Authentisierung in verteilten Systemen

Markus Kuhn – Sarntal – September 1994

In großen verteilten Systemen wie etwa einem akademischen Campus-Netz haben sehr viele Benutzer vollständigen Zugang zur am System teilhabenden Hardware. Typisch für diese Systeme ist die große räumliche Ausdehnung, die Anonymität der unüberschaubaren Benutzergruppe und der Umstand, daß nur Teile der eingesetzten Hard- und Software der Kontrolle von vertrauenswürdigen Systemadministratoren unterliegen. Dadurch ergeben sich Schutzprobleme, die ohne kryptographische Techniken nicht gelöst werden können. Beispielsweise verlaufen in vielen Universitäten Datenleitungen z.T. durch die Wohnräume neugieriger und technisch versierter Studenten [3] und streng vertrauliche Daten werden in Firmen- und Behördennetzen über mit geringem Aufwand abhörbare Telefonleitungen, Richtfunk- und Satellitenverbindungen ausgetauscht. Somit ergeben sich neue Anforderungen an die Sicherheitsmechanismen in Netzwerk- und Betriebssystemsoftware, die sich grundlegend von den Anforderungen klassischer zentraler Mainframe-Zugriffskontrollsysteme unterscheiden.

1 Kryptologische Grundlagen

Ein *symmetrisches Kryptosystem* besteht aus zwei Funktionen E und D, die eine unverschlüsselte Nachricht (*plain text*) p mit einem geheimen Schlüssel K in eine verschlüsselte Nachricht (*cipher text*) c verwandeln und wieder entschlüsseln:

$$E(K,p) = c$$
$$D(K,c) = p$$

Ein geheimer Schlüssel (*secret key*) ist dabei eine möglichst kurze Bitfolge (in der Regel etwa 128 Bit lang), die den miteinander kommunizierenden Partnern, aber sonst niemandem, bekannt ist. Ein gutes symmetrisches Kryptosystem zeichnet sich dadurch aus,

a) daß es für einen Angreifer extrem schwer ist, K zu ermitteln, wenn ihm sehr viele damit verschlüsselte (p,c) Paare bekannt sind (*known plain text attack*) oder der Angreifer gar die Möglichkeit hat, die mit K zu verschlüsselnden Texte p gezielt auszusuchen (*chosen plain text attack*).

b) daß es für einen Angreifer extrem schwer ist, p zu ermitteln, wenn zwar c, nicht aber K bekannt sind (*secrecy*).

c) daß es für einen Angreifer extrem schwer ist, c zu ermitteln, wenn p vorgegeben aber K nicht bekannt ist (*integrity*).

d) daß es sehr effizient ist und sich leicht als Hardware realisieren läßt.

Die Forderung a) stellt sicher, daß das Kryptosystem nicht durch systematisches Abhören angegriffen werden kann und b) garantiert, daß verschlüsselte Nachrichten geheim bleiben. Punkt c) garantiert, daß das Kryptosystem auch für digitale Unterschriften eingesetzt werden kann, da nur jemand der K kennt die Nachricht c erzeugt haben kann. Da in verteilten Systemen evtl. sehr große Datenmengen (z.B. sämtliche Zugriffe eines Programmes auf einen Massenspeicher) verschlüsselt werden müssen, ist für den Alltagseinsatz Forderung d) von erheblicher Bedeutung. „Extrem schwer“ bedeuted, daß es nach Möglichkeit keinen Algorithmus geben sollte, der eines der Probleme schneller löst, als dies durch systematisches Durchsuchen aller Werte für K möglich wäre (*brute force attack*).

Neben den älteren symmetrischen Verfahren kennt man heute auch eine Reihe von *asymmetrischen Kryptosystemen* (*public key cryptosystems*). Diese benutzen zwei verschiedene Schlüssel, einen *public key* K und einen *secret key* K^{-1}. Diese beiden Schlüssel werden als Paar zusammen erzeugt. Der Schlüssel K^{-1} kann benutzt werden, um Texte zu entschlüsseln, die mit K verschlüsselt wurden:

$$D(K^{-1}, E(K,p)) = p.$$

Bei guten public key Kryptosystemen ist es für einen Angreifer extrem schwer aus K den dazugehörigen K^{-1} zu ermitteln und umgekehrt. Wenn man nur den public key kennt, dann kann man eine Nachricht die man gerade damit verschlüsselt hat auch selbst nicht mehr entschlüsseln. Ansonsten gelten sinngemäß die Anforderungen für symmetrische Verfahren.

Der besondere Nutzen von asymmetrischen Kryptosystemen liegt darin, daß einer der beiden Schlüssel veröffentlicht werden kann, ohne daß die Sicherheit der Kommunikation gefährdet wird. Zum Beispiel kann jeder Teilnehmer in einem Kommunikationssystem seinen public key in einer Art Telefonbuch veröffentlichen. Wenn Benutzer A („Adam“) an Benutzer B („Bob“) eine Nachricht verschlüsselt schicken will, dann sucht A aus dem Telefonbuch den öffentlichen Schlüssel K_B von B heraus und schickt ihm den Text p damit verschlüsselt:

$$A \to B: \quad E(K_B, p)$$

So ist sichergestellt, daß nur B als rechtmäßiger Empfänger den Text $p = D(K_B^{-1}, E(K_B,p))$ lesen kann.

