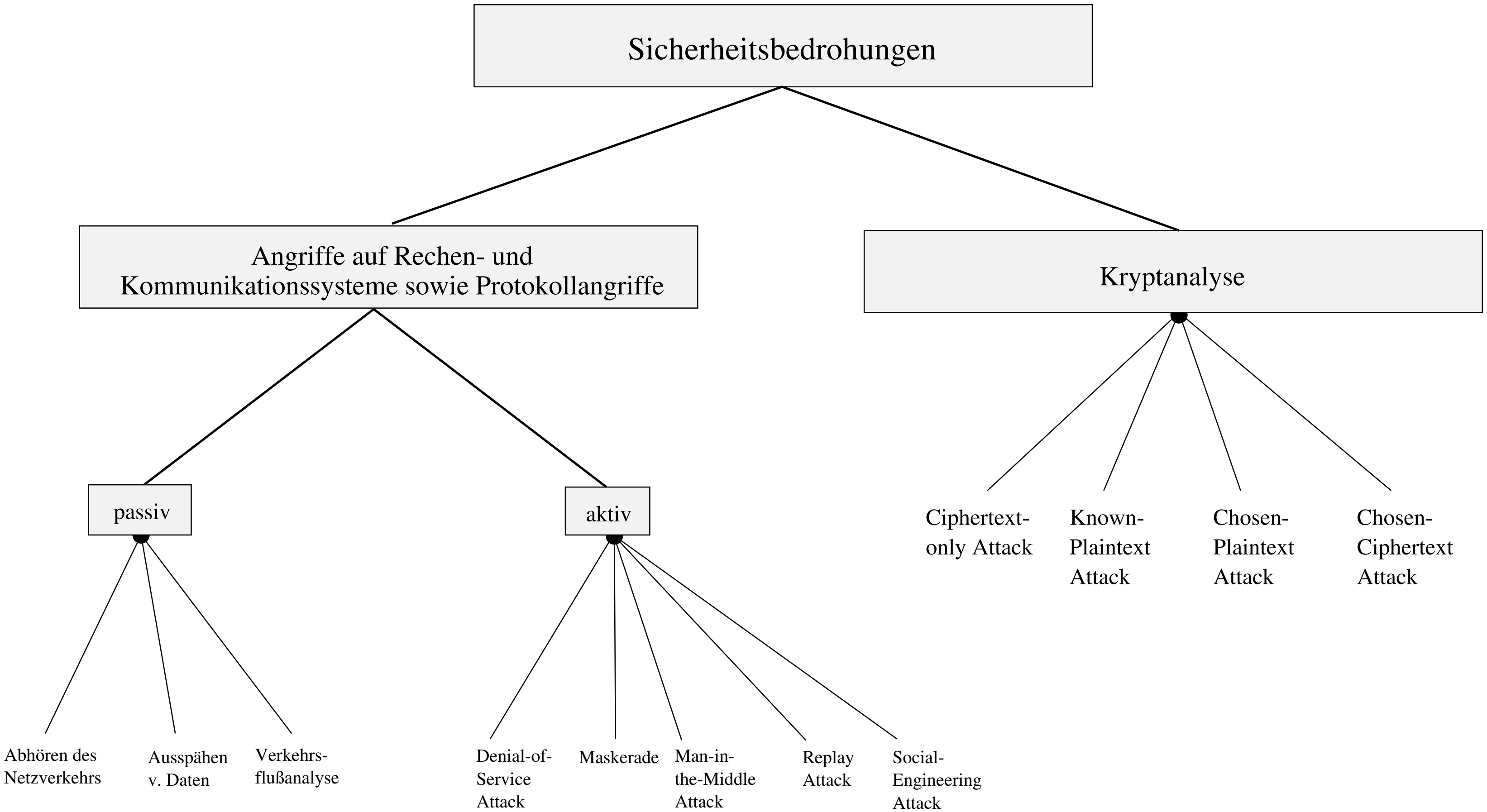

Abbildung 2.2: Klassifikation der Sicherheitsbedrohungen

- **Aktiver Angriff**

  Versucht der Angreifer zusätzlich Daten zu verändern, zu entfernen, selbst zu erzeugen oder die Verfügbarkeit von Ressourcen zu stören, so spricht man von *aktiven Angriffen* (*Tampering*). Hierunter fallen

  - **Denial–of–Service Attack**: Der Angreifer versucht dabei die Verwendbarkeit bzw. die Performance von Netz– oder Systemressourcen einzuschränken. Im schlimmsten Fall kann der berechtigte Nutzer nicht mehr auf das System zugreifen oder einen bestimmten Dienst nicht mehr in Anspruch nehmen. Beispiele hierfür wären das „Überfluten“ eines Rechners mit Nachrichten oder die Überlastung eines Rechners (*Overload–Attack*).
  - **Masquerade (Maskerade)**: Der Angreifer versucht einem System oder einem Kommunikationspartner eine andere Identität vorzuspiegeln. Hierunter fällt auch *IP–Spoofing*, bei dem der Angreifer die Absender–IP–Adresse verändert, um dadurch den Anschein zu erwecken, daß das IP–Paket von einem anderen Rechner kommt. Eine weitere Art der Maskerade sind *Trojanische Pferde*. Dabei handelt es sich um Programme, die neben ihren spezifizierten Systemfunktionen noch zusätzliche versteckte (Schad–) Funktionalitäten enthalten. Ein Beispiel wäre ein verändertes Login–Programm, das alle eingegebenen Paßwörter speichert.
  - **Man-in-the-Middle Attack**: Unter einer Man-in-the-Middle Attack faßt man das Unterbrechen einer Verbindung, das Einfügen, Löschen und Verändern von Nachrichten, sowie das Zurückschicken oder Umleiten von Nachrichten zusammen. Dieser Angriff kann auch als Spezialfall der Maskerade aufgefaßt werden. *Alice*[2] und *Bob* wollen kommunizieren. *Mallet* greift die Kommunikation an, indem er die Nachrichten abfängt und sich gegenüber Alice als Bob und gegenüber Bob als Alice ausgibt (vgl. auch S. 36).
  - **Replay Attack (Wiedereinspielung)**: Dabei versucht ein Angreifer eine abgehörte Nachricht zu einem späteren Zeitpunkt nochmals (evtl. in veränderter Form) an den rechtmäßigen Empfänger zu senden und dadurch eine neue Nachricht bzw. eine neue Verbindung vorzutäuschen. Ein Beispiel für einen derartigen Angriff ist das Abhören eines (Klartext–) Paßworts und eine spätere Maskerade mit Hilfe dieses Paßworts.
  - **Social–Engineering Attack**: Der Angreifer versucht durch die Täuschung von berechtigten Nutzern in den Besitz von Zugangsberechtigungen oder Paßwörtern zu gelangen. Der Angreifer könnte die Nutzer bspw. anrufen und sich als Systemadministrator ausgeben, um die Nutzer zu bitten, ihre Paßwörter auf ein von ihm vorgegebenes zu ändern.

[2]Zur Erläuterung von Protokollabläufen, bzw. Protokollangriffen werden traditionsgemäß verschiedene imaginäre Personen verwendet. So eröffnet bspw. immer Alice die Kommunikation mit Bob. Ein Angreifer, der den Verkehr abhört, wird mit *Eve* (*eaversdropper*) bezeichnet. Derjenige, der einen aktiven Angriff durchführt, heißt traditionell Mallet (*malicious*) [Sch94].

2.2.2 Angriffe auf Kryptosysteme

Zur Durchsetzung von Sicherheitsanforderungen werden Verschlüsselungsverfahren und kryptographische Protokolle (z.B. zum sicheren Schlüsselaustausch) eingesetzt. Auch diese Kryptosysteme können angegriffen werden, man unterscheidet die *Kryptanalyse (Cryptanalysis)* und *Protokollangriffe (Protocol Attack)*.

- **Cryptanalysis (Kryptanalyse)**

  Der Angreifer versucht dabei eine verschlüsselte Nachricht zu „brechen“. Der Versuch des unberechtigten Entschlüsselns wird als Angriff bezeichnet. Ein erfolgreicher Angriff heißt *Methode*. Bei der Untersuchung und Klassifizierung der Kryptanalyse wird angenommen, daß der Angreifer den Verschlüsselungsalgorithmus kennt, da ein Verfahren, das von der Geheimhaltung des Algorithmus abhängt, im allgemeinen als nicht ausreichend sicher angesehen wird.
  Die Angriffsarten gegen Verschlüsselungsverfahren werden im folgenden nach zunehmender Mächtigkeit des Verfahrens angeordnet.

  - **Ciphertext-only Attack**: Der Kryptanalyst ist im Besitz von chiffrierten Texten und versucht damit den Klartext bzw. den Schlüssel abzuleiten.
  - **Known-Plaintext Attack**: Der Angreifer kennt neben den verschlüsselten Texten auch die dazugehörigen Klartexte und versucht den Schlüssel zu brechen.
  - **Chosen-Plaintext Attack**: Hierbei kann der Kryptanalyst beliebige Klartextblöcke wählen und diese mit dem zu brechenden Schlüssel chiffrieren lassen.
  - **Chosen-Ciphertext Attack**: Der Angreifer kann verschiedene verschlüsselte Texte wählen und diese mit dem zu brechenden Schlüssel dechiffrieren lassen.

  Die Angriffsarten selbst sagen dabei noch nichts über das konkrete Vorgehen des Angreifers aus. Die einfachste i.d.R. aber auch ineffizienteste Methode wird als *Brute–Force Attack* bezeichnet. Der Angreifer versucht dabei systematisch alle möglichen Schlüssel (bzw. alle möglichen Klar– oder Geheimtexte) durchzuprobieren.

- **Protocol Attack (Protokollangriff)**

  Die Einteilung der Protokollangriffe erfolgt analog zu den Angriffen auf Rechensysteme (vgl. Abschnitt 2.2.1).

2.3 Architekturelle Dimension

Die zu schützende Kommunikationsbeziehung wird innerhalb des OSI–Modells betrachtet. Es muß festgelegt werden, in welcher Schicht Maßnahmen zur Sicherung durchzuführen sind. Dies ist notwendig, da die technischen Erfordernisse und die technischen Möglichkeiten je nach betrachteter Schicht sehr unterschiedlich sein können. Außerdem bestimmt sich die Transparenz der eingesetzten Maßnahmen für den Benutzer unter anderem auch aus der architekturellen Dimension, innerhalb derer die Maßnahme zur Sicherung der Verbindung zu tragen kommt.

Würden auf Seite der Anforderungen bspw. nur die Vertraulichkeit und damit Verschlüsselungstechniken betrachtet, so könnte man explizit für jede Schicht die technischen Möglichkeiten untersuchen. Für komplexere Sicherheitsanforderungen ist diese Einteilung jedoch zu fein.

Da sich die Arbeit mit TCP/IP–basierter Kommunikation beschäftigt, werden die physikalische Schicht und die Sicherungsschicht außer acht gelassen. Die restlichen Schichten werden zu zwei Klassen zusammengefaßt. Einerseits wird die TCP/IP–Ebene betrachtet, andererseits werden Maßnahmen, die im Bereich der oberen Schichten und hier insbesondere solche, die in der Anwendungsschicht greifen, erörtert.

2.4 Dimension der Kommunikationspartner

Um die Sicherheit einer Kommunikationsbeziehung bewerten bzw. um überhaupt geeignete Maßnahmen zur Sicherung der Verbindung auswählen zu können, müssen die Kommunikationspartner, die an den Endpunkten einer Verbindung stehen, betrachtet werden. Innerhalb dieser Dimension können Endbenutzer und Prozesse auf End– oder auf Transitsystemen als mögliche Kommunikationspartner auftreten, die in beliebigen Konstellationen untereinander kommunizieren.

Die Partner determinieren die einzusetzenden Verfahren. Soll bspw. ein Prozeß als Sender oder Empfänger auftreten, so stellt sich die Frage, inwieweit einzusetzende Produkte eine automatische Verarbeitung zulassen.

Ein Endbenutzer stützt sich, um an einer Kommunikation teilnehmen zu können, letztlich immer auf Prozesse, indem er deren Dienste aufruft. Allerdings bestimmt die Dienstschnittstelle des Prozesses, die Eingriffs– und Wahlmöglichkeiten des Benutzers und damit auch seine Rolle hinsichtlich sicherheitsrelevanter Fragestellungen. Eine Möglichkeit ist, daß der Benutzer alle sicherheitstechnischen Parameter und die Sicherheitsanforderungen selbst bestimmen kann und deswegen die Mechanismen sehr gut kennen und verstehen muß, um die Sicherheit nicht durch falsche Konfiguration zu schwächen. Das andere Extrem ist die *vollkommene Transparenz* für den Benutzer, der gar nicht merkt, ob und welche Maßnahmen der Prozeß zur Absicherung der Kommunikationsbeziehung verwendet. Zwischen diesen beiden Polen sind viele Mischformen möglich.

Neben diesen vier Dimensionen sind bei der Untersuchung und Bewertung von Maßnahmen zur sicheren Kommunikation auch die Netzwerke, über die die Kommunikation abläuft, sowie die Dienste, die dabei verwendet werden, zu betrachten. Bei den Netztypen kann Subnetz, Intranet, Extranet, öffentliches Netz und Internet unterschieden werden. Die Auswahl der Verfahren und Maßnahmen zur Sicherung ist abhängig vom Netztyp und dem Betreiber des Netzwerkes bzw.

der entsprechenden Netzwerkeinrichtungen. Die Netztypen sind aber nicht notwendigerweise disjunkt und werden nicht immer von genau einem Betreiber oder dem Unternehmen selbst unterhalten. So kann bspw. ein Extra- oder Intranet auch Teile eines öffentlichen Netzes oder eines Providernetzes enthalten. Es ist klar, daß auf einem „fremden“ Netz nicht alle technisch möglichen Verfahren auch eingesetzt werden dürfen oder können.

Auf Seiten der Dienste müssen grundsätzlich alle Protokolle, die sich auf TCP/IP abstützen und im Unternehmensnetz zum Einsatz kommen, betrachtet werden. Um den Rahmen dieser Arbeit nicht zu sprengen, werden nur repräsentative bzw. vielgenutzte Dienste, wie Mail, WWW, entfernte Sitzungen und Einwahl mit ihren Protokollen betrachtet.

Kapitel 3

Grundlegende Konzepte zur Durchsetzung von Sicherheitsanforderungen

In diesem Abschnitt werden einige grundlegende Konzepte zur Durchsetzung der Sicherheitsanforderungen erläutert. Die in späteren Kapiteln betrachteten Verfahren, Protokolle und Produkte verwenden einen Teil dieser Konzepte, um bestimmte Sicherheitsanforderungen durchzusetzen. Es werden auch potentielle Angriffsmethoden oder Schwächen angegeben, um eine Bewertung und Einordnung der Verfahren, die diese Konzepte verwenden, zu erleichtern.

3.1 Authentisierung

Mit Hilfe der Authentisierung weist sich ein Objekt gegenüber seinem Kommunikationspartner aus; es belegt seine Identität. Im folgenden Abschnitt wird derjenige, der einen Dienst zur Verfügung stellt, nachdem sich der Partner authentisiert hat, als Server, derjenige der sich authentisiert oder den Authentisierungsvorgang initiiert, als Client bezeichnet.

Verfahren, die zur Authentisierung verwendet werden, sind Paßwörter, Challenge–Response Verfahren, Authentisierungsprotokolle und kryptologische Verfahren.

- **Paßwörter**

  Paßwörter oder *PIN (Personal Identification Number)* sind das wohl am weitesten verbreitete Authentisierungsverfahren. Es beruht auf einem gemeinsamen Geheimnis zwischen den beiden Kommunikationspartnern. In der Regel wird ein *Two–Way–Handshaking* Protokoll verwendet. D.h. der Client sendet die Authentisierungsanforderung mit seinem

Namen und seinem Paßwort an den Server, der das gemeinsame Geheimnis validiert und daraufhin die Authentisierung akzeptiert oder ablehnt. Es handelt sich hierbei um eine einseitige Authentisierung, da der Server sich gegenüber dem Client nicht authentisiert. Um eine zweiseitige Authentisierung zu erreichen, müßten Paßwörter in beide Richtungen ausgetauscht werden.

Diese einfachen Paßwortmechanismen können durch Protokollanalyse und Replay Attack angegriffen werden. Da Paßwörter in TCP/IP–Netzen oftmals im Klartext übertragen werden, genügt es einem Angreifer, den Netzverkehr abzuhören, um in den Besitz von Paßwörtern zu gelangen.

- **One–Time Password**

  Bei der Authentisierung mit *One–Time Paßwörtern* wird jedes Paßwort nur einmal verwendet. Hierzu wird in der Regel auf Seiten des Client spezielle Hardware in Form von SmartCards verwendet, die nach Eingabe einer PIN und unter Verwendung der Systemzeit das Paßwort berechnen und anzeigen. Der Server kann mit Hilfe der Nutzer–ID und der Sendezeit dieses Paßwort ebenfalls berechnen und validieren (vgl. z.B. S/KEY [Hal95]). Dadurch können Replay Attack und das Ausspähen von Paßwörtern verhindert werden. Bei One–Time Paßwörtern handelt es sich um einseitige Verfahren, da sich nur der Client authentisiert.

- **Challenge–Response Verfahren**

  Eine Möglichkeit der zweiseitigen Authentisierung bieten Challenge–Response Verfahren. Dabei sendet der Server, nach Aufforderung, eine Zufallszahl (*Seed*) an den Client, der die Antwort aus einem gemeinsamen Geheimnis und dieser Zufallszahl berechnet. Nur der Server kann ebenfalls diesen Wert berechnen, da nur er das Geheimnis kennt. Dieses Verfahren verhindert das Abhören von Paßwörtern sowie Replay Angriffe.

- **Authentisierungsprotokolle**

  Bei Authentisierungsprotokollen gibt es im Kommunikationssystem typischerweise einen vertrauenswürdigen und sicheren Server S, der mit allen Clients ein gemeinsames Geheimnis teilt. Wollen Alice und Bob kommunizieren und sich gegenseitig authentisieren, so dient S als Vermittler, der einen Sitzungsschlüssel für Alice und Bob berechnet, diesen Schlüssel sicher an beide übermittelt und damit eine sichere Verbindung zwischen beiden ermöglicht. Viele dieser Protokolle zur Authentisierung basieren auf kryptographischen Protokollen, z.B. auf dem Protokoll von *Needham und Schroeder* (vgl. [Sch94] S.52). Das bekannteste Authentisierungsprotokoll ist das am MIT entwickelte *Kerberos* (vgl. [KN93]).

- **Kryptographische Verfahren**

  Mit Hilfe von asymmetrischen oder hybriden Verschlüsselungsverfahren (vgl. Abschn. 3.3 und Anhang) läßt sich auch Authentisierung durchsetzen. Da nur der Server seinen geheimen Schlüssel kennt, ist nur er in der Lage die verschlüsselte Nachricht des Client bzw. den entsprechenden Kommunikationsschlüssel zu dekodieren. Der Client kann sich sicher sein, daß nur der gewünschte Empfänger die Nachricht auch entschlüsseln kann, d.h. der Server authentisiert sich durch den Besitz seines privaten Schlüssels und der damit verbundenen Möglichkeit, die Informationen in der Nachricht zu verwerten, beim Client. Bei diesen Verfahren ist es jedoch möglich, daß der Client seine Identität verschleiert, d.h. die Authentisierung des Client ist nicht gewährleistet.

Um auch die Authentisierung des Client durchzusetzen, wird das Konzept der digitalen Signatur („digitale Unterschrift") eingesetzt (vgl. Abschn. 3.5). Dabei unterschreibt der Client die Nachricht, indem er sie mit seinem privaten Schlüssel verschlüsselt. Diesem Geheimtext stellt er Informationen über seine Identität (im Klartext) voran und verschlüsselt die so erhaltene Nachricht mit einem hybriden Verfahren. Der Server entschlüsselt den Sitzungsschlüssel und die Nachricht und authentisiert sich durch den Besitz seines geheimen Schlüssels. Der Server kann mit Hilfe der Identitätsinformation den öffentlichen Schlüssel des Client bestimmen und damit den inneren Geheimtext entschlüsseln. Er verifiziert dadurch die Unterschrift und authentisiert den Sender, da nur der Sender den geheimen Schlüssel kennt und somit nur er die Nachricht unterschreiben konnte.

3.2 Zugriffskontrolle

Die Zugriffskontrolle sichert Ressourcen vor unberechtigtem Zugriff. Gewöhnlich wird hierbei der Zugriff auf Rechen– und Kommunikationssysteme verstanden. Da die Endsystemsicherheit nicht Thema dieser Arbeit ist, wird hier unter Zugriffskontrolle der Schutz vor unberechtigtem Zugriff auf Daten (z.B. Schlüssel) verstanden. Als Maßnahmen kommen Access Control Lists, Capabilities, Authorization Server oder Filter in Frage.

- **Access Control Lists (Zugriffskontrollisten)**:

  Access Control Lists legen für jedes Objekt des Systems (z.B. Dateien, Ressourcen) eine Liste von Zugriffsrechten der verschiedenen Benutzergruppen fest.
  Damit kann einfach festgestellt werden, welche Rechte an einem bestimmten Objekt bestehen. Für ein bestimmtes Subjekt des Systems (z.B. Benutzer, Prozesse) ist es aber schwierig festzustellen, welche Rechte dieses Subjekt besitzt.

- **Capabilities:**

  Capabilities binden, im Gegensatz zu Access Control Lists, die Rechte an das Subjekt im System. Sie wirken wie ein Ticket bzw. eine Eintrittskarte. Ist das Subjekt im Besitz eines gültigen Tickets für ein bestimmtes Objekt, so darf es das Objekt in entsprechender Weise nutzen.
  Bei Capabilities sind die Rechte eines Subjekts leicht zu ermitteln. Es sind dies die Summe seiner Capabilities. Dagegen ist es für ein Objekt schwierig, die daran bestehenden Rechte zu bestimmen.

- **Authorization Server (Autorisierungsserver):**

  Der *Autorisierungsserver* nutzt eine Datenbank, die die Zugriffsrechte der einzelnen Benutzer enthält. Diese Rechte werden vom Server durchgesetzt. Zentrale Authentisierungsserver können neben der Authentisierung auch zur Durchsetzung der Zugriffskontrolle und zum Auditing verwendet werden.

- **Filter:**

  Mit Hilfe von Filtermechanismen auf den verschiedenen Schichten können bestimmte Datenströme „geblockt“ werden.

- **Verschlüsselung:**

  Sollen Daten vor dem Zugriff durch Dritte geschützt werden, so kann der Eigentümer die Daten verschlüsselt speichern. Damit haben nur diejenigen Personen Zugriff, die den Schlüssel kennen. Mit Hilfe eines hybriden Verfahrens kann sogar erreicht werden, daß jeder Berechtigte seinen eigenen Schlüssel besitzt und nicht alle Beteiligten denselben Schlüssel kennen und benutzen müssen (vgl. Abschn. 3.3 und Anhang).

3.3 Vertraulichkeit

Die Vertraulichkeit von Daten ist gewährleistet, wenn ein unberechtigter Dritter sie nicht ausspähen, abhören oder verwerten kann. Als Maßnahmen zur Durchsetzung von Vertraulichkeit lassen sich Filter, Firewalls und Routing Mechanismen sowie Verschlüsselungstechniken einsetzen.

- **Filter, Firewall, Routing Mechanismen**

  Filter, Firewalls und Routing Mechanismen können dazu verwendet werden, Nachrichten nur über „sichere“ Netze zu übertragen bzw. nur Nachrichten, die aus sicheren Netzen kommen, zu akzeptieren. Dazu müssen die Subnetze bzw. LAN, die als sicher gelten sollen, vollkommen im Einflußbereich des Unternehmens liegen und die eingesetzte Hardware muß physisch geschützt werden.

  Der Begriff des „sicheren“ Netzes ist jedoch relativ, da nie ausgeschlossen werden kann, daß ein Angreifer dennoch physikalischen Zugriff auf die Leitung erlangt und den Verkehr abhört. Mit Hilfe dieser Verfahren läßt sich lediglich schwache Vertraulichkeit durchsetzen, da alle Rechensysteme, die an das sichere Netz angeschlossen sind, die übertragenen Nachrichten potentiell mitlesen können. Diese Verfahren bieten also keinen Schutz gegen Angreifer von innen. Berechtigte Benutzer die ihre Zugriffsrechte auf die Systeme böswillig erweitern, können relativ einfach den gesamten Verkehr im entsprechenden lokalen Netz abhören. Starke Vertraulichkeit läßt sich nur durch Verschlüsselungsverfahren erreichen.

- **Verschlüsselung (Chiffrierung)**

  Verschlüsselungsverfahren dienen dazu, einen sicheren Kanal zwischen zwei (oder mehreren) Kommunikationspartnern, über eine unsichere Verbindung, aufzubauen. Dazu werden die zu sendenden Daten mit Hilfe eines Schlüssels kodiert und verschlüsselt übertragen. Ein Angreifer, der die Daten abhört, aber nicht im Besitz des Schlüssels ist, kann die Daten nicht verwerten. Man unterscheidet symmetrische, asymmetrische und hybride Verfahren (vgl. Anhang).

  - **Symmetrische Verfahren:** Diese Klasse der Verschlüsselungsverfahren beruht auf bilateral geheimen Schlüsseln, die Sender und Empfänger über einen sicheren Kanal austauschen müssen. Diese Verfahren sind in der Regel sehr performant, haben aber den Nachteil, daß der symmetrische Schlüssel geheimgehalten werden muß und ein sicherer Kanal zum Austausch des Schlüssels zwischen den beiden Partnern erforderlich ist.
  - **Asymmetrische Verfahren (Public Key Verfahren):** Bei *Public Key Verfahren* besitzt jeder Kommunikationspartner zwei Schlüssel, einen geheim zu haltenden, privaten und einen öffentlichen Schlüssel. Der öffentliche Schlüssel kann allgemein bekanntgegeben werden, der geheime Schlüssel darf nur dem Besitzer bekannt sein. Diese Verfahren werden deshalb auch als Public Key Verfahren bezeichnet.
    Bei dieser Klasse von Techniken ist also kein sicherer Kanal für den Schlüsselaustausch nötig. Allerdings muß der öffentliche Schlüssel sicher an die Identität des Besitzers gebunden sein (z.B. durch Zertifizierung), da das System sonst durch eine „Man-in-the-Middle Attack“ angegriffen werden kann (vgl. S. 36). Diese Verfahrensklasse zeichnet sich dadurch aus, daß die Verschlüsselung im Vergleich zu symmetrischen Verfahren sehr lange dauert und damit zu ineffizient für die Verschlüsselung großer Datenmengen ist.
  - **Hybride Verfahren:** *Hybride Verschlüsselungsverfahren* versuchen, die Vorteile von symmetrischen und asymmetrischen Verfahren zu kombinieren. Dabei wird für die Verschlüsselung der Nachricht ein schnelles symmetrisches Verfahren verwendet. Der gemeinsame Schlüssel (*Kommunikationsschlüssel*) zur Verschlüsselung der Nachricht wird vom Sender zufällig gewählt, mit Hilfe eines asymmetrischen Verfahrens (und dem öffentlichen Schlüssel des Empfängers) verschlüsselt und mit dem Geheimtext an den Empfänger gesendet. Dieser kann mit seinem privaten Schlüssel den Kommunikationsschlüssel und damit auch die eigentliche Nachricht entschlüsseln.

3.4 Integrität

Ist die Integrität einer Nachricht gewährleistet, so ist die Nachricht beim Empfänger unverändert und ohne Verzögerung (außer der normalen Übertragungszeit) angekommen. Andernfalls kann der Empfänger Veränderungen oder Wiedereinspielungen erkennen. Mögliche Maßnahmen sind Sequenznummern und die Berechnung und Übertragung eines Message Authentication Codes.

- **Sequenznummern**

  Sequenznummern, die beginnend bei einer zufällig gewählten Zahl mit jedem gesendeten Datenpaket (verschlüsselt) mitübertragen und inkrementiert werden, können bei der Erkennung von Replay Angriffen Verwendung finden. Die erste Sequenznummer muß zufällig gewählt werden, da sonst die Gefahr einer *Sequence–Number Attack* besteht (vgl. [Bel89]). Dabei kann ein Angreifer, der den Netzverkehr mithören kann, die Sequenznummern mitzählen oder „erraten“, falls er das Verfahren kennt, mit dem die Sequenznummern inkrementiert werden.

  Sequenznummern sind als alleiniges Mittel nicht geeignet, Veränderungen der Nachricht zu erkennen.

- **Message Authentication Code (MAC)**

  Ein *Message Authentication Code (MAC)*, auch *Message Digest (MD)* genannt, ist ein Hash–Wert, der mittels einer Hash–Funktion über ein Datenpaket beliebiger Länge berechnet wird. Er hat eine feste, i.d.R. sehr viel kürzere Länge (z.B. 16 Byte bei MD5) als die eigentliche Nachricht. Die verwendete Funktion zeichnet sich dadurch aus, daß sie kryptologisch sicher ist. Das bedeutet, daß es zu schwierig ist, eine Kollision, d.h. zwei Nachrichten M und M' zu finden, die denselben Hash–Wert ($MAC(M) = MAC(M')$) besitzen. Außerdem führt die Komplementierung nur eines Bits der Nachricht nicht nur zu einer lokalen Änderung im MAC, sondern zu erheblichen Veränderungen, die sich über den gesamten MAC erstrecken können.

  Der MAC wird mit der Nachricht oder getrennt von dieser an den Empfänger übermittelt, der den Hash–Wert über die empfangene Nachricht berechnet, mit dem gesendeten MAC vergleicht und damit Veränderungen an den Daten erkennen kann. Der MAC, der mitübertragen wird, ist natürlich zu sichern, da sonst ein Angreifer, der den Hash–Algorithmus kennt, neben der Nachricht auch den MAC verändern könnte. Dagegen kann der Sender sich auf zwei Arten schützen. Entweder wird der MAC vor der Übertragung verschlüsselt oder die Daten werden vor der Berechnung des Hash–Wertes mit einem Schlüssel (gemeinsames Geheimnis) konkateniert, um dann den MAC über die so erhaltene Zeichenkette zu berechnen. Es ist klar, daß ein Angreifer ohne Kenntnis des Geheimnisses den MAC nicht berechnen kann (außer durch eine Brute–Force Attack).

3.5 Verbindlichkeit

Die Verbindlichkeit einer Nachricht kann nur mittels einer *digitalen Unterschrift* (Digital Signature) gewährleistet werden. Zur Realisierung einer digitalen Unterschrift können Public Key Verschlüsselungsverfahren verwendet werden. Dazu „unterschreibt“ der Absender seine Nachricht mit seinem geheimen Schlüssel, d.h. er verschlüsselt die Nachricht mit seinem geheimen Schlüssel. Der Empfänger kann dann den öffentlichen Schlüssel des Senders zur Validierung der Unterschrift verwenden. Für einen Dritten ist es, ohne Kenntnis des privaten Schlüssels des Senders, nicht möglich, eine Unterschrift zu fälschen.

Wird die Vertraulichkeit einer Kommunikation beispielsweise durch ein hybrides oder symmetrisches Verfahren geschützt und unterschreibt der Sender die gesamte Nachricht, so verliert er die Vorteile des hybriden Verfahrens, da die Erzeugung der Unterschrift i.d.R. sehr viel mehr Zeit benötigt, als das gesamte hybride oder symmetrische Verfahren. In diesem Fall könnte auch ein vollständig asymmetrisches Verfahren zur Sicherung der Vertraulichkeit Verwendung finden, mit dem Nachteil, daß die Performance einer solchen Lösung sehr schlecht wäre. Deshalb unterschreibt der Sender in der Praxis nicht die Nachricht selbst, sondern den MAC der Nachricht. Ein Empfänger kann die Signatur auch nicht für eine andere Nachricht wiederverwenden, da er dann eine Kollision in der Hash–Funktion finden müßte. Aus demselben Grund kann die unterzeichnete Nachricht später auch nicht mehr verändert werden.

Ebenso wie die Sicherheit der Unterschrift auf Papier von Rahmenbedingungen abhängt (z.B. Strafe für Urkundenfälschung) ist die Sicherheit der digitalen Signatur an bestimmte Prämissen gebunden. Der Sender darf bspw. seinen privaten Schlüssel nicht bekanntgeben. Dies wäre damit vergleichbar, daß ein Unterzeichner freiwillig einen Blankoscheck unterschreibt.

Die digitale Unterschrift einer Nachricht, die mit einem hybriden Verfahren verschlüsselt wurde, erfüllt neben der Verbindlichkeit auch die Forderung nach Authentisierung der Kommunikationspartner. Ist der öffentliche Schlüssel des Empfängers sicher mit seiner Identität verknüpft (z.B. durch ein Zertifikat), so kann nur er den Kommunikationsschlüssel und damit die Nachricht entschlüsseln, da nur er im Besitz des entsprechenden geheimen Schlüssels ist. Der Empfänger verifiziert die Unterschrift des Senders mit Hilfe des öffentlichen Schlüssels des Senders. Er kann die Unterschrift auch gegenüber Dritten ohne Mithilfe des Unterzeichners belegen.

3.6 Auditing und Logging

Auditing und Logging dient im allgemeinen dazu, Verbindungsdaten, Daten über Dienstnutzung, Ressourcenverbrauch u.ä. zu erhalten. Die Daten werden in Dateien oder in einer Datenbank für die spätere Auswertung gespeichert. Sie werden für die Durchsetzung verschiedenster Ziele verwendet. Die Daten können für Abrechnungszwecke, Lastuntersuchungen oder sonstige statistische Zwecke verwendet werden. Aus sicherheitspolitischen Gründen werden die Daten erfaßt, um Einbrüche oder unberechtigte Ressourcenbenutzung erkennen zu können.

- **Protokollierungsdienste des Betriebssystems**

  Bei Loggingdiensten auf Betriebssystemebene können i.d.R. über Konfigurationsdateien bestimmte Dienstaufrufe, Netzverbindungen o.ä. protokolliert werden. In UNIX können mit dem Syslog–Dämon Meldungen von anderen Dämonprozessen aufgezeichnet werden. Damit lassen sich bspw. Meldungen des Betriebssystemkerns von Authentisierungsdiensten wie z.B. `login`, des PPP– oder anderer Dämonen zur Abwicklung von Kommunikationsprotokollen u.a. speichern.

- **Sniffer, LAN–Monitore**

  Mithilfe von *Sniffern* bzw. *LAN–Monitoren*, kann der gesamte Verkehr innerhalb eines LAN–Segmentes mitgelesen werden, um den Verdacht auf einen Angriff oder Einbruch bestätigen und protokollieren zu können. Diese Programme versetzen die Netzwerkkarte in den sog. „*Promiscuous Mode*“, in dem alle Pakete, nicht nur die an den entsprechenden Rechner adressierten, entgegengenommen werden. Damit die Datenmenge nicht zu groß wird, kann über entsprechende Filter ein Teil des Verkehrs ausgewählt werden. Trotzdem kann die gespeicherte Datenmenge sehr schnell extrem groß und die Last an dem Rechner ziemlich hoch werden. Daher werden diese Programme in der Regel nur über kürzere Zeiträume auf einem eigens dafür bestimmten Rechner gestartet. Als Mittel, um im normalen Betrieb Auditingdaten zu erhalten, sind Sniffer nur bedingt geeignet. Außerdem stellen Maschinen, die derartige Programme ausführen, lohnende Ziele für Angreifer dar.

- **TCP–Wrapper**

  Der *TCP–Wrapper* [Ven92] ist ein Programm, um Verbindungsanfragen für TCP/IP–Dienste zu protokollieren. Er wird über `inetd.conf` dem eigentlichen Dienst vorgeschaltet und ruft nach der Protokollierung der entsprechenden Daten den angeforderten Dienst auf.

- **Dienstspezifisches Auditing**

  Bei vielen Diensten und Programmen besteht die Möglichkeit, das Programm so zu konfigurieren, daß die gewünschten Daten vom jeweiligen Dienst protokolliert werden. Die Implementierung, die Auswahl und der Umfang der zu protokollierenden Daten ist sehr unterschiedlich realisiert und muß daher beim entsprechenden Programm betrachtet werden.

Kapitel 4

Netzstruktur der BMW AG

Die BMW AG betreibt ein unternehmens– und weltweites Kommunikationsnetz. Das *Corporate Network*, auf TCP/IP–Basis, setzt sich aus einem LAN/WAN–Verbundnetz zusammen. Zukünftig soll es durch ein *Extranet*, das aus mehreren *ISP* (Internet Service Provider) Netzen bestehen wird, erweitert werden. In einer ersten Ausbaustufe sollen damit die deutschen Händler mit dem Corporate Network und untereinander verbunden werden.

Das *LAN–Verbundnetz* des Kern– bzw. Zentralbereiches München baut auf einem 100 Mbit/s FDDI Backbone auf, der die Münchner Standorte verbindet. Der FDDI Ring setzt sich aus BMW–eigenen Netzteilen, Mietleitungen und Ortsvermittlungsstellen der Telekom zusammen. An das Backbone werden über 15 (CISCO–) Brouter die lokalen Netze angeschlossen. Der Anschluß der LAN erfolgt parallel oder sequentiell. Außerdem gibt es einige LAN, die zusätzlich zur Verbindung über FDDI auch direkt untereinander verbunden sind (z.B. aus Backup–Gründen). Die Struktur ist exemplarisch in Abbildung 4.1 dargestellt.

Gegenwärtig sind rund 150 Ethernetsegmente (mit steigender Tendenz), mehr als 40 Token Ring Segmente (mit abnehmender Tendenz) sowie einige DEC– und Apple Talk Netze an das Backbone angeschlossen. Der LAN–Verbund repräsentiert zur Zeit rund 20.000 Endsysteme vom Mainframe unter MVS über Workstations mit zahlreichen UNIX–Varianten, Personal Computer unter verschiedenen Windows–Versionen bis hin zu unterschiedlichsten „Exoten“.

Mit Hilfe des *WAN–Verbundnetzes* werden die verschiedenen nationalen und internationalen Produktionsstätten, Vertriebszentren, Tochter– und Beteiligungsgesellschaften an das Corporate Network angeschlossen. Dazu existiert ein weiterer FDDI Ring, der mit dem Backbone des Zentralbereiches über einen Brouter verbunden ist. An diesem Ring sind rund 10 weitere Brouter angeschlossen, um den WAN Verkehr abzuwickeln. Die Verbindungen zu den nationalen und internationalen Standorten erfolgt über Mietleitungen, öffentliche Netze und private Carrier.

An den entfernten Standorten werden die LAN ebenfalls über Brouter bzw. Router mit dem Corporate Network verbunden. Weltweit sind im Moment rund 100 Router bzw. Brouter im Einsatz. In näherer Zukunft dürfte sich diese Zahl auf 150 erhöhen.

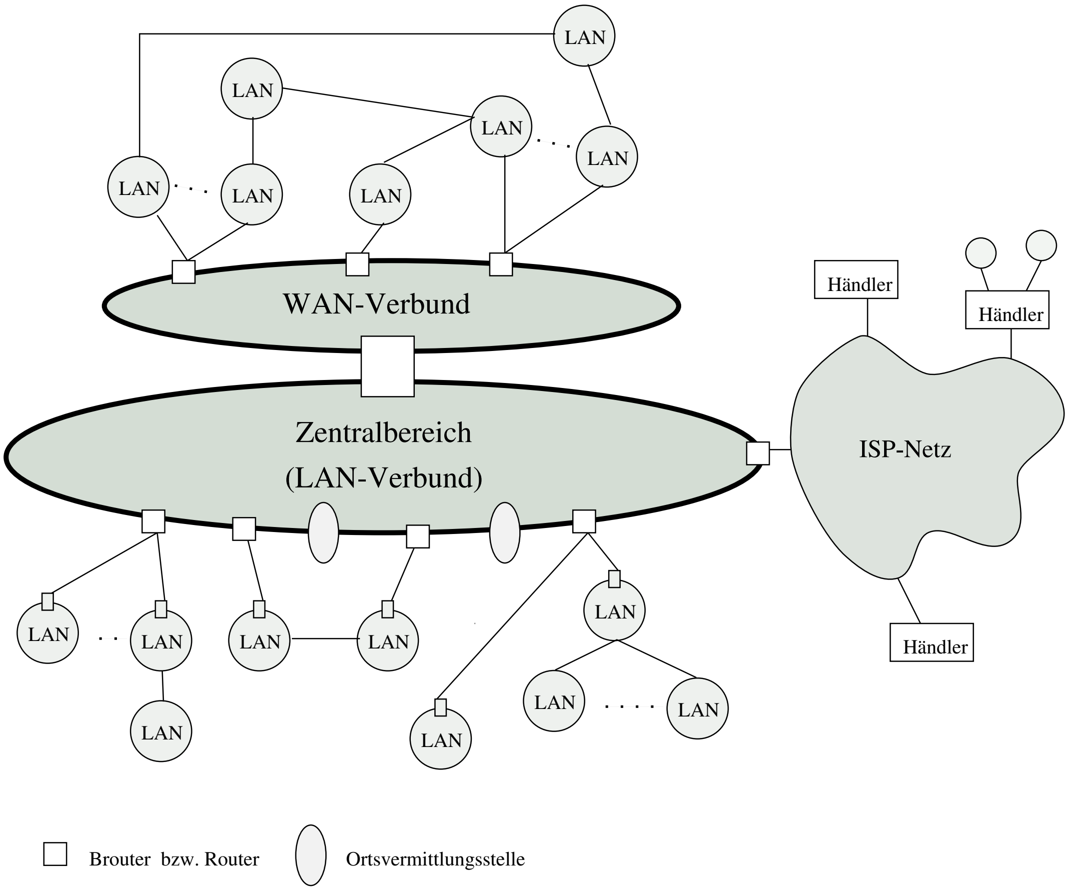

Abbildung 4.1: Netzstruktur der BMW AG (exemplarisch)

Da die entfernten Standorte teilweise auch untereinander auf nationaler bzw. internationaler Ebene vernetzt sind, entsteht eine stark vermaschte Topologie im WAN–Verbundnetz.

Auf Seite der Netztechnologien und Architekturen wird überwiegend TCP/IP eingesetzt. Daneben existieren aber auch SNA (Systems Network Architecture), OSI mit entsprechenden Diensten und reine PC–Netze unter SPX/IPX oder LANManager. Das Management für die FDDI–Ringe und den LAN–Verbund innerhalb Münchens erfolgt zentral. Im WAN–Verbund wird das Management dezentral organisiert.

Das Corporate Network der BMW AG ist ein teilweise stark vermaschtes und sehr heterogenes Netz. Die geographische Verteilung der Standorte sowie die Größe des Netzes sind erheblich. Diese Eigenschaften und die verteilte Verantwortlichkeit sowie das verteilte Management erschweren eine einheitliche Sicherheitspolitik.

Im Rahmen dieser Arbeit wird primär nicht das weltweite Netz, sondern der LAN–Verbund des Zentralbereiches betrachtet. Trotzdem sind natürlich bestimmte Konzepte, Protokolle und

Produkte, die untersucht werden, auch im WAN–Verbund einsetzbar und auf das weltweite Corporate Network übertragbar. Da TCP/IP das strategische Übertragungsprotokoll bei der BMW AG ist, und andere Protokolle immer weniger Verwendung finden, wird nur die TCP/IP–basierte Kommunikation betrachtet.

Kapitel 5

Sicherheitspolitik

Grundlage für alle Maßnahmen, die die Sicherheit eines Unternehmens oder einer Organisation betreffen, ist eine unternehmensweite „*Security Policy*“ *(Sicherheitspolitik)*. Dabei werden zwei Klassen von Politiken unterschieden. Die Klasse der *benutzerbestimmbaren Politiken (Discretionary Policies)* erlaubt den Benutzern des Systems die Zugriffsrechte an den Objekten, die sie besitzen (Owner Recht), d.h. die sie erzeugt haben, individuell festzulegen und zu vergeben. Die zweite Klasse von Politiken ist die der *systemglobalen Politiken (Mandatory Policies)*, in der die Vergabe von Zugriffsrechten global auf der Basis von a priori Festlegungen kontrolliert wird.

Die Festlegung bzw. Einführung einer Sicherheitspolitik bedeutet, einen Plan zu entwickeln und formal zu spezifizieren, der festlegt, wie zukünftig mit der Sicherheit von Computern und Netzsystemen umgegangen werden soll. Die Sicherheitspolitik dient als Grundlage, um Abläufe, Methoden und Werkzeuge zu ermitteln, mit denen die spezifizierten Sicherheitsanforderungen durchgesetzt werden können. Für den Fall der Gefährdung der Sicherheit, z.B. bei einem Angriff oder Einbruch in ein System, soll die Policy auch Hinweise geben, um auf die entsprechende Situation reagieren zu können.

Im folgenden wird nur auf die systemglobale Policy eingegangen, die allerdings auch Richtlinien für die Festlegung bzw. Beschränkung der benutzerbestimmbaren Politiken enthalten kann.

Eine Policy muß mindestens folgende Fragestellungen beantworten können:

- Was bzw. wer soll geschützt werden? (Dimension der Kommunikationspartner)
- Vor wem soll es geschützt werden? (Dimension der Bedrohungen)
- Welche Sicherheitsanforderungen müssen durchgesetzt werden? (Dimension der Sicherheitsanforderungen)
- Wo sind geeignete Maßnahmen zu implementieren? (Architekturelle Dimension)
- Wie kann der Schutz durchgesetzt werden? (Realisierung)

5.1 Organisatorische Gesichtspunkte

Eine Sicherheitspolitik muß die strategischen und operativen Unternehmensziele beachten. Sie soll so entwickelt werden, daß bestehende Politiken, Regeln und Vorschriften, die im Unternehmen eingehalten werden müssen, beachtet werden.

Es müssen Verantwortungsbereiche festgelegt und klar verteilt werden, damit sichergestellt ist, daß es für jede Art von eventuell auftretenden Problemen einen Ansprechpartner gibt, der diese lösen kann bzw. für die Behebung der Probleme verantwortlich ist. Die Mitarbeiter sind über die Sicherheitspolitik zu informieren und langfristig soll eine „Sicherheitskultur“ im Unternehmen aufgebaut werden.

Bei der Aufstellung einer Sicherheitspolitik müssen Fachpromoter, die über das technische Know How verfügen und Machtpromoter, die die Politik durchsetzen können, zusammenarbeiten, denn eine Sicherheitspolitik, die weder implementiert noch durchgesetzt werden kann, ist nutzlos.

