Einsatz kryptographischer Methoden zur Authentisierung

Jörg Czeranski

TU Clausthal

czeranski@informatik.tu-clausthal.de

Zusammenfassung

Client-Server-Architekturen erfordern sichere Mechanismen zur Authentisierung. Es wird ein Überblick über bei Unix-Systemen verbreitete Verfahren gegeben und auf ihre Unzulänglichkeiten eingegangen. Als Ergänzung und Alternative wird ein kryptographisches Verfahren, Secure RPC, vorgestellt und die Schlüsselverwaltung dieses Verfahrens detailliert erläutert.

1 Wozu Authentisierung?

Eine wichtige Anforderung an Client-Server-Architekturen ist, daß Anfragen des Clients an den Server – wie Auslesen oder Speichern von Daten oder Ausführen von Programmen durch den Server – nur bearbeitet werden, wenn der Client berechtigt ist, diese Anfragen zu stellen. Um dies zu erreichen, verlangt man, daß der Client seine Berechtigung nachweist, sich authentisiert.

Im folgenden wird untersucht, welche Möglichkeiten zur Authentisierung es gibt, wenn Client und Server auf getrennten Maschinen laufen, die durch ein Netzwerk verbunden sind. In den Beispielen wird angenommen, daß die Maschinen unter Unix laufen und per TCP/IP verbunden sind, die Mechanismen lassen sich jedoch leicht auf andere Systeme übertragen.

2 Wie authentisiert Unix über TCP/IP?

Unix kennt eine ganze Reihe von verschiedenen Authentisierungsmechanismen, die jeweils für einen bestimmten, zum Teil sehr speziellen Typ von Servern entworfen wurden. Oft müssen mehrere Mechanismen kombiniert werden, um eine wirkungsvolle Kontrolle zu erreichen.

Im einzelnen sind dies:

- Host-Listen

  Der Client wird authentisiert, indem seine IP-Adresse beziehungsweise sein Name, der aus der IP-Adresse abgeleitet wird, mit einer Liste von zugelassenen Adressen verglichen wird. Diesen Authentisierungstyp verwenden zum Beispiel `rsh/rlogin` (über `/etc/hosts.equiv`), NFS (über `/etc/exports`) und X11 (über das `xhost`-Programm). Dieser Mechanismus schränkt alleine noch nicht den Zugriff auf einzelne Benutzer ein. Außerdem kann nicht ohne zusätzliche Maßnahmen sichergestellt werden, daß die IP-Adresse nicht gefälscht wird (siehe auch Abschnitt 3).

- Übertragen der UserID

  Um den Zugriff auf einzelne Benutzer einzuschränken, wird in der Anfrage der Accountname (die „UserID“) mit übertragen. Diese Authentisierung wird von `rsh/rlogin` und NFS als Ergänzung der Host-Listen verwendet, von `telnet` und `ftp` anstatt der Host-Listen. Wie die IP-Adresse des Clients kann der Server auch diese Information nicht überprüfen. Deshalb muß noch ein weiterer Mechanismus verwendet werden, um die Korrektheit der UserID zu gewährleisten und streng genommen erst so eine Authentisierung zu erreichen.

- Übertragen des Benutzer-Paßworts

  Clients wie `telnet` und `ftp` übertragen in der Anfrage zusätzlich zur UserID des Benutzers noch dessen Paßwort, das der Server durch Vergleich mit dem in `/etc/passwd` oder NIS (früher „Yellow Pages“) gespeicherten verifizieren kann.

- „Trusted Ports“

  Das „Trusted Port“-Konzept verlangt, daß der Zugriff auf bestimmte TCP/IP-Ports (alle mit Nummern unter 1024) dem Superuser vorbehalten ist. Der Server läßt dann eine Anfrage nur zu, wenn der Client einen solchen „Trusted Port“ verwendet. Dadurch wird bei Diensten wie `rsh/rlogin` und NFS sichergestellt, daß die in der Anfrage angegebene UserID durch den Superuser auf der Clientmaschine überprüft werden kann. Dieser Mechanismus setzt natürlich voraus, daß erstens der Server dem Superuser des Clients vertraut, zum Beispiel weil der Client von denselben Personen administriert wird, und daß zweitens auch auf der Clientmaschine dieses Konzept implementiert ist.

