Man in the Middle Attack auf asymmetrische Verschlüsselungsverfahren, insbesondere auf die Secure Shell.

Eine Studienarbeit von Markus Geyer und David Gümbel.
Betreut von Professor Dr. P. Hauck

25.6.2003

Inhaltsverzeichnis

Einleitung

Das Anliegen, geheim miteinander zu kommunizieren, ist sicherlich so alt wie die Kommunikation selbst. Dieses Anliegen wurde immer größer, je wichtiger die Kommunikation für das Leben der Menschen wurde und es entstand eine ganze Wissenschaft, die sich mit der Geheimhaltung von Nachrichten befasst.

Diese Studienarbeit befasst sich mit der Verletzbarkeit von sicheren Kommunikationsmethoden. Im besonderen werden wir einen Angriff auf asymmetrische Verfahren untersuchen, den *Man in the middle Angriff*.

Im ersten Teil werden verschiedene Verfahren zur Verschlüsselung dargestellt und im zweiten Kapitel kurz analysiert. Dort werden auch die theoretischen Hintergründe des Man in the middle Angriffs erläutert. Abschnitt drei befasst sich mit einem der wichtigsten Protokolle sicherer Kommunikation, dem Secure Shell Protokoll (SSH Protokoll). Im vierten Teil wird kurz eine Implementation des SSH Protokolls in der Sprache Java dargestellt und im Kapitel fünf geben wir einen Überblick über das Angriffsprogramm *jmitm2*.

1 Kryptoverfahren

1.1 Allgemeine Bemerkungen

Bei dem Versuch geheim miteinander zu kommunizieren, stehen sich zwei völlig unterschiedliche Ansätze gegenüber, die Kryptologie und die Steganographie. Diese beiden Ansätze unterscheiden sich insofern, als sie ein jeweils anderes Verständnis einer „geheimen"Kommunikation haben. Bei der Steganographie, auf die wir hier nicht näher eingehen werden, soll nicht nur die Nachricht selber , sondern der gesamte Vorgang der Kommunikation geheimgehalten werden. Die Geheimhaltung beruht nun darauf, eine (geheime) Nachricht so zu „verstecken", daß jemand, der nicht weiß, wo und wie er danach suchen muß, diese Nachricht nicht finden kann. Dieses Verstecken kann zum Beispiel in einer unverfänglichen Nachricht oder einem Bild geschehen, ein Verfahren, das im heutigen Computerzeitalter besonders leicht genutzt werden kann. So läßt sich in einer Bilddatei sehr einfach eine Nachricht verstecken, ohne daß ein Betrachter die Manipulation bemerken könnte (beispielsweise mit dem Programm *Steghide*[1] von Stefan Hetzl). Ein weiteres steganographisches Verfahren ist die Verkleinerung von Nachrichten mittels Mikrophotographie. So wurden im 2. Weltkrieg von deutschen Dienststellen sogenannte *Microdots* von der Größe eines Schreibmaschinenpunktes verwendet, die den Inhalt einer ganzen DinA4 Seite fassen konnten. Bei der Stega-

[1] http://freshmeat.net/projects/steghide/ [Mai 2003]

nographie ist also zwischen den einzelnen Kommunikationsteilnehmern ein geheimes Absprechen der verwendeten Technik nötig, damit das Verfahren funktioniert. Aus diesem Grund werden steganographische Methoden wohl vor allem von Geheimdiensten oder von Kriminellen verwendet, bei denen es nicht nur auf die Geheimhaltung der Nachricht, sondern auch auf die Verheimlichung einer (geheimen) Kommunikation überhaupt ankommt.

Im Gegensatz dazu ist die Zielsetzung der Kryptologie eine andere. Hier geht es nur um die Geheimhaltung des Inhalts einer Nachricht. Die oben kurz dargestellte Steganographie kann nicht zum Gebiet der Kryptologie gezählt werden, da sie einer der – von Claude E. Shannon formulierten – Maximen der Kryptologie, „The enemy knows the system being used“, widerspricht. Nach Shannon darf also die Sicherheit eines Kryptoverfahrens nicht auf der Geheimhaltung dieses Verfahrens beruhen, da man jene sowieso nicht gewährleisten kann, wie uns zahlreiche Beispiele aus der Geschichte der Kryptologie zeigen.

Im Laufe der Zeit entwickelten sich immer ausgefeiltere Kryptosysteme zur Verschlüsselung von Nachrichten. Bis 1976 hatten jedoch alle diese Systeme gemein, daß man eine Schlüsselinformation des Verfahrens (den sogenannten „Schlüssel“) auf geheimem Wege übertragen musste, um die Ver- und spätere Entschlüsselung durchzuführen. Solche Verfahren heißen *symmetrische Verschlüsselungsverfahren*, da bei ihnen zur Verschlüsselung und zur Entschlüsselung im wesentlichen die selben Schritte unter Verwendung derselben Schlüsselinformation durchgeführt werden (siehe Abb.1). Die sym-

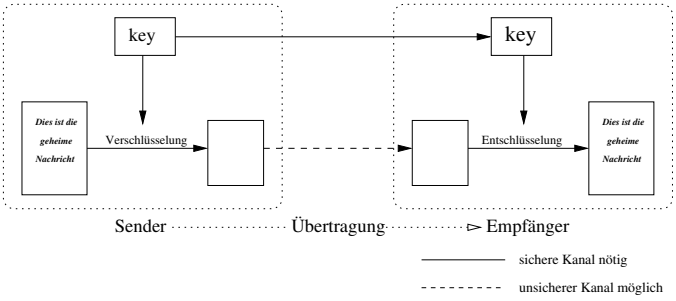

Abbildung 1: Verschlüsselung mittels eines symmetrischen Verfahrens

metrischen Verfahren benötigen also einen sicheren Kanal, um den Schlüssel auszutauschen. Dabei stellt sich natürlich die Frage, warum man nicht gleich die Nachricht auf sicherem Wege übermittelt, wenn man sowieso schon einen sicheren Kanal benötigt. Da man sich den Zeitpunkt der Schlüsselübertragung vor Übermittlung der Nachricht frei wählen und natürlich auch mehrere Schlüssel auf einmal übermitteln kann, hat man mit symmetrischen Verfahren die Möglichkeit, zu einem späteren Zeitpunkt eine geheime Nachricht

über einen unsicheren Kanal zu übermitteln.
Ein weiteres Problem kann in der Anzahl der benötigten Schlüssel liegen. Bei n Partnern, die jeder mit jedem gesichert Daten austauschen möchten, sind $\binom{n}{2} = \frac{n \cdot (n-1)}{2}$ Schlüssel nötig, um bei symmetrischen Verfahren die sichere Kommunikation zwischen allen Teilnehmern sicherzustellen. Bei nur 1000 Teilnehmern wären dies bereits 499500 Schlüssel.

Das Problem mit der sicheren Schlüsselübertragung war bis vor kurzem nicht besonders hinderlich gewesen, da die hauptsächlichen Nutzer von Verschlüsselungsverfahren, nämlich Geheimdienste und das Militär, Möglichkeiten hatten, ihre Schlüssel geheim zu verteilen. Erst als sichere Kommunikation eine immer größere Rolle auch im zivilen Leben spielte, vor allem auch durch die Verbreitung von Computer und Internet, wurde darüber nachgedacht, wie man diesen, für den Privatmenschen schwer durchführbaren, Schlüsselaustausch vermeiden oder vereinfachen könne. Das war die Geburtsstunde der sogenannten *asymmetrischen Verschlüsselungsverfahren*, die ihren Namen der Tatsache verdanken, daß man zwei unterschiedliche Schlüssel benötigt, einen zur Verschlüsselung, einen anderen zur Entschlüsselung von Nachrichten (siehe Abb.2).

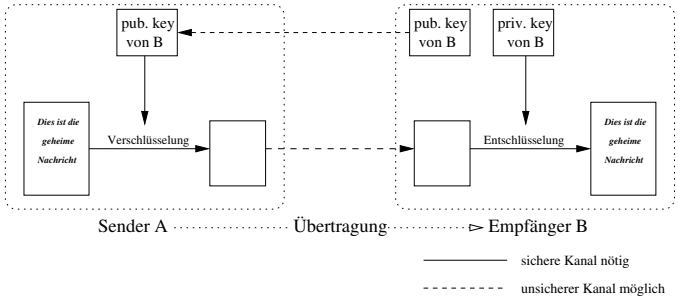

Abbildung 2: Verschlüsselung mittels eines asymmetrischen Verfahrens

Damit ist das Problem der sicheren Schlüsselübertragung gelöst, da der Schlüssel zur Verschlüsselung öffentlich gemacht werden kann und somit jedem, der eine Nachricht an den Besitzer dieses Schlüssels senden will, zur Verfügung steht. Natürlich muß hierfür sichergestellt werden, daß man, ausgehend vom sogenannten *öffentlichen* Schlüssel (also dem Schlüssel, der für die Verschlüsselung verwendet wird), keine Rückschlüsse auf den sogenannten *geheimen* Schlüssel (also dem zur Entschlüsselung verwendeten Schlüssel) ziehen kann.