5.2 Security–Engineering

Um ein sicheres System zu konstruieren, sind analog zum Prozeß des Software–Engineerings verschiedene Phasen, u.U. iterierend, zu durchlaufen. Diese Phasen werden unter dem Begriff *Security–Engineering* zusammengefaßt.

Ausgehend von einer Bedrohungs- und Risikoanalyse und einer Human Resource Analysis werden Sicherheitsanforderungen und Sicherheitsklassen spezifiziert und daraus die Security Policy abgeleitet, die wiederum Grundlage für die Implementierung von konkreten Methoden ist.

Um den Erfolg und die Performance dieser Methoden bestimmen zu können, müssen quantifizierbare Ziele und adäquate Meßverfahren festgelegt werden. Die Einhaltung der Sicherheitsanforderungen, der Erfolg der Methoden sowie die Performance sind kontinuierlich zu überwachen und die Policy ggf. zu verbessern.

5.2.1 Risikoanalyse

Ein Rechensystem ist vielfältigen *Bedrohungen (Threats)*, die den Verlust der Datenintegrität, der Vertraulichkeit oder der Verfügbarkeit betreffen, ausgesetzt.

Unter der Risikoanalyse versteht man einen Prozeß mit dem versucht wird, alle potentiellen Bedrohungen zu ermitteln. Dazu müssen die internen und externen Bedrohungen und mögliche Schwachpunkte der Systeme sowie die zu schützenden Ressourcen ermittelt werden.

Für die Sicherheitspolitik sind die potentiellen Sicherheitsrisiken der Systeme mit den Kosten abzuwägen, die durch die Einführung von Maßnahmen entstehen, die diese Risiken beseitigen oder abschwächen (*Risikobewertung*). D.h. es ist eine Kosten/Nutzen–Abwägung durchzuführen, da für den Schutz einer Ressource nicht mehr ausgegeben werden soll, als deren tatsächlicher Wert ist.

5.2.2 Human Resource Analysis

Neben der eigentlichen Risikoanalyse sind in einer *Human Resource Analysis* auch die Benutzer und deren Rechte zu untersuchen. Die Nutzer der Rechen– und Kommunikationssysteme werden dabei in Gruppen eingeteilt, die die Zugriffsrechte und -beschränkungen des einzelnen Nutzers bestimmen.

Dabei ist auch zu klären, wie diese Zugriffsrechte durchgesetzt bzw. geschützt werden können (z.B. in Form von Paßwörtern oder Schlüsseln) und wie diese sicher und mit möglichst wenig Aufwand an die Mitarbeiter verteilt werden können.

Außer den Rechten, die die Benutzer erhalten, müssen auch deren spezielle Sicherheitsanforderungen, die sie explizit oder implizit stellen können, berücksichtigt werden. Die Akzeptanz eines Systems oder einer Maßnahme nimmt auch dadurch zu, daß der Benutzer seine Kommunikation entsprechend seiner Anforderungen geschützt weiß.

5.2.3 Sicherheitsanforderungen und Sicherheitsklassen

Aus der Risiko- und der Human Resource Analysis und hierbei insbesondere auf Grundlage der potentiellen Bedrohungen und der Sicherheitsanforderungen wird eine Einteilung der Benutzer in Sicherheitsklassen abgeleitet. Die Sicherheitsklasse, der ein Nutzer zugeteilt wird, bestimmt dessen Zugriffsrechte.

Nachdem die Sicherheitsanforderungen bzw. die Zugriffsbeschränkungen für jede Sicherheitsklasse feststehen, muß festgelegt werden, wo Maßnahmen zur Durchsetzung dieser Anforderungen bzw. zur Beschränkung des Zugriffs eingesetzt werden können. Hierzu ist das technische und organisatorische Umfeld einer jeden Sicherheitsklasse auf mögliche Zugangs-, Kontroll- und Überwachungspunkte hin zu untersuchen.

5.3 Beispiel einer Sicherheitspolitik

Zur Darstellung der Konzepte und Methoden wird im folgenden ein stark vereinfachtes Modell einer Sicherheitspolitik, die nur auszugs– und andeutungsweise dargestellt wird[1], verwendet. Eine vereinfachte und damit „schlanke“ Policy ist besser zu überblicken. Außerdem ist die Sicherheitspolitik eines Unternehmens aus naheliegenden Gründen nicht zur Veröffentlichung

[1] Die folgenden Ausführungen beruhen auf [Jun95, Jun96], die dem Autor nur in Auszügen vorlagen.

freigegeben. Es wird daher lediglich eine Einteilung in grobgranulare Sicherheitsklassen und Kontrollpunkte vorgestellt.

In der beispielhaften Sicherheitspolitik werden alle Kommunikationsteilnehmer zuerst in zwei große Sicherheitszonen aufgeteilt. In der Sicherheitszone A gilt der Grundsatz des expliziten Verbietens, d.h. der Nutzer aus Zone A besitzt grundsätzlich freien Zugang zu den Ressourcen, es sei denn, der Zugriff wird explizit untersagt. Im Gegensatz dazu gilt in Sicherheitszone B der Grundsatz des expliziten Erlaubens, d.h. der Nutzer hat hier nur Zugriff auf Ressourcen, falls ihm dieser explizit gestattet wird.

Innerhalb der beiden Zonen werden die Nutzer in Klassen und Subklassen unterteilt. Das Entscheidungskriterium für die Klasseneinteilung ist der Standort des Endsystems bzw. die Art des Zugriffs auf das Unternehmensnetz. Hier wird die Klasse I, der Internen von denen der Externen (Klasse E) unterschieden. Die Klasse I greift „von innen“ auf das Unternehmensnetz zu; die Klasse E entsprechend von „außen“.

Die Einteilung in Subklassen erfolgt nach den Beschränkungen, denen der Nutzer unterliegt. Der Subklasse F werden die Nutzer zugeordnet, die eher „freizügig“ zu behandeln sind, in Subklasse R diejenigen, deren Beschränkungen mehr „restriktiv“ sind.

Aus diesen Unterteilungen lassen sich nun Benutzergruppen bzw. Sicherheitsklassen bilden, die von der Klasse AIF mit den größten Freiheiten bis zur Klasse BER mit den größten Einschränkungen reicht.

Als Überwachungs- bzw. Kontrollpunkte werden der Zugang zum Endsystem bzw. zu einer Anwendung und der Zugang zum Unternehmensnetz betrachtet. Daneben ist auch die Übertragung von Daten im Unternehmensnetz („Bewegung im Netz“) zu überwachen. Dabei ist bspw. zu prüfen, ob ein Nutzer mit Hilfe des Netzes eine Verbindung zu einem anderen Endsystem oder zu einem anderen (Sub–) Netzwerk aufbauen darf, aber auch, ob bei einer berechtigten Übertragung von Daten Integrität, Vertraulichkeit oder Verbindlichkeit gewährleistet werden müssen.

Die Sicherheitsklassen und die Kontrollpunkte spannen eine Matrix auf. Als Elemente dieser Matrix (Kreuzungspunkte von Sicherheitsklassen und Kontrollpunkten) ist festzulegen, ob und ggf. welche Sicherheitsanforderungen für die spezielle Klasse am jeweiligen Kontrollpunkt erforderlich sind.

Kapitel 6

Schlüssel– und Zertifikatsmanagement

Bei nahezu allen Verfahren, die zur sicheren Kommunikation eingesetzt werden, sind kryptologische Protokolle oder diverse Verschlüsselungsverfahren implementiert, die Schlüssel oder Zertifikate benötigen. Diese Schlüssel bzw. Zertifikate müssen erzeugt, sicher gespeichert, verteilt und eventuell widerrufen werden. Diese Aufgaben werden unter dem Begriff des *Schlüssel– und Zertifikatsmanagments (Key– and Certificate Management)* zusammengefaßt.

6.1 Schlüsselmanagement

6.1.1 Schlüsselgenerierung

Zur Erzeugung von Schlüsseln ist, abhängig vom Verschlüsselungsverfahren, spezielle Software nötig. Das Ziel der Schlüsselgenerierung ist die Erzeugung kryptologisch sicherer Schlüssel sowie die „Perfect Forward Security". Hierunter versteht man die unabhängige Generierung von Schlüsseln, d.h. keiner der Schlüssel darf vom vorher generierten abhängen oder sich aus einem vorhergehenden ableiten lassen.

Bei symmetrischen Verfahren wird ein Schlüssel meist zufällig gewählt. In diesem Fall ist ein starker Zufallszahlengenerator notwendig, dessen Initialisierung auch zufällig erfolgt und nicht etwa über die Systemzeit oder den Prozeß–ID. Sonst kann ein Angreifer diese Kenntnisse oder die Schwäche des Zufallszahlengenerators ausnützend, den möglichen Schlüsselraum stark einschränken und über diese kleinere Schlüsselmenge eine Brute-Force Attack durchführen. Für *DES (Data Encryption Standard)* [Nat93] werden bspw. 56 Bit lange Schlüssel verwendet. Werden diese Schlüssel durch einen Pseudozufallszahlengenerator erzeugt, der mit einer 8 Bit Zahl initialisiert wird, dann braucht der Angreifer nur 256 Möglichkeiten anstatt der 2^{56} möglichen

Schlüssel zu testen, indem er den Zufallszahlengenerator mit den 2^8 möglichen Initialisierungen startet. In dem 56 Bit Schlüssel, der auf diese Weise erzeugt wird, sind lediglich 8 Bit „echte Information“ enthalten.

Bei asymmetrischen Verfahren müssen i.d.R. sehr große Primzahlen ermittelt werden. Hierzu werden zufällig gewählte Zahlen in einem vorgegebenen Intervall auf ihre Primzahleigenschaft getestet. Da der Beweis, daß eine Zahl prim ist, äquivalent zum Problem der Faktorzerlegung und dies für derartig große Zahlen zu aufwendig ist[1], werden probabilistische Algorithmen zum Primzahltest eingesetzt (vgl. z.B. *Rabin-Miller Algorithmus* [Mil76, Rab80]). Bei diesen Algorithmen läßt sich über die Anzahl i der wiederholten Tests die Fehlerwahrscheinlichkeit beliebig klein halten (bei Rabin-Miller z.B. $P(n \text{ prim}) > 1 - \left(\frac{1}{4}\right)^i$). Aber selbst diese Algorithmen benötigen relativ viel Rechenzeit, um große Primzahlen zu bestimmen.

Die Erzeugung von Schlüsseln kann zentral oder dezentral erfolgen. Beim zentralen Ansatz erzeugt ein *Key–Server* die Schlüssel und verteilt sie an die Clients. In diesem Fall kann die Hard– und Software den entsprechenden Anforderungen angepaßt werden und die Endsysteme werden nicht mit rechenintensiven Prozessen belastet. Der zentrale Server kann sich aber auch zum „Flaschenhals“ des Systems entwickeln, insbesondere dann, wenn auch Kommunikationsschlüssel zentral erzeugt werden. Außerdem muß dieser Rechner besonders geschützt werden, da er den zentralen Angriffspunkt darstellt. Auch die sichere Verteilung der Schlüssel ist zu gewährleisten.

Beim dezentralen Ansatz werden die Schlüssel auf den jeweiligen Endsystemen erzeugt. Da der Benutzer sehr selten Schlüssel für asymmetrische Verfahren erzeugt, kann deren Generierung trotz des hohen Rechenaufwands auch auf die Endsysteme verlagert werden, soweit für die entsprechende Plattform geeignete Software zur Verfügung steht. Besteht auch die Möglichkeit einen starken Zufallszahlengenerator in den Endsystemen zu installieren, so können mit dem dezentralen Ansatz die Nachteile des zentralen Key–Servers vermieden werden. Allerdings stellt sich hier die Frage nach der Sicherung der Endsysteme und der sicheren Speicherung der Schlüssel.

6.1.2 Schlüsselspeicherung

Die Schlüssel müssen so gespeichert werden, daß sie nicht von Unbefugten verändert werden können, da ein Angreifer die Schlüssel sonst durch seine eigenen ersetzen und damit u.U. den Verkehr mitlesen oder eine Maskerade durchführen könnte.

Die sicherste aber auch aufwendigste Methode ist, die Schlüssel auf einem externen Medium zu speichern, auf das nur lesend zugegriffen werden kann. Dieses Medium soll nur solange mit dem Rechensystem verbunden bleiben, solange die Schlüssel tatsächlich benötigt werden, andernfalls muß es getrennt und sicher aufbewahrt werden. Eine weitere Möglichkeit ist die

[1] Der effizienteste bekannte Algorithmus zur Faktorisierung einer Zahl n benötigt näherungsweise $e^{\sqrt{\ln(n)\cdot\ln(\ln(n))}}$ Operationen (vgl. Fußnote S. 34). Für eine 512 Bit lange Zahl (rund 154 Dezimalstellen) würde dieser Algorithmus über $6 \cdot 10^{19}$ Operationen zur Faktorisierung benötigen.

Schlüssel im System zu speichern und durch geeignete Zugriffskontrollmechanismen zu schützen. Die Sicherheit dieser Vorgehensweise hängt aber stark vom verwendeten Betriebssystem ab. Dabei ist insbesondere zu beachten, daß auch ein Administrator oder „Superuser" keinen Zugriff auf die geheimen Schlüssel erlangen darf. Um dies in einem Betriebssystem wie UNIX, in dem der Superuser alle Dateien lesen und auch beschreiben darf, zu gewährleisten, können die Schlüssel selbst, mit einem Paßwort als Schlüssel, verschlüsselt werden.

Auch bei der Speicherung der Schlüssel kann ein dezentraler oder ein zentraler Ansatz gewählt werden. Eine dezentrale Speicherung ist dann möglich, wenn die verwendeten Systeme die oben genannten Forderungen erfüllen können. Dabei ist aber der „Schwachpunkt" Benutzer gesondert zu betrachten. Speichert jeder Benutzer seine Schlüssel selbst, kann trotz intensiver Schulung nicht gewährleistet werden, daß der Benutzer die Sicherheit nicht bewußt oder unbewußt schwächt, indem er z.B. ein Wechselmedium mit den Schlüsseln immer im System beläßt, oder seine Schlüssel nicht durch geeignete Zugriffskontrollmechanismen schützt.

Die strengen organisatorischen und technischen Forderungen lassen sich leichter auf einem zentralen System mit speziellem Betriebssystem und spezieller Hardware durchsetzen. Auch bei der Speicherung hat der zentrale Ansatz dieselben Nachteile wie die zentrale Erzeugung der Schlüssel.

6.1.3 Schlüsselverteilung

Damit zwei oder mehrere Kommunikationspartner ein kryptographisches Verfahren anwenden können, müssen vorher symmetrische oder asymmetrische Schlüssel ausgetauscht werden. Dazu ist es notwendig, daß die Partner sich bzw. ihre Schlüssel authentisieren. Andernfalls kann ein Angreifer die Schlüssel durch seine eigenen ersetzen und damit die verschlüsselten Daten mitlesen (vgl. Man–in–the-Middle Attack S. 36).

Die sicherste Methode des Schlüsselaustausches sowohl für symmetrische als auch für öffentliche Schlüssel ist der persönliche Austausch oder zumindest der Austausch „*Out-of-Band*", d.h. außerhalb des Kommunikationssystems, das zur späteren Datenübertragung verwendet wird. Dies setzt allerdings die generelle Möglichkeit einer Out-of-Band Kommunikation voraus. Außerdem müssen die Partner sich „kennen" oder sich über festgelegte Merkmale authentisieren. Für Prozesse, Hosts oder Netzwerkeinrichtungen ist diese Art des Schlüsselaustausches nur sehr eingeschränkt durchführbar. Auch für Benutzer ist ein derartiger Schlüsselaustausch sehr aufwendig, insbesondere wenn mit sehr vielen, geographisch weit entfernten Partnern, mit Hilfe verschiedenster Verschlüsselungsverfahren kommuniziert werden soll.

Eine weitere Methode ist, den Schlüssel in mehrere Teile aufzuspalten und diese Teile zu verschiedenen Zeiten mit Hilfe unterschiedlicher Übertragungsprotokolle an den Empfänger zu übermitteln und zu hoffen, daß der Angreifer nicht alle Nachrichten abhören kann. Dieses Verfahren ist jedoch sehr schwach, da ein Angreifer, auch mit Teilen des Schlüssels, den Schlüsselraum unter Umständen sehr stark einschränken kann.

Zur Verteilung von symmetrischen Schlüsseln kann ein asymmetrisches Verfahren verwendet werden. Einer der Kommunikationspartner generiert einen zufälligen Schlüssel und verschlüsselt ihn mit dem öffentlichen Schlüssel des Partners. Das Problem der sicheren Schlüsselverteilung wird hier allerdings nur auf die sichere Verteilung bzw. sichere Verknüpfung der öffentlichen Schlüssel mit der Identität des Benutzers verlagert. Die Verteilung dieser Schlüssel erfolgt i.d.R. über Zertifikate (vgl. Abschnitt 6.2).

Eine Möglichkeit, sich auf einen gemeinsamen Schlüssel zu einigen ohne vorher oder Out-of-Band Daten auszutauschen, bietet das *Diffie–Hellman Protokoll* [DH76]. Alle Daten, die hierbei benötigt werden, können über einen unsicheren Kanal übertragen werden. Das Diffie–Hellman Protokoll beruht auf Berechnungen in einer multiplikativen, endlichen, zyklischen Gruppe G mit Generator p, in der die Berechnung diskreter Logarithmen zu aufwendig ist, und stellt sich wie folgt dar.

Zuerst einigen sich Alice und Bob auf zwei große ganze Zahlen p und g mit $1 < g < p$. Diese Zahlen brauchen weder geheimgehalten noch sicher übertragen zu werden, sondern sie können über den unsicheren Kanal ausgehandelt werden. Das Protokoll ist wie folgt durchzuführen:

1. Alice wählt zufällig eine große Integerzahl x und berechnet $X = g^x \bmod p$
2. Bob wählt zufällig eine große Integerzahl y und berechnet $Y = g^y \bmod p$
3. Alice sendet X an Bob und Bob sendet Y an Alice. Alice behält x und Bob behält y geheim.
4. Alice berechnet $k = Y^x \bmod p$
5. Bob berechnet $k' = X^y \bmod p$

Es gilt dann $k = k'$ da $k = g^{xy} \bmod p = k'$. Damit kann k als Schlüssel für ein symmetrisches Verfahren verwendet werden. Kein Angreifer, der den Kanal abhört, kann k berechnen, solange er nicht das Problem des diskreten Logarithmus zur Berechnung von x und y lösen kann[2], da ihm nur g, p, X und Y bekannt sind.

Die Wahl von g und p haben entscheidenden Einfluß auf die Sicherheit. Die Zahl p sollte eine Primzahl und auch $(p-1)/2$ sollte prim sein. Für g sollte eine primitive Wurzel mod p gewählt werden. Heute gelten 512 Bit für die Länge von p nur noch als bedingt sicher und es wird empfohlen, p mit 1024 oder 2048 Bit Länge zu wählen.

6.1.4 Widerruf von Schlüsseln

Neben der Erzeugung, Verteilung und Speicherung ist der Widerruf von Schlüsseln eine wichtige Funktion des Schlüsselmanagements. Ein Nutzer muß seinen Schlüssel für ungültig erklären

[2] Der schnellste derzeitig bekannte Algorithmus für die Gruppe Z_p^* ist der *Number Field Sieve* mit einer Komplexität von $O(e^{c(\log p)^{1/3}(\log\log p)^{2/3}})$ mit $c < 2$ [LLMP90, LL93].

können, falls er den Verdacht hat, daß dieser kompromittiert wurde, d.h. daß ein unberechtigter Dritter Kenntnis des Schlüssels erlangt hat. Auch für den Fall, daß der Eigentümer selbst seinen Schlüssel „verloren“ hat oder nicht mehr auf diesen zugreifen kann, muß er die Möglichkeit haben, seinen Schlüssel zurückzunehmen, da er andernfalls die an ihn gerichteten Nachrichten nicht mehr entschlüsseln kann.

Bei symmetrischen Schlüsseln, die zwischen zwei Partnern ausgetauscht wurden, genügt es, daß die Partner sich über den Widerruf des Schlüssels verständigen. Dazu muß aber eine Vorgehensweise gewählt werden, die es einem Angreifer nicht möglich macht, einen Denial-of-Service Angriff durchzuführen, indem er als einer der Partner auftritt, den Schlüssel für ungültig erklärt und damit die sichere Kommunikation zwischen den Partnern unterbricht.

Die sicherste Methode des Widerrufs ist auch hier eine Out-of-Band Verbindung. Ist dies nicht möglich, so können die beiden Partner neben einem gemeinsamen Schlüssel auch noch ein geheimes Kennwort vereinbaren, das nur zum Widerruf des Schlüssels verwendet werden darf.

Der Widerruf von asymmetrischen Schlüsseln wird in Abschnitt 6.2.3 beim Zertifikatsmanagement behandelt.

Generell ist es sinnvoll, für Schlüssel eine feste *Gültigkeitsdauer (Lebensdauer)* zu vereinbaren, um die kryptologische Sicherheit zu erhöhen. Denn je länger ein Schlüssel verwendet wird, desto größer ist die Gefahr, daß er kompromittiert wird und desto größer ist der Verlust, der dadurch entsteht, daß ein Angreifer große Mengen von Daten nachträglich entschlüsseln kann, wenn er im Besitz des Schlüssels ist. Für jedes kryptographische Verfahren muß die Sicherheitspolitik festlegen, wie lange die Schlüssel maximal gültig sein sollen. Dabei gilt grundsätzlich, je öfter die Schlüssel verwendet werden und je größer die damit verschlüsselte Datenmenge bzw. je sensibler die Daten sind, desto kürzer soll die Lebensdauer des Schlüssels sein.

6.2 Zertifikatsmanagement

Ein *Zertifikat* bindet die Identität eines Kommunikationspartners (Netzwerk, Host, Nutzer oder Anwendung) sicher an seinen öffentlichen Schlüssel. Dazu wird eine vertrauenswürdige Instanz benötigt. Derjenige, der ein Zertifikat anfordert (*Subject*), muß einer *Zertifizierungsstelle (Certification Authority, CA)* seine Identität und die Richtigkeit seines Schlüssels beweisen. Die Zertifizierungsinstanz (*Issuer*) überprüft die Angaben und falls der Schlüssel vom angegebenen *Subject* stammt, signiert die CA den Schlüssel mit ihrem geheimen Schlüssel. Das Subject, das mittels einer *Zertifikatsanforderung (Certification Request)* ein Zertifikat beantragt, muß über einen *Distinguished Name (DN)* die Informationen bezüglich seiner Identität bekanntgeben. Ein DN besteht aus `<Variable>:<Wert>` Paaren, z.B.

- `Country (C)`: Landescode (zwei Buchstaben) z.B. DE
- `State or Provience (ST)`: Staat oder Provinz (Bundesland)

- `Organization (O)`: Name der Organisation oder Firma
- `Organizational Unit (OU)`: Abteilung
- `Locality (L)`: Ort, in der die Firma bzw. Abteilung ihren Sitz hat
- `Common Name (CN)`: Name des Subject; dies kann entweder eine E–Mail Adresse oder der volle Name eines Benutzers oder aber die Bezeichnung eines Servers (`hostname.domain`) sein.

Es wird davon ausgegangen, daß der öffentliche Schlüssel der CA allseits bekannt ist, d.h. jeder Kommunikationspartner kann die Signatur eines Zertifikates und damit das Zertifikat selbst verifizieren.

Ist die Identität des Subjekts nicht sicher an seinen öffentlichen Schlüssel gebunden, so kann der böswillige Angreifer Mallet durch eine Man-in-the-Middle Attack die Kommunikation zwischen Alice und Bob folgendermaßen angreifen.

1. Alice sendet Bob ihren öffentlichen Schlüssel. Mallet fängt diesen Schlüssel ab und sendet stattdessen seinen eigenen.
2. Bob schickt Alice seinen öffentlichen Schlüssel. Auch diesen ersetzt Mallet durch seinen eigenen.
3. Wenn Alice nun eine Nachricht an Bob schickt, die mit „Bob's“ öffentlichem Schlüssel verschlüsselt ist, kann Mallet die Nachricht abfangen und, da sie in Wirklichkeit mit seinem öffentlichen Schlüssel verschlüsselt wurde, auch entschlüsseln. Da er Bob's öffentlichen Schlüssel kennt, kann er die Nachricht, evtl. verändert, mit Bob's Schlüssel verschlüsselt an ihn weiterschicken.
4. Analog kann er alle Nachrichten von Bob an Alice abfangen, lesen, verändern und weiterleiten.

Dieser Angriff funktioniert auch, wenn die öffentlichen Schlüssel in einer Datenbank gespeichert sind. Mallet muß dann lediglich die Antworten auf Alice's bzw. Bob's Anfragen an die Datenbank abfangen und entsprechend verändern.

Der Angriff ist besonders schwerwiegend, da weder Bob noch Alice etwas davon merken können, falls Mallet so schnell ist, daß keine merkliche Verzögerung im Kommunikationsablauf eintritt, und die beiden Kommunikationspartner nicht Out-of-Band miteinander kommunizieren. Dies ist insbesondere dann der Fall, wenn Alice und Bob stellvertretend für zwei Prozesse stehen.

Dieses Szenario verdeutlicht auch, welche katastrophalen Folgen es haben kann, falls der Schlüssel einer Zertifizierungsstelle gebrochen werden kann und Mallet damit in der Lage wäre, „gültige“ Zertifikate mit seinem öffentlichen Schlüssel für beliebige Subjekte ausstellen zu können.

6.2.1 Generierung von Zertifikaten

Für die Erzeugung von Zertifikaten kann ein zentraler bzw. hierarchischer oder ein dezentraler Ansatz gewählt werden. Im zentralen bzw. hierarchischen Fall gibt es eine ausgezeichnete CA (*Root–CA*) die ihrerseits weitere Sub–CA's oder Endbenutzer zertifizieren. Auf diese Weise entsteht eine Baumstruktur von Zertifizierungsstellen. Um in diesem Fall ein Endbenutzerzertifikat zu verifizieren, müssen die Zertifikate der Sub–CA's verifiziert werden, solange bis eine vertrauenswürdige CA, von der der öffentliche Schlüssel bekannt ist, oder die Wurzel–CA der Zertifizierungshierarchie erreicht ist. Diese Kette von zu prüfenden Zertifikaten wird als *Zertifizierungspfad (Certification Path)* bezeichnet.

Im dezentralen Fall übernehmen die Kommunikationspartner selbst die Aufgabe von Zertifizierungsinstanzen. Jeder kann als sog. *Introducer* für einen anderen Kommunikationspartner auftreten. Alice kann bspw. den öffentlichen Schlüssel von Bob signieren, nachdem sie sich vergewissert hat, daß dieser Schlüssel auch wirklich Bob gehört. Bob kann seinen Schlüssel auch noch von anderen unterschreiben lassen und seinerseits die Schlüssel von Dritten signieren. Auf diese Art entsteht ein „*Web of Trust*“. Die Länge des Zertifizierungspfades besitzt in diesem Fall nicht notwendigerweise eine obere Schranke, wie dies beim hierarchischen Ansatz der Fall ist. Allerdings kann der Nutzer im dezentralen Fall selbst sehr genau bestimmen, wem er als Zertifizierungsinstanz traut und wem nicht. Im hierarchischen Fall ist er gezwungen, den Zertifizierungsstellen, insbesondere der Root–CA, und deren Politiken zur sicheren Prüfung von Zertifikatsanforderungen „blind“ zu vertrauen.

Im neuen *Gesetz zur digitalen Signatur (SiG)* als Teil des *Informations– und Kommunikationsdienste–Gesetz (IuKDG)* [Bun97], das am 1. August in Kraft getreten ist, wird ein zentraler Ansatz verfolgt. Die Regulierungsbehörde[3] zertifiziert als Root–CA die Schlüssel von Zertifizierungsstellen. Der Betrieb dieser Zertifizierungsstellen kann privatwirtschaftlich erfolgen und muß von der Regulierungsbehörde genehmigt werden. Die Voraussetzungen, d.h. die Policy, für die Genehmigung von Zertifizierungsstellen sind in §4 SiG geregelt. Sie sind durch eine von der Bundesregierung zu erlassende Ausführungsverordnung zu präzisieren.

6.2.2 Speicherung und Verteilung von Zertifikaten

Da die Zertifikate durch die digitale Signatur der CA vor Veränderung geschützt sind, sind keine Vorkehrungsmaßnahmen hinsichtlich der Speicherung oder der Verteilung zu treffen. Um ein gültiges Zertifikat unerkannt verändern zu können, müßte der Angreifer im Besitz des privaten Schlüssels der CA sein.

Damit gültige Zertifikate nicht unbeabsichtigt gelöscht werden können, muß jedoch gefordert

[3] Nach telefonischer Auskunft vom 31.08.97 durch Herrn Tettenborn vom Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie existiert diese Regulierungsbehörde derzeit noch nicht und wird voraussichtlich im Herbst 1997 eingerichtet. In der Übergangszeit ist das Ministerium für Post und Telekommunikation für die Zertifizierung von CA's verantwortlich.

werden, daß sie sowohl von der CA als auch auf Seiten des Zertifizierten entsprechend durch Sicherheitskopien geschützt werden.

Bei der Verteilung ist nur darauf zu achten, daß der Anfordernde das Zertifikat auch tatsächlich erhält. Da die Initiative vom Empfänger ausgeht, entweder durch eine Zertifikatsanforderung für sein eigenes Zertifikat oder im Rahmen eines Verifikationsprozesses für ein Zertifikat eines Dritten, kann davon ausgegangen werden, daß der Empfänger solange „nachfragen“ wird, bis er das Zertifikat erhalten hat. Als Mechanismen zur Verteilung können alle Verfahren verwendet werden, die auch zur Übertragung normaler Daten Verwendung finden, z.B. E-Mail, WWW, ftp, Telnet, usw.

6.2.3 Widerruf von Zertifikaten

Zertifikate besitzen i.d.R. eine bestimmte Gültigkeitsdauer, d.h. einen im Zertifikat festgelegten Zeitraum, innerhalb dessen sie verwendet werden können. Vor bzw. nach diesem Zeitraum sind die Zertifikate als ungültig zu betrachten. Trotzdem muß auch bei asymmetrischen Verschlüsselungsverfahren die Möglichkeit des Schlüsselwiderrufs gewährleistet sein, falls bspw. der private Schlüssel kompromittiert wurde. In diesem Fall muß der öffentliche Schlüssel bzw. das entsprechende Zertifikat für ungültig erklärt werden.

Im Gegensatz zu symmetrischen Verfahren, bei denen dem Besitzer eines Schlüssels genau bekannt ist, mit wem er dieses gemeinsame Geheimnis teilt, ist es bei asymmetrischen Verfahren nicht möglich zu überprüfen, wer im Besitz des öffentlichen Schlüssels eines bestimmten Kommunikationspartners ist. D.h. falls ein Schlüssel widerrufen werden muß, kann der Eigentümer die Nutzer seines öffentlichen Schlüssels nicht ermitteln und damit auch nicht benachrichtigen. Die Zertifizierungsstelle übernimmt daher die Aufgabe, die öffentlichen Schlüssel bzw. die von ihr ausgestellten Zertifikate zu widerrufen.

Die CA vereinbart dazu mit dem Zertifizierten ein Paßwort, mit Hilfe dessen der Zertifizierte sein Schlüsselpaar zurücknehmen kann. Die CA führt eine *Certificate Revocation List (CRL)*, in der alle widerrufenen Zertifikate, deren Gültigkeit noch nicht abgelaufen ist, aufgenommen werden. In dieser CRL wird nicht das gesamte Zertifikat aufgenommen, sondern nur dessen eindeutige Seriennummer. Um die CRL vor Veränderung bzw. Fälschung zu schützen, wird sie von der CA signiert. Diese CRL wird in regelmäßigen Abständen veröffentlicht und jeder Kommunikationspartner kann überprüfen, ob die von ihm verwendeten bzw. akzeptierten Zertifikate überhaupt noch gültig sind. Dazu muß er sich allerdings regelmäßig die CRL von den verschiedenen CA's besorgen und die Gültigkeit der Zertifikate auch überprüfen. Andernfalls ist er nicht in der Lage, eventuelle Angriffe zu erkennen, solange er von der Gültigkeit der Zertifikate überzeugt ist.

Wird das Zertifikat bzw. der entsprechende private Schlüssel zur digitalen Signatur verwendet, so muß auch der Zeitpunkt des Widerrufs betrachtet werden. Ein Sender könnte sonst eine Vereinbarung oder einen Vertrag signieren und vor der Erfüllung seiner Vertragspflichten sein Zertifikat zurücknehmen und behaupten, er habe den Vertrag nie signiert. Insbesondere wenn der Widerruf eines Zertifikates rückwirkend erfolgen soll, ist besondere Vorsicht geboten. Diese Problematik und entsprechende Regeln müssen sowohl in der Policy der CA als auch in der Sicherheitspolitik des Unternehmens geklärt werden.

Nach einem gültigen Widerruf darf ein Zertifikat, das zur digitalen Signatur verwendet werden konnte, nicht gelöscht werden. Um auch zu einem späteren Zeitpunkt eine digitale Signatur belegen zu können (z.B. vor Gericht), muß das Zertifikat, zumindest von der CA, archiviert werden.

Kapitel 7

Geschlossene Benutzergruppen und Tunneling

Sind mehrere Mitarbeiter eines Projektes räumlich verteilt in einer Intra– bzw. Internetumgebung und kommunizieren diese, mit dem Ziel der Zusammenarbeit, über ein Rechen– bzw. Netzsystem, so spricht man von *CSCW (Computer Supported Cooperative Work)*. Aus Sicht der Netzsicherheit ist dabei insbesondere die Bildung von *geschlossenen Benutzergruppen* über offene Netze ein zu lösendes Problem. Die geschlossene Benutzergruppe muß über ein offenes Netz so kommunizieren können, als ob alle Beteiligten sich innerhalb eines abgeschlossenen LAN befinden würden. Insbesondere sollen Nachrichten, die an Gruppenmitglieder geschickt werden, nur von diesen gelesen werden können.

Eine Möglichkeit zur Bildung geschlossener Benutzergruppen stellt Tunneling dar. Mit Hilfe von Tunneling läßt sich eine als „*Tunnel*“ bezeichnete Verbindung zwischen zwei (oder mehreren) Gateways bzw. Routern und damit zwischen LAN's oder aber auch zwischen Endsystemen, über offene Netze realisieren.

Eine im Rahmen der Migration zu *IPv6 (Internet Protocol, Version 6)*[1] [DH96, HD96] angewandte Technik ist *IP in IP Tunneling* [Sim95, GN96]. Hierbei wird eine IPv6–*PDU (Protocol Data Unit)* in eine IPv4 Nachricht verpackt, um die IPv6–PDU über Netze übertragen zu können, auf denen lediglich IPv4 implementiert ist. Damit wird eine Benutzergruppe von IPv6 Anwendern über IPv4 Netze gebildet. Da die IPv6–PDU's im Klartext innerhalb der IPv4 Pakete übertragen werden, wird jedoch das Mitlesen durch Dritte nicht verhindert.

Um eine geschlossene Benutzergruppe im Sinne dieser Arbeit realisieren zu können, muß ein Tunnel die Vertraulichkeit und Integrität der Verbindung sichern. Eine Möglichkeit zur Realisierung eines solchen Tunnels sind Verschlüsselungsalgorithmen und –Protokolle, die auf TCP/IP aufsetzen (vgl. Abschn. 8 und 9) oder innerhalb der Anwendungsschicht realisiert sind (vgl.

[1] Working Group der IETF: IP Next Generation (IPng), zur Entwicklung der nächsten Generation des IP–Protokolls: http://playground.sun.com/pub/ipng/html/ipng-main.html

Abschn. 10 und 11). Damit lassen sich entweder Tunnel für einzelne Anwendungen realisieren oder relativ kleine, verstreute Gruppen bilden. Will man aber zwei LAN über ein offenes Netz zu einer geschlossenen Benutzergruppe verbinden, so braucht man Mechanismen, die nicht nur auf Endsystemen, sondern auch auf Vermittlungssystemen, wie z.B. Routern implementiert werden können.

7.1 Tunneling über IP

Zur Absicherung des IP–Verkehrs wurden 1995 zwei Protokollerweiterungen spezifiziert [Atk95a]. *IP Authentication Header (AH)* [Atk95b] ist in der Lage, die Authentisierung und die Integrität von IP–Paketen durchzusetzen. Zur Sicherung der Vertraulichkeit wurde *IP Encapsulation Security Payload (ESP)* [Atk95c] definiert. Diese Protokolle können auch zur Realisierung von Tunneling verwendet werden.

7.1.1 IP Authentication Header (AH)

AH sichert die Integrität eines IP–Paketes dadurch, daß der IP–Header durch den Authentication Header erweitert wird. Der AH wird nach den Headerteilen, die von Vermittlungsrechnern benötigt werden und vor den Informationen, die von Vermittlungsrechnern nicht gelesen werden, eingefügt (vgl. Abb. 7.1).

IP Version 4 (IPv4)

IPv4–Header	Authentication Header (AH)	User Data (höhere Protokolle, z.B. TCP)

IP Version 6 (IPv6)

IPv6–Header	Hop–by–Hop/Routing	Authentication Header	Andere Header	UD

Abbildung 7.1: Authentication Header bei IPv4 und IPv6

Bei IPv4 wird der AH zwischen IP–*PCI (Protocol Control Information)* und IP–*UD (User Data)* eingefügt. IPv6 unterscheidet einen *Basis–* und *Erweiterungsheader*. AH stellt einen Erweiterungsheader dar, der nach dem Basisheader und ggf. nach den Erweiterungsheadern Hop–by–Hop bzw. Routing eingefügt wird. Der Authentication Header selbst ist in Abb. 7.2 dargestellt.

Die Integrität der IP–Nachricht wird durch die Berechnung eines Message Authentication Codes (MAC) gesichert. Der MAC wird über das gesamte IP–Paket berechnet. Veränderliche Felder, d.h. Felder, die in Transitsystemen verändert werden, z.B. Time to Live bei IPv4 oder Hop Counter bei IPv6, werden bei der Berechnung des MAC–Wertes auf Null gesetzt. Der berechnete Hash–Wert wird im Feld Authentication Data an den Empfänger übertragen. Um den MAC selbst zu schützen, wird er entweder verschlüsselt (z.B. mittels ESP) oder vor der Berechnung des MAC wird die IP–PDU mit einem geheimen Schlüssel konkateniert (vgl. Abschn. 3.4).

Bit	8	16	32
	Next Header	Length	RESERVED
	Security Parameter Index (SPI)		
	Authentication Data (variabel)		

Abbildung 7.2: IP Authentication Header

Eine als *Security Association* bezeichnete sichere Verbindung, bestimmt die verwendeten Algorithmen, die Modi, Schlüssel und sonstige Optionen. Die Security Association selbst ist unidirektional, d.h. in jede Richtung der Kommunikation können verschiedene Algorithmen verwendet werden. Sie wird durch die Zieladresse und den *Security Parameter Index (SPI)*, eine 32 Bit lange Zahl, festgelegt und ist zwischen den Partnern zu vereinbaren. Die Authentisierung der Partner wird durch die Kenntnis der geheimen Schlüssel gewährleistet.

Der Empfänger kann aus der Adresse und dem SPI die Security Association und damit die verwendeten Algorithmen und Schlüssel bestimmen. Er berechnet ebenfalls den Hash–Wert über das IP–Paket und vergleicht diesen mit dem im Authentication Data Feld übertragenen. Stimmen die Werte überein, wird die Nachricht akzeptiert, andernfalls wird die Nachricht verworfen, nachdem der SPI, Datum und Zeit der Ankunft, sowie Sender und Empfängeradresse in einem Log gespeichert wurden.

Als MAC–Algorithmen sind *MD5 (Message Digest Nr. 5)* [Riv92, MS95a], *HMAC-MD5-96* [OG97, KBC97], *SHA (Secure Hash Standard)* [Nat95, MS95b] und *HMAC-SHA-1-96* [CG97b] spezifiziert. Falls HMAC-SHA-1-96 bzw. HMAC-MD5-96 verwendet wird, ist das Authentication Data Feld auf eine Länge von 96 Bit beschränkt. Von dem 128 Bit langen MD5 Wert bzw. dem 160 Bit langen SHA Wert werden dann nur die ersten 96 Bit verwendet.

7.1.2 IP Encapsulation Security Payload (ESP)

Die Vertraulichkeit von IP–Daten wird durch Verschlüsselung und Übermittlung in ESP erreicht. Encapsulation Security Payload wird innerhalb der IP–UD (*Payload*) übertragen und setzt sich selbst aus einem ESP–Header und einem –Body zusammen. Der erste Teil von ESP enthält unverschlüsselte Felder, der zweite Teil verschlüsselte Daten (vgl. Abb. 7.3).

Der Header selbst hängt vom verwendeten Verschlüsselungsverfahren ab. Das erste Feld ist aber immer der SPI, der in Verbindung mit der Zieladresse die Security Association und damit die verwendeten Algorithmen bestimmt.

ESP kennt zwei Betriebsmodi. Im *Transport Mode* werden nur die IP–User Data, d.h. die Daten der höheren Protokolle verschlüsselt und in ESP verpackt. Im *Tunnel Mode* wird die gesamte IP–PDU verschlüsselt und innerhalb von ESP übertragen, d.h. die ESP–PDU wird in ein neues

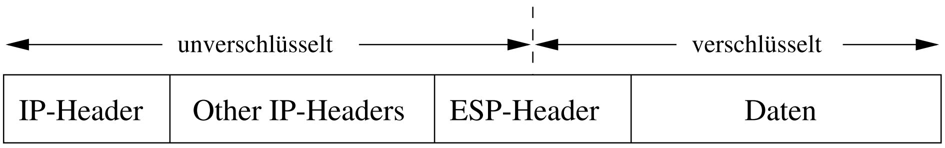

Abbildung 7.3: Encapsulation Security Payload

IP–Paket verpackt und dann verschickt. Mit beiden Modi kann ein Tunnel zwischen zwei oder mehreren End– bzw. Vermittlungssystemen realisiert werden.

Um auch die Integrität zu sichern, kann ESP in Verbindung mit AH verwendet werden. Wird ESP im Transport Mode betrieben, so wird der AH nach ESP angewendet. Zuerst wird die Verschlüsselung durchgeführt und ESP innerhalb einer IP–PDU verpackt, dann wird über dieses IP–Paket der Hash–Wert berechnet und der AH hinzugefügt.

Im Tunnel Mode kann AH vor ESP angewendet werden, d.h. zuerst wird der Hash–Wert über die IP–PDU berechnet und der AH dem IP–Paket hinzugefügt. Dann wird das gesamte Paket verschlüsselt in ESP verpackt und um einen neuen IP–Header erweitert. Als Verschlüsselungsalgorithmus werden DES im *CBC (Cipher Block Chaining)* Mode [Nat93, Nat80, MKS95], *3DES* im CBC Mode [PT97], *DESX* [SB97], *ARCFOUR* [Tha97], *Blowfish* [Sch93, Ada97a], und *IDEA* [LM90, Lai92, Ada97b] spezifiziert.

7.2 Bewertung

ESP ist in Verbindung mit AH in der Lage, ein breites Spektrum in der Dimension der Kommunikationspartner abzudecken. Es bietet die Möglichkeit Tunnel, sowohl zwischen Endsystemen als auch zwischen End– und Vermittlungssystemen oder nur zwischen Vermittlungssystemen, zu realisieren. Auf Seite der Sicherheitsanforderungen läßt sich Integrität, Vertraulichkeit, Authentisierung und bei AH einfaches Auditing durchsetzen. Maßnahmen zur Sicherung der Verbindlichkeit sind nicht vorgesehen. Die beiden Protokolle bieten auch keinen Schutz vor Verkehrsflußanalysen.

Durch die architekturelle Einordnung innerhalb des IP–Protokolls, lassen sich prinzipiell alle Datenströme, die mit Hilfe von IP übertragen werden, absichern. Allerdings besteht das Problem, daß pro Endsystem und nicht pro Endbenutzer gesichert werden kann, da die Security Association, und damit die Schlüssel, an den IP–Protokoll Stack gebunden werden. Das bedeutet, daß i.allg. alle Benutzer, die Zugang zu dem entsprechenden System haben, die übertragenen Daten entschlüsseln könnten. Zur Bildung geschlossener Benutzergruppen oder zur Verbindung von Subnetzen mittels Tunneling ist die Kombination der beiden Protokolle jedoch sehr gut geeignet.

Ein großes Problem ist, daß die Spezifikation keinerlei Aussagen zum Schlüsselaustausch oder zur Aushandlung von Security Associations macht, sondern auf ein zu spezifizierendes Protokoll zum Schlüsselaustausch verweist. Zum jetzigen Zeitpunkt gibt es verschiedene Vorschläge für ein Schlüsselmanagement (z.B. [HFP97, BHH97, FA97, CF97, MSS97, HC97, Mye97, Hor97, KSH97]), aber bisher hat sich noch kein Konzept durchgesetzt. Die *IP Security Protocol Working Group (IPSEC)*[2] vertritt ISAKMP/Oakley [MSS97, HC97] und will dies als *Proposed Standard* bei der *IETF (Internet Engineering Task Force)* standardisieren.

AH und ESP werden sowohl für IPv4 als auch für IPv6 spezifiziert. Allerdings gibt es keine IPv4 Implementierung, die AH und ESP unterstützt und auch bei IPv6 ist die Implementierung der Protokolle noch nicht sehr verbreitet.

Bei einer zukünftigen Migration zu IPv6 und einer breiteren Unterstützung der Protokolle ist AH und ESP eine gute Möglichkeit, vertrauliche Tunnel zu realisieren.

[2] http://www.ietf.cnri.reston.va.us/html.charters/ipsec-charter.html

Kapitel 8

Secure Socket Layer (SSL)

SSL ist ein Protokoll, das die Vertraulichkeit und Integrität einer Kommunikation sichert und die Kommunikationspartner gegenseitig authentisiert. Es wurde von der Netscape Communication Corporation entwickelt und ist im Moment als Internet–Draft veröffentlicht ([FKK96, DA97]). Das SSL–Protokoll selbst besteht aus zwei Teilen, dem SSL Handshake Protocol und dem SSL Record Layer. SSL liegt unterhalb der Anwendungsschicht und oberhalb der TCP–Ebene (vgl. Abb. 8.1). Damit lassen sich Anwendungen und Protokolle, die auf TCP/IP aufsetzen, wie bspw. HTTP, Telnet, ftp, NNTP u.ä. sichern.