Bei den gängigen public key Kryptosystemen besteht kein prinzipieller Unterschied zwischen dem secret key und dem public key (außer eben, daß einer der beiden nur dem Besitzer bekannt ist). Daher kann auch der secret key zum Verschlüsseln und der dazu passende public key zum Entschlüsseln benutzt werden. Damit ist eine digitale Unterschrift möglich:

$$A \to B: \quad E(K_A^{-1}, p)$$

Wenn A an B einen Text sendet, den er mit seinem eigenen secret key K_A^{-1} verschlüsselt hat, dann kann jeder mit dem allgemein bekannten dazu passenden public key K_A die Nachricht entschlüsseln und erhält dadurch die Gewißheit, daß nur A diese Nachricht verschlüsselt haben kann, da nur A den entsprechenden K_A^{-1} kennt. Entsprechend lassen sich Verschlüsselung und Unterschrift auch kombinieren

$$A \to B: \quad E(K_B, E(K_A^{-1}, p))$$

und nun kann A sicher sein, daß nur B die Nachricht lesen kann, und B kann sicher sein, daß die Nachricht p von A stammt. Sollten in einem asymmetrischen Kryptosystem nicht beide Schlüssel sowohl zum Ver- als auch Entschlüsseln geeignet sein, dann kann dies dadurch ausgeglichen werden, daß jeder Teilnehmer T zwei Schlüsselpaare $\hat{K}_T$, $\hat{K}_T^{-1}$, $\tilde{K}_T$ und $\tilde{K}_T^{-1}$ erzeugt und davon $(\hat{K}_T, \tilde{K}_T^{-1})$ veröffentlicht. So steht jeweils ein öffentlicher key zum Ver- und Entschlüsseln bereit.

Das bekannteste asymmetrische Kryptosystem ist RSA [6] und seine Sicherheit beruht auf der Annahme, daß es extrem aufwendig ist, das Produkt zweier sehr großer Primzahlen in diese Faktoren zu zerlegen. Symmetrische Kryptosysteme sind beispielsweise DES (56-bit key) und IDEA (128-bit key), die durch häufig iterierte Bitpermutationen, Additionen und Tabellenersetzungen dafür sorgen, daß alle Ergebnisbits scheinbar völlig zufällig von den Eingangsbits abhängen (z.B. muß ein einziges geändertes Eingangsbits jedes Ausgangsbit etwa mit einer Wahrscheinlichkeit 0.5 ändern). Leider sind die bislang bekannten asymmetrischen Kryptosysteme wesentlich CPU intensiver als symmetrische Systeme. Eine gute Softwareimplementation des verbreiteten symmetrischen DES Verfahrens ist etwa doppelt so schnell wie eine RSA Hardware und eine moderne DES Hardware ist gar um über 5000× schneller [2]. Damit sind Verfahren wie RSA derzeit nicht geeignet, direkt die kompletten Datenströme von leistungsfähigen Netzwerkverbindungen zu verschlüsseln. In der Regel werden daher asymmetrische Kryptosysteme eingesetzt, um einen nur für eine begrenzte Zeit gültigen, nur den beiden Kommunikationspartnern bekannten *session key* für ein effizientes symmetrisches Verschlüsselungsverfahren auszutauschen. Auf diese Weise lassen sich die Vorteile der public key Verfahren (öffentliche „Telefonbücher“) und der symmetrischen Verfahren (Softwareimplementationen bis 400 kBit/s, Hardwareimplementationen bis 1.2 GBit/s [2]) zugleich nutzen.

Um weniger effiziente asymmetrische Kryptosysteme für digitale Unterschriften über große Datenmengen einsetzen zu können, unterschreibt man nur ein *message digest* des Textes. Dabei handelt es sich um das Ergebnis einer one-way Funktion, die ähnlich einer Prüfsumme oder einer Hashfunktion den gesamten Text berücksichtigt um das relativ kurze (typisch sind 128 Bits) message digest zu erzeugen. Dabei wird eine Funktion eingesetzt, die es extrem schwierig macht, zu einem gegebenen message digest einen passenden Text zu finden. Ein verbreiteter und effizienter message digest Algorithmus ist z.B. MD5 [7].

Als Anschauungsbeispiel dafür, wie ein asymmetrisches Kryptosystem mathematisch funktioniert, sei hier das ElGamal-System erläutert. Seine Sicherheit beruht darauf, daß das Problem des *diskreten Logarithmus* schwierig ist, d.h. die Aufgabe, zu bekannten natürlichen Zahlen s, p (p prim) und $s^a \bmod p$ den Exponenten a effizient zu ermitteln [5]. Sei nun p eine allgemein bekannte große Primzahl und $1 < s < p-1$ eine allgemein bekannte natürliche Zahl. Gerechnet wird im Körper der natürlichen Zahlen modulo p. A denkt sich eine natürliche Zahl a als secret key aus und veröffentlicht s^a als seinen public key. Entsprechend erzeugt B seinen secret key b und veröffentlicht s^b. Will nun A eine Nachricht m an B verschlüsseln, so erzeugt er $k = (s^b)^a = s^{ab}$ als session key und verschlüsselt die Nachricht mit irgend einem symmetrischen Verfahren und k als Schlüssel. Zum Entschlüsseln berechnet B die Zahl $(s^a)^b = s^{ab} = k$ und erlangt so nur anhand des eigenen secret keys b und A's public key s^a die zum Entschlüsseln notwendige Zahl k. Wenn nicht wie hier gezeigt mit einem B bekannten public key s^a von A auch gleich eine Unterschrift mitgeliefert werden soll, dann erzeugt A eben zufällig für jede Nachricht ein neues a und liefert s^a mit der Nachricht mit: $A \to B: \quad s^a, E(k, m)$.

Bislang wurde allerdings noch nicht bewiesen, daß das ElGamal-Verfahren genau so schwierig ist, wie der diskrete Logarithmus. Offensichtlich ist nur, daß bei Bekanntwerden eines effizienten Algorithmus für den diskreten Logarithmus aus den public keys leicht die secret keys ermittelt werden könnten. Das ElGamal-System läßt sich mit beliebigen endlichen Körpern GF(p^n), nicht nur mit natürlichen Zahlen modulo p einsetzen.