Im Gegensatz zu den symmetrischen Verfahren benötigen jetzt zwei Partner A und B für eine wechselseitige Nachrichtenverbindung vier Schlüssel:

den öffentlichen Schlüssel (*public key*) von A, den B zur Verschlüsselung benutzt, den privaten Schlüssel (*private key*) den nur A kennt und den A zur Entschlüsselung verwendet und natürlich den öffentlichen und den privaten Schlüssel von B (in Abb.2 ist nur die Übermittlung der Daten von Sender A zu Empfänger B dargestellt, weshalb hier nur 2 Schlüssel benötigt werden). Trotz dieser scheinbaren Vervierfachung der benötigten Schlüssel im Gegensatz zu den symmetrischen Verfahren ist die Anzahl der benötigten Schlüssel für die Kommunikation mit mehreren Partner bei asymmetrischen Verfahren deutlich geringer als bei den symmetrischen. Denn jeder Partner der mit A sicher kommunizieren möchte, benutzt nun denselben öffentlichen Schlüsel von A. Somit benötigt man in einem Netzwerk von n Partnern nur noch n asymmetrische Schlüsselpaare im Gegensatz zu den $\binom{n}{2}$ symmetrischen Schlüsseln.

Allerdings sind die meisten symmetrischen Verschlüsselungsverfahren deutlich schneller bei der Ver- und Entschlüsselung von Nachrichten, als asymmetrische Verfahren, bei vergleichbarer Sicherheit. Aus diesem Grund wird häufig ein asymmetrisches Verfahren dazu verwendet, einen Schlüssel für ein symmetrisches Verfahren geheim zu übermitteln, während die eigentliche Kommunikation danach mit einem symmetrischen Verfahren verschlüsselt wird. Solche kombinierten Verfahren nennt man auch *hybride Verschlüsselungsverfahren*. Außerdem tritt bei den asymmetrischen Verschlüsselungsverfahren ein neues Problem auf. Der öffentliche Schlüssel des Empfängers ist allgemein bekannt, also ist jeder in der Lage, ihm eine verschlüsselte Nachricht zukommen zu lassen. Der Empfänger steht daher vor dem Problem, bei einer empfangene Nachricht sicherzustellen, daß die Nachricht wirklich von dem vorgegebenen Empfänger stammt. Diese Problematik wird von Signaturverfahren aufgegriffen, die auf den asymmetrischen Verschlüsselungsverfahren beruhen.

Da diese Arbeit vor allem einen Angriff auf asymmetrische Verfahren, der *man in the middle attack*, zum Thema hat, werden im folgenden Abschnitt die asymmetrischen Verschlüsselungsverfahren näher erläutert. Dabei gehen wir auch auf einige Standardbeispiele ein und vergleichen diese mit den symmetrischen Verfahren.

1.2 Asymmetrische Verschlüsselungs- und Signaturverfahren

Das Konzept der *asymmetrischen Verschlüsselungsverfahren* oder auch *public key*–Verfahren stammt von Whitfield Diffie und Martin E. Hellman [**?**]. Alle asymmetrischen Verfahren beruhen auf Funktionen, für die eine Umkehrfunktion ohne die Kenntnis einer Zusatzinformation nicht in vertretbarer Zeit berechnet werden kann. Dazu sollte man festhalten, daß es keinen Existenzbeweis für diese sogenannten *Ein-Weg-Falltür Funktionen (One-Way-Trapdoor Functions)* gibt. Allerdings gibt es eine Reihe von Kandidaten solcher Funktionen, für die man zwar nicht beweisen kann, daß sie diesen Anforderungen genügen, bei denen aber auch nach intensiven und langjährigen Studium kein Algorithmus bekannt ist, eine Umkehrfunktion in vertretbarer Zeit auch ohne Zusatzinformation zu berechnen.

Im folgenden werden wir kurz auf einige der wichtigsten Beispiele asymmetrischer Verschlüsselungsverfahren eingehen. Dabei heißen die beiden Kommunikationspartner Alice und Bob oder kurz A und B.

1.2.1 Der Diffie–Hellman–Schlüsselaustausch

Dieses von Diffie und Hellman 1976 entwickelte Verfahren benutzte erstmals Funktionsweisen, die auch bei asymmetrischen Verschlüsselungsverfahren verwendet werden. Allerdings ist der Diffie–Hellman–Schlüsselaustausch kein Verschlüsselungsverfahren im eigentlichen Sinne, da gar keine Nachricht übermittelt wird. Es beschäftigt sich nur mit der Frage, wie zwei Personen, denen nur ein potentiell unsicherer Kanal zur Kommunikation zur Verfügung steht, über ebendiesen Kanal einen Schlüssel vereinbaren können, ohne daß ein eventueller Lauscher in den Besitz dieses Schlüssels kommt. Wenn das gelungen ist, können die beiden Kommunikationspartner mittels eines symmetrischen Verfahrens geheim Nachrichten austauschen. Die Funktion, die in diesem Verfahren die Sicherheit garantieren soll, ist der diskrete Logarithmus, also der Logarithmus in einem endlichen Körper. Im folgenden werden wir das Verfahren Schritt für Schritt darstellen.

Zur Vereinfachung nehmen wir an, daß der (geheime) Schlüssel auf den sich die beiden Partner einigen wollen eine ganze Zahl ist. Zunächst einigen sich Alice und Bob auf eine Primzahl p und ein Element $g \in \mathrm{F}_p^\star$. Im nächsten Schritt wählt Alice (geheim) eine zufällige ganze Zahl $k_A < p$ von etwa der selben Größenordnung wie p, berechnet $g^{k_A} \bmod p$ und sendet das Ergebnis an Bob. Analog wählt sich Bob ein k_B und sendet $g^{k_B} \bmod p$ an Alice.

Der Schlüssel auf den sie sich nun verständigt haben, ist die Zahl

$$g^{k_A \cdot k_B} \bmod p.$$

Bob erhält diesen Schlüssel, indem er die von Alice erhaltene Zahl $g^{k_A} \mathrm{mod} p$ in die k_B–te Potenz modulo p erhebt und analog erhält Alice den selben Schlüssel, wenn sie die von Bob erhaltene Zahl in die k_A–te Potenz modulo p erhebt. Ein möglicher Angreifer müsste also aus dem abgehörten Nachrichtenverkehr den Schlüssel errechnen. Anders ausgedrückt müsste er aus der Kenntnis von p, g, g^{k_A} und g^{k_B} den Schlüssel $g^{k_A} \cdot k_B$ berechnen. Dieses Problem nennt man auch das „Diffie–Hellman–Problem“. Man sieht leicht, daß jemand, der in der Lage wäre, den diskreten Logarithmus effektiv zu berechnen, sofort auch das Diffie–Hellman–Problem gelöst hätte. Er könnte aus g und g^{k_A} effektiv k_A berechnen und erhielte dann sofort (wie Alice) $g^{k_B \cdot k_A}$. Über die umgekehrte Implikation ist jedoch nichts bekannt, d.h. es wäre unter Umständen möglich, das Diffie–Hellman–Problem zu lösen, ohne überhaupt einen diskreten Logarithmus zu berechnen. Der Diffie–Hellman–Schlüsselaustausch wird heute noch in zahlreichen hybriden Verschlüsselungsverfahren verwendet.

Im Gegensatz zu diesem reinen Schlüsselaustauschverfahren stellt das von T. ElGamal entwickelte *ElGamal–Verfahren* (vgl. [**?**]) ein vollständiges asymmetrisches Verschlüsselungsverfahren auf der Basis des diskreten Logarithmus dar. Anders als das RSA–Verfahren, das wir in (Abschnitt 1.2.3) noch genauer kennenlernen werden, sind diese Verfahren variabler. Der diskrete Logarithmus läßt sich nicht nur in endlichen Körpern formulieren, sondern in beliebigen endlichen Gruppen. Aus diesem Grund ist die Auswahl an algebraischen Grundstrukturen für diese Art von Verfahren besonders reichhaltig. Als Schlagwort seien hier nur die Verschlüsselungsverfahren auf elliptischen Kurven genannt. Es scheint so, als wären die Verfahren, die auf dem diskreten Logarithmus beruhen, zumindest in der näheren Zukunft die sichersten und schnellsten public–key Verfahren.

Bei der bisherigen Untersuchung der Schlüsselaustauschverfahren blieb bisher ein zentraler Punkt der asymmetrischen Kryptologie, die Authentifizierung von Nachrichten, unbeachtet. Diese ist bei der asymmetrischer Kryptologie von entscheidender Bedeutung. Eine Nachricht von der der Empfänger nicht sicher weiß, von wem sie geschickt wurde, ist in vielerlei Hinsicht wertlos. Im folgenden werden wir den prominentesten Vertreter eines Signaturverfahren auf Basis des diskreten Logarithmus genauer untersuchen.

1.2.2 Der Digital Signature Algorithm (DSA)

Bei DSA handelt es sich um einen offiziellen US-Standard namens DSS (Digital Signature Standard), der von der amerikanischen Behörde „National Institute of Standards and Technology"(NIST) vorgeschlagen wurde. Das Verfahren basiert auf dem diskreten Logarithmus Problem in einem Primkörper F_p. Um in der Lage zu sein, digitale Signaturen nach dem DSS-Standard durchzuführen, geht jeder Benutzer wie folgt vor (siehe Abb.3):

1. Wähle eine Primzahl q von etwa 160 Bit Länge.
2. Wähle eine zweite Primzahl $p = 1 \bmod q$ von etwa 500 Bit Länge. Die vom Standard vorgegebene Länge liegt bei einem Vielfachen von 64 Bits und zwischen 512 und 1024 Bit.
3. Wähle einen Erzeuger g der zyklischen Untergruppe F_p^*. Dies geschieht durch Berechnung von $g_0^{\frac{(p-1)}{q}} \bmod p$ für ein zufälliges g_0; falls diese Zahl ungleich eins ist, ist g_0 ein Erzeuger.
4. Wähle eine zufällige Ganzzahl x im Intervall $0 < x < q$ als privaten Schlüssel und lege den öffentlichen Schlüssel mit $y = g^x \bmod p$ fest.