⋮
SSL Handshake Protocol
SSL Record Layer
TCP
IP
⋮

Abbildung 8.1: Architekturelle Einordnung von SSL

In SSL werden Sitzungen (*Sessions*) und Verbindungen (*Connections*) unterschieden. Ein Kommunikationspartner kann mehrere SSL–Sitzungen gleichzeitig geöffnet haben, und eine Sitzung kann mehrere Verbindungen beinhalten. Die Sitzung bzw. Verbindung wird durch die, in Tabelle 8.1 angegebenen Attribute, bestimmt.

SSL verwendet ein hybrides Verschlüsselungsverfahren. Zur Übertragung von gemeinsamen Geheimnissen werden asymmetrische Verschlüsselungsverfahren verwendet. Zur Verwendung kommen hierbei *RSA* [RSA78], Diffie–Hellman (vgl. S. 34) und *Fortezza* [Nat96]. Nutzdaten werden mit einem symmetrischen Verfahren verschlüsselt. Hier können *RC2*, *RC4*, DES, 3DES sowie Fortezza eingesetzt werden. Die Integrität der Daten wird mittels MAC gesichert und die Authentisierung mit Hilfe von Zertifikaten durchgeführt. Der MAC wird zusätzlich durch einen Schlüssel gesichert, d.h. die Daten werden vor der Berechnung des MAC mit dem Schlüssel konkateniert. Als Algorithmen sind hier MD5 und SHA spezifiziert.

Attribut	Bedeutung
SSL–Sitzung	
`SessionID`	beliebige Bytesequenz, die der Server zur Identifikation einer Sitzung festlegt
`Peer Certificate`	X509v3–Zertifikat des jeweiligen Kommunikationspartners
`CompressionMethod`	Kompressionsalgorithmus
`ChipherSpec`	verwendete Verschlüsselungs– und MAC–Algorithmen sowie die entsprechenden kryptographischen Attribute
`MasterSecret`	gemeinsames Geheimnis zwischen Client und Server (48 Bit)
`isResumable`	Boolescher Wert, der angibt, ob diese Sitzung zum Aufbau neuer Verbindungen genutzt werden kann
SSL–Verbindung	
`Server.Random` `Client.Random`	Zufallszahl für jede neue Verbindung; zur Berechnung der Kommunikationsschlüssel
`server_write_MAC_secret` `client_write_MAC_secret`	Schlüssel, mit dem Server bzw. Client ihren MAC sichern
`server_write_key` `client_write_key`	Schlüssel für symmetrisches Verschlüsselungsverfahren
`server_write_IV` `client_write_IV`	Initialisierungsvektor für symmetrisches Verschlüsselungsverfahren
`SequenceNumber`	Jeder Partner führt separate Sequenznummern für gesendete und empfangene Pakete.

Tabelle 8.1: Attribute einer SSL–Sitzung und einer SSL–Verbindung

8.1 SSL Record Layer

Der *SSL Record Layer* erhält Daten von höheren Schichten in nichtleeren Blöcken beliebiger Länge. Diese Blöcke werden ggf. fragmentiert und in `SSLPlaintext` Blöcke mit maximal 2^{14} Bytes Länge verpackt. Die Blockstruktur einer höheren Schicht wird dabei nicht notwendigerweise beibehalten. Es können bspw. mehrere unabhängige Blöcke desselben höheren Protokolls in einem `SSLPlaintext` Block zusammengefaßt werden.

Jeder `SSLPlaintext` Block einer bestimmten Sitzung wird, entsprechend des vereinbarten Kompressionsverfahrens, in den `SSLCompressed` Block komprimiert. Das `CipherSpec` Attribut der

Sitzung bestimmt den MAC- und den Verschlüsselungsalgorithmus. Zuerst wird der MAC des `SSLCompressed` Blocks berechnet. Er wird durch einen geheimen Schlüssel gesichert. Außerdem wird die `SequenceNumber` mit in die MAC–Berechnung einbezogen, um Replay Angriffe zu verhindern. Auf MAC und `SSLCompressed` Block wird dann das Verschlüsselungsverfahren angewendet, dadurch entsteht der `SSLCiphertext`. Der `SSLCiphertext` Block wird mittels TCP an den Kommunikationspartner übermittelt.

8.2 SSL Handshake Protocol

Wenn ein Client und ein Server eine SSL–Verbindung aufbauen wollen, dient das *SSL Handshake Protocol* dazu, die verwendete Protokollversion, die verwendeten kryptologischen Algorithmen und die Public Key Verfahren zur Generierung des Kommunikationsschlüssels abzustimmen sowie ggf. (optional) die Partner gegenseitig zu authentisieren.

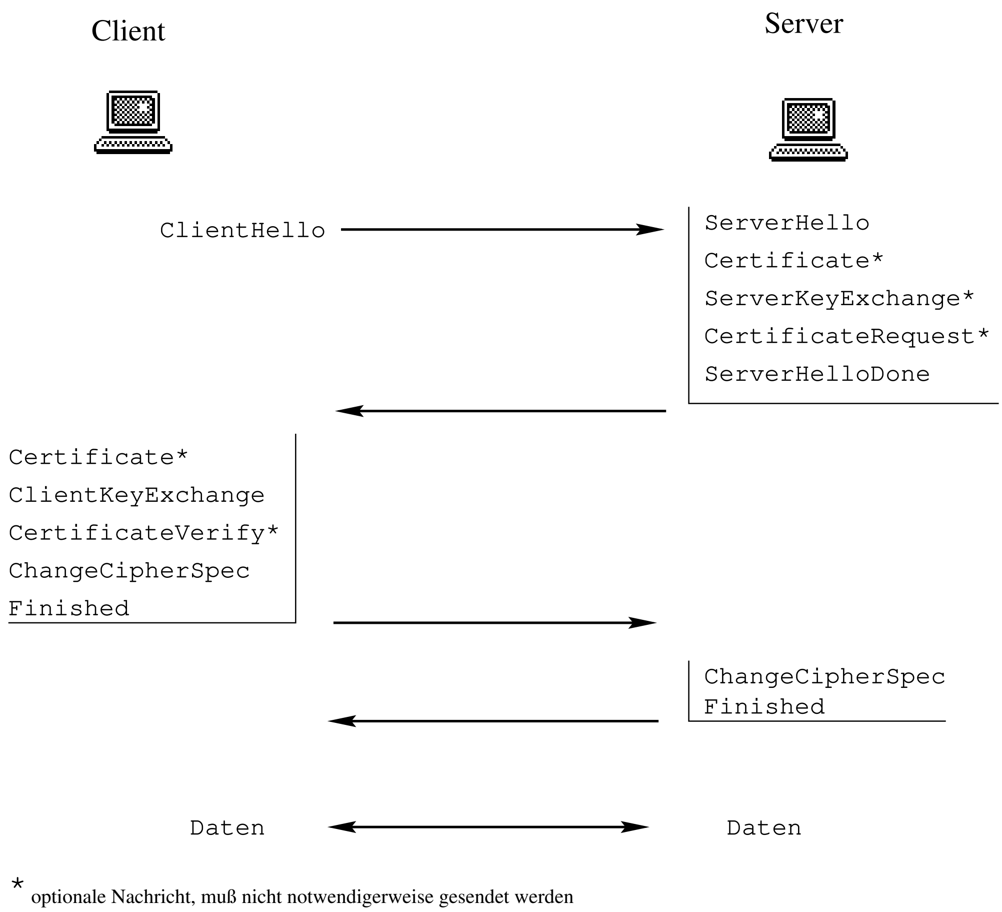

Abbildung 8.2: SSL Handshake Protocol

Der Protokollablauf ist in Abb. 8.2 dargestellt. Der Client, der die Verbindung initiiert, sendet ein `ClientHello`, das der Server mit einem `ServerHello` beantworten muß, andernfalls wird die Verbindung abgebrochen. Mit Hilfe der Hello Nachrichten legen die beiden die verwendete Protokollversion, die `SessionID` sowie das Verschlüsselungs–, MAC– und Kompressionsverfahren fest. Der Client schlägt dazu in seiner `ClientHello` Nachricht eine Liste von Verfahren, nach absteigender Präferenz sortiert, vor. Der Server wählt davon jeweils ein Verfahren aus und schickt den entsprechenden Identifikator in seiner `ServerHello` Nachricht zurück. Zusätzlich werden die beiden Zufallszahlen `ClientHello.random` und `ServerHello.random` ausgetauscht, die später zur Berechnung der Kommunikationsschlüssel dienen.

Falls der Server sich authentisieren muß, sendet er sein Zertifikat in der `Certificate` Nachricht an den Client. Er kann seine öffentlichen Schlüssel aber auch in einer `ServerKeyExchange` Nachricht senden, wenn er bspw. kein Zertifikat besitzt, oder das Zertifikat nur für Signaturzwecke (z.B. bei *DSS; Digital Signature Standard* [Nat94]) verwendet werden darf. Zusätzlich kann er mit einem `CertificateRequest` den Client zur Authentisierung auffordern. Zum Abschluß sendet der Server ein `ServerHelloDone` um den Abschluß der Hello Phase anzuzeigen und auf die Antwort des Client zu warten.

Falls der Server das Zertifikat des Client angefordert hat, muß dieser mit der `Certificate` Nachricht oder einem `NoCertificate` Alarm antworten. Die `ClientKeyExchange` Nachricht hängt vom ausgehandelten Public Key Verfahren ab und dient zur Übertragung eines gemeinsamen Geheimnisses (`premaster_secret`), das zur Berechnung der Kommunikationsschlüssel verwendet wird. Wird z.B. RSA verwendet, so enthält die `ClientKeyExchange` Nachricht das 48 Byte lange `premaster_secret`, das mit dem öffentlichen Schlüssel des Servers aus dessen Zertifikat oder aus der `ServerKeyExchange` Nachricht verschlüsselt wird.

Falls der Client ein Zertifikat geschickt hat, das auch die Möglichkeit der digitalen Signatur erlaubt, sendet er nun eine signierte `CertificateVerify` Nachricht, um sein Zertifikat bzw. den darin enthaltenen öffentlichen Schlüssel zu verifizieren. Er beweist durch die Signatur der Nachricht, daß er im Besitz des entsprechenden privaten Schlüssels ist.

Nun sendet der Client die `ChangeCipherSpec` Nachricht, um auf das vereinbarte Verschlüsselungsverfahren, mit den entsprechenden Initialisierungen, zu wechseln. Dann sendet er sofort die `Finished` Nachricht, die bereits mit den entsprechenden Schlüsseln und dem vereinbarten Verfahren verschlüsselt wird. Der Server antwortet mit einer eigenen `ChangeCipherSpec` und `Finished` Nachricht. Damit ist das Handshake Protocol abgeschlossen und die Datenübertragung kann beginnen.

Falls Client und Server eine vorausgegangene Sitzung wiederaufnehmen oder eine bestehende Sitzung duplizieren wollen, so greift ein verkürztes Handshake Protocol (vgl. Abb. 8.3). Der Client sendet eine `ClientHello` Nachricht mit der `SessionID` der Sitzung, die wiederaufgenommen werden soll. Der Server überprüft seinen Cache, ob er die Attribute dieser Sitzung noch gespeichert hat. Falls er die `SessionID` mit den zugehörigen Parametern findet und gewillt ist, die Sitzung wiederaufzunehmen, sendet er sein `ServerHello` mit derselben `SessionID`. Dann müssen Client und Server die `ChangeCipherSpec` sowie die `Finished` Nachricht senden, um das

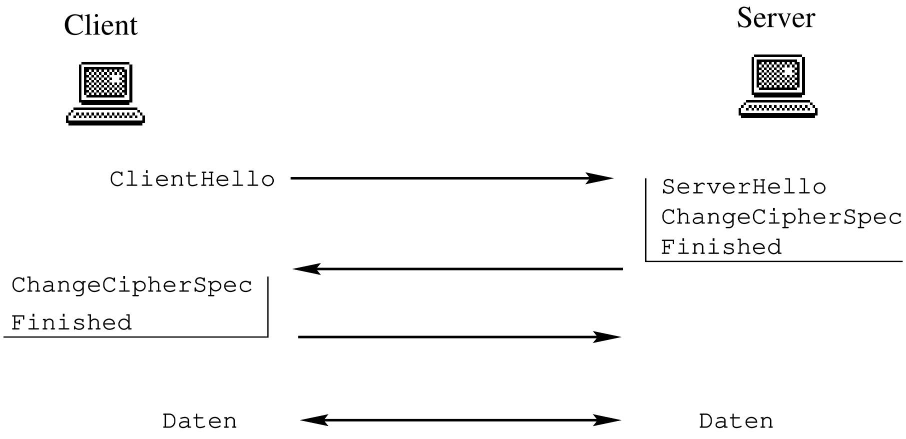

Abbildung 8.3: Verkürztes SSL Handshake Protocol

Protokoll abzuschließen.

8.3 Berechnung der Sitzungsschlüssel

Das `premaster_secret`, das im SSL Handshake Protocol ausgetauscht wurde, und die Zufallszahlen aus den Hello Nachrichten dienen dazu, einen `key_block` zu berechnen. Aus dem `key_block` wiederum werden Schlüssel zur Sicherung des MAC (`server_write_MAC_secret`, `client_write_MAC_secret`), die Kommunikationsschlüssel (`server_write_key`, `client_write_key`) und evtl. Initialisierungsvektoren (`server_write_IV`, `client_write_IV`) abgeleitet.

Zur Berechnung des `key_block` werden die Hash–Algorithmen MD5 und SHA verwendet. Im folgenden wird unter $MD5(x)$ die Anwendung des MD5 Algorithmus auf den Bitstring x und unter $x + y$ die Konkatenation der Bitstrings x und y verstanden. Die Länge des `key_block` hängt von den erforderlichen Schlüssellängen und den ausgehandelten Algorithmen ab.

```
key_block :=
MD5(premaster_secret + SHA('A' +
      premaster_secret + ServerHello.random + ClientHello.random)) +
MD5(premaster_secret + SHA('BB' +
      premaster_secret + ServerHello.random + ClientHello.random)) +
MD5(premaster_secret + SHA('CCC' +
      premaster_secret + ServerHello.random + ClientHello.random)) + [...];
```

In den Vereinigten Staaten fallen Software–Produkte, die kryptologische Protokolle oder Verschlüsselungsverfahren implementieren, unter strenge *Exportbeschränkungen*, da sie mit Kriegswaffen gleichgesetzt werden. Es dürfen nur solche Produkte aus den USA exportiert werden, deren maximale Schlüssellänge beschränkt wird. Daher werden die verwendeten Schlüssel in der US–Version auf andere Weise als in der Exportversion berechnet.

8.3.1 US–Version

Von dem `key_block` werden in der folgenden Reihenfolge Bitstrings der angegebenen Länge entfernt und den entsprechenden Variablen zugewiesen.

```
client_write_MAC_secret := key_block[CipherSpec.hash_size]
server_write_MAC_secret := key_block[CipherSpec.hash_size]
client_write_key := key_block[CipherSpec.key_material]
server_write_key := key_block[CipherSpec.key_material]
client_write_IV := key_block[CipherSpec.IV_size]
server_write_IV := key_block[CipherSpec.IV_size]
```

8.3.2 Exportversion

Bei der Exportversion von SSL sind die Schlüssel zur Verschlüsselung auf 40 Bit, die zur Sicherung des MAC auf 128 Bit beschränkt, d.h. der `key_block` hat eine Länge von 42 Byte. Zu seiner Berechnung müssen lediglich drei Ausgaben von MD5 konkateniert werden. Die Schlüssel selbst werden wie folgt berechnet:

```
client_write_MAC_secret := key_block[0...15]
server_write_MAC_secret := key_block[16...31]
client_write_key := key_block[32...36]
client_write_key :=
          MD5(client_write_key + ClientHello.random + ServerHello.random);
server_write_key := key_block[37...41]
server_write_key :=
          MD5(server_write_key + ServerHello.random + ClientHello.random);
```

Auch die Initialisierungsvektoren werden auf sehr einfache Weise berechnet.

```
client_write_IV := MD5(ClientHello.random + ServerHello.random)
server_write_IV := MD5(ServerHello.random + ClientHello.random)
```

8.4 Bewertung

SSL wird architekturell oberhalb von TCP/IP und unterhalb der Anwendungsschicht eingeordnet. Es stellt grundlegende Konzepte zur Sicherung von Verbindungen und eine Dienstschnittstelle für darüberliegende Anwendungen bereit. Daher lassen sich viele Anwendungen und Protokolle über SSL absichern. Zudem wird eine große Anzahl von Verschlüsselungs– und Schlüsselverteilungsalgorithmen unterstützt. SSL läßt sich daher sehr flexibel anpassen und einsetzen.

Auf Seite der Sicherheitsanforderungen kann die zweiseitige Authentisierung, die Vertraulichkeit sowie die Integrität von Nachrichten durchgesetzt werden. Maßnahmen, um die Verbindlichkeit der ausgetauschten Nachrichten zu sichern, sind nicht vorgesehen. Sollte dies erforderlich sein, so müßte die Anwendung, die auf SSL aufsetzt, die Verbindlichkeit sichern.

Da SSL primär zur Sicherung von HTTP–Verbindungen entwickelt wurde, sind Produkte, die SSL unterstützen, in diesem Bereich am weitesten verbreitet. Clients zur Sicherung von Telnet oder ftp gibt es bisher nur als freie Software und noch nicht als kommerzielle Produkte.

Die Spezifikation von SSL ist im Moment nur als Internet–Draft veröffentlicht. Ein Proposed Standard in Form eines RFC existiert nicht. Da die Verbreitung von SSL sehr groß ist und Netscape offene Standards unterstützt, kann davon ausgegangen werden, daß SSL über die IETF standardisiert werden soll.

Die kommerzielle Verfügbarkeit von Produkten, die SSL unterstützen, beschränkt sich derzeit auf Firmen, die aus den Vereinigten Staaten kommen. Aus diesem Grund greifen die strengen Exportbeschränkungen. Wird SSL in Europa eingesetzt, sind die Schlüssellängen beschränkt. Für symmetrische Verfahren werden die Schlüssel i.d.R. auf 40 Bit[1] und für asymmetrische, wie z.B. RSA, auf maximal 512 Bit beschränkt. Dadurch wird die kryptologische Sicherheit stark eingeschränkt. Selbst in der Spezifikation von SSL ([DA97] Abschnitt F.3) wird davor gewarnt, kurze öffentliche Schlüssel und 40 Bit lange symmetrische Schlüssel zu verwenden. Schlüssel mit 40 Bit sollten, wenn überhaupt, nur mit größter Vorsicht verwendet werden.

Auch die Verwendung von 512 Bit langen RSA–Schlüsseln wird für alle Anwendungen, außer denen mit sehr geringen Sicherheitsanforderungen, als unsicher betrachtet. Die Sicherheit von RSA beruht auf der Schwierigkeit der Faktorisierung großer Primzahlprodukte. Ein 512 Bit langer Schlüssel hat rund 154 Dezimalstellen. Im Jahr 1964 konnte eine 20–stellige, 1974 eine 45–stellige, 1984 eine 71–stellige und weitere 10 Jahre später bereits eine 129–stellige Zahl, die Produkt zweier Primzahlen war, faktorisiert werden [Odl95]. Dies verdeutlicht die Zunahme der Rechenleistung. Im April 1996 wurde ein 130–stelliges Primzahlprodukt faktorisiert. Interessant ist dabei, daß für die Faktorisierung der 129–stelligen Zahl 5000 *MIPS–Jahre*[2] verbraucht wurden, für die 130–stellige Zahl allerdings nur noch 500 MIPS–Jahre[3] [O.V96]. Dies veranschaulicht die enorme Verbesserung der Algorithmen und läßt erwarten, daß ein 512 Bit Schlüssel sehr bald nicht mehr sicher sein wird.

[1] Die Firmen Netscape und Microsoft dürfen Krytologiesoftware mit 128 Bit Schlüssellänge exportieren, allerdings nur an Banken (http://www.microsoft.com/germany/presseservice/Optimum.htm und Computer Zeitung vom 3.7.1997).

[2] 1 MIPS–Jahr ist die Leistung eines Rechners der eine Million Instruktionen pro Sekunde ausführt und ein Jahr läuft.

[3] Für die 130-stellige Zahl wurde ein quadratisches Sieb verwendet, für die 129-stellige Zahl der Number Field Sieve Algorithmus.

Adi Shamir, der den RSA–Algorithmus mitentwickelt hat, schrieb bereits 1995: "It is clear that the standard size of 512 bits no longer provides adequate protection, and shold be substantially increased" [Sha95].

Kapitel 9

Secure Shell (SSH)

Secure Shell (SSH)[1] wurde ursprünglich entwickelt, um über unsichere Netze ein Log on auf einem entfernten Rechner durchzuführen, dort Befehle auszuführen oder Dateien von einem Rechner auf den anderen zu übertragen [Ylo95]. Secure Shell kann die Berkeley „*r–Dienste*“ wie `rlogin`, `rsh` und `rcp` vollständig ersetzen. Es bietet starke Authentisierung, Integrität und Vertraulichkeit. Die gesamte Kommunikation zwischen zwei Rechnern wird automatisch und transparent für den Endbenutzer verschlüsselt. Bei einem Log on auf dem entfernten Rechner wird automatisch eine sichere Umlenkung der X11–Grafikausgabe durchgeführt, soweit beide Rechner das X11–Window System installiert haben.
Mittlerweile wird SSH als allgemeines Protokoll, das oberhalb von TCP liegt, spezifiziert [Ylo97a, Ylo97b, Ylo97c]. Es ist damit prinzipiell vergleichbar mit SSL.

Eine Verbindung mittels SSH wird immer vom Client (`ssh`) initiiert. Der Server (`sshd`) wartet am Port 22 auf eingehende Verbindungsaufbauwünsche. Das SSH–Protokoll läßt sich in ein *Transport Layer Protocol*, ein *Authentication Protocol* und ein *Connection Protocol* aufteilen. SSH verwendet ein eigenes Paketformat mit 32768 Bytes maximaler Länge. Die Länge soll ein Vielfaches von acht Bytes sein, daher muß die eigentliche Nachricht unter Umständen mit Fülldaten (Padding) entsprechend verlängert werden. Jede SSH–PDU enthält die in Tabelle 9.1 angegebenen Felder. Die gesendeten Pakete werden mit, bei Null beginnenden, Sequenznummern gekennzeichnet. Bei jedem gesendeten Paket wird die Sequenznummer um eins inkrementiert. Die Sequenznummer selbst wird nicht mitübertragen, sondern von beiden Seiten als interner Zähler geführt. Sie dient zur Integritäts– und zur Reihenfolgesicherung.

Nach dem Austausch des Kommunikationsschlüssels wird das gesamte SSH–Paket, mit Ausnahme des MAC–Feldes, verschlüsselt. Für den Austausch des Kommunikationsschlüssels wird RSA verwendet. Das Diffie–Hellman Protokoll soll zukünftig ebenfalls unterstützt werden. Als symmetrische Verschlüsselungsverfahren finden IDEA, DES, 3DES und Blowfish im CBC–Mode

[1] http://www.ssh.fi

Feldbezeichner	Inhalt
`length`	Länge der PDU in Byte
`padding length`	Länge der Fülldaten in Byte
`payload`	Nutzdaten
`padding`	Fülldaten
`MAC`	MAC Hash–Wert (optional)

Tabelle 9.1: Felder des SSH–Pakets

sowie ARCFOUR Verwendung. ARCFOUR ist äquivalent zum RC4 Algorithmus[2], der von RSA Data Security, Inc. urheberrechtlich (Trademark) geschützt ist. Zur Sicherung der Integrität werden SHA bzw. MD5 verwendet.

9.1 Verbindungsaufbau und Schlüsselaustausch

Der Aufbau einer SSH–Verbindung teilt sich in die Protokoll–Identifikationsphase und die Phase des Schlüsselaustausches auf (ein möglicher Abblauf ist in Abb. 9.1 dargestellt). Dabei werden alle nötigen Informationen über Verschlüsselungs– und MAC–Verfahren sowie die entsprechenden Schlüssel ausgetauscht. Es können in beide Richtungen verschiedene Verschlüsselungs– und MAC–Algorithmen verwendet werden. Die Verbindung wird vom Client mit einer `connect`–Nachricht an den Server aufgebaut. Der Server antwortet mit der von ihm verwendeten Protokollversion. Auch der Client schickt nun eine Nachricht mit seiner Protokollversion an den Server. Diese Nachrichten können Kommentare enthalten, die bspw. Auskunft über verwendete Hardware und Betriebssystem geben. Für den Fall, daß ein Partner die Protokollversion der anderen Seite nicht unterstützt, muß dieser die Verbindung schließen.

Nachdem die Nachricht mit der Protokollversion verschickt ist, wird auf das SSH–Paketformat gewechselt. Jeder der Partner kann dann in die Phase des Schlüsselaustausches eintreten, ohne auf die Nachricht mit der Protokollversion des Partners zu warten. Dazu wird die `SSH_MSG_KEXINIT` Nachricht verschickt. Mit Hilfe dieser Nachricht werden die verwendeten Algorithmen vereinbart. Die dabei übertragenen Attribute sind in Tabelle 9.2 zusammengefaßt.

Jede Seite besitzt ihre bevorzugten Algorithmen. Beim Absenden der Nachricht ist aber der Algorithmus des Partners für den Schlüsselaustausch nicht unbedingt bekannt. In diesem Fall „rät“ der Sender diesen Algorithmus, d.h. er trägt den Algorithmus als ersten in die `kex_algorithms` Liste ein, den er selbst verwenden will. Er kann dann sofort mit dem Schlüsselaustausch fortfahren. In diesem Fall wird `first_kex_packet_follows` auf true gesetzt. Haben beide Seiten die `SSH_MSG_KEXINIT` Nachricht erhalten, wissen sie, ob ihre Prognosen richtig waren. In diesem Fall sparen sie sich eine Nachricht. Sollte sich die Vorhersage als falsch herausstellen, muß ein, von der Gegenseite angegebenes Verfahren, gewählt und das `SSH_MSG_KEXINIT` Paket nochmals

[2] vgl. die Dateien arcfour.h und arcfour.c aus den SSH–Quelltexten

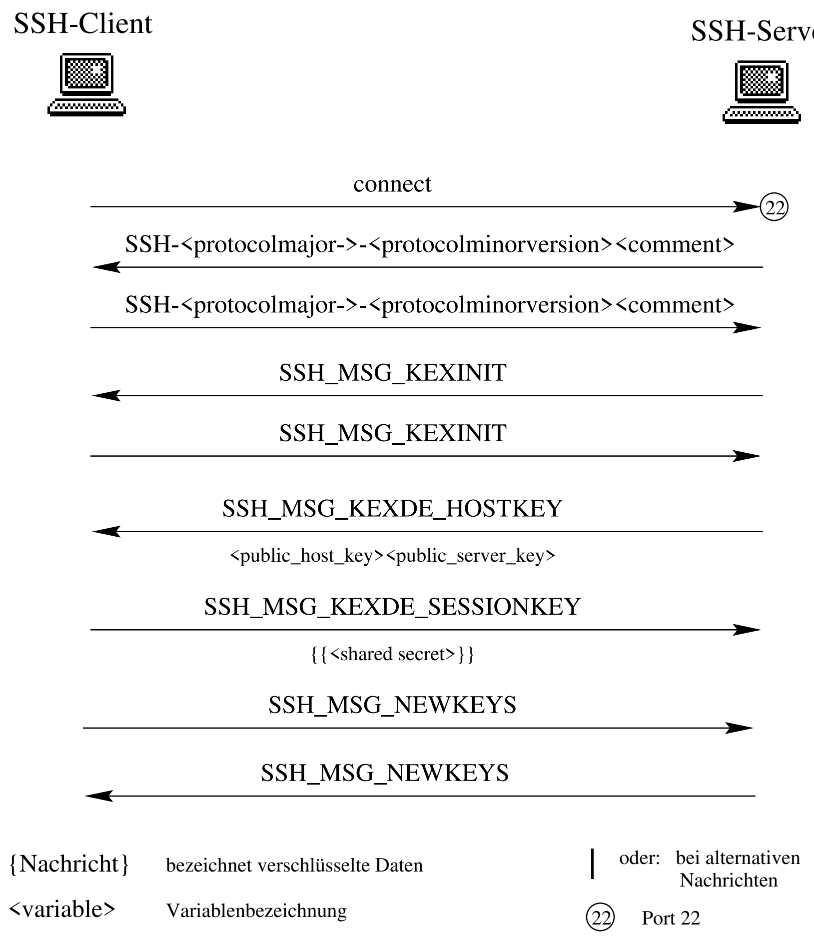

Abbildung 9.1: Verbindungsaufbau von SSH

geschickt werden. Die Gegenseite verwirft in diesem Fall die bei ihr eventuell eingegangenen Schlüsseldaten.

Zum Austausch des Kommunikationsschlüssels muß der Server in der `SSH_MSG_KEXDE_HOSTKEY` Nachricht den `public_host_key` und den `public_server_key` an den Client senden. Der host_key ist der öffentliche Schlüssel des Rechners, auf dem der SSH–Server läuft. Dieser Schlüssel hat eine längere Lebensdauer als der server_key, der öffentliche Schlüssel des SSH–Servers, der periodisch gewechselt wird. Die Spezifikation empfiehlt den server_key spätestens nach einem Gigabyte übertragener Datenmenge oder spätestens nach einer Stunde zu wechseln.

Der Client muß den host_key verifizieren, d.h. er muß prüfen, ob ihm dieser Schlüssel bekannt ist. Über den host_key kann der Client den Hostrechner authentisieren, auch ohne die IP–Adresse kennen zu müssen. Dadurch werden Angriffe, die auf der Fälschung der IP–Adresse

Attribut	Bedeutung
cookie	Eine Zufallszahl, die auf beiden Seiten berechnet wird. Die Cookies werden zur Berechnung der Kommunikationsschlüssel verwendet.
kex_algorithms	Die für den Schlüsselaustausch verwendbaren Algorithmen.
server_host_key_algorithms	Eine Liste der Algorithmen, für die der Server host_keys (öffentliche Schlüssel) besitzt, bzw. die der Client bereit ist, als Public Key Algorithmen zu akzeptieren.
encryption_algorithms_client_to_server encryption_algorithms_server_to_client	Eine Liste der unterstützten symmetrischen Verschlüsselungsverfahren nach absteigender Präferenz geordnet. Der gewählte Algorithmus ist der erste in dieser Liste.
mac_algorithms_client_to_server mac_algorithms_server_to_client	analog für MAC Algorithmen
compression_algorithms_client_to_server compression_algorithms_server_to_client	analog für Kompressionsalgorithmen
hash_algorithms	Eine Liste von Hash–Algorithmen, die später zur Berechnung der Sitzungsschlüssel verwendet werden (im Moment ist nur MD5 und SHA spezifiziert).
first_kex_packet_follows	Boolescher Wert, der anzeigt, daß diesem Paket unmittelbar das erste Paket zum Schlüsselaustausch folgt.

Tabelle 9.2: SSH-Attribute zur Vereinbarung der verwendeten Algorithmen

(IP–Spoofing) beruhen, erkannt. Der Client berechnet dann eine 256 Bit lange Zufallszahl als gemeinsames Geheimnis (*Shared Secret*). Um sicherzustellen, daß die Schlüssel nicht manipuliert wurden, berechnet er den `exchange_hash` Wert, indem er die folgenden Daten in der angegebenen Reihenfolge konkateniert und den SHA Hash–Wert über die so entstandene Zeichenkette berechnet.

1. Die Länge des `payload` Feldes der `SSH_MSG_KEXINIT` Nachricht des Client.
2. Das `payload` Feld der `SSH_MSG_KEXINIT` Nachricht des Client.
3. Die Länge des `payload` Feldes der `SSH_MSG_KEXINIT` Nachricht des Servers.
4. Das `payload` Feld der `SSH_MSG_KEXINIT` Nachricht des Servers.

5. Die Länge des `payload` Feldes der `SSH_MSG_KEXDE_HOSTKEY` Nachricht des Servers.
6. Das `payload` Feld der `SSH_MSG_KEXDE_HOSTKEY` Nachricht des Servers.
7. Die 256 Bit des gemeinsamen Geheimnisses.

Der `exchange_hash` Wert des ersten Schlüsselaustausches innerhalb einer Verbindung wird als `session-identifier` für diese Verbindung verwendet. Er wird für das spätere Authentisierungsprotokoll benötigt.

Daran anschließend erzeugt der Client eine Zeichenkette, indem er sechs Nullbytes, die ersten 10 Bytes des `exchange_hash` und das gemeinsame Geheimnis konkateniert. Die resultierende Bitkette wird zuerst mit dem server_key und dann mit dem host_key verschlüsselt und an den Server in der `SSH_MSG_KEXDE_SESSIONKEY` Nachricht übertragen.

Der Server entschlüsselt das gemeinsame Geheimnis, berechnet den `exchange_hash` und vergleicht diesen mit dem gesendeten Wert. Falls die Werte nicht übereinstimmen, unterbricht er die Verbindung. In diesem Fall ist die Integrität der Nachrichten, die durch den `exchange_hash` gesichert werden, nicht gewährleistet. Andernfalls berechnet er mit Hilfe des gemeinsamen Geheimnisses und der beiden Cookies alle benötigten Schlüssel und Initialisierungsvektoren. Die Schlüsselgenerierung ist in Abb. 9.2 zusammengefaßt. Dabei bezeichnet Secret[$i..j$] den Bitstring von Byte i bis Byte j des gemeinsamen Geheimnisses, und Hash den mit Hilfe der `SSH_MSG_KEXINIT` Nachrichten ausgehandelten Hash–Algorithmus.

IV Client to Server	=	Hash(Client-Cookie + Server-Cookie + Secret[0..15])
IV Server to Client	=	Hash(Client-Cookie + Server-Cookie + Secret[1..16])
Encryption Key Client to Server	=	Hash(Client-Cookie + Server-Cookie + Secret[5..20])
Encryption Key Server to Client	=	Hash(Client-Cookie + Server-Cookie + Secret[8..23])
MAC Key Client to Server	=	Hash(Client-Cookie + Server-Cookie + Secret[13..28])
MAC Key Server to Client	=	Hash(Client-Cookie + Server-Cookie + Secret[16..31])

Abbildung 9.2: Schlüsselgenerierung bei SSH

Zur Bestätigung, daß der Server die Schlüssel berechnen konnte, schickt er eine `SSH_MSG_NEWKEYS` Nachricht, die der Client mit der gleichen Nachricht beantworten muß. Dann wechseln beide Partner auf die vereinbarten Algorithmen und ab diesem Zeitpunkt wird der gesamte Verkehr zwischen Client und Server mit Hilfe dieser Verfahren verschlüsselt.

Im bisherigen Protokollablauf wurden keine Informationen über den Endbenutzer ausgetauscht, d.h. ein Angreifer, der den Verkehr mithört, weiß nicht, wer welchen Dienst auf dem entfernten Rechner ausführt.

Jede Seite kann, zu einem beliebigen Zeitpunkt innerhalb der Kommunikation, die Durchführung eines erneuten Schlüsselaustausches fordern. Dazu muß eine erneute `SSH_MSG_KEXINIT` Nachricht an den Partner geschickt werden. Es muß dann das oben angegebene Verfahren durchlaufen werden. Die PDU's die während der erneuten Schlüsselaustauschphase verschickt werden, werden mit den ausgehandelten Algorithmen verschlüsselt. Erst wenn beide Seiten die `SSH_MSG_NEWKEYS` Nachricht empfangen haben, wird auf die neuen Schlüssel gewechselt. Es ist auch möglich, neben den Schlüsseln die verwendeten Algorithmen und den `host_key` zu wechseln. Auch wenn der SSH–Server seinen server_key ändert, wird ein neuer Schlüsselaustausch gestartet.

9.2 Authentisierungsverfahren

Das SSH Authentication Protocol setzt auf das SSH Transport Layer Protocol auf und erhält von diesem den `session-identifier`. Der Client, der seinen Nutzer authentisieren will, sendet eine `SSH_MSG_USERAUTH_REQUEST` Nachricht an den Server. In dieser Nachricht muß er den Benutzernamen (`<user>`), den gewünschten Dienst (`<service>`), die Bezeichnung des Authentisierungsverfahrens (`<methode name>`) und, abhängig vom Verfahren, die entsprechenden Authentisierungsdaten angeben. In Tabelle 9.3 sind die vom SSH Authentication Protocol unterstützten Authentisierungsverfahren mit ihren Bezeichnern angegeben.

Bezeichner `<method name>`	Authentisierungsverfahren
none	Authentisierung mittels `.rhosts`
password	Paßwortverfahren
secureid	Authentisierung mit SecureID
skey	S/KEY Einmalpaßwortverfahren [Hal95]
opie	NRL OPIE Einmalpaßwortverfahren
publickeytest	Test, ob der öffentliche Schlüssel beim Server bekannt ist
publickey	Beweis des Besitzes eines privaten Schlüssels
hostbased	Besitz eines Schlüsselpaares und `.rhosts`
kerberos4	Kerberos Version 4
kerberos5	Kerberos Version 5
kerberos-afs	Andrew File System Kerberos

Tabelle 9.3: Authentisierungsverfahren von SSH

Ist die Authentisierung erfolgreich, sendet der Server eine `SSH_MSG_USERAUTH_SUCCESS` Nachricht zurück. Falls er die Authentisierung mit dem gewünschten Verfahren ablehnt, oder die gesendete Nachricht nicht auflösen kann, sendet er eine `SSH_MSG_USERAUTH_FAILURE` Nachricht mit einer Liste von Authentisierungsverfahren, die zu akzeptieren er noch bereit ist, zurück. Der Server kann auch mehrere Authentisierungen mit verschiedenen Verfahren verlangen, um den Nutzer

erfolgreich anmelden zu können. Der Server sendet dann trotz einer erfolgreichen Einzelauthentisierung solange … `_FAILURE` Nachrichten mit den noch zu absolvierenden Verfahren zurück, bis alle erfolgreich waren.
Der Client kann solange beliebig viele Versuche durchführen und verschiedene Verfahren verwenden, bis der Server die Verbindung schließt (z.B. nach einem Timeout) oder die Authentisierung erfolgreich war. Im folgenden wird auf die einzelnen Authentisierungsverfahren und deren Sicherheit eingegangen.

9.2.1 Authentisierung mittels `.rhosts`

Hierbei handelt es sich um das von `rlogin`, `rcp` oder `rsh` verwendete Authentisierungsverfahren [GS91]. Der Administrator auf Rechner A kann eine `/etc/host.equiv` Datei anlegen, in der alle „vertrauenswürdigen“ Rechensysteme (*Trusted Hosts*) eingetragen werden. Jeder Nutzer, der auf einem entfernten Trusted Host B dieselbe Kennung wie auf Rechner A besitzt, kann sich dann ohne die Eingabe seines Paßwortes von B aus auf A anmelden. Außerdem kann Alice (als normale Benutzerin, d.h. ohne Systemverwalterrechte) in ihrem Home–Verzeichnis eine `.rhosts` Datei mit eigenen Trusted Hosts anlegen. Damit kann sie sich von diesen Rechensystemen aus unter ihrer Kennung auf dem Rechner, auf dem die `.rhosts`–Datei gespeichert ist, anmelden. Zusätzlich kann sie in dieser Datei auch vertrauenswürdige Benutzer (*Trusted Users*) eintragen. Gibt sie bspw. Bob als vertrauenswürdigen Benutzer auf Rechner B an, so kann auch Bob sich, ohne Paßwort, von B aus, auf A in den Account von Alice einwählen.

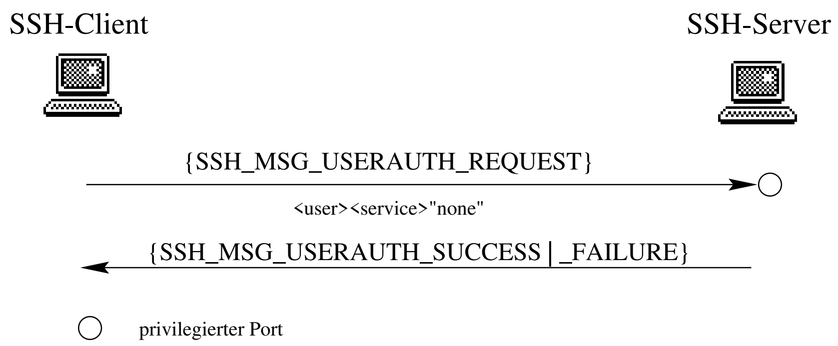

Abbildung 9.3: SSH Authentisierung mittels `.rhosts`

Will ein SSH–Client dieses Verfahren nutzen, so schickt er eine `SSH_MSG_USERAUTH_REQUEST` Nachricht mit dem `method name` “none” an den Server (vgl. Abb. 9.3). Der Server überprüft die Einträge in der `/etc/hosts.equiv`– und der `.rhosts`–Datei im Home–Verzeichnis des Benutzers. Damit die Authentisierung akzeptiert werden kann, muß der Server überprüfen, ob die Anforderung von einem *privilegierten Port* (d.h. Portnummer < 1024) angekommen ist und keine IP–Optionen wie z.B. Source Routing gesetzt wurden.

Diese Art der Authentisierung vertraut dem entfernten Rechensystem, dem Name Service und dem Netzwerk. Der Administrator des entfernten Systems könnte sich als beliebiger Benutzer ausgeben und so Zugriff auf den lokalen Rechner erhalten. Außerdem kann ein Angreifer, der den vom Server kommenden Verkehr abhören kann, einen Angriff mittels IP–Spoofing durchführen, da er dann die IP–Adressen der Kommunikationspartner des Servers kennt. Wegen der großen Sicherheitsprobleme wird empfohlen, diese Art der Authentisierung nicht zu verwenden.

9.2.2 Host–basierte Authentisierung

Das größte Problem bei der Authentisierung mittels `.rhosts` ist, daß Absender–IP–Adressen sehr leicht gefälscht werden können. Um diesen Mangel zu beseitigen, muß sich der entfernte Rechner beim SSH–Server authentisieren. Dieser Ansatz wird bei der Host–basierten Authentisierung verfolgt (vgl. Abb. 9.4).

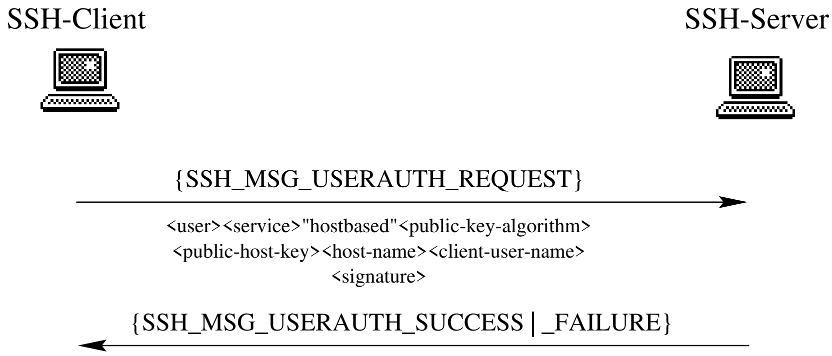

Abbildung 9.4: Host–basierte Authentisierung bei SSH

Zur Identifizierung des gewählten Verfahrens sendet der Client in diesem Fall den `method name` “hostbased” mit dem Namen des Benutzers, dem gewünschten Dienst, dem asymmetrischen Verschlüsselungsverfahren, dem öffentlichen Schlüssel des entfernten Rechners (`host_key`), dem Namen (bzw. die IP–Adresse) des Hosts, die Kennung, unter der der SSH-Client läuft und eine Signatur. Der Server authentisiert den Benutzer mittels `.rhosts` bzw. `host.equiv`. Falls diese Authentisierung erfolgreich ist, muß der Server zusätzlich den entfernten Rechner authentisieren. Dazu überprüft er, ob er den `host_key` des angegeben Rechners kennt.

Falls die beiden Schlüssel übereinstimmen, besitzt der Server noch nicht die Gewißheit, daß der entfernte Rechner auch der ist, der er vorgibt zu sein. Da die öffentlichen Schlüssel frei verteilt werden, könnte ein Angreifer die IP–Adresse fälschen und sich den öffentlichen Schlüssel, der zu dieser Adresse gehört, besorgen. Würde der Zugriff nach `.rhosts`–Authentisierung und Schlüsselvergleich gewährt, wäre keine zusätzliche Sicherheit gewonnen. Daher muß der Client mittels einer digitalen Signatur beweisen, daß er auch den zum `host_key` gehörenden privaten Schlüssel kennt. Dazu muß er den `hash`–Wert über einen Bitstring mit seinem privaten Schlüssel verschlüsseln. Der Bitstring entsteht dadurch, daß der `session-identifier` und jeweils die Länge und der Feldinhalt der folgenden Felder in der angegebenen Reihenfolge konkateniert werden: <user>, <service>, <public-key-algorithm>, <public-host-key>, <host-name>, <client-user-name>.

Der Server kann mit dem öffentlichen Schlüssel des Hosts die Signatur und dann den Hash–Wert verifizieren. Außerdem vergleicht er die Absender IP–Adresse mit dem angegebenen <host-name>. Stimmen der von ihm berechnete und der vom Client gesendete Wert und die Adressen überein, so schickt er eine SSH_MSG_USERAUTH_SUCCESS, andernfalls eine ..._FAILURE Nachricht zurück.

Durch die Berechnung des Hash–Wertes werden Chosen–Plaintext Angriffe gegen das asymmetrische Verschlüsselungsverfahren verhindert, da der session_identifier den MAC–Wert an eine bestimmte Session bindet.

Bei diesem Authentisierungsverfahren traut der Server dem Administrator des entfernten Rechners, der sich als beliebiger Benutzer maskieren könnte. Außerdem vertraut er dem host_key. Der IP–Adresse oder einem Name Service muß er nicht mehr vertrauen. Im Prinzip wäre es möglich, den Host–Namen vollständig durch den host_key zu ersetzen. Ebensowenig muß der Server auf ein sicheres Netzwerk vertrauen, da auch durch das Abhören des Netzverkehrs keine Gefahr besteht, daß ein Angreifer in den Besitz des privaten Schlüssels gelangen kann.