2 Das Key-Management Problem

Wenn A und B, die noch nie zuvor Kontakt hatten, über eine unsichere Datenleitung kommunizieren wollen, dann könnten sie auf den ersten Blick z.B. wie folgt vorgehen: Zuerst teilen sie sich gegenseitig ihre public keys K_A und K_B mit, da kein „Telefonbuch“ mit diesen vorhanden ist. Nun können scheinbar bereits geheime Nachrichten abhörsicher (da verschlüsselt) und fälschungssicher (da unterschrieben) ausgetauscht werden:

(a) $A \to B: \quad K_A$

(b) $B \to A: \quad K_B$

(c) $A \to B: \quad E(K_B, E(K_A^{-1}, p_1))$

(d) $B \to A: \quad E(K_A, E(K_B^{-1}, p_2))$

Diese Vorgehensweise ist tatsächlich bereits sicher, solange es einem Angreifer E („Eve“) allerhöchstens möglich ist, die ausgetauschten Daten mitzulesen. Wenn aber E in der Lage ist, Daten abzufangen und durch andere Daten zu ersetzen, ohne daß A und B dies bemerken, dann kann E die gesamte Kommunikation sowohl im Klartext verfolgen, als auch verfälschen. Dazu generiert E zwei public/secret key Paare $(K_{E_A}, K_{E_A}^{-1})$ und $(K_{E_B}, K_{E_B}^{-1})$ und ersetzt während des anfänglichen public key Austausches K_A und K_B durch die eigenen public keys. Anschließend wird E alle Nachrichten zwischen A und B abfangen, entschlüsseln, lesen und/oder verfälschen, erneut verschlüsseln und weiterleiten:

(a) $A \to E: \quad K_A$

(b) $E \to B: \quad K_{E_A}$

(c) $B \to E: \quad K_B$

(d) $E \to A: \quad K_{E_B}$

(e) $A \to E: \quad E(K_{E_B}, E(K_A^{-1}, p_1)) = c_1$

(f) $E \to B: \quad E(K_B, E(K_{E_A}^{-1}, D(K_A, D(K_{E_B}^{-1}, c_1))))$

(g) $B \to E: \quad E(K_{E_A}, E(K_B^{-1}, p_2)) = c_2$

(h) $E \to A: \quad E(K_A, E(K_{E_B}^{-1}, D(K_B, D(K_{E_A}^{-1}, c_2))))$

Wegen

$$D(K_A, D(K_{E_B}^{-1}, c_1)) = p_1 \qquad \text{und}$$
$$D(K_B, D(K_{E_A}^{-1}, c_2)) = p_2$$

hat E also bei obigem „Sicherheitsprotokoll“ durch das unbemerkte Austauschen der public keys K_A und K_B gegen die Fälschungen K_{E_A} und K_{E_B} die volle Kontrolle über die Kommunikation erhalten. Diese Möglichkeit zu unterbinden, ist das Ziel sorgfältig entwickelter Sicherheitsprotokolle. Die Grundidee dabei ist im wesentlichen, daß veröffentliche public keys zwar jedem bekannt sein dürfen, aber dennoch gegen Fälschung gesichert sein müssen. Dies ist möglich durch public keys, die durch vertrauenswürdige Dritte unterschrieben worden sind. Der public key dieses Dritten muß aber allgemein und längerfristig bekannt sein.

3 Hierarchische Zertifikate

Das Konzept der von vertrauenswürdigen Dritten unterschriebenen keys kann zu hierarchischen Authentifizierungssystemen erweitert werden. Beispielsweise könnte die Bundesregierung ihren public key so veröffentlichen, daß dieser allgemein bekannt ist. Nun kann z.B. jede Universität ihren public key von der Bundesregierung unterschreiben lassen. Dieses unterschriebene Zertifikat wird dabei eine Nachricht sein, die den public key, den Namen des Inhabers des keys (z.B. Uni-Erlangen, Rechenzentrum) und den Namensbereich eines Netzes enthält, über den der Inhaber das Recht hat, weitere Zertifikate zu verteilen (z.B. im Internet uni-erlangen.de). Entsprechend stellt dann in diesem Beispiel die Universität einem Systemadministrator ein Zertifikat für den Namensbereich cip.informatik.uni-erlangen.de aus, und letzterer stellt ein Zertifikat für den Benutzer, der z.B. über seine e-mail Adresse identifiziert wird aus. Das Zertifikat eines Benutzers enthält also seinen Namen, seinen public key, die Unterschrift des Administrators, daß dieser Name und key echt sind, die Unterschrift der Universität, daß der Name und key des Administrators echt sind und die Unterschrift der Bundesregierung, daß Name und key der Universität echt sind. So kann mit einem einzigen wohlbekannten zentralen public key der Regierung leicht jeder zertifizierte andere public key überprüft werden. Sollte jemand falsche Zertifikate ausstellen, so ist der Schuldige anhand der in den Zertifikaten vorhandenen unterschriebenen Namen leicht festzustellen.

Die Zertifizierungskette läßt sich zusätzlich auch von unten nach oben aufbauen, so daß z.B. der Systemadministrator, den der Benutzer persönlich kennt, den public key der Universität zertifiziert und die Universität unterschreibt wiederum den public key der Regierung oder auch die public keys anderer Universitäten. Damit wären dann selbst wenn jemand den secret key der höchsten zentralen Instanz (z.B. Regierung) stehlen könnte immer noch sichere Kommunikation in einer Untergruppe (z.B. zwischen allen Uni-Netzen) möglich. Außerdem vermeidet die zusätzlich von unten nach oben durchgeführte Zertifizierung, daß ein ununterschriebener zentraler key (z.B. der der Regierung) allgemein ohne Fälschungsmöglichkeit bekannt sein muß, was in Computernetzwerken nur mit manuellem Aufwand möglich wäre. In diesem Beispiel muß dem Benutzer zur Authentifizierung nur der echte public key des Systemadministrators bekannt sein, was z.B. durch persönliches Aushändigen oder durch Benutzen des entsprechend installierten Systems leicht sicherzustellen ist.