Um nun tatsächlich eine Nachricht zu signieren, berechnet der Benutzer zunächst einen Hashwert[2] H mit $0 < H < q$ der zu signierenden Nachricht. Dann wählt er eine Zufallszahl k im gleichen Intervall, berechnet $g^k \bmod p$, und setzt $r = (g^k \bmod q) \bmod p$. Schließlich findet er noch eine Zahl s mit $sk = H + xr \bmod q$. Die Signatur ist nun das Paar (r, s) von Ganzzahlen modulo q. Die Verifikation von Nachrichten funktioniert nun wie folgt (siehe Abb.4):

1. Berechne $u = s^{-1}H \bmod q$.
2. Berechne $t = s^{-1}r \bmod q$.
3. Überprüfe $g^u y^t \bmod q = r$.

[2]Eine Hashfunktion h berechnet zu einer beliebig langen Datenmenge M einen Hashwert $h(M)$ fester Länge m, wobei folgende Eigenschaften gelten müssen:

1. Bei gegebener Datenmenge M ist der Hashwert $h(M)$ leicht zu berechnen.
2. Bei gegebenem Hashwert $h(M)$ ist es nahezu unmöglich, M zu berechnen.
3. Es ist praktisch unmöglich, zu einer Nachricht M mit Hashwert $h(M)$ eine weitere Nachricht M' mit dem gleichen Hashwert zu generieren.

Falls diese Gleichung erfüllt ist, ist die Signatur gültig. Zu beachten ist, daß nur jemand mit Kenntnis von x (dem privaten Schlüssel) in der Lage ist, ein gültiges s zu berechnen. Daß die Beziehung $g^u y^t \bmod q = r$ tatsächlich die Gültigkeit einer Signatur angibt, sieht man durch simples Einsetzen, denn es gilt:

$$g^u y^t \equiv g^u g^{xt} \equiv g^{\frac{(H+xr)}{s}} \equiv g^k \equiv r \bmod q.$$

Nachdem wir bisher nur Kryptosysteme auf der Basis des diskreten Logarithmus betrachtet haben, wenden wir uns nun einem weiteren Prominenten Vertreter der public–key Verfahren zu, dem RSA Verfahren.

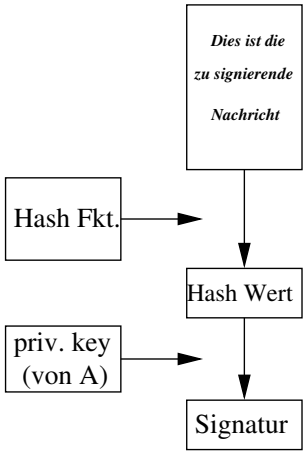

Abbildung 3: Signieren einer Nachricht

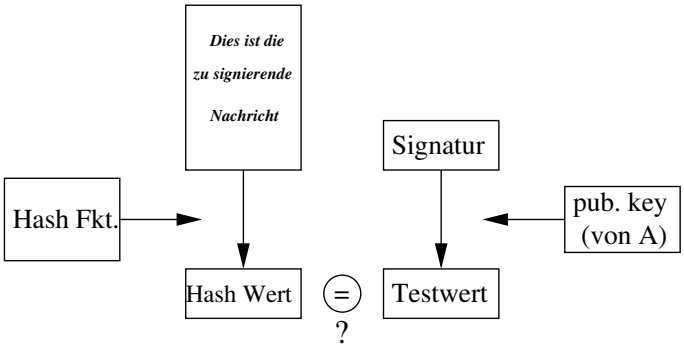

Abbildung 4: Prüfen der Signatur einer Nachricht

1.2.3 Das RSA–Verschlüsselungs- und Signaturverfahren

Nur kurz nach der Veröffentlichung von Diffie und Hellman, stellten Ron Rivest, Adi Shamir und Leonard Adleman 1977 ein voll funktionsfähiges public–key Verschlüsselungsverfahren vor [**?**], das unter dem Namen RSA bekannt wurde. Es ist sicher eines der am häufigst benutzten public–key Verfahren überhaupt, auch wenn es langsam aber sicher von leistungsfähigeren Verfahren abgelöst wird. Das Verfahren ist immer noch in viele Cryptostandards integriert und wird es aufgrund seiner Verbreitung sicher auch noch eine ganze Zeit lang bleiben. Die Funktion, die dem RSA Verfahren zugrunde liegt, ist die Multiplikation, bzw. das Faktorisieren, in einem endlichen Körper. Dazu sei hier nur gesagt, daß es viel einfacher ist, $a \cdot b (= c)$ zu berechnen, als $c (= a \cdot b)$ in seine Primfaktoren zu zerlegen. RSA verwendet, da wir es mit einem public–key Verfahren zu tun haben, je einen öffentlichen Schlüssel und einen dazugehörigen privaten Schlüssel für jeden Kommunikationsteilnehmer. Die Schlüssel werden nun wie folgt berechnet:

1. Erzeuge zwei Primzahlen p und q von ungefähr halber gewünschter Schlüssellänge.
2. Berechne $n = pq$ und $\varphi(n) = (p-1)(q-1)$.
3. Wähle e mit $\mathrm{ggT}(e, \varphi(n)) = 1$ (mittels euklidschem Algorithmus).
4. Berechne das durch die Bedingung $e \cdot d \equiv 1 (\mathrm{mod}\ \varphi(n))$ eindeutig bestimmte d.

Nun kann man den öffentlichen Schlüssel (n, e) bekannt machen und muß nur dafür sorgen, daß niemand den privaten Schlüssel d zu sehen bekommt. Die Verschlüsselung einer Nachricht $m \in [0, n-1]$ erfolgt nun wie folgt (auch hier nehmen wir die zu verschlüsselnde Nachricht als eine natürliche Zahl an). Die Nachricht wird mit dem öffentlichen e potenziert und modulo n reduziert. Also ist die verschlüsselte Nachricht nun $m^e (\mathrm{mod}\ n)$. Die Entschlüsselung funktioniert ganz ähnlich: die erhaltene Nachricht wird mit dem geheimen d potenziert und wiederum modulo n reduziert, man erhält also $(m^e)^d = m^{ed} (\mathrm{mod}\ n)$. Da aber e und d so gewählt wurden, daß $ed \equiv 1 (\mathrm{mod}\ \varphi(n))$ also insbesondere $ed \equiv 1$ modulo $(p-1)$ bzw. modulo $(q-1)$ gilt, ergibt sich mit dem kleinen Satz von Fermat[3] sofort, daß $m^{ed} \equiv m (\mathrm{mod}\ p)$ und $m^{ed} \equiv m (\mathrm{mod}\ q)$ und damit natürlich auch $m^{ed} \equiv m (\mathrm{mod}\ n)$.

[3] (Fermat 1640) Sei p Primzahl und p teilt nicht $a \in \mathbb{Z}$. Dann ist $a^{p-1} \equiv 1 \bmod p$. Für alle $a \in \mathbb{Z}$ ist $a^p \equiv a \bmod p$. (vgl. [**?**] S.35)

Im folgenden werden wir noch auf das zweite Anwendungsgebiet von RSA eingehen, das digitale Signieren. Man verwendet hierzu dieselben Schlüssel wie bei der Chiffrierung, nur tauschen sie die Rollen.

Eine RSA Signatur für eine Nachricht M wird mit Hilfe einer Hashfunktion H folgendermaßen erzeugt (vgl. Abb.3):

1. Bilde den Hashwert h von M: $h = H(M)$.
2. Bilde die Signatur s durch $s = h^d \bmod n$.

Die zu der Nachricht M gehörige Signatur ist das berechnete s. Für die Verifikation der Signatur werden die Nachricht M, die Signatur s und der öffentliche Schlüssel (e, n) benötigt. Sie umfaßt folgende Schritte (vgl. Abb.4):

1. Berechne den Hashwert h von M.
2. Potenziere die Signatur s mit dem öffentlichen Exponenten e und reduziere modulo n. Berechne also $h' = s^e \bmod n$.
3. Gilt nun $h = h'$, so ist s eine gültige Signatur für M. Sonst ist die Signatur ungültig.

Die Authentizität wird hierbei dadurch sichergestellt, daß nur der Besitzer des privaten Schlüssel die Signatur erzeugen kann.

In Abbildung 5 wird nun eine typische signierte Kommunikation mittels eines asymmetrischen Verfahren dargestellt (z.b. das signierte Versenden und Empfangen einer email).

