Modul

Netzsicherheit 2

Studienbrief 1: TCP/UDP

Studienbrief 2: SSL

Studienbrief 3: SSH

Studienbrief 4: PGP

Studienbrief 5: S/MIME

Studienbrief 6: DNSSEC

Autor:

Prof. Dr. Jörg Schwenk

---

1. Auflage

Ruhr - Universität Bochum

Ruhr - Universität Bochum
Universitätsstraße 150
44801 Bochum

1. Auflage (27. März 2014)

Didaktische und redaktionelle Bearbeitung sowie Produktion:
Jörg Schwenk & Christoph Bader

Inhaltsverzeichnis

Einleitung zu den Studienbriefen

I. Abkürzungen der Randsymbole und Farbkodierungen

Axiom	A
Beispiel	B
Definition	D
Exkurs	E
Kontrollaufgabe	K
Merksatz	M
Quelle	Q
Satz	S
Übung	Ü

II. Zu den Autoren

Prof. Jörg Schwenk leitet den Lehrstuhl Netz- und Datensicherheit an der Ruhr-Uni Bochum seit dem Jahr 2003. Von 1993 bis 2001 arbeitete er im Bereich Sicherheit der Deutschen Telekom in verschiedenen Industrieprojekten. Anschließend lehrte er zwei Jahre lang an der Georg-Simon-Ohm FH Nürnberg. Er hat mehr als 60 Patente und mehr als 40 wissenschaftliche Publikationen verfasst. Seine Forschungsinteressen umfassen kryptographische Protokolle (SSL/TLS, IPSec), XML- und Webservice-Security, sowie Web- und Internetsicherheit.

III. Modullehrziele

Die Kryptographie war lange Zeit eine Wissenschaft für Spezialisten. Bis in die zweite Hälfte des letzten Jahrhunderts beschäftigten sich mit ihr nur Militärs und Diplomaten. Mit dem Aufkommen der elektronischen Datenverarbeitung und der digitalen Kommunikation kamen weitere Spezialisten hinzu: Bankangestellte, Datenschützer, Mobilfunker, Pay-TV-Anbieter und auch die ersten Hacker.
Mit dem Erfolg des Internet änderte sich die Situation grundlegend. Jetzt hatte jeder die Gelegenheit, persönliche Nachrichten per E-Mail zu versenden, oder Waren mit einem Mausklick im World Wide Web zu kaufen.
Gleichzeitig wuchs das Bewusstsein, wie unsicher das neue Medium ist: Durch Abhörpläne der nationalen Regierungen (die ähnlich wie beim Brief- und Fernmeldegeheimnis auch hier Einschränkungen im Rahmen der Verbrechensbekämpfung durchsetzen wollten) und durch kriminelle Aktivitäten (wie z.B. den Diebstahl von Kreditkartennummern von Webservern).

Zum Glück wurde im Jahr 1976 die Public Key-Kryptographie erfunden, die sich bestens für das offene Internet eignet. Die erste Implementierung des bekanntesten Public Key-Verfahrens RSA wurde unter dem Namen „Pretty Good Privacy (PGP)“ bekannt und begann, sich unter den Internet-Nutzern zu verbreiten. Damit begann der Siegeszug der Kryptographie im Internet: „Privacy Enhanced Mail“ und „Secure MIME“ für E-Mail, sowie SSL für das World Wide Web folgten. Schließlich wurde das Internet-Protokoll IP selbst mit den verschiedenen IPSec-Standards abgesichert.

Ziel dieses Moduls ist es, dem Leser eine fundierte Einführung in verschiedene Gebiete der Netzsicherheit zu geben. Damit soll er in die Lage versetzt werden, sich anschließend mit geringer Mühe in den Referenzdokumenten zurechtzufinden und mit ihrer Hilfe das passende Sicherheitssystem zu konzipieren oder zu implementieren.
Diesem Ziel dienen auch die zahlreichen Abbildungen, mit denen jeweils versucht wird, das im Text Gesagte (oder zu Ergänzende) graphisch möglichst anschaulich und einprägsam darzustellen.

Das Modul ist streng praxisorientiert, d.h. es wird nur die heute im Internet tatsächlich eingesetzte Kryptographie behandelt.
Zum didaktischen Konzept des Moduls gehören auch die angeführten erfolgreichen Angriffe auf Sicherheitsstandards (PGP, SSL, SSH): Aus den Fehlern anderer Standards kann man lernen. Diese Darstellungen sollen also nicht dazu verleiten, eigene Angriffe auf bestehende Infrastrukturen zu starten, sondern der Leser soll sie beim Entwurf eigener Konzepte stets im Hinterkopf behalten.

Studienbrief 1 TCP/UDP

1.1 Lernziele

Sie haben die Aufgabe und Funktion von Transportprotokollen verstanden. Sie kennen Eigenschaften, die ein Transportprotokoll, je nach Anwendung, haben sollte.

Speziell lernen sie die in der Praxis häufig verwendeten Protokolle TCP und UDP, deren Vor - und Nachteile, sowie deren Anwendungen kennen.

1.2 Advanced Organizer

Nahezu alle Daten im Internet werden heutzutage über die Transportschichtprotokolle TCP und UDP übertragen. Diese Protokolle stellen eine logischen Kommunikationskanal zwischen Prozessen auf verschiedenen Hosts zur Verfügung. Die in den nachfolgenden Kapiteln behandelten Anwendungen laufen alle über eines dieser beiden Protokolle.
In diesem Studienbrief erhalten sie eine knappe Einführung in den Aufbau von TCP-, bzw. UDP-Segmenten. Sie lernen Aufgaben, sowie Eigenschaften von Transportschichtprotokollen kennen und verstehen, wie TCP zuverlässigen Datentransfer realisiert.

1.3 Das Internet Protokoll IP

Das „Internet Protocol“ (IP) bildet die Grundlage für den folgenden Studienbrief. Wir werden daher im Folgenden zunächst knapp auf dieses Protokoll eingehen. Detailliertere Grundlagen über IP können in Schwenk [2010] und, etwas detaillierter, in Comer [2003] nachgelesen werden. IP ist in der Vermittlungsschit des OSI - Schichtenmodell angesiedelt.

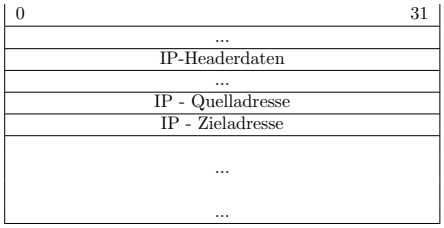


Abb. 1.1: Schematischer Aufbau eines IP - Pakets.

Der grundlegende Aufbau eines IP-Pakets ist stets gleich: Auf einen Header folgen die sogenannten Nutzdaten. Der Header enthält Informationen (u.a.) über die Länge des nachfolgenden Datenpakets und darüber wie die Daten zu interpretieren sind. Die wichtigsten Headerfelder für unsere Zwecke sind das Quell - und das Zieladressfeld des Pakets. In diesen Feldern stehen die IP - Adressen des Senders, bzw. des Empfängers. Mit der IP - Adresse lässt sich ein Host in einem Netzwerk (z.B. im Internet) eindeutig identifizieren. Das Paket wird an den Host „geliefert“, der der Zieladresse zugeordnet ist. Die Angabe der Quelladresse ist wichtig, denn so können etwaige Antworten auch richtig adressiert werden. Ein weiteres Feld (das TTL-Feld) im IP- Header gibt an, nach welcher Zeit das Paket verworfen werden kann, falls es bis dahin noch nicht zugestellt worden ist.

Im Datenteil überträgt ein IP-Paket Protokollpakete höherer Schichten des OSI-Schichtenmodells, etwa TCP- oder UDP-Pakete. Das verwendete Protokoll wird in einem speziellen Feld des IP-Headers angegeben.
Es stellt sich sofort die Frage, wie die Daten eines IP - Pakets nach erfolgreicher Zustellung vom Client/Server verarbeitet werden, bzw. welches Programm dafür zuständig ist? Diese Informationen sind in einem IP - Header nicht zu finden. Die in diesem Abschnitt behandelten Protokolle UDP und TCP beantworten diese Fragen. Weiterhin werden wir mit TCP ein verbindungsorientiertes Protokoll einführen, welches Daten verlässlich transportiert. Diese Protokolle sind in der Transportschicht angesiedelt und setzen direkt auf der Vermittlungs - bzw. IP - Schicht auf, siehe Abbildung 1.2.

7. Anwendungsschicht	Anwendungsschicht
6. Darstellungsschicht	
5. Sitzungsschicht	
4. Transportschicht	Transportschicht
3. Vermittlungsschicht	IP - Schicht
2. Sicherungsschicht	Netzzugangs-
1. Bitübertragungsschicht	schicht

Abb. 1.2: Einordnung der Protokolle TCP und UDP in das OSI Schichtenmodell, bzw. in das TCP/IP Referenzmodell

Wir werden zunächst das *verbindungslose*, *zustandslose* Protokoll UDP erläutern und diskutieren und im Anschluss mit TCP ein *verlässliches*, *verbindungsorientiertes* Protokoll einführen.

1.4 UDP

1.4.1 UDP - Paket

Eine UDP - Nachricht wird auch als (UDP-)Paket bezeichnet. Ein solches ist wie folgt aufgebaut: Einem Body (eigentlicher Inhalt des Pakets) ist ein Header vorangestellt. Abbildung 1.3 stellt den Aufbau eines Pakets schematisch dar. Den Aufbau dieser beiden Bestandteile werden wir im Folgenden näher betrachten.

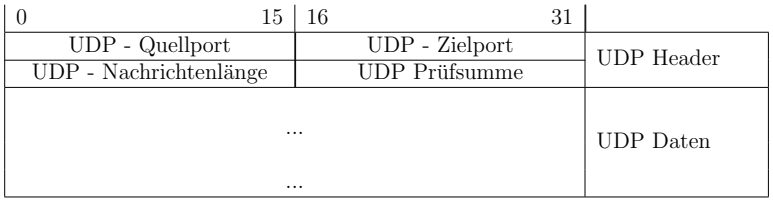


Abb. 1.3: Aufbau eines UDP Pakets, schematisch

Header

Wir schauen uns nun die einzelnen Bestandteile des Headers eines Pakets an. Der Header enthält zuerst zwei 16 - bit (=2 Byte) Zahlen, die Quell - und Zielport angeben. Es folgt eine 16 - bit Zahl, die die gesamte Paketlänge (inkl. Header) in Bytes angibt. Abschließend ist im Header eine 16-bit Prüfsumme zu finden.
Der Header eines IP - Pakets enthält die Quell - und Zieladresse dieses Pakets in Form der IP Adresse des Absenders und des Empfängers. Diese Informationen reichen bei einem Paket nicht aus: UDP ist ein „high - level protocol“, d.h. ein Paket ist an einen bestimmten Prozess, bzw. an einen bestimmten Dienst auf dem

Server gerichtet. Dieser Prozess, bzw. dieser Dienst, muss in der Zieladresse genannt werden, damit das ankommende Paket an den richtigen Prozess weitergeleitet werden kann. Es ist oftmals jedoch nicht bekannt, welcher Prozess auf dem Server einen bestimmten Dienst bereitstellt. Weiterhin sind Prozesse dynamisch, d.h. sie können sich auf - und abbauen. Deshalb ist es ungünstig ein Paket direkt

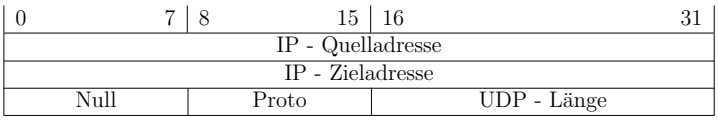
Ports

an einen Prozess zu adressieren. Ein Paket ist daher an einen sogenannten *Port* adressiert. Durch die Addressierung an Ports werden Datenpakete vom Betriebssystem Programmen zugeordnet. Die Portnummer ist Teil der Netzwerkadresse. Jedem Port ist ein Dienst oder eine Funktion zugeordnet. Wenn der Absender eines Pakets einen bestimmten Dienst auf einem Server ansprechen möchte, genügt es nun, wenn er die zugehörige Portnummer kennt. Er muss nicht wissen, welcher Prozess auf dem Server die gewünschte Funktion ausführt. Die Zuordnung: Dienst $\rightarrow$ Portnummer wird vom lokalen Betriebssystem (des Servers) bereitgestellt. Viele Portnummern sind auch reserviert, so spricht z.B. die Portnummer 21 ein file-transfer-protocol - (ftp) Programm an oder die Portnummer 80 ein HTTP - Programm.
Die Felder Quell - bzw. Zielport im UDP - Header enthalten genau diese Information. Natürlich muss der Quellport mitgenannt werden, um evtl. Antworten

Socket

des Empfängers richtig adressieren zu können. Der Begriff *Socket* bezeichnet die Kombination aus IP - Adresse und Portnummer, die einen Dienst eindeutig identifiziert.
Die anschließende (optionale) Prüfsumme dient dazu Übertragungsfehler im Header zu erkennen. Für dessen Berechnung wird zunächst ein *UDP - Pseudoheader* erstellt.

Abb. 1.4: Aufbau des UDP Pseudoheader



Der Pseudoheader enthält auch die IP - Quell - und Zieladresse (je 32 - bit lang). Dies ist aus dem Grund sinnvoll, dass ein Paket an einen bestimmten Dienst (über die Portnummer identifiziert) auf einem speziellen Rechner (über die IP - Adresse identifiziert) adressiert ist. Im Feld Proto (8 bit lang) steht ein Typencode für das Transportprotokoll (17 für UDP). Das UDP - Längenfeld gibt die Länge des Pakets (ohne Pseudeheader) an. Das „Null“ - Feld sorgt lediglich dafür, dass die Länge des Pseudoheaders ein Vielfaches von 16 bit hat. Dieser Pseudoheader wird dem Paket vorangestellt. Anschließend wird nach einem festgelegten Algorithmus die UDP - Prüfsumme berechnet. Der Pseudoheader wird nicht versendet. Zum Überprüfen des Pakets erstellt der Empfänger seinerseits den Pseudoheader, berechnet die Prüfsumme und vergleicht diese mit der Prüfsumme im empfangenen Paket.

K

Kontrollaufgabe 1.1

Was ist ein Port?
Wieso werden Ports benötigt?
Wieviele Ports stellt ein Rechner sinnvollerweise maximal bereit?

Kontrollaufgabe 1.2

Was ist ein Socket?
Sockets werden auch Verbindungsendpuntke genannt - warum ist diese Bezeichnung sinnvoll?

K

Kontrollaufgabe 1.3

Welchen Zweck erfüllt die Prüfsumme im UDP - Header? Wie wird sie gebildet? Warum muss der sog. Pseudoheader nicht mitgeschickt werden, um die Prüfsumme zu überprüfen?

K

Body

Der Body eines Pakets enthält den eigentlichen Inhalt der Nachricht. Diese Informationen werden vom im Header über die Portnummer angesprochenen Dienst weiterverarbeitet.

Beispiel 1.1

Ein Browser lade über einen bestimmten Port Daten von einem Server herunter. Es kann nun wie folgt ein weiterer Download vom Server im selben Browser gestartet werden: Der Browser öffnet einen neuen Port (dies ist möglich, da Portnummern „virtuelle“ Adressen sind) und baut eine weitere Verbindung zum Server auf. Die Zuordnungen der gesandten Pakete zum jeweiligen Download ist mit Hilfe der Ziel - Portnummer im UDP Header einfach durchzuführen.