Während durch den hierarchischen Aufbau einer Zertifizierungskette leicht zu garantieren ist, daß zwischen allen Kommunikationspartnern eine Zertifizierungskette existiert, so daß jeder sicher sein kann, nur den

Originalschlüssel des anderen zu benutzen, so ist diese Baumstruktur nicht unbedingt zwingend. Querzertifikate oder mehrere höchste Instanzen erlauben Zertifizierungssysteme aufzubauen, die auch das Bekanntwerden einzelner secret keys oder unvertrauenswürdige einzelne Zertifizierungsstellen überstehen.

Systeme wie PEM (Internet Privacy Enhanced Mail) oder ITU-T X.509 zur Verwaltung solcher Zertifikate sind bereits im Einsatz. Völlig ohne vorgegebene zentrale Struktur bauen beim PGP (Pretty Good Privacy) System [8] die Benutzer selbst ein *web-of-trust* auf, in der Hoffnung, daß an Kommunikation interessierte Benutzergruppen sich selbst manuell die notwendigen Zertifikate ausstellen. Der PGP Ansatz ist für den Endbenutzer aufwendiger und schwieriger zu verstehen, erfordert dafür aber in kleinen Gruppen kaum Planungs- und Koordinationsaufwand, da kein Namensraum festgelegt werden muß.

4 Das Kerberos-Protokoll

Im Rahmen des Projektes Athena am MIT, das eine Systemumgebung für ein großes verteiltes Workstation-Netz entwickelt, wurde auch das Authentifizierungssystem *Kerberos* entwickelt [4]. Es führt eine sichere Verteilung von Schlüsseln zur client/server Kommunikationen durch und erlaubt damit sowohl die wechselseitige Authentifizierung von client und server als auch eine verfälschungs- und abhörsichere Kommunikation. Dabei wird ausschließlich ein symmetrisches Kryptosystem eingesetzt (in der Regel DES).

In einem Kerberos-System existiert eine einzige zentrale Datenbank, der Kerberos-Server, der alle Benutzer und ihre Passwörter kennt und weiß, auf welche weiteren Server sie Zugriffsrechte haben. Benutzer können eine beliebe Workstation benutzen. Durch einmalige Eingabe eines Passwortes während einer Sitzung können sie sich mittels des Kerberos-Servers bei jedem anderen Server authentifizieren und gesichert Daten mit diesem austauschen (z.B. e-mail lesen, Dateien drucken, Datenbank abfragen, etc.). Dabei ist es im Gegensatz zur klassischen Passwort-Authentifizierung einem Angreifer, der volle Kontrolle über die Netzwerkverbindungen hat, nicht möglich, sich unter falschem Namen gegenüber einem Server zu authentifizieren um z.B. die e-mail anderer Leute zu lesen.

Im folgenden wird statt $E(K,p)$ die etwas kürzere Notation $\{p\}_K$ benutzt, um eine mit dem Schlüssel K verschlüsselte Nachricht p zu beschreiben. Aus dem vom Benutzer A zu Beginn der Sitzung eingegebenem Passwort wird mittels einer one-way Funktion der Schlüssel K_{as} erzeugt, den nur der Benutzerprozeß und der Kerberos-Server S kennen. Mit diesem Schlüssel ist also eine sichere Kommunikation zwischen A und S möglich. Die folgenden vier Schritte erlauben es sowohl A als auch B im Vertrauen auf S sich gegenseitig von ihrer Identität zu überzeugen:

(a) $A \to S:\quad A, B$

(b) $S \to A:\quad \{T_s, L, K_{ab}, B, \{T_s, L, K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$

(c) $A \to B:\quad \{T_s, L, K_{ab}, A\}_{K_{bs}}, \{A, T_a\}_{K_{ab}}$

(d) $B \to A:\quad \{T_a + 1\}_{K_{ab}}$

Zunächst teilt A an S mit, daß A mit dem Server B kommunizieren will (a). Der Server S kennt in seiner Datenbank auch B und einen Schlüssel K_{bs} der sonst nur noch B bekannt ist. S schickt daraufhin an A eine nur für A lesbare Nachricht zurück (b). Diese enthält die aktuelle Uhrzeit des Servers T_s, eine Lebensdauer L des übermittelten Tickets, einen Schlüssel K_{ab} der für eine sichere Kommunikation zwischen A und B eingesetzt werden kann, und ein Ticket für B. Dieses Ticket ist eine Nachricht von S, die da mit K_{bs} verschlüsselt nur für B lesbar ist, und die ebenfalls die Erzeugungszeit des Tickets und seine Lebensdauer, den session key K_{ab} und den Namen von A enthält. Dieses Ticket schickt A nun an B (c) und damit kennen beide Stationen den session key K_{ab} und können mit der gesicherten Kommunikation beginnen.

Da Passwörter durch unglückliche Umstände bekannt werden können, stellt der Kerberos-Server Tickets für die Kommunikation mit einem Server nur mit einer begrenzten Lebensdauer (wenige Minuten) aus. Da B anhand von T_s, L und seiner eigenen Uhr T_b überprüft, ob das Ticket bereits verfallen ist und daher stets nur frische Tickets benutzt werden können, werden Änderungen des Passworts oder der Zugriffsrechte im Server innerhalb weniger Minuten wirksam.

Wenn die folgende Kommunikation komplett verschlüsselt würde, dann wäre die einfache Mitteilung des Tickets an B bereits ausreichend. Da in der Regel heute in Workstations jedoch keine effiziente Krypto-Hardware vorhanden ist und eine Software-Implementation eines symmetrischen Algorithmus für große Datenmengen zu ineffizient ist, verzichtet man derzeit oft auf die vollständige Sicherung. Man geht von der (natürlich sehr gefährlichen) Annahme aus, daß eine einmal hergestellte Verbindung für einen Angreifer nicht mehr umlenkbar ist. Da aber jeder zuvor abgehörte und evtl. noch gültige Tickets verschicken kann, müssen A und B zu Beginn der Kommunikation wenigstens einmal mit K_{ab} verschlüsselte Daten austauschen. Dazu sendet A an B seinen Namen und seine Uhrzeit (c) und B antwortet als Beweis, daß er diese Nachricht empfangen hat und auch entschlüsseln konnte (und damit K_{ab} kennt und folglich K_{bs} kennt also nur B sein kann wenn wir B und S vertrauen) mit der um eine Sekunde erhöhten Uhrzeit (d).