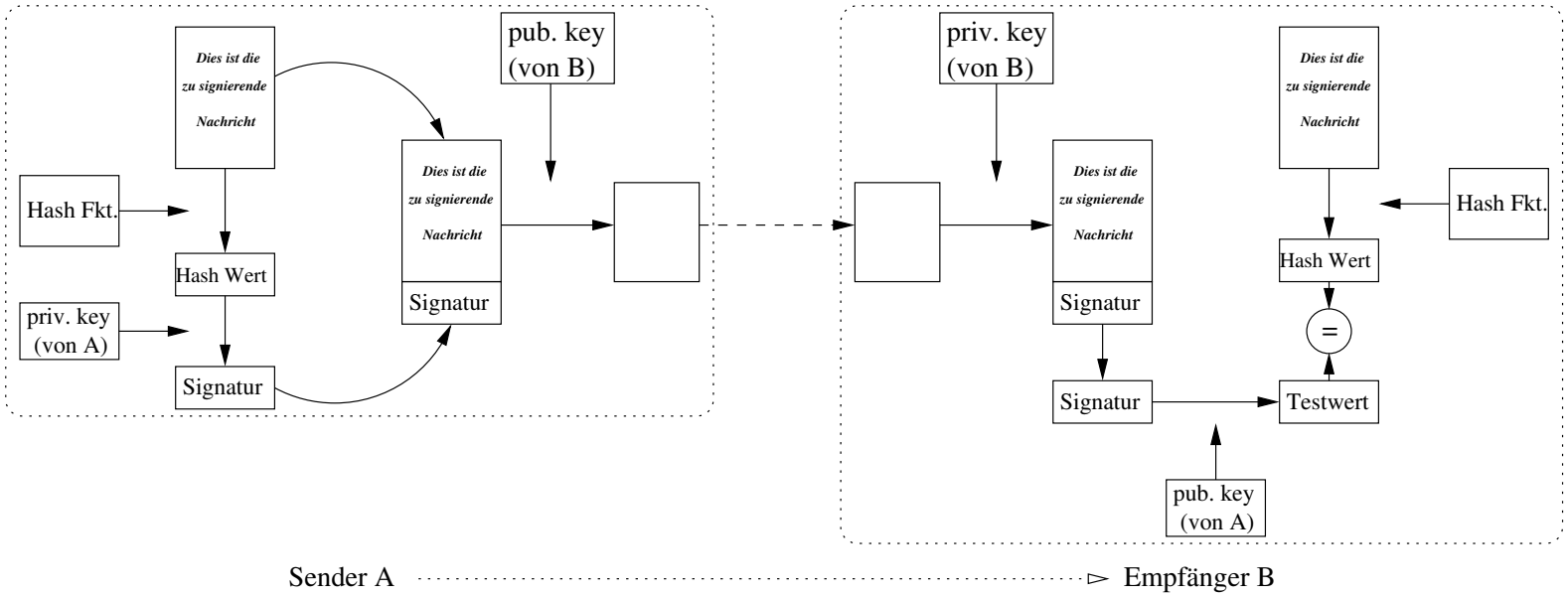

Abbildung 5: Verschlüsselte Kommunikation mit digitaler Signatur

2 Kryptoanalyse

2.1 Historie

Parallel zu den Kryptoverfahren entwickelten sich auch immer bessere Verfahren zur Kryptoanalyse, also zum Brechen von Verschlüsselungen.

Die quantitative Bewertung der Sicherheit der Verfahren wurde jedoch erst durch die bahnbrechenden Ideen von *Claude E. Shannon*[4] möglich. Wie wenig diese Bewertung der Brauchbarkeit eines Verfahrens selbst heute noch ausgeprägt ist zeigt sich bei der Betrachtung einiger Kryptoverfahren, die ihren Zweck eindeutig verfehlt haben. Als Beispiel sei das CSS – Verfahren genannt, mit dem alle DVD–Filme verschlüsselt werden, welches aus kryptologischer Sicht jedoch völlig unbrauchbar ist. Aber auch das eigentlich renomierte Unternehmen der deutschen Telekom verrechnete sich bei der Bewertung der Sicherheit des von ihm entwickelten Kryptostandard *Magenta*[5], den es in den Wettbewerb um den neuen Kryptostandard AES (Advanced Encryption Standard) einbrachte. Bereits wenig später war das Verfahren von den Kryptospezialisten gebrochen worden.

Bei der Untersuchung von Schwächen von Verschlüsselungsverfahren, muß man verschiedene Schwächen unterscheiden. Da sind zum einen Angriffsmöglichkeiten, die durch Schwächen im verwendeten Kryptoverfahren ermöglicht werden. Das sind natürlich die am meisten ge- und untersuchten Angriffe, da sie sich am besten eignen, das untersuchte Kryptoverfahren zu testen und unter Sicherheitsaspekten einzuschätzen. Hier sind sowohl Kryptoanalysen einzuordnen, die aufzeigen, daß das dem Verfahren zugrunde liegende mathematische Problem doch (evtl. in Spezialfällen) leichter zu lösen ist als angenommen, aber auch solche Analysen, die eine Entschlüsselung ohne Lösung des zugrunde liegenden Problems (aber auf mathematischem Wege) zustande bringen. Beiden dieser Analysen ist jedoch gemein, daß sie nur die theoretischen Aspekte der Kryptoverfahren berücksichtigen. Sie gehen also von einer perfekt implementierten Version der Verfahren aus und benützen nur die anerkannt unsicheren Infomationen der Verfahren, nämlich die Kenntnis des verwendeten Verfahrens, bei public–key–Verfahren die Kenntnis der öffentlichen Schlüssel und natürlich die abgehörte (chiffrierte) Nachricht.

[4] C.E.Shannon stellte fünf Maßstäbe zusammen, die an ein Kryptoverfahren angelegt werden sollten: Zum ersten die Krytologische Sicherheit, zum zweiten die Schlüssellänge, zum dritten die Praktische Durchführung der Chiffrierung und Dechiffrierung, zum vierten die Aufblähung des Geheimtextes gegenüber des +Klartextes, und zum fünften die Verschleppung von Chiffrierfehlern. [**?**]

[5] Eine Kryptoanalyse des Magenta Verfahrens findet sich unter http://www.counterpane.com/magenta-cryptanalysis.ps.gz [Mai 2003]

Eine weitere Klasse von Angriffsmöglichkeiten eröffnet sich, wenn man die Probleme der Wirklichkeit mit einbezieht. Neben tatsächlichen Fehlern in der Implementation von Kryptoverfahren gibt es jedoch auch Grundsätzliche Schwächen der Verwendeten Infrastruktur. Auf diese werden wir in Kap.3.3 näher eingehen. Im nächsten Abschnitt werden wir kurz auf die Analyse der in Teil 1.2 beschriebenen Verfahren eingehen, wobei hier unser Augenmerk der Untersuchung der mathematischen Verfahren gilt und nicht etwa den Schwächen in der Implementierung oder der Infrastruktur.

2.2 Analyse der asymmetrischen Verfahren

Eine Problem bei der Bewertung der Sicherheit eines Kryptoverfahrens ist, daß fast nie eine positive Aussage über die Sicherheit gemacht werden kann. Oder mit anderen Worten kann man zumeist nur Schwächen und mögliche Angriffspunkte aufdecken. Ein Beweis der Sicherheit eines Verfahrens ist – bis auf eine Ausnahme [6] – nicht möglich. Diese Schwierigkeit liegt an zwei kritischen Stellen. Zum ersten ist nicht genau bekannt, wie schnell sich die verwendeten Verschlüsselungsfunktionen invertieren lassen. Zum anderen, wäre selbst dann noch nicht sichergestellt, daß es keinen anderen Weg zum Klartext gibt, also unter Umgehung zumindest von Teilen des Verschlüsselungsverfahren. Somit ist auch bei den asymmetrischen Verfahren nur ein Einschätzen der kryptologischen Sicherheit möglich.

2.2.1 Diffie Hellman / DSA

Das zugrunde liegende Problem dieses Verfahrens ist es, den diskreten Logarithmus in einer endlichen Gruppe zu berechnen. Dazu gibt es eine Reihe von Ansätzen, die alle eine andere Strategie verfolgen und zum Teil nur in einigen speziellen endlichen Gruppen eingesetzt werden können. Am besten auf dieses Problem hin untersucht, sind die multiplikativen Gruppen endlicher Körper $\mathbb{F}_p$. Hier finden sich auch die schnellsten Algorithmen zum Berechnen des diskreten Logarithmus. Allerdings besitzen sogar sie noch eine Laufzeit von $O(e^{(1+o(1))\cdot\sqrt{ln(p)ln^2(p)}})$. Der wichtigste Vertreter dieser Algorithmen ist wohl die Index–Calculus–Methode. Geht man bei diesem Krypto–System jedoch auf andere Gruppen über, greifen die meisten dieser Algorithmen nicht mehr und man muß auf allgemeinere aber auch deutlich langsamere Algorithmen zurückgreifen. Im Falle der Verschlüsselung auf Basis elliptischer Kurven sind

[6] Es gibt ein Kryptoverfahren, das beweisbar maximale Sicherheit gewährleistet. Dieses symmetrische Kryptoverfahren benutzt einen Schlüssel der mindestens genauso lang ist wie die Nachricht und führt ein logisches xor zwischen Nachricht und Schlüssel durch. Man nennt dieses Verfahren den *One Time Pad*

noch keine vergleichbaren allgemeinen Algorithmen zur Lösung des diskreten Logarithmus Problem bekannt. Nur für spezielle Klassen von von elliptischen Kurven sind einigermaßen schnelle Algorithmen bekannt [**?**].

2.2.2 RSA

Bei diesem Verfahren ist das der Invertierung zugrunde liegende mathematische Problem die Faktorisierung großer Zahlen. Dieses Problem ist in der Mathematik schon sehr lange bekannt und auch eingehend untersucht worden. Auch hier gibt es eine reiche Anzahl an Algorithmen unterschiedlichster mathematischer und algorithmischer Komplexität. Zu den schnellsten Algorithmen zählen hier das Zahlkörpersieb und das Faktorisieren in einem endlichen Körper $\mathbb{F}_p$ mittels elliptischer Kurven. Die Laufzeit dieser Algorithmen beträgt $O(e^{(1,92+o(1))\cdot\sqrt[3]{ln(n)ln^2(n)}})$, bzw. $O(e^{(1+o(1))\cdot\sqrt{2ln(p)ln^2(p)}})$.

Diese kurze Einführung in die Komplexität der Algorithmen soll hier für unsere Bedürfnisse genügen. Mehr zu den einzelnen Algorithmen finden sich z.B. in [**?**], [**?**] oder auch in [**?**].

2.2.3 Die „Man in the Middle Attack“

Nach diesen allgemeinen Vorbemerkungen widmen wir uns nun dem zentralen Thema unserer Arbeit, der man in the middle attack. Hier werden wir zuerst einmal die theoretischen Hintergründe dieses Angriffs darlegen, während wir in Abschnitt 3.3.3 auf die technischen Details einer Implementierung dieses Angriffes eingehen werden.