B

1.4.2 Eigenschaften von UDP/IP

UDP bietet keine Funktion zur Überprüfung des Status (Zustand) einer Zustellung an. Falls ein Paket während der Übertragung verloren geht, wird dies nicht bemerkt. Das Protokoll wird deshalb als *zustandslos* bezeichnet. Weiterhin wird jedes Paket im Zuge eines Downloads von einem Server zu einem Client (oder umgekehrt) unabhängig von den anderen, diesen Download betreffenden, Pakete gesendet. Es ist daher möglich, dass Pakete, aufgrund unterschiedlicher Länge der „Leitwege“, in der falschen Reihenfolge oder gar nicht beim Client ankommen. Man nennt UDP daher auch *verbindungslos*. Das Aufbauen einer Verbindung zwischen Client und Server wird in Abschnitt 1.5.2 erläutert.
Diese Eigenschaften mögen hinderlich erscheinen. In vielen Fällen ist es aber gar nicht notwendig, dass der Absender etwas darüber erfährt, ob das Paket angekommen ist. Insbesondere bei Echtzeit- Anwendungen ist dieses nicht nötig. Wird z.B. von einem Server ein Videoclip heruntergeladen, so benötigt der Server keine Information darüber, ob ein Paket angekommen ist und selbst im Falle eines nicht zugestellten Pakets, wird das Bild beim Empfänger nur kurz gestört. UDP wird außerdem meist als Transportprotokoll für Internettelefonie (**V**oice **o**ver **IP**) verwendet, siehe z.B.: [RFC3550].

Dennoch entstehen durch die genannten Eigenschaften auch Probleme: Wenn ein Paket nicht beim Client ankommt, so ist es möglich, dass es falsch geroutet wurde und schließlich verworfen wurde, weil der TTL Zähler im IP Header heruntergezählt ist. Es ist aber auch möglich, dass ein Angreifer (z.B. ein Router, über den die Pakete geleitet werden) Daten abfängt. Weil der Client per UDP den Server nicht darüber informieren kann, dass er nichts mehr empfängt, sendet der Server die Daten weiter. Außerdem ist es einem Angreifer möglich eigene Pakete in den Datentransfer zwischen Client und Server einzuschleusen. Hierzu muss er lediglich Pakete schicken, die als Quell - IP - Adresse und als Quellport die des Servers enthalten. Der Client kann nicht feststellen, dass die Pakete nicht vom Server stammen. Diese Technik wird als *IP Spoofing* bezeichnet.

IP Spoofing

K

Kontrollaufgabe 1.4

Welche Eigenschaften hat ein zustandsloses / verbindungsloses Protokoll?

K

Kontrollaufgabe 1.5

Erläutern Sie, wo UDP in der Praxis eingesetzt wird - warum?

K

Kontrollaufgabe 1.6

Was versteht man unter IP Spoofing?

1.5 TCP

Der vorherige Abschnitt hat das Transportprotokoll UDP behandelt. UDP ist ein zustandsloses, verbindungsloses Protokoll und daher können während des Datenaustauschs gewisse Probleme auftreten. In diesem Abschnitt werden wir das *Transmission Control Protocol* (TCP) einführen, ein *verlässliches*, *verbindungsorientiertes* Transportprotokoll. Mittels TCP wird eine virtuelle Verbindung zwischen Client und Server aufgebaut, die es beiden ermöglicht Informationen bezüglich der Qualität des Datentransfers austauschen. Es ist auf diesem Wege möglich die richtige Reihenfolge der Pakete zu kommunizieren (verbindungsorientiertes Protokoll). Weiterhin lässt sich feststellen, ob ein Paket während der Übertragung verloren gegangen ist (verlässliches Protokoll). Anschaulich wird die Übertragung „kontrolliert“. Während man sich UDP als Postkarte vorstellen kann, entspricht TCP einem „Einschreiben mit Rückschein“, Schwenk [2010].
Abbildung 1.5 stellt den schematischen Aufbau eines TCP Pakets dar. Der Datenteil interessiert uns weniger, ebenso wie weitere, nicht aufgeführte Inhalte des Headers. Wir wollen eher auf die in der Abbildung aufgeführten Inhalte des Headers eingehen und verstehen, wie mit diesen die o.g. Eigenschaften des Protokolls erreicht werden.

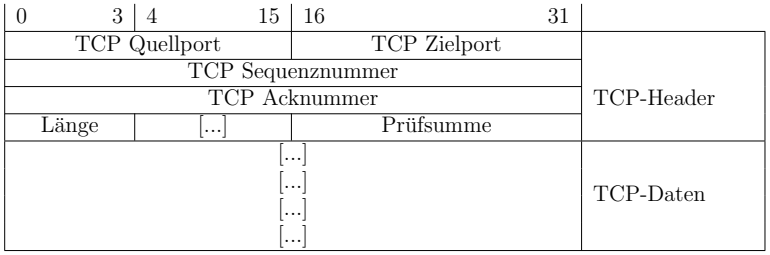


Abb. 1.5: Aufbau eines TCP - Pakets, schematisch

1.5.1 TCP - Header

Die Inhalte der Felder TCP Quellport - bzw. TCP Zielport sind analog zu deren Inhalt in einem UDP Header. Die Berechnung der Prüfsumme erfolgt ebenso auf ähnliche Weise, wie bei UDP (und erfüllt auch die gleiche Funktion). Das Längenfeld gibt die Länge des Headers in Vielfachen von 32 bit an.
Die mit einem TCP übertragenen Bytes werden fortlaufend durchnummeriert. Die 32 - bit Sequenznummer gibt dabei die Nummer des ersten in diesem Datenpaket enthaltenen Bytes an, die 32 - bit ACK-Nummer die des letzten empfangenen Bytes. Auf diesem Wege ist eine Synchronisation zwischen Client und Server möglich. Die Angabe dieser beiden unscheinbaren Informationen ist daher, wie wir im Folgenden sehen werden, entscheidend für die Verlässlichkeit von TCP.

Sequenznummer

ACK-Nummer

1.5.2 TCP/IP Verbindung

Verbindungsaufbau

In Abschnitt 1.4.2 wurde erläutert, dass UDP ein verbindungsloses Protokoll darstellt und daher möglicherweise Fehler bei der Datenübertragung auftreten können. In diesem Abschnitt werden wir sehen, wie es möglich ist diese Probleme durch den Aufbau einer virtuellen Verbindung zwischen Client und Server zu umgehen.

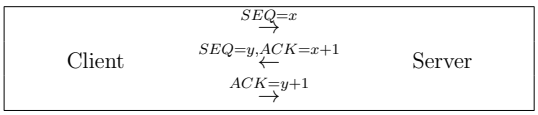


Abb. 1.6: TCP drei Wege Handschlag

In Abbildung 1.6 ist die Vorgehensweise zum Aubauen einer TCP - Verbindung schmatisch dargestellt. Der Client stößt den Aufbau dieser Verbindung in der Regel an. Er sendet ein TCP - Paket zum Server, dessen Inhalt einzig die vom Client gewählte Sequenznummer x enthält. Diese Nummer sollte beim Aufbau einer neuen Verbindung stets neu und zufällig gewählt werden. Hat der Server x empfangen, so schickt er $x + 1$ als Bestätigung (**ACK**nowledgement) zum Client zurück. Zusätzlich sendet der Server selbst eine (zufällig gewählte) Sequenznummer y an den Client und dieser bestätigt diese, indem er seinerseits ein Acknowledement $y + 1$ an den Server sendet. Sequenznummern, die der Client sendet, beziehen sich stets auf Pakete, die Daten vom Client zum Server transportieren. Umgekehrt beziehen sich die ACK-Nummern, die der Client sendet auf Pakete, die vom Server zum Client gesendet werden. Mit den Sequenz - bzw. ACK - Nummern, die der Server sendet, verhält es sich genau umgekehrt.
Im Zuge des Verbindungsaufbaus einigen sich beide Seiten außerdem auf die Größe der Pakete. Durch den Austausch der Sequenz - und ACK - Nummern, ist

Vollduplexverbindung

es nun möglich, dass Daten gleichzeitig sowohl vom Server zum Client, als auch in die entegengesetzte Richtung geschickt werden können. Man spricht von einer *Vollduplexverbindung*.

Verbindungsendpunkt

Die aufgebaute Verbindung wird über den Socket eindeutig identifiziert. Der Socket wird (im Zusammenhang mit TCP) auch *Verbindungsendpunkt* genannt.

B

Beispiel 1.2

Man nehmen an, dass über einen Port eines Hosts zwei TCP - Verbindungen aufgebaut werden. Mit Hilfe der Sequenznummern ist es möglich die eingehenden Pakete richtig zu verarbeiten, da diese bei jedem Aufbau einer Verbindung neu und zufällig gewählt werden. Die Wahrscheinlichkeit, dass in beiden Verbindungen die selbe Sequenznummer gewählt wird, ist sehr gering.

K

Kontrollaufgabe 1.7

Welche Eigenschaften hat ein verbindugsorientiertes Protokoll?

K

Kontrollaufgabe 1.8

Wodurch wird der Einsatz von TCP motiviert?

K

Kontrollaufgabe 1.9

Welche Eigenschaften erfüllt ein verlässliches Transportprotokoll?
Welche Rolle spielen dabei ACK-, bzw. Sequenznummer?

K

Kontrollaufgabe 1.10

Erläutern Sie den TCP - drei Wege Handschlag. Wieso reicht der Handschlag, um einen verlässlichen Datentransfer zu gewährleisten?

K

Kontrollaufgabe 1.11

Die Sequenznummern werden im TCP Handschlag stets unabhängig und zufällig gewählt. Wie groß ist die Wahrscheinlichkeit, dass zwei Sequenznummern übereinstimmen?

Datentransfer

Nach dem TCP - Handschlag besteht nun eine (virtuelle) Vollduplexverbindung zwischen Server und Client. Wir schauen uns im Folgenden an, wie hierdurch die Verlässlichkeit gewährt wird. Wie im Abschnitt 1.5.1 erläutert, werden alle Bytes, die während einer TCP - Verbindung übertragen werden, ausgehend von der im Handshake verienbarten Sequenznummer, durchnummeriert. Der Empfänger eines Pakets kann die Nummer des letzten empfangenen Bytes ($\hat{=}$ ACK-Nummer) leicht berechnen: Er kennt die Sequenznummer des ersten Bytes des Pakets (wird mitgesendet) und die Länge des Pakets. Im Folgenden werden wir die ACK-Nummer, die sich auf ein Paket mit der Sequenznummer x (Paket x) bezieht mit „ACK x“ bezeichnen.
Wenn ein Paket x gesendet wird, so startet der Absender zeitgleich einen Countdown $C_x(t) = t_x - t$, wobei t die abgelaufene Zeit, gemessen in einer bestimmten Zeiteinheit (z.B. ms), nach Absenden des Pakets ist. Nach t_x Zeiteinheiten ist der Wert 0 erreicht. Wenn $C_x(t) = 0$ gilt, bevor ACK x beim Absender eingetroffen ist, wird Paket x erneut versandt.
Im Folgenden betrachten wir einen Datenstrom von einem Server zu einem Client, d.h. der Server schickt angeforderte Daten an den Client. Die Überlegungen hängen jedoch nicht von dieser Richtung ab.
Die einfachste Art und Weise Daten zu transferieren besteht nun darin, dass der Server ein Paket $i + 1$ erst dann sendet, wenn er die Bestätigung ACK i für das Paket i empfangen hat. Dies kann jedoch sehr zeitaufwendig sein: Wenn Paket i nicht beim Empfänger ankommt, wartet der Server t_i Zeiteinheiten, bis er das Paket erneut sendet. Nicht nur der Transfer dieses Pakets wird verzögert, sondern auch der aller nachfolgender Pakete.
Mit der Methode der *sliding windows* kann man diesem Problem begegnen. Der Server sendet nun nacheinander n Pakete, wobei n eine festgelegte, natürliche Zahl ist. Paket $n+1$ wird erst dann versandt, wenn ACK 1 beim Server eingetroffen ist. Paket $n + 2$ wird erst dann versandt, wenn er ACK1 und ACK 2 erhalten hat, d.h., wenn Paket $n+1$ bereits verschickt wurde und zusätzlich ACK 2 eingetroffen ist, usw. Diese Idee ist aus zwei Gründen sinnvoll:

Sliding Windows

- In der Zeit, die verstreicht, bis ACK k für ein Paket k beim Absender eintrifft, werden bereits weitere Pakete versendet. Die Dauer des Datentransfers wird somit verkürzt.
- Wenn nun Paket k verloren geht, so wird der Absender ACK k zunächst nicht empfangen und nach t_k Zeiteinheiten dieses Paket erneut senden. Die nachfolgenden Pakete bis einschließlich Paket $k + n - 1$ sind jedoch schon beim Empfänger eingetroffen, bzw. „unterwegs“. In der Zeit, die verstrichen ist, um zu erkennen, dass Paket k nicht beim Empfänger eingetroffen ist, wurden somit weitere Pakete versandt und daher wird auch in diesem Fall die Dauer des Datenverkehrs verkürzt.

Die Fenstergröße (die Wahl von n) hat offenbar großen Einfluss auf die Geschwindigkeit der Übertragung. Es gilt: je größer das Fenster, desto schneller der Datentransfer. Der Wert n kann jedoch nicht beliebig groß gewählt werden, denn die Pakete werden, unabhängig von der Reihenfolge, in der sie beim Client eintreffen, in der richtigen Reihenfolge (d.h. entsprechend ihrer Sequenznummern) verarbeitet. Falls ein Paket verlorengeht, müssen alle nachfolgenden Pakete vom Client zunächst solange in einem Puffer gespeichert werden, bis das fehlende Paket eingetroffen ist. Die Größe von n hängt daher von den Pufferkapazitäten des Client ab. Weiterhin kann der Server Pakete nicht unendlich schnell hintereinander versenden. Er muss erst den Header erstellen und die Prüfsumme berechnen. Es gibt aber noch weitere bestimmende Faktoren. Server und Client können die

Größe des Fensters während des Datenaustausches stets optimal anpassen.

Beenden einer Verbindung

Nachdem der Datentransfer beendet ist, wird die Verbindung aktiv beendet. Wir erinnern uns, dass TCP eine Vollduplexverbindung darstellt. Der Datentransfer muss in jede Richtung dieser Verbindung geschlossen werden. Dies kann auch unabhängig voneinander geschehen.

Abb. 1.7: TCP Verbindungsabbau, schematisch

Abbildung 1.7 zeigt schematisch den Ablauf des Verbindungsabbaus. Es gibt hier keine Regel, welche Partei den Abbau anstößt. Wenn ein Programm alle Daten gesendet hat (und der Emfpang dieser Daten bestätigt wurde), kann es veranlassen, dass die TCP - Verbindung (zumindest in der entsprechenden Richtung) geschlossen wird. Hierfür wird ein Paket mit dem Inhalt *FIN* gesendet. Um den Abbau abzuschließen muss diese Nachricht bestätigt werden. Der Abbau der anderen Richtung der Verbindung wird von der Gegenseite angestoßen und verläuft analog.

K

Kontrollaufgabe 1.12

Erläutern Sie die Idee der Methode der Sliding Windows. Nennen Sie zwei Gründe, warum diese Methode der einfachen Variante vorzuziehen ist.