9.2.3 Public Key Authentisierung

Dieses Verfahren beruht auf dem Besitz eines Schlüsselpaares für ein asymmetrisches Verschlüsselungsverfahren.
Der Client schickt in seiner SSH_MSG_USERAUTH_REQUEST Nachricht den öffentlichen Schlüssel des Benutzers (vgl. Abb. 9.5). Der Server überprüft, ob er diesen Schlüssel kennt und ob diese Art der Authentisierung zulässig ist. Falls dies zutrifft, schickt er eine SSH_MSG_USERAUTH_PK_OK Nachricht an den Client.

Auch in diesem Fall muß der Client durch eine digitale Signatur beweisen, daß er den entsprechenden privaten Schlüssel kennt. Dazu muß er den hash–Wert über einen Bitstring mit seinem privaten Schlüssel unterschreiben. Der Bitstring entsteht dadurch, daß der session-identifier und jeweils die Länge und der Feldinhalt der folgenden Felder in der angegebenen Reihenfolge konkateniert werden: <user>, <service>, <public-key-algorithm> und <public-key>.

Der Server prüft die Korrektheit der Signatur und des Hash–Wertes und schickt entsprechend eine ... _SUCCESS oder ... _FAILURE Nachricht zurück.

Bei dieser Art der Authentisierung vertraut der Server weder dem entfernten Host, dem Name Service, dem Netz oder irgend etwas sonst. Die Authentisierung beruht einzig und allein auf dem Besitz eines privaten RSA Schlüssels, der zu einem entsprechenden, dem Server bekannten, öffentlichen Schlüssel gehört.

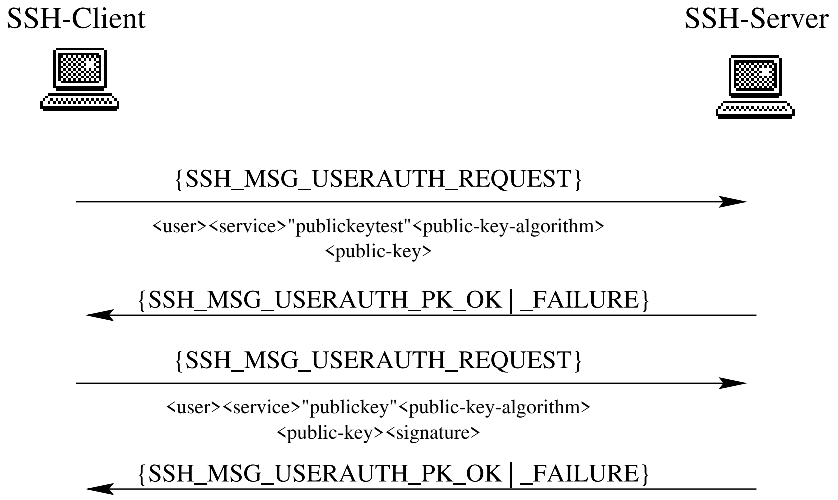

Abbildung 9.5: Public Key Authentisierung bei SSH

9.2.4 Authentisierung mittels Paßwort

Neben den bisher aufgeführten Verfahren ist bei SSH auch die Authentisierung über Benutzerkennung (Name) und Paßwort sowie über diverse One-Time-Password Systeme oder Kerberos möglich. In Abb. 9.6 ist das gewöhnliche Paßwortverfahren dargestellt.

Abbildung 9.6: SSH Authentisierung mit Paßwort

Der Client schickt dazu das Klartextpaßwort innerhalb der `SSH_MSG_USERAUTH_REQUEST` Nachricht an den Server. Da aber die gesamte PDU mit dem Sitzungsschlüssel verschlüsselt wird, kann ein Angreifer das Paßwort nicht abhören. Der Server verifiziert das Paßwort, indem er Dienste des Betriebssystems in Anspruch nimmt.

Bei One-Time-Password Systemen oder bei Kerberos verlangt der Betriebssystemdienst i.d.R. nach weiteren Eingaben durch den Benutzer. Diese Anforderung gibt der Server in der `SSH_MSG_USERAUTH_OTP_PROMPT` Nachricht an den Client weiter, der die entsprechende Zeichenkette dem Benutzer anzeigt. Die Zeichenkette, die vom Benutzer daraufhin eingegeben wird, schickt der Client in der SSH_MSG_USERAUTH_OTP_RESPONSE Nachricht zurück an den Server. Abhängig von der Ausgabe des entsprechenden Dienstes schickt der Server eine ..._SUCCESS oder ..._FAILURE Nachricht zurück.

Anzumerken ist, daß der Endbenutzer nie mit einem `login` Programm interagiert, sondern daß das Paßwort vom SSH–Client abgefragt und vom SSH–Server verifiziert wird. Dadurch wird der Endbenutzer vor Trojanischen Pferden innerhalb von `login` Diensten geschützt.

Auch dieses Verfahren vertraut weder dem entfernten Host, dem Netzwerk und dem Name Server sondern nur dem Besitz eines Paßwortes.

9.3 Connection Protocol

Das *SSH Connection Protocol* setzt auf das Authentication Protocol auf, d.h. nach erfolgreicher Authentisierung bietet es interaktive Sitzungen, die Ausführung von Kommandos auf entfernten Rechnern sowie das Forwarding von TCP/IP– und X11–Verbindungen.

Alle diese Dienste werden über sichere Kanäle (Channels) zwischen Client und Server abgewickelt. Auf diesen Kanälen wird, mit Hilfe der Fenstertechnik, Flußkontrolle durchgeführt. Alle Daten, die über einen sicheren Kanal übertragen werden, sind verschlüsselt. Innerhalb einer SSH–Sitzung können mehrere Kanäle geöffnet werden. Ein Kanal wird durch eine Nummer repräsentiert, die auf beiden Seiten unterschiedlich sein kann. Falls ein Partner einen Dienst anfordern will, öffnet er lokal einen Kanal, indem er eine Kanalnummer vergibt. In der SSH_MSG_CHANNEL_CREATE_SESSION Nachricht überträgt er diese Kanalnummer an die Gegenseite, um dann einen bestimmten Dienst anzufordern. Kann der Kanal geöffnet werden, antwortet die Gegenseite mit der eigenen Kanalnummer in der SSH_MSG_CHANNEL_OPEN_CONFIRMATION Nachricht. Andernfalls wird ein SSH_MSG_CHANNEL_OPEN_FAILURE zurückgeschickt.

Nachdem der Kanal geöffnet wurde, können vom Client die, in Tabelle 9.4 zusammengefaßten Dienste, beim Server angefordert werden. Falls die Dienstanforderung erfüllt werden kann, wird ein SSH_MSG_CHANNEL_SUCCESS, andernfalls ein ..._FAILURE zurückgeschickt. Zur Übertragung von Daten zwischen dem Client und der Anwendung wird der Nachrichtentyp SSH_MSG_STREAM_DATA verwendet.

Für den Fall, daß eine Seite keine weiteren Daten mehr über einen bestimmten Kanal übertragen will, sendet sie eine SSH_MSG_CHANNEL_EOF Nachricht, die nicht beantwortet werden muß. Der Kanal bleibt weiterhin geöffnet, damit die Gegenseite noch Daten übertragen kann. Soll der Kanal geschlossen werden, so muß eine SSH_MSG_CHANNEL_CLOSE Nachricht verschickt werden, die mit der gleichen Nachricht beantwortet werden muß. Der Kanal wird erst geschlossen, wenn beide Seiten diese Nachricht empfangen haben.

Nachricht (mit dem Präfix SSH_MSG_)	Operation
SESSION_REQUEST_PTY	Der Server alloziert ein Pseudo Terminal für den Client.
SESSION_X11_REQUEST_FORWARDING	Der SSH–Server muß die X11–Verbindungen auf dem entfernten Rechner über einen sicheren Kanal auf die lokale Maschine umleiten.
REQUEST_TCPIP_PORT_FORWARDING	Der Server muß einen angegebenen TCP/IP–Port über den sicheren Kanal auf den lokalen Rechner umlenken.
SESSION_EXEC_SHELL	Der Server muß eine Shell für den Benutzer starten und in die interaktive Session wechseln.
SESSION_EXEC_COMMAND	Der Server muß das angegebene Kommando ausführen und in die interaktive Session wechseln.
SESSION_ENVIRONMENT_VARIABLE	Der Client setzt Umgebungsvariablen auf dem Server–Rechner
SESSION_EXEC_PREDEFINED	Der Server bringt ein vordefiniertes Subsystem zur Ausführung (z.B. zur Übertragung von Dateien)

Tabelle 9.4: Dienstanforderungen innerhalb des SSH Connection Protocols

9.3.1 Umlenkung der X11–Grafikausgabe

Das *X11–Window System* [SG90] ist als Client/Server Architektur implementiert. Auf dem Rechner, auf dem eine Grafikausgabe bzw. die Eingabe erfolgen soll, läuft ein X–Server, der die Grafikausgabe auf ein sogenanntes X–Display durchführt. Das Programm, das eine Ausgabe tätigen will, ruft als X–Client auf dem Server entsprechende Dienste auf. Die (Terminal–) Eingabe, die auf dem X11–Server erfolgt, wird an den entfernten Rechner und an das entsprechende Programm weitergeleitet. Da das X–Grafiksystem konzeptionell ein netzwerkbasiertes System ist, kann grundsätzlich jeder Benutzer von einem Rechner im Netz, auf ein bestimmtes X–Display schreiben, oder über den X–Server Daten aus einem X–Display auslesen. Dies stellt ein großes Sicherheitsproblem dar. Da der X–Server auch für die Verarbeitung der Tastatureingabe verantwortlich ist, kann jeder Benutzer im Netzwerk, der Zugriff auf den X–Server hat, die Tastatureingaben und die Grafikausgaben mitlesen. Aus diesem Grund gibt es im X–Window System zwei Clients, `xhost` und das „*MIT-MAGIC-COOKIE*“ Verfahren, die den Zugriff auf den X–Server beschränken. Die `xhost` Methode verfolgt einen sehr naiven Ansatz. Dabei ist es nur möglich, bestimmten Rechnern explizit Zugriff auf den X–Server zu gewähren oder zu untersagen. Damit können aber immer noch alle Benutzer auf einem Rechner, dem der Zugriff auf den X–Server gestattet wurde, Daten vom X–Server auslesen.

Das zweite Verfahren arbeitet paßwortbasiert. Dabei vereinbaren Benutzer und X–Server beim Start des Servers ein spezielles Paßwort, das als MAGIC-COOKIE bezeichnet wird. Versucht nun ein Benutzer ohne korrektes COOKIE auf den Server zuzugreifen, wird die Verbindung abgelehnt.

Bei SSH läuft der X11–Server auf dem lokalen Rechner A, auf dem sich der SSH–Client befindet. Für die Umlenkung der Grafikausgabe baut SSH einen sicheren Kanal zwischen entferntem X–Client auf dem Rechner B, auf dem sich der SSH–Server befindet, und dem lokalen X–Server auf (vgl. Abb. 9.7).

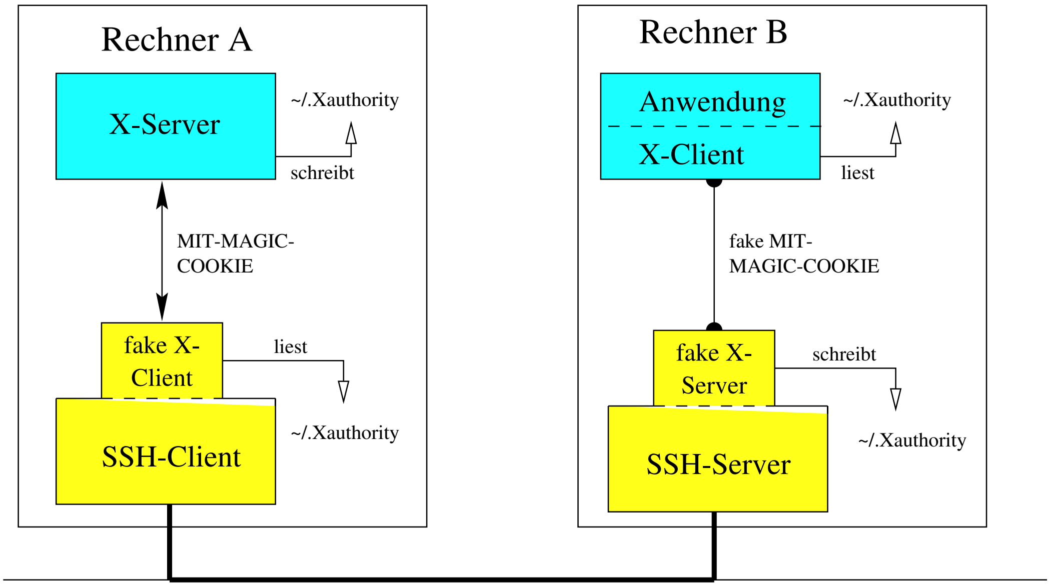

Abbildung 9.7: X11–Grafikumlenkung bei SSH

Nachdem der SSH–Server die `SSH_CMSG_X11_REQUEST_FORWARDING` Nachricht empfangen hat, wird ein Socket geöffnet und als X11–Display dem X11–Client bekanntgegeben, d.h. der SSH–Server tritt als X11–Server auf. Der X11–Client schickt seine Dienstanforderungen an diesen Proxy X–Server (*fake X–Server*) beim SSH–Server. Die Daten werden vom SSH–Server verschlüsselt und an den SSH–Client geschickt. Nach der Entschlüsselung werden sie vom Client an den „echten" X–Server weitergegeben.

Das SSH–Protokoll verwendet das sichere MAGIC–COOKIE basierte Zugriffsverfahren auf den X–Server. D.h. zwischen dem SSH–Client und dem X–Server wird auf dem lokalen Rechner ein MAGIC–COOKIE vereinbart. Mit der ...`X11_REQUEST_FORWARDING` Nachricht schickt der SSH–Client aber nicht dieses COOKIE, sondern ein „gefälschtes" (fake MAGIC–COOKIE) an den SSH–Server. Erhält der SSH–Client Daten für den X11–Server, überprüft er, ob das entsprechende fake MAGIC–COOKIE gesendet wurde, ersetzt dieses durch das richtige und reicht die Daten an den X–Server weiter.

Da das X11–Protokoll verlangt, daß das MAGIC–COOKIE in der `.Xauthority`–Datei im Home–Verzeichnis des jeweiligen Benutzers gespeichert wird, verhindert dieses Vorgehen, daß echte Cookies auf fremden, entfernten Rechnern gespeichert werden. Dies wäre insbesondere dann sicherheitskritisch, wenn diese Cookies über längere Zeiträume verwendet werden.

9.3.2 TCP/IP–Port Umlenkung

Mit Hilfe der `SSH_MSG_REQUEST_TCPIP_PORT_FORWARDING` Nachricht kann ein Kommunikationspartner die Umlenkung eines TCP/IP–Ports auf der Gegenseite anfordern. Dazu muß er in der Nachricht die Adresse und den Port angeben, der auf den sicheren SSH–Kanal umgelenkt werden soll. Wenn ein Verbindungsaufbauwunsch auf dem entfernten Rechner an einem Port, der umgelenkt wurde, eingeht, wird die `SSH_MSG_TCPIP_REMOTE_PORT_OPEN` Nachricht vom entfernten an den lokalen Rechner geschickt. Diese Nachricht enthält die Kanalnummer auf dem entfernten Rechner, die initiale Fenstergröße, die Portnummer des Ports, an dem der Verbindungsaufbauwunsch einging, die IP–Adresse und den Port des Anfordernden und die Zeichenkette, die an den Port geschickt wurde.

Auch lokal kann ein Port auf einen SSH–Kanal umgelenkt werden. Wird ein Verbindungsaufbauwunsch an einen solchen Port geschickt, so sendet der lokale Rechner die `SSH_MSG_TCPIP_PORT_OPEN` Nachricht an den entfernten Rechner. Diese Nachricht enthält u.a. den Host und den Port, mit dem der entfernte Rechner eine Verbindung aufbauen soll, sowie die IP–Adresse und den Port des Anfordernden.

Mit Hilfe dieses Verfahrens ist es sehr einfach, unsichere TCP/IP–basierte Dienste über einen sicheren SSH–Kanal umzuleiten. Dazu erzeugt der SSH–Client bspw. einen lokalen Proxy für einen entfernten Dienst wie HTTP, pop oder auch für entfernte Datenbankabfragen. Der lokale Proxy–Server wartet an einem Socket auf Dienstanforderungen, die dann vom SSH–Client auf den entfernten Rechner umgeleitet werden. Der SSH–Server leitet die eingegangenen Daten nach der Entschlüsselung an den spezifizierten Dienst bzw. Port, weiter. Auch die Ausgabedaten des Dienstes können wieder über den SSH–Kanal auf den lokalen Proxy und von dort zum Anwender umgeleitet werden. Um dieses Verfahren anwenden zu können, muß jedoch die Client–Software des entsprechenden Dienstes so konfiguriert werden, daß sie eine Verbindung zum lokalen Proxy, anstatt zum entfernten Rechner aufbaut.

9.4 Schlüsselverteilung

Bei SSH existiert im Moment kein umfassendes Konzept zur Schlüsselverteilung. Ein Client, der zum ersten Mal eine Verbindung zu einem entfernten SSH–Server aufbaut, erhält von diesem den host_key des Server–Rechners. Dieser host_key ist bekannt und wird akzeptiert, falls er in der Datei `/etc/ssh_known_hosts` gespeichert ist. Falls der host_key nicht bekannt ist, wird er, beim ersten Verbindungsaufbau, in die Datei `~/.ssh/known_hosts` des Benutzers aufgenommen.

Ein Angreifer, der in diese Kommunikation eindringen kann, könnte seine eigenen Schlüssel anstelle des host_keys beim Client registrieren und damit einen Man-in-the-Middle Angriff bzw. eine Maskerade durchführen. Da die Möglichkeit zur Durchführung eines solchen Angriffes nur einmal bei der ersten Verbindung zwischen zwei Rechnern besteht, darf diese Schwachstelle nicht überbewertet werden. Für Systeme, die besonders schützenswerte Ressourcen darstellen, besteht die Möglichkeit, deren host_keys bereits bei der SSH–Installation auf allen Rechnern in die `/etc/ssh_known_hosts`–Datei einzutragen.

Eine Möglichkeit zur Verteilung der Schlüssel wäre, einen ausgezeichneten Rechner als Key–Server zu bestimmen. Der host_key dieses Key–Servers muß bei der Installation von SSH allen anderen Rechnern bekannt gemacht werden. Sobald ein Rechner einen eigenen host_key erzeugt hat, kann er mit Hilfe von SSH eine Verbindung zum Key–Server aufbauen, seinen Schlüssel dort registrieren und bereits registrierte Schlüssel anderer Rechner in seine `/etc/ssh_known_host`–Datei integrieren.

Auf den ersten Blick scheint dieser Ansatz sicherer zu sein, als den Schlüssel eines fremden Rechners jeweils bei der ersten Verbindung mit diesem System zu übernehmen. Dies ist aber nicht der Fall. Durch den zentralen Key–Server entsteht auch ein zentraler Angriffspunkt im Unternehmensnetz.

Nachdem Alice ihren host_key erzeugt hat, baut sie eine Verbindung zum Key–Server auf, der Alice zu diesem Zeitpunkt weder über ihren host_key noch über ein Public Key Verfahren authentisieren kann. Eine sichere Authentisierung könnte lediglich durch die Verwendung eines Paßwortes realisiert werden. Aus praktischen Erwägungen kann dies jedoch nicht verwirklicht werden, da sonst für jedes System im Netz eine Kennung und ein Paßwort auf dem zentralen Key–Server eingerichtet bzw. sicher verteilt werden müßte. Der zentrale Key–Server kann also zu diesem Zeitpunkt den öffentlichen Schlüssel von Alice nicht sicher mit ihrer Identität verknüpfen. Ein Angreifer Mallet, der sich für Alice ausgibt, könnte deshalb vor Alice seinen eigenen Schlüssel unter der Adresse von Alice registrieren. Da dieser Schlüssel vom Key–Server an alle anderen Rechner im Netz verteilt wird, kann sich Mallet durch einen einzigen Angriff bei allen Rechnern als Alice maskieren. Falls kein zentraler Key–Server realisiert wird, kann Mallet jeweils nur eine Verbindung angreifen, d.h. wenn Bob das erste Mal eine Verbindung mit Alice aufbaut, könnte er sich dann, allerdings nur gegenüber Bob, als Alice ausgeben und nicht gegenüber allen Rechnern im Unternehmensnetz.

9.5 Bewertung und Vergleich mit SSL

SSH wurde ursprünglich entwickelt, um die Berkeley r–Dienste zu ersetzen, SSL hingegen, um die Kommunikation im WWW zu sichern. Mittlerweile werden jedoch beide als allgemeine Protokolle oberhalb von TCP/IP spezifiziert und erfüllen ähnliche Aufgaben. Insofern sind sie als Protokoll vergleichbar, obwohl die Unterstützung von Diensten unterschiedlich realisiert wird bzw. die Implementierung von Clients sehr verschieden ist. Bei SSL gibt es bspw. noch keinen

Client bzw. Server, um die r–Dienste zu ersetzen und bei SSH wird die Sicherung einer HTTP–Verbindung völlig anders realisiert als bei SSL. Im folgenden wird SSH bewertet und, soweit dies möglich ist, mit SSL verglichen.

Da SSH in der Lage ist, die r–Dienste für den Benutzer vollkommen transparent zu unterstützen, kann man die r–Dienste auch völlig durch SSH ersetzen. Für den Fall, daß eine Verbindung zu einem entfernten Rechner aufgebaut werden soll, der kein SSH unterstützt, existiert ein Fallback–Modus und der SSH–Client tritt in diesem Fall als Client für den entsprechenden r–Dienst auf. Werden die r–Dienste durch SSH ersetzt, verliert man keine Funktionalität, sondern gewinnt zusätzliche Sicherheit. Außerdem muß sich der Benutzer nicht mehr „per Hand“ um die sichere Umlenkung der X11–Verbindung kümmern, da dies von SSH automatisch durchgeführt wird. Ein Benutzer, der sich von einem X–fähigen Rechner auf einem entfernten System mit X11-Window-System anmeldet, kann sofort alle entfernten Programme benutzen, und die Daten werden gesichert übertragen. Für Microsoft Windows existiert eine kommerzielle Implementierung eines SSH–Client, der Telnet oder eine Teminalemulation ersetzen kann (vgl. Abschn. 13.2).

Secure Shell wurde in Europa entwickelt und wird auch von Europa aus vertrieben. Daher greifen die amerikanischen Exportgesetze und die damit verbundene Reduzierung der Schlüssellänge nicht. Bei den asymmetrischen Verfahren kann der Endbenutzer bzw. der Administrator seine Schlüssellänge bzw. die Schlüssellänge für server_ und host_key frei wählen. Defaultwert im Falle von RSA ist ein host_ und client_key von 1024 Bit und ein server_key von 768 Bit. Auch bei den symmetrischen Verfahren werden volle Schlüssellängen verwendet (d.h. bei DES 56 Bit, bei 3DES 168 Bit, bei IDEA 128 Bit sowie eine variable Schlüssellänge bei ARCFOUR und Blowfish). Die Anzahl und die Qualität der bei SSH implementierten Algorithmen ist denen von SSL gleichwertig.

SSH besitzt durch die Trennung von host_ und server_key ein sehr flexibles und mächtiges Konzept zum Schlüssel– und Algorithmenwechsel innerhalb einer SSH–Sitzung bzw. während der Laufzeit des Servers. Der Server–Administrator kann eine Lebensdauer für den server_key vorgeben (Defaultwert: eine Stunde). Nach Ablauf dieses Zeitraums wechselt der SSH–Server sein Schlüsselpaar und es wird ein neuer Schlüsselaustausch angestoßen. Durch das zweistufige Verfahren und da der Server regelmäßig seinen Schlüssel wechselt, kann im Fall der Kompromittierung des server_keys und des host_keys nur maximal der Verkehr bis zum nächsten Schlüsselwechsel gebrochen werden. Bei SSL gibt es ein derartiges Konzept nicht. Es können zwar die Algorithmen und der Sitzungsschlüssel gewechselt werden; für den Schlüsselaustausch werden jedoch Server–Zertifikate verwendet, die eine relativ lange Lebensdauer (i.d.R mehrere Monate oder Jahre) haben. Das Kompromittieren eines privaten Schlüssel bzw. eines Zertifikates bei SSL kann u.U. dazu führen, daß ein Angreifer sehr lange Zeit den Verkehr brechen kann. Bei SSH hingegen müssen zwei asymmetrische Schlüssel gebrochen werden, um den Verkehr, für relativ kurze Zeit, mitlesen zu können.

Auch bei der Authentisierung verwendet SSL als alleiniges Konzept das der Zertifikate. Bei SSH gibt es eine größere Zahl von unterstützten Authentisierungsverfahren. Allerdings existiert bei SSH bisher kein umfassendes Konzept zur Verteilung öffentlicher Schlüssel. Ein Client, der zum ersten Mal eine Verbindung zu einem entfernten Host aufbaut, erhält von diesem den host_key, den er natürlich noch nicht kennt. D.h beim erstmaligen Austausch des host_key besitzt das Protokoll eine Schwäche, die ein Angreifer ausnützen könnte, um seine Schlüssel als bekannt einzuführen. Im Umfeld eines Unternehmens wäre es jedoch auch denkbar, daß „wichtige“ host_keys bereits bei der Installation der SSH–Software mit auf den Rechnern gespeichert werden. Zukünftig soll SSH auch X.509v3–Zertifikate für die host_keys unterstützen, und es könnte dann eine unternehmenseigene Zertifizierungshierarchie für die Verteilung der Schlüssel genutzt werden.

Soll SSH eingesetzt werden, um den HTTP–Verkehr zu sichern, so müssen auf Client– und Server–Seite Proxies eingerichtet werden. Die HTTP–Pakete werden über den sicheren Kanal zwischen SSH–Client und –Server übertragen. Dazu müssen die Ports für den HTTP–Verkehr auf den SSH–Port (Port 22) umgelenkt werden. Dieses Vorgehen ist sehr sicher und auch problemlos durchführbar, solange zwischen SSH–Client und –Server kein Firewall liegt. Andernfalls muß auf dem Firewall der Verkehr, der über Port 22 abgewickelt wird, freigegeben werden. Dies ist insofern problematisch, als die Daten, die über diesen Port übertragen werden, naturgemäß verschlüsselt sind, d.h. auf dem Firewall kann nicht in Abhängigkeit von verwendeten höheren Protokollen gefiltert werden. Da auch das entfernte Log on über eine Port 22 Verbindung abgewickelt wird, kann auf dem Firewall nicht entschieden oder gefiltert werden, ob es sich um HTTP–Verbindungen oder aber um eine verschlüsselte Sitzungsverbindung, vergleichbar mit rlogin oder Telnet, handelt. Da aber i.d.R. Telnet–Verkehr von außen über einen Firewall nicht erlaubt wird, darf auch der Verkehr über Port 22 nicht freigegeben werden.

Um SSH zum Schutz des HTTP–Verkehrs auch über einen Firewall nutzen zu können, müßten auch auf dem Firewall Proxies installiert werden. Der HTTP–Client müßte also eine Verbindung zum lokalen Proxy aufbauen. Der Proxy wiederum stellt eine SSH–Verbindung zum Firewall her, der die Nachricht entschlüsselt, filtert und ggf. mit Hilfe von SSH eine erneute gesicherte Verbindung zum Zielrechner aufbaut. Ob eine Installation von SSH und der entsprechenden Proxies auf dem Firewall möglich ist, hängt von der Sicherheitspolitik für den Firewall ab und kann hier nicht abschließend beurteilt werden.

Kapitel 10

Sichere Kommunikation im World Wide Web (WWW)

Die BMW AG betreibt neben ihrem öffentlichen WWW Server[1] auch unternehmensinterne und sogar abteilungsbezogene WWW–Server als Informations– und Kommunikationssystem für die Mitarbeiter. Neben der reinen Informationsfunktion, nehmen WWW–gestützte Anwendungen immer mehr zu. Im Zusammenhang mit der Händlervernetzung wird bspw. ein, als *Online Ordering* bezeichnetes Projekt durchgeführt. Damit werden Händler ihre Fahrzeuge direkt, über WWW, bei der BMW AG bestellen können. Auch die Administration von anderen Servern (z.B. Zertifizierungsserver, Datenbankserver, u.ä.) wird häufig durch WWW–gestützte Anwendungen realisiert.

Dies verdeutlicht, daß immer mehr unternehmenskritische und sensible Daten mittels *HTTP* (*Hypertext Transfer Protocol* [FGM+97]) übertragen werden. Es ist klar, daß Möglichkeiten zur Sicherung dieses Datenverkehrs untersucht werden müssen.

10.1 Secure Hypertext Transfer Protocol (S-HTTP)

S-HTTP [RS97b] ist eine Erweiterung des HTTP Protokolls, um die Integrität, Vertraulichkeit, Authentisierung und die Verbindlichkeit von HTTP–Nachrichten zu sichern. Dies wird durch die drei Betriebsmodi Signature, Authentication und Encryption realisiert. Eine S-HTTP Nachricht kann also signiert, verschlüsselt oder authentisiert sein sowie eine beliebige Kombination dieser Eigenschaften realisieren.

Die S-HTTP Nachrichten sind syntaktisch mit den HTTP–Nachrichten vergleichbar, d.h. sie bestehen aus einem Header mit `<content-typ>:<Wert>` Paaren und ei-

[1] http://www.bmw.de oder http://www.bmw.com

nem Body. Zur Differenzierung von HTTP–Nachrichten wird der Protocol Designator `Secure-HTTP/<majorversion>.<minorversion>` definiert. Mit Hilfe der `<content-typ>:<Wert>` Felder werden Informationen über verwendete Algorithmen, Schlüssel u.ä. zwischen HTTP–Client und –Server ausgetauscht.

Die Integrität einer HTTP–Nachricht wird durch die Berechnung eines MAC gesichert. Dazu wird die HTTP–Nachricht, der Zeitpunkt der Berechnung und ein geheimer Schlüssel konkateniert und ein Hash–Wert über diesen Bitstring berechnet. Da die Zeit mit in die MAC–Berechnung eingeht, können Replay Angriffe verhindert werden. Im `MAC-Info` Feld des S-HTTP–Headers werden alle nötigen Informationen, wie verwendete Algorithmen, Zeitpunkt der Berechnung, eine Schlüssel–ID und der Hash–Wert selbst, an den Empfänger übertragen, der mit Hilfe dieser Informationen den MAC verifizieren kann. Da der Sender den geheimen Schlüssel kennt, und ihn bei der Berechnung des MAC verwendet, authentisiert er sich beim Empfänger. Als MAC–Algorithmen finden *MD2* (*Message Digest Nr. 2* [Kal92]), MD5 und SHA Verwendung.

Die Vertraulichkeit wird durch die, in Tabelle 10.1 angegebenen, symmetrischen Verschlüsselungsalgorithmen realisiert.

Kurzbezeichnung	Algorithmus
DES–CBC	DES im Cipher Block Chaining Mode
DES–EDE–CBC	3DES mit zwei Schlüsseln im Encrypt-Decrypt-Encrypt-CBC–Mode
DES–EDE3–CBC	3DES mit drei Schlüsseln im Encrypt-Decrypt-Encrypt-CBC–Mode
DESX–CBC	DESX Algorithmus, von der Firma RSA, im CBC–Mode [Rog96]
IDEA–CBC	IDEA im CBC–Mode
RC2–CBC	RC2 Algorithmus von RSA im CBC–Mode
CDMF–CBC	CDMF Algorithmus von IBM im CBC–Mode

Tabelle 10.1: Verschlüsselungsalgorithmen bei S-HTTP

Die HTTP–Nachricht wird verschlüsselt und in einem S-HTTP–Body übertragen. Bestimmte Angaben aus dem Original HTTP–Header werden von HTTP–Agenten und Transitsystemen benötigt (z.B. Absenderadresse u. Port) und daher in den S-HTTP–Header übernommen. Unterstützt werden auch die kryptographischen Nachrichtentypen PKCS #7 (vgl. Abschnitt 11.3) und MOSS [CFGM95].

Die Verbindlichkeit wird durch die digitale Signatur der HTTP–Nachricht bzw. des MAC der Nachricht, mittels RSA, realisiert. Dazu müssen Client und Server ihre öffentlichen Schlüssel austauschen. Durch die digitale Unterschrift authentisiert sich der Unterzeichner bei seinem Partner, da nur er seinen privaten Schlüssel kennt und folglich nur er in der Lage ist, die HTTP–Nachricht zu signieren.

In S-HTTP sind auch Content–Typen zum Austausch von öffentlichen Schlüsseln mittels Zertifikaten definiert. Dazu wird auch das HTML–Protokoll [BLC95] zu SHTML [RS97a] erweitert, damit Zertifikate auf HTML–Seiten abgelegt und abgerufen werden können. Der Austausch der symmetrischen Schlüssel erfolgt entweder innerhalb der Nachricht, oder sie werden über sichere Kanäle Out-of-Band ausgetauscht. Im ersten Fall werden die Schlüssel mit einem asymmetrischen Verschlüsselungsverfahren gesichert.

10.2 Hypertext Transfer Protocol over SSL (HTTPS)

Eine weitere Möglichkeit zur Sicherung des HTTP–Verkehrs ist, sich auf bestehende Protokolle abzustützen. Diesen Ansatz verfolgt *Hypertext Transfer Protocol over SSL (HTTPS)*, das die Funktionalität von SSL (vgl. Abschn. 8) nutzt.

Damit kann die Integrität und die Vertraulichkeit von HTTP–Nachrichten und die zweiseitige Authentisierung der Kommunikationspartner realisiert werden. Auf Seiten des HTTP–Servers (WWW–Server) und des HTTP–Client (WWW–Browser) muß dazu der SSL Protocol–Stack implementiert sein. Für den HTTPS–Verkehr wird anstelle des Standard WWW–Ports (80) der Port 443 verwendet. Außerdem wird ein eigenes URL–Format [BLMM94] definiert, das anstatt mit `http://` mit `https://` beginnt. Alle Daten, die mit diesem URL–Format abgefragt werden, oder über den Port 443 gehen, werden verschlüsselt. Ein WWW–Server mit SSL kann damit neben normalem HTTP–Verkehr auch HTTPS–Verkehr abwickeln.

Da SSL eine zertifikatsbasierte Authentisierung realisiert und der Kommunikationsschlüssel mit Hilfe eines asymmetrischen Verfahrens verschlüsselt wird, benötigen sowohl Server als auch Client gültige Zertifikate. Beide Seiten können die Authentisierung des anderen Kommunikationspartners verlangen oder darauf verzichten. Dazu werden Funktionen des SSL Handshake Protocols (vgl. Abschn. 8.2) verwendet.

10.3 Bewertung

Sowohl HTTPS als auch S-HTTP sind in der Lage, Vertraulichkeit, Integrität und Authentisierung durchzusetzen. S-HTTP bietet zusätzlich noch die Möglichkeit, HTTP–Nachrichten digital zu signieren. Damit läßt sich Verbindlichkeit realisieren. Bei HTTPS müßte dies in der jeweiligen Anwendung realisiert werden.

Beide Protokolle verwenden zur Verteilung der Kommunikationsschlüssel asymmetrische Verfahren. Die öffentlichen Schlüssel werden durch Zertifikate verteilt und an die Identität des Benutzers gebunden. S-HTTP führt auch die Schlüsselverteilung Out-of-Band in der Spezifikation auf. Falls die Kommunikationsschlüssel auf diese Weise verteilt werden, kann die S-HTTP–Verbindung auch ohne Client–Zertifikat aufgebaut werden.

Betrachtet man die architekturelle Dimension, unterscheiden sich die beiden Protokolle erheblich. S-HTTP ist auf der Anwendungsschicht realisiert. Konzeptionell und vom Design steht es PEM, MIME und S/MIME (vgl. Abschn. 11) nahe, die zur Sicherung bzw. Erweiterung des E–Mail–Verkehrs entwickelt wurden. S-HTTP erweitert die HTTP–Header Felder um zusätzliche Content–Typen, um notwendige Informationen zu übertragen. Verschlüsselte Daten werden im S-HTTP–Body übertragen. Im Gegensatz dazu stützt sich HTTPS voll auf SSL ab, das architekturell oberhalb von TCP und unterhalb der Anwendungsschicht einzuordnen ist. Daher sind keine Veränderungen am HTTP–Protokoll notwendig. Die entsprechenden Clients bzw. Server erstellen die HTTP–Nachrichten, bauen mit Hilfe des SSL Handshake Protocols eine Verbindung auf, und das SSL Record Layer Protocol übernimmt die Verschlüsselung der HTTP–Nachrichten.

Auf Seite der WWW–Clients kann, da S-HTTP auf Anwendungsebene realisiert ist, die Anwendungslogik durch ein sog. *Plug-in* in den WWW–Browser integriert werden. Bei HTTPS muß ein SSL–Protokoll–Stack implementiert sein, und der WWW–Browser muß in der Lage sein, die SSL–Dienste aufzurufen. In der Regel wird dies „fest“ im Browser und nicht durch, bei Bedarf nachzuladende, Plug-ins realisiert.

Die Protokolle müssen aber auch von den entsprechenden HTTP–Servern unterstützt werden. SSL wird von sehr vielen Servern unterstützt, wohingegen die Verbreitung von S-HTTP sehr gering ist.

Kapitel 11

Sichere Elektronische Post

Electronic–Mail (E-Mail) wird in zunehmendem Maße als relativ schnelles, billiges und asynchrones Kommunikationsmittel in Unternehmen eingesetzt. Zur Übertragung von Nachrichten im Internet bzw. in TCP/IP–Netzen wird das *Simple Mail Transfer Protocol (SMTP)* [Pos82, Cro82] verwendet und um *Multipurpose Internet Mail Extensions (MIME)* [BF93, Moo93, Bor93] erweitert. Als Architekturmodell kann dabei das *Message Handling System Model* [CCI88a] verwendet werden (vgl. Abb. 11.1). Danach besteht das *Mail System* aus einer Menge von verbundenen *Message Transfer Agents (MTA)*, die Nachrichten untereinander austauschen. Ein Benutzer, der eine E-Mail versenden möchte, verwendet dazu ein E-Mail Programm, das im Modell als *User Agent (UA)* bezeichnet wird. Der UA übergibt die Nachricht an den MTA. Der Message Transfer Agent vermittelt die Nachricht an einen weiteren MTA, der die Nachricht auch weiterleitet, solange, bis der Empfänger–MTA erreicht ist. Dieser Agent übergibt die empfangene Nachricht dem UA. Der Empfänger kann die Nachricht mit Hilfe seines UA lesen, speichern, beantworten oder andere Funktionen ausführen.

- **Simple Mail Transfer Protocol (SMTP)**

  Mit Hilfe des E-Mail Protokolls SMTP ist es möglich, Text–Nachrichten im 7-Bit ASCII Format im Internet zu versenden.

- **Multipurpose Internet Mail Extensions (MIME)**

  Die strikte Beschränkung auf den 7-Bit Zeichensatz wurde mit MIME aufgehoben. Nach dem MIME–Standard ist es nun möglich, neben Textnachrichten in verschiedenen Zeichensätzen auch Audio-, Video-, Bild- und sogar Binärdaten zu versenden.
  Damit diese Daten im Internet übertragen werden können, nutzt MIME das SMTP–Protokoll, d.h. zu übertragende Daten werden in 7-Bit ASCII Zeichen, bzw. den sogenannten *base64–Code* konvertiert (drei aufeinanderfolgende Byte der Eingabe, werden in vier 7-Bit ASCII Zeichen übersetzt) und als SMTP–Nachricht verschickt. Werden die Daten in einem Intranet oder in einem Subnetz übertragen, das auch weitere Zeichensätze zuläßt, so sind auch andere Kodierungen (z.B. proprietäre oder 8-Bit ASCII) möglich.

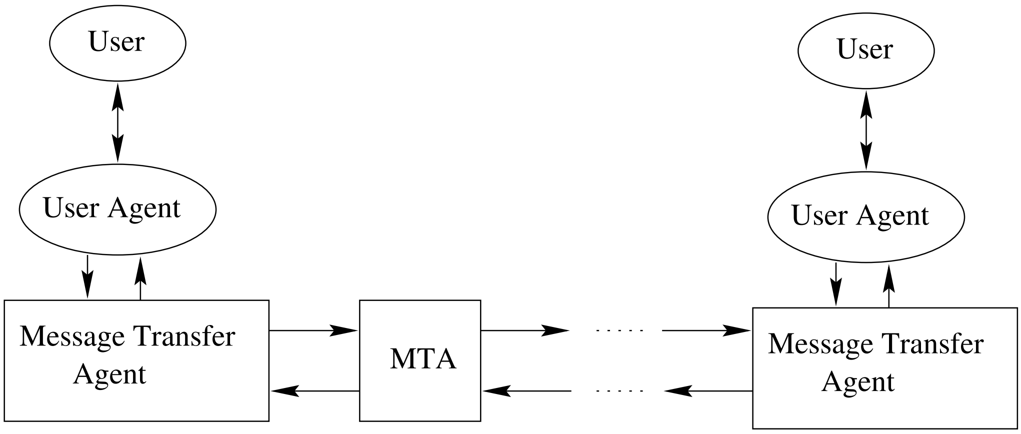

Abbildung 11.1: Message Handling System Model

Um auf Empfängerseite die Daten richtig dekodieren und die richtige Applikation zur Darstellung auswählen zu können, werden den MIME–Nachrichten u.a. Informationen über den Nachrichtentyp (`Content-Type`) sowie das verwendete Kodierungsverfahren (`Content-Transfer-Encoding`) vorangestellt.

11.1 Sicherheitsanforderungen bei E–Mail

E-Mail Nachrichten werden im Klartext übertragen oder im Falle von MIME kodiert, wobei diese Kodierung eigentlich eine Konvertierung und nicht etwa eine kryptologische Funktion darstellt. E-Mail Nachrichten können daher sehr leicht abgehört und auf jedem Vermittlungsrechner bzw. auf jedem MTA verändert oder vernichtet werden. Sollen sensible oder private Daten übertragen werden, ergibt sich daraus die Forderung nach Vertraulichkeit. Um die Vertraulichkeit bei E-Mail (auch über offene Netze) zu gewährleisten, muß die Nachricht in irgendeiner Weise verschlüsselt werden. Die Integrität der Nachricht ist zu sichern, um eventuelle Veränderungen erkennen zu können.

Ein weiteres sicherheitstechnisches Problem bei Elektronischer Post ist die Authentisierung. Die Absenderangabe in einer E-Mail erfüllt diese Sicherheitsanforderung in keinster Weise, da es für jeden Absender trivial ist, eine beliebige (auch ungültige) Netzadresse als Absender in seine E-Mail zu schreiben. Im Internet werden außerdem sogenannte *Remailer* als Dienst angeboten, die auch die Route, über die die Nachricht versandt wurde, verschleiern. Damit ist es dem Empfänger auch nicht möglich, den Rechner oder das Subnetz, aus dem die Nachricht kam, zu ermitteln.

Soll die Elektronische Post für Nachrichten verwendet werden, die den Charakter von Rechtsgeschäften oder Verträgen haben, muß zudem die Verbindlichkeit gesichert werden.

Im folgenden werden die Protokolle bzw. Protokollerweiterungen PEM und S/MIME und die Anwendung PGP für sichere E-Mail Kommunikation vorgestellt und auf ihre Eignung zur Durchsetzung der genannten Sicherheitsanforderungen untersucht.

11.2 Privacy Enhancement for Internet Electronic Mail (PEM)

Privacy Enhancement for Internet Electronic Mail (PEM) war einer der ersten und umfassendsten Versuche, einen Internet–Standard für die Absicherung von E-Mail zu definieren. PEM besteht aus vier Teilen, Part I [Lin93] beschäftigt sich mit Verschlüsselung, Authentisierung und Integrität, Part II [Ken93] mit dem Management von Schlüsseln mit Hilfe von Zertifikaten, Part III [Bal93] mit den konkreten Algorithmen und Part IV [Kal93b] mit dem Management der Zertifikate selbst.

PEM kann die Integrität der Nachricht, die Authentisierung und die Vertraulichkeit sichern. Falls ein asymmetrisches Verschlüsselungsverfahren verwendet wird, kann auch die Verbindlichkeit in Form einer digitalen Signatur durchgesetzt werden.

Zur Sicherung der Integrität wird ein Hash–Verfahren verwendet. Die zu übertragende Zeichenkette bzw. das Verfahren wird bei PEM als *Message Integrity Check (MIC)* bezeichnet. Der MIC erfüllt dieselbe Funktion wie der MAC und ist somit nur ein Synonym für diesen Begriff.

Es werden vier verschiedene PEM–Nachrichten Typen (`Proc-Type`) definiert:

- `ENCRYPTED` für signierte und verschlüsselte Nachrichten
- `MIC-ONLY` für signierte, unverschlüsselte Nachrichten
- `MIC-CLEAR` für signierte, unverschlüsselte Nachrichten, die auch ohne PEM–fähigen User Agent gelesen werden können, da sie im Klartext (ohne base64–Kodierung) übertragen werden. Ohne PEM–fähigen UA kann der MIC aber nicht verifiziert werden.
- `CRL` für Certificate Revocation Lists, die der Rücknahme und dem Widerruf von Zertifikaten dienen.

Bei PEM wird eine zweistufige Schlüsselhierarchie definiert. Der *Data Encrypting Key (DEK)* dient zum Verschlüsseln von Nachrichten und sichert damit die Vertraulichkeit. Für jede Nachricht wird ein neuer DEK berechnet. Ein vorheriger Austausch ist nicht notwendig. Für alle Nachrichten, die zwischen Sender und Empfänger ausgetauscht werden, wird der *Interchange Key (IK)* verwendet. Er dient zur Verschlüsselung von DEK und MIC, damit diese mit in der Nachricht übertragen werden können. Diese Funktion des IK wird in PEM als Schlüsselmanagement bezeichnet. Um Verwechslungen zu vermeiden, soll im folgenden von DEK–MIC–Management gesprochen werden.

Es können sowohl symmetrische als auch asymmetrische Verschlüsselungsverfahren für das DEK–MIC–Management verwendet werden. Wird ein symmetrisches Verfahren verwendet, so dient der IK zur Verschlüsselung von DEK und MIC, bei einem asymmetrischen Verfahren wird der öffentliche Schlüssel des Empfängers zur Verschlüsselung des DEK und der private Schlüssel des Senders zur Verschlüsselung des MIC verwendet. In diesem Fall besteht der IK aus zwei Komponenten.