Als erstes werden wir einen beispielhaften Angriff eines Lauschers E auf eine Kommunikation von A und B näher betrachten (siehe Abb. 6). Als erstes fängt E den öffentlichen Schlüssel von B ab und schickt an A seinen eigenen öffentlichen Schlüssel. A ist jetzt der Ansicht den öffentlichen Schlüssel von B erhalten zu haben und beginnt ihre Nachricht zu verschlüsseln und an B zu senden. Der Lauscher E fängt diese Nachricht ab, und entschlüsselt sie (was ihm ja möglich ist, da er ja A seinen eigenen öffentlichen Schlüssel untergeschoben hat). Dann verschlüsselt er sie mit dem öffentlichen Schlüssel von B, den er ja zu Beginn abgefangen hat und schickt die so verschlüsselte Nachricht an B. B entschlüsselt ganz normal diese Nachricht und weder A noch B bemerken, daß E die (entschlüsselte) Nachricht mitgelesen oder sogar verändert hat.

Die beiden Voraussetzungen für einen man in the middle Angriff werden aus diesem Beispiel deutlich. Zum einen muß der Angreifer schon vor einem Kommunikationsversuch auf eben diesen warten. Die zweite und augenscheinlich schwieriger zu erfüllende Voraussetztung ist die Kontrolle über

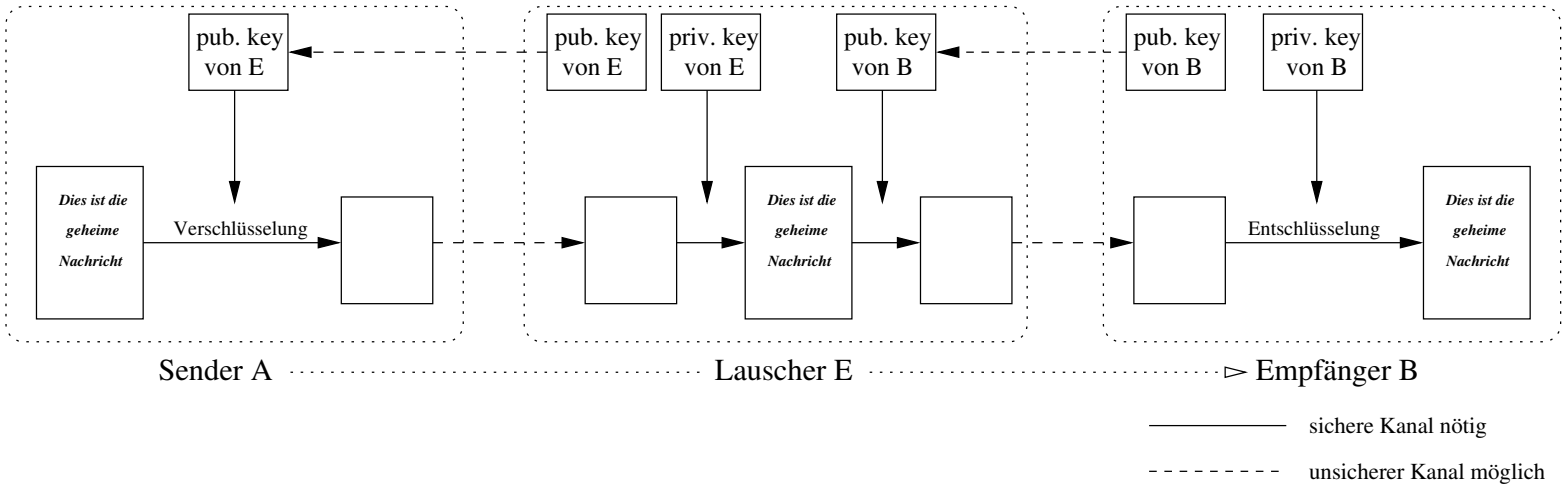

Abbildung 6: Man in the middle Angriff

den Kommunikationskanal. Denn der Angreifer muß ja verhindern, daß der „echte“öffentliche Schlüssel von B den Sender A erreicht. Außerdem darf auch die verschlüsselte Nachricht von A den Empfänger B nicht erreichen, da dieser dann Verdacht schöpfen könnte, wenn er die Nachricht nicht entschlüsseln kann. Daß nicht nur große Organisationen, wie Geheimdienste oder Kommunikationsdienstleister, solche Möglichkeiten haben, werden wir im nächsten Teil unserer Arbeit sehen, wenn wir uns näher mit den technischen Details befassen.

3 Das SSH-Protokoll

3.1 Historie

Die erste Inkarnation des SSH-Protokolls wurde 1995 von dem Finnen Tatu Ylönen entwickelt. Im Juli des gleichen Jahres stellte er eine Implementierung als Open Source im Internet zur Verfügung und dokumentierte SSH in einem IETF[7]-Draft. Ende des Jahres gab es bereits etwa 20 000 Benutzer, weshalb Ylönen eine Firma (SSH Communications Security[8]) gründete, um SSH kommerziell zu entwickeln, unterstützen und verkaufen. Gleichzeitig änderte er die Lizenzbedingungen für die nun erscheinenden neueren Versionen seiner SSH-Implementierung in restriktivere.
1996 stellte SSH Communications Security eine neue Version des SSH-Protokolls (SSH2) vor, und die IETF gründete hierauf eine öffentliche Arbeitsgruppe für die Standardisierung der „Secure Shell“, SECS. Im Februar 1997 war der erste IETF-Draft fertiggestellt, und 1998 gab SSH Communications Security eine Implementierung von SSH2 frei.

Basierend auf der letzten mit einer hinreichend liberalen Lizenz ausgestatteten Version von Ylönens Software begannen einzelne Entwickler[9], den Code als Open Source im Rahmen des OpenBSD-Projektes[10] weiter zu entwickeln, was im Dezember 1999 zur Veröffentlichung von OpenSSH 1.2.12 in OpenBSD 2.6[11] führte. Die IETF-Drafts für SSH2 wurden kontinuierlich aktualisiert und liegen zur Zeit der Erstellung dieser Arbeit in der fünfzehnten Revision vor.

Das Protokoll ist in der Version 2 in vier Dokumenten spezifiziert, von denen sich drei jeweils mit einer Schicht des Protokolls beschäftigen. Das vierte

[7] Internet Engineering Task Force
[8] http://www.ssh.com/
[9] http://www.openssh.org/history.html
[10] http://www.openbsd.org/
[11] http://www.openbsd.org/26.html

gibt einen allgemeinen Überblick über das Zusammenspiel der drei Schichten und legt Allgemeinheiten wie beispielsweise die Paketgröße, verwendete Datentypen und ihre Repräsentation, Nomenklatur-Schemata, oder Nummern für SSH-Pakete und ihre Semantik fest.

3.2 Aufbau

Im folgenden werden wir kurz die Struktur dieses Protokolls vorstellen, um dann im Abschnitt 3.3 auf die Schwächen und Angriffsmöglichkeiten einzugehen.

3.2.1 Allgemeine Architektur

Das SSH2-Protokoll besteht insgesamt aus drei aufeinander aufbauenden Schichten:

Die Transport-Schicht (Transport Layer) bildet die Basis einer jeden SSH-Verbindung. Sie ist im IETF-Draft „SSH Transport Layer Protocol“spezifiziert, welches neben Kompatibilitätsfragen mit alten SSH1-Clients und -Servern auch das verwendete Paketformat und die zulässigen Kompressions- und Verschlüsselungsalgorithmen dieser Schicht sowie Schlüsselaustauschmethoden festschreibt. Ebenso wird festgelegt, wie die Datenintegrität in dieser Schicht sichergestellt wird.

Die Benutzer-Authentifizierungs-Schicht (Userauth Layer) spezifiziert, wie die Authentifizierung eines Benutzers am Server abläuft, und legt mehrere Authentifizierungsmethoden fest. Sie befindet sich oberhalb der Transport-Schicht.

Der nach erfolgreicher Authentifizierung existierende verschlüsselte Tunnel zum Server wird von der Verbindungs-Schicht (Connection Layer) in mehrere logische Unterkanäle aufgesplittet, die einzelne Anfragen nach Diensten, die SSH bereitstellt, übertragen.

3.2.2 Transport-Schicht

Eine SSH-Verbindung kann über jedes Protokoll, das 8-bit-Daten unterstützt, abgewickelt werden; dieses sollte jedoch die Datenübertragung gegen Übertragungsfehler schützen, da diese den Abbruch der SSH-Verbindung zur Folge haben. Im Internet ist das kanonische Protokoll hierfür TCP, weshalb für SSH von der IANA[12] der Port 22 für SSH zugewiesen wurde. Beide Seiten - Client und Server - tauschen direkt nach erfolgreicher (TCP-)Verbindung Identifikations-Strings der Form SSH-(protocolversion)-(softwareversion) aus.

[12]Internet Assigned Numbers Authority, http://www.iana.org/

Die Kompatibilität zwischen alten und neuen Clients bzw. Servern wird vom Standard ebenfalls geregelt; da wir uns im vorliegenden Dokument aber lediglich mit SSH2 beschäftigen werden, ist diese Regelung nicht weiter wichtig. Ein möglicher Angriff auf SSH, auf den im Folgenden noch eingegangen werden wird, setzt aber genau an dieser Stelle des Protokolls an.

3.2.3 Benutzer-Authentifizierungs-Schicht

Das SSH–Authentifizierungs Protokoll ist ein All–Zweck Authentifizierungsprotokoll. Es setzt auf der SSH Transport–Schicht auf und geht von Integrität und Vertrauenswürdigkeit dieses zugrunde liegenden Protokolls aus.