1.5.3 Angriffe auf TCP/IP

Im vorigen Abschnitt haben wir mit TCP ein verlässliches Porotokoll eingeführt. Bei Ausführung dieses Protokolls wird der Empfang von Paketen bestätigt. Bedeutet das, dass alle Pakete auch beim Emfänger ankommen? Ist es einem Angreifer trotz Sequenznummern möglich Pakete einzuchleusen? Wir werden im Folgenden unter anderem diese Fragen klären.
Es sei zunächst darauf hingewiesen, dass alle Inhalte, die in einem TCP - Paket übertragen werden, nicht verschlüsselt sind. Ein Angreifer, der einen Router auf dem Leitweg der gesendeten Pakete kontrolliert, kann alle Daten lesen. Er kann die Daten sogar ändern. In diesem Fall muss er zusätzlich die Prüfsumme neu berechnen und an der entsprechenden Stelle im Header einfügen. Insbesondere kennt ein solcher Angreifer auch die Sequenznummern der entsprechenden Verbindung, da diese auch als Klartext im TCP - Header zu finden sind. Es ist ihm daher möglich eigene Pakete in die Verbindung einzuschleusen. Man beachte jedoch: Der Leitweg von Paketen zwischen Server und Client ist dynamisch, d.h. Pakete werden nicht immer über die selben Router geleitet. Es genügt jedoch bereits, wenn während einer Verbindung ein einziges Paket über einen „bösartigen“ Router geleitet wird. Dann kennt der Router die Sequenznummer und kann Pakete einschleusen. Wenn ein eingeschleustes Paket schneller beim Client ankommt,

als ein entsprechendes „Originalpaket“, so wird letzteres vom Client verworfen: Aus Client - Sicht handelt es sich um das Duplikat eines bereits empfangenen Pakets. Es ist daher auch einem an dem Transfer überhaupt nicht beteiligten Angreifer möglich Pakete einzuschleusen, indem er die Sequenznummer rät. Dieser Angriff wird als *Sequence Prediction Attack* bezeichnet. Dieser Angriff betrifft in den meisten Fällen eher den Client.

Sequence Prediction Attack

Es besteht jedoch auch die Gefahr, dass TCP serverseitig angegriffen wird. Stößt ein Client eine TCP - Verbindung mit einem Server an, so sendet er ihm die erste Nachricht des Handschlags, die die vom Client gewählte Sequenznummer enthält. Der Server bestätigt und sendet die ACK - Nummer. Anschließend sendet er eine selbst gewählte Sequenznummer. Zur weiteren Kommunikation ist es nötig, dass er beide Nummern speichert, um die nächsten Pakete zuordnen zu können. Die beiden Nummern bleiben eine bestimmte Zeit im Speicher des Servers, bevor sie verworfen werden, falls keine Pakete folgen. Durch sehr viele Anfragen zum Aufbau einer TCP - Verbindung an den Server in kurzer Zeit, kann die Speicherkapazität ausgereizt werden. Falls der Speicher voll ist, kann keine neue TCP - Verbindung aufgebaut werden. Der Server ist nicht mehr erreichbar. Ein Angreifer kann diese Eigenschaft ausnutzen, indem er mittels IP - Spoofing viele TCP - Anfragen an einen Server sendet. Nachdem der Speicher des Servers voll ist, ist dieser nicht mehr verfügbar. Dieser Angriff wird *SYN Flood* genannt und ist ein Beispiel für einen *Denial of Service* (DOS) - Angriff. Die Antworten des Servers an gefälschte IP - Adressen landen entweder bei einem fremden (zu der IP - Adresse gehörigen Rechner) oder sie laufen aus (TTL Zähler im IP - Header heruntergezählt) und werden von einem Router gelöscht.

SYN flood
Denial of Service

Kontrollaufgabe 1.13

Wie funktioniert der Sequence Prediciton Angriff auf TCP?
Welche Eigenschaften des Protokolls werden ausgenutzt?

K

Kontrollaufgabe 1.14

Was versteht man unter einem DOS - Angriff?
Wie funktioniert der in Abschnitt 1.5.3 erläuterte SYN - flood Angriff auf einen Server mittels TCP?

K

1.6 Zusammenfassung

In diesem Studienbrief haben wir mit TCP und UDP zwei in der Praxis wichtige Transportprotokolle kennengelernt. Die Pakete beider Protokolle sind an Ports adressiert. UDP ist ein zustandsloses, verbindungsloses Protokoll, wohingegen TCP verlässlich und verbindungsorientiert ist. Das Hauptaugenmerk liegt bei UDP auf der Geschwindigkeit der Datenübertragung. Es wird in Kauf genommen, dass Pakete verloren gehen. Bei TCP ist der Schwerpunkt genau entgegengesetzt gewählt.
Beide Protokolle bieten keine Sicherheit der Datenübertragung. Insbesondere können Sie nicht verhindern, dass sensible Daten ausgespäht werden. Dies wird in der Praxis jedoch häufig gefordert. Die Wahrung der „Privatssphäre“ ist Aufgabe höherliegende Schichten des OSI - bzw. TCP - Modells. Im Studienbrief 2 werden wir mit SSL ein Protokoll einführen, welches sichere Datenübertragung

gewährleistet.

1.7 Übungsufgaben

Ü

Übung 1.1

Betrachten Sie erneut Beispiel 1.3. Wie lange dauert der gesamte Datenaustausch, wenn ein beliebiges Paket im ersten Versuch nicht zugestellt werdenkann? Ist diese Dauer abhängig von der Paketnummer?

Ü

Übung 1.2

Man nehme an, ein Server habe zum Speichern von ACK - und Sequenznummern einen eigenen Speicherraum. Dieser habe eine Größe von 1 GB. Nach wievielen TCP - Anfragen ist der Server nicht mehr zu erreichen? (Es sei angenommen, dass die Nummern ewig im Speicher gespeichert bleiben).

Ü

Übung 1.3

Ein Client sendet mittels TCP ein Paket x an einen Server und empfängt anschließend ACK x. Kann der Client sicher sein, dass der Server Paket x empfangen hat?

Ü

Übung 1.4

Aus welchem Grund wird der in Abschnitt 1.5.3 erläuterte SYN-flood Angriff auf einen Server mittels IP - Spoofing durchgeführt? Was wären die Konsequenzen für den Angreifer, wenn er IP - Spoofing nicht anwenden würde?

1.8 Lösung zu den Kontrollaufgaben

- Kontrollaufgabe **??** auf Seite **??**:
  Ohne Lösung
- Kontrollaufgabe **??** auf Seite **??**:
  Die kleinste Zahl, die im UDP Längenfeld stehen kann, ist „8“: Jedes UDP Paket besteht mindestens aus dem Header. Dieser besteht aus 4 Feldern zu je 2 Byte.
- Kontrollaufgabe 1.1 auf Seite 10:
  - Port: Teil der Netzwerkadresse. Gibt an, an welchen Prozess ein Paket eines Transportprotokolls gerichtet ist.
  - Warum Ports: Identifizieren Prozesse. Ermöglichen es, Verbindungen eindeutig zuzuordnen.
  - Das Quell- bzw. Zielportfeld eines UDP Headers hat 16 bit. Ein Rechner stellt daher sinnvollerweise maximal 2^{16} Ports bereit.

- Kontrollaufgabe 1.2 auf Seite 11:
  - Socket: Das Paar (Ip-Adresse, Portnummer)
  - Eine Verbindung wird durch IP-Adresse und Portnummer eindeutig identifiziert.
- Kontrollaufgabe 1.3 auf Seite 11:
  - Die Prüfsumme dient dazu Übertragungsfehler zu erkennen.
  - Der Pseudoheader muss nicht mitgesendet werden, da der Empfänger des Pakets alle Informationen hat, die er zur Berechnung braucht. Er kann den Pseudoheader somit selber berechnen und daraus die Prüfsumme ableiten.
- Kontrollaufgabe 1.4 auf Seite 12:
  - Zustandslos: Der Absender eines Pakets erhält keine Information darüber, ob das Paket beim Empfänger angekommen ist.
  - Verbindungslos: Pakete werden in der Reihenfolge vom Empfänger verarbeitet, in der sie ankommen. Keine Prüfung auf Vollständigkeit.
- Kontrollaufgabe 1.5 auf Seite 12:
  - Streaming, VoIP, etc.
  - Große Datenmengen müssen übertragen werden. Falls ein Paket verloren geht oder Pakete in der falschen Reihenfolge eintreffen, stört dies nur kurz.
- Kontrollaufgabe 1.6 auf Seite 12:
  Das Versenden von IP-Paketen, wobei die Absender IP-Adresse nicht der eigenen IP-Adresse entspricht.
- Kontrollaufgabe 1.7 auf Seite 14:
  Die Pakete werden in der Reihenfolge vom Empfänger an die höheren Schichten weitergegeben, in der sie auch beim Absender abgesendet wurden. Diese Reihenfolge wird kommuniziert.
- Kontrollaufgabe 1.8 auf Seite 14:
  Es gibt Anwendungen, die verlässlichen Transport benötigen. Weiterhin ist es oft nötig, dass die Daten in der richtigen Reihenfolge verarbeitet werden.
- Kontrollaufgabe 1.9 auf Seite 14:
  - Falls ein Paket nicht zugestellt werden konnte, wird es erneut versandt. Alle Pakete kommen beim Empfänger an.
  - Jedes Paket wird mit einer SEQ-Nummer versendet und mit einer ACK-Nummer bestätigt. Ist zu einem bestimmten Paket das Paket mit der entsprechenden ACK-Nummer nach gewisser Zeit nicht beim Absender eingetroffen, schickt er das Paket erneut.
- Kontrollaufgabe 1.10 auf Seite 14:
  Der Absender A wählt eine Sequenznummer SEQ und teilt diese dem Empfänger E mit. Umgekehrt wählt dieser eine ACK-Nummer ACK und teilt diese dem Absender mit. Alle von A gesendeten Pakete werden, beginnend mit SEQ, durchnummeriert. Genauso nummeriert E die von ihm gesendeten Pakete. Dabei schickt E das Paket mit der Nummer ACK+x genau dann ab, wenn er von A das Paket mit der Nummer SEQ+x. So wird garantiert, dass alle Pakete ankommen. Zusätzlich können die Pakete so in der richtigen Reihenfogle an höhere Schichten weitergegeben werden.
- Kontrollaufgabe 1.11 auf Seite 14:

Jede Sequenznummer ist 32 Bit lang. Die Wahrscheinlichkeit, dass zwei Nummern gleich gewählt werden, liegt somit bei $p = \frac{1}{2^{32}}$.

- Kontrollaufgabe 1.12 auf Seite 16:
  Die einfachste Methode mittels TCP Daten zu übertragen ist folgende: Schicke ein Paket. Schicke das nächste erst, wenn die Bestätigung für das vorangegangene Paket eingegangen ist. Ein anderes Extrem könnte so aussehen: Schicke alle Pakete „gleichzeitig“. Schicke diejenigen Pakete nochmal, die nicht bestätigt wurden.
  Die erste Variante ist sehr zeitintensiv, benötigt jedoch kaum Speicherplatz auf der Seite des Empfängers. Die Methode der Sliding-Windows zielt hier auf einen Kompromiss. Es werden mehrere Pakete „gleichzeitig“ versendet. Das nächste Paket (das erste, das noch nicht versandt wurde) wird erst dann versandt, wenn die Bestätigung für das erste Paket eingetroffen ist. So wird Zeit gespart, es wird jedoch empfängerseitig mehr Speicher benötigt.
- Kontrollaufgabe **??** auf Seite **??**: Berechnen der Prüfsumme, erstellen des Headers dauert Zeit. A kann nicht unendlich schnell Pakete versenden. Speicherplatz bei E limitiert die Größe des Fensters.
- Kontrollaufgabe 1.13 auf Seite 17:
  der Angreifer liest die Sequenznummer der Pakete mit (oder, falls er keinen Lesezugriff hat, rät er diese) und schleust eigene Pakete in die Verbindung ein. Pakete werden nicht authentisiert.
- Kontrollaufgabe 1.14 auf Seite 17:
  Es werden in kurzer Zeit sehr viele TCP-Verbindungsanfragen an einen Server gestellt. Dieser muss für jede Verbindung die SEQ- und die ACK-Nummer speichern. Werden ausreichend viele Anfragen gestellt, so ist der Speicherplatz für die ACK- bzw. SEQ- Nummern voll. Der Server kann keine weiteren Verbindungen aufbauen. Der Angreifer sollte Pakete mit gefälschter Absender IP-Adresse verwenden, sonst landen alle Antworten des Servers bei ihm und sein Speicher für SEQ-/ ACK-Nummern ist auch voll.

Studienbrief 2 SSL

2.1 Lernziele

Sie haben verstanden, warum Verbindungen im Internet abgesichert werden müssen.
Sie erklären, wie dies mitttels SSL geschieht. Sie zeigen die Bedeutung der hierfür notwendigen Public Key Infrastruktur auf, und stellen die praktische Umsetzung dieser dar.

Sie geben Angriffe auf SSL wieder, sowie die Gründe für deren Erfolg.

2.2 Advanced Organizer

SSL ist der Standard im Internet zur sicheren Übertragung von Webseiten. Es fügt praktisch eine weitere Schicht im Protokoll-Stack zwischen IP- und Transportschicht ein, den Record-Layer. Sie werden in diesem Abschnitt den Record-Layer ebenso, wie die verschiedenen SSL-Protokolle (Handshake-, Change-Cipher-Spec- und Alertprotokoll) kennenlernen. Um den Aufbau von SSL zu verstehen, werfen wir zunächst einen Blick in die Vergangenheit und schauen uns die Ürsprünge dieses Standards an.

Im letzen Abschnitt dieses Studienbriefs lernen Sie Angriffe auf SSL kennen.

2.3 Angreifermodell

In diesem und in den folgenden Abschnitten werden wir Protokolle aus der Praxis kennenlernen. Wir wollen auch untersuchen, welche Möglichkeiten ein Angreifer hat, um bestimmte Eigenschaften der Protokolle zu brechen. Wir nehmen hier stets einen Angreifer an, der Kontrolle über das gesamte Netzwerk, z.B. das Internet, hat und somit Pakete beliebig manipulieren und erneut versenden kann, ebenso, wie eigens generierte Pakete in das Netzwerk einschleusen kann.

2.4 Übungsaufgaben

Übung 2.1

- Ist die im RFC 2617 definierte „HTTP Basic Authentication“-Methode sicher? Begründen Sie ihre Antwort.
- Welche Vorteile bietet die im selben RFC definierte „HTTP Digest Authentication“-Methode? Gegen welche Angriffe ist diese Methode resistent?

Ü

Übung 2.2

Wodurch unterscheiden sich S-HTTP und SSL?

Ü

Ü

Übung 2.3

- Welchem Zweck dient die *Finished* Nachricht des SSL - Handshakes?
- Ab welcher Nachricht kann der Client sicher sein, dass der Server tatsächlich derjenige ist, für den er sich ausgibt?

Ü

Übung 2.4

Kann es sinnvoll sein die Ciphersuite TLS_ECDH_anon_WITH_NULL_SHA zu verwenden?