- Übertragen eines „Magic Cookie“

  Ist bereits einmal eine Authentisierung erfolgt, kann man die Authentisierung bei weiteren Anfragen desselben Clients an denselben Server vereinfachen, indem der Server dem Client nach der ersten Authentisierung einen „Magic Cookie“ (irgendein möglichst zufälliger Datenblock) übermittelt. Dieser „Cookie“ wird dann bei weiteren Anfragen vom Client wieder an den Server übertragen, der so feststellen kann, daß er noch mit demselben Client kommuniziert. Dieses Verfahren verwenden NFS, das sogenannte Filehandles als „Cookie“ verwendet, und X11 über `.Xauthority` und das `xauth`-Programm mit „MIT-MAGIC-COOKIE-1“ als Authentisierungstyp.

- keine Authentisierung

  Einige Dienste, wie `finger`, `rdate` und `anonymous ftp` benötigen keine Authentisierung, sondern erlauben jedem Client, Anfragen zu stellen.

3 Wozu kryptographische Methoden?

Für den Benutzer ist es wünschenswert, einen Authentisierungsmechanismus zu verwenden, der von möglichst vielen Clientmaschinen aus für möglichst viele Dienste verwendet werden kann, weitgehend transparent ist und hinreichend sicher ist, um für seine Daten einen wirkungsvollen Schutz zu gewährleisten.

Die bisher vorgestellten Mechanismen haben mehrere Schwachstellen, sie sind:

- zu Unix-spezifisch

  Das „Trusted Port“-Konzept setzt voraus, daß auf der Clientmaschine Accounts und ein vertrauenswürdiger Superuser vorhanden sind. Das schließt zum Beispiel öffentlich zugängliche PCs als Clients aus.

- abhörbar

  Paßwörter und „Magic Cookies“ werden im Klartext über das Netzwerk übertragen. Falls ein Angreifer diese „abhören“ kann – indem er zum Beispiel einen PC oder eine Workstation, auf der er Superuser ist, alle über sein lokales Netz übertragenen Daten mitlesen läßt – kann er sich mit diesen später unberechtigterweise authentisieren.

- fälschbar

  Das Vergleichen von IP-Adressen mit Listen von zugelassenen Adressen setzt voraus, daß man darauf vertrauen kann, daß eine IP-Adresse nur von der Maschine verwendet wird, der sie zugeordnet ist. TCP/IP sieht allerdings keine Mechanismen vor, die diese garantieren, so daß ein Angreifer durch Verwenden von Adressen anderer Maschinen den Server täuschen kann.

Um diese Schwachstellen auszuschalten, kann man ein Verfahren verwenden, das ähnlich dem Paßwort-Mechanismus bei jeder Anfrage an den Server einen Code überträgt, den nur der Benutzer selber kennt, diese Übertragung jedoch mit einem kryptographischen Verfahren verschlüsselt.

Eine Implementation eines solchen Authentisierungsmechanismus ist das von Sun Microsystems entwickelte „Secure RPC,“ das auch von anderen Herstellern, wie etwa IBM, verwendet wird.

4 Wie funktioniert Secure RPC?

4.1 Was ist RPC?

RPC („Remote Procedure Call“) ist eine auf TCP und UDP aufgesetzte Protokollschicht, die ein allgemeines Client-Server-Modell mit maschinenunabhängiger Datenrepräsentation implementiert. Dazu gehört auch ein Entwicklungswerkzeug, das Prozeduraufrufe automatisch in Netzwerkanfragen umsetzt.

Verwendet wird RPC sowohl von komplexen Systemen wie NFS, NIS und NIS+ als auch von vergleichsweise einfachen Programmen wie `rusers`, `ruptime` und `rwall`.