Der Server beginnt die Authentifizierung, indem er dem Client eine Liste von unterstützten Authentifizierungsmethoden anbietet. Der Client kann sich nun eine beliebige Methode aus der gesendeten Liste heraussuchen und anwenden. Das heisst, der Server hat völlige Kontrolle darüber welche Möglichkeiten zur Authentifizierung er zur Verfügung stellen will. Gleichzeitig kann sich der Client relativ flexibel innerhalb der Liste die von ihm am besten unterstützte oder für den User günstigste Methode auswählen. Desweiteren werden hier ein Timeout des Servers und eine Limitierung der möglichen Versuche bei einer einzelnen Sitzung vorgeschlagen.

3.2.4 Verbindungs-Schicht

Die SSH–Verbindungs–Schicht stellt interaktive Login Sitzungen, Fernausführung von Befehlen und weitergeleitete X11 Verbindungen. Die SSH–Verbindungs–Schicht läuft auf der Transport–Schicht und auf der Benutzer-Authentifizierungs-Schicht. Dies ist nun die oberste Schicht des SSH Protokolls und verwaltet alle Prozesse, die über diese SSH Verbindung kommunizieren wollen. Dabei ist es besonders wichtig mehrere Prozesse, die über die gleiche Verbindung kommunizieren, zu einem Datenstrom zusammenzufassen und natürlich den ankommenden Datenstrom an die wartenden Prozesse zu verteilen.

3.3 Bekannte Schwächen

Das SSH Protokoll hat aber einige bekannte Schwächen. Neben dem sogenannten Man–in–the–Middle Angriff, den wir später noch genau untersuchen und durchführen werden, gibt es noch eine Reihe von Schwächen, die vor allem durch Implementierungsfehler entstanden sind.

3.3.1 Team Teso

Dieser Angriff, den Team Teso[13] Mitte 2002 veröffentlichte, demonstriert eine Schwäche in vielen SSH Implementationen. Diese Schwäche rührt von dem Versuch vieler Implementationen her, beide Protokollversionen von SSH, nämlich SSH1 und SSH2, zu unterstützen. Da die meisten SSH–Client–Implementationen beide Protokollversionen unterstützen, aber gleichzeitig eine der beiden Versionen bevorzugen, kann ein böswilliger Server als Antwort auf diese Protokollanfrage jeweils nur die nicht bevorzugte Protokollversion anbieten. Da der Client normalerweise aber nur die Server–Authentifikation der anderen Protokollversion kennt, kommt statt einer Angriffswarnung nur eine Standardmeldung zum Akzeptieren des angeblich unbekannten RSA Schlüssels.

3.3.2 Timing- und Padding-Angriffe

Es gibt zwei weitere Schwächen im SSH Protokoll:

- Zum ersten werden die zu sendenden Datenpackete nur bis zu einer Größe von acht Byte mit Zufallsdaten aufgestockt (Padding). Dies ermöglicht Rückschlüsse auf die ungefähre Größe der ursprünglichen Daten.

- Zum zweiten bietet das SSH Protokoll den sogenannten „interactive mode "als Übertragungsmodus an. Hierbei wird jeder einzelne Tastendruck sofort verschlüsselt und gesendet, nachdem der Benutzer diese Taste gedrückt hat. Dies lässt Rückschlüsse auf die Zeiträume zwischen zwei Tastendrücken des Benutzers zu.

Diese augenscheinlich eher kleinen Schwächen aber ein schwerwiegendes Sicherheitsrisiko nach sich. In ihrem Papier „Timing Analysis of Keystrokes and Timing Attacks on SSH "[14] beschreiben Dawn Xiaodong Song, David Wagner und Xuqing Tian, wie diese beiden Schwächen ausgenutzt werden können um geziehlt Packete mit sensitiven Daten, wie z.B. Passworte, aus dem Strom der gesendeten Daten herauszufiltern und durch Analyse der Verzögerungen zwischen den Tastendrücken sowohl die Person, wie auch eine sehr genau Analyse der betätigten Tasten. Daraus lassen sich letztendlich auch die Passworte in relativ kurzer Zeit entschlüsseln.

[13] http://www.team-teso.net/ [Mai 2003]
[14] http://crypto.stanford.edu/ dabo/abstracts/ssl-timing.html [Mai 2003]

3.3.3 Man-in-the-middle-Angriffe

Den Man-in-the-middle-Angriffen wird sowohl von Seiten der Sicherheitsexperten wie auch der Hacker ein immer größerer Stellenwert zugeordnet[15]. Vor allem im Internet (bzw. in Intranetzen) sind die Bedingungen für einen solchen Angriff sehr leicht zu erreichen. Die Netzwerkprotokolle bieten verschiedene Möglichkeiten, die es einem Angreifer ermöglichen sich tatsächlich zwischen die beiden Kommunikationspartner zu setzen und anderweitige Kommunikation der beiden Partner zu unterbinden. Eine genauere Betrachtung dieser Möglichkeiten, zu denen es auch bereits etliche Implementierungen gibt, setzt allerdings eine tiefgehende Kenntnis der zugrunde liegenden Netzwerkprotokolle vorraus, weshalb wir hier darauf verzichten wollen[16].

4 Das J2SSH-Framework

4.1 Historie

J2SSH ist ein derzeit sehr junges Projekt, welches Ende 2002 ins Leben gerufen wurde. Es handelt sich hierbei um eine objekt-orientierte Implementierung des SSH2-Protokolls in Java, welche mit einer mächtigen und gut erweiterbaren API ausgestattet ist. J2SSH umfaßt sowohl einen Server als auch einen Client. Unterstützt werden u.a. Port- und X11-Forwarding, Public-Key- und Paßwort-Authentifizierung, sowie SFTP.

Die erste Alpha-Version (0.0.3) wurde im Dezember 2002 freigegeben, die erste Beta-Release erfolgte mit der Version 0.1.0 beta Ende Januar 2003, welche Keyboard-Interactive-Authentifizierung, Unterstützung für verschiedene Schlüsselformate und Verbesserungen im SFTP-Subsystem enthielt. Die vorliegende Studienarbeit basiert auf J2SSH 0.1.0 beta.

4.2 Aufbau

Das J2SSH-Framework ist mit über 250 Java-Klassen sehr umfangreich, weshalb hier nicht auf alle Einzelheiten eingegangen werden kann. Es soll lediglich ein grober Überblick über den Aufbau und den Zusammenhang der einzelnen Klassen und Untersysteme gegeben werden.

[15] Bei einer Umfrage ist das man in the middle Angriff Programm *ettercap* das einen man in the middle Angriff auf das SSH Protokoll 1 durchführt unter die ersten zehn Plätze der beliebtesten Hackertools gewählt worden. http://www.heise.de/newsticker/data/dab-07.05.03-002/default.shtml [Juni 2003]

[16] Näheres dazu findet sich in [**?**]

Die Implementierung ist vom Stil und von der Nomenklatur her sehr nahe an die Protokoll-Spezifikation und die dort definierten Begriffe angelehnt. So ist das Transport-Layer weitestgehend in den Klassen TransportProtocolServer und TransportProtocolClient gekapselt, das Authentication-Layer in AuthenicationProtocolServer und AuthenticationProtocolClient. Zusammengefaßt sind die einzelen Teile in Packages unterhalb von com.sshtools.j2ssh, also beispielsweise com.sshtools.j2ssh.transport oder com.sshtools.j2ssh.authentication.

Neben der reinen Protokoll-Implementierung finden sich auch noch einige plattformspezifische Objekte bzw. Schnittstellen, welche z.B. ein generisches Interface für Klassen, die einen Benutzer an einem System authentifizieren können, definieren. Zudem existieren einzelne Hilfsobjekte, darunter beispielsweise ein IOStreamConnector, welcher einen Eingabe- und einen Ausgabe-Stream verbindet.

J2SSH verfügt über einen flexiblen Mechanismus zum Logging, welcher über Log4J, ein Teil des Apache-Jakarta-Projekts[17] realisiert wird. Mittels Log4j[18] ist sowohl die Granularität des Loggings als auch das Format (Datei, Konsole, Stream, uvm.) problemlos steuerbar.

Die Konfiguration des SSH-Servers erfolgt im für Java leicht zu parsenden XML-Format. Konfigurierbar sind neben den kanonischen Einstellungen (zu bindende Netzwerk-Adresse und Port, erlaubte und erforderliche Authentifizierungs-Verfahren, maximale Verbindungsanzahl, Host-Key, ...) auch Plattform-Spezifika (mittels welcher Klasse kann ein Benutzer am System authentifiziert werden, mittels welcher Klasse lassen sich Kommandos ausführen) und verschiedene Algorithmen, die vom System benutzt werden können. Es lassen sich so auch eigene Algorithmen in das Framework einfügen.