Ü

Übung 2.5

Geben Sie eine Ciphersuite an, die folgenden Bedingungen genügt:

- Server **und** Client müssen Schlüsselmaterial für die Berechnung des Pre-Master Secrets beitragen.
- Alle in das Pre-Master-Secret eingehenden Werte sollen **frisch** sein, d.h. das Schlüsselmaterial soll **nicht bereits in vorherigen Sitzungen** verwendet worden sein.

Begründen Sie ihre Entscheidung

Ü

Übung 2.6

- Sind Man-in-the-Middle Angriffe auf das SSL - Protokoll grundsätzlich möglich? Begründen Sie ihre Antwort.
- Ist ein Benutzer in der Lage einen möglichen Angriff auf die SSL- Session zu erkennen? Begründen Sie ihre Antwort.
- Ist ein Angreifer in der Lage eine bereits aufgebaute Sesseion zu manipulieren? Begründen Sie ihre Antwort.

Studienbrief 3 SSH

3.1 Lernziele

Sie führen Gründe an, die den Einsatz von SSH motivieren. Sie zeigen die Funktionsweise von SSH auf und schildern Angriffe auf SSH.
Speziell erarbeiten Sie die Problematik von verschlüsselten Längenfeldern.

3.2 Advanced Organizer

Mittels SSH ist sicherer remote-Zugriff auf entfernte Hosts möglich. In diesem Studienbrief lernen Sie die Protokolle SSH-1 und SSH-2 kennen. Der Schwerpunkt liegt hier auf den unterschiedlichen Authentifizierungsmethoden. Sie lernen mit Nutzer- und Host-Schlüsseln eine Möglichkeit kennen Authezität im Internet ohne Zertifikate zu garantieren. Sie verstehen, wie SSH diese Verfahren einsetzt.

3.3 Motivation

Im letzten Abschnitt haben wir gesehen, wie mit Hilfe eines Public Key Zertifikats eine sichere Verbindung zu einem vertrauenswürdigen Server hergestellt werden kann. Ein zentraler Bestandteil beim Aufbau einer SSL-Verbindung ist die Authentifizierung des Servers. Es ist oft jedoch zusätzlich noch nötig, dass der Client authentifiziert werden muss. Man stelle sich z.B. vor, dass ein Server eines Rechenzentrums von einem Systemadministrator ferngewartet werden muss. Der Administrator benötigt hierfür besondere Rechte auf dem entfernten Server. Diese kann er bekommen, indem er sich etwa auf dem Server als Administrator anmeldet. Ein anderes Szenario besteht darin, dass eine Nutzerin sich für ein Nutzerkonto auf einem entfernten Rechner authentisieren möchte (z.B. hat sie ein Benutzerkonto in einem Rechenzentrum und möchte auf dieses Konto von zuhause aus zugreifen, um eine Datei auf ihren Rechner zuhause zu kopieren). Eine Möglichkeit der Realisierung besteht in der Authentifizierung des Clients über SSL. Hierfür benötigt jeder Client ein eigenes Zertifikat. Im Falle des Systemadministrators mag dies noch umsetzbar sein. Es ist allerdings sehr aufwendig jedem (potentiellen) Nutzer ein Zertifikat auszustellen, welches der Server akzeptiert. SSH (**S**ecure **Sh**ell) bietet die Möglichkeit den Client ohne den Einsatz von SSL zu authentifizieren und eine sichere Verbindung zwischen Client und Server auzubauen. SSH ist ein Protokoll, für dessen Einsatz sowohl auf dem Client-PC, als auch auf dem Server entsprechende Software installiert werden muss.
Für eine SSH-Verbindung sollten die selben Sicherheitsanforderungen gelten, wie für eine SSL-Verbindung, d.h. es sollte gewährleistet werden, dass nur authentifizierte Nutzer Zugriff auf ein bestimmtes Benutzerkonto bekommen, dass Daten während der Übertragung nicht verändert werden können und dass Daten verschlüsselt übertragen werden.

Kontrollaufgabe 3.1

Worin liegt die Motivation für die Entwicklung von SSH?

K

3.4 Einführung

Das erste SSH - Produkt, d.h. die erste SSH verwendende Software, wurde von dem finnischen Informatiker Tatu Ylönen für Unix erfunden. Remote-Befehle, wie *rlogin*, *rcp* oder *ftp* zum entfernten Login oder zum entfernten Kopieren schienen Ylönen zu unsicher (remote (engl.) $\hat{=}$ entfernt). Seit der Entwicklung von SSH sollten für die genannten Funktionen die Befehle *slogin*, *scp*, bzw. *sftp*, verwendet werden.
Mittlerweile gibt es SSH Produkte für viele Betriebssysteme. Die urpsrüngliche Version SSH 1.0 und ihre Weiterentwicklungen 1.3 und 1.5 wiesen noch Sicherheitsmängel auf und wurden schließlich durch SSH 2.0, siehe [RFC4251], ersetzt. Wir werden im Folgenden kurz SSH-1 (SSH-2) schreiben, Barrett and Silverman [2002].
Mittels SSH ist es möglich sich (authentifizierten) Zugang zu einem Benutzerkonto auf einem entfernten Computer zu verschaffen und Daten von diesem Computer sicher zur eigenen Arbeitsstation zu übertragen. Außerdem lassen sich mittels SSH Befehle auf einem entfernten Rechner sicher ausführen. Die Eigenschaft „sicher“ bedeutet hier, wie oben angedeutet, dass ein Befehl (oder eine Datei) von Dritten nicht eingesehen oder verändert werden kann. Der Rechner, von dem aus auf den entfernten Rechner zugegriffen wird, wird SSH - Client genannt und der, auf den zugegriffen wird, wird als SSH - Server bezeichnet.

3.5 Bereitstellung von Schlüsseln

Für die Authentifikation auf einem SSH - Server muss die Identität des SSH - Clients festgestellt werden. Nach erfolgreich bestandener Prüfung kann auf das Benutzerkonto zugegriffen werden. Die Identität lässt sich leicht mittels einer Username/ Passwort Abfrage prüfen. Diese birgt jedoch viele Risiken, bzw. Probleme:

- Das Passwort muss sicher vom Client zum Server übertragen werden. Selbst nach sicherer Übertragung kann ein Passwort auf einem korrumpierten Server abgefangen werden.
- Ein sicheres Passwort lässt sich nur schwer merken.
- Die meisten Betriebssysteme unterstützen nur ein Passwort pro Nutzerkonto. Gemeinsam genutzte Konten können so nur schwer, bzw. umständlich verwaltet werden (z.B. bei Passwortänderung).

Aus diesem Grund unterstützt SSH die Authentifizierung mit Hilfe von öffentlichen Schlüsseln. Hierdurch können die obigen Probleme behoben werden.
Um dieses Authentifikationsverfahren nutzen zu können, muss jede Nutzerin zunächst ein Schlüsselpaar, bestehend aus einem öffentlichem Schlüssel und einem (zu diesem öffentlichen Schlüssel passenden) privaten Schlüssel, generieren. Den privaten Schlüssel behält die Nutzerin selbst (er wird verschlüsselt gespeichert und mit einem Passwort geschützt), während der öffentliche Schlüssel auf dem Server installiert werden muss. Um dies zu tun, wird eine zum Nutzerkonto gehörige Liste bearbeitet (oder neu angelegt). In dieser Liste wird der öffentliche Schlüssel in einem neuen Eintrag gespeichert. Es ist darauf zu achten, dass **nur** Nutzer des entsprechenden Kontos Schreibrechte für diese Datei erhalten.
Der private Schlüssel wird auf dem Rechner der Nutzerin (verschlüsselt) abgespeichert. Für jeden Rechner, von dem aus sie remote - Zugriff auf ein bestimmtes Benutzerkonto haben möchte, muss sie entweder ein eigenes Schlüsselpaar generieren und den öffentlichen Schlüssel in der entsprechenden Liste auf dem Server speichern oder die Datei des bereits vorhandenen privaten Schlüssel auch auf

den anderen Rechnern abspeichern. Der öffentliche Schlüssel zu einem speziellen privaten Schlüssel kann auch auf verschiedenen SSH - Servern installiert werden. So muss der Client nur ein Schlüsselpaar generieren und nur einen privaten Schlüssel speichern. Ein von einer Nutzerin generiertes Schlüsselpaar wird als *User Schlüssel* bezeichnet.

User Schlüssel

```
2048 53 81169407586118224007991876252275750848786756533261922 8...
⋮                                                              ⋮
...24565059747734212044729802225605135837291 1 text
2048 53 73487563148561023983472395866103587631540123756347503 3...
⋮                                                              ⋮
...08527462805873265723568734563820348756303 4 text
```

Abb. 3.1: Darstellung eines in einer Liste gespeicherten öffentlichen Schlüssel: Die erste Zahl gibt die Bitlänge des Modulus an, die zweite den Wert des öffentlichen Schlüssels und die dritte den Modulus. Anschließend kann ein beliebiger Text angefügt werden.

Die Authentifizierung für ein Benutzerkonto auf einem SSH - Server läuft nun wie folgt ab: Wenn die Nutzerin sich authentisieren möchte gibt sie das Passwort ein, das den privaten Schlüssel schützt. Ist das Passwort korrekt, so wird mit Hilfe des privaten Schlüssels ein sog. „Authenticator“ an den SSH - Server gesendet. Dieser wird geprüft und nach bestandener Prüfung erhält die Nutzerin Zugang zum Konto. Ein Vorteil dieser Variante gegenüber der einfachen Authentifizierung mittels Nutzername/ Passwort liegt darin, dass das Passwort nicht über das Netzwerk übertragen wird. Es dient lediglich dazu den privaten auf dem SSH - Client **lokal gespeicherten** Schlüssel freizugeben. Darüber hinaus ist der private Schlüssel nicht vom Menschen erzeugt. Er lässt sich, bei geeignet gewählten Parametern, praktisch nicht erraten (oder berechnen).

Neben der Authentisierung des Clients ist es auch sinnvoll, dass dieser den Server authentifiziert. SSH verwendet hier das Prinzip der *Known Hosts*. Wenn ein SSH - Server zum ersten Mal kontaktiert wird, wird die Nutzerin gefragt, ob Sie wirklich mit dem fremden (potentiell gefährlichen) Server Kontakt aufnehmen möchte, siehe Abbildung 3.2. Wird die Frage mit „yes“ beantwortet, so wird der Server in die Liste der bekannten Hosts aufgenommen. Insbesondere wird der öffentliche Schlüssel des Server - Schlüsselpaars vom Client lokal gespeichert. Mit diesem ist es dem Client möglich die Identität des Servers bei der nächsten Kontaktaufnahme zu verifizieren. Diese vom Server generierte Schlüsselpaar wird als *Host Schlüssel* bezeichnet.

Known Hosts

Host Schlüssel

```
Host key not found from the list of known hosts.
Are you sure you want to continue connecting (yes/no)?
```

Abb. 3.2: Kontaktaufnahme mit einem unbekannten Host.

Falls zu einem späteren Zeitpunkt etwa Anfragen an einen speziellen Server auf einen korrumpierten Server umgeleitet werden, so kann der Client dies mit Hilfe des lokal gespeicherten öffentlichen Schlüssels bemerken. Diese Known Host Authentifizierung funktioniert nur, wenn der Server bei der ersten Kontaktaufnahme noch nicht korrumpiert war. Dieses kann man aber bei der ersten Kontaktaufnahme nicht wissen. Es sollte daher geprüft werden, ob der Server wirklich authentisch ist, bevor der Server in die Liste der known Hosts aufgenommen

wird. Mit Hilfe der PKI, siehe Studienbrief 2, wäre eine solche Authentifizierung auch möglich. Es ist allerdings sehr aufwendig jedem SSH- Server ein Zertifikat auszustellen. Darüber hinaus stellt sich die Frage, wer die Authenzität des Servers prüft. Die Methode der Known Hosts bietet hier einen guten Mittelweg.
Für die Ausführung des SSH-1 Protokolls benutzt der Server einen zusätzlichen Schlüssel, den sog. Serverschlüssel. Mit dem öffentlichen Schlüssel dieses asymmetrischen Schlüsselpaars wird während des Verbindungsaufbaus der Session Schlüssel übertragen. Der Server - Schlüssel wird etwa einmal pro Stunde neu generiert, um sog. Forward - Secrecy zu gewährleisten.

Mit Hilfe des User Schlüssels und des Host - Schlüssels handeln Client und Server zu Beginn einer Session einen symmetrischen Session - Schlüssel aus. Mit diesem werden die während der Sitzung übertragenen Daten verschlüsselt. Wie dies genau funktioniert, wird in den nächsten Abschnitten erläutert.

E

Exkurs 3.1

Wenn sich eine Nutzerin wie oben beschrieben für verschiedene Nutzerkonten authentisieren möchte, so muss sie jedes mal ihr Passwort erneut eingeben, um einen privaten Schlüssel „freizugeben“. Mit Hilfe des sog. SSH - Agenten ist es möglich dies zu umgehen. Der Agent ist ein Programm, das auf dem lokalen Rechner der Nutzerin installiert ist. Nach einmaliger Anmeldung hält dieses Programm die privaten Schlüssel bis zum Logout im Speicher und übernimmt die Authentisierung bei den kontaktierten SSH - Servern.

K

Kontrollaufgabe 3.2

- Welche Probleme treten auf, wenn der Client lediglich durch eine Username/ Passwort Abfrage authenifiziert wird? Ist dieses Verfahren sicher?
- Welche Vorteile hat die Authentifizierung mit public Keys?

K

Kontrollaufgabe 3.3

- Lässt sich die Identität eines Servers bereits bei der ersten Kontaktaufnahme sicher feststellen? Welche „Hilfsmittel“ werden benötigt?
- Erläutern Sie die Server Authentifizierungsethode der Known Hosts. Welchem Zweck dient Sie?

Kontrollaufgabe 3.4

Warum ist bei der Installation eines öffentlichen Schlüssels auf einem Server darauf zu achten, dass nur authentifizierte Nutzer Schreibzugriff auf die Liste der öffentlichen Schlüssel haben?

K

3.6 SSH - 1

3.6.1 Server Authentifizierung

Wie beim Aufbau einer SSL Verbindung führen Client und Server zu Beginn einen Handshake aus, um sich auf die zu verwendenden kryptografischen Parameter zu einigen. Beim Protokoll SSH-1 läuft dieser Handshake folgendermaßen ab:

1. Der Client kontaktiert den Server und signalisiert, dass eine SSH - Verbindung aufgebaut werden soll.

2. Client und Server einigen sich auf eine Protokollversion.

3. Der Server beweist seine Identität. Weiterhin sendet er Listen von unterstützten Algorithmen.

4. Der Client wählt passende Parameter aus dieser Liste aus und schickt einen geheimen Session - Schlüssel an den Server.

5. Die Server Authentifizierung wird abgeschlossen.

6. Die abgesicherte Verbindung ist aufgebaut.

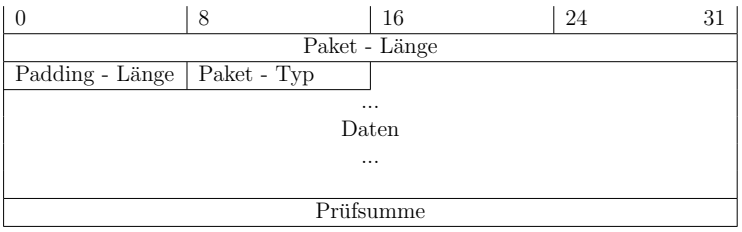
Nachdem die sichere Verbindung aufgebaut ist, wird der Client authentifiziert. Die einzelnen Schritte des Handshakes werden im Folgenden näher erläutert, siehe auch Ylonen [1996]:

1. Der Client sendet eine Verbindungsanfrage an den Server. Diese ist per Konvention an Port 22 gerichtet.

2. Server und Client geben Protokollversionen an, die sie unterstützen. Falls eine Version existiert, die beide unterstützen, so wird eine Verbindung aufgebaut. Falls keine gemeinsame Version gefunden wird, so wird der Verbindungsaufbau abgebrochen (und es wird eine Fehlermeldung ausgegeben).