RPC unterstützt bisher vier Authentisierungstypen:

- „None“: keine Authentisierung, zum Beispiel für `rusers`.
- „Unix“: Authentisierung durch Übertragen der UserID, zum Beispiel für NFS.
- „DES“: Authentisierung durch ein kryptographisches Verfahren (Secure RPC), zum Beispiel für NFS mit der Option `secure`, X11 mit dem Authentisierungstyp „SUN-DES-1“ und NIS+.
- „Kerberos“: Authentisierung über einen zentralen Authentisierungsserver, wird für NFS mit der Option `kerberos` verwendet.

Die Einbindung der Authentisierung in RPC ist so angelegt, daß später weitere Authentisierungstypen definiert werden können.

Auf die Typen „None“ und „Unix“ wurde bereits in Abschnitt 2 eingegangen. Die Implementierung des Typs „Kerberos“ deckt bisher nur einen sehr eingeschränkten Bereich von RPC-Diensten ab und ist wohl zur Zeit nur von Bedeutung, wenn man Kerberos schon unabhängig von RPC verwendet. Das gesamte Kerberos-System und das Zusammenspiel mit RPC vorzustellen, würde den Rahmen dieses Textes sprengen. Wegen der wachsenden Bedeutung von NIS+ fiel deshalb die Wahl auf Secure RPC. Einen Überblick über Kerberos findet man in [1] und [2].

4.2 Wie authentisiert Secure RPC?

Secure RPC verwendet das Diffie-Hellman-Verfahren zur Erzeugung eines gemeinsamen Schlüssels für Client und Server. Dieses Verfahren und der darauf aufbauende Authentisierungsmechanismus werden im folgenden vorgestellt.

Jedem Benutzer und jeder Maschine werden ein sogenannter Netzname und ein Paar aus einem geheimen und einem öffentlichen Schlüssel zugeordnet.

Der Netzname eines Benutzers hat die Form `unix.<UserID>@<Domain>`, der einer Maschine die Form `unix.<Adresse>@<Domain>`. Der Superuser auf einer Maschine wird hier mit der Maschine gleichgesetzt und verwendet als Netzname `unix.<Adresse>@<Domain>`, nicht `unix.0@<Domain>`.

Der geheime Schlüssel D wird als zufällige Zahl kleiner als P gewählt, der öffentliche Schlüssel E nach der Formel

$$E \equiv \alpha^D \pmod P$$

berechnet. Hierbei sind α und P Konstanten, die für Secure RPC von Sun global festgelegt sind.

Will ein Benutzer mit dem Schlüsselpaar D_B, E_B mit einem Server mit Schlüsselpaar D_S, E_S kommunizieren, kann der Client für den Benutzer aus dessen geheimem Schlüssel D_B und dem öffentlichen Schlüssel E_S des Servers einen Schlüssel

$$K \equiv {E_S}^{D_B} \pmod P$$

berechnen. Ebenso berechnet der Server aus seinem geheimen Schlüssel D_S und dem öffentlichen Schlüssel E_B des Benutzers einen Schlüssel

$$\hat{K} \equiv {E_B}^{D_S} \pmod P.$$

Wegen

$$K \equiv {E_S}^{D_B} \equiv (\alpha^{D_S})^{D_B} \equiv \alpha^{D_S \cdot D_B} \equiv (\alpha^{D_B})^{D_S} \equiv {E_B}^{D_S} \equiv \hat{K} \pmod P$$

sind K und $\hat{K}$ äquivalent und können als gemeinsamer Schlüssel verwendet werden.

Der Client authentisiert den Benutzer in Anfragen an den Server, indem er mit der Anfrage die aktuelle Uhrzeit zum einen im Klartext und zum anderen mit dem gemeinsamen Schlüssel K verschlüsselt überträgt. Der Server kann dann überprüfen, daß die verschlüsselte Uhrzeit mit der im Klartext übertragenen übereinstimmt und weiß somit, daß der Client den gemeinsamen Schlüssel K und dafür den geheimen Schlüssel D_B des Benutzers kennen muß.

Zum Verschlüsseln der Uhrzeit wird das DES-Verfahren („Data Encryption Standard“) verwendet.