PEM–Nachrichten werden mittels SMTP–Nachrichten übertragen. Die folgenden *Encapsulation Boundaries* begrenzen die PEM–Nachricht innerhalb der SMTP–Nachricht.

```
-----BEGIN PRIVACY-ENHANCED MESSAGE-----
-----END PRIVACY-ENHANCED MESSAGE-----
```

Die PEM–Nachricht selbst setzt sich aus einem Header und der, durch eine Leerzeile getrennten, eigentlichen Nachricht zusammen. Der Header besteht aus <Variable> ":"<Wert> Paaren, die die nötigen Informationen zur Verarbeitung durch den Empfänger enthalten.

Die Erzeugung einer PEM–Nachricht umfaßt folgende Schritte:

1. Schreiben der Nachricht im Zeichensatz der Maschine
2. Konvertierung in 7-Bit SMTP–Format
3. MIC–Berechnung und Erzeugung eines Schlüssels, falls der MIC mit einem Schlüssel gesichert werden soll
4. Berechnung des DEK und Verschlüsselung (nur für `ENCRYPTED`)
5. base64–Kodierung (für `ENCRYPTED` und `MIC-ONLY` nicht für `MIC-CLEAR`)
6. Bildung des PEM–Pakets (Encapsulation Boundaries, Header, Nachricht)

Es werden alle Nachrichten signiert, d.h. es gibt keine Nachricht, die „nur“ verschlüsselt, aber nicht signiert ist.

Verschlüsselung der Nachricht

Bisher ist nur ein symmetrisches Verfahren zur Verschlüsselung spezifiziert. Es wird DES im CBC–Mode verwendet. Der Sender berechnet den DEK und einen zufälligen Initialisierungsvektor (IV) für DES. Im `DEK-Info` Feld des PEM–Headers ist das verwendete Verfahren sowie zusätzliche Informationen, in diesem Fall der IV, anzugeben. Das generische Format des `DEK-Info` Feldes macht es leicht möglich, weitere Verschlüsselungsverfahren in den Standard aufzunehmen. Der DEK selbst wird verschlüsselt an den Empfänger übermittelt (s.u.).

Berechnung des MIC

Der Message Integrity Check wird nur über den PEM–Body und nicht über die Header Felder berechnet. Als Algorithmus wird MD2 oder MD5 verwendet. Der MIC wird abhängig vom Verschlüsselungsverfahren, das beim DEK–MIC–Management verwendet wird, verschlüsselt. Durch die Verschlüsselung des MIC wird die Authentisierung gewährleistet.

DEK–MIC–Management

Zur Verschlüsselung von DEK und MIC können sowohl symmetrische als auch asymmetrische Verfahren Verwendung finden.

- **symmetrische Verfahren**

  Als symmetrische Verfahren werden DES im *Electronic Codebook (ECB)* Mode oder im *Encrypt–Decrypt–Encrypt (EDE)* Mode spezifiziert.
  DEK und MIC werden mit dem IK verschlüsselt und im `Key-Info` Feld an den Empfänger übertragen. Es setzt sich in diesem Fall aus einem String (ID), der das Verschlüsselungsverfahren spezifiziert, einem ID für den MIC–Algorithmus, dem DEK und dem MIC zusammen.

  Im `Originator-ID-Symmetric` Feld werden Informationen übermittelt, damit der Empfänger den IK zur Entschlüsselung von DEK und MIC bestimmen kann. Dieses Feld kann weggelassen werden, falls der Empfänger die benötigten Informationen aus dem `Recipient-ID-Symmetric` Argument ermitteln kann.

- **asymmetrische Verfahren**

  Als einziges asymmetrisches Verfahren wird RSA festgelegt. Der DEK wird mit dem öffentlichen Schlüssel des Empfängers verschlüsselt und im `Key-Info` Feld übertragen. In diesem Fall besteht der Attributwert nur aus dem ID des Verschlüsselungsalgorithmus und dem DEK.

  Der MIC wird mit dem privaten Schlüssel des Senders signiert und im `MIC-Info` Feld eingetragen. Der Wert dieses Feldes besteht aus einem ID für den MIC–Algorithmus, einem ID für den Signaturalgorithmus und dem MIC.

  Die Informationen, die der Empfänger braucht, um den IK bestimmen zu können, entnimmt er aus dem `Originator-ID-Asymmetric` und dem `Recipient-ID-Asymmetric` Feldern. Die Attributwerte enthalten den ID der Zertifizierungsstelle, ein ID des entsprechenden Zertifikates und die Versionsnummer bzw. Gültigkeitsinformationen.

  Der Sender kann sein Zertifikat im PEM–Header mitübertragen. Dafür wird das `Originator-Certificate` Feld verwendet. Wird diese Möglichkeit genutzt, so darf das `Originator-ID-Asymmetric` Feld nicht benutzt werden. Zusätzlich können beliebig viele `Issuer-Certificate` Felder angegeben werden. Hiermit kann der Sender zusätzlich Zertifikate von Zertifizierungsstellen im PEM–Header an den Empfänger übermitteln. Damit können im Prinzip alle Zertifikate des vollständigen Zertifizierungspfades vom Senderzertifikat bis zu dem Zertifikat jener Zertifizierungsstelle, die von Sender und Empfänger gleichermaßen als vertrauenswürdig akzeptiert wird, in der PEM–Nachricht mitübertragen werden. In diesem Fall muß der Empfänger keinerlei Zertifikate von irgendwelchen Datenbanken, Zertifizierungsstellen, Directory–Diensten oder sonstigen Repositories besorgen.

PEM erlaubt auch, daß ein Sender mehrere IK's mit demselben Empfänger vereinbart und daß der Sender die Nachricht an mehrere Empfänger gleichzeitig senden kann. Deshalb kann der PEM–Header mehrere Blöcke enthalten, in denen jeweils ein `Originator-ID` und ein oder mehrere `Recipient-ID`'s angegeben sind. Jeder dieser Blöcke bestimmt den IK einer oder mehrerer Sender–Empfänger Verbindungen.

11.3 Security Services for MIME (S/MIME)

S/MIME [PKC95, DHR97] ist eine Erweiterung des MIME–Protokolls, die von RSA Data Security Inc. definiert wurde. Die RSA–Laboratories haben eine Reihe von sog. *Public–Key Cryptography Standards (PKCS)* definiert, die u.a. die Verschlüsselung mittels RSA, den Diffie–Hellman–Schlüsselaustausch, Zertifikate und die Syntax von kryptographischen Nachrichten festlegen. S/MIME verwendet die in PKCS #7 (*Cryptographic Message Syntax Standard* [RSA93c]) festgelegte Syntax für die Nachrichten und für Zertifikatsanforderungen das in PKCS #10 (*Certification Request Syntax Standard* [RSA93b]) beschriebene Format. Die Verschlüsselungsverfahren sind in PKCS #1 [RSA93a] festgelegt.

Alle (Daten–) Objekte, die in PKCS definiert werden, sind in *ASN.1* (*Abstract Syntax Notation One* [CCI88b]) spezifiziert. Damit die ASN.1 Objekte zwischen heterogenen Systemen übertragen werden können, muß eine Kodierung in einem „Netzformat" festgelegt werden. Hierzu wird entweder *BER* (*Basic Encoding Rules* [CCI88c]) oder eine Untermenge davon: *DER (Distinguished Encoding Rules)* verwendet.

Bisher wurde nicht versucht, die PKCS-Standards mittels RFC's beim IAB zu standardisieren. Offensichtlich wird dies auch nicht angestrebt, da RSA Laboratories in ihrem Standard schreiben: "... the standards making process is not the usual committee–oriented method"([Kal93a]).

S/MIME definiert zwei neue Content-Types für MIME:

1. `application/x-pkcs7-mime`
2. `application/x-pkcs10`

Der erste wird für signierte bzw. verschlüsselte E-Mails, der zweite für Zertifikatsanforderungen verwendet.

Verschlüsselung und Signatur (`application/x-pkcs7-mime`)

E–Mails im MIME–Format können als signiertes, verschlüsseltes oder signiertes und verschlüsseltes Dokument innerhalb eines `application/x-pkcs7-mime` Pakets verschickt werden. Hierzu werden die `ContentInfo`–Typen `SignedData`, `EnvelopedData` und `SignedAndEnvelopedData`, die in PKCS #7 definiert sind, verwendet. S/MIME sichert damit die Vertraulichkeit und die Verbindlichkeit der Nachricht.

Der Inhalt eines `x-pkcs7-mime` Pakets muß selbst wieder ein gültiger MIME–Body sein, damit nach der Verifikation bzw. nach der Entschlüsselung dieser MIME–Body an den MIME–Agenten zur Weiterverarbeitung übergeben werden kann. Die Erzeugung umfaßt folgende Schritte:

1. Erzeugung eines Standard MIME–Body–Parts
2. Abschluß durch `<CR><LF>` (Carriage Return und Line Feed)
3. Verschlüsselung und/oder Signatur (s.u.)
4. Bildung des entsprechenden PKCS #7 `ContentInfo`–Objektes
5. BER–Kodierung („Netzformat“, 8-Bit)
6. base64–Kodierung („SMTP-Format“, 7-Bit)
7. Einfügen in `application/x-pkcs7-mime`–Body–Part

Im folgenden wird die Erzeugung der verschiedenen `ContentInfo`–Typen nach PKCS #7 beschrieben:

- **Signierte E-Mail** (`SignedData`)

  Jeder Unterzeichner berechnet den MAC des MIME-Bodies, falls verschiedene MAC–Algorithmen verwendet werden, sonst wird der Hash–Wert nur einmal berechnet. Der Standard–MIME–Body muß in 7 Bit ASCII kodiert sein, da der MAC nicht über 8 Bit- oder Binärdaten berechnet werden darf. Der MAC sichert die Integrität der Nachricht.

  Der MAC und zusätzliche Informationen über den verwendeten Algorithmus werden als `DigestInfo` zusammengefaßt und anschließend BER–kodiert. Die daraus resultierende Bitkette wird mit dem privaten Schlüssel des jeweiligen Unterzeichners verschlüsselt. Für jeden Unterzeichner werden diese Daten, Informationen über evtl. Zertifikate des Unterzeichners und evtl. weitere Attribute im `SignerInfo`–Objekt zusammengefaßt. Die `SignerInfo`–Objekte aller Unterzeichner evtl. Zertifikate oder Certificate Revocation Lists (CRL, optional), Informationen über die Art der unterschriebenen Daten und der MIME–Body bilden zusammen den `ContentInfo`–Typen `SignedData`.

  Für das Senden einer Nachricht soll MD5 verwendet werden. Beim Empfang soll der S/MIME–Agent zusätzlich MD2 [Kal92] als Hash–Algorithmus akzeptieren. Für die Berechnung der Signatur wird das RSA Verfahren verwendet.

- **Verschlüsselte E–Mail** (`EnvelopedData`)

  Zur Erzeugung eines „digitalen Briefumschlags“ muß der Sender einen Schlüssel generieren. Dieser Kommunikationsschlüssel wird für jeden Empfänger mit dessen öffentlichem Schlüssel verschlüsselt und bildet mit empfängerspezifischen Informationen das `RecipientInfo` Objekt. Der MIME–Body wird mit dem Kommunikationsschlüssel verschlüsselt und bildet zusammen mit allen `RecipientInfo` Objekten aller Empfänger den Typ `EnvelopedData`.

  Als symmetrisches Chiffrierverfahren wird RC2 im CBC–Mode (Cipher–Block–Chaining), DES im CBC–Mode oder DES EDE3-CBC vorgeschlagen. Auch hier wird als Public Key Verfahren RSA verwendet.

- **Signierte und verschlüsselte E–Mail** (`SignedAndEnvelopedData`)

  Die zu übertragende Nachricht wird wie oben beschrieben von allen Unterzeichnern signiert. Die digitalen Signaturen werden zusätzlich mit dem generierten Kommunikationsschlüssel überschlüsselt, d.h. der MAC wird doppelt verschlüsselt, zuerst mit dem privaten Schlüssel des Unterzeichners und zusätzlich mit dem Kommunikationsschlüssel. Dann wird die Nachricht wie beim Typ `EnvelopedData` für alle Empfänger verschlüsselt. `SignedAndEnvelopedData` besteht also aus den `SignerInfo`'s aller Unterzeichner, den `RecipientInfo`'s für jeden Empfänger und dem verschlüsselten MIME–Body.

Zertifikatsanforderung (`application/x-pkcs10`)

Der Typ `application/x-pkcs10` definiert ein Format, mit dessen Hilfe Zertifizierungsanforderungen (Certification Request) sicher an Zertifizierungsstellen (Certification Authority, CA) übermittelt werden können. Die CA kann aus dem Certificate Request ein Zertifikat erzeugen, das die Identität des Nutzers (über den Distinguished Name) an seinen öffentlichen Schlüssel bindet. Dazu werden die Daten aus dem Certificate Request in ein gültiges X.509–Format [CCI88d] überführt und von der CA signiert. D.h. die CA bürgt dafür, daß der im Zertifikat angegebene öffentliche Schlüssel auch dem angegebenen Subjekt (dem Zertifizierten) gehört.

Um eine Zertifikatsanforderung zu erzeugen, ist der folgende Vorgang auszuführen:

1. Erzeugung eines Schlüsselpaares (öffentlicher und privater Schlüssel)
2. Bildung eines PKCS #10 Certificate Request
3. Kodierung in BER oder DER
4. Kodierung in base64
5. Erzeugung des `application/x-pkcs10`–Body

Zur Erzeugung des PKCS #10 Certificate Request muß der Sender seinen Distinguished Name (DN, vgl. Abschn. 6.2) erzeugen und mit seinem öffentlichen Schlüssel zum

`CertificateRequestInfo` Objekt verbinden, das dann DER–kodiert wird. Das resultierende Datenobjekt wird vom Sender signiert, damit ein Angreifer den öffentlichen Schlüssel nicht durch seinen eigenen ersetzen kann (vgl. Abschn. 2.2.1). Die Signatur bildet zusammen mit Informationen über den Signaturalgorithmus und dem `CertificateRequestInfo` den `CertificateRequest`.

11.4 Pretty Good Privacy (PGP)

Bei *Pretty Good Privacy (PGP)* [Zim94a, Zim94b, ASZ96] handelt es sich im Gegensatz zu PEM oder S/MIME nicht um ein Protokoll, sondern um eine Anwendung, die zur Verschlüsselung von E-Mail und Dateien entwickelt wurde und mittlerweile auf sehr vielen Plattformen verfügbar ist.

PGP kann Integrität, Vertraulichkeit und Verbindlichkeit einer Nachricht sowie die Authentisierung sichern. Zudem wird PGP zur Schlüsselverwaltung und –verteilung eingesetzt. Mittels PGP können signierte, verschlüsselte oder signierte und verschlüsselte Nachrichten erzeugt werden.

Verschlüsselung einer Nachricht

PGP verwendet ein hybrides Verschlüsselungsverfahren. Zur Verschlüsselung der Nachricht wird IDEA verwendet. Für jede Nachricht wird ein eigener IDEA–Schlüssel zufällig gewählt. Dieser Schlüssel wird mit dem öffentlichen Schlüssel des Empfängers verschlüsselt und der Nachricht hinzugefügt. Als asymmetrisches Verschlüsselungsverfahren wird RSA verwendet.

Signieren einer Nachricht

Eine digitale Signatur bei PGP besteht aus einem MAC, der über die Nachricht berechnet und mit dem privaten Schlüssel des Senders verschlüsselt wird. Als MAC–Algorithmus findet MD5 und als asymmetrisches Verfahren RSA Verwendung.

Zusätzliche Merkmale

Damit die PGP–Nachricht in einem SMTP–Body verschickt werden kann, besteht die Möglichkeit, die verschlüsselte oder signierte Nachricht mit Hilfe der base64–Kodierung in 7-Bit ASCII Zeichen zu konvertieren. Da die base64–Kodierung den Umfang einer Nachricht um rund 17 % vergrößert, wird die Nachricht vor der Verschlüsselung bzw. vor der Signatur komprimiert. Neben der Verringerung der Nachrichtenlänge wird dadurch auch die kryptologische Sicherheit erhöht. Zur Komprimierung werden die ZIP–Routinen verwendet. Die Komprimierungsfunktion kann abgeschaltet werden, damit Nachrichten, die nicht verschlüsselt sondern nur digital signiert werden, für den Empfänger, auch ohne Anwendung von PGP, lesbar bleiben.

Schlüsselverwaltung

PGP verfolgt keinen zentralen sondern einen dezentralen Ansatz zur Schlüsselverwaltung bzw. Zertifizierung.

Ein PGP Nutzer besitzt zwei sog. Keyrings, einen auf dem sich sein privater Schlüssel befindet (`secring`) und einen auf dem sich alle öffentlichen Schlüssel seiner Kommunikationspartner befinden (`pubring`). Sein geheimer Schlüssel wird vor der Speicherung mit einer Phrase oder einem Paßwort als Schlüssel verschlüsselt und damit vor Ausspähen oder Veränderung, auch von Seiten eines böswilligen Administrators, geschützt.

Der öffentliche Schlüssel eines Partners muß sicher mit dessen Identität verbunden sein. Um dies zu gewährleisten, verwendet auch PGP Zertifikate. Allerdings werden diese nicht von einer Zertifizierungsstelle ausgestellt, sondern die Kommunikationspartner zertifizieren sich gegenseitig. Zur Verteilung der öffentlichen Schlüssel wird i.d.R. E-Mail verwendet.

Es gibt zwei Möglichkeiten, um einen öffentlichen Schlüssel eines Kommunikationspartners in den eigenen `pubring` aufzunehmen. Haben die beiden Partner die Möglichkeit Out-of-Band, d.h. nicht über E-Mail oder sonstige Netzverbindungen, zu kommunizieren, so signiert Alice ihren eigenen öffentlichen Schlüssel und schickt ihn per E-Mail an Bob. Bob verifiziert die Signatur und berechnet einen Hash-Wert (MAC) über den öffentlichen Schlüssel. Dieser Wert wird Out-of-Band (z.B. mittels Telefon) mit dem von Alice berechneten Wert verglichen. Ist die Signatur gültig und stimmen beide Hash-Werte überein, so kann Bob den Schlüssel in seinen Keyring aufnehmen. Bob kann nun den öffentlichen Schlüssel von Alice signieren und an sie zurückschicken. Bob tritt damit als „Introducer“ von Alice auf, denn mit diesem Zertifikat kann Alice ihren öffentlichen Schlüssel sicher an alle Kommunikationspartner übermitteln, die bereits im Besitz von Bob's öffentlichem Schlüssel sind. Eine Out-of-Band Kommunikation ist in diesem Fall nicht notwendig.

Die Partner, die das Zertifikat von Alice erhalten haben und Bob und Alice trauen, können nun ebenfalls Alice's Schlüssel signieren. Auf diese Weise entsteht ein Web of Trust von Kommunikationspartnern, die einander trauen und sich gegenseitig zertifiziert haben.

Die Partner müssen den Schlüssel von Alice nicht signieren, da Vertrauen nicht notwendigerweise transitiv sein muß. D.h. falls Bob Alice traut und Charly wiederum Bob, so muß Charly nicht notwendigerweise auch Alice trauen. Deshalb kann Charly bei der Aufnahme des öffentlichen Schlüssels von Alice in seinen Keyring Alice als `unknown`, `untrusted`, `marginally trusted` oder `completeley trusted` klassifizieren. Erhält er nun öffentliche Schlüssel oder Nachrichten, in denen Alice als Introducer auftritt, kann er diese, je nach Klassifikation, akzeptieren oder ablehnen. Außerdem kann die maximale Länge des Zertifizierungspfades eingestellt werden. Der Benutzer bestimmt damit selbst, wieviel Vertrauen er bereit ist, anderen Kommunikationspartnern entgegenzubringen. Setzt er den Wert bspw. auf Null, so werden nur Nachrichten akzeptiert, die von Sendern kommen, die er selbst zertifiziert hat.

11.5 Bewertung und Vergleich

Im folgenden werden PEM, S/MIME und PGP hinsichtlich ihrer Einsatzmöglichkeiten bewertet. Die Ergebnisse sind in Tabelle 11.1 zusammengefaßt, sie spiegeln die Entwicklung zum jetzigen Zeitpunkt wider. Da das betrachtete Marktsegment sehr dynamisch ist und sehr viel Entwicklungsarbeit stattfindet, ist es möglich, daß diese Bewertung in einigen Monaten völlig anders ausfallen würde.

Kriterien	PEM	S/MIME	PGP
Integrität	+	+	+
Vertraulichkeit	+	+	+
Authentisierung	–/+	+	+
Verbindlichkeit	nur mit asymm. Verf.	+	+
marktfähige Produkte	– –	–/+	–
Plattformunabhängigkeit		+	++
maximale Schlüssellänge (symm./asymmetrisches Verf.)		40 Bit/512 Bit	128 Bit/1024 Bit
MIME–Tauglichkeit	–	+	+
Integration in MIME	–	–	+
Infrastrukturaufwand		hoch	gering
Installationsaufwand		gering	gering
Transparenz		+	–/+
Integration in UA		– –	+ +

Tabelle 11.1: Bewertung von PEM, S/MIME und PGP

Das wichtigste Kriterium für die Bewertung der Verfahren ist deren Fähigkeit, die Sicherheitsanforderungen durchzusetzen. Sowohl PEM und S/MIME als auch PGP sind in der Lage, die geforderten Sicherheitsanforderungen zu erfüllen. PEM bietet die Möglichkeit, bei der Verschlüsselung von Kommunikationsschlüssel (DEK) und MIC zwischen einem symmetrischen und einem asymmetrischen Verschlüsselungsverfahren zu wählen. Da die anderen beiden Verfahren diese Möglichkeit nicht bieten, ist hier ein direkter Vergleich nicht möglich. Die Authentisierung bei PEM beruht im symmetrischen Fall lediglich auf dem gemeinsamen Schlüssel und ist daher als nicht ausreichend anzusehen. Für das DEK–MIC–Management sollte daher im praktischen Einsatz ein asymmetrisches Verfahren Verwendung finden. In diesem Fall sind die Verfahren hinsichtlich der Durchsetzung der Sicherheitsanforderungen äquivalent.

Um die Verfahren in einem heterogenen Unternehmensnetz einsetzen zu können, sind marktfähige Produkte, die auf verschiedenen Plattformen zur Verfügung stehen sollten, notwendig. Bei PEM gibt es bisher sehr wenige Implementierungen, die zudem nicht als Software–Produkte verkauft, sondern als freie Software und damit ohne Support im Internet angeboten werden (z.B. RIPEM[1]). Da es zu wenig Implementierungen gibt, kann die Frage nach der

[1] http://www.cs.indiana.edu/ripem

Plattformunabhängigkeit nicht abschließend beantwortet werden. Auch bei S/MIME gibt es im Moment wenige Implementierungen. Die Firma OpenSoft[2] liefert einen E-Mail User Agent für Windows 95 und Windows NT. Der Communicator der Firma Netscape unterstützt S/MIME ebenfalls. Eine Übersicht über die Hersteller von S/MIME–fähigen UA findet man auch bei der Firma RSA[3]. Die meisten Plattformen werden derzeit von PGP unterstützt. Da der Quellcode frei verfügbar ist, wurde es auf sehr viele Plattformen portiert. Bisher wird PGP nicht in Europa, sondern nur in den USA[4] kommerziell vertrieben.

Im Zusammenhang mit dem Vertrieb von Software (SW) zur Durchsetzung von Sicherheitsanforderungen stellt sich die Frage nach der kryptologischen Sicherheit und hier insbesondere nach der maximalen Schlüssellänge, die die Produkte zulassen. Auch für Produkte, die zur Absicherung von Mailverkehr verwendet werden, gelten die strengen Exportbeschränkungen der Vereinigten Staaten (vgl. auch Bemerkung auf S. 51 f.). Es dürfen lediglich solche Produkte aus den USA exportiert werden, deren maximale Schlüssellänge bei symmetrischen Verfahren, i.d.R. auf 40 Bit, beschränkt wird. Für die Verschlüsselung von Nachrichten, bei denen für jede Nachricht ein neuer Schlüssel verwendet wird, mag diese Schlüssellänge noch ausreichen. Unter dem Gesichtspunkt der rasanten technischen Entwicklung auf Seiten der Hardware und dem zunehmenden Einsatz massiv paralleler Systeme dürfte klar sein, daß eine Schlüssellänge von 40 Bit für den Kommunikationsschlüssel und ein 512 Bit langer Schlüssel zur Signatur von Zertifikaten, die u.U. mehrere Jahre gültig sein sollen, unzureichend sind.
Da alle Produkte, die S/MIME implementieren aus den USA kommen und deshalb nur beschränkte Schlüssellängen zulassen, ist dies bei der Auswahl besonders zu berücksichtigen. Von PGP existiert eine sog. internationale Version und damit ist es im Moment das einzige Produkt, das größere Schlüssellängen (bis 1024 Bit) zuläßt.

Die kryptologische Sicherheit wird nicht nur von der Schlüssellänge bestimmt. Ein weiterer wichtiger Faktor ist die fehlerfreie Implementierung bzw. die Validierung der SW. Da die Verifikation von SW ein extrem aufwendiger Prozeß ist, kann sie im Rahmen dieser Arbeit nicht geleistet werden. Trotzdem muß dieser Punkt beachtet und in den Entscheidungsprozeß mit aufgenommen werden. Von den betrachteten Produkten stellt lediglich PGP Quelltexte, ohne die ein Verifikationsprozeß nicht durchführbar ist, zur Verfügung. Bei den anderen Herstellern muß der Nutzer auf die Sorgfalt des entsprechenden Herstellers vertrauen.

Da in europäischen Unternehmen in den seltensten Fällen ein 7-Bit Zeichensatz für die Kommunikation mittels E-Mail ausreicht, ist die Einbindung eines Verfahrens in MIME bzw. die Fähigkeit eines Produktes, MIME zu unterstützen von zentraler Bedeutung. Der Standard für PEM unterstützt kein MIME, da MIME später als PEM spezifiziert wurde. Es gibt auch keine MIME–Erweiterungen, um PEM in MIME einzubinden. Von S/MIME wird MIME insofern unterstützt, als ein S/MIME–Body selbst wieder ein MIME–Body sein muß, d.h. S/MIME entspricht einem Aufsatz auf MIME. Der S/MIME–Agent wird einfach dem MIME User Agent als Filter vorangestellt. PGP ist natürlich auch in der Lage, MIME–Nachrichten zu verarbeiten. Es

[2] ExpressMail: http://www.opensoft.com/products/expressmail/overview/client/
[3] http://www.rsa.com/rsa/S-MIME/html/products.html
[4] http://www.pgp.com

kann analog zu S/MIME wie ein Filter für den MIME–Agenten verwendet werden. Zusätzlich kann PGP auch in MIME selbst integriert werden. Dazu wurden die drei neuen `Content-Types` `application/pgp-signature`, `application/pgp-encrypted` und `application/pgp-keys` definiert. Um MIME–Nachrichten diesen Typs direkt im MIME User Agent verarbeiten zu können, ist PGP für die aufgeführten `Content-Types` beim MIME–Agenten zu registrieren. Die Erzeugung dieser Typen wird in [Elk96] beschrieben und erfordert abhängig vom *API (Application Programming Interface)* des MIME–Agenten nur geringfügige Änderungen.

Alle betrachteten Verfahren wirken auf die End-to-End Kommunikation zwischen zwei oder mehreren Benutzern bzw. UA's. Daher ist die Transparenz und die Benutzerfreundlichkeit der Verfahren zu bewerten. Eine Anwendung ist vollständig transparent, wenn der Nutzer die Maßnahme zur Sicherung der Nachricht weder anstoßen muß noch dies überhaupt erkennen kann. Die Vorteile der Transparenz sind der geringere Schulungsaufwand der Endbenutzer und die höhere Sicherheit, da ein Nutzer die Sicherheit nicht durch falsche Bedienung schwächen kann. Sind die Verfahren vollkommen transparent, dürften auch keine Akzeptanzprobleme von Seiten der Nutzer auftreten, die andernfalls die Bedienung der Software erlernen müßten.

Nachteilig ist die Unmöglichkeit für den Endbenutzer, das System seinen Anforderungen und Wünschen anzupassen, wenn für bestimmte Verbindungen höhere Sicherheitsanforderungen z.B. größere Schlüssellängen oder andere Verschlüsselungsverfahren einzusetzen sind.

Da der Benutzer zumindest die Möglichkeit haben muß zu entscheiden, ob er die E-Mail signieren und/oder verschlüsseln will, ist vollständige Transparenz von keinem der Verfahren durchzusetzen. Im Prinzip lassen sich alle drei Verfahren so in einen UA integrieren, daß der Benutzer nur angeben muß, ob er die Nachricht verschlüsselt und/oder signiert haben möchte. Da es von PEM nur eine Implementierung unter UNIX gibt, die in das `mail` Programm integriert wird, können hier über Transparenz und Benutzerfreundlichkeit keine abschließenden Angaben gemacht werden.

Die Implementierung von S/MIME im Netscape Communicator läßt sich sehr einfach bedienen, falls die notwendigen Zertifikate bereits vorhanden sind. Der Benutzer muß lediglich die seinen Wünschen entsprechenden Buttons klicken und dann die Nachricht abschicken. Signiert er eine Nachricht, wird automatisch sein Zertifikat in der Nachricht mitübertragen. Problematischer ist, wenn der Nutzer noch kein Zertifikat besitzt, oder ein Zertifikat eines Kommunikationspartners benötigt, um ihm eine verschlüsselte Nachricht schicken zu können. Im letzten Fall reicht es aus, den Partner zu bitten, sein Zertifikat mittels E-Mail zu schicken. Falls der Nutzer selbst noch kein Zertifikat besitzt, muß er sich an eine CA wenden und ein Zertifikat beantragen. Dies kann entweder über WWW oder über E-Mail erfolgen. Dazu muß er wissen, wie ein Schlüsselpaar generiert wird und wie ein Zertifikat aufgebaut ist, um die entsprechenden Daten richtig angeben zu können. S/MIME benötigt von einer CA ausgestellte Zertifikate.

Bei PEM werden grundsätzlich alle Zertifikate des gesamten Zertifizierungspfades innerhalb der Nachricht mitübertragen. Dies erleichtert zwar die Verifikation, allerdings erhöht sich dadurch der Umfang der Nachricht erheblich. Die Kette der Zertifikate kann sehr viel länger als die eigentliche Nachricht werden.

Da PGP neben der Schlüsselerzeugung auch für die Zertifizierung verwendet wird und der Benutzer hier seine speziellen Sicherheitsanforderungen einstellen und durchsetzen kann, hat er mit diesem Werkzeug die größten Freiheiten. Um diese nutzen zu können und das System nicht zu schwächen, muß er aber auch die Prinzipien, die bei PGP verwendet werden, verstehen. Er muß die grundlegenden Funktionsweisen eines asymmetrischen Verschlüsselungsverfahrens und des hybriden Verfahrens von PGP im Besonderen kennen. Außerdem muß er wissen, was es bedeutet, einen signierten Schlüssel in den eigenen `pubring` aufzunehmen oder selbst einen Schlüssel zu unterschreiben.

PGP wurde als Kommandozeilenprogramm entwickelt, d.h. in der ursprünglichen Form verwendet, ist es für den Benutzer in keinster Weise transparent. Da die Quellen frei verfügbar und die Verbreitung sehr hoch ist, gibt es für PGP mit Abstand die meisten Erweiterungen und Anpassungen, um es in die verschiedensten UA einbinden zu können. Mit einigen dieser Erweiterungen läßt sich eine ähnliche Transparenz wie mit S/MIME User Agenten erreichen.

Ein wichtiges Bewertungskriterium ist auch die Frage nach dem Infrastrukturaufwand. Der Aufwand für die Installation von S/MIME bzw. PGP auf einem einzelnen Rechner ist gering. In einem Unternehmensnetz in der Größe des BMW–Netzes ist es notwendig, eine oder mehrere Zertifizierungsstellen einzurichten, da die Zertifizierung aller Endsysteme bzw. Nutzer durch eine externe CA zu teuer sein dürfte. Der Infrastrukturaufwand durch die Errichtung einer Zertifizierungshierarchie für S/MIME ist erheblich.

Da S/MIME den zentralen und PGP den dezentralen Zertifizierungsansatz vertritt, sind die beiden Methoden nur schwer direkt zu vergleichen. Allerdings ließe sich durch organisatorische Maßnahmen die dezentrale Zertifizierung von PGP auf eine zentrale Zertifizierungshierarchie abbilden. In diesem Fall würden die CA's des Unternehmens die öffentlichen Schlüssel aller PGP–Nutzer signieren. Wird allerdings der dezentrale Ansatz von PGP verfolgt und zertifizieren sich die Nutzer gegenseitig, so ist der Infrastrukturaufwand sehr gering, da keine zentrale CA benötigt wird.

Kapitel 12

Entfernter Zugriff über Remote Access Server

Die BMW AG dehnt ihr internes Netz in zunehmendem Maße dadurch aus, daß Mitarbeiter von zu Hause über ISDN– oder sonstige Wählverbindungen Zugriff auf das Unternehmensnetz haben. Neben den Vorteilen, die diese Telearbeitsplätze bieten, wird die Einhaltung der Anforderungen der Sicherheitspolitik zunehmend schwieriger und komplexer. Grundsätzlich darf der Zugriff über Wählverbindungen auf Netz– und Systemressourcen nur über wohldefinierte Zugangspunkte (Modem– bzw. ISDN–Server), die als *Remote Access Server (RAS)* oder *Terminal Access Controller (TAC)* bezeichnet werden, erfolgen. Auf alle Fälle ist zu verhindern, daß an Endsystemen von Endbenutzern Modems oder ISDN–Hardware installiert werden. Andernfalls ist es nicht zu unterbinden, daß Zugänge ins Unternehmensnetz geschaffen werden, die nicht der Sicherheitspolitik entsprechen oder den, für die Durchsetzung der Netzsicherheit Verantwortlichen, gar nicht bekannt sind. Da jede Maßnahme zur Verbesserung der Sicherheit nur so stark wie ihr schwächstes Glied ist, darf den Benutzern nicht erlaubt werden, „Hintertüren“ im System zu etablieren.

Auf einem oder mehreren zentralen Zugangspunkten ist es auch leichter Software zu installieren, die spezielle Sicherheitsanforderungen durchsetzt. Im folgenden werden die Protokolle TACACS+ und RADIUS für Terminal Access Controller und der RAS Dienst von Windows NT untersucht.

Bei TACACS+ und RADIUS handelt es sich um Protokolle zur Durchsetzung von Sicherheitsanforderungen und zur sicheren Übertragung von Authentisierungs–, Zugriffskontroll–, Konfigurations– und Auditinginformationen. Beide Protokolle sind als Client/Server–Architektur spezifiziert. Auf dem TAC läuft ein Client, der seine Links authentisieren will, und im internen Unternehmensnetz befindet sich ein *Authentisierungsserver*, der die Authentisierung, Zugriffskontrolle und das Auditing durchführt (vgl. Abb. 12.1).

Letztlich soll der Endbenutzer eine Verbindung zu einer Ressource oder einem Dienst im in-

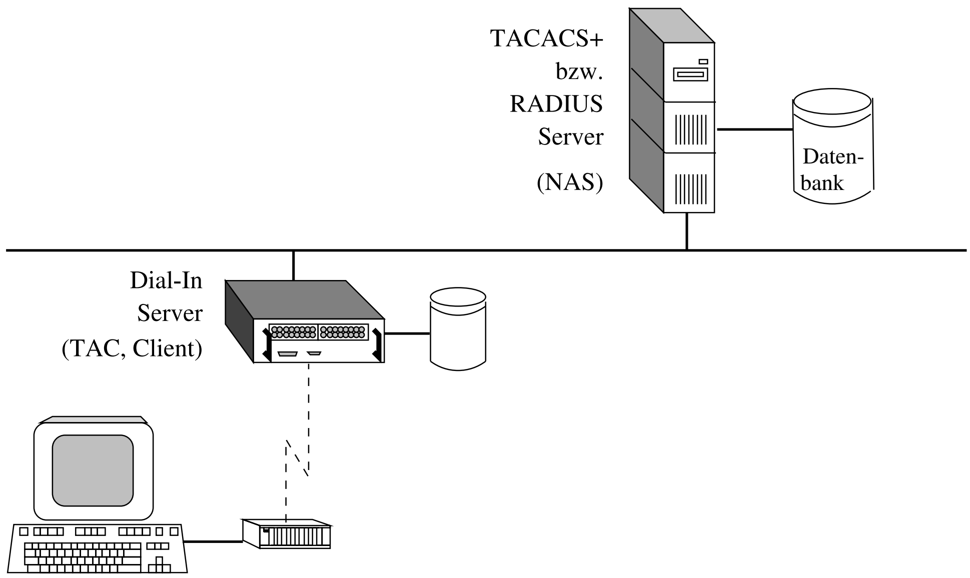

Abbildung 12.1: TACACS+ und RADIUS im Unternehmensnetz

ternen Unternehmensnetz erhalten. Dazu müssen die verschiedensten Protokolle und Dienste unterstützt werden. Die von TACACS+ und RADIUS unterstützten Dienste und Protokolle sind in Tabelle 12.1 zusammengefaßt.

12.1 TACACS und TACACS+

Die Spezifikation von *TACACS (Terminal Access Control Access Control System)* wurde bereits im Jahr 1993 veröffentlicht [Fin93] und 1996 von CISCO Systems zu *TACACS+* [CG97a] erweitert.

In der ursprünglichen Version aus dem Jahr 1993 handelt es sich um ein reines Request/Response Protokoll. Der Client schickt Requests, die der *Network Access Server (NAS)* mit Responses beantwortet. Es gibt dabei drei verschiedene Diensttypen. Neben einer Verbindung, die nur Authentisierung zuläßt, kann auch eine Login– oder eine SLIP–Verbindung aufgebaut werden.

Zur Authentisierung sendet der TACACS–Client für jede Dial–In–Verbindung eine User–ID und ein Paßwort an den NAS, der die Authentisierung vornimmt und entweder ein Accept oder eine Deny–Nachricht zurückschickt. Dabei können auf Serverseite zusätzlich Authentisierungsverfahren oder Filter installiert werden, mit denen sich komplexere Bedingungen für die Authentisierung bzw. für die Zugriffskontrolle (bei TACACS als Authorization bezeichnet) spezifizieren lassen. Als Transportprotokoll wird UDP [Pos80] (Port 49) oder TCP verwendet. Bei beiden Übertragungsverfahren wird jedoch User–ID und Paßwort im Klartext übertragen.

Unterstützte Protokolle und Dienste	TACACS+	RADIUS
SLIP (Serial Line Internet Protocol)	+	+
IPX over SLIP (proprietär von Xylogics)		+
IP over PPP (Point to Point Protocol, [Sim94])	+	+
IPX over PPP	+	+
ARAP (Apple Talk Remote Access Protocol)	+	+
LAT (Local Area Transport Protocol von DEC)	+	+
Telnet	+	+
TN3270	+	+
tty–daemon	+	
vines (Virtual Network System)	+	
vpdn (Virtual Private Data Network)	+	
rcmd (Remote Router Commands)	+	
rlogin	+	+
portmaster		+
ftp	+	
http	+	
deccp	+	
osicp	+	
unknown	+	

Tabelle 12.1: Von TACACS+ und RADIUS unterstützte Protokolle und Dienste

Um diesen Mangel zu beheben, wurde TACACS+ definiert, bei dem der Verkehr zwischen Client und NAS verschlüsselt wird. Zusätzlich werden die Funktionen und Nachrichten für Authentisierung, Zugriffskontrolle und Auditing getrennt und können auch auf unterschiedlichen Servern mit entsprechenden Datenbanken implementiert werden. Außerdem wurde die Anzahl der unterstützten Dienste und Protokolle stark erweitert. TACACS+ verwendet TCP (Port 49) als Transportprotokoll. Aus Kompatibilitätsgründen wird aber weiterhin UDP (Port 49) unterstützt.

Die Kommunikation zwischen Client und Server wird in Sessions eingeteilt. Eine TACACS+–Session ist eine Sequenz von Nachrichten für genau eine Authentisierung, zur Durchsetzung einer Zugriffskontrolle oder eine Folge von Auditing Nachrichten. Die Session wird durch eine SessionID eindeutig gekennzeichnet.
Ein TACACS+ Paket setzt sich aus Header und Body zusammen. Der Header wird dabei immer im Klartext übertragen, der Body, der die eigentlichen Nutzdaten enthält, wird verschlüsselt. Das Header Format ist in Abb. 12.2 dargestellt.

Die ersten beiden Felder enthalten die Versionsnummer des verwendeten Protokolls. Das `type` Feld gibt Auskunft darüber, ob der Body Authentisierungs–, Zugriffskontroll– oder Auditingdaten enthält. Das `seq_no` Feld enthält die Sequenznummer der PDU beginnend mit der Nummer

Bit	4	8	16	24	32
	major version	minor version	type	seq_no	flags
	session_id				
	length				

Abbildung 12.2: Format des TACACS+–Header

eins. Da TACACS+ ein strenges Request/Response Protokoll ist und immer der Client die Verbindung initiiert, sendet dieser immer die ungeraden, der Server die geraden Sequenznummern. Ist die Sequenznummer 255 erreicht und die Authentisierung, die Zugriffskontrolle oder das Accounting noch nicht abgeschlossen, muß die Verbindung beendet werden. Das `flag` Feld gibt Auskunft darüber, ob Multiplexing mehrerer TACACS+–Sessions über eine TCP Verbindung erlaubt ist bzw. ob der Body für Debuggingzwecke unverschlüsselt übertragen wird. `Length` enthält die Länge des Bodies. Das Body Format selbst ist abhängig vom jeweiligen Diensttyp.

Standardmäßig wird der gesamte Body verschlüsselt. Die Verschlüsselungsfunktion ist wie in Abb. 12.3 dargestellt definiert.

$$\text{Encrypt(data)} := \text{data } XOR \ \texttt{pseudo\_pad}$$
$$\texttt{pseudo\_pad} := MD5_1 + MD5_2 + \cdots + MD5_n \quad \text{(abgeschnitten auf die Länge von data)}$$
$$MD5_1 := MD5(\texttt{session\_id, key, sequ\_no})$$
$$MD5_2 := MD5(\texttt{session\_id, key, sequ\_no}, MD5_1)$$
$$\vdots$$
$$MD5_n := MD5(\texttt{session\_id, key, sequ\_no}, MD5_{n-1})$$

Abbildung 12.3: Verschlüsselungsfunktion von TACACS+

Die Daten werden mit dem `pseudo_pad` XOR verknüpft. Dazu muß das `pseudo_pad` dieselbe Länge wie der TACACS+–Body haben. Es wird durch Konkatenation von MD5 Hash–Werten gebildet, die nach der angegebenen Vorschrift erzeugt werden.

12.1.1 Authentisierung

Mit Hilfe der Authentisierung wird die Identität des Nutzers, der den Anruf (*Call–In*) ausführt, überprüft. Der Client sendet zu Beginn eine `START` Nachricht, die die Art der Authentisierung enthält und die bereits die Nutzerkennung und evtl. Paßwortinformationen enthalten kann, an den Server. Der Server validiert die Angaben und antwortet mit einer `REPLY` Nachricht, die anzeigt, ob die Authentisierung bereits erfolgreich oder erfolglos beendet ist, oder fortgesetzt werden muß.

Im letzten Fall beschreibt das REPLY, welche weiteren Informationen erforderlich sind oder ob auf ein anderes Authentisierungsverfahren gewechselt werden soll. Der Client schickt die angeforderten Daten in einem CONTINUE Paket, das vom Server wieder mit einem REPLY beantwortet wird. Dieser Austausch von CONTINUE-REPLY Nachrichten wird solange wiederholt, bis die Authentisierung erfolgreich oder erfolglos beendet wird oder die Sequenznummer der Session den Wert 255 erreicht hat.

Durch dieses Verfahren lassen sich leicht Challenge–Response Verfahren, One–Time Paßwörter und Authentisierungsverfahren von Drittanbietern (z.B. S/KEY [Hal95]) in TACACS+ integrieren. Die in der Spezifikation beschriebenen, möglichen Authentisierungsverfahren sind in Tabelle 12.2 zusammengefaßt.

Authentisierungsfunktion	TACACS+	RADIUS
Standard Login (z.B. UNIX)	+	+
PPP Authentisierung mittels PAP	+	+
PPP mittels PAP vom NAS aus auf entferntem Rechner	+	+
PPP Authentisierung mittels CHAP	+	+
PPP mittels CHAP vom NAS aus auf entferntem Rechner	+	+
ARAP Authentisierung	+	+
Kerberos	+	+
Paßwort gebunden an Dial–In Leitung	+	
NAS lokales Paßwort	+	+
Secure–ID		+
TACACS+	+	
RADIUS	+	+
Erweiterbar durch Drittanbieter	+	+
Funktion zur Änderung des User–Paßworts	+	
Funktion zur Änderung des ARAP–Paßworts	+	

Tabelle 12.2: Authentisierungsverfahren von TACACS+ und RADIUS

12.1.2 Zugriffskontrolle (Authorization)

Bei TACACS+ wird die Zugriffskontrolle auf entfernte Dienste als *Authorization* bezeichnet. Diese Funktion bestimmt was einem Benutzer erlaubt wird. Die Authorization beinhaltet für die angeforderten Dienste eine weitere Authentisierungsfunktion, d.h. ein Nutzer kann auch ohne den Authentisierungsdialog aus Abschnitt 12.1.1 einen Dienst aufrufen.

Der Client sendet einen REQUEST an den Server. Der REQUEST soll eine feste Anzahl von Feldern zur Authentisierung des Nutzers enthalten. Für die Authentisierung der gewünschten Dienste, mit den entsprechenden Optionen, kann der REQUEST eine variable Anzahl von Feldern beinhalten. Der Server antwortet mit einem RESPONSE, das Statusinformationen bezüglich der

Authentisierung enthält. Außerdem wird für jede Dienstanforderung aus dem `REQUEST` die Ausgabe oder eine Fehlermeldung für den Fall, daß der Benutzer nicht berechtigt ist, den Dienst auszuführen, mitgeschickt. Damit besteht die Möglichkeit, die Netzdienste abhängig vom Nutzer sehr feingranular einzuschränken. Außerdem können als Access–List bezeichnete Nutzerprofile erstellt werden, die nicht auf ein spezielles Interface beschränkt sind, sondern dynamisch an einen Eingangsport gebunden werden können. Damit lassen sich sehr komplexe Zugriffsbedingungen realisieren, z.B. kann Alice, die sich über den Port 1 eingewählt hat, nach 13:00 Uhr auf die Subnetze eins, zwei, und sieben zugreifen, Bob, der denselben Port benutzt, kann nur auf die Subnetze zwei und drei zugreifen.