In einem echten Kerberos-System werden noch eine Reihe von Erweiterungen des eben beschriebenen Verfahrens genutzt, um einen effizienteren und ausfallsichereren Betrieb zu gewährleisten. So besteht der Kerberos-Server aus zwei getrennten Prozessen, dem eigentlichen Kerberos-Server mit der Benutzerdatenbank und einem Ticket-Granting-Server mit der Serverdatenbank. Zu Beginn einer Sitzung gibt ein Benutzer ein Passwort ein und erhält dadurch vom Kerberos-Server ein Ticket-Granting-Ticket. Mit diesem kann er z.B. 24 Stunden lang von jedem Ticket-Granting-Server Tickets für die gerade benötigte Kommunikation erhalten. Dieses Ticket-Granting-Ticket-Prinzip hat eine Reihe von Vorteilen:

- Das Ticket-Granting-Ticket darf eine viel längere Laufzeit haben, und das Passwort, das nötig ist um es zu erlangen, muß nur einmal zu Beginn der Sitzung und nicht bei jeder Verbindung zu einem Server eingegeben werden.
- Der dauerhafte Schlüssel K_{as} wird weniger oft benutzt und damit stehen einem Angreifer weniger Pakete für einen known-plaintext-attack zur Verfügung.
- Der Ticket-Granting-Server kann auf anderen Rechnern laufen als der Kerberos-Server. Da der Ticket-Granting-Server wesentlich öfter benötigt wird als der Kerberos-Server, kann so eine zentrale Maschine entlastet werden. Ticket-Granting-Server sind wegen der kleineren Serverdatenbank mit selteneren Änderungen auch leichter zu replizieren.

5 Weitere Authentifizierungssysteme

Kerberos authentifiziert nur einen client mit einem server. Dabei wird stillschweigend angenommen, daß die Software der lokalen Workstation vertrauenswürdig ist und nicht z.B. das Passwort weitergibt. Auch bietet Kerberos nicht an, etwa per remote-job-entry einem auf einem weiteren Rechner laufendem Program einen Auftrag und die zur Abwicklung dieses Auftrages notwendigen Tickets zukommen zu lassen. In [2] berichten Lampson et al. über ein System und eine zugrunde liegende Theorie, die weit über die von Kerberos gebotenen Dienste hinaus gehen.

Beispielsweise enthält in diesem System jede Workstation in der als Firmware installierten Bootsoftware einen eigenen secret key, den nur die Bootsoftware kennt. Durch geeignete Hardware wird sichergestellt, daß nicht einmal das lokal laufende Betriebssystem Zugriff darauf hat. Nach dem Laden des Systemkerns fertigt die Bootsoftware ein message digest des Kernels zusammen mit einem Zeitstempel und dem Namen der Maschine an. Auf Wunsch stellt die Bootsoftware dem laufenden Betriebssystem ein Zertifikat über dessen Version aus, das dann das Betriebssystem als Beweis für seine Identität weitergeben kann. So kann z.B. ein entfernter Benutzer sicher sein, daß er mit einer bestimmten Hardware kommuniziert und eine bestimmte vertrauenswürdige Kernelversion vorliegen hat, unter der Annahme, daß die Hardware, nicht aber die Systemsoftware gegen Manipulationen geschützt ist (eine realistische Annahme an Universitäten und in Entwicklungsabteilungen).

Entsprechend kann ein laufendes Betriebssystem nun wiederum mit einem unterschriebenen message digest des Binärcodes eines Servers oder sonst eines entfernt gestarteten Programmes dessen Version bestätigen, bevor der Client vertrauliche Daten oder Tickets an diese Programmversion übergibt. Damit kann ein Programm Rechte (z.B. in Form von Zertifikaten) an andere Programme mit Auflagen abgeben (z.B. „Dieses Ticket erlaubt dem Programmcode x der unter dem Betriebssystem y auf Maschine z läuft für die nächsten 20 Minuten auf Datei f lesend zuzugreifen.“) Mit derartigen Mechanismen kann man in beschränktem Maße sogar auf das Vertrauen auf die Systemadministratoren verzichten, sofern der secret key der Bootsoftware schwer zugänglich in die Hardware (z.B. in den Microcode des Prozessors) integriert wurde. Damit ist ein guter Schutz gegen „Viren“ und „Trojanische Pferde“ möglich.

In [2] wird ein Formalismus beschrieben, der auf einer **speaks for** Relation zwischen Objekten (Programmen, Keys, Kommunikationskanäle, etc.) aufbaut. Beispielsweise führt ein gültig zertifizierter public key dazu, daß diesem key genauso vertraut wird, wie dessen Besitzer direkt. Der key „spricht“ an Stelle des Besitzers und ebenso spricht eine mit diesem Key unterschriebene Nachricht oder eine unterschriebene Software an Stelle des Besitzers für ihn. Mit einer derartigen Theorie soll sichergestellt werden, daß vom System erlaubtes Vertrauen in empfangene Informationen stets formal nachprüfbar bleibt.

6 Formale Protokollverifikation

In [1] stellen Burrows, Abadi und Needham eine einfache Prädikatenlogik zur Überprüfung von Authentisierungsprotokollen vor. Damit lassen sich die gültigen Annahmen, von denen die Kommunikationspartner nach Anwendung eines Protokolls ausgehen dürfen, an Hand von Voraussetzungen, Regeln und den ausgetauschten Daten formal herleiten. Diese Logik ist einfach genug anwendbar um auch als Hilfe beim Verständnis von Protokollen zu dienen und Schwachstellen oder Redundanzen aufzeigen zu können.