Die Sicherheit dieses Authentisierungsverfahrens hängt davon ab, daß aus den öffentlichen Daten (E, α und P) auf die mit ihnen verknüpften geheimen Daten (D) nicht geschlossen werden kann. Ein Angreifer müßte also versuchen, ein $\hat{D}$ zu finden, das die Bedingung

$$E \equiv \alpha^{\hat{D}} \pmod P$$

erfüllt. Dies erfordert nach dem momentanen Kenntnisstand bei hinreichend großem P zuviel Rechenzeit, um praktisch durchführbar zu sein, siehe auch Abschnitt 5.2.

4.3 Wie werden die Schlüssel gespeichert?

Um sich im Netzwerk authentisieren zu können, muß ein Benutzer seinen eigenen geheimen Schlüssel und die öffentlichen Schlüssel der Server, die er ansprechen will, abfragen können.

Zur Speicherung werden die geheimen Schlüssel mit dem Login-Paßwort des jeweiligen Benutzers beziehungsweise des Superusers verschlüsselt, so daß ein Benutzer nur Zugriff auf seinen eigenen geheimen Schlüssel bekommen kann. Zur Verschlüsselung wird auch hier das DES-Verfahren verwendet.

Die Schlüsselpaare der Benutzer und Maschinen werden dann in der Form
`<Netzname> <öffentlicher Schlüssel>:<codierter geheimer Schlüssel>`
an zentraler Stelle gespeichert. Für die Speicherung wird

- bei NIS die Map `publickey.byname`,
- bei NIS+ die Table `cred.org_dir`[3],
- andernfalls die Datei `/etc/publickey`

verwendet.

4.4 Wie wird auf den geheimen Schlüssel zugegriffen?

Wenn sich der Benutzer einloggt, wird sein geheimer Schlüssel in der verschlüsselten Form abgefragt, dann mit seinem Login-Paßwort entschlüsselt und einem lokal laufenden `keyserv`-Dämon übertragen, der ihn für den Benutzer bereit hält.

Sobald der Benutzer einen Dienst verwendet, der über Secure RPC authentisiert wird, läßt der aufgerufene Clientprozess den `keyserv`-Dämon aus diesem Schlüssel und dem öffentlichen Schlüssel der Servers den gemeinsamen Schlüssel K berechnen und verwendet diesen zur Authentisierung.

Falls sich der Benutzer nicht über `telnet` oder per `login` an einem direkt angeschlossenen Terminal, sondern per `rlogin` eingeloggt hat, hat er der Maschine sein Login-Paßwort nicht mitgeteilt und der `keyserv`-Dämon kennt den geheimen Schlüssel nicht. In diesem Fall kann er das `keylogin`-Programm verwenden um seinen geheimen Schlüssel zu entschlüsseln und dem `keyserv`-Dämon mitzuteilen.

Benötigt der Benutzer seinen geheimen Schlüssel nicht mehr (weil er sich ausloggt), kann er dieses über das `keylogout`-Programm dem `keyserv`-Dämon mitteilen, der den geheimen Schlüssel dann aus seinem Speicher löscht.

Auf einer Maschine, auf der ein über Secure RPC authentisierter Server läuft, muß der `keyserv`-Dämon den geheimen Schlüssel des Superusers bereithalten. Dies ist notwendig, da der Server bei einer Anfrage eines Benutzers durch einen Client den gemeinsamen Schlüssel K aus dem öffentlichen Schlüssel des Benutzers und dem geheimen Schlüssel des Superusers (also der Servermaschine) berechnet.

Da der geheime Schlüssel des Superusers dem `keyserv`-Dämon nach einem Reboot nicht bekannt ist – und deshalb jemand als Superuser `keylogin` aufrufen müßte, bevor wieder über Secure RPC authentisierte Dienste angeboten werden können – wird der geheime Schlüssel des Superusers durch den `keyserv`-Dämon zusätzlich in der Datei `/etc/.rootkey` im Klartext gespeichert.