3. Nachdem beide sich auf eine Protokollversion geeinigt haben, werden alle Daten in Form von Paketen gesendet. Ein SSH-1.5 Paket ist in Abbildung 3.3 zu sehen.
   Der Server sendet dem Client den öffentlichen Host Schlüssel, den öffentlichen Server Schlüssel, eine Folge von acht zufälligen (Prüf-)Bytes, sowie von ihm unterstützte Authentifizierungs-, und (symmetrische) Verschlüsselungssalgorithmen.
   Beide berechnen mit diesen Informationen eine **eindeutige** SessionID, die sich als MD5-Hash des Host - Schlüssels, des Server - Schlüssels und der Prüfbytes zusammensetzt.

Abb. 3.3: Darstellung eines SSH-1.5 Pakets



Das Feld „Paket - Länge“ gibt die Länge des Pakets (in Bytes) an, wobei das Feld selber, sowie die Länge des Paddings nicht mitgezählt werden. Die Länge eines Pakets ist stets ein Vielfaches von 8 Bytes. Die Menge des Paddings ist deshalb variabel: Das Padding hat eine Länge von $(8 - (\text{Länge} \mod 8))$ Bytes. Die Länge des Padding - Feldes liegt also zwischen 1 und 8 Byte. Der Inhalt des Padding Feldes wird zufällig gewählt, so werden sog. known - plaintext Angriffe erschwert.
Das Feld „Paket - Typ“ gibt an, wie der Datenteil zu interpretieren ist. Steht in diesem Feld etwa der Wert „2“, so enthält das Datenfeld der Public Key des Servers.
Die Prüfsumme wird als 32-bit CRC mit einem festgelegten Polynom über die Inhalte der Felder „Padding“, „Daten - Typ“ und „Daten“ gebildet. Diese Art der Integritätsprüfung ist nicht sehr stark und wird deshalb in SSH-2 nicht mehr verwendet.

4. Der Client prüft zuerst den Host Schlüssel des Servers. Falls der Server in der Liste der Known - Hosts zu finden ist und der dort gespeicherte Schlüssel mit dem soeben vom Server erhaltenen Schlüssel übereinstimmt, so akzeptiert der Client ihn zunächst. Die Kenntnis des zugehörigen privaten Teils des Host Schlüssels wird zu einem späteren Zeitpunkt implizit geprüft.
Falls der empfangene Host Schlüssel nicht mit dem in der Liste gespeicherten Schlüssel über einstimmt oder der Schlüssel noch nicht bekannt ist, so muss sich der Client entscheiden, ob der dem Server traut oder nicht, etwa durch eine Entscheidung der Nutzerin, siehe Abbildung 3.2.
Wenn der Client den Host Schlüssel nicht akzeptiert, so wird der Verbindungsaufbau abgebrochen. Andernfalls wählt der Client aus der vom Server empfangenen Liste einen Verschlüsselungsalgorithmus aus und generiert einen zu diesem Algorithmus passenden Schlüssel K. Dieser wird doppelt verschlüsselt (mit beiden (welchen?) öffentlichen Schlüsseln des Servers) und zusammen mit den ausgewählten Algorithmen und den Prüfbytes aus Schritt 3 an den Server gesendet.
Ab diesem Zeitpunkt werden alle Nachrichten verschlüsselt übertragen.
Version SSH-1.5 unterstützt die folgenden Verschlüsselungsalgorithmen:

Tabelle 3.1: Verschlüsselungsalgorithmen in SSH-1.5

SSH_CIPHER_NONE	n.e.	keine Verschlüsselung
SSH_CIPHER_IDEA	o.	Verschlüsselung mit IDEA im CFB Modus
SSH_CIPHER_DES	o.	Verschlüsselung mit DES im CBC Modus
SSH_CIPHER_3DES	v.	Verschlüsselung mit Triple-DES im CBC Modus
SSH_CIPHER_RC4	o.	Verschlüsselung mit RC4

n.e. $\hat{=}$ nicht empfohlen; o. $\hat{=}$ optional; v. $\hat{=}$ verbindlich

Bemerkung: Jeder Server muss das Verschlüsselungsverfahren 3DES (verbindlich) unterstützen und dem Client anbieten, die anderen Verfahren sind optional, bzw. nicht empfohlen.

5. Der Server entschlüsselt den Schlüssel K und sendet eine konstante Bestätigung an den Client. Diese Nachricht verschlüsselt er mit dem soeben von

ihm entschlüsselten Schlüssel K. Der Client, entschlüsselt diese Nachricht mit dem Schlüssel K. Ist der Klartext der Nachricht identisch mit der (konstanten) Bestätigung, so kann der Client sicher sein, dass der Host den privaten Host Schlüssel kennt. Die Authentifizierung des Servers ist somit abgeschlossen.

6. Die abgesicherte Verbindung ist nun aufgebaut. Alle Nachrichten werden gemäß dem vom Client in Schritt 4 gewählten Verschlüsselungsalgorithmus unter dem Schlüssel K verschlüsselt. Anschließend wird der Client authentifiziert.

Kontrollaufgabe 3.5

Im Längenfeld eines SSH-1.5 Pakets steht der Wert „x“. Wie lang ist der Datenteil dieses Pakets?

K

Kontrollaufgabe 3.6

Der Session Schlüssel wird im Handshake von SSH-1 doppelt verschlüsselt übertragen - wie? Aus welchem Grund wird der Schlüssel doppelt verschlüsselt?
Tip: Was passiert, wenn der Host Schlüssel nach Beenden der Session gebrochen wird (und der Server Schlüssel bereits gelöscht wurde)?

K

Kontrollaufgabe 3.7

- Wieso muss die wärend SSH-1 vom Server gesendete Bestätigung konstant sein?
- Wieso kann der Client nach empfangener (korrekter) Bestätigung (im SSH-1 Protokoll) sicher sein, dass der Server authentisch ist?

K

3.6.2 Client Authentifizierung

Nachdem die sichere Verbindung aufgebaut ist, wird der Client vom Server authentifiziert. Wie oben bereits erwähnt, wird der Client authentifiziert, indem er einen sog. Authenticator an den Server sendet. Hierfür wird eine der Authentifizierungsmethoden, die der Server während des Handshakes vorgeschlagen hat, verwendet. Der Client benutzt diese Methoden in belibiger Reihenfolge, um sich zu authentisieren. Nachdem er mit einer dieser Methoden erfolgreich authenfiziert wurde, ist die Authentifizierung des Clients abgeschlossen. Dem Client bieten sich u.a. folgende Möglichkeiten zur Authentisierung:

- Rhosts
- RhostsRSA
- Passwort
- Public-Key

- Einmal - Passwort

Die Funktionsweise der einzelnen Authentifizierungsmethoden werden im weiteren Text erläutert.

Rhosts

Die Authentifikation mittels Rhosts funktioniert ohne, dass der Client beweist, dass er Kenntnis eines Geheimnisses (z.B. eines Passworts oder eines geheimen Schlüssels) hat. Vielmehr benutzt der Server bei dieser Authentifikationsmethode eigene Authorisierungsregelen. Der Ablauf ist wie folgt:

1. Der Client signalisiert dem Server, dass er authentifiziert werden möchte.

2. Der Server ordnet dem Client (d.h. der Quelladresse der eingehenden Verbindung) einen (Host-) Namen zu. Dieser Name ist nicht abhängig vom Server, d.h. alle Server ordnen dem Client einen identischen Hostnamen zu.

3. Der Server prüft lokal, ob der Client vertrauenswürdig ist. Hierzu sucht er den Hostnamen in einer nach bestimmten Autorisierungsregeln erstellte Liste der sog. RemoteHosts (Rhosts). Findet er den Hostnamen nicht, so wird die Authentifizierung abgebrochen.

4. Es wird sichergestellt, dass das Client-Programm, das Kontakt zum Server aufgenommen hat, vertrauenswürdig ist.

5. Nach dieser Prüfung ist die Authentifizierung abgeschlossen.

Auch wenn das Rhosts - Verfahren einige Vorteile bietet, wie etwa, dass es die Eingabe von Passwörtern umgeht, ist diese Methode heutzutage nicht mehr als sicher anzusehen. Einerseits ist es leicht die IP - Adresse eines Rechners zu ändern (Schritt 2). In diesem Fall ordnet der SSH - Server dem Client einen falschen Namen zu - Warum? - und authentifiziert im schlimmsten Fall einen nicht vertrauenswürdigen Client. Andererseits ist Schritt 4 in der urpsrünglichen Variante nicht auf allen Betriebssystemen durchführbar: Die Prüfung der Authenzität des Programms auf dem Client wurde unter Linux (SSH wurde ursprünglich für Linux entwickelt) über die verwendete Portnummer geprüft. Die Portnummern 1 bis 1023 konnten nur von vom Administrator (SuperUser) installierten (priviligierten) Programmen genutzt werden. Ein vom Administrator installiertes Programm wird als vertrauenswürdig eingestuft. Unter Windows oder MacOS funktioniert diese Prüfung nicht, da entweder nicht zwischen Benutzern (MacOS) oder nicht zwischen priviligierten und nicht priviligierten Ports (Windows) unterschieden werden kann. Aus diesen Gründen ist Rhosts standardmäßig deaktiviert, Barrett and Silverman [2002].

RhostsRSA

Die RhostsRSA Authentifikationsmethode stellt eine Weiterentwicklung von Rhosts dar, die diese Schwächen behebt. Die auftretenden Probleme werden mit Hilfe des User Schlüssels behoben. Anstatt den Namen eines Hosts aus seiner Quelladresse abzuleiten, stellt der Client dem Server den öffentlichen Teil des User Schlüssels bereit. Mittels eines Challenge and Response Verfahrens kann der Client so seine Identität (d.h., dass er den privaten Teil des Schlüssels kennt) explizit beweisen. Auf diese Art und Weise lässt sich gleichzeitig das Problem aus Schritt 4 umgehen, indem der private Schlüssel des User Schlüssels so abgespeichert wird, dass er nur von autorisierten Nutzern ausgelesen werden kann.

Beispiel 3.1

Barrett and Silverman [2002] Eine Authentifizierung des Clients mittels der RhostsRSA Methode könnte folgendermaßen ablaufen:
Der Client hat bereits eine Verbindung zum Server A aufgebaut und möchte sich nun zusätzlich bei Server B authentisieren. Die Autorisierungsregeln von B schreiben u.a. vor, dass B jedem Client vertraut, dem auch A vertraut. B erhält also den öffentlichen Schlüssel des Clients und prüft seine Identität. Anschließend kontaktiert er A und erfährt, dass das Programm des Clients vertrauenswürdig ist. Nachdem B noch geprüft hat, dass das Programm von A vertrauenswürdig ist, akzeptiert er den Client.

B

Passwortauthentifizierung

Bei der Authentifizierung mittels Passwort, wird die Nutzerin aufgefordert das zum entfernten Konto passende Passwort einzugeben. Dieses wird dann über die sichere Verbindung an den Server gesendet. Ist das Passwort korrekt, erhält die Nutzerin Zugang zum Konto. Diese Methode ist sehr einfach durchzu führen, weil die Nutzerin keine zusätzliche Software benötigt, bzw. kein Schlüsselpaar generieren muss. Aufgrund der Tatsache, dass die Verbindung zum Server bereits abgesichert ist, kann das Passwort auch nur schwer von Dritten abgehört werden. Es wird jedoch beim Server entschlüsselt und kann daher von einem korrumpierten Server ausgelesen werden. (Erinnern Sie sich, dass Server mit dem Known Host Prinzip authentifiziert werden. Bei einem, bis zum Beginn der Verbindung, unbekannten Server kann nicht ausgeschlossen werden, dass er korrumpiert ist.)

Public-Key-Authentifizierung

Dies lässt sich durch die Public-Key-Authentifizierung umgehen. Dieses Verfahren läuft, wie in Abschnitt 3.5 beschrieben ab: Der Client teilt dem Server seinen öffentlichen Schlüssel mit. Falls dieser in einer Liste der „autorisierten Schlüssel“ zu finden ist, schickt der Server ihm eine Challenge, bestehend aus einer Folge von 32 zufälligen Bytes, die mit dem öffentlichen User Schlüssel verschlüsselt ist. Der Client entschlüsselt die Challenge mit seinem privaten Schlüssel, kombiniert sie mit der SessionID (siehe Schritt 3 der Serverauthentifizierung) und bildet den MD5-Hashwert des Ergebnisses. Anschließend sendet er diesen Wert zurück an den Server. Dieser kann den Hashwert ebenso berechnen und vergleicht beide Werte. Stimmen sie überein ist der Client authentifiziert.
Es mag umständlich erscheinen, dass der Client nicht einfach die Challenge entschlüsselt und zurücksendet. Diese einfachere Variante bietet jedoch einige Sicherheitslücken: Zum einen sind ohne SessionID Replay-Attacken möglich, falls die Challenge durch einen schwachen Zufallszahlengenerator generiert wurde. Zum anderen ist es ohne Bildung des Hashwerts möglich, dass ein korrumpierter Server dem Client als Challenge keinen zufälligen Wert sendet, sondern eine (vom Client) verschlüsselte Nachricht, die er aus einer anderen Verbindung abgefangen hat. Auf diese Art und Weise würde der Client dem Server die Nachricht entschlüsseln.

Einmal - Passwort

Falls eine Nutzerin oft unterschiedliche Host - Rechner benutzt, so können weder die Authentifizierung mittels Passwort, noch die Public - Key - Authentifizierung garantieren, dass alle Daten nicht abhörbar werden. Dies liegt daran, dass der fremde Host korrumpiert sein kann. Auf so einem Rechner möchte man weder den privaten Teil eines Nutzer Schlüsselpaars speichern, noch möchte man hier ein Passwort eingeben, da es evtl. mitgelesen wird. Aus diesem Grund bietet SSH-1 auch die Möglichkeit sich über sog. One Time Passwords (OTP) zu authentisieren. Das einzugebende Passwort ist in diesem Fall nicht statisch, sondern bei jedem Login einer (ausgedruckten und mitgeführten) Liste zu entnehmen oder von einer Software zu berechnen. Es gibt auch kommerzielle Lösungen für OTPs, etwa SecurID von RSA Security Inc. oder ein System von Trusted Information Systems, Inc, für nähere Details siehe Barrett and Silverman [2002].

Um die Integrität der übertragenen Nachrichten zu garantieren, enthalten alle während des Protokolls übertragenen Nachrichten eine Integritätsprüfung. Hierfür wird nach einem bestimmten Algorithmus eine Prüfsumme über den Inhalt der Nachricht gebildet. Im Falle von SSH-1 wird CRC-32 verwendet. Die Berechnung der Prüfsumme auf diese Art und Weise reicht aus, um (zufällige) Übertragungsfehler zu bemerken. Einem böswilligen Angreifer ist es jedoch möglich den Inhalt der Nachricht zu verändern, ohne, dass die Prüfsumme ihren Wert verändert. Die Intgeritätsprüfung ist daher eine Schwachstelle in SSH-1 und ist bei SSH-2 verbesert worden.