Die Logik unterscheidet im wesentlichen drei Objekttypen: Kommunikationspartner (*principals*), Schlüssel und Aussagen. Im folgenden bezeichen wie bisher die Buchstaben A, B und S bestimmte Kommunikationspartner, K ist ein public key und K^{-1} ein secret key.

Als Variablen für Kommunikationspartner in Regeln stehen P, Q und R, und für beliebige Aussagen stehen X und Y. In Regeln repräsentiert das Komma eine Konjunktion (logisches „und").

Die folgenden Konstrukte werden in der Logik genutzt um Aussagen zu treffen:

- P **believes** X bedeuted, daß P davon ausgeht, daß die Aussage X wahr ist oder Grund dazu hat, davon auszugehen.
- P **sees** X bedeuted, daß P die Aussage X als Nachricht empfangen hat, sie falls notwendig erfolgreich entschlüsselt hat und sie lesen und weitersenden kann.
- P **said** X bedeuted, daß P irgendwann einmal die Aussage X gesendet hat und zu diesem Zeitpunkt galt P **believes** X.
- P **controls** X bedeuted, daß P die Authorität für die Aussage X ist und daß P vertraut werden kann, daß die Aussage X richtig ist wenn sie von P kam.
- **fresh**(X) bedeuted, daß die Aussage X irgend eine beliebige neue Nachricht ist, die noch nie zuvor im Protokoll aufgetaucht ist. Das kann z.B. die aktuelle Uhrzeit sein, oder eine gerade neu erzeugte große Zufallszahl (*nonce*). Diese Aussage wird gebraucht, damit die Regeln *replay attacks* vermeiden können, bei denen ein Angreifer einfach alte abgehörte Pakete neu versendet.
- $P \overset{K}{\leftrightarrow} Q$ bedeuted, daß K ein key ist, der zur sicheren Kommunikation zwischen P und Q genutzt werden kann, und den nur P und Q sowie Partner, denen diese vertrauen kennen.
- $\overset{K}{\rightarrow} P$ bedeuted, daß K der public key von P ist und daß nur P oder jemand dem P vertraut K^{-1} kennt.
- $P \overset{X}{\rightleftharpoons} Q$ bedeuted, daß X ein Geheimnis (z.B. ein Passwort) ist, das nur P und Q kennen oder Partner denen diese vertrauen.
- $\{X\}_K$ ist wie schon bisher eine Schreibweise dafür, daß die Aussage X mit den Schlüssel K verschüsselt wurde.
- $\langle X\rangle_Y$ bedeuted, daß an die Nachricht X das Geheimnis Y angehängt wurde um die Identität dessen zu belegen, der $\langle X\rangle_Y$ gesendet hat.

Die Logik betrachtet Aussagen getrennt für die Vergangenheit und die Gegenwart, d.h. Aussagen, die in der Vergangenheit zutrafen werden nicht automatisch in die Gegenwart übernommen. Nachrichten, die nicht in einem einzigen Block verschlüsselt waren, werden wie getrennt ankommende Nachrichten behandelt, da getrennt verschlüsselte Blöcke von Angreifern ausgetauscht werden können. Für die Logik werden nur verschlüsselt übertragene Informationen betrachtet, da nur sie überhaupt mit Sicherheit aus einer vorgegebenen Quelle stammen können. Unverschlüsselt übertragene Daten sind nur zusätzliche Hinweise, welche Information in einem verschlüsselten Paket enthalten sein sollte (z.B. die erste Kerberos Nachricht, daß A mit B kommunizieren will). Sie haben aber keine Relevanz für die Sicherheit des Protokolls, da sie jederzeit gefälscht worden sein könnten. Wenn eine Nachricht verschlüsselt wurde, dann geht die Logik stets davon aus, daß die Nachricht genügend Redundanz enthält, so daß nach dem Entschlüsseln offensichtlich ist, ob der Schlüssel korrekt war.

Die Ableitungsregeln der Logik sind die folgenden:

a) Wenn P davon ausgehen kann, daß nur P und Q den Schlüssel K kennen und P irgendwoher eine mit K verschlüsselte Nachricht X empfangen hat, dann darf P davon ausgehen, daß Q irgendwann in der Vergangenheit die Nachricht X abgegeben hat.

$$\frac{P\ \textbf{believes}\ Q \overset{K}{\leftrightarrow} P, \quad P\ \textbf{sees}\ \{X\}_K}{P\ \textbf{believes}\ Q\ \textbf{said}\ X}.$$

Da außer Q auch noch P den Schlüssel K kennt, muß zusätzlich noch abgeklärt werden, daß $\{X\}_K$ nicht irgendwann mal von P selbst erzeugt worden ist. Wir schreiben für verschlüsselte Pakete daher im Zweifelsfall auch $\{X\}_K$ *from* R und wenden die Regel nur bei $R \neq P$ an.

b) Zu der Regel a) für symmetrische Kryptosysteme gibt es eine entsprechende Regel für asymmetrische Systeme:

$$\frac{P\ \textbf{believes}\ \overset{K}{\rightarrow} Q, \quad P\ \textbf{sees}\ \{X\}_{K^{-1}}}{P\ \textbf{believes}\ Q\ \textbf{said}\ X}.$$

c) Ebenso kann der Ursprung von Nachrichten dadurch bestätigt werden, daß die Nachricht ein Passwort enthält:

$$\frac{P\ \textbf{believes}\ Q \overset{Y}{\rightleftharpoons} P, \quad P\ \textbf{sees}\ \langle X\rangle_Y}{P\ \textbf{believes}\ Q\ \textbf{said}\ X}.$$