Ändert der Benutzer sein Login-Paßwort, so wird der geheime Schlüssel durch das `passwd`-Programm mit dem alten Login-Paßwort entschlüsselt, mit dem neuen verschlüsselt und wieder in der zentralen Schlüsselliste gespeichert. Dadurch kann der Benutzer beim nächsten Einloggen mit seinem dann aktuellen Login-Paßwort wieder auf den geheimen Schlüssel zugreifen.

Der Secure RPC-Mechanismus ist auch auf einem Singleuser-System wie etwa einem PC implementierbar. Der Benutzer muß nur zusätzlich dem System seinen Netznamen mitteilen, weil ein solches System keine UserIDs verwendet, aus denen der Netzname abgeleitet werden könnte. Der übrige Ablauf kann unverändert übernommen werden, da bei der Authentisierung keinerlei Eigenschaften oder Namen der Maschine eingehen, auf der der Client läuft.

5 Wie sicher ist Secure RPC?

5.1 Was für Stärken hat Secure RPC?

Die Authentisierung findet vollständig im Server statt. Der Server muß nicht wie beim „Trusted Port“-Konzept darauf vertrauen, daß die Daten, die der Client ihm übermittelt (zum Beispiel die UserID), korrekt sind. Dadurch ist es prinzipiell auch möglich, Clients mit nur geringen bis gar keinen eigenen Sicherheitsmechanismen wie etwa PCs an authentisierten Diensten teilhaben zu lassen.

Es werden keine geheimen Daten, die der Authentisierung dienen, im Klartext über das Netzwerk übertragen. Die Daten sind entweder verschlüsselt – wie der geheime Schlüssel, wenn er von der zentralen Schlüsselliste abgefragt wird – oder werden nur implizit übertragen – wie der gemeinsame Schlüssel K, der nur zur Verschlüsselung der Uhrzeit verwendet wird. Folglich kann ein Angreifer, der uneingeschränkten Zugriff auf das Netzwerk hat, keine Informationen „abhören,“ die er später verwenden könnte, um sich unberechtigterweise zu authentisieren.

Es ist kein zentraler Authentisierungsserver notwendig, dem Clients und Server eines authentisierten Dienstes vertrauen müßten.

5.2 Was für Schwächen hat Secure RPC?

Die größte Schwäche von Secure RPC ist die verwendete Schlüssellänge: P und damit D und E sind auf eine Größenordnung von 2^{192}, also ihre Länge auf 192 Bit festgelegt. Bei Schlüsseln dieser Größenordnung kann man durch ein Siebverfahren den zu einem öffentlichen Schlüssel gehörenden geheimen berechnen. Der Rechenaufwand beträgt für jeden Wert von P, den man berücksichtigt, einmalig einige Tage Rechenzeit auf einem Pool von wenigen Workstations. Mit den in diesem Schritt vorberechneten Daten dauert das Finden eines einzelnen geheimen Schlüssels einige Stunden auf einer einzelnen Workstation[4]. Die Vorberechnung muß für Secure RPC nur ein einziges Mal durchgeführt werden, da der Wert von P konstant ist. Dieses Problem könnte gelöst werden, indem die Schlüssellänge erhöht würde, zum Beispiel auf die bei PGP – einem Public Domain Dateiverschlüsselungs-Programm – üblichen 1024 Bit (PGP verwendet zwar nicht das Diffie-Hellman-Verfahren, aber ein Verfahren, dessen geheime Schlüssel etwa mit demselben Aufwand zu finden sind). Der Rechenaufwand bei der Bestimmung des gemeinsamen Schlüssels K steigt bei einer Verdopplung der Schlüssellänge etwa um den Faktor 8, während der Aufwand, einen geheimen Schlüssel zu knacken, fast exponentiell zunimmt. Dadurch erreicht man gegenüber einem Angreifer bei einer Verlängerung des Schlüssels schnell einen uneinholbaren Vorsprung. Der Anstieg des Rechenaufwands fällt auch nicht sehr stark ins Gewicht, da der gemeinsame Schlüssel nur bei der ersten Anfrage erzeugt werden muß und dann durch den `keyserv`-Dämon aufbewahrt werden kann.