K

Kontrollaufgabe 3.8

Welche Möglichkeiten der Client - Authentifizierung bietet SSH-1. Beschreiben Sie die einzelnen Methoden kurz. Gehen Sie auf Vor - und Nachteile ein.

K

Kontrollaufgabe 3.9

Im vierten Schritt der Client - Authentifizierung mittels Rhosts, wird vom Server geprüft, ob das entfernte Programm vertrauenswürdig ist. Warum funktioniert die o.g. Prüfung nicht unter MacOS, d.h. warum muss zwischen Benutzern des Client Pc unterschieden werden?

3.7 SSH - 2

Im Gegensatz zu den Weiterentwicklungen von SSL, die jeweils auf der vorhergehenden Protokollversion aufgebaut haben und die jeweiligen Schwachstellen beheben sollten, stellt SSH-2 keine „Verbesserung“ von SSH-1, sondern ein komplett neues Produkt dar. Der größte Unterschied besteht darin, dass SSH-1 aus nur einem Protokoll besteht, das mehrere Funktionen in sich zusammenfasst, SSH-2 jedoch aus drei Protokollen aufgebaut ist, die alle verschiedene Funktionen haben. Die „Bausteine“ von SSH-2 sind die Folgenden Protokolle, [RFC4251]:

SSH-TRANS: Dieses Protokoll entspricht in etwa dem „Handshake“ von SSH-1. Es werden Algorithmen und eine SessionID vereinbart und weiterhin wird ein Session Schlüssel etabliert. Nachdem dieses Protokoll erfolgreich beendet wurde, ist die bestehende Verbindung abgesichert, d.h. die Datenübertragung ist verschlüsselt und integer.
Im Gegensatz zu SSH-1, werden ab der Version 2.0 mehr Verschlüsselungsalgorithmen unterstützt, siehe Tabelle 3.2. SSH-2 ermöglicht es auch, die Authenzität eines Servers mit Hilfe von Zertifikaten zu prüfen. Bei der ersten Kontaktaufnahme kann die Identität eines Servers so sichergestellt werden. Außerdem handeln bei SSH-2 beide Parteien gemeinsam einen Schlüssel aus. Jede SSH-2 Implementierung muss verbindlich die Methoden *diffie- hellman-group1-sha1* und *diffie-hellman-group14-sha1* zum Schlüsseltausch unterstützen. Es handelt sich bei diesen Methoden um einen Diffie - Hellman Schlüsseltausch, wobei als Gruppe die Oakley - Gruppe 2(!), bzw. die Oakley Gruppe 14, siehe Exkurs 3.2 , verwendet wird und zum Hashen der sha-1 Algorithmus. Nach dem der Schlüsseltausch abgeschlossen ist, wird bei SSH-2 das gesamte Paket verschlüsselt. Dazu zählt, im Gegensatz zu SSH-1.5, insbesondere auch das Paketlängenfeld.
In Beispiel 3.2 wird der Schlüsseltausch mittels *diffie-hellman-group1-sha1* erläutert.

3des-cbc	v.	Drei Schlüssel 3DES im CBC Modus
blowfish-cbc	o.	Blowfish im CBC Modus
twofish256-cbc	o.	Twofish im CBC Modus mit einem 256-bit Schlüssel
twofish192-cbc	o.	Twofish mit einem 192-bit Schlüssel
twofish128-cbc	o.	Twofish mit einem 128-bit Schlüssel
aes256-cbc	o.	AES im CBC Modus mit einem 256-bit Schlüssel
aes192-cbc	o.	AES mit einem 192-bit Schlüssel
aes128-cbc	e.	AES mit einem 128-bit Schlüssel
serpent256-cbc	o.	Serpent im CBC Modus mit einem 256-bit Schlüssel
serpent192-cbc	o.	Serpent im CBC Modus mit einem 192-bit Schlüssel
serpent128-cbc	o.	Serpent im CBC Modus mit einem 128-bit Schlüssel
arcfour	o.	Die RC4 Stromchiffre mit einem 128-bit Schlüssel
idea-cbc	o.	IDEA im CBC Modus
cast128-cbc	o.	CAST-128 im CBC Modus
none	n.e.	keine Verschlüsselung

Tabelle 3.2: Liste der von SSH-2 unterstützten Verschlüsselungsalgorithmen

e. $\hat{=}$ empfohlen; n.e. $\hat{=}$ nicht empfohlen; o. $\hat{=}$ optional; v. $\hat{=}$ verbindlich

Im Gegensatz zur schwachen CRC-32 Integritätsprüfung von SSH-1 werden bei SSH-2 sog. Message - Authentication - Codes (MAC) verwendet, um die Integrität einer Nachricht zu garantieren. Jede Implementierung muss den Algorithmus *hmac-sha1* unterstützen. Es können opional auch andere Algorithmen verwendet werden. Der Server teilt dem Client mit, welche Algorithmen er unterstützt und der Client wählt aus dieser Liste einen beliebigen (von ihm auch unterstützten) Algorithmus aus.

Exkurs 3.2

E

Oakley - Gruppen [RFC3526], [RFC2414]:
Oakley - Gruppen sind spezielle Gruppen, in denen der Diffie - Hellman Schlüsseltausch durchgeführt wird. Sie entstanden aus Überlegungen zum Oakley Protokoll. Die Oakley Gruppen 2 und 14 sind folgendermaßen spezifiziert:

O. G. 2:	Primzahl:	$p = 2^{1024} - 2^{960} - 1 + 2^{64} \cdot (\lfloor 2^{894}\pi \rfloor + 129093)$
	Generator:	2 oder 22
O. G. 14:	Primzahl:	$q = 2^{2048} - 2^{1984} - 1 + 2^{64} \cdot (\lfloor 2^{1918}\pi \rfloor + 124476)$
	Generator:	2 oder 22

Hierbei ist $\lfloor x \rfloor$ die größte natürliche Zahl kleiner oder gleich x.
Die Oakley Gruppe 2 beschreibt also eine 1024 - bit Gruppe, die Oakley Gruppe 14 eine 2048- bit Gruppe. Die Primzahlen p und q sind beide sog. *Sophie Germain Primzahlen*, d.h. es gilt $\frac{p-1}{2} \in \mathbb{P}$ und $\frac{q-1}{2} \in \mathbb{P}$, wobei $\mathbb{P}$ die Menge der Primzahlen ist. Somit ist die Berechnung mit aktuellen Techniken des diskreten Logarthmus praktisch unmöglich.
In Binärdarstellung haben die ersten 64 und die letzten 64 Bits von p den Wert 1. Um die Bits dazwischen möglichst zufällig erscheinen zu lassen, sind diese von der Binärdarstellung der Zahl π abhängig.
Die Binärdarstellung der primzahl q weist eine ähnliche Struktur auf.

SSH-AUTH: Dieses Protokoll entspricht der Client - Authentifizierung in SSH-1. Wie oben stehen Server und Client verschiedene Authentifikationsverfahren zur Verfügung. Diese sind hier allerdings beschränkt auf die Verfahren „publickey“, „password“ und „hostbased“. Diese Verfahren ähneln den Verfahren „Public -

Key“, „Passwort“, bzw. „RhostsRSA“ des SSH-1 Protokolls. Es ist vorgeschrieben, dass jede Implementierung die Methode „publickey“ unterstützt. Die anderen Methoden sind optional. Es besteht auch die Möglichkeit auf die Authentifizierung zu verzichten, dies wird jedoch nicht empfohlen.

SSH-CONN: Die Authentifizierung der beiden Hosts ist abgeschlossen. Über dieses Protokoll werden die Befehle der Nutzerin übertragen und ausgeführt.
Man beachte, dass SSH-CONN (im Protokollstapel) nicht *über* SSH-AUTH angesiedelt ist, sondern, dass beide Protokolle auf einer Ebene, über SSH-TRANS, angeordnet sind. Kann etwa auf die Client - Authentifizierung verzichtet werden, weil keine sensiblen Daten versendet werden, so werden die Befehle und Daten mittels SSH-CONN übertragen. Dass SSH-AUTH Protokoll wird in diesem Fall nicht ausgeführt.

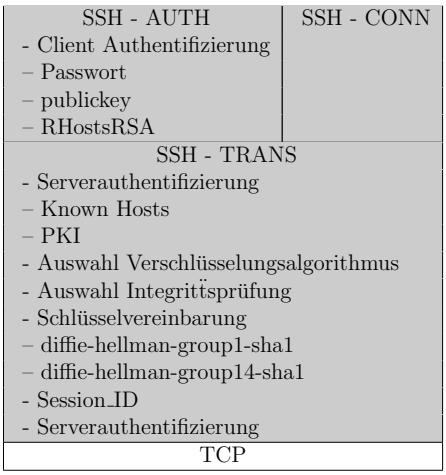


Abb. 3.4: Die Bestandteile von SSH-2 (grau hinterlegt) und ihre Lage im Protokollstack.

B

Beispiel 3.2

diffie-hellman-group1-sha1, [RFC4253]
Es sei:
C der Client,
S der Server,
p eine große Primzahl,
q ein Generator einer Untergruppe von $\mathbb{Z}/\mathbb{Z}_p^*$,
V_S eine Zeichenfolge zur Identifikation von S,
V_C eine Zeichenfolge zur Identifikation von C,
K_S der öffentlicher Teil des Host Schlüssels von S und
hash ein Hash - Algorithmus

Im ersten Schritt berechnen C und S ein gemeinsames Geheimnis:

B

Beispiel 3.3

- C wählt zufällig, gemäß Gleichverteilung, eine Zahl x, $1 < x < q$, und berechnet $e = g^x \mod p$. C sendet e an S.
- S wählt zufällig, gemäß Gleichverteilung, eine Zahl y, $1 < y < q$, und berechnet $f = g^y \mod p$. Weiterhin berechnet S die Werte $K = e^y \mod p$ und H = *hash*$(V_C||V_S||I_C||K_S||e||f||K)$. Dabei sind I_C und I_S bestimmte Nachrichten, die in der selben Session vor der Schlüssel verhandlung gesendet worden sind. Es sei hierbei $V_C||V_S||I_C||K_S||e||f||K$ eine Binär darstellung der entpsrechenden Strings.
  Anschließend signiert S den Wert H. Als geheimen Schlüssel verwendet S den geheimen Teil des Host - Schlüssels. Die Signatur σ sendet er, zusammen mit f und K_S an C.
- C prüft, ob K_S der ihm bekannte öffentliche Schlüssel des Servers ist, s.o. C berechnet dann $K = f^x \mod p$ und H = *hash*$(V_C||V_S||I_C||K_S||e||f||K)$ und prüft mit dem Schlüssel K_S die Signatur σ.

Folgende Informationen sind zu diesem Zeitpunkt sowohl Client, als auch Server bekannt:

- Ein Verschlüsselungsverfahren, mit dem die Nachrichten später verschlüsselt werden sollen.
- Ein Session - Identifier: session_ID $\hat{=} H$.
- Ein Master Secret K.

Hieraus werden verschiedene Schlüssel berechnet:

- Der Initialisierungsvektor zur Verschlüsselung von Nachrichten, die an den Client gesendet werden, ergibt sich zu *hash*$(K||H||$„A“$||$session_ID). Hierbei ist „A“ die ASCII - Codierung des Buchstabens A.
- Der Initialisierungsvektor zur Verschlüsselung von Nachrichten, die an den Server gesendet werden, ergibt sich zu *hash*$(K||H||$„B“$||$session_ID). Hierbei ist „B“ die ASCII - Codierung des Buchstabens B.
- Der (symmetrische) Schlüssel, mit dem Nachrichten an den Server verschlüsselt werden, ergibt sich zu *hash*$(K||H||$„C“$||$session_ID). Hierbei ist „C“ die ASCII - Codierung des Buchstabens C.
- Der (symmetrische) Schlüssel, mit dem Nachrichten an den Client verschlüsselt werden, ergibt sich zu *hash*$(K||H||$„D“$||$session_ID). Hierbei ist „D“ die ASCII - Codierung des Buchstabens D.
- Der (symmetrische) Schlüssel, mit dem MACs für Nachrichten berechnet werden, die an den Client gesendet werden, ergibt sich zu *hash*$(K||H||$„E“$||$session_ID). Hierbei ist „E“ die ASCII - Codierung des Buchstabens E.
- Der (symmetrische) Schlüssel, mit dem MACs für Nachrichten berechnet werden, die an den Client gesendet werden, ergibt sich zu *hash*$(K||H||$„F“$||$session_ID). Hierbei ist „F“ die ASCII - Codierung des Buchstabens F.

K

Kontrollaufgabe 3.10

Erläutern Sie die drei Bestandteile von SSH-2. Welchen Zweck erfüllen die einzelnen Protokolle?

3.8 Sicherheit von SSH

In diesem Abschnitt werden wir sehen, welche Sicherheitseigenschaften SSH bietet. Wie im Fall von SSL ist SSH - bei geeignet gewählten Parametern - theoretisch sicher. Die nächsten beiden Abschnitte begründen kurz diese Sicherheit. Es gibt dennoch Publikationen, etwa Albrecht et al. [2009], die Angriffe auf SSH beschreiben. Diese Angriffe brechen jedoch nicht die zugrundeliegenden kryptografischen Primitiven, sondern nutzen Schwachstellen einzelner SSH Implementierungen aus.

Insertion-Angriffe

Ein aktiver Angreifer kann SSH Pakete abfangen, verändern und anschließend weiterleiten. So könnte er etwa einen Befehl verändern (z.B. statt „Datei kopieren“ den Befehl „Datei löschen“ weiterleiten). Die Integritätsprüfung am Ende jedes SSH-Pakets soll diese Art von Angriffen abwehren. Wie oben bereits erwähnt, ist die CRC-32 Prüfung von SSH-1 nicht sehr stark. Es war einem Angreifer daher möglich SSH-1 Pakete zu verändern. Ein solcher Angriff wurde 1998 von Futoransky und Kargieman erfolgreich durchgeführt, siehe FK9 [1998]. Durch die Einführung von MACs in SSH-2 ist dieser Angriff mit aktuellen Methoden praktisch unmöglich.

3.8.1 Angriffe auf SSH

In Albrecht et al. [2009] geben die Autoren eine Reihe von Angriffen auf die OpenSSH Implementierung an. (Im Folgenden werden wir kurz OpenSSH, statt OpenSSH - Implementierung, schreiben.) Empfängt ein Teilnehmer (z.B. der Server) des Protokolls eine fehlerhaftes Paket (etwa ein Paket, bei dem die Integritätsprüfung fehlschlägt) oder ein fehlerhaft formatiertes Paket (etwa ein zu langes Paket oder eines, welches eine „falsche“ Länge hat), so sendet der Teilnehmer eine Fehlermeldung an den Gegenüber (den Client). Die gesendete Fehlermeldung hängt von der Fehlerursache ab. Genau dies nutzen die Autoren für ihre Angriffe aus. Diese Technik kennen wir bereits aus Abschnitt **??**. Im Folgenden wird einer dieser Angriffe erläutert. Es sei nochmal darauf hingewiesen, dass die Schwachstelle in der Implementierung von OpenSSH lag.