Diese Regel macht Sinn, da die Logik wie oben erwähnt ohnehin nur insgesamt verschlüsselte Nachrichten betrachtet und daher das Passwort nicht ausgespäht werden kann.

d) Die folgende Regel besagt: wenn P weiß, daß Q in der Vergangenheit einmal von X ausging, und X eine neue (in der Gegenwart erzeugte) Nachricht ist, dann weiß P, daß Q derzeit von X ausgeht:

$$\frac{P\ \textbf{believes}\ \textbf{fresh}(X), \quad P\ \textbf{believes}\ Q\ \textbf{said}\ X}{P\ \textbf{believes}\ Q\ \textbf{believes}\ X}$$

.

e) Wenn P weiß, daß Q die Autorität für die Aussage X ist und weiß, daß Q auch X glaubt, dann darf P auch an X glauben:

$$\frac{P \textbf{ believes } Q \textbf{ controls } X, \quad P \textbf{ believes } Q \textbf{ believes } X}{P \textbf{ believes } X}.$$

f) Ferner benötigen wir einige Regeln, die festlegen, wann überhaupt P eine Nachricht lesen kann. (X, Y) bedeuted dabei, daß X und Y beides Komponenten der gleichen Nachricht sind.

$$\frac{P \textbf{ sees } (X, Y)}{P \textbf{ sees } X}, \qquad \frac{P \textbf{ sees } \langle X \rangle_Y}{P \textbf{ sees } X},$$

$$\frac{P \textbf{ believes } Q \overset{K}{\leftrightarrow} P, \quad P \textbf{ sees } \{X\}_K}{P \textbf{ sees } X},$$

$$\frac{P \textbf{ believes } \overset{K}{\rightarrow} P, \quad P \textbf{ sees } \{X\}_K}{P \textbf{ sees } X},$$

$$\frac{P \textbf{ believes } \overset{K}{\rightarrow} Q, \quad P \textbf{ sees } \{X\}_{K^{-1}}}{P \textbf{ sees } X}.$$

Entsprechend wie bei a) dürfen die Regeln nur angewendet werden, wenn $\{X\}_K$ nicht von P selbst erzeugt wurde.

g) Wenn Teile einer Nachricht aus der Gegenwart stammen, dann auch die ganze Nachricht:

$$\frac{P \textbf{ believes fresh}(X)}{P \textbf{ believes fresh}(X, Y)}.$$

h) Die Aussage A **believes** S **controls** $A \overset{K}{\leftrightarrow} B$ steht für alle möglichen K, also in Quantorenschreibweise: A **believes** $\forall K.(S$ **controls** $A \overset{K}{\leftrightarrow} B)$. Die folgende Regel erlaubt uns, die Variablen, die in diesem Fall nach dem Quantor stehen können, durch konkrete Werte zu ersetzen:

$$\frac{P \textbf{ believes } \forall V_1, \ldots, V_n.(Q \textbf{ controls } X)}{P \textbf{ believes } Q' \textbf{ controls } X'},$$

wobei Q' und X' durch gleichzeitiges Einsetzen der Variablen $V_1, \ldots, V_n$ entsteht.

i) Wenn eine Aussage X vor einer Nachricht $P \rightarrow Q : Y$ gilt, dann gilt nach der Nachricht sowohl X als auch Q **sees** Y.

j) Wenn Y aus X abgeleitet werden kann, dann gilt Y immer wenn X gilt (modus ponens).

Bei normalen Protokollbeschreibungen, wie etwa beim obigen Kerberos-Protokoll, werden nur die in den Nachrichten mitgeführten Daten notiert, nicht aber ihre Bedeutung. Die Logik hantiert aber mit Aussagen, und daher muß das Protokoll erst in eine Form gebracht werden, in der Aussagen statt nur Daten übermittelt werden. Beispielsweise besagt

$$A \rightarrow B : \quad \{A, K_{ab}\}_{K_{bs}}$$

lediglich, daß die Daten A und K_{ab} an B geschickt werden. Dagegen ist in

$$A \rightarrow B : \quad \{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}$$

sofort klar, daß an B übermittelt wird, daß K_{ab} ein nur A und B bekannter key ist. Die Übersetzung des reinen Paketformates in diese idealisierte Protokollbeschreibung ist zwar kaum automatisch durchführbar, bereitet jedoch in der Praxis wenig Probleme. Außerdem erleichtert diese Notation das Verständnis von Protokollen, da die Bedeutung der übertragenen Daten klar wird. Das idealisierte Kerberos-Protokoll sieht wie folgt aus:

$$\begin{array}{lll}
\text{(a)} & S \rightarrow A: & \{T_s, A \overset{K_{ab}}{\leftrightarrow} B, \{T_s, A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}\}_{K_{as}} \\
\text{(b)} & A \rightarrow B: & \{T_s, A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}, \\
 & & \{T_a, A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{ab}} \textit{ from } A \\
\text{(c)} & B \rightarrow A: & \{T_a, A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{ab}} \textit{ from } B
\end{array}$$

Die unverschlüsselte Anforderung für ein Ticket ignorieren wir, da sie zur Sicherheit des Protokoll nichts beiträgt und nur den Server zu Nachricht (a) veranlaßt. Die Lebensdauer L des Tickets wurde zur Vereinfachung mit der Uhrzeit T_s zusammengefaßt.

Als Ergebnis einer Protokollverifikation könnten zum Beispiel die Aussagen

$$A \textbf{ believes } A \overset{K}{\leftrightarrow} B$$

$$B \textbf{ believes } A \overset{K}{\leftrightarrow} B$$

stehen, das heißt, beide Partner wissen, daß sie den korrekten key erhalten haben. Manche Protokolle erlauben sogar noch die stärkeren Aussagen

$$A \textbf{ believes } B \textbf{ believes } A \overset{K}{\leftrightarrow} B$$

$$B \textbf{ believes } A \textbf{ believes } A \overset{K}{\leftrightarrow} B$$

die besagen, daß der jeweils andere Partner auch den Beweis erhält, daß der key beim richtigen Partner angekommen ist. Andere Ziele einer Verifikation könnten Aussagen wie A **believes** B **believes** X sein, also daß A den Beweis hat, daß B jetzt X empfangen und als korrekt erkannt hat.