Der Benutzer hat bisher keine Möglichkeit, nach einem `rlogin` dem `keyserv`-Dämon auf der Zielmaschine seinen geheimen Schlüssel zu übergeben, ohne ein `keylogin` auszuführen. Dafür muß er aber sein Login-Paßwort über die unverschlüsselte `rlogin`-Verbindung eingeben und das Paßwort wird wieder im Klartext über das Netz übertragen, was ja eigentlich vermieden werden sollte. Es wäre wünschenswert, einen `rkeylogin`-Dienst zu haben, der den geheimen Schlüssel über eine authentisierte und verschlüsselte RPC-Verbindung an die Zielmaschine sendet. Auch Einweg-Paßwort-Systeme wie S/Key können nicht implizit `keylogin` aufrufen und führen zu derselben Schwäche.

Wie bei einem expliziten `keylogin` wird auch beim Einloggen über `telnet/login` das Paßwort im Klartext übertragen. Es fehlt bisher eine „secure“-Option für `telnet`, die bewirkt, daß – statt UserID und Paßwort zu übertragen – über Secure RPC authentisiert wird.

Die Datei `/etc/.rootkey` enthält den geheimen Schlüssel des Superusers im Klartext (siehe Abschnitt 4.4). Dies ist zwar notwendig, will man nicht bei jedem Reboot das Superuser-Passwort eingeben müssen, aber damit wird auch ein zusätzliches Risiko geschaffen. Es werden nämlich von Zeit zu Zeit Sicherheitslücken entdeckt, die dazu führen, daß man zwar keine Superuser-Privilegien erlangen kann, aber gegen Lesen geschützte Dateien trotzdem lesen kann. Das bedeutet dann in diesem Fall, daß man sowohl gegenüber Servern auf dieser Maschine jede Authentisierung fälschen als auch sich gegenüber anderen Maschinen und Benutzern als Superuser beziehungsweise Server auf dieser Maschine ausgeben kann. Dadurch erhöht sich also die Wahrscheinlichkeit einer „gefährlichen“ Sicherheitslücke.

Da das Login-Paßwort des Benutzers zum Verschlüsseln des geheimen Schlüssels verwendet wird, ist der Benutzer gezwungen, auf allen Maschinen im Netzwerk dasselbe Paßwort zu verwenden. Allerdings erspart das System so dem Benutzer, sich noch ein weiteres Paßwort ausdenken, merken und hin und wieder ändern zu müssen, was vermutlich dazu führen würde, daß der Benutzer sich mit diesem zusätzlichen Paßwort nicht viel Mühe gibt. Deshalb ist dieser Mangel vermutlich nicht sehr gravierend. Es ist optional auch möglich, verschiedene Paßwörter zu verwenden. Dies ist aber nicht die Standardeinstellung und macht es erforderlich, daß der Benutzer jeweils von Hand `keylogin` aufruft, siehe auch [3].

6 Fazit

Secure RPC löst einen großen Teil der Probleme anderer Authentisierungsverfahren. Es trägt trotz seiner Schwächen zu sichererer Client-Server-Kommunikation bei.

Man sollte aber nicht ein Allheilmittel erwarten, das auf einen Schlag eine sichere, funktionelle und fehlerfreie Lösung des Authentisierungsproblems liefert.

Deshalb muß man wie bei jedem Sicherheitssystem überprüfen, ob es in der eigenen Netzwerkinstallation einen Gewinn an Sicherheit bringt, der den zusätzlichen Wartungsaufwand und die neu entstehenden Probleme überwiegt.

Secure RPC ist von der Konzeption her den traditionell von Unix verwendeten Authentisierungsmechanismen deutlich überlegen und wird durch seine Verwendung in Entwicklungen wie DCE in nächster Zeit noch stark an Bedeutung gewinnen.

Es bleibt zu hoffen, daß Sun in naher Zukunft das Hauptproblem behebt und die Schlüssellänge auf den aktuellen Stand der Technik bringt. Spätestens dann wird es ein System sein, dessen Benutzung man auf jeden Fall in Erwägung ziehen sollte.