Eigenschaften von OpenSSH

Wenn ein SSH-Paket beim Empfänger eintrifft, wird zunächst geprüft, ob die Länge dieses Pakets korrekt ist. Hierfür wird der Wert des Längenfelds betrachtet. In OpenSSH muss die Länge eines Pakets zwischen (einschließlich) 5 und 2^{18} liegen. Steht im Längenfeld ein Wert, der nicht in diesem Intervall liegt, so wird eine Fehlermeldung, im Folgenden *Fehlermeldung1*, gesendet und die Verbindung wird beendet. Abbildung 3.5 zeigt den Teil des Codes von OpenSSH, der dies veranlasst.

Wurde die „Längenprüfung“ erfolgreich bestanden, so wird geprüft, ob die Paketlänge, bzw. die Menge an Bytes, die im Zuge dieses Pakets noch erwartet werden, ein Vielfaches der Blocklänge der Blockchiffre ist. Ist dies nicht der Fall, so wird

```
if (packet_length < 1 + 4 || packet_length > 256 * 1024) {
   buffer_dump(&incoming_packet);
   packet_disconnect(,,Bad packet length % d.‘‘,
   packet_length); }
```

Abb. 3.5: OpenSSH Quelltext zu Fehlermeldung1

keine Fehlermeldung ausgegeben. Die Verbindung wird jedoch auf TCP - Ebene beendet.
Anschließend wird die Integritätsprüfung durchgeführt. Schlägt diese fehl, so wird eine Fehlermeldung, im Folgenden *Fehlermeldung2*, ausgegeben. Abbildung 3.6 zeigt den Code hierzu. Dabei ist `need = 4 + packet_length - block_size` die Anzahl an Bytes, die in diesem Paket noch erwartet werden, `buffer_len(&input)` die Anzahl an Bytes an, die für dieses Paket schon empfangen wurden und `maclen` gibt die Länge des MAC - Feldes in Bytes an. Solange die Ungleichung erfüllt ist wird keine Nachricht gesendet (SSH_MSG_NONE). Erst, wenn die Ungleichung nicht mehr erfüllt ist, wird der MAC berechnet und es wird evtl. Fehlermeldung2 ausgegeben.
Es folgen weitere Prüfungen, die jedoch für den Angriff nicht von Interesse sind.

```
if (buffer_len(&input) < need + maclen)
   return SSH_MSG_NONE;
```

Abb. 3.6: OpenSSH Quelltext zu Fehlermeldung2

Notation, Grundlagen

Es sei K der Schlüssel, $F_K(F_K^{-1})$ die Verschlüsselungs- (Entschlüsselungs-) Operation und L die Blocklänge der Blockchiffre. Wie in Tabelle 3.2 zu sehen, werden die meisten Blockchiffren bei SSH-2 im CBC - Modus ausgeführt. Der Klartext wird hier in Blöcke, $p_1, p_2, \ldots, p_m$, der Länge L eingeteilt. Die Chiffretextblöcke , $c_1, c_2, \ldots, c_m$, werden dann rekursiv aus diesen Klartextblöcken berechnet:

$$c_i = F_K(c_{i-1} \oplus p_i),\ i = 1,\ 2,\ \ldots,\ m \tag{3.1}$$

wobei c_0 der Initialisierungsvektor ist. Dieser wird zu Begin einer neuen SSH-Session wie in Beispiel 3.2 gebildet. Ansonsten wird paketübergreifend stets der letzte Chiffretextblock verwendet.
Der entsprechende Klartextblock p_i ergibt sich somit zu:

$$p_i = c_{i-1} \oplus F_K^{-1}(c_i),\ i = 1,\ 2,\ \ldots,\ m \tag{3.2}$$

Durchführung des Angriffs

Man nehme an, dass ein Angreifer einen Chiffretextblock c_i^* abgefangen hat, über dessen Klartext p_i^* er Informationen erhalten möchte. Es gilt nach Gleichung 3.2:

$$p_i^* = c_{i-1}^* \oplus F_K^{-1}(c_i^*) \tag{3.3}$$

Hierbei ist c_{i-1}^* der letzte, vor c_i^* gesendete Chiffretextblock und p_i^* der entsprechende Klartextblock, den der Angreifer untersucht.
Man nehme nun an, der Angreifer „injiziere“ den abgefangenen Block c_i^* in eine SSH - Verbindung. Es sei c' der letzte Chiffretextblock, der vor der Injektion gesendet wurde (und somit nun die Funktion des Initialisierungsvektors hat), d.h. es gilt:

$$p' = c' \oplus F_K^{-1}(c_i^*) \tag{3.4}$$

für den resultierenden Klartext p'. Durch Kombination der Gleichungen 3.3 und 3.4 erhält man:

$$p_i^* = c' \oplus c_{i-1}^* \oplus p' \tag{3.5}$$

Wenn der Angreifer die Injektion so platziert, dass sie den ersten Block eines Pakets darstellt, so stellen die ersten 32 Bit des Klartextes p' das Längenfeld des nachfolgenden Pakets dar. Falls der Angreifer nun Fehlermeldung1 empfängt, so liegt die Zahl in diesem Feld nicht im zulässigen Intervall. Andernfalls wird die Verbindung auf TCP - Ebene beendet (Paketlänge ist kein Vielfaches der Blocklänge) oder die Verbindung bleibt bestehen (beide Tests bestanden). Der Angreifer kann also implizit feststellen, ob der Wert des Längenfeldes in Ordnung ist, dadurch, dass die letzten beiden Fälle nicht eintreten.
Für den Wert l des Längenfelds gilt im Falle einer bestandenen Längenprüfung: $5 \leq l \leq 2^{18}$. Das Längenfeld ist allerdings 32 Bit lang. Daher müssen die ersten 14 Bit in diesem Feld alle den Wert 0 haben.
Mit Gleichung 3.5 erhält man so die ersten 14 Bit des Klartextes p_i^*, da c' und c_{i-1}^* bekannt sind.
Wie wahrscheinlich ist so ein Angriff? Zunächst sei darauf hingewiesen, dass es keine große Schwierigkeit darstellt den Block c^* so zu injizieren, dass dieser der erste Block eines Pakets ist. Der Angreifer wartet, bis der Kanal ruhig ist und beginnt dann den Angriff.
Hat er das Paket passend injiziert, so ist der Angreifer genau dann erfolgreich, wenn die ersten 14 Bit des Klartextes p' den Wert 0 haben. Die Wahrscheinlichkeit hierfür ist offenbar $P_A = 2^{-14}$. Man beachte, dass der Angreifer auch 14 Bits raten könnte. Auch hier sind mit Wahrscheinlichkeit $P_R = 2^{-14}$ alle Bits richtig geraten. Der Unterschied zwischen dem „Rateangriff“ und dem oben beschriebenen Angriff besteht jedoch darin, dass der Angreifer nach Durchführung des letzteren die Bits sicher kennt. Wenn die Bits geraten ist, weiß er nicht, ob er richtig geraten hat. Der Angriff ist daher mächtiger als reines Raten.
Falls der Angreifer die Blocklänge der Chiffre kennt, so kann er mit diesem Angriff sogar noch mehr Informationen erhalten. Wenn die Blocklänge etwa 16 ist und der Block beide Längenprüfungen bestanden hat, so ist klar, dass die ersten 14 Bits den Wert 0 haben (s.o.) und außerdem, dass die letzten vier Bits die Zahl 12 codieren. Dies liegt daran, dass die gesamte Paketlänge ein Vielfaches der Blocklänge sein muss. Der Wert des Längenfeldes berücksichtigt das Längenfeld selber (mit einer Länge von 4 Byte) jedoch nicht. Dieses Ereignis wird etwa mit Wahrscheinlichkeit $P = 2^{-18}$ eintreten. In diesem Fall kennt der Angreifer 18 Bit des Klartextes.
Der Angriff lässt sich noch weiterführen. Unter Berücksichtigung von Fehlermeldung2 ist es möglich 32 Bit des Klartextes mit Wahrscheinlichkeit $P = 2^{-18}$ „aufzudecken“. Interessierte Leser seien hier auf Albrecht et al. [2009] verwiesen.
Die Angriffe lassen sich verhindern, wenn das Längenfeld dem Chiffretext unverschlüsselt vorangestellt wird und in die MAC - Prüfung mit einzubeziehen. Dies würde die Länge eines Pakets verraten. Außerdem kann der Angreifer nun nach Belieben den Wert in diesem Feld ändern, diese Unstimmigkeit würde erst später bei der MAC - Prüfung auffallen.

K

Kontrollaufgabe 3.11

Erläutern Sie, wie SSH vor Man-in-the-Middle Angriffen schützt.

Kontrollaufgabe 3.12

- Erläutern Sie, welche Probleme auftreten können, wenn das Längenfeld eines Paketes verschlüsselt wird.
- Welche Probleme treten auf, wenn es nicht verschlüsselt wird?

K

Kontrollaufgabe 3.13

Erläutern Sie den Angriff von Albrecht et al. auf SSH.

K

3.9 Übungsaufgaben

Übung 3.1

- Ist es im Falle eines Mehrbenutzerkontos möglich, dass der Server für das Konto nur einen Schlüssel speichert und trotzdem alle Nutzer Zugriff auf das Konto haben?
- Warum ist dieses Vorgehen nicht zu empfehlen?
  Tip: Was passiert, wenn ein Nutzer seinen Schlüssel weitergibt? Wie groß ist der Aufwand den Nutzer zu sperren, bzw. den weitergegeben Schlüssel auszuschließen?

Ü

Übung 3.2

Warum ist es sinnvoll, wenn neben dem SSH - Client auch der SSH - Server authentifiziert wird?
Tip: Warum wird bei SSL der Server authentifiziert?

Ü

Übung 3.3

Welche Sicherheitsprobleme können bei der Nutzung eines SSH - Agenten auftreten?

Ü

Übung 3.4

Erläutern Sie, wie die Schlüsselverinbarung mittels *diffie-hellman-group1-sha1* funktioniert. Gehen Sie insbesondere darauf ein, wie Server und Client den Wert K berechnen. Wie können beide diesen Wert berechnen?

Ü

Ü

Übung 3.5

Man betrachte den Angriff auf OpenSSH von Albrecht et al. [2009], siehe Abschnitt 3.8.1. Der Block c_i^* ist aus einer bestehenden Verbindung mitgeschnitten worden. Warum kann es dennoch sein, dass nach Injektion die Prüfung des entsprechenden Längenfeldes einen Fehler hervorruft?

3.10 Lösungen zu den Kontrollaufgaben

- Kontrollaufgabe 3.1 auf Seite 23:
  Sicherer Remote-Zugriff durch Client-Authentifizierung ohne Zertifikate.
- Kontrollaufgabe 3.2 auf Seite 26:
  - Probleme: Siehe Abschnitt 3.5. Das Verfahren ist nicht sehr sicher: Gute Passwörter lassen sich schwer merken. Das Passwort während der Übertragung/ auf einem korrumpierten Server abgefangen werden.
  - Das Passwort wird nicht über ein Netzwerk übertragen. Der private Schlüssel lässt sich praktisch nicht erraten.
- Kontrollaufgabe 3.3 auf Seite 26:
  - Bei der ersten Kontaktaufnahme lässt sich die Identität eines Servers nicht sicher feststellen. Es sollte daher auf eine andere Art und Weise die Authenzität des Servers bestätigt werden, etwa durch persönliche Kommunikation und vergleich der öffentlichen Schlüssel.
  - Bei Kontaktaufnahme mit einem „bekannten“ Server wird diesem mit Hilfe seines öffentlichen Schlüssels eine Challenge gesendet. Diese kann er nur korrekt beantworten, falls er den zugehörigen privaten Schlüssel kennt.
    Falls der Server zum Zeitpunkt der ersten Kontaktaufnahme authentisch war, hilft diese Methode dem Client einen Angriff auf seine Verbindung zu erkennen.
- Kontrollaufgabe 3.4 auf Seite 27:
  Falls auch nicht authentisierte Nutzer Schreibzugriff haben, könnten sie eigene Schlüssel in der Liste speichern und sich somit Remote-Zugriff auf das Konto bekommen.
- Kontrollaufgabe 3.5 auf Seite 29:
  x-6 Byte.
- Kontrollaufgabe 3.6 auf Seite 29:
  Der Host-Schlüssel ist ein langfristiger Schlüssel des Servers. Der Server-Schlüssel ist ein kurzweiliger Schlüssel. Dieser wird regelmäßig erneuert. Der Client verschlüsselt den von ihm gewählten Schlüssel mit beiden öffentlichen Schlüsseln nacheinander. Der Server kann diesen Wert entschlüsseln, da er beide privaten Schlüssel kennt. Dies dient der „Forward-Secrecy“. Man nehme an, dass der Server zu einem späteren Zeitpunkt von einem Angreifer korrumpiert wird. Dieser sei an einem bestimmten geheimen Schlüssel „x“ (für ein symmetrisches Verfahren) interessiert. Er kennt das Chiffrat dieses Schlüssels unter dem Host-Schlüssel und einem Server-Schlüssel K. Falls K bereits gelöscht wurde, wird der Angreifer den Wert „x“ nicht berechnen können. Somit bleiben alle Nachrichten, die unter „x“ verschlüsselt wurden, geheim.

- Kontrollaufgabe 3.7 auf Seite 29:
  - Andernfalls müssten sich Client und Server zunächst auf einen Wert einigen, den beide unter dem gemeinsamen Schlüssel verschlüsseln und anschließend vergleichen. Dadurch, dass dieser Wert konstant ist, sparen sie Kommunikationsaufwand.
  - Falls der Client das korrekte Chiffrat erhalten hat, so bedeutet dies, dass der Server den gewählten symmetrischen Schlüssel korrekt entschlüsselt hat. Er kennt also insbesondere den geheimen Host-Schlüssel. Somit ist er authentisch.
- Kontrollaufgabe 3.8 auf Seite 32:
  - Rhosts: Server hat eigene Authorisierungsregeln. Jedem Client wird ein eindeutiger Name zugeordnet. Server prüft, ob der entsprechende Client die Authorisierungsregeln erfüllt. Es wird zusätzlich geprüft, ob das Programm, das Kontakt zum Server aufnimmt, vertrauenswürdig ist.
    Besteht der Client alle Prüfungen, wird er authentifiziert.
    Vorteile: Keine Passworteingabe. Transparent für Nutzer.
    Nachteile: Wie wird jedem Client ein eindeutiger Name zugeordnet? Wie wird geprüft, ob ein Programm vertrauenswürdig ist?
  - RhostsRSA: Wie Rhosts. Client wird mittels User-Schlüssel authentifiziert. Dieser ist jedem Client eindeutig zuzuordnen.
    Vorteile: Client wird eindeutig identifiziert. Es ist möglich nur authorisierten, d.h. vertrauenswürdigen, Programmen Lesezugriff auf den zugeörigen private key zu erlauben. Die Nachteile von Rhosts sind somit behoben.
  - Passwort: Nutzer gibt Passwort zu Nutzerkonto auf entfernten Rechner ein. Ist es korrekt, wird der Nutzer authentifiziert.
    Vorteile: Keine zusätzliche Software. *Nachteile:* Falls Server korrumpiert ist, kennt der Angreifer das Passwort.
  - PublicKey: Client authentifiziert sich mit Hilfe seines public Keys. Dieser ist in einer Liste auf dem entsprechenden Server zu speichern. Ist dies der Fall, so sendet der Server dem Client eine (zufällige) Challenge, die der Client mit dem zugehörigen private Key entschlüsselt. Aus diesem Wert wird die Antwort des Client an den Server generiert. *Vorteile:* Selbst korrumpierter Server gelangt nicht in Besitz des Geheimnisses (Passwort, private Key) des Nutzers.
    Nachteile: Öffentlicher Schlüssel des Nutzers muss vorher auf Server gespeichert werden.
  - Einmal Passwort: Wie Passwort-Authentifizierung. Hier ändert sich jedoch das Passwort bei jedem Login. Es ist einer Liste zu entnehmen oder mittels Software zu generieren (vgl. TAN Liste im Online Banking).
    Vorteile: Sicher, auch wenn Host-Rechner korrupmiert ist. Zugriff von vielen verschiedenen Rechnern möglich, ohne auf allen einen privateKey zu speichern. *Nachteile:* Spezielle Software nötig, bzw. Liste ist immer mitzuführen.
- Kontrollaufgabe 3.9 auf Seite 32: Kein Nutzer mit Administratorfunktion. Es lässt sich daher nicht prüfen, ob ein Programm vertrauenswürdig ist. (Vom Admin installierte Programme werden als vertrauenswürdig angesehen.)
- Kontrollaufgabe 3.10 auf Seite 36: SSH-TRANS: Serverauthentifizierung. Absicherung der Verbidnung.