Bei der Analyse des Kerberos-Protokolls gehen wir von folgenden Voraussetzungen aus:

$$A \textbf{ believes } A \overset{K_{as}}{\leftrightarrow} S$$

$$B \textbf{ believes } B \overset{K_{bs}}{\leftrightarrow} S$$

$$S \textbf{ believes } A \overset{K_{as}}{\leftrightarrow} S$$

$$S \textbf{ believes } B \overset{K_{bs}}{\leftrightarrow} S$$

$$S \textbf{ believes } A \overset{K_{ab}}{\leftrightarrow} B$$

$$A \textbf{ believes } S \textbf{ controls } A \overset{K}{\leftrightarrow} B$$

$$B \textbf{ believes } S \textbf{ controls } A \overset{K}{\leftrightarrow} B$$

$$A \textbf{ believes } \textbf{fresh}(T_s)$$

$$B \textbf{ believes } \textbf{fresh}(T_s)$$

$$B \textbf{ believes } \textbf{fresh}(T_a)$$

Nachdem Nachricht (a) von S and A empfange wurde gilt wegen Regel i)

$$A \textbf{ sees } \{T_s, A \overset{K_{ab}}{\leftrightarrow} B, \{T_s, A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}\}_{K_{as}}$$

und wegen der Annahme A **believes** $A \overset{K_{as}}{\leftrightarrow} S$ können wir mit Regel a) folgern, daß

$$A \textbf{ believes } S \textbf{ said } (T_s, A \overset{K_{ab}}{\leftrightarrow} B, \{T_s, A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}).$$

Nun entfernen wir mit Regel f) das Ticket und da laut Voraussetzung gilt A **believes fresh**(T_s) folgern wir mit Regel g)

$$A \textbf{ believes } S \textbf{ believes } (T_s, A \overset{K_{ab}}{\leftrightarrow} B)$$

und daher auch

$$A \textbf{ believes } S \textbf{ believes } A \overset{K_{ab}}{\leftrightarrow} B.$$

Nun wenden wir auf die Annahme

$$A \textbf{ believes } S \textbf{ controls } A \overset{K}{\leftrightarrow} B$$

die Regel h) mit dem speziellen key K_{ab} an und erhalten

$$A \textbf{ believes } S \textbf{ controls } A \overset{K_{ab}}{\leftrightarrow} B$$

womit wir schließlich mit Regel e) das erste Ziel

$$A \textbf{ believes } A \overset{K_{ab}}{\leftrightarrow} B$$

bereits nach der ersten Nachricht erhalten. Nachdem A in Nachricht (b) das Ticket an B weitergeleitet hat, können wir mit einer analogen Herleitung

$$B \textbf{ believes } A \overset{K_{ab}}{\leftrightarrow} B$$

zeigen. Nun kann B auch die zweite Hälfte von Nachricht (b) entschlüsseln und eine ähnliche Herleitung führt zu der Aussage

$$B \textbf{ believes } A \textbf{ believes } A \overset{K_{ab}}{\leftrightarrow} B$$

und nach der letzten Nachricht (c) kann entsprechend auch

$$A \textbf{ believes } B \textbf{ believes } A \overset{K_{ab}}{\leftrightarrow} B$$

hergeleitet werden.

Bei genauer Betrachtung der Herleitung fällt auf, daß die doppelte Verschlüsselung des Tickets mit K_{bs} und K_{as} in (a) garnicht notwendig war. Es wurde daher bereits vorgeschlagen, aus Effizienzgründen das Ticket nicht mit K_{as} zu verschlüsseln, sondern einfach nur anzuhängen.

In [1] wird diese Technik außerdem noch auf das Andrew RPC Protokoll, das public key Needham-Schröder Protokoll und die X.509 Authentifizierung angewendet. Interessant ist insbesondere eine intuitiv nicht sofort einsichtige Schwäche des X.509 Protokolls, die eine Verifikation mit dieser Logik jedoch sofort aufdeckt. Dieses Beispiel (X.509 ist schließlich eine Norm, die von vielen Experten zuvor überprüft wurde) und die Einsicht daß die Sicherheit prinzipiell nicht durch Testen und nur mit Schwierigkeiten durch intuitives Nachvollziehen überprüft werden kann, zeigen, wie wichtig formale Arbeitstechniken beim Entwurf von Sicherheitsprotokollen sind.

Literatur:

[1] Burrows, M.; et al.: *A Logic of Authentication*, ACM Trans. on Computer Systems 8, 1 (February 1990).

[2] Lampson, B.; et al.: *Authentication in Distributed Systems: Theory and Practice*, Proc. 13th ACM Symposium on Operating Systems Principles (October 1991).

[3] Satyanarayanan, M.: *Integrating Security in a Large Distributed System*, ACM Trans. on Computer Systems 7, 3 (August 1989).

[4] Steiner, J. G.; et al.: *Kerberos: An Authentication Service for Open Network Systems*, Proc. 1988 USENIX Winter Conf.

[5] Beutelsbacher, Albrecht: *Kryptologie*, Vieweg, Braunschweig 1991.

[6] Rivest, R.; Shamir, A.; Adleman, L.: *A method for obtaining digital signatures and public-key cryptosystems*, CACM 21, 2, Feb. 1978, pp 120-126.

[7] Rivest, R.: *The MD5 Message-Digest Algorithm*, RFC 1321, Apr. 1992.

[8] Zimmermann, P.: *PGP – Pretty Good Privacy Software*, anonymous ftp, `ftp.dsi.unimi.it`, `pub/security/crypt/PGP/`, Internet.