Vor allem aber sollte man sich nicht darauf verlassen, daß durch die Installation von Secure RPC alle Sicherheitsprobleme behoben sind. Kryptographische Authentifikation kann immer nur ein Teilaspekt einer Sicherheitskonzeption sein.

Glossar

DCE: „Digital Computing Environment,“ ist ein Ansatz, Netzwerkdienste wie File Service, Name Service, Time Service und ähnliche plattformübergreifend zu integrieren.

DES: „Data Encryption Standard,“ ein Verschlüsselungsverfahren, das Daten mit einem geheimen Schlüssel verschlüsselt und in einem anderen Modus mit demselben Schlüssel wieder entschlüsselt.

Domain: ein Pool von Maschinen, die gemeinsam NIS- beziehungsweise NIS+-Daten verwenden.

`keyserv`: ein Dämon, der für die Benutzer auf der lokalen Maschine ihre geheimen Schlüssel im Klartext bereithält, um sie zur Authentisierung zu verwenden, siehe auch Abschnitt 4.4.

Klartext: Im kryptographischen Sinne hat „im Klartext“ die Bedeutung „unverschlüsselt“. Auch binäre oder auf für jeden umkehrbare Weise codierte Daten (zum Beispiel mittels `uuencode`) gelten in diesem Sinne als Klartext.

Netzname: ein Name, der einen Benutzer beziehungsweise eine Maschine im Netz eindeutig identifiziert, zum Beispiel `unix.167@in.tu-clausthal.de` für einen Benutzer oder `zaphod.in.tu-clausthal.de@in.tu-clausthal.de` für eine Maschine, siehe auch Abschnitt 4.2.

NFS: „Network File System,“ ein Dienst, der lokale Filesysteme transparent anderen Maschinen zur Verfügung zu stellen.

NIS: „Network Information System“ früher „Yellow Pages,“ ein System zur zentralen Haltung von Systemdateien. NIS ist aus der Notwendigkeit entstanden, Dateien wie `/etc/passwd, /etc/group` und `/etc/hosts` in Pools von vielen Maschinen konsistent zu halten.

NIS+: Nachfolger von NIS mit hierarchischer Domain-Struktur und der Möglichkeit, Subdomains an andere Server zu deligieren und Zugriffsrechte für die einzelnen Tabellen zu vergeben.

PGP: „Pretty Good Privacy,“ ein von Philip Zimmermann entwickeltes Verschlüsselungsprogramm.

RPC: „Remote Procedure Call,“ ein Konzept zum einfachen Programmieren von Netzwerkdiensten mit Client-Server-Architektur, siehe Abschnitt 4.1.

S/Key: ein Einweg-Paßwort-System von Phil Karn, Neil M. Haller und John S. Walden. Bei diesem System wird für den Benutzer im voraus eine Liste von Paßwörtern erzeugt, die er jeweils nur einmal verwenden kann. Dadurch kann ein Angreifer, der ein Paßwort mitbekommt, während sich der Benutzer einloggt, dieses später nicht selbst zum Einloggen verwenden.

Superuser: das Unix-`root`-Account, Secure RPC verwendet die Server, die unter dem `root`-Account laufen, und die Maschine selbst synonym und gibt beiden denselben Netznamen.

Literaturverzeichnis

[1] W. R. Stevens: *Programmierung von UNIX-Netzen,* Kapitel 9, München, Wien: Hanser, 1992

[2] J. G. Steiner, C. Neuman und J. I. Schiller: *Kerberos: An Authentication Service for Open Network Systems,* Proceedings of the 1988 Winter Usenix Conference, Dallas, Tex., pp. 191-202, 1988

[3] Rick Ramsey: *All About Administering NIS+,* SunSoft Press, 1993

[4] B. A. LaMacchia und A. M. Odlyzko: *Computation of Discrete Logarithms in Prime Fields,* Designs, Codes and Cryptography, 1, 1991

[5] Dieses Dokument selbst ist auch über WWW verfügbar:
`http://www.in.tu-clausthal.de/~injc/secure.rpc/`