12.1.3 Auditing

Mit Hilfe der Auditing Funktion kann der Client Sitzungsdaten der Verbindung beim NAS–Server in einer zentralen Datenbank speichern. Dazu sendet er ein oder mehrere `REQUEST` Paketes mit den zu loggenden Daten im Body an den Server. Der Server bestätigt die erfolgreiche Eintragung in einem `REPLY` Paket. Die Attribute der Verbindung, die gelogged werden können, sind in Tabelle 12.4 angegeben.

12.2 RADIUS

RADIUS (Remote Authentication Dial In User Service) [WRSR97, Rig97] wurde ursprünglich von der Firma Livingston entwickelt. Das Protokoll dient zur sicheren Übertragung von Authentisierungs–, Zugriffskontroll– und Auditinginformationen zwischen einem TAC (RADIUS–Client) und einem verteilten RADIUS–Server. Ein RADIUS–Server kann dabei als Proxy–Client bei anderen RADIUS–Servern oder anderen Authentisierungsservern auftreten.

Der Server erhält von den Clients Dienst– bzw. Verbindungsanforderungen der Benutzer. Er muß die Nutzer authentisieren und alle Konfigurationsinformationen, die für die Bereitstellung des Dienstes vom Client an den Benutzer erforderlich sind, an den Client zurückschicken. Die RADIUS–Kommunikation wird in Sessions eingeteilt. Eine Session wird durch einen Dienst, den der NAS einem Nutzer zur Verfügung stellt, determiniert. Die Session beginnt, wenn der Dienst dem Benutzer zur Verfügung gestellt wird und endet mit dessen Terminierung.

Als Übertragungsprotokoll verwendet RADIUS UDP (Port 1812). Es ist als Request/Response Protokoll definiert. Zwischen Client und Server werden nur die zwei Nachrichtentypen `Access` und `Accounting` definiert. `Access` dient der Authentisierung, Zugriffskontrolle und Dienstanforderung bzw. –erbringung. `Accounting` erfüllt die Auditing Funktion.

Für alle Dienste und Nachrichten wird das in Abb. 12.4 dargestellte, einheitliche Paketformat verwendet.

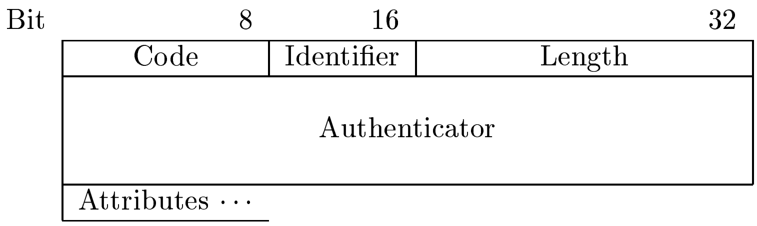

Abbildung 12.4: Format des RADIUS–Pakets

Das `Code` Feld beschreibt den Type des RADIUS–Pakets (`Access` oder `Accounting`). Der Identifier dient zur Identifizierung einer RADIUS–Session. `Length` gibt die Länge der gesamten RADIUS–PDU an. Die minimale Länge beträgt 20, die maximale 4096 Bytes. Das 16 Byte lange `Authenticator` Feld wird zur zweiseitigen Authentisierung zwischen Client und Server verwendet. Im variablen `Attributes` Feld werden die Nutzdaten übertragen.

Damit sensible Daten geschützt werden können, müssen Client und Server einen `key` als gemeinsames Geheimnis teilen.

12.2.1 Access Nachrichten

Mit Hilfe des `Access` Nachrichtentyps kann der Client für den anrufenden Nutzer Dienste beim RADIUS–Server anfordern. Dazu muß der Nutzer vom Server authentisiert werden.

Der Client schickt eine `Access-Request` Nachricht, um bestimmte Dienste anzufordern. Der `Identifier` muß dabei bei jeder Änderung des `Attributes` Feldes und beim Erhalt eines gültigen Replies geändert werden. D.h. mit jeder neuen Dienstanforderung beginnt eine neue Session. Im `Authenticator` Feld gibt der Client eine Zufallszahl an. Diese Zahl sollte während der Gültigkeitsdauer des gemeinsamen Geheimnisses (`key`) nur einmal verwendet werden. Da der `Authenticator` im Klartext übertragen wird, werden dadurch Replay Angriffe verhindert. Das `Attributes` Feld beinhaltet den angeforderten Dienst und die dafür notwendigen Optionen bzw. Attribute sowie Authentisierungsinformationen. Wird ein Paßwort mitübertragen, so wird es mit Null Bits auf ein Vielfaches von 16 Byte aufgefüllt und verschlüsselt. Dazu wird es in Blöcke p_i aufgeteilt und mit 16 Byte langen Blöcken b_i XOR verknüpft. Das Verschlüsselungsverfahren ist in Abb. 12.5 zusammengefaßt. Der String, der durch die Konkatenation von $C_1 + \cdots + C_n$ entsteht, wird übertragen.

Falls der Server die Dienstanforderung akzeptiert, schickt er ein `Access-Accept` Paket mit demselben `Identifier` und den Konfigurationsinformationen, die für die Diensterbringung benötigt werden, zurück. Im `Authenticator` Feld muß er sich gegenüber dem Client authentisieren. Dazu berechnet er folgenden Wert:
$\texttt{ResponseAuthenticator} := MD5(\texttt{Code} + \texttt{Identifier} + \texttt{Length} + \texttt{RequestAuthenticator} + \texttt{ResponseAttributes} + \texttt{key})$

$$\boxed{\begin{array}{l} C_1 := p_1 \text{ XOR } b_1 \qquad \cdots \qquad C_n := p_n \text{ XOR } b_n \text{ mit} \\ b_1 := MD5(\texttt{key} + \texttt{RequestAuthenticator}) \\ b_2 := MD5(\texttt{key} + C_1) \\ \vdots \\ b_n := MD5(\texttt{key} + C_{n-1}) \end{array}}$$

Abbildung 12.5: Paßwort Verschlüsselungsfunktion von RADIUS

Dieser MD5 Hash–Wert wird in das `Authenticator` Feld eingetragen. Durch dieses Vorgehen werden Maskeraden und Replay Angriffe verhindert, da ein Angreifer für einen erfolgreichen Angriff den `key` kennen müßte. Für den Fall, daß eines der empfangenen Attribute nicht akzeptiert werden kann, sendet der Server eine `Access-Reject` Nachricht.

Falls der Server weitere Informationen zur Authentisierung benötigt, sendet er ein `Access-Challenge` Paket, das der Client mit einem erneuten `Access-Request` beantworten muß. Auf diese Weise können Challenge–Response Authentisierungsprotokolle mit RADIUS realisiert werden.

12.2.2 Accounting Nachrichten

Falls der Client für Auditing bzw. Accounting konfiguriert wurde, so sendet er zu Beginn der Diensterbringung eine `Accounting-Start` Nachricht an den RADIUS–Server. Dieses Paket enthält Informationen über den zu erbringenden Dienst, sowie Informationen über den Benutzer. Es kann dieselben Attribute wie ein `Access-Request` Paket enthalten, darf aber kein Paßwort oder CHAP–Paßwort enthalten. Der Server sendet eine Bestätigung, falls er die Daten aus der Nachricht speichern konnte, andernfalls darf er kein `Response` zurückschicken. Nach Abschluß des Dienstes schickt der Client eine `Accounting-Stop` Nachricht mit Informationen zum erbrachten Dienst und statistischen Informationen (optional). Auch diese Nachricht wird vom Server bestätigt, falls die Daten gespeichert werden konnten.

Das Paket Format ist dasselbe wie bei `Access` Nachrichten. Allerdings ist das `Authenticator` Feld im Client Request keine Zufallszahl, sondern wird als MD5 Hash–Wert über `Code`, `Identifier`, `Length`, 16 Null Bytes, den `RequestAttributes` und dem `key` berechnet. Der Server berechnet seinen `ResponseAuthenticator` als MD5 Hash–Wert über `Code`, `Identifier`, `Length`, `AccountingRequestAuthenticator`, den `ResponseAttributes` und dem `key`.

Die Daten, die gespeichert werden können, sind in Tabelle 12.4 zusammengefaßt.

12.3 Vergleich und Bewertung von TACACS+ und RADIUS

Im folgenden werden TACACS+ und RADIUS gegenübergestellt und bewertet, um die Auswahl zwischen den beiden Protokollen zu erleichtern.

RADIUS ist als Proposed Standard (RFC) in den Standardisierungsprozeß der IETF eingebracht worden. TACACS+ wurde bisher nur als Internet Draft veröffentlicht und stellt ein proprietäres Protokoll der Firma CISCO dar.

Das von beiden unterstützte Protokoll– bzw. Dienstspektrum ist sehr ähnlich (vgl. Tabelle 12.1). Insbesondere die gängigsten Protokolle (vor allem IP und PPP), werden von beiden unterstützt. RADIUS bietet zusätzlich noch die Implementierung eines proprietären (Xylogics) IPX over SLIP Dienstes an. Da IPX aber auch über PPP betrieben werden kann, ist dies nicht als echter Vorteil anzusehen.

TACACS+ unterstützt eine größere Anzahl von Diensten, wobei Unterstützung bedeutet, daß für die angegebenen Dienste eigene Argumenttypen innerhalb des TACACS+–Bodies definiert sind. Da es sich bei den meisten Diensten aber um solche handelt, die auf TCP/IP aufsetzen, ist dieser Vorteil nur relativ. RADIUS unterstützt IP und damit können alle Dienste, die auf IP aufsetzen, auch mittels einer von RADIUS aufgebauten Verbindung betrieben werden, selbst wenn die entsprechenden Dienste nicht explizit innerhalb der Spezifikation von RADIUS aufgeführt sind.

Auch bei den Authentisierungsverfahren werden von beiden Protokollen die verbreitesten Verfahren unterstützt. Außerdem können Verfahren von Drittanbietern und Challenge–Response Verfahren bei beiden integriert werden. Beide ermöglichen auch den Zugriff auf „*Outgoing–Devices*“ des NAS, d.h. der Nutzer, der über TACACS+ oder RADIUS auf das Unternehmensnetz zugreift, kann auch auf entfernte Rechner zugreifen, die nur über eine Wählverbindung erreichbar sind.

Ein grundlegender Unterschied wird bei der prinzipiellen Philosophie der Protokolle deutlich. RADIUS erlaubt pro Session nur einen Dienst, und in der NAS Datenbank ist für jeden Benutzer genau ein Authentisierungsverfahren vorgesehen. TACACS+ erlaubt pro Nachrichten–Body mehrere Dienstanforderungen, Authentisierungsverfahren und Nutzerprofile pro Nutzer. Um bei RADIUS mehrere Authentisierungsverfahren und Nutzerprofile zu ermöglichen, müssen die Benutzernamen, z.B. durch Prä– oder Suffixe parametrisiert werden.

Bei RADIUS wird ein verteilter Server verwendet, d.h. der Client kann mehrere Server konsultieren und ein Server kann als Proxy bei einem anderen Server auftreten. TACACS+ verwendet i.d.R. nur einen Server pro Netz bzw. Subnetz. Die höhere Komplexität der Architektur von RADIUS schlägt sich auch in der Anzahl der Auditing Attribute nieder (vgl. Tabelle 12.4). In RADIUS hat der Administrator deutlich mehr Möglichkeiten, Informationen, die die Verbindung betreffen, zu speichern, da die Auditing Funktionalität von RADIUS viel umfangreicher als die von TACACS+ ist.

Funktion	TACACS+	RADIUS
Mehrere Authentisierungsfunktionen pro Nutzer	+	
Callback–Funktion	+	+
Autocommand	+	
Autoselect	+	+
System Skript	+	
Absolute Time Out	+	+
Inactivity Time Out	+	+
Port Limit pro Nutzer		+
Access List pro Nutzer	+	
Privilegierter Zugriff auf NAS		+
Gestufte Zugriffsrechte (Privilege Levels)	+	

Tabelle 12.3: Zusätzliche Funktionen von TACACS+ und RADIUS

Auch bei den zusätzlichen Funktionen ist der Unterschied erheblich (vgl. Tabelle 12.3). TACACS+ bietet die Möglichkeit in der NAS Datenbank Skripte abzulegen und diese bei der Einwahl des Benutzers automatisch ausführen zu lassen (*Autocommand*). Über die sog. *System Skript* Funktion kann ein Benutzer transparent auf ein bestimmtes Endsystem vermittelt werden. Auf diesem System können wiederum automatisch Skripte ausgeführt werden. Damit kann der Benutzer bei seiner Einwahl bspw. direkt mit seinem Arbeitsplatzrechner verbunden werden, ohne auf dem NAS eine Terminal– oder Telnet–Sitzung eröffnen zu müssen.

Beide Protokolle bieten eine *Callback–Funktion*, d.h. die Verbindung zum Anrufer wird unterbrochen und über eine beim NAS oder TAC gespeicherte Telefonnummer von seiten des TAC wieder aufgebaut. Außerdem können verschiedene Timer gesetzt werden, um die Verbindung nach einer bestimmten Zeit zu unterbrechen. Auch die, bei TACACS+ als *Autoselect* bezeichnete Funktion wird von beiden Protokollen realisiert. Sie versorgt den Nutzer mit dem Protokoll, das er braucht, um seine Verbindung aufzubauen. D.h. ein Apple–Nutzer muß keine Terminal– oder Telnet–Sitzung zum NAS aufbauen, sondern ihm wird sofort eine ARAP Verbindung zur Verfügung gestellt.

Vergleicht man die Sicherheit der beiden Dienste, so muß TACACS+ besser bewertet werden, da es die gesamten Nutzdaten verschlüsselt. RADIUS dagegen verschlüsselt nur das Paßwort des Nutzers. Insbesondere Ein– und Ausgabedaten von Diensten, die u.U. sehr sensible Informationen enthalten können, werden nicht verschlüsselt und können deshalb sehr einfach mitgelesen werden.

Bei der Verschlüsselungsfunktion selbst bestehen keine großen Unterschiede. Beide Protokolle verwenden eine XOR Verknüpfung mit Zeichenketten aus MD5 Ausgaben, die durch einen Schlüssel gesichert werden. Die kryptologische Sicherheit der Verschlüsselungsfunktion darf als gleichwertig betrachtet werden.

Keines der beiden Protokolle sichert die Integrität der übertragenen Nachrichten, d.h. eine

Veränderung der Nachricht auf dem Weg vom Client zum Server kann vom Server nicht erkannt werden.

12.4 Remote Access Service von Windows NT (RAS)

Im Betriebssystem Microsoft Windows NT kann ein als *Remote Access Service (RAS)* bezeichneter Dienst installiert werden. Damit ist es für entfernte Benutzer möglich, sich über Wählleitung in einen NT–Rechner, ein Windows– oder in ein TCP/IP–basiertes Netzwerk einzuwählen.

Auf dem NT–Rechner läuft dazu ein RAS–Server. RAS–Clients gibt es für Windows NT, Windows 95, Windows 3.11, DOS, OS/2, UNIX und Macintosh. Der RAS–Server auf Windows NT Workstation ermöglicht eine Verbindung, der Windows NT Server (ab Version 4.0) kann bis zu 256 gleichzeitige Verbindungen verwalten. Ferner ist die Server Version von NT mit einer Multilink Funktion ausgestattet, d.h. der RAS–Server kann mehrere physikalische Verbindungen zu einer logischen Verbindung mit höherer Übertragungsrate zusammenfassen. Der RAS–Server läßt auch ausgehende Verbindungen zu, damit kann ein lokal am NT–Server angemeldeter Benutzer über eine Wählverbindung auf entfernte Rechner zugreifen.

Als Verbindungsprotokoll wird PPP bzw. PPTP [HPV+96] unterstützt und darauf aufbauend können die Protokolle TCP/IP, IPX/SPX und NetBEUI verwendet werden. Der NT–Server kann auch als IPX/SPX– bzw. als TCP/IP–Router konfiguriert werden.

12.4.1 Authentisierung und Zugriffskontrolle

Zur Authentisierung der PPP–Verbindung werden die Standardverfahren *PAP* (*Password Authentication Protocol* [LS92]) und *CHAP* (*Challenge Handshake Authentication Protocol* [LS92, Sim96]) sowie das proprietäre *SPAP (Shiva Password Authentication Protocol)* unterstützt. Für die Anmeldung des Benutzers wird die Authentisierungsfunktion von Windows NT, d.h. eine Benutzerkennung mit Paßwort verwendet. Um die Sicherheit zu erhöhen, besteht die Möglichkeit, den Anrufenden vom Server zurückrufen zu lassen (Callback–Funktion). Der RAS–Server kann außerdem so konfiguriert werden, daß alle Nachrichten, die zwischen Client und Server ausgetauscht werden, mit dem RC4 Algorithmus verschlüsselt werden.

Bei der Zugriffskontrolle kann beim RAS–Server angegeben werden, ob die Clients Zugriff nur auf den Server–Rechner oder aber auf das gesamte Netzwerk erhalten sollen. Eine benutzerdefinierte Differenzierung ist hier nicht möglich. Auf dem Server–Rechner, auf dem sich der Benutzer mit seinem Paßwort anmelden muß, wird die Zugriffskontrolle über das Windows NT Betriebssystem durchgesetzt, d.h. der Benutzer hat dieselben Zugriffsrechte, die er bei einer lokalen Anmeldung an diesem Rechner haben würde.

12.4.2 Bewertung

Der Installationsaufwand für RAS ist sehr gering, falls die Kommunikationshardware und die entsprechenden Treiber bereits installiert sind. Auch die Benutzerverwaltung ist einfach und schnell durchführbar, der Betriebsaufwand ist gering. Für den RAS–Dienst fallen außer evtl. zusätzlichen Hardwarekosten keine Lizenzgebühren an.

Beim RAS–Dienst handelt es sich nicht um ein standardisiertes Protokoll, sondern um einen proprietären Dienst, der sich allerdings auf standardisierte Protokolle abstützt. Die kommerzielle Verfügbarkeit beschränkt sich mit der Firma Microsoft auf einen Anbieter.

Im wesentlichen stellt RAS einen PPP–Link zur Verfügung, auf den drei Protokolle aufgesetzt werden können. Zur Sicherung der Vertraulichkeit kann der Administrator nur RC4 verwenden. Eine Sicherung der Integrität ist nicht vorgesehen. Die Zugriffskontrolle ist zu grobgranular, da der Administrator nur den Rechner oder das gesamte Netzwerk freigeben kann und zwar nicht benutzerspezifisch sondern nur für alle Clients.

Durch die relativ große Verbreitung des Windows NT Betriebssystems, das den RAS–Dienst enthält und die einfachen Anschlußmöglichkeiten für ISDN– oder Modemhardware sowie die problemlose Installation von RAS ist es sehr einfach, „Hintertüren“ im System zu etablieren. Es gibt keinen zentralen Zugangspunkt mehr zum Unternehmensnetz, sondern im Prinzip kann jeder Mitarbeiter seinen Arbeitsplatzrechner zum Netzzugangspunkt machen, der dann natürlich nicht zwangsläufig der Kontrolle durch die Sicherheitspolitik unterliegen muß. Vielfach werden keinerlei Auditingdaten über RAS–Verbindungen erfaßt. Aus diesem Grund kann die Verwendung von RAS nicht empfohlen werden.

Accounting Daten	TACACS+	RADIUS
User–ID	+	+
Session–ID		+
Multi–Session–ID	+	+
Vom User verwendetes Protokoll	+	+
ID, um mehrere Sessions im Log zu verknüpfen		+
Anzahl der parallelen Sessions in einer Multitask Session		+
NAS–ID, NAS IP–Adresse, NAS–Port, Typ d. NAS–Port		+
Diensttyp		+
IP–Adresse, IP–Netmask		+
ID der anrufenden und der angerufenen Station		+
MTU (Maximum Transmission Unit)		+
Datenkompression		+
Login–Host, –Service und –TCP–Port		+
LAT–Service, –Node, –Group, –Port		+
AppleTalk–Link, –Network, –Zone		+
Routing Informationen		+
Callback–ID und –Nummer		+
Session–Timeout, Idle–Timeout		+
Delay–Time (wie lange hat der Client versucht, das Paket zu senden)		+
Art der Dienstterminierung		+
Grund der Dienstterminierung		+
Fehlermeldung des Dienstes	+	
Statusinformation des Dienstes	+	+
Empfangene und gesendete Bytes	+	+
Empfangene und gesendete Pakete	+	+
Sitzungsdauer	+	+
Start– und Stoppzeitpunkt des Dienstes	+	
Art der Authentisierung		+
Port–Limit		+

Tabelle 12.4: Accounting Daten von TACACS+ und RADIUS

Kapitel 13

Testplattform und Testszenarien

Um die zu untersuchenden Protokolle und Verfahren auf ihre Einsatzmöglichkeiten bei der BMW AG hin evaluieren zu können, müssen diese getestet werden. Es ist klar, daß Tests, insbesondere wenn Software verwendet wird, die sich teilweise noch im Beta–Stadium befindet, nicht in der Produktionsumgebung durchgeführt werden können. Daher wurde bei der BMW AG die in Abbildung 13.1 dargestellte Testplattform installiert, auf die sich typische im Unternehmen auftretende Szenarien und Kommunikationsbeziehungen abbilden lassen.

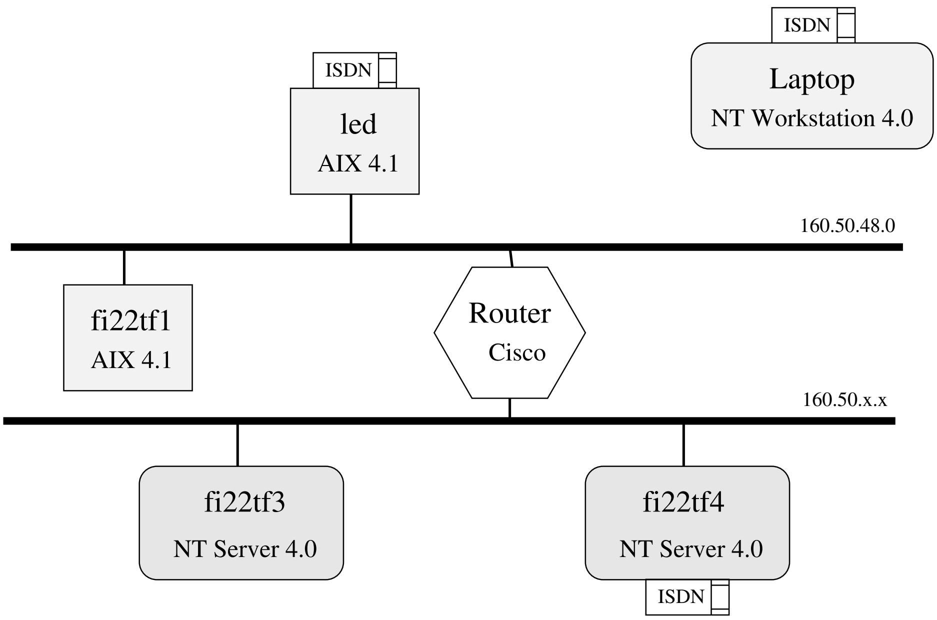

Abbildung 13.1: Testplattform bei der BMW AG

Die schattierten Rechtecke in der Abbildung bezeichnen die Endsysteme. Innerhalb der Rechtecke ist zuerst der symbolische Name und darunter das verwendete Betriebssystem angegeben. Soweit ISDN–Hardware eingebaut ist, wird dies durch ein entsprechendes Symbol gekennzeichnet.

Wichtige Entscheidungskriterien für den Einsatz von Software im BMW–Unternehmensnetz ist neben der Stabilität und dem Funktionsumfang auch der Installations–, Wartungs– und Infrastrukturaufwand für die entsprechenden Produkte. Auch darüber sollten die Testfälle Informationen liefern.

Da nicht alle Produkte zur Verfügung standen und im Rahmen dieser Arbeit auch nicht alle theoretisch betrachteten Protokolle getestet werden konnten, wurden SSL, S/MIME, SSH und RAS auf der Testplattform installiert und getestet.

13.1 SSL und S/MIME

E-Mail ist ein bei der BMW AG weitverbreitetes und zunehmend genutztes Kommunikationssystem. Ein Testszenario sollte sich daher mit der Absicherung des E-Mail–Verkehrs befassen. Es sollte signierte, verschlüsselte sowie signierte und verschlüsselte Kommunikation exemplarisch durchgeführt werden. Als Protokoll war S/MIME zu verwenden.

Da S/MIME ein hybrides Verschlüsselungsverfahren benutzt, werden für das asymmetrische Verfahren Zertifikate einer Zertifizierungsstelle benötigt. Für die BMW AG mit ihrer großen Anzahl von Endsystemen und Benutzern bietet es sich aus wirtschaftlichen Gründen an, eine eigene Zertifizierungshierarchie aufzubauen, da eine Zertifizierung aller Benutzer durch eine der kommerziellen externen Zertifizierungsstellen (z.B. Verisign) eine nicht unerhebliche finanzielle Belastung darstellen würde. Aus diesem Grund wurde eine Zertifizierungsinstanz innerhalb der Testplattform eingerichtet und getestet. Als Produkt wurde der Netscape Certification Server Version 1.01 verwendet. Da dieser Server einen funktionierenden WWW–Server sowie einen Message Transfer Agent voraussetzt, wurde auf Wunsch der BMW AG der existierende Netscape Enterprise Server verwendet und der Netscape Mail Server Version 2.0 installiert. Alle Server wurden auf dem Rechner `fi22tf1` installiert.

Da es bisher nur sehr wenige S/MIME–Implementierungen gibt, wurde hier als Software der Netscape Communicator Version 4 Beta Release 3 bzw. 4, getestet. Diese Produkte waren zum Zeitpunkt der Testdurchführung nicht für AIX, sondern nur für Windows95 bzw. Windows NT erhältlich. Die Software wurde deshalb auf den Rechnern `fi22tf3` und `fi22tf4` installiert. Die Installation ist vollkommen problemlos. Nach der Dekompression des Paketes muß lediglich das Install–Programm aufgerufen und ein Zielverzeichnis angegeben werden.

Der Netscape Certification Server (CS) wurde unter AIX eingesetzt. Auch hier muß nach Entkomprimierung ein `setup`–Skript aufgerufen werden. Im anschließenden Dialog sind Fragen zu Rechnername, Administratorkennung, Paßwörtern u.ä. anzugeben, und es können bereits

Schlüssel für die Zertifizierungsstelle (CA) und für SSL erzeugt werden [Rad97]. Die spätere Administration und Bearbeitung von Zertifikaten erfolgt über einen WWW–Browser. Um Angriffe auf den Certification Server bzw. die CA zu verhindern, wird die HTTP–Kommunikation mittels SSL gesichert, d.h. es muß ein SSL–fähiger WWW–Server sowie ein SSL–fähiger Browser im Netz vorhanden sein. Damit der CS IP–Adressen und Rechnernamen auflösen kann, muß ein Resolver bzw. ein Nameserver installiert sein. Der CS setzt zur Speicherung der Zertifikate eine Informix Datenbank ein, die mit dem Server installiert wird. Die verwendete Version des CS ist in Bezug auf die Datenbankdokumentation mangelhaft und das Installationsskript ist nicht vollständig. Nach einem reboot des Rechners, auf dem der CS läuft, läßt sich die Datenbank und damit auch der CS nicht mehr starten. Es ist notwendig, nach jedem reboot im Verzeichnis, in dem die Datenbank installiert wurde, den Befehl `make dev` aufzurufen. Dieser Befehl ist weder dokumentiert noch wird er vom Installationsskript in eine der `rc.`*x* Dateien eingetragen, die nach jedem booten automatisch ausgeführt werden.

Da die Administratorkennungen und alle privaten Schlüssel mit Paßwörtern gesichert sind, müssen beim Start des Servers drei Paßwörter, für den privaten CA–Schlüssel, für den privaten SSL–Schlüssel sowie für den Datenbankadministrator eingegeben werden. Wird der CS administriert oder werden Zertifikate bearbeitet, so ist im WWW–Browser die Server–Kennung und ein weiteres Paßwort einzugeben. Es ist klar, daß diese Paßwörter verschieden sein sollten. Durch die Trennung der CS und der Datenbank–Kennung sowie die getrennten Paßwörter für SSL– und CA–Schlüssel ließe sich einfach ein *Vier–Augen–Prinzip* für den CS durchsetzen, d.h. von zwei Administratoren kennt jeder zwei der vier benötigten Paßwörter, und der CS kann nur gestartet werden, wenn beide ihre Paßwörter gemeinsam eingeben.

Auch die Installation des Netscape Mail Servers erfolgt problemlos über ein `setup`–Skript. Auch hier wird über einen WWW–Browser, gesichert über SSL, administriert. Für jeden Nutzer, der Mail Dienste wahrnehmen will, muß auf dem Mail Server eine Kennung mit Paßwort eingerichtet werden. Als Protokolle sind SMTP [Pos82], POP3 [MR94] und IMAP4 [Cri94] realisiert.

Nach der Installation der benötigten Server konnte S/MIME getestet werden. Dies sollte ursprünglich mit dem Netscape Communicator Version 4 Beta Release 3 erfolgen. Bei dieser Version können zwar Zertifikate, die von einer eigenen CA signiert wurden, geladen werden. Für die Signatur oder Verschlüsselung von Mails werden aber nur Zertifikate akzeptiert, deren Zertifizierungspfad ein Zertifikat einer offiziellen CA enthält. Eine offizielle CA in diesem Sinne ist eine der CA's, deren öffentlicher Schlüssel fest im Communicator „eingebaut" ist, wie bspw. Verisign[1]. Eine Forderung, die an die Tests gestellt wurde, war aber der exemplarische Aufbau einer eigenen autarken CA. Der Mangel des Beta Release 3 wurde von Release 4 behoben, mit der auch eigene Zertifizierungsstellen bzw. deren Zertifikate in den Browser integriert werden können.

Will Alice S/MIME verwenden, muß sie bei der CA ein Zertifikat beantragen. Dazu muß sie mit Hilfe ihres WWW–Browsers (im Testfall der Netscape Navigator als Teil des Communicators) beim unternehmenseigenen CS ein Formular für den Certification Request ausfüllen. Ihr

[1] http://www.verisign.com

Browser generiert dabei ein Schlüsselpaar und überträgt den öffentlichen Schlüssel sowie den Distinguished Name (DN, vgl. Abschnitt 6.2) an den CS.

Der Administrator des CS findet diese Zertifikatsanforderung bei den zu bearbeitenden Certification Requests. Er muß die Angaben überprüfen, eine Gültigkeitsdauer für das Zertifikat festlegen und die Anforderung signieren. Bei den Tests wurde ein weiterer Bug des CS gefunden. Wird der Certification Request ohne Veränderung signiert, so kann der Nutzer das Zertifikat nicht für S/MIME–Dienste verwenden. Im DN des Zertifikates muß ein Verwendungszweck angegeben werden. Im Formular für den Certification Request ist aber keine entsprechende Eintrags– oder Auswahlmöglichkeit vorhanden. Deshalb muß der Administrator den DN des Certification Requests um das Attribut `E=<email-Adresse>` erweitern, das besagt, daß das Zertifikat für E-Mail Kommunikation verwendet werden kann. Nachdem er den so veränderten Certification Request signiert hat, muß der Administrator nur noch einen Button drücken, um damit eine E-Mail an Alice zu schicken. Diese Nachricht enthält einen `https`–Link, mit dessen Hilfe Alice ihr Zertifikat in ihren Communicator integrieren kann.

Damit Alice verschlüsselt mit Bob kommunizieren kann, braucht sie dessen öffentlichen Schlüssel und Bob benötigt seinerseits den öffentlichen Schlüssel von Alice. Alice kann an Bob eine signierte E-Mail schicken. Sie muß dazu nur einen entsprechenden Checkbutton des Messengers (E-Mail UA des Communicators) anklicken. Mit der signierten Nachricht wird das Zertifikat von Alice an Bob übertragen. Die Antwort an Alice kann bereits verschlüsselt erfolgen, da Bob den öffentlichen Schlüssel von Alice nun kennt. Damit auch Alice das Zertifikat von Bob bekommt, sollte dieser die Rückantwort auch signieren, soweit er bereits ein eigenes Zertifikat besitzt. Sobald Alice die verschlüsselte und signierte Nachricht empfangen hat, kann in beide Richtungen gesichert kommuniziert werden.

Der Messenger zeigt eine signierte, verschlüsselte oder signierte und verschlüsselte Nachricht immer im Klartext mit einem entsprechenden Symbol (Signed Message, Encrypted Message bzw. Signed and Encrypted Message) an. Die Signatur der Nachricht wird bei der Abholung der E-Mail für den Benutzer transparent geprüft und das Ergebnis der Prüfung (gültige oder ungültige Signatur) im entsprechenden Symbol angezeigt. Um eine verschlüsselte Nachricht zu lesen, muß der Benutzer sein Paßwort für seinen privaten Schlüssel eingeben. Die Nachricht erscheint dann sofort im Klartext. Klickt der Benutzer auf das Symbol, das die Art der S/MIME–Nachricht anzeigt, so erhält er zusätzliche Informationen über das Zertifikat des Absenders.

13.2 SSH

Im Unternehmensnetz der BMW AG gibt es viele Rechner mit UNIX als Betriebssystem. Diese Rechner bzw. die darauf installierten Dienste werden auch über das Unternehmensnetz genutzt. Hierzu werden entweder Telnet–, rlogin– oder sonstige Terminalverbindungen verwendet. Da bei diesen Diensten die Daten im Klartext auf dem Netz übertragen werden, stellt die Administration eines UNIX–Rechners über das Netzwerk ein erhebliches Sicherheitsrisiko dar. Viele

Mitarbeiter, die UNIX–Systeme administrieren, verwenden PC's mit Windows als Arbeitsplatzrechner. Ein Produkt zur Absicherung der entfernten Befehlsausführung muß also mindestens diese beiden Betriebssysteme abdecken.

Im Testszenario für SSH wurde deshalb der entfernte Zugriff und die entfernte Befehlsausführung auf UNIX–Rechner, von UNIX– und Windows–Rechnern aus, getestet. Zusätzlich wurden die Umleitung der X11–Grafikausgabe und verschiedene Authentisierungsverfahren getestet.

Zur Installation von SSH unter UNIX ist ein C–Compiler notwendig, da die Quelltexte auf der jeweiligen Maschinenarchitektur übersetzt werden müssen. Dazu muß ein `configure` Skript aufgerufen werden, das die Makefiles entsprechend der Hardwarearchitektur automatisch anpaßt. Mit einem Aufruf von `make` werden die ausführbaren Programme für SSH–Server (`sshd`) und SSH–Client (`ssh`) sowie einige weitere Dienstprogramme (z.B. zur Schlüsselgenerierung) erzeugt. Mit `make install` werden die Programme, sowie die Manual–Pages installiert und der host_key für den Rechner erzeugt. Die Installation unter AIX ist problemlos durchzuführen, es ist lediglich darauf zu achten, daß genügend Platz auf dem entsprechenden Dateisystem vorhanden ist, da die Object–Dateien während des Compiler–Laufs auf über 100 MB anwachsen. Die Übersetzung und Installation auf `led` dauert rund 20 Minuten (IBM RS6000 /570 mit 128 MB Hauptspeicher).

SSH–Server und SSH–Client, Version 1.2.20, wurden auf `led` installiert. Da auf dem Rechner `fi22tf1` kein Compiler zur Verfügung stand, wurden die ausführbaren Programme von `led` kopiert und die host_keys mit Hilfe der entsprechenden Dienstprogramme erzeugt. Für die Windows–Rechner `fi22tf3` und `fi22tf4` wurde eine Testversion von F–Secure[2], einem kommerziellen SSH–Client für Windows 95 und NT, installiert. Auch hier erfolgte die Installation und die Schlüsselgenerierung, nach dem Aufruf des entsprechenden Install–Programmes, problemlos.

Auf den UNIX–Rechnern, zu denen eine Verbindung aufgebaut werden soll, muß der SSH–Server laufen. Im Testszenario wurde `sshd` auf `led` und `fi22tf1` gestartet. Wird der SSH–Client unter Windows gestartet, so muß zuerst der Rechner angegeben werden, zu dem die Verbindung aufgebaut werden soll. Im Anschluß daran kann der Benutzer zwischen Authentisierung mittels Paßwort oder den anderen unterstützten Authentisierungsverfahren wählen. Nach dem Verbindungsaufbau und erfolgreicher Authentisierung erhält der Benutzer ein Fenster mit einer Terminalemulation und einer Shell. Er kann damit alle Befehle auf dem UNIX–Rechner ausführen, die keine Grafikausgabe benötigen.

Wird der SSH–Client auf einem UNIX–Rechner mit X11–Window–System mittels `ssh <rechnername>` gestartet und verläuft die Authentisierung erfolgreich, erhält der Benutzer ebenfalls einen Shell–Prompt. Das SSH–Protokoll sorgt in diesem Fall auch für eine, für den Benutzer transparente, Umlenkung der X11–Grafikausgabe, d.h. der Benutzer kann alle Programme auf dem entfernten Rechner ausführen.

Im Testszenario wurden verschiedene Verbindungen zwischen den Testrechnern aufgebaut und die wichtigsten Authentisierungsverfahren getestet. Um die Authentisierung mittels `.rhosts`

[2]http://www.datafellows.com/f-secure

```
# sshd -d
debug: sshd version 1.2.20 [rs6000-ibm-aix4.1.4.0]
debug: Initializing random number generator; seed file /etc/ssh_random_seed
log: Server listening on port 22.
log: Generating 768 bit RSA key.
Generating p:  ...++ (distance 8)
Generating q:  ....................................++ (distance 472)
Computing the keys...
Testing the keys...
Key generation complete.
log: RSA key generation complete.
debug: Server will not fork when running in debugging mode.
log: Connection from 160.50.48.160 port 1023
debug: Client protocol version 1.5; client software version 1.2.20
debug: Sent 768 bit public key and 1024 bit host key.
debug: Encryption type: idea
debug: Received session key; encryption turned on.
debug: Attempting authentication for root.
debug: Trying rhosts with RSA host authentication for root
debug: Rhosts RSA authentication: canonical host fi22tf1.muc
debug: Rhosts with RSA host authentication denied: unknown or invalid host key
debug: RhostsRSA authentication failed for 'root', remote 'root', host 'fi22tf1.muc'.
log: RSA authentication for root accepted.
log: ROOT LOGIN as 'root' from fi22tf1.muc
debug: Allocating pty.
debug: Received request for X11 forwarding with auth spoofing.
debug: Allocated channel 0 of type 1.
debug: Forking shell.
debug: Entering interactive session.
```

Abbildung 13.2: Ausgabe des SSH–Servers im Debug–Modus

zu erlauben, muß dies in der Konfigurationsdatei `/etc/sshd_config` eingestellt werden. Standardmäßig ist die Authentisierung mit `.rhosts` nicht erlaubt. Startet der Benutzer, der sich mittels `.rhosts` oder `/etc/hosts.equiv` authentisieren kann, den SSH–Client, erhält er sofort, ohne weitere Eingaben, den Shell–Prompt.

Auch bei der Host–basierten Authentisierung muß der Benutzer nach dem Start des Client keine weiteren Eingaben tätigen und erhält sofort den Prompt. Dieses Verfahren ist aber viel sicherer als die Authentisierung mittels `.rhosts`, da der Client–Rechner über sein asymmetrisches Schlüsselpaar authentisiert wird und dadurch IP–Spoofing Angriffe verhindert werden (vgl. Abschn. 2.2.1 und 9.2.2).

Bei der Authentisierung mittels Paßwort muß der Benutzer sein User–Paßwort auf dem entfernten Rechner beim SSH–Client eingeben. Da SSH–Client und SSH–Server zu diesem Zeitpunkt bereits einen Kommunikationsschlüssel vereinbart haben, wird das Paßwort verschlüsselt übertragen.

Zur Verdeutlichung der Public Key Authentisierung, wurden der Server auf `led` (vgl. Abb. 13.2) und der Client auf `fi22tf1` (vgl. Abb. 13.3) im Debug–Modus gestartet.

```
1  # ssh -v led
2  SSH Version 1.2.20 [rs6000-ibm-aix4.1.4.0], protocol version 1.5.
3  fi22tf1: Reading configuration data /etc/ssh_config
4  fi22tf1: ssh_connect: getuid 0 geteuid 0 anon 0
5  fi22tf1: Connecting to led [160.50.48.166] port 22.
6  fi22tf1: Allocated local port 1023.
7  fi22tf1: Connection established.
8  fi22tf1: Remote protocol version 1.5, remote software version 1.2.20
9  fi22tf1: Waiting for server public key.
10 fi22tf1: Received server public key (768 bits) and host key (1024 bits).
11 fi22tf1: Host 'led' is known and matches the host key.
12 fi22tf1: Initializing random; seed file /home/root/.ssh/random_seed
13 fi22tf1: Encryption type: idea
14 fi22tf1: Sent encrypted session key.
15 fi22tf1: Received encrypted confirmation.
16 fi22tf1: Trying rhosts or /etc/hosts.equiv with RSA host authentication.
17 fi22tf1: Remote: Accepted by .rhosts.
18 fi22tf1: Remote: Your host key cannot be verified: unknown or invalid host key.
19 fi22tf1: Remote: The host name used to check the key was 'fi22tf1.muc'.
20 fi22tf1: Server refused our rhosts authentication or host key.
21 fi22tf1: Trying RSA authentication with key 'root@fi22tf1'
22 fi22tf1: Received RSA challenge from server.
23 Enter passphrase for RSA key 'root@fi22tf1':
24 fi22tf1: Sending response to host key RSA challenge.
25 fi22tf1: Remote: RSA authentication accepted.
26 fi22tf1: RSA authentication accepted by server.
27 fi22tf1: Requesting pty.
28 fi22tf1: Requesting X11 forwarding with authentication spoofing.
29 fi22tf1: Requesting shell.
30 fi22tf1: Entering interactive session.
31 Last login: Wed Jun  4 08:10:49 1997 from samba.muc
32 ******************************************************************************
33 *                                                                            *
34 *  Welcome to AIX Version 4.1!                                               *
35 *                                                                            *
36 ******************************************************************************
37 Running xauth add led:10.0 MIT-MAGIC-COOKIE-1 58a5ea488f71cecb06ff68d0f2b4f280
38 Running xauth add 160.50.48.166:10.0 MIT-MAGIC-COOKIE-1 58a5ea488f71cecb06ff68d0f2b4f280
39 [led][root]:/home/led>>
```

Abbildung 13.3: Ausgabe des SSH–Clients im Debug–Modus

Nachdem der Server gestartet wurde, erzeugt er seine server_keys (Abb. 13.2 Zeile 2–11) und wartet auf eingehende Verbindungen. Der Client baut die Verbindung zum Server auf (Abb. 13.3 Zeile 2–7), dann werden die Protokollversionen ausgetauscht, der öffentliche server_ und host_key an den Client verschickt und die Algorithmen sowie die entsprechenden Schlüssel ausgehandelt (Abb. 13.3 Zeile 8–11, Abb. 13.2 Zeile 14–17).

In der nun folgenden Authentisierungsphase versucht der Client die Host–basierte Authentisierung durchzuführen (Abb. 13.3 Zeile 16–20). Der Server lehnt dies ab, da er den host_key von `fi22tf1` nicht kennt (Abb. 13.3 Zeile 20). Der Client versucht nun, seinen Benutzer root mit dem Public Key Verfahren RSA zu authentisieren (Abb. 13.3 Zeile 21). Dazu schickt er den öffentlichen Schlüssel von root an den Server. Um aber den Besitz des entsprechenden privaten Schlüssels beweisen zu können, muß eine digitale Signatur erstellt werden (vgl. Abschn. 9.2.3). Der Server fordert diese Signatur an (Abb. 13.3 Zeile 22). Der Benutzer root muß nun das Paßwort für seinen privaten Schlüssel eingeben (Abb. 13.3 Zeile 23), der Client berechnet die Signatur und schickt diese an den Server (Abb. 13.3 Zeile 24), der daraufhin die Authentisierung akzeptiert (Abb. 13.2 Zeile 23). Der Client fordert dann ein Terminal, die Umlenkung der X11–Grafikausgabe und eine Shell beim Server an (Abb. 13.3 Zeile 27–29). Nachdem der Server das Terminal zur Verfügung stellt, beginnt die interaktive Sitzung. In Abb. 13.3 Zeilen 37, 38 wird das fake X11–Display angelegt und der Benutzer erhält anschließend den Eingabeprompt.

13.3 RAS

Das Testszenario für RAS bildet den Anschluß eines Telearbeitsplatzes an das Unternehmensnetz auf die Testplattform ab. Der RAS–Server wurde dazu auf dem Rechner `fi22tf3` unter Windows NT Server, der Client auf einem Laptop unter Windows NT Workstation installiert.

Die Installation der beiden RAS–Programme ist problemlos und schnell durchführbar, soweit die entsprechenden Treiber für die ISDN–Karten installiert sind. Allerdings waren diese auf den Testrechnern nicht installiert. RAS beendet in diesem Fall die Installation mit der Meldung „Keine RAS–fähigen Geräte vorhanden“. Die Installation von ISDN–Treibern ist ein aufwendiger Prozeß, da die wenigen Fehlermeldungen von NT ungenügend sind und verschiedenste Parameter ausprobiert werden mußten. In der Regel ist bei jeder Änderung der Treiberkonfiguration ein Neustart von NT nötig.

Nachdem Treiber und RAS Software installiert waren, sollte eine Verbindung vom Client zum Server aufgebaut werden. Dazu wurde die ISDN–Karte des NT–Servers direkt angewählt. Die PPP–Authentisierung und die Anmeldung des Benutzers konnten erfolgreich durchgeführt werden. Allerdings war es auf der Seite des Clients nicht möglich, auf Laufwerke des Servers zuzugreifen, da der Server auf entsprechende Anfragen nicht reagierte. Daher wurde eine neue Verbindung vom Server zum Client aufgebaut, um dort Laufwerke freizugeben.