4.2.1 Server

Ein vollständiger SSH-Server ist in der Klasse SshServer gekapselt. Die prinzipielle Arbeitsweise einer Server-Implementierung ist durch das Protokoll weitestgehend determiniert:

Zunächst lauscht der Server auf einem konfigurierbarem TCP-Port, i.d.R. 22. Jeden dort ankommenden Verbindungsversuch nimmt er bis zu einer konfigurierbaren Höchstanzahl an und versucht, mit dem Kommunikationspartner die Transport-Schicht des Protokolls zu etablieren. In J2SSH geschieht

[17] http://jakarta.apache.org/, 30. Mai 2003
[18] http://jakarta.apache.org/log4j/docs/index.html, 30. Mai 2003

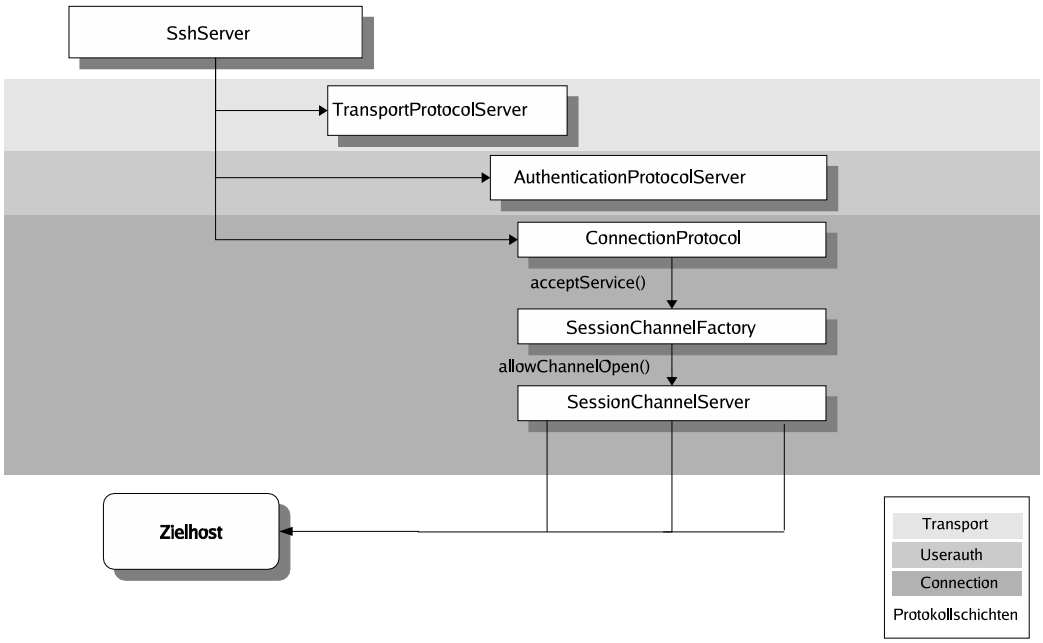

Abbildung 7: Architektur des SSH-Servers in J2SSH

dies in der Klasse TransportProtocolServer, welche vom SshServer[19] instanziiert wird.

Sobald mit einem Client die Transport-Schicht erfolgreich etabliert worden ist, muß sich dieser am System mit (mindestens) einer der vorhandenen Methoden authentifizieren. Diese Aufgabe übernimmt die Klasse AuthenticationProtocolServer, welche die für die Authentifizierung notwendigen Klassen (z.B. PasswordAuthenticationServer) instanziiert[20] und diesen die vom Client übergebenen Parameter (z.B. Username und Paßwort) übergibt. Jede Klasse, die einen Benutzer an einem System authentifiziert, implementiert das NativeAuthenticationProvider-Interface. Es werden in Version 0.1.0 beta keine solchen Klassen mitgeliefert.

Hat sich der Benutzer erfolgreich authentifiziert, so kann er nun über das Connection-Layer des Protokolls verschiedene Dienste anfordern, beispielsweise die Ausführung eines Kommandos, ein Pseudo-Terminal, oder eine Shell. Das Öffnen neuer Session-Kanäle (Session-Channels) geschieht in der SessionChannelFactory, die für jeden Session-Request eine neue Instanz eines SessionChannelServer-Objektes zurückgibt. Dieses behandelt die einzelnen Anfragen, z.B. nach einer Shell oder der Ausführung eines Kommandos, in seiner onChannelRequest()-Methode.

4.2.2 Client

Ein vollständiger SSH2-Client ist in der Klasse SshClient zu finden. Da sich eine Client-Implementierung immer nur um genau einen Kanal zum Server kümmern muß und der Teil des Transport-, Userauth- und Connection-Layers, der von ihm übernommen wird, deutlich kleiner ausfällt als der des Servers, ist die vorliegende Inkarnation weit weniger komplex geraten als das SshServer-Objekt. Prinzipiell genügt für eine Verbindung zu einem SSH2-Server die Instanziierung eines SshClient-Objekts und der Aufruf seiner connect()-Methode mit geeigneten Parametern (Zielhost, Port, Username, Paßwort - gekapselt in einem SshConnectionProperties-Objekt).

Die unteren beiden Ebenen des Protokolls (Transport und Userauth) sind im vorliegenden Client-Objekt nicht direkt zugänglich, wohl aber die Verbindungs-Schicht. Man erhält nach erfolgreicher Verbindung und Authentifizierung (d.h. nach erfolgreichem connect()-Aufruf) mittels getSession-

[19] Genauer betrachtet instanziiert SshServer einen ConnectionListener, der die ankommenden Verbindungsversuche annimmt und für jede erfolgreiche Verbindung eine ConnectedSession erzeugt, welche wiederum den TransportProtocolServer instanziiert.

[20] Die für die Authentifizierung notwendigen Klassen sind immer Unterklassen der abstrakten Klasse SshAuthenticationServer und werden mittels der statischen Methode SshAuthenticationServerFactory.newInstance() erzeugt.

ChannelClient() ein SessionChannelClient-Objekt, welches das Gegenstück zur serverseitigen SessionChannelServer-Klasse bildet und über seine Methoden die Anforderung einer Shell, eines Pseudo-Terminals etc. erlaubt. Beispielsweise erhielte man eine Shell über den Aufruf von startShell() oder ein Pseudo-Terminal mittels requestPseudoTerminal().
Einem SessionChannelClient zugeordnet sind jeweils ein Eingabe- und ein Ausgabe-Strom – in Java gekapselt in einem InputStream und einem OutputStream – welche sich durch die Methoden getInputStream() und getOutputStream() abrufen lassen.

5 jmitm2 - das Angriffsprogramm

5.1 Aufbau

5.1.1 Überblick

Bei der Implementierung des Angriffsprogrammes wurds versucht, so viel wie möglich aus den vorhandenen Objekten zu erben, um die Menge des eigenen Codes gering zu halten und von eventuellen Weiterentwicklungen von J2SSH zu profitieren. Daher war lediglich die Erstellung von zehn Klassen notwendig, von denen fünf Unterklassen von J2SSH-Server-Klassen sind. Diese sind zudem relativ klein gehalten.

Um aus einer vorhandenen objektorientierten Server- und Client-Implementierung (wie sie mit J2SSH vorlag) ein Man-in-the-middle-Angriffsprogramm zu erstellen, benötigt man – bildlich formuliert – ein Objekt, welches die beiden Teile so „aneinanderklebt“, daß sie einem Opfer des Angriffs den von ihm anvisierten Zielhost vorgaukeln und sich mit diesem verbinden, um transparent seine Tastatureingaben an den Zielhost und dessen Ausgaben an das Opfer weiterzuleiten. Diese Arbeit wird vom MitmGlue-Objekt erledigt, das das Herzstück des Angriffsprogrammes darstellt.

Aus dieser gewählten Architektur ergibt sich zwingend, daß jeder Verbindung genau ein MitmGlue-Objekt zugeordnet wird, und daß dieses auch in Kenntnis von Benutzername und Paßwort (sobald eingegeben) sein muß, um diese an den Zielhost weiterzureichen.

5.1.2 Server

Da das MitmGlue-Objekt auf allen drei Ebenen (Transport, Userauth, Connection) des Protokolls Server und Client verbinden muß, und diese Ebenen in der gewünschten Weise im SshServer-Objekt im Gegensatz zum SshClient-Objekt nicht zugänglich sind, wurde der Server mittels Unterklassen dahin-

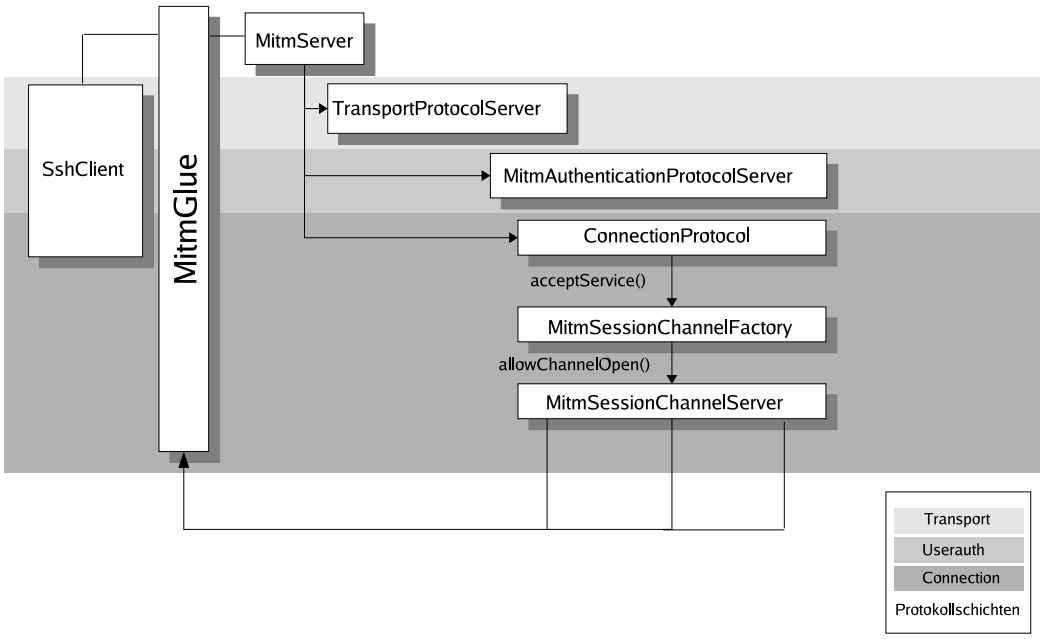

Abbildung 8: Architektur des Angriffsprogrammes, Überblick

gehend erweitert, daß er auf allen Ebenen Kenntnis von einem MitmGlue-Objekt hat. Die Konstruktoren der relevanten Klassen wurden also um einen Parameter vom Typ MitmGlue erweitert. Dieses Objekt wird bei seinem Weg durch diese erweiterten Server-Klassen mit Informationen gefüttert, sobald diese vorhanden sind (beispielsweise Benutzername und Paßwort), und für jede neue Verbindung eines Opfers mittels clone() geklont.