SSH-AUTH: Client-Authentifizierung.
SSH-CONN: Ausführung von Befehlen.

- Kontrollaufgabe **??** auf Seite **??**: Durch die Methode der Known-Hosts wird ein Man-in-the-Middle Angriff sehr unwahrscheinlich: Der Client sendet dem Server einen von ihm gewaählten, doppelt verschlüsselten Wert. Der Klartext hiervon wird als Schlüssel für das ausgehandelte symmetrische Verschlüsselungsverfahren benutzt (SSH-1). Bei SSH-2 wird der Schlüssel zwischen beiden verianbart.
  Der Angreifer sieht zwar die gesendeten Nachrichten. Ändert er sie ab, wird dies von Client und/oder Server bemerkt und der Verbindungsaufbau wird beendet. Lässt er sie passieren, haben Client und Server einen gemeinsamen Schlüssel. Sie senden nur noch verschlüsselte Nachrichten. Der Angreifer kann diese nicht mitlesen. Falls er sie ändert, wird dies ebenfalls bemerkt.

- Kontrollaufgabe **??** auf Seite **??**: Probleme bei Verschlüsselung: Angenommen, dass das Längenfeld eines Pakets die Länge k hat, d.h. die maximale Zahl, die sich mit diesem Feld codieren lässt, ist $2^k - 1$. In diesem Fall sollte die tatsächliche Paketlänge auch nur durch 0 und $2^k - 1$ beschränkt sein. Die Implementierung sollte also mit Paketen der Länge bis zu $2^k - 1$ umgehen können und insbesondere sollte die Längenprüfung nur dies prüfen. Andernfalls besteht die Möglichkeit Informationen über einen bestimmten Klartextblock zu erhalten.
  Falls das Längenfeld nicht verschlüsselt wird, besteht für den Angreifer die Möglichkeit eigene (möglicherweise sinnlose) Pakete in die Verbidnung zu injizieren.

- Kontrollaufgabe 3.13 auf Seite 39: Die Autoren nutzen eine Schwachstelle in der Implementierung von OPENSSH aus. Die Implementierung gibt, in Abhängigkeit von der Formatierung eines Pakets eine Fehlermeldung aus. Speziell wird ausgenutzt, dass OPENSSH nur Pakete der Länge l mit $5 \leq l \leq 2^{18} - 1$ akzeptiert. Das entsprechende Längenfeld im Paket ist jedoch 32 Bit lang. Falls bekannt ist, dass ein Paket die Längenprüfung bestanden hat, sind also (min.) 14 Bit des Klartextes bekannt. Die Autoren führen diesen Angriff so aus, dass sie die ersten 14 Bit Klartext für beliebige Chiffretextblöcke bestimmen können, nicht nur für die jeweils ersten Blöcke eines Pakets.

Studienbrief 4 PGP

4.1 Lernziele

Sie benennen die Ziele und Funktionen von PGP. Sie geben den Aufbau von PGP Paketen wieder und interpretieren den Aufbau eines Pakets.
Sie erzeugen eigene PGP Schlüsselpaare und verwenden diese.

4.2 Übungsaufgaben

Übung 4.1

Wieso gibt es verschiedene Tags für ein Signaturpaket eines Binärfiles, bzw. eines Textdokuments?

Ü

Übung 4.2

Ein Chipkarten-Hersteller verwendet das RSA-Signaturverfahren auf seinen Chipkarten. Die Signatur wird mithilfe des Chinesischen Restsatzes (CRT) berechnet.

1. Wie sieht der private RSA-Schlüssel aus der auf der Karte gespeichert ist?

2. Gegeben ist ein RSA Public Key (n, e) mit Modulus n und öffentlichem Exponenten e, wobei $e = 0b10001$ und $n = pq$ mit $p = 739.003$, $q = 535.937$.

   a) Berechnen Sie die geheimen Exponenten die zur Signatur mit dem CRT verwendet werden.

   b) Berechnen Sie die Werte a und b in der Formel $1 = ap + bq$.

   c) Berechnen Sie Sie eine gültige Signatur für den Namen 'Bruce'. *Hinweis: Fassen Sie die Binärdarstellung des ASCII-kodierten Wortes als Binärdarstellung einer Ganzen Zahl auf.*

   d) Erzeugen Sie eine fehlerhafte Signatur s' indem Sie m_p für das Wort 'Brucy' berechnen und verwenden.

   e) Bestimmen Sie p und q mit Hilfe von m und s'.

Ü

Ü

Übung 4.3

Gegeben ist die PGP DSA-Schlüsseldatei `secring.skr` mit folgenden Werten:

$p = 929$	(Modulus)
$q = 29$	(Teiler von p-1)
$g = 72$	(Generator der Untergruppe mit Ordnung q)
$y = 568$	(public key)
$x =??$	(secret key fehlt)

Ein Angreifer modifiziert den Wert p aus der Datei `secring.skr` zu $p' = 11$.

1. Warum muss der Wert g für diesen Wert von p' nicht geändert werden?

2. Gegeben ist eine fehlerhafte Signatur $(r, s) = (8, 25)$ für eine Nachricht m mit $h(m) = 0x2a$, die mit den Parametern aus der modifizierten Datei berechnet wurde. Bestimmen Sie den geheimen Schlüssel x aus (r, s).

Ü

Übung 4.4

1. Berechnen Sie eine gültige DSA-Signatur (r, s) für eine Nachricht m mit $h(m) = 0x42$ und Zufallswert $k = 42$ mit den Werten aus der unveränderten Schlüsseldatei `secring.skr`.

2. Verifizieren Sie die oben berechnete Signatur.

Notation

Für eine DSA-Signatur der Nachricht m bezeichnen wir mit r und s die Werte

$$r = (g^k \bmod p) \bmod q$$

und

$$s = (k^{-1} \bmod q) \cdot (h(m) + x \cdot (g^k \bmod p)) \bmod q.$$

Studienbrief 5 S/MIME

5.1 Lernziele

Sie stellen die aktuellen MIME Formate dar und geben die (historischen) Gründe dafür an. Sie interpretieren den Inhalt codierter S/MIME Pakete.
Sie erzeugen eigene MIME-Zertifikate und importieren diese in ihren E-Mail Client. Sie erfassen, welche Teile einer Nachricht durch Signatur geschützt sind.

5.2 Advanced Organizer

S/MIME hat sich heute als Standard für sichere E-Mail weitgehend durchgesetzt. Er soll hier schrittweise anhand seiner Herkunft erläutert werden. Dazu müssen wir das ursprüngliche E-Mail-Format, wie es in [RFC822] beschrieben wurde (aktuelle Version: [RFC2822]), und seine Beschränkungen betrachten. Diese Beschränkungen wurden durch die MIME-Standards aufgehoben, indem neue Datentypen und Codierungen eingeführt wurden. Auf dieser Basis ist es möglich, spezielle Datentypen für die sichere E-Mail-Kommunikation zu definieren.

Auf alternative Vorschläge zur Absicherung von E-Mail-Kommunikation, wie PEM und OpenPGP, gehen wir anschließend kurz ein.

Zum Abruf von E-Mails von einem Mail-Server werden heute zwei Protokolle verwendet: POP3 und IMAP. Die Authentisierungsmechanismen dieser Protokolle runden das Kapitel ab.

5.3 Übungsaufgaben

Übung 5.1

Vergleichen Sie das Schlüsselmanagement bei S/MIME mit dem Schlüsselmanagement bei PGP.
Wie wird bei der öffentliche Schlüssel eines Adressaten bestimmt? Wie bei S/MIME?

Ü

Übung 5.2

Gegeben seien die Bytes $B_1 = 10011001,\ B_1 = 10101001$ und $B_3 = 01101110$. Geben Sie die

- ASCII - Codierung
- Base64 - Codierung
- quoted printable Codierung
- binary Codierung

von $B_1||B_2||B_3$ an.

Ü

Ü

Übung 5.3

Erstellen Sie sich ein selbstsigniertes CA - Zertifikat. Nutzen Sie dieses, um ein MIME-Zertifikat auf ihren Namen auszustellen. Importieren Sie beide Zertifikate in ihren E-Mail Client.
Schicken Sie nun eine S/MIME signierte E-Mail, an die eine Datei angehängt ist, an sich selbst.

Studienbrief 6 DNSSEC

6.1 Lernziele

Sie erklären die Funktionsweise des DNS und geben an, wie Adressen im Internet aufgelöst werden.
Sie benennen Schwachstellen des DNS und begründen die Entwicklung des DNSSEC. Speziell verstehen Sie Angriffe auf das DNS und wie diese durch das DNSSEC verhindert.
Sie zeigen auf, welchen Schutz das DNSSEC bietet.

6.2 Advanced Organizer

Ein zentraler Dienst im Internet ist das Domain Name System (DNS). Er dient dazu, die für menschliche Nutzer leicht zu merkenden Servernamen wie `www.firma.de` oder `mailhost.company.com` in die eigentlichen Adressen des Internet, die IP-Adressen, zu übersetzen (oder auch umgekehrt).
In diesem Abschnitt werden Sie den Ablauf von DNS-Abfragen, sowie den Aufbau von DNS-Paketen kennenlernen. Sie verstehen, warum das DNS in seiner ursprünglichen Form unsicher ist und warum DNSSEC eingeführt wurde.

6.3 Angreifermodell

In diesem Abschnitt verwenden wir anderes Angreifermodell, als in den vorigen Abschnitten. Wir nehmen nun nicht mehr an, dass der Angreifer Kontrolle über das gesamte Netzwerk hat, sondern lediglich, dass er Kontrolle über bestimmte Hosts im Netzwerk hat. Speziell nehmen wir an, dass der Angreifer Zugang zu DNS-Servern hat.

6.4 Übungsaufgaben

Übung 6.1

Ermitteln Sie die IP-Adresse des Webservers `www.nds.rub.de`.

Ü

Übung 6.2

Welche Einträge sind in einer DNS-Response mit DNSSEC eines DNS-Servers enthalten? Wie prüft der Resolver die Antwort?

Ü

Übung 6.3

Ist der Resolver nach der Verifikation der DNS-Response von der Echtheit der Daten überzegut? Begründen Sie ihre Antwort.

Ü

Verzeichnisse

I. Abbildungen

II. Exkurse

III. Kontrollaufgaben

IV. Literatur

Berkley internet name domain, version 9. URL http://www.isc.org/sw/bind.

Technical report. URL http://www.pgpdump.net/.

URL http://zdnet.com.com/2100-1107-851515.html.

URL http://www.wireshark.org/.

URL http://www.thoughtcrime.org/software/sslstrip/.

An attack on crc-32 integrity checks on encrypted channels using cbc and cfb modes. Technical report, 1998. URL http://www.coresecurity.com/files/attachments/CRC32.pdf.

Information technology - abstract syntax notation one (asn.1): Specification of basic notation, 1998a. URL http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-X.680.

Information technology - asn.1 encoding rules: Specification of basic encoding rules (ber), canonical encoding rules (cer) and distinguished encoding rules (der)., 1998b. URL http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-X.690.

The evolving threats to the availability and security of the domain name service, 2003. URL www.sans.org.

André Adelsbach, Sebastian Gajek, and Joerg Schwenk. Phishing - die täuschung des benutzers zur preisgabe geheimer benutzerdaten. In *9. Deutscher IT-Sicherheitskongress des BSI*, 2005.

Martin R. Albrecht, Kenneth G. Paterson, and Gaven J. Watson. Plaintext recovery attacks against ssh. In *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy*, SP '09. IEEE Computer Society, 2009.

Daniel J. Barrett and Richard E. Silverman. *Secure Shell - ein umfassendes Handbuch*. O'Reilly Verlag, 2002.

A. Beutelspacher, J. Schwenk, and K.-D. Wolfenstetter. *Moderne Verfahren der Kryptographie*. Vieweg Verlag, 2001.

Daniel Bleichenbacher. Chosen Ciphertext Attacks against Protocols Based on RSA Encryption Standard PKCS #1. *Advances in Cryptology –CRYPTO'98*, pages 1–12, 1998. URL http://www.bell-labs.com/user/bleichen/bib.html.

D: Boneh, A. DeMillo, and R. J. Lipton. On the importance of checking computations. In *Proceedings Eurocrypt 1997*, 1997.

Douglas E. Comer. *TCP/IP: Konzepte, Protokolle und Architekturen*. Pearson Education Inc., 2003.

Miek Gieben. Dnssec: The protocol, deployment, and a bit of development. *Internet Protocol Journal*, 2004.

Johan Håstad and Mats Näslund. The security of individual rsa bits. In *FOCS*, pages 510–521, 1998.

The SSL Protocol. IETF, 1995. URL http://tools.ietf.org/html/draft-hickman\discretionary{-}{}{}netscape-ssl-00l.

Information technology – Open systems interconnection – The Directory: Public-key and attribute certificate frameworks. ITU-T, 2008. URL http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.509-200811\discretionary{-}{}{}I!!PDF-E&type=items.

Dan Kaminski. This is the end of the cache as we know it, 2008.

V. Klima and T. Rosa. Attack on private signature keys of the openpgp format, pgp tm programs and other applications compatible with openpgp, 2002. URL http://eprint.iacr.org/2002/076.pdf.

RSA Security LLC, 2012. URL http://www.rsasecurity.com/rsalabs/pkcs/.

OpenSSLDevelopmentTeam, 2012. URL http://www.openssl.org/.

Joerg Schwenk. *Sicherheit und Kryptographie im Internet*. Vieweg + Teubner Verlag, 2010.

Joe Stewart. Dns cache poisoning – the next generation. URL http://www.lurhq.com/cachepoisoning.html.

Tatu Ylonen. *The SSH (Secure Shell) Remote Login Protocol*. IETF, 1996. URL http://www.openssh.org/txt/ssh-rfc-v1.5.txt.

Michael Zalewski. *Browser Security Handbook*. Google Inc., 2009.

V. Tabellen