Mit RAS kann auf entfernte Ressourcen wie Dateien und Drucker zugegriffen werden. Falls ausführbare Programme auf dem entfernten Rechner ausgewählt werden, so werden diese auf den lokalen Rechner übertragen und dort ausgeführt, d.h. der Nutzer kann Software auf dem lokalen Rechner nutzen, ohne sie installiert zu haben.

Auch auf der Testplattform wurde klar, daß ein RAS–Zugang dazu herausfordert, zentrale Systeme zu umgehen. Der Laptop wählte direkt den NT–Server, ohne über das ISDN–Net–Gateway auf dem Remote Access Server `led` zu gehen. Es fallen daher keine Auditingdaten auf `led` an. Die Verbindungsdaten werden nur auf dem NT–Server sichtbar. Unter der Annahme, daß der NT–Rechner lokal verwaltet wird, kann dadurch ein Zugang zum System geschaffen werden, der völlig unbemerkt bleibt.

Kapitel 14

Anwendungsszenario im Corporate Network der BMW AG

In den bisherigen Abschnitten wurden verschiedene Protokolle und Produkte, die zur Realisierung einer sicheren TCP/IP–basierten Kommunikation verwendet werden, vorgestellt, bewertet und verglichen. In diesem Kapitel wird nun ein Anwendungsszenario im Unternehmensnetz der BMW AG beschrieben.

Von besonderem Interesse ist dabei die Frage, welche der Protokolle oder Produkte im Umfeld der BMW AG tatsächlich eingesetzt werden sollen. Im Hinblick darauf sind Managementgesichtspunkte zu klären, z.B. die Fragen wie eine Migration auf die neuen Dienste zu erfolgen hat, welche Auswirkungen und Abhängigkeiten sich auf Betrieb und Wartung ergeben bzw. welche Betriebskonzepte für die neuen Dienste erforderlich werden. Auch organisatorische Aspekte müssen betrachtet werden, und hier ist insbesondere zu klären, welche Vorgehensweisen den Anwendern, durch die Sicherheitspolitik oder auf administrativem Wege, vorgeschrieben werden müssen.

Der praktische Einsatz von Tunneling mittels AH und ESP (vgl. Abschn. 7.1) kann in diesem Kapitel nicht behandelt werden. Einerseits wird bei der BMW AG bisher IPv6 nicht verwendet und andererseits hat der Status von Produkten, die diese Protokolle auch implementieren, bisher nur experimentellen Charakter erreicht.

In den folgenden Abschnitten wird davon ausgegangen, daß Schlüssel bzw. Zertifikate vorhanden sind bzw. sicher ausgetauscht werden können. Mit der Problematik des Schlüssel– und Zertifikatsmanagements befassen sich Abschnitt 6.2 und 14.5.

14.1 Entfernte Sitzungen

Die entfernte Sitzung auf UNIX–Rechnern mit Hilfe von Telnet, Terminalemulationen oder den r–Diensten ist unter Sicherheitsaspekten problematisch, da alle Daten zwischen den Rechnern im Klartext übertragen werden. Da diese Dienste häufig verwendet werden, um entfernte Systeme zu administrieren, sind sie lohnende Ziele für potentielle Angreifer. Eine Absicherung dieser Dienste hat daher höchste Priorität.

Im Unternehmensnetz von der BMW AG wird empfohlen, SSH (vgl. Abschnitt 9) zu verwenden, um diese Dienste zu sichern bzw. zu ersetzen. SSH kann leicht an vielfältige Benutzeranforderungen angepaßt werden, es unterstützt eine große Anzahl von Algorithmen und verschiedene Authentisierungsverfahren. Die kryptologische Sicherheit kann ebenfalls den Anforderungen entsprechend angepaßt werden. Für SSH gibt es keine Begrenzung der Schlüssellängen, durch die das Protokoll geschwächt werden würde. Auch der periodische Schlüsselwechsel des Servers erhöht die kryptologische Sicherheit. Jeder SSH–Client kann zu jedem Zeitpunkt innerhalb einer Kommunikation eine neue Schlüsselaustauschphase verlangen. Die transparente und sichere Umlenkung der X11–Grafikausgabe bietet einen weiteren nicht zu unterschätzenden Vorteil von SSH. Durch die Vorgehensweise bei der Umlenkung der Grafikausgabe werden einige Sicherheitsrisiken des X11–Systems vermieden.

14.1.1 Installation und Migration

SSH sollte auf allen UNIX–Systemen installiert werden. Es ist darauf zu achten, daß der SSH–Server (`sshd`) nach jedem reboot automatisch gestartet wird, z.B. durch einen entsprechenden Eintrag in einer `/etc/rc.`x–Datei. Auf allen Windows–Rechnern, von denen auf UNIX–Systeme zugegriffen wird, muß ein SSH–Client installiert werden. Die Installation von SSH ist einfach durchführbar.

Auf UNIX–Rechnern muß SSH aus den Quelltexten übersetzt werden. Dies ist ein relativ zeitaufwendiger Prozeß, der bei der großen Zahl von Systemen erheblichen Aufwand bedeutet. Um diesen Aufwand zu verringern, kann die Übersetzung von SSH einmal pro Maschinenarchitektur erfolgen. Die ausführbaren Programme sowie die Konfigurationsdateien müssen dann auf das Zielsystem kopiert und die Schlüssel auf dem entsprechenden Rechner erzeugt werden. Dieser Prozeß kann durch zu erstellende Shell–Skripten zum großen Teil automatisch durchgeführt werden.

Nach der Installation von SSH wird empfohlen, die r–Dienste nicht sofort zu löschen, sondern umzubenennen und dann durch SSH zu ersetzen. Erst nach einer längeren Testphase im praktischen Einsatz können die r–Dienste von den Systemen entfernt werden.

14.1.2 Betriebsaspekte und Politik

SSH bietet die Möglichkeit, die Authentisierung mittels `.rhosts` durchzuführen (vgl. Abschnitt 9.2.1). Da diese Option ein erhebliches Sicherheitsrisiko bedeutet, ist darauf zu achten, daß in den Konfigurationsdateien die Authentisierung mittels `.rhosts` abgeschaltet wird. Zudem sollte die Sicherheitspolitik den Anwendern diese Art der Authentisierung untersagen.

Bei der Schlüssellänge für den server_key ist eine Länge von mindestens 768 Bit und ein Intervall für den periodischen Wechsel von maximal einer Stunde anzustreben. Beim host_key und den Schlüsseln für die Benutzer empfiehlt sich mindestens 1024 Bit lange Schlüssel zu verwenden. Abhängig von der Schlüssellänge ist auch die Lebensdauer des host_key festzulegen. Nach der Installation bedarf es keiner Wartung für SSH.

14.2 WWW–basierte Kommunikation

Zur Sicherung der WWW–basierten Kommunikation ist aus S-HTTP, HTTPS und SSH (vgl. Abschn. 9 und 10) das für die BMW AG geeigneteste Protokoll auszuwählen. Die Sicherheitsanforderungen Integrität, Vertraulichkeit und Authentisierung lassen sich mit allen drei Protokollen durchsetzen. Mit S-HTTP kann zusätzlich die Verbindlichkeit gewährleistet werden. Da die Verbreitung von S-HTTP sehr beschränkt ist und nur von wenigen Server–Herstellern unterstützt wird, kann eine Verwendung im BMW–Unternehmensnetz jedoch nicht empfohlen werden.

Alle drei Protokolle sind bisher nicht standardisiert und lediglich als Internet–Drafts spezifiziert. Da insbesondere S-HTTP und HTTPS dasselbe Dienstspektrum abdecken, kann davon ausgegangen werden, daß nur eines dieser beiden Protokolle den Standardisierungsprozeß durchlaufen bzw. sich im Markt durchsetzen wird. Zum momentanen Zeitpunkt scheint dies HTTPS und nicht S–HTTP zu sein.

Ein weiterer Vorteil von HTTPS gegenüber S-HTTP ist die architekturelle Einordnung von SSL, das direkt auf TCP aufsetzt. Bei S-HTTP handelt es sich um ein Protokoll der Anwendungsschicht und daher läßt sich damit nur der HTTP–Verkehr absichern. Über SSL können, bei entsprechender Client–Software, auch andere Protokolle wie z.B. ftp oder Telnet abgesichert werden.

Auch die Verwendung von SSH zur Sicherung des WWW–Verkehrs ist noch nicht sehr weit verbreitet. Außerdem gibt es bei SSH Probleme bei der Kommunikation über einen Firewall (vgl. Bemerkung auf S. 71).

HTTPS hat die größte Verbreitung und die breiteste Unterstützung der HTTP–Server und –Client Software. Es wird zudem eine große Zahl von Verschlüsselungs– und MAC–Algorithmen unterstützt. Sind entsprechende Zertifikate vorhanden und in den Clients integriert, ist die Verwendung von HTTPS für den Benutzer nahezu transparent, i.d.R. muß er keine Kenntnisse über

SSL besitzen. Es besteht die Möglichkeit einer zweiseitigen Authentisierung mit HTTPS, allerdings kann eine verschlüsselte Kommunikation auch ohne Authentisierung des Client erfolgen. Dies ist insbesondere für externe HTTP–Anfragen, z.B. von Kunden wichtig, die kein Zertifikat besitzen, aber trotzdem vertrauliche Daten an den Server übermitteln wollen.
Aus diesen Gründen wird der grundsätzliche Einsatz von HTTPS empfohlen.

Da jedoch alle derzeit kommerziell verfügbaren SSL–Implementierungen aus den Vereinigten Staaten kommen, greift das Exportverbot für Produkte, die starke Kryptologie realisieren und damit die Beschränkung der Schlüssellängen (vgl. Abschnitt 8.3 und Bemerkung S. 53). Symmetrische Schlüssel mit einer Länge von 40 Bit können nur als bedingt sicher betrachtet werden, daher kann der Einsatz von SSL bzw. HTTPS in hochsensiblen Bereichen nicht empfohlen werden.

Unter der Annahme, daß eine HTTP–Kommunikation zwischen hochsensiblen Bereichen nur innerhalb des Unternehmensnetzes bzw. nur zwischen wenigen wohlbekannten Partnern stattfindet, können derartige Verbindungen auch über SSH abgesichert werden. Bei SSH sind die Schlüssellängen nicht beschränkt und können frei gewählt werden.

Zur Sicherung der WWW–basierten Kommunikation ist also je nach Sicherheitsstufe, HTTPS oder SSH, zu verwenden.

14.2.1 Installation und Migration

Um HTTPS verwenden zu können, müssen alle HTTP–Server und –Clients, die SSL nicht unterstützen, durch SSL–fähige Produkte ersetzt werden. Die SSL– bzw. HTTPS–fähigen Server unterstützen auch normalen HTTP–Verkehr, d.h. bestehende Server können unter Beibehaltung des Datenbestandes gegen HTTPS–fähige Server ausgetauscht werden. Der bestehende Datenbestand kann weiterhin mit HTTP genutzt werden und es ist nicht notwendig, zwei Server parallel zu betreiben und zu pflegen.

Die Installation eines HTTPS–Servers ist einfach durchzuführen. Allerdings ist die Konfiguration eines solchen Servers ein kritischer Prozeß. Denn durch eine fehlerhafte Konfiguration kann die Sicherheit des Servers erheblich geschwächt werden. Soll HTTPS–Verkehr auch über einen Firewall abgewickelt werden, so muß auf diesem der Verkehr über Port 443 freigegeben werden.

Für Bereiche mit erhöhten Sicherheitsanforderungen müssen auf Client– und Serverseite SSH sowie entsprechende Proxies installiert werden, um den HTTP–Verkehr über SSH abzuwickeln.

14.2.2 Betriebsaspekte

Nach der Installation der Software müssen die zu schützenden Ressourcen in Form von HTTP–gestützten Anwendungen oder normalen HTML–Dokumenten ermittelt werden. Außerdem ist für jede Ressource festzulegen, ob sich der anfordernde Client beim Server authentisieren muß.

Danach müssen die Zugriffspfade auf diese Ressourcen so geändert werden, daß nur noch über HTTPS darauf zugegriffen werden kann und der Client sich ggf. authentisieren muß. Dazu muß ein Betriebskonzept für den unternehmensinternen, ebenso wie für externen HTTP–Verkehr aufgestellt werden. Insbesondere im externen Verkehr macht es wenig Sinn, allgemeinzugängliche Informationen wie bspw. Produktinformationen über HTTPS zu sichern. Soll dem externen Nutzer die Möglichkeit gegeben werden, eigene, insbesondere personenbezogene Daten, an den Server zu senden, so muß dieser Verkehr verschlüsselt werden.

Zur Erhöhung der kryptologischen Sicherheit sind lange Schlüssel essentiell. Ihre Verwendung ist aber nur dann möglich, wenn die US–Exportbeschränkungen gelockert werden oder die HTTPS– bzw. SSL–Software außerhalb der Vereinigten Staaten, z.B. in Europa, entwickelt wird. Auf diese Aspekte hin sollte der Markt beobachtet werden, um ggf. auf ein sicheres Produkt wechseln zu können.

14.3 E–Mail

Zur Sicherung von Elektronischer Post wurden PEM, S/MIME und PGP untersucht. Die gestellten Sicherheitsanforderungen werden von allen drei Diensten erfüllt. Obwohl PEM ein durchdachtes Konzept darstellt, hat es bisher keine marktrelevante Verbreitung gefunden. Es verwendet einen 7–Bit Zeichensatz und unterstützt MIME nicht. Aus diesen Gründen kann PEM für den praktischen Einsatz bei der BMW AG nicht empfohlen werden.

S/MIME ist ein Protokoll, das auf MIME aufsetzt, d.h. es können alle MIME–Nachrichten und insbesondere Nachrichten im 8–Bit Zeichensatz mit S/MIME übertragen werden. Die Integration von S/MIME in den Netscape Communicator stellt eine sehr benutzerfreundliche Anwendung dar. Allerdings gibt es bisher nur wenige UA's die S/MIME unterstützen. Alle diese Produkte werden in den USA hergestellt und damit gelten auch hier die Exportbeschränkungen und die Beschränkung der Schlüssellängen. Die kryptologische Sicherheit von S/MIME ist unter diesen Bedingungen nicht sehr hoch.

Im Gegensatz dazu ist die Schlüssellänge bei PGP nicht beschränkt und für asymmetrische Verfahren können Schlüssel bis 1024 Bit verwendet werden. PGP kann über zwei neue `Content-Type`s in MIME integriert werden, d.h. es ist mit PGP auch möglich, einen 8–Bit Zeichensatz zu verwenden oder andere MIME–Typen zu übertragen.

Für „normalen“ E-Mail–Verkehr kann sowohl S/MIME als auch PGP empfohlen werden. Bei einer Kommunikation, die höchster Sicherheitsanforderungen bedarf, muß S/MIME mit beschränkter Schlüssellänge als nicht ausreichend sicher angesehen werden, in diesen Bereichen kann daher nur PGP Verwendung finden.

Je nach Anforderung an die Höhe der kryptologischen Sicherheit ist also S/MIME oder PGP, zu verwenden. Im folgenden wird daher sowohl auf Migration und Installation von S/MIME als auch von PGP eingegangen.

14.3.1 Installation und Migration

Zur Verwendung von S/MIME, müssen nahezu alle UA's ausgetauscht werden, da S/MIME von bestehenden UA's i.d.R. nicht unterstützt wird und auch nachträglich nicht in diese integriert werden kann. Allerdings steht mit dem Communicator von Netscape eine Implementierung zur Verfügung, die S/MIME in einem SSL– und HTTPS–fähigen WWW–Browser integriert. Sollte dieses Produkt auch als HTTPS–Client im Unternehmen eingesetzt werden, würden sich bei der Installation erhebliche Synergieeffekte ergeben. In diesem Fall müßte im Rahmen der Umstellung auf HTTPS nur der Communicator als HTTPS–Client und auch als S/MIME–UA im Unternehmensnetz installiert werden. Auch eine für HTTPS eingerichtete Zertifizierungshierarchie könnte dann von S/MIME benutzt werden. Der Installationsaufwand wäre in diesem Fall sehr gering.

Für relativ viele bestehende UA's existieren Erweiterungen, um PGP in den UA zu integrieren, d.h. bei einem Einsatz von PGP könnte der Benutzer unter Umständen seinen gewohnten UA beibehalten. Es muß auf allen Anwenderrechnern PGP und abhängig vom jeweilig verwendeten UA eine entsprechende Erweiterung installiert werden. Da nicht für alle UA's eine Erweiterung für PGP existiert, kann nicht verhindert werden, daß einige Benutzer auch bei der Einführung von PGP ihren UA wechseln oder aber PGP als Kommandozeilenprogramm verwenden müssen.

14.3.2 Betriebsaspekte

Für den Fall, daß S/MIME und PGP bei der BMW AG verwendet werden, ist zu berücksichtigen, daß die beiden Dienste nicht kompatibel sind, d.h. PGP kann weder S/MIME–Nachrichten erzeugen noch entschlüsseln. Dasselbe gilt für S/MIME in Bezug auf PGP.

Bei S/MIME benötigen die Benutzer Zertifikate, die sie bei einer Zertifizierungsstelle beantragen müssen. Darüber müssen die Nutzer informiert werden, d.h. sie sind insoweit zu schulen, daß sie in der Lage sind, ein Zertifikat bei der CA zu beantragen und ihre E-Mail entsprechend zu signieren oder zu verschlüsseln.

Bei PGP sind die für jeden Benutzer notwendigen Schlüssel, für das asymmetrische Verschlüsselungsverfahren zu erzeugen. Zur Zertifizierung kann entweder ein zentraler Zertifizierungsansatz nachgebildet werden, d.h. es muß in diesem Fall eine CA für PGP eingeführt werden, oder aber es wird der dezentrale Ansatz von PGP verwendet. Im letztgenannten Fall zertifizieren sich die Nutzer gegenseitig. Die Nutzer müssen sehr genau darüber unterrichtet werden, wie die Zertifizierung anderer Nutzer zu erfolgen hat und nach welchen Richtlinien der Sicherheitspolitik diese Zertifikate ausgestellt werden dürfen, um die Sicherheit des Systems nicht zu schwächen. Insgesamt dürfte der Schulungs– bzw. Einarbeitungsaufwand bei PGP höher als bei S/MIME sein.

14.4 Remote Access Server

Zur Anbindung von Telearbeitsplätzen an das Unternehmen sind Wähl– bzw. ISDN–Verbindungen nötig. Zur Realisierung und Sicherung solcher Wählverbindungen mittels Remote Access Server bzw. Terminal Access Controller (TAC) wurden die Protokolle TACACS+, RADIUS und der RAS–Dienst untersucht (vgl. Abschn. 12).

Um die Einwahl von Mitarbeitern kontrollieren und Einbrüche in das System erkennen zu können, sind Auditingfunktionen essentiell. Zur Durchsetzung der Sicherheitspolitik ist es notwendig, einige wenige zentrale Zugangspunkte zu installieren, die auch unter zentraler Verantwortung verwaltet werden.

Der RAS–Dienst von Windows NT wird mit jedem Windows NT–System ausgeliefert. Damit ist es relativ einfach, von jedem Arbeitsplatzrechner aus einen Zugang ins Unternehmensnetz zu schaffen. Dies widerspricht der Forderung nach wenigen, zentral verwalteten Zugangspunkten. Jeder Benutzer kann potentiell eine „Hintertür" ins System etablieren. Auch die Auditingfunktion des RAS–Dienstes entspricht nicht den Anforderungen. Aus diesen Gründen kann RAS zur Sicherung von Wählverbindungen nicht empfohlen werden. Es ist in Erwägung zu ziehen, die Verwendung von RAS durch die Sicherheitspolitik zu untersagen oder explizit auf sehr wenige Zugänge zu beschränken.

TACACS+ stellt ein proprietäres Protokoll der Firma CISCO dar, sodaß RADIUS als Protokoll in einem heterogenen Umfeld grundsätzlich besser geeignet ist. Da die Produkte von CISCO jedoch eine strategische Plattform im BMW Unternehmensnetz darstellen, ist dieser Nachteil von TACACS+ im Umfeld der BMW AG derzeit nicht entscheidend.

TACACS+ bietet im Gegensatz zu RADIUS die Möglichkeit, mit Access Lists spezifischere Nutzerprofile und feingranularere und komplexere Zugriffskontrollen durchzuführen. Es erlaubt, für jeden Benutzer mehr als ein Authentisierungsverfahren festzulegen. Um dies bei RADIUS nachzubilden, müßten bspw. die Nutzerkennungen durch Prä- bzw. Suffixe parametrisiert werden. RADIUS besitzt allerdings eine umfassendere Auditingfunktionalität als TACACS+, d.h. mit RADIUS können mehr verbindungsorientierte Daten gelogged werden. Bei der Sicherung der Nachrichtenpakete wird von RADIUS lediglich die Paßwortinformation verschlüsselt, wohingegen TACACS+ den gesamten Nachrichtenbody chiffriert.

Die Auswahl zwischen TACACS+ und RADIUS hängt auch von der verwendeten Hardware für den Dial–In–Server und den Network Access Server ab. Sollen hier auch künftig Produkte von CISCO verwendet werden, so wird der Einsatz von TACACS+ nahegelegt. In einem heterogenen Umfeld oder auf anderer Hardware wird empfohlen, RADIUS einzusetzen.

14.4.1 Installation und Migration

Vor einer Installation müssen Anzahl, Standorte und Hardware der TAC festgelegt werden. Dabei ist der Grundsatz zu beachten, daß mit der Zahl der geschaffenen Zugangspunkte auch der Aufwand für deren Sicherung steigt. Es sollte maximal ein Zugangspunkt pro Subnetz eingerichtet werden, besser wäre es, mehrere Subnetze über einen TAC zu versorgen.

Abhängig von der eingesetzten Hardware muß TACACS+ oder RADIUS auf dem TAC installiert werden. Neben diesem Rechner muß innerhalb des jeweiligen Netzbereiches auch ein Network Access Server als dezidierter TACACS+–, bzw. RADIUS–Server eingerichtet werden.

14.4.2 Betriebsaspekte

Für den Betrieb eines Remote Access Servers muß eine Politik festgelegt werden, die die Art des Zugriffs auf das Unternehmensnetz, die Verantwortlichen für das System und insbesondere das Auditing regelt.

Für die Nutzer, die externen Zugang über Wählverbindungen erhalten sollen, sind Kennungen einzurichten. Dabei sind Zugriffsrechte und Authentisierungsfunktionen festzulegen. Es wird empfohlen, grundsätzlich für jeden Benutzer eine Nummer für den Rückruf festzulegen und die Callback–Funktion bei jedem eingehenden Anruf anzuwenden, da dies den Schutz vor Angreifern erhöht. Es muß insbesondere auf die sichere Verteilung der Paßwörter für die Berechtigten geachtet werden. Im laufenden Betrieb müssen die Logs regelmäßig kontrolliert und ausgewertet werden, um eventuelle Angriffsversuche oder einen Mißbrauch des Systems feststellen zu können.

Zugänge über Wählverbindungen, die neben den Zugängen des TAC existieren, sind abzubauen. Die Installation von Hardware und Software, die einen „privaten“ Zugang ermöglicht, d.h. einen Zugang, der nicht über den Remote Access Server abgewickelt wird, ist zu verbieten.

14.5 Schlüssel– und Zertifikatsmanagement

In Abschnitt 6 wurde das Schlüssel– bzw. Zertifikatsmanagement bereits generell behandelt. Da alle in diesem Kapitel für den Einsatz bei der BMW AG empfohlenen Produkte und Protokolle Schlüssel bzw. Zertifikate benötigen, stellt sich die Frage nach der praktischen Realisierung des Schlüssel– und Zertifikatsmanagements. Wünschenswert wäre ein organisatorisch und technisch einheitliches Konzept zum Keymanagement. Dies läßt sich im Moment allerdings nicht realisieren, da die Formate für Schlüssel bzw. Zertifikate zwischen den einzelnen Protokollen zum Teil nicht kompatibel und nicht austauschbar sind.

Im folgenden wird deshalb für jedes Protokoll das Management von Schlüsseln und Zertifikaten gesondert betrachtet.

14.5.1 SSH

Im SSH–Paket ist ein Dienstprogramm (`make-ssh-known-hosts`) enthalten, das die Generierung der `/etc/ssh_known_hosts` Dateien automatisiert. Wird dieses Programm auf einem Rechner gestartet, so versucht das Programm mit Hilfe von DNS–Abfragen die Rechneradressen in entsprechend zu definierenden Netzsegmenten zu ermitteln. Daran anschließend wird versucht, eine SSH–Verbindung zu den Rechnern, deren Schlüssel noch nicht bekannt sind, aufzubauen. Ist dies möglich, wird der host_key dieses Rechners in der `/etc/ssh_known_hosts`–Datei gespeichert. Da dieses Vorgehen vollautomatisch erfolgen kann und dadurch keine zu großen Sicherheitsrisiken entstehen (vgl. Abschn. 9.4), wird die Verwendung von `make-ssh-known-hosts` empfohlen.

Zukünftig soll SSH auch X.509v3–Zertifikate unterstützen. Sobald dies möglich ist, könnte auch ein unternehmenseigener Zertifizierungsdienst, wie in Abschnitt 14.5.2 beschrieben, zur Verteilung der SSH–Schlüssel verwendet werden.

Die host_keys der einzelnen Rechner müssen regelmäßig durch neue Schlüssel ersetzt werden. Dadurch wird die kryptologische Sicherheit erhöht. Bei 1024 Bit langen Schlüsseln kann eine maximale Lebensdauer von ein bis zwei Jahren derzeit als ausreichend erachtet werden. Da der host_key eines Rechners A bei allen Rechnern bzw. Benutzern, die mit dem Rechner A kommunizieren entweder in der Datei `/etc/ssh_known_hosts` oder in `~/.ssh/known_hosts` gespeichert ist, muß der Austausch der Schlüssel vorher genau geplant und den Benutzern bekanntgegeben werden.

Die Administratoren der anderen Rechner müssen vor dem Wechsel des Schlüssels von Rechner A, diesen aus den entsprechenden Dateien löschen und anschließend den neuen Schlüssel durch einen Aufruf von `make-ssh-known-hosts` wieder in die `/etc/ssh_known_hosts` integrieren. Die Benutzer, die mit Rechner A kommunizieren, müssen den Schlüssel u.U. aus ihrer `~/.ssh/known_hosts`–Datei entfernen. Falls der neue host_key nicht in der Datei `/etc/ssh_known_hosts` gespeichert wurde, wird er beim nächsten Verbindungsaufbau zum Rechner A in die `~/.ssh/known_hosts`–Datei aufgenommen.

Wird der alte Schlüssel nicht gelöscht und versucht eine Verbindung zum Rechner A aufzubauen, schlägt die Authentisierung und damit der Verbindungsaufbau fehl, und SSH gibt die in Abbildung 14.1 dargestellte Fehlermeldung aus.

14.5.2 SSL, HTTPS und S/MIME

SSL, HTTPS und S/MIME stützen sich auf Zertifikate (vgl. Abschn. 6.2) ab. Um diese Protokolle einsetzen zu können, müssen alle Server und Clients bzw. Benutzer ein Zertifikat besitzen.

Die Zertifikate können bei kommerziellen Zertifizierungsstellen beantragt werden. Es dürfte sich aus wirtschaftlichen Gründen verbieten, allen Benutzern bei der BMW AG von einer entsprechenden CA ein Zertifikat ausstellen zu lassen. Daher muß unternehmensintern mittelfristig