5.1.3 MitmGlue

Das MitmGlue-Objekt ist das Herzstück des Angriffsprogrammes und wird von der Kommandozeile aus mit den Parametern Ziel-Hostname (oder IP-Addresse) und Port gestartet. Zunächst erzeugt es eine neue Instanz eines MitmServers, also der um die Kenntnis des MitmGlue-Objektes erweiterten SshServer-Klasse. Diese lauscht dann auf der in der Server-Konfiguration angegebenen Addresse und dem dort eingetragenen Port - ganz analog zum SshServer. MitmGlue enthält ebenfalls eine Instanz eines SshClient-Objekts, mittels dessen die Verbindung zum Zielhost etabliert werden kann.

Sobald eine neue TCP-Verbindung[21] von einem Opfer aufgebaut wurde, beginnt der MitmServer, die Transport-Schicht des Protokolls zu etablieren. Dieser Vorgang geschieht völlig analog zu dem, den der SshServer durchgeführt hätte, mit dem einzigen Unterschied, daß das MitmGlue-Objekt jeweils an die währenddessen erzeugten relevanten Instanzen anderer Objekte übergeben wird. Für jede angenommene Verbindung wird das MitmGlue-Objekt geklont, denn die Verbindungen sind unabhängig voneinander und benötigen deshalb jeweils eine eigene Instanz. Gleichzeitig mit der Verbindung des Opfers zum Angriffsprogramm wird auch die Verbindung vom Angriffsprogramm zum Zielhost aufgebaut, um den Angriff möglichst weitgehend zu verschleiern.

Als nächstes wird versucht, das Userauth-Layer zu etablieren – hierzu erzeugt der Server mittels einer MitmAuthenticationServerFactory neue Instanzen von MitmAuthenticationProtocolServer und MitmAuthentication-Server. Die vom Benutzer, d.h. in diesem Fall dem Opfer des Angriffs übergebenen Informationen (Benutzername und Paßwort) werden an eine neue Instanz des in der Konfiguration festgelegten NativeAuthenticationProvider zur Überprüfung übergeben. Im Falle eines Mitm-Angriffes muß diese Klasse also sich mit dem Zielhost (den das Opfer eigentlich erreichen wollte) verbinden, sich dort mit den übergebenen Informationen anzumelden versuchen, und das Ergebnis dieses Versuchs zurückliefern. Hierzu erhält es ebenfalls

[21] jmitm2 arbeitet nur auf OSI-Protokollschicht vier und darüber, d.h. das in der Praxis für die Durchführung notwendige ARP- und DNS-Spoofing muß von einem Hilfsprogramm geleistet werden. Hier haben sich dnsspoof und arpspoof aus dem dsniff-Paket bewährt.

Kenntnis vom MitmGlue-Objekt, übergibt diesem Benutzername und Paßwort, und versucht dann, die Userauth-Schicht mittels des Aufrufs der Methode doAuthentication() zu etablieren. MitmGlue benutzt die vorhandene und ja bereits verbundene SshClient-Instanz, um sich beim Zielhost anzumelden.

Nach erfolgreicher Authentifizierung am Zielhost steht nun ein verschlüsselter Tunnel vom Opfer zum Angriffsprogramm und von dort zum Zielhost zur Verfügung. Sobald das Opfer einen der Dienste des Ssh-Protokolls anfordert, wird diese Anfrage über MitmGlue und dessen SshClient-Instanz an den Zielhost weitergeleitet. Ist ein Dienst erfolgreich angefordert worden (Pseudo-Terminal, Shell o.Ä.), dann werden die Ein- und Ausgabeströme von Server und Client (d.h. von Opfer und Zielhost) mitttels der vorhandenen Hilfsklasse IOStreamConnector verbunden und so Ein- und Ausgabe transparent weitergeleitet.

5.1.4 Erweiterungsmöglichkeiten

Das Angriffsprogramm ließe sich mit geringem Aufwand noch deutlich erweitern. Beispielsweise wäre es mit dem benutzten Logging-System ein Leichtes, ganze SSH-Sitzungen eines Benutzers zu protokollieren. Ebenfalls möglich wäre es, Sitzungen zu übernehmen, d.h. nach erfolgreicher Authentifizierung am Zielhost die Verbindung zum Opfer trennen und selbst mit der Session weiterzuarbeiten. Auch das Einfügen von Zeichen (z.B. Kommandos oder gefälschte Server-Ausgaben) in den verschlüsselten Tunnel vom Opfer zum Zielhost ist möglich.

6 Fazit

Obwohl jmitm2 nicht das erste Programm ist, das einen Man-in-the-middle-Angriff auf das SSH-Protokoll duchführt, so ergeben sich aus seinem Vorhandensein dennoch Konsequenzen. Nach der Veröffentlichung des umfangreichen dsniff-Pakets[22], das unter anderem auch einen Mitm-Angriff auf SSH enthielt, erschienen zahlreiche Artikel in der Fachpresse, die die Leichtigkeit, mit der dieser und andere Angriffe durchgeführt werden konnten aufzeigten und zu größerer Sorgfalt bei Wahl von Paßwörtern und genereller Vorsicht mahnten. Die erste Veröffentlichung[**?**], die sich mit dsniff und seinen Folgen im Detail beschäftigte, stellte in ihrem Titel sogar die provokante Frage: „The End of SSL and SSH?“.

Obwohl diese mit Sicherheit verneint werden konnte und kann, so machte dennoch das Vorhandensein der Möglichkeit, diese als sicher geltenden

[22] http://www.monkey.org/ dugsong/dsniff/, 30. Mai 2003

Protokolle beinahe auf Tastendruck und nahezu oder völlig unbemerkt anzugreifen klar, daß die Benutzer in Zukunft vorsichtiger vorgehen müssten und tatsächlich die Host-Keys - also die öffentlichen Schlüssel der Rechner, auf denen sie sich einloggen möchten - auf ihre Echtheit hin überprüfen. Diese Notwendigkeit wurde und wird allerdings davon eingeschränkt, daß dsniff und auch andere Angriffsprogramme wie ettercap[23] lediglich SSH1 beherrschen. Da dieses Vorgehen in der Realität oft sehr aufwendig oder gar praktisch nicht durchführbar ist (z.B. bei Rechnern, die sich in geographisch großer Entfernung befinden), sind MITM-Angriffe nach wie vor eine ernstzunehmende Bedrohung.

jmitm2 ist derzeit das erste verfügbare Programm, das einen Mitm-Angriff auf das SSH2-Protokoll beherrscht[24]. Obwohl sicher niemand mehr das Ende von SSH prophezeien wird, so ist jetzt dennoch wieder einmal um so deutlicher geworden, wie wichtig es ist, die Echtheit der Host-Keys selbst zu überprüfen und den Meldungen des SSH-Programmes Aufmerksamkeit zu schenken.

[23] http://ettercap.sourceforge.net/, 30. Mai 2003

[24] ssharp von team teso wäre allerdings mit geringen Modifikationen auch dazu in der Lage

Abbildungsverzeichnis

Literatur

[Bau97] Bauer, Friedrich L.: *Entzifferte Geheimnisse. Methoden und Maximen der Kryptologie.* 2. erw. Auflage. Springer, 1997

[BSW98] Beutelsbacher, Albrecht ; Schwenk, Jörg ; Wolfenstetter, Klaus-Dieter: *Moderne Verfahren der Kryptographie.* 2. Auflage. Vieweg, 1998

[DH76] Diffie, Whitfield ; Hellman, Martin E.: New Directions in Cryptography. In: *IEEE Transactions on Information Theory* IT-22 (1976), Nr. 6, S. 644–654

[Kob94] Koblitz, Neal: *A Course in Number Theory and Cryptography.* Springer, 1994

[Kob99] Koblitz, Neal: *Algebraic Aspects of Cryptography.* Springer, 1999

[MOV01] Menezes, Alfred J. ; van Oorschot, Paul C. ; Vanstone, Scott A.: *Handbook of Applied Cryptography.* CRC Press, August 2001. – http://www.cacr.math.uwaterloo.ca/hac/ [Juni 2003]

[RSA77] Rivest, R. L. ; Shamir, A. ; Adelman, L. M.: A Method for Obtaining Digital Signatures and Public–Key Cryptosystems. 1977 (MIT/LCS/TM-82). – Forschungsbericht. – 15 S. theory.lcs.mit.edu/ cis/pubs/rivest/rsapaper.ps

[Sei01] Seifried, Kurt: *The End of SSL and SSH?* 2001. – http://www.seifried.org/security/cryptography/20011108-end-of-ssl-ssh.html

[Sha49] Shannon, C. E.: Communication Theory of Secrecy Systems. In: *Bell System Technical Journal* 28 (1949), Nr. 4, S. 656–715

[Wag01] Wagner, Robert: *Address Resolution Protocol Spoofing and Man-in-the-Middle Attacks.* August 2001. – http://www.sans.org/rr/paper.php?id=474 [Juni 2003]

[Wer02] Werner, Anette: *Elliptische Kurven in der Kryptographie.* Springer, 2002

[Wol96] Wolfart, Jürgen: *Einführung in die Zahlentheorie und Algebra.* Vieweg, 1996