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@       WARNING: HOST IDENTIFICATION HAS CHANGED!         @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the host key has just been changed.
Please contact your system administrator.
Add correct host key in /user/home/helmut/.ssh/known_hosts to get rid of this message.
Password authentication is disabled to avoid trojan horses.
```

Abbildung 14.1: Ausgabe von SSH bei falschem host_key

eine eigene Zertifizierungshierarchie aufgebaut werden. Für die Einführung von HTTPS und S/MIME im Zentralbereich der BMW AG wird empfohlen, eine flache Zertifizierungshierarchie zu installieren. Pro Abteilung bzw. Subnetz werden CA's installiert, die Server, Clients und Benutzer im entsprechenden Subnetz zertifizieren. Diese CA's wiederum werden von einer Root–CA zertifiziert.

Für den Fall, daß der Einsatz der Protokolle im gesamten Corporate Network geplant ist, könnte eine weitere Stufe in die Zertifizierungshierarchie eingeführt werden. In diesem Fall müßte eine konzernweite CA die Root–CA des Zentralbereiches und die dann einzurichtenden CA's der Tochter– und Beteiligungsgesellschaften zertifizieren.

14.5.3 PGP

PGP verwendet ein hybrides Verschlüsselungsverfahren. Ein, für jede Kommunikationsbeziehung neu zu berechnender Kommunikationsschlüssel, wird mit Hilfe des öffentlichen Schlüssels des Empfängers verschlüsselt und mit der Nachricht an den Empfänger übermittelt. Für PGP benötigt also jeder Benutzer ein Schlüsselpaar für das verwendete asymmetrische Verschlüsselungsverfahren. Die Erzeugung dieser Schlüssel mit PGP ist einfach. Es wird empfohlen, den Benutzern die Verwendung von 1024 Bit langen Schlüsseln vorzuschreiben. Da mit dem asymmetrischen Verfahren lediglich Kommunikationsschlüssel und MAC's verschlüsselt werden und diese Zeichenfolgen „zufällig“ sind, muß das Schlüsselpaar für das asymmetrische Verfahren nicht so häufig gewechselt werden. Grundsätzlich gilt, je größer die verschlüsselte Datenmenge und je sensibler die Daten, desto kürzer sollte die Lebensdauer der Schlüssel sein.

Um mit einem Partner kommunizieren zu können, benötigt ein Benutzer den öffentlichen Schlüssel dieses Partners. PGP verwendet zur Verteilung und zur Zertifizierung dieser Schlüssel ein dezentrales Zertifizierungsverfahren (vgl. S. 86). Soll dieses Verfahren auch im Unternehmensnetz eingesetzt werden, so müssen die Benutzer genau darüber unterrichtet werden, was es bedeutet, einen Schlüssel zu signieren und welche Konsequenzen die Signatur eines „falschen“ Schlüssels haben kann.

Es ist alternativ auch für PGP möglich einen hierarchischen Zertifizierungsansatz nachzubilden. Jeder Benutzer muß dann seinen öffentlichen Schlüssel von einer CA unterschreiben lassen. In diesem Fall muß den Benutzern vorgeschrieben werden, daß für die Kommunikation nur Schlüssel verwendet und akzeptiert werden dürfen, die von einer offiziellen CA signiert wurden.

Es ist zu erwarten, daß zukünftige Versionen von PGP auch X.509–Zertifikate unterstützen. In diesem Fall könnte die bestehende Zertifizierungshierarchie für HTTPS und SSL auch für PGP verwendet werden. Voraussetzung dafür ist allerdings, daß diese CA's auch lange Schlüssel zur Signatur der Zertifikate verwenden können.

Grundsätzlich wird empfohlen, für PGP eine eigene Zertifizierungshierarchie aufzubauen. Für die Einführungsphase von PGP und solange, bis die einheitliche Zertifizierungshierarchie eingerichtet ist und verwendet werden kann, ist es auch möglich, das dezentrale Zertifizierungskonzept von PGP zu verwenden. Dann müssen aber die Zertifikate, die sich die Nutzer gegenseitig ausstellen, nach der Übergangsphase von der zentralen CA signiert werden.

14.5.4 TACACS+ bzw. RADIUS

Bei den Protokollen für Terminal Access Server TACACS+ und RADIUS handelt es sich um eine Client/Server–Architektur. Der Verkehr zwischen Client und Server wird entweder vollständig verschlüsselt oder es werden zumindest die übertragenen Klartextpaßwörter gesichert. Um dies realisieren zu können, muß zwischen Client und Server ein gemeinsames Geheimnis bestehen, d.h. Client und Server müssen einen gemeinsamen Schlüssel besitzen. Da i.d.R. nur ein oder bestenfalls ein paar wenige TAC als RADIUS– bzw. TACACS+–Client auftreten und die entsprechenden Server im Unternehmensnetz ebenfalls nicht sehr zahlreich sind, bedarf es keiner aufwendigen Schlüsselverteilungsstrategie. Die Schlüssel können bei der Installation festgelegt und sofort installiert werden. Auch bei einem späteren Wechsel der Schlüssel kann dies „per Hand“ erfolgen.

Kritischer ist die sichere Verteilung der Paßwörter bzw. der Authentisierungsinformationen, die die entsprechenden Benutzer benötigen. Zu deren Verteilung können jedoch bereits bestehende, sichere Verbindungen, z.B. über E-Mail oder gesicherte WWW–Verbindungen verwendet werden.

14.6 Unternehmensweite Software Distribution

Die Unterschiede im Aufwand für die Einführung der empfohlenen Produkte im Zentralbereich München sind erheblich. Bei TACACS+ bzw. RADIUS kann die Installation unter Umständen noch manuell erfolgen, da die Zahl der zu installierenden Systeme gering bleibt. So müssen, auch wenn die Protokolle im gesamten Zentralbereichsnetz eingesetzt werden, verhältnismäßig wenige Clients und entsprechende Server im Unternehmensnetz installiert werden.

Bei der Einführung von SSH, SSL, HTTPS, S/MIME oder PGP ist der Aufwand unverhältnismäßig höher. So muß bei SSH auf allen UNIX–Systemen die Software für SSH–Server und SSH–Client installiert werden. Auf allen Windows–Systemen, von denen aus auf UNIX–Rechner zugegriffen wird, ist ein SSH–Client einzurichten. Außerdem muß für jeden Rechner ein host_key generiert werden. Für alle Benutzer von SSH ist es nötig, user_keys zu erzeugen.

Bei SSH und HTTPS müssen alle Clients und Server, die SSL nicht unterstützen, durch SSL–fähige Produkte ersetzt werden, und es ist erforderlich für alle Client– und Server–Rechner sowie für die Benutzer Zertifikate zu beantragen und auszustellen.

Auch bei S/MIME und PGP muß im schlimmsten Fall auf allen Endsystemen Software installiert werden. Auch hier sind für alle Benutzer Schlüssel bzw. Zertifikate zu erzeugen.

Bei rund 20.000 Endsystemen in Zentralbereich München ist klar, daß die Installation und die Generierung von Schlüsseln bzw. Zertifikaten nicht manuell erfolgen kann. Es bietet sich an, ein Software–Verteilungssystem für diese Aufgaben zu nutzen, um den Aufwand zu verringern.

Ein Beispiel für ein solches System wäre *QLINE®/ASDIS (Automated Software Distribution and Inventory Management System)* [BB-97] der Firma BB–Data. Mit diesem System lassen sich die, zur Verteilung vorgesehenen Objekte (Software, Daten), zu Paketen (*Verteileinheiten*) zusammenstellen. Innerhalb dieser Verteileinheiten können beliebig viele Versionen und für jede Version beliebig viele Varianten parallel verwaltet werden.

ASDIS arbeitet auftragsorientiert. Es können u.a. Aufträge zur

- Installation oder Deinstallation,
- zur Verteilung mit oder ohne Installation und
- zur Bestandsaufnahme von Hardware und Software

erteilt werden. Die Verteilung der Pakete aufgrund eines Auftrages, erfolgt asynchron im Hintergrund. Ein Auftrag kann für einzelne Systeme oder für Gruppen von Systemen formuliert werden. Die Gruppen können explizit oder implizit über vorzugebende Systemkonfigurationen definiert werden. Bei der Auftragsannahme expandiert ASDIS an mehrere Systeme gerichtete Aufträge (*Globalaufträge*) automatisch in *Elementaraufträge*. Es ordnet anhand des Systemtyps jedem Zielsystem automatisch die dafür vorgesehene Variante des Paketes zu. Die Pakete werden über bestehende Netzstrukturen verteilt und auf heterogenen Umgebungen installiert. Dabei wird ein Inventarverzeichnis erstellt, das die Hardware– und die installierte Software–Konfiguration enthält. Diese Datenbasis liefert aktuelle Informationen über alle Systeme, die mit ASDIS verwaltet werden.

Mit dieser Funktionalität kann für alle besprochenen Produkte und Protokolle die notwendige Software mit der systemspezifischen Konfiguration automatisch auf heterogenen Endsystemen installiert werden. ASDIS bietet zudem die Möglichkeit, beliebige Kommandos auf dem Zielsystem abzusetzen. Damit kann, nach Erstellung entsprechender Skripten, die Generierung von

Schlüsseln und Zertifikaten für die Endsysteme und unter Umständen sogar für die Benutzer automatisiert werden.

Kapitel 15

Zusammenfassung und Ausblick

Die BMW AG betreibt ein Corporate Network auf Basis von TCP/IP. Daten, die über offene Netze und über TCP/IP übertragen werden, sind relativ leicht angreifbar. Daher muß die Kommunikation im Unternehmensnetz der BMW AG vor diesen Angriffen geschützt werden.

Hierzu war es im ersten Teil dieser Arbeit (Kapitel 2 bis 6) nötig die Grundlagen der sicheren TCP/IP–basierten Kommunikation festzulegen. Dazu mußte der Begriff bzw. die Dimensionen der sicheren Kommunikation, ebenso wie der Begriff der Sicherheitspolitik näher gefaßt und dargestellt, sowie grundlegenden Konzepte des Schlüssel– und Zertifikatsmanagements erörtert werden.

Im zweiten Teil (Kapitel 7 und 14) wurden konkrete Protokolle und Produkte untersucht, bewertet, verglichen und auf die Einsatzfähigkeit bei der BMW AG hin beurteilt. Dabei konnten Lösungen zur Realisierung geschlossener Benutzergruppen mittels Tunneling und zur Sicherung des E-Mail sowie des WWW–Verkehrs aufgezeigt, sowie Konzepte zur Sicherung des Log on auf entfernten Rechnern und zur Sicherung der entfernten Einwahl über Wählverbindungen entwickelt werden. Außerdem wurden Protokolle, die auf TCP aufsetzen und Prozessen der höheren Schichten Dienste zur Sicherung der Kommunikation anbieten, erörtert. Mit Hilfe dieser Protokolle lassen sich prinzipiell alle Dienste, die TCP benutzen, sichern. Abschließend wurden konkrete Empfehlungen ausgesprochen, welche Protokolle im Unternehmensnetz der BMW AG tatsächlich eingesetzt werden sollen. Dabei lag der Focus auf Managementfragen, Installations- und Migrationsszenarien sowie Betriebsaspekten. Durch die Einführung der ausgewählten Protokolle läßt sich ein erheblicher Zuwachs an Sicherheit bei der BMW AG erzielen.

In der vorliegenden Arbeit konnte nur ein Ausschnitt der komplexen Thematik der sicheren Kommunikation untersucht werden. Im folgenden werden daher weiterführende Fragestellungen aus diesem Themenkreis angegeben.

- **Schlüssel– und Zertifikatsmanagement**

  Im Rahmen dieser Arbeit wurden grundlegende Probleme des Schlüssel– und Zertifikatsmanagements geklärt. Der Aufbau einer einheitlichen, unternehmensweiten Zertifizierungshierarchie bei der BMW AG ist ein wichtiger weiterer Schritt zur Verbesserung der Sicherheit. Dabei muß insbesondere eine Zertifizierungspolitik festgelegt werden. Die derzeit einsetzbaren Protokolle und Produkte zeichnen sich auch dadurch aus, daß Schlüssel bzw. Zertifikate nicht kompatibel sind. In diesem Bereich wäre die Vereinheitlichung der verschiedenen Formate bzw. die Verwaltung in einer einheitlichen Zertifizierungshierarchie zu untersuchen.

- **Tunneling mit IPv6**

  Bisher wurde IPv6 im Corporate Network der BMW AG nicht eingesetzt. Da es im Moment noch keine ausgereiften Produkte gibt, die AH oder ESP implementieren, konnten keine Tests zur Realisierung von Tunneling mit Hilfe dieser Protokolle durchgeführt werden. Um die Möglichkeiten von AH und ESP und die Vorteile von IPv6, auch in Bezug auf die Netzsicherheit, zu testen, sollte ein IPv6–Testbed aufgebaut und der Einsatz von IPv6 im Unternehmensnetz evaluiert werden.

- **Internationalisierung, Exportbeschränkungen**

  Die vorliegende Arbeit beschäftigt sich überwiegend mit dem Schutz der Kommunikation innerhalb des Zentralbereiches der BMW AG und damit innerhalb Deutschlands. Da das Unternehmen jedoch ein weltweites Netz betreibt, sollte auch die Sicherung der internationalen Verbindungen untersucht werden. Hierbei sind insbesondere bestehende rechtliche Beschränkungen innerhalb einzelner Länder zu berücksichtigen.

  Die Exportbeschränkungen für kryptologische Produkte aus den Vereinigten Staaten und die damit verbundene Beschränkung der Schlüssellängen schwächt die Sicherheit einiger der betrachteten Produkte erheblich. Der Markt und die zukünftige Entwicklung muß mit dem Ziel beobachtet werden, eine Beschränkung der Schlüssellänge zu vermeiden.

- **Software Verteilung**

  Bei einer Einführung der besprochenen Produkte ist Software auf tausenden von Endsystemen zu installieren und zu konfigurieren. In dieser Arbeit wurde die automatische Distribution von Software nur am Rande behandelt. Soweit bei der BMW AG noch kein Software Verteilungssystem verwendet wird, sollte der Einsatz eines solchen Werkzeugs in Erwägung gezogen wrden werden.

- **UDP–basierte Dienste**

  Gegenstand dieser Arbeit war die Sicherung der TCP/IP–basierten Kommunikation. Auch bei der UDP–basierten Kommunikation gibt es Dienste, für die untersucht werden muß, wie sie vor Angriffen geschützt werden können. Als Beispiele seien hier nur das Simple Network Management Protocol (SNMP) oder das Network File System (NFS) genannt. Auch über einen Angriff UDP–basierter Dienste kann einem Unternehmen erheblicher Schaden entstehen.

Anhang A

Kryptologie

Kryptologie kann als die Wissenschaft der algorithmischen Methoden und Protokolle zur Informationssicherung definiert werden. Unter dem Begriff der *Kryptosysteme* faßt man die Verschlüsselungsverfahren und die kryptographischen Protokolle zusammen. Man unterteilt die Kryptologie in die Untergebiete *Kryptographie* (Entwurf von Systemen) und *Kryptanalyse* (Brechen von Systemen). [Bau94, Sch94]

Bei Nachrichten, die verschlüsselt (*encrypt*, *encode*) werden sollen, spricht man vom *Klartext (Cleartext)*, die verschlüsselte Nachricht heißt *Geheimtext (Ciphertext)*. Der Algorithmus, der die Verschlüsselung realisiert, heißt *Chiffrierverfahren*, *Chiffrierung* oder *Verschlüsselungsverfahren (Cipher)*. Wird ein Geheimtext einer weiteren Chiffrierung unterzogen, so spricht man von *Überschlüsselung* oder *Überchiffrierung*. Bei den Verschlüsselungsverfahren unterscheidet man symmetrische, asymmetrische und hybride Verfahren.

Da nicht alle Verschlüsselungsverfahren, die von den untersuchten Protokollen verwendet werden, auch beschrieben werden können, wird im folgenden für jede Klasse ein Vertreter aufgeführt und erläutert, um die grundlegende Funktionsweise solcher Verfahren zu verdeutlichen.

A.1 Symmetrische Chiffrierverfahren (IDEA)

Symmetrische Verschlüsselungsverfahren zeichnen sich dadurch aus, daß Sender und Empfänger denselben Schlüssel verwenden, um die Nachricht zu ver– bzw. entschlüsseln. Das wohl bekannteste symmetrische Verfahren ist DES, bei dem aber die kurze Schlüssellänge (56 Bit) als Schwäche angesehen wird[1]. In diesem Abschnitt soll deshalb, mit IDEA, ein Verfahren mit einer Schlüssellänge von 128 Bit vorgestellt werden.

[1] Am 17. Juni 1997 wurde DES durch einen Brute–Force Angriff gebrochen. Der Angriff wurde von einem Team im Internet koordiniert und es wird geschätzt, daß sich über 70.000 Rechner weltweit an der Suche nach dem Schlüssel beteiligt haben. http://www.rsa.com/des und http://www.frii.com/~rcv/deschall.htm

IDEA wurde 1990 [LM90] von Xuejia Lai und James Massey eingeführt. Nachdem Biham und Shamir die differentielle Kryptanalyse entwickelt hatten, wurde der IDEA–Algorithmus geändert, um Schutz vor diesem Angriff zu bieten [Lai92, Sch94].

IDEA stellt eine *Blockchiffrierung* dar, es arbeitet auf 64 Bit langen Klartextblöcken. Derselbe Algorithmus wird für die Verschlüsselung ebenso wie für die Entschlüsselung verwendet. Das Designprinzip, das hinter der Entwicklung von IDEA steht, ist die „Mischung von Operationen aus verschiedenen algebraischen Gruppen". Verwendet werden drei algebraische Gruppen, deren Operationen gemischt verwendet werden und die leicht sowohl in Hardware als auch in Software implementiert werden können:

- XOR
- Addition modulo 2^{16}
- Multiplikation modulo $2^{16} + 1$

Abbildung A.1 stellt ein Ablaufdiagramm von IDEA dar. Der 64 Bit Klartextblock wird in vier 16 Bit lange Blöcke X_1, X_2, X_3 und X_4 aufgeteilt. Diese vier Blöcke bilden den Input für die erste Runde des Algorithmus, der insgesamt acht Runden durchläuft. In jeder Runde werden die vier Blöcke, untereinander und mit sechs 16 Bit langen Teilblöcken des Schlüssels, XOR verknüpft, addiert und multipliziert. Zwischen den einzelnen Runden werden der dritte und der vierte Teilblock vertauscht.
In jeder einzelnen Runde werden die folgenden Operationen durchgeführt:

1. Multiplikation von X_1 mit dem ersten Schlüsselblock
2. Addition von X_2 mit dem zweiten Schlüsselblock
3. Addition von X_3 mit dem dritten Schlüsselblock
4. Multiplikation von X_4 mit dem vierten Schlüsselblock
5. XOR Verknüpfung der Resultate aus Schritt 1 und 3
6. XOR Verknüpfung der Resultate aus Schritt 2 und 4
7. Multiplikation der Ergebnisse aus Schritt 5 mit dem fünften Schlüsselblock
8. Addition der Ergebnisse aus Schritt 6 und 7
9. Multiplikation des Ergebnisses aus Schritt 8 mit dem sechsten Schlüsselblock
10. Addition der Ergebnisse aus Schritt 7 und 9
11. XOR Verknüpfung der Resultate aus Schritt 1 und 9
12. XOR Verknüpfung der Resultate aus Schritt 3 und 9

13. XOR Verknüpfung der Resultate aus Schritt 2 und 10
14. XOR Verknüpfung der Resultate aus Schritt 4 und 10

Die Ergebnisse der Schritte 11, 12, 13 und 14 bilden die vier Subblöcke für die nächste Runde. Nach acht Runden wird die folgende Ausgabetransformation durchgeführt:

1. Multiplikation von X_1 mit dem ersten Schlüsselblock
2. Addition von X_2 mit dem zweiten Schlüsselblock
3. Addition von X_3 mit dem dritten Schlüsselblock
4. Multiplikation von X_4 mit dem vierten Schlüsselblock

Diese vier Ergebnisse werden konkateniert und bilden einen Block des Geheimtextes.

Um die Schlüsselblöcke zu generieren, wird der 128 Bit Schlüssel zuerst in acht 16 Bit Schlüsselblöcke aufgeteilt. Von diesen werden sechs Schlüsselblöcke für die erste Runde und zwei Schlüsselblöcke für die zweite Runde verwendet. Dann wird der Schlüssel um 25 Bit nach links rotiert und wiederum in acht Schlüsselblöcke aufgeteilt, von denen die ersten vier in Runde zwei und die letzten vier in Runde drei verwendet werden. Der Schlüssel wird wieder um 25 Bit nach links rotiert, um die nächsten acht Schlüsselblöcke zu erhalten. Der oben angegebene Vorgang wird solange wiederholt, bis alle Schlüsselblöcke für die acht Runden erzeugt sind.

Runde	Schlüsselblöcke zum Verschlüsseln						Schlüsselblöcke zum Entschlüsseln					
1	$Z_{1(1)}$	$Z_{2(1)}$	$Z_{3(1)}$	$Z_{4(1)}$	$Z_{5(1)}$	$Z_{6(1)}$	$Z_{1(9)}^{-1}$	$-Z_{2(9)}$	$-Z_{3(9)}$	$Z_{4(9)}^{-1}$	$Z_{5(8)}$	$Z_{6(8)}$
2	$Z_{1(2)}$	$Z_{2(2)}$	$Z_{3(2)}$	$Z_{4(2)}$	$Z_{5(2)}$	$Z_{6(2)}$	$Z_{1(8)}^{-1}$	$-Z_{2(8)}$	$-Z_{3(8)}$	$Z_{4(8)}^{-1}$	$Z_{5(7)}$	$Z_{6(7)}$
3	$Z_{1(3)}$	$Z_{2(3)}$	$Z_{3(3)}$	$Z_{4(3)}$	$Z_{5(3)}$	$Z_{6(3)}$	$Z_{1(7)}^{-1}$	$-Z_{2(7)}$	$-Z_{3(7)}$	$Z_{4(7)}^{-1}$	$Z_{5(6)}$	$Z_{6(6)}$
4	$Z_{1(4)}$	$Z_{2(4)}$	$Z_{3(4)}$	$Z_{4(4)}$	$Z_{5(4)}$	$Z_{6(4)}$	$Z_{1(6)}^{-1}$	$-Z_{2(6)}$	$-Z_{3(6)}$	$Z_{4(6)}^{-1}$	$Z_{5(5)}$	$Z_{6(5)}$
5	$Z_{1(5)}$	$Z_{2(5)}$	$Z_{3(5)}$	$Z_{4(5)}$	$Z_{5(5)}$	$Z_{6(5)}$	$Z_{1(5)}^{-1}$	$-Z_{2(5)}$	$-Z_{3(5)}$	$Z_{4(5)}^{-1}$	$Z_{5(4)}$	$Z_{6(4)}$
6	$Z_{1(6)}$	$Z_{2(6)}$	$Z_{3(6)}$	$Z_{4(6)}$	$Z_{5(6)}$	$Z_{6(6)}$	$Z_{1(4)}^{-1}$	$-Z_{2(4)}$	$-Z_{3(4)}$	$Z_{4(4)}^{-1}$	$Z_{5(3)}$	$Z_{6(3)}$
7	$Z_{1(7)}$	$Z_{2(7)}$	$Z_{3(7)}$	$Z_{4(7)}$	$Z_{5(7)}$	$Z_{6(7)}$	$Z_{1(3)}^{-1}$	$-Z_{2(3)}$	$-Z_{3(3)}$	$Z_{4(3)}^{-1}$	$Z_{5(2)}$	$Z_{6(2)}$
8	$Z_{1(8)}$	$Z_{2(8)}$	$Z_{3(8)}$	$Z_{4(8)}$	$Z_{5(8)}$	$Z_{6(8)}$	$Z_{1(2)}^{-1}$	$-Z_{2(2)}$	$-Z_{3(2)}$	$Z_{4(2)}^{-1}$	$Z_{5(1)}$	$Z_{6(1)}$
AT	$Z_{1(9)}$	$Z_{2(9)}$	$Z_{3(9)}$	$Z_{4(9)}$			$Z_{1(1)}^{-1}$	$-Z_{2(1)}$	$-Z_{3(1)}$	$Z_{4(1)}^{-1}$		

AT = Ausgabetransformation

Tabelle A.1: IDEA Schlüsselblöcke zur Ver– und Entschlüsselung

Zur Entschlüsselung wird derselbe Algorithmus verwendet. Allerdings werden hier als Schlüsselblöcke entweder additive oder multiplikative Inverse der Schlüsselblöcke verwendet, die zur Verschlüsselung dienen. In Tabelle A.1 sind die Schlüsselblöcke für die Verschlüsselung und für die Entschlüsselung angegeben. Z^{-1} bezeichnet dabei das multiplikative Inverse von Z, $-Z$ das additive Inverse. $Z_{1(2)}$ bezeichnet den ersten Schlüsselblock der zweiten Runde bei der Verschlüsselung.

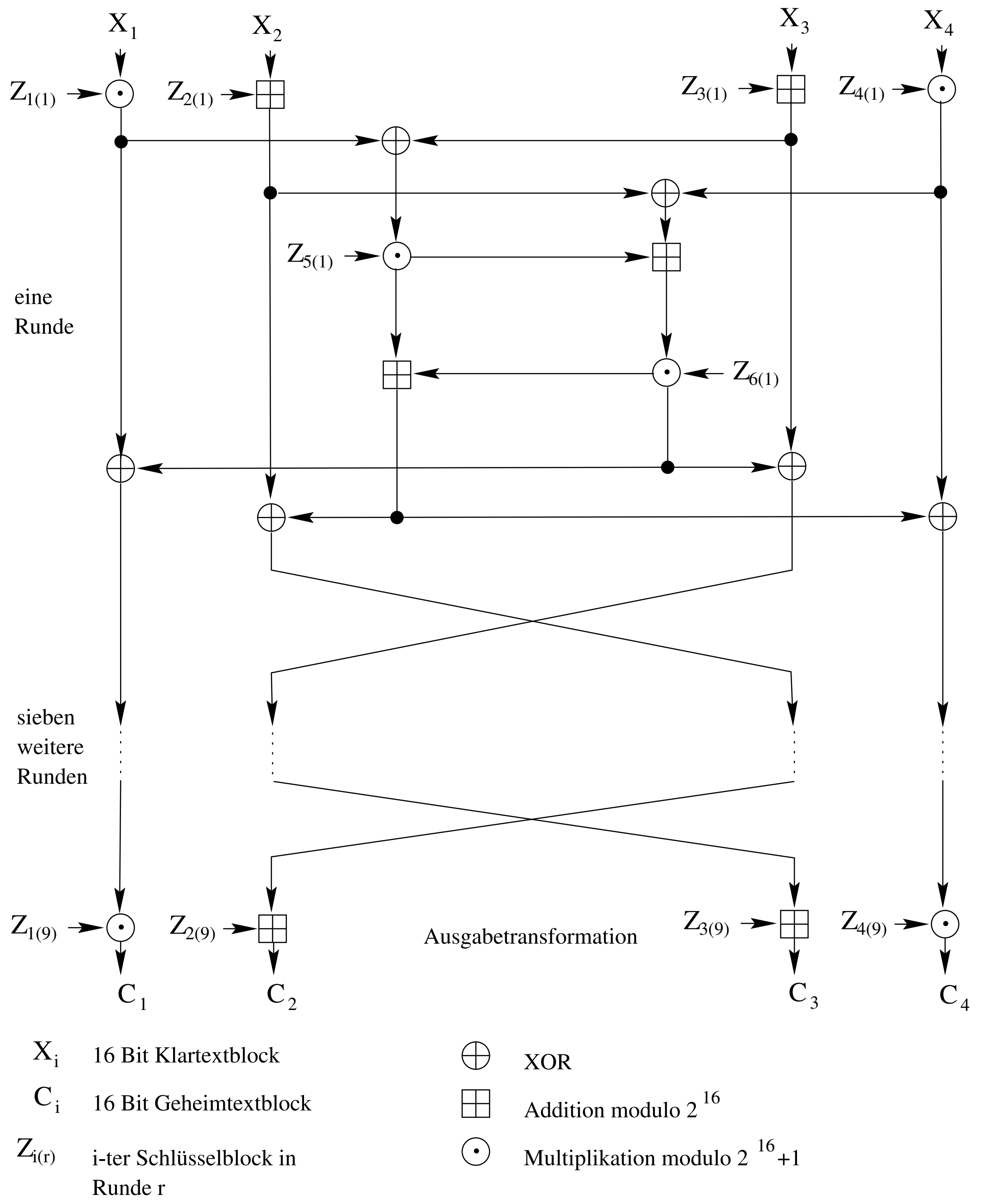

Abbildung A.1: IDEA Algorithmus

A.2 Asymmetrische Chiffrierverfahren (RSA)

In einem asymmetrischen Verschlüsselungsverfahren, auch Public Key Verfahren genannt, unterscheidet man ein Verschlüsselungsverfahren E, das öffentlich bekannt gemacht werden darf, von einem Entschlüsselungsverfahren D, das geheimgehalten werden muß. Ein Public Key Verfahren muß folgende Bedingungen erfüllen.

1. Das Entschlüsseln einer verschlüsselten Nachricht M ergibt M, d.h. $\forall M : \; D(E(M)) = M$
2. Sowohl E als auch D können einfach berechnet werden.
3. Aus der Kenntnis von E kann ein Angreifer in vertretbarem Aufwand D nicht berechnen.
4. Falls auf eine Nachricht M vorher das Entschlüsselungsverfahren D und danach das Verschlüsselungsverfahren E angewendet wird, erhält man M als Ergebnis. Diese Bedingung ist optional, sie ermöglicht allerdings die Erstellung digitaler Signaturen.

Im folgenden wird das bekannteste asymmetrische Verfahren RSA [RSA78] beschrieben.

Bei RSA wird ein öffentlicher und ein privater Schlüssel unterschieden. Der öffentliche Schlüssel darf bekanntgegeben werden, wohingegen der private Schlüssel geheimgehalten werden muß.

Die Sicherheit von RSA beruht auf der Schwierigkeit der Faktorisierung großer Zahlen. Der öffentliche und der private Schlüssel sind Funktionen von einem Paar sehr langer Primzahlen (150 oder mehr Dezimalstellen). Um die Schlüssel zu erzeugen, müssen zwei große Primzahlen p und q gewählt werden. Aus diesen Primzahlen wird das Produkt n berechnet:

$$n = p \cdot q$$

Daran anschließend wird eine Zahl d so gewählt, daß d und $(p-1) \cdot (q-1)$ relativ prim sind, d.h.

$$\mathrm{ggt}(d, (p-1) \cdot (q-1)) = 1$$

Dann wird mit Hilfe des Euklidischen Algorithmus eine Zahl e als multiplikatives Inverses von $d \bmod (p-1) \cdot (q-1)$ berechnet, d.h.

$$e \cdot d = 1 \bmod (p-1) \cdot (q-1)$$

Die Zahlen e und n stellen den öffentlichen Schlüssel dar, d ist der private Schlüssel. Die Zahlen p und q werden nicht länger benötigt, sie können gelöscht werden, dürfen aber auf keinen Fall bekanntgegeben werden.

Um eine Nachricht M zu verschlüsseln, wird die binäre Darstellung von M in Blöcke m_i aufgeteilt, die als Zahl interpretiert zwischen 0 und $n-1$ liegen. Auf diese Blöcke wird folgende Verschlüsselungsfunktion angewendet.

$$c_i = E(m_i) = {m_i}^e \bmod n$$

Die Konkatenation der c_i Blöcke liefert den Geheimtext. Um eine Nachricht zu entschlüsseln, muß auf jeden c_i Block folgende Funktion angewendet werden.

$$m_i = D(c_i) = {c_i}^d \bmod n$$

Im folgenden wird gezeigt, daß

$$E(D(m)) \overset{(1)}{=} D(E(m)) \overset{(2)}{=} m$$

gilt, d.h. daß die Entschlüsselungsfunktion auch tatsächlich wieder den Klartext liefert (2) und daß RSA die Bedingung für die Erstellung digitaler Signaturen erfüllt (1). Grundlegend für den Beweis ist der Eulersche Satz

$$m^{\varphi(n)} \bmod n = 1 \quad \text{falls } \mathrm{ggt}(m, n) = 1$$

wobei $\varphi(p) = p - 1$ und $\varphi(n) = \varphi(p \cdot q) = \varphi(p) \cdot \varphi(q) = (p-1)(q-1)$, sowie $\varphi(p^r) = (p-1)p^{r-1}$ falls p, q prim und $r \in N$.

1. Mit $e \cdot d = 1 \bmod \varphi(n)$ gilt
   $D(E(m)) = m^{ed} \bmod n = m^{k\varphi(n)+1} \bmod n = m^{de} \bmod n = E(D(m))$

2. In diesem Fall muß unterschieden werden, ob p Teiler von m ist.

   - Fall 1: p teilt m, dann gilt mit geeignetem $i \in N$
     $m^{ed} \bmod p = m^{k\varphi(n)+1} \bmod p = \underbrace{(ip)}_{=0 \bmod p}{}^{k\varphi(n)+1} \bmod p = 0 = m \bmod p$
   - Fall 2: p ist nicht Teiler von m, dann gilt mit dem Satz von Euler
     $m^{ed} \bmod p = m^{k\varphi(n)+1} \bmod p = m \cdot \underbrace{(m^{p-1})}_{=1}{}^{(q-1)k} \bmod p = m \bmod p$

   Es gilt also immer $m^{ed} \bmod p = m^{k\varphi(n)+1} \bmod p = m \bmod p$. Analog gilt für q: $m^{ed} \bmod q = m \bmod q$

   Es existieren also $i, j \in N$ mit $ip + m = m^{ed} = jq + m$ und damit $ip = jq$. Da p und q Primzahlen sind, gilt dann $\exists r \in N : j = rp$ und $i = rq$. Damit gilt schließlich $m^{ed} = jq + m = rpq + m = rn + m$ also

   $$m^{ed} \bmod n = rn + m \bmod n = m$$

A.3 Message Authentication Code Verfahren (SHA)

Ein Message Authentication Code (MAC), ist ein Hash–Wert h, der über ein Datenpaket beliebiger Länge berechnet wird. Er hat eine feste, i.d.R. sehr viel kürzere Länge als die eigentliche Nachricht M. Der Wert wird mittels einer sogenannten *One–Way Hash Funktion* berechnet: $h = MAC(M)$. Für diese Funktion gilt:

1. Bei gegebener Nachricht M ist es einfach den MAC h zu berechnen.
2. Bei gegebenem MAC h ist es schwierig, die Nachricht M zu berechnen.
3. Es ist schwierig, eine Kollision zu finden, d.h. bei gegebener Nachricht M eine Nachricht M' mit $MAC(M) = MAC(M')$.

Die Definition des Begriffes „schwierig“ hängt von den spezifischen Sicherheitsanforderungen ab. Bei den meisten Implementierungen wird eine Größenordnung von 2^{64} oder mehr Operationen als schwierig definiert. D.h. um eine Kollision zu finden, muß der Angreifer mindestens 2^{64} Operationen ausführen.

Im folgenden wird der Secure Hash Algorithm (SHA) als Teil des Secure Hash Standards [Nat95] vorgestellt. SHA erzeugt aus Nachrichten mit einer Länge kleiner als 2^{64} Bits einen 160 Bit langen Hash–Wert. Um einen Brute–Force Angriff gegen SHA auszuführen, d.h. um eine Kollision zu finden, muß der Angreifer mindestens 2^{80} Operationen ausführen (vgl. Birthday Attack [Sch94] S. 322).

Der Algorithmus arbeitet auf 512 Bit langen Blöcken der Nachricht M. Der Klartext muß dazu auf ein Vielfaches von 512 Bit ($n \cdot 512$ Bit) aufgefüllt werden (*Padding*). Dazu wird er zuerst auf eine Länge von $n \cdot 512 - 64$ Bit gebracht, indem eine 1 und soviele 0'en, wie benötigt, ergänzt werden. Dann wird an diesen Bitstring die Länge des Klartextes (vor dem Padding) als 64 Bitzahl angehängt. Diese beiden Schritte garantieren, daß die Länge der Nachricht M ein Vielfaches von 512 Bit ist und gewährleisten, daß zwei verschiedene Klartexte nach dem Padding nicht dieselbe Nachricht M ergeben.

Bei SHA wird eine Sequenz von 80 Funktionen $f_0, f_1, \ldots, f_{79}$ verwendet. Jede Funktion f_t, $0 \leq t \leq 79$ erhält drei 32 Bit lange Worte X, Y, Z als Eingabe und liefert ein 32 Bit langes Wort $f_t(X, Y, Z)$ als Ausgabe. Die Funktionen sind wie folgt definiert.

$$\begin{array}{ll}
f_t(X,Y,Z) = (X \wedge Y) \vee ((\neg X) \wedge Z) & 0 \leq t \leq 19 \\
f_t(X,Y,Z) = X \text{ XOR } Y \text{ XOR } Z & 20 \leq t \leq 39 \\
f_t(X,Y,Z) = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z) & 40 \leq t \leq 59 \\
f_t(X,Y,Z) = X \text{ XOR } Y \text{ XOR } Z & 60 \leq t \leq 79
\end{array}$$

Es werden die folgenden vier, in Hexadezimalschreibweise angegebenen Konstanten, bei der Anwendung der Funktionen benötigt.

$$\begin{array}{lllllll}
K_t = & \text{5A} & \text{82} & \text{79} & \text{99} & & 0 \leq t \leq 19 \\
K_t = & \text{6E} & \text{D9} & \text{EB} & \text{A1} & & 20 \leq t \leq 39 \\
K_t = & \text{8F} & \text{1B} & \text{BC} & \text{DC} & & 40 \leq t \leq 59 \\
K_t = & \text{CA} & \text{62} & \text{C1} & \text{D6} & & 60 \leq t \leq 79
\end{array}$$

Zur Berechnung des MAC wird die Nachricht M verwendet. Zuerst werden fünf 32 Bit lange Variablen wie folgt initialisiert.

$H_0 =$	67	45	23	01
$H_1 =$	EF	CD	AB	89
$H_2 =$	98	BA	DC	FE
$H_3 =$	10	32	54	76
$H_4 =$	C3	D2	E1	F0

Dann beginnt der Hauptteil des Algorithmus, der auf den jeweils 512 Bit langen Blöcken $M_1, M_2, \ldots, M_n$ von M ausgeführt wird. $S^i(X)$ bezeichnet dabei einen zyklischen Linksshift von X um i Bits und $+$ kennzeichnet die Addition modulo 2^{32}.
Auf einem Block M_i werden folgende Operationen ausgeführt.

1. M_i wird in 16 Worte $W_0, W_1, \ldots, W_{15}$ aufgeteilt
2. **for** $t = 16$ **to** 79 **do** $W_t = S^1(W_{t-3}$ XOR W_{t-8} XOR W_{t-14} XOR $W_{t-16})$ **od**
3. $A = H_0$, $B = H_1$, $C = H_2$, $D = H_3$, $E = H_4$
4. **for** $t = 0$ **to** 79
   do
   TEMP $= S^5(A) + f_t(B, C, D) + E + W_t + K_t$
   $E = D$
   $D = C$
   $C = S^{30}(B)$
   $B = A$
   $A =$ TEMP
   od
5. $H_0 = H_0 + A$
   $H_1 = H_1 + B$
   $H_2 = H_2 + C$
   $H_3 = H_3 + D$
   $H_4 = H_4 + E$

Nachdem der Algorithmus auf dem letzten Block M_n ausgeführt wurde, repräsentiert die Konkatenation von H_0, H_1, H_2, H_3 und H_4 den 160 Bit langen Ergebnis–Hash–Wert.

Literaturverzeichnis

[Ada97a] Adams, R.: The ESP Blowfish-CBC Algorithm Using an Explicit IV. IAB, June 1997. (Internet Draft). gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-ipsec-ciph-blowfish-cbc-00.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-ipsec-ciph-blowfish-cbc-00.txt

[Ada97b] Adams, R.: The ESP IDEA-CBC Algorithm Using Explicit IV. IAB, June 1997. (Internet Draft). gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-ipsec-ciph-idea-cbc-00.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-ipsec-ciph-idea-cbc-00.txt

[ASZ96] Atkins, D.; Stallings, W.; Zimmermann, P.: RFC 1991: PGP message exchange formats. IAB, August 1996. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1991.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1900-1999/rfc1991

[Atk95a] Atkinson, R.: RFC 1825: Security architecture for the Internet Protocol. IAB, August 1995. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1825.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1800-1899/rfc1825

[Atk95b] Atkinson, R.: RFC 1826: IP authentication header. IAB, August 1995. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1826.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1800-1899/rfc1826

[Atk95c] Atkinson, R.: RFC 1827: IP Encapsulating Security Payload (ESP). IAB, August 1995. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1827.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1800-1899/rfc1827

[Bal93] Balenson, D.: RFC 1423: Privacy enhancement for Internet electronic mail: Part III: Algorithms, modes, and identifiers. IAB, February 1993. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1423.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1400-1499/rfc1423

[Bau94] Bauer, F. L.: Kryptologie: Methoden und Maximen. 2. Aufl. Springer-Verlag, Berlin, 1994.

[BB-97] BB–Data: ASDIS – Heterogene DV–Welten fest im Griff; Verteilung, Verwaltung und Installation von Daten und Software in großen DV–Netzen. BB–Data Systemhaus, Berlin, März 1997. (Technischer Bericht)

[BCD97] Bossert, G.; Cooper, S.; Drummond, W.: RFC 2084: Considerations for Web transaction security. IAB, January 1997. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc2084.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-2000-2099/rfc2084

[BD95] Bradley, J.; Davies, N.: Analysis of the SSL Protocol. University of Bristol, June 28 1995. (Technical Report). http://www.cs.bris.ac.uk/~bradley/publish/SSLP/Appendix/SSLP-Analysis.ps.gz

[Bel89] Bellovin, S. M.: Security Problems in the TCP/IP Protocol Suite. – In: Computer Communication Review (ACM SIGCOMM): 19 (no.2), pp. 32–48, April 1989.

[BF93] Borenstein, N.; Freed, N.: RFC 1521: MIME (Multipurpose Internet Mail Extensions) part one: Mechanisms for specifying and describing the format of Internet message bodies. IAB, September 1993. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1521.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1500-1599/rfc1521

[BHH97] Boeyen, S.; Housley, R.; Howes, T.: Internet Public Key Infrastructure Part 2: Operational Protocols. IAB, March 1997. (Internet Draft). gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-pkix-ipki2opp-00.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-pkix-ipki2opp-00.txt

[BLC95] Berners-Lee, T.; Connolly, D.: RFC 1866: Hypertext Markup Language — 2.0. IAB, November 1995. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1866.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1800-1899/rfc1866

[BLFN96] Berners-Lee, T.; Fielding, R.; Nielsen, H.: RFC 1945: Hypertext transfer protocol — HTTP/1.0. IAB, May 1996. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1945.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1900-1999/rfc1945

[BLMM94] Berners-Lee, T.; Masinter, L.; McCahill, M.: RFC 1738: Uniform Resource Locators (URL). IAB, December 1994. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1738.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1700-1799/rfc1738

[Bor93] Borenstein, N.: RFC 1523: The text/enriched MIME content-type. IAB, September 1993. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1523.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1500-1599/rfc1523

[BS95] Borghoff, U. M.; Schlichter, J. H.: Rechnergestützte Gruppenarbeit; Eine Einführung in verteilte Anwendungen. Springer-Verlag, Berlin, 1995.

[Bun97] Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie: Gesetz zur Regelung der Rahmenbedingungen für Informations– und Kommunikationsdienste (Informations– und Kommunikationsdienste–Gesetz – IuKDG). Bonn, Juni 1997. (Deutscher Bundestag; BT–Drs. 13/7934). http://www.iid.de/rahmen/iukdgbt.html oder http://www.iid.de/rahmen/iukdgbt.pdf

[CB96] Cheswick, W. R.; Bellovin, S. M.: Firewalls und Sicherheit im Internet: Schutz vernetzter Systeme vor cleveren Hackern. Addison–Wesley, Bonn, 1996.

[CCI88a] CCITT: Data Communication Networks; Message Handling System; Recommendations X.400–X.420, vol. VIII.7. CCITT, Melbourne, November 1988. (Blue Book)

[CCI88b] CCITT: Recommendation X.208: Specification of Abstract Syntax Notation One (ASN.1). – In: Data Communication Networks; Open Systems Interconnection (OSI); Recommendations X.200–X.219. CCITT, Melbourne, 1988. (Blue Book; VIII.4)

[CCI88c] CCITT: Recommendation X.209: Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1). – In: Data Communication Networks; Open Systems Interconnection (OSI); Recommendations X.200–X.219. CCITT, Melbourne, 1988. (Blue Book; VIII.4)

[CCI88d] CCITT: Recommendation X.509: The Directory — Authentication Framework. – In: Data Communication Networks; Directory; Recommendations X.500–X.521. CCITT, Melbourne, 1988. (Blue Book; VIII.8)

[CF97] Chokhani, S.; Ford, W.: Internet Public Key Infrastructure Part IV: Certificate Policy and Certification Practices Framework. IAB, March 1997. (Internet Draft). gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-pkix-ipki-part4-00.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-pkix-ipki-part4-00.txt

[CFGM95] Crocker, S.; Freed, N.; Galvin, J.; Murphy, S.: RFC 1848: MIME object security services. IAB, October 1995. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1848.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1800-1899/rfc1848

[CG97a] Carrel, D.; Grant, L.: The TACACS+ Protocol. IAB, July 1997. (Internet Draft). gopher://ds0.internic.net:70/00/internet-drafts/draft-grant-tacacs-01.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-grant-tacacs-01.txt

[CG97b] Chang, S.; Glenn, R.: HMAC-SHA-1-96 IP Authentication with Replay Prevention. IAB, March 1997. (Internet Draft). gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-ipsec-ah-hmac-sha-1-96-00.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-ipsec-ah-hmac-sha-1-96-00.txt

[CLR90] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.: Introduction to Algorithms. The MIT Press, Cambridge, 1990.

[Cri94] Crispin, M.: RFC 1730: Internet Message Access Protocol — VERSION 4. IAB, December 1994. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1730.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1700-1799/rfc1730

[Cro82] Crocker, D.: RFC 822: Standard for the format of ARPA Internet text messages. IAB, August 1982. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc0822.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-0800-0899/rfc0822

[DA97] Dierks, T.; Allen, C.: The TLS Protocol Version 1.0. IAB, May 1997. (Internet Draft). gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-tls-protocol-03.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-tls-protocol-03.txt

[DH76] Diffie, W.; Hellman, M. E.: New Directions in Cryptography. – In: IEEE Transactions on Information Theory: IT-22 (no.6), pp. 644–654, November 1976.

[DH96] Deering, S.; Hinden, R.: RFC 1883: Internet protocol, version 6 (IPv6) specification. IAB, January 1996. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1883.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1800-1899/rfc1883

[DHR97] Dusse, S.; Hoffman, P.; Ramsdell, B.: S/MIME Message Specification. IAB, May 1997. (Internet Draft). gopher://ds0.internic.net:70/00/internet-drafts/draft-dusse-smime-msg-01.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-dusse-smime-msg-01.txt

[Eck96] Eckert, C.: Skript zur Vorlesung Sichere Rechensysteme im WS 1995/96. Fakultät für Informatik, Technische Universität München, 1996.

[ECS94] Eastlake, D.; Crocker, S.; Schiller, J.: RFC 1750: Randomness recommendations for security. IAB, December 1994. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1750.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1700-1799/rfc1750

[Elk96] Elkins, M.: RFC 2015: MIME security with Pretty Good Privacy (PGP). IAB, October 1996. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc2015.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-2000-2099/rfc2015

[FA97] Farrell, S.; Adams, C.: Internet Public Key Infrastructure Part III: Certificate Management Protocols. IAB, June 1997. (Internet Draft). gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-pkix-ipki3cmp-02.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-pkix-ipki3cmp-02.txt

[FGM+97] Fielding, R.; Gettys, J.; Mogul, J.; Frystyk, H.; Berners-Lee, T. et al.: RFC 2068: Hypertext transfer protocol — HTTP/1.1. IAB, January 1997. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc2068.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-2000-2099/rfc2068

[Fin93] Finseth, C.: RFC 1492: An Access Control Protocol, sometimes called TACACS. IAB, July 1993. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1492.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1400-1499/rfc1492

[FKK96] Freier, A.; Karlton, P.; Kocher, P.: The SSL Protocol Version 3.0. IAB, November 1996. (Internet Draft). gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-tls-ssl-version3-00.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-tls-ssl-version3-00.txt

[Fri91] Frisch, Æ.: Essential System Administration. O'Reilly & Assosiates, Inc., Sebastopol, 1991.

[GN96] Gilligan, R.; Nordmark, E.: RFC 1933: Transition mechanisms for IPv6 hosts and routers. IAB, April 1996. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1933.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1900-1999/rfc1933

[GS91] Garfinkel, S.; Spafford, G.: Practical Unix Security. O'Reilly & Assosiates, Inc., Sebastopol, 1991.

[Gul88] Gulbins, J.: UNIX; Eine Einführung in Begriffe und Kommandos von UNIX. Dritte Aufl. Springer-Verlag, Berlin, 1988.

[HA93] Hegering, H.-G.; Abeck, S.: Integriertes Netz– und Systemmanagement. Addison–Wesley, Bonn, 1993.

[Hal95] Haller, N.: RFC 1760: The S/KEY one-time password system. IAB, February 1995. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1760.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1700-1799/rfc1760

[Hau95] Hauck, R.: Sicherheitskonzept für den BMW–Internet–Zugang. Diplomarbeit, Technische Universität München, München, 1995.

[HC97] Harkins, D.; Carrel, D.: The resolution of ISAKMP with Oakley. IAB, February 1997. (Internet Draft). gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-ipsec-isakmp-oakley-03.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-ipsec-isakmp-oakley-03.txt

[HD96] Hinden, R.; Deering, S.: RFC 1884: IP Version 6 addressing architecture. IAB, January 1996. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1884.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1800-1899/rfc1884

[HFP97] Housley, R.; Ford, W.; Polk, T.: Internet Public Key Infrastructure Part I: X.509 Certificate and CRL Profile. IAB, March 1997. (Internet Draft). gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-pkix-ipki-part1-04.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-pkix-ipki-part1-04.txt

[Hor97] Horowitz, M.: Key Derivation for Authentication, Integrity, and Privacy. IAB, March 1997. (Internet Draft). gopher://ds0.internic.net:70/00/internet-drafts/draft-horowitz-key-derivation-01.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-horowitz-key-derivation-01.txt

[HPV+96] Hamzeh, K.; Pall, G. S.; Verthein, W.; Taarud, J.; Little, W. A.: Point–to–Point Tunneling Protocol — PPTP. IAB, June 1996. (Internet Draft)

[HR91] Holbrook, P.; Reynolds, J.: RFC 1244: Site security handbook. IAB, July 1991. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1244.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1200-1299/rfc1244

[HT97] Hohenegg, C.; Tauscheck, S.: Rechtliche Problematik digitaler Signaturverfahren. – In: Betriebs–Berater; Zeitschrift für Recht und Wirtschaft: 52 (Nr.31), S. 1541–1548, 1997.

[Jun95] Jungfleisch, N. Zugangsregelung im BMW–Netz. Unveröffentlicht, 1995.

[Jun96] Jungfleisch, N. Sicherheitspolitik – Zugangskontrolle. Unveröffentlicht, 1996.

[Kal92] Kaliski, B.: RFC 1319: The MD2 Message-Digest Algorithm. IAB, April 1992. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1319.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1300-1399/rfc1319

[Kal93a] Kaliski, B.: An Overview of the PKCS Standards. RSA Laboratories, Redwood City, November 1993. (Technical Report). http://www.rsa.com/pub/pkcs

[Kal93b] Kaliski, B.: RFC 1424: Privacy enhancement for Internet electronic mail: Part IV: Key certification and related services. IAB, February 1993. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1424.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1400-1499/rfc1424

[KBC97] Krawczyk, H.; Bellare, M.; Canetti, R.: RFC 2104: HMAC: Keyed-hashing for message authentication. IAB, February 1997. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc2104.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-2100-2199/rfc2104

[Ken93] Kent, S.: RFC 1422: Privacy enhancement for Internet electronic mail: Part II: Certificate-based key management. IAB, February 1993. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1422.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1400-1499/rfc1422

[KN93] Kohl, J.; Neuman, B.: RFC 1510: The Kerberos Network Authentication Service (V5). IAB, September 1993. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1510.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1500-1599/rfc1510

[KSH97] Kikuchi, H.; Sakurai, M.; Hattori, H.: Internet Public Key Infrastructure: Web-based Certificate and CRL Repository. IAB, March 1997. (Internet Draft). gopher://ds0.internic.net:70/00/internet-drafts/draft-kikuchi-web-cert-repository-00.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-kikuchi-web-cert-repository-00.txt

[Lai92] Lai, X.: On the Design and Security of Block Ciphers. PhD thesis, ETH, Zürich, 1992. Diss. ETH No. 9752.

[Lin93] Linn, J.: RFC 1421: Privacy enhancement for Internet electronic mail: Part I: Message encryption and authentication procedures. IAB, February 1993. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1421.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1400-1499/rfc1421

[LL93] Lenstra, Arjen K.; Lenstra, Hendrik W. (Eds.): The Development of the Number Field Sieve. Springer-Verlag, Berlin, 1993. (Springer Lecture Notes in Mathematics; 1554)

[LLMP90] Lenstra, A. K.; Lenstra, H. W.; Manasse, M. S.; Pollard, J. M.: The Number Field Sieve. – In: Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing, Baltimore, Maryland, May 14–16, 1990. Ed.: ACM SIGACT. ACM Press, Baltimore, 1990, pp. 564–572.

[LM90] Lai, X.; Massey, J. L.: A Proposal for a New Block Encryption Standard. – In: Advances in Cryptology – EUROCRYPT '90, Aarhus, Denmark, May 21–24, 1990. Ed.: I. B: Damgård. Springer-Verlag, Berlin, 1990, pp. 389–404. (Lecture Notes in Computer Science; 473)

[LS92] Lloyd, B.; Simpson, W.: RFC 1334: PPP authentication protocols. IAB, October 1992. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1334.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1300-1399/rfc1334

[Mil76] Miller, G. L.: Riemann's Hypothesis and Tests for Primality. – In: Journal of Computer and System Sciences: 13 (no.3), pp. 300–317, 1976.

[MKS95] Metzger, P.; Karn, P.; Simpson, W.: RFC 1829: The ESP DES-CBC transform. IAB, August 1995. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1829.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1800-1899/rfc1829

[Moo93] Moore, K.: RFC 1522: MIME (Multipurpose Internet Mail Extensions) part two: Message header extensions for non-ASCII text. IAB, September 1993. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1522.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1500-1599/rfc1522

[MR94] Myers, J.; Rose, M.: RFC 1725: Post Office Protocol — version 3. IAB, November 1994. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1725.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1700-1799/rfc1725

[MS95a] Metzger, P.; Simpson, W.: RFC 1828: IP authentication using keyed MD5. IAB, August 1995. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1828.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1800-1899/rfc1828

[MS95b] Metzger, P.; Simpson, W.: RFC 1852: IP authentication using keyed SHA. IAB, October 1995. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1852.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1800-1899/rfc1852

[MSS97] Maughan, D.; Schertler, M.; Schneider, M.: Internet Security Association and Key Management Protocol (ISAKMP). IAB, February 1997. (Internet Draft). gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-ipsec-isakmp-07.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-ipsec-isakmp-07.txt

[Mye97] Myers, J.: Simple Authentication and Security Layer (SASL). IAB, May 1997. (Internet Draft). gopher://ds0.internic.net:70/00/internet-drafts/draft-myers-auth-sasl-11.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-myers-auth-sasl-11.txt

[Nat80] National Institute of Standards and Technology (NIST): DES Modes of Operation. U.S. Department of Commerce, Gaithersburg, December 1980. (Federal Information Processing Standards Publication; FIPS-PUB 81). http://www.itl.nist.gov/div897/pubs/fip81.htm

[Nat93] National Institute of Standards and Technology (NIST): Data Encryption Standard (DES). U.S. Department of Commerce, Gaithersburg, December 1993. (Federal Information Processing Standards Publication; FIPS-PUB 46-2). http://www.itl.nist.gov/div897/pubs/fip46-2.htm

[Nat94] National Institute of Standards and Technology (NIST): Digital Signature Standard (DSS). U.S. Department of Commerce, Gaithersburg, May 1994. (Federal Information Processing Standards Publication; FIPS-PUB 186). http://www.itl.nist.gov/div897/pubs/fip186.htm

[Nat95] National Institute of Standards and Technology (NIST): Secure Hash Standard (SHA). U.S. Department of Commerce, Gaithersburg, April 1995. (Federal Information Processing Standards Publication; FIPS-PUB 180-1). http://www.itl.nist.gov/div897/pubs/fip180-1.htm

[Nat96] National Security Agency : Fortezza Program Overview. National Security Agency (NSA), February 1996. (Technical Report; Version 4.0a). http://www.armadillo.huntsville.al.us/Fortezza_docs/basic.html

[NM95] Nebel, E.; Masinter, L.: RFC 1867: Form-based file upload in HTML. IAB, November 1995. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1867.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1800-1899/rfc1867

[Odl95] Odlyzko, A. M.: The Future of Integer Factorization. – In: CryptoBytes: 1 (no.2), pp. 5–12, 1995. http://www.rsa.com/rsalabs/pubs/cryptobytes/crypto1n2.pdf.

[OG97] Oehler, M.; Glenn, R.: HMAC-MD5-96 IP Authentication with Replay Prevention. IAB, March 1997. (Internet Draft). gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-ipsec-ah-hmac-md5-96-00.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-ipsec-ah-hmac-md5-96-00.txt

[O.V96] O.V.: RSA-130 Factored. – In: CryptoBytes: 2 (no.2), p. 7, 1996. http://www.rsa.com/rsalabs/pubs/cryptobytes/crypto2n2.pdf.

[PKC95] PKCS Steering Group: PKCS Security Services for MIME ("S/MIME"). RSA Data Security, Inc., Redwood City, April 1995. (Technical Report). http://www.rsa.com/pub/pkcs

[Pos80] Postel, J.: RFC 768: User Datagram Protocol. IAB, August 1980. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc0768.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-0700-0799/rfc0768

[Pos81a] Postel, J.: RFC 791: Internet protocol. IAB, September 1981. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc0791.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-0700-0799/rfc0791

[Pos81b] Postel, J.: RFC 793: Transmission Control Protocol. IAB, September 1981. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc0793.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-0700-0799/rfc0793

[Pos82] Postel, J.: RFC 821: Simple Mail Transfer Protocol. IAB, August 1982. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc0821.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-0800-0899/rfc0821

[PR85] Postel, J.; Reynolds, J.: RFC 959: File Transfer Protocol. IAB, October 1985. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc0959.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-0900-0999/rfc0959

[PT97] Pereira, R.; Thayer, R.: The ESP 3DES-CBC Algorithm Using an Explicit IV. IAB, July 1997. (Internet Draft). gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-ipsec-ciph-3des-expiv-00.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-ipsec-ciph-3des-expiv-00.txt

[Rab80] Rabin, M. O.: Probabilistic Algorithm for Testing Primality. – In: Journal of Number Theory: 12 (no.1), pp. 128–138, 1980.

[Rad97] Radisic, I.: Aufbau einer Testumgebung für die sichere TCP/IP–basierte Kommunikation. Institut für Informatik der Ludwig–Maximilians Universität, München, 1997. (Fortgeschrittenenpraktikum)

[Rig97] Rigney, C.: RFC 2059: RADIUS accounting. IAB, January 1997. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc2059.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-2000-2099/rfc2059

[Riv92] Rivest, R.: RFC 1321: The MD5 Message-Digest Algorithm. IAB, April 1992. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1321.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1300-1399/rfc1321

[Rog96] Rogaway, P.: The Security of DESX. – In: CryptoBytes: 2 (no.2), pp. 8–11, 1996. http://www.rsa.com/rsalabs/pubs/cryptobytes/crypto2n2.pdf.

[RS97a] Rescorla, E.; Schiffman, A.: Security Extensions For HTML. IAB, March 1997. (Internet Draft). gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-wts-shtml-03.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-wts-shtml-03.txt

[RS97b] Rescorla, E.; Schiffman, A.: The Secure HyperText Transfer Protocol. IAB, March 1997. (Internet Draft). gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-wts-shttp-04.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-wts-shttp-04.txt

[RSA78] Rivest, R.; Shamir, A.; Adleman, L.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. – In: Communications of the ACM: 21 (no.2), pp. 120–126, 1978.

[RSA93a] RSA Laboratories: PKCS #1: RSA Encryption Standard. RSA Laboratories, Redwood City, November 1993. (Technical Report). http://www.rsa.com/pub/pkcs

[RSA93b] RSA Laboratories: PKCS #10: Certification Request Syntax Standard. RSA Laboratories, Redwood City, November 1993. (Technical Report). http://www.rsa.com/pub/pkcs

[RSA93c] RSA Laboratories: PKCS #7: Cryptographic Message Syntax Standard. RSA Laboratories, Redwood City, November 1993. (Technical Report). http://www.rsa.com/pub/pkcs

[RSA93d] RSA Laboratories: PKCS #9: Selected Attribute Types. RSA Laboratories, Redwood City, November 1993. (Technical Report). http://www.rsa.com/pub/pkcs

[SB97] Simpson, W. A.; Baldwin, R.: The ESP DES-XEX3-CBC Transform. IAB, July 1997. (Internet Draft). gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-ipsec-ciph-desx-00.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-ipsec-ciph-desx-00.txt

[Sch93] Schneier, B.: Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish). – In: Fast Software Encryption, Cambridge Security Workshop, Cambridge, December 9-11, 1993. Ed.: R. Anderson. Springer-Verlag, Berlin, 1993, pp. 191–204. (Lecture Notes in Computer Science; 809). http://www.counterpane.com/bfsverlag.html

[Sch94] Schneier, B.: Applied Cryptography: Protocols, Algorithms and Source Code in C. John Wiley & Sons, Inc., New York, 1994.

[SG90] Scheifler, R. W.; Gettys, J.: X Window System: The complete reference to X lib, X Protocol, ICCCM, XFLD. 2nd edition. Digital Press, Bedford, 1990.

[Sha95] Shamir, A.: RSA for Paranoids. – In: CryptoBytes: 1 (no.3), pp. 1–4, 1995. http://www.rsa.com/rsalabs/pubs/cryptobytes/crypto1n3.pdf.

[Sim94] Simpson, W.: RFC 1661: The Point-to-Point Protocol (PPP). IAB, July 1994. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1661.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1600-1699/rfc1661

[Sim95] Simpson, W.: RFC 1853: IP in IP tunneling. IAB, October 1995. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1853.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1800-1899/rfc1853

[Sim96] Simpson, W.: RFC 1994: PPP Challenge Handshake Authentication Protocol (CHAP). IAB, August 1996. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc1994.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-1900-1999/rfc1994

[Ste95] Stevens, W. R.: Programmierung in der Unix–Umgebung. Addison–Wesley, Bonn, 1995.

[Tha97] Thayer, R.: The ESP ARCFOUR Algorithm. IAB, June 1997. (Internet Draft). gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-ipsec-ciph-arcfour-00.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-ipsec-ciph-arcfour-00.txt

[Ven92] Venema, W.: TCP WRAPPER: Network monitoring, access control and booby traps. Baltimore, September 1992, pp. 85–92. (Proceedings of the Third Usenix UNIX Security Symposium). ftp://ftp.win.tue.nl/pub/security/tcp_wrapper.ps.Z

[WRSR97] Willens, S.; Rubens, A.; Simpson, W.; Rigney, C.: RFC 2058: Remote Authentication Dial In User Service (RADIUS). IAB, January 1997. (Technical Report). http://rfc.fh-koeln.de/rfc/html/rfc2058.html oder ftp://ftp.leo.org/pub/comp/doc/standards/rfc/rfc-2000-2099/rfc2058

[Ylo95] Ylonen, T.: The SSH (Secure Shell) Remote Login Protocol. Helsinki University of Technology, Helsinki, 15. November 1995. (Technical Report). http://www.cs.hut.fi/ssh/RFC

[Ylo97a] Ylonen, T.: SSH Authentication Protocol. IAB, March 1997. (Internet Draft). gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-secsh-userauth-00.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-secsh-userauth-00.txt

[Ylo97b] Ylonen, T.: SSH Connection Protocol. IAB, March 1997. (Internet Draft). gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-secsh-connect-00.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-secsh-connect-00.txt

[Ylo97c] Ylonen, T.: SSH Transport Layer Protocol. IAB, March 1997. (Internet Draft). gopher://ds0.internic.net:70/00/internet-drafts/draft-ietf-secsh-transport-00.txt oder http://info.internet.isi.edu:/0/in-drafts/files/draft-ietf-secsh-transport-00.txt

[Zim94a] Zimmermann, P.: PGP User's Guide Volume I: Essential Topics. Phil's Pretty Good Software, Boulder, October 1994. (Technical Report)

[Zim94b] Zimmermann, P.: PGP User's Guide Volume II: Special Topics. Phil's Pretty Good Software, Boulder, October 1994. (Technical Report)

Index