PowerKey 1/94

Inhalt

Editorial

Liebe Leserin, lieber Leser

In der Redaktion des ***PowerKey*** hat sich ein Wechsel vollzogen. Adriano Gabaglio, der bisherige Redaktor, hat sein Amt aus beruflichen Gründen nach zwei Jahren niedergelegt. Von dieser Ausgabe an, übernehme ich sein Amt und führe es in seiner Tradition weiter. Es ist eine hohe Messlatte die er mir gesetzt hat, aber ich bin sicher, dass es mir (mit Eurer Hilfe) gelingen wird, eine gleichbleibende Qualität des ***PowerKey*** zu gewährleisten. Ich möchte Adriano an dieser Stelle im Namen aller Vorstandsmitglieder ganz herzlich für seine Arbeit danken!

Trotz neuer Redaktion ist der Fortbestand des ***PowerKey*** allerdings noch lange nicht gesichert, denn es ist ganz klar nicht die Aufgabe des Redaktors die ganze Zeitung selbst zu schreiben. Gefragt ist jeder – jeder von Euch kann dazu beitragen, dass das ***PowerKey*** auch in Zukunft regelmässig und in vollem Umfang erscheint. Wer irgendwo ein interessantes Programm, ein Buch, eine günstige CD, eine allen dienende Information oder gar einen Bug entdeckt hat, der sollte sich an seinen Computer setzen und ein paar Zeilen darüber schreiben. Keiner erwartet schriftstellerische Meisterleistungen – was Uns am Herzen liegt sind ganz einfach interessante Informationen von Usern für User.

Der Fortbestand des ***PowerKey*** ist auch dadurch gefährdet, dass sich bis zur Drucksetzung dieses Editorials immer noch niemand für das Amt des **Verlegers** gemeldet hat! Intressenten sollen sich bitte so schnell wie möglich beim Vorstand melden.

An der diesjährigen GV wurde darüber beraten, wieviele Ausgaben des ***PowerKey*** pro Jahr erscheinen sollten. Es wurde schnell klar, dass diese Frage nicht in Form eines zwingenden Beschlusses entschieden werden kann, denn jede Ausgabe für sich ist direkt von der Anzahl der Artikel abhängig, die bis zum Redaktionsschluss verfügbar sind. Es ist im Sinne aller Leserinnen und Leser, dass Ihr Eure Artikel frühzeitig einreicht und dass das Datum des Redaktionsschlusses verbindlich eingehalten wird.
Angestrebt werden vorerst einmal vier Ausgaben, jeweils auf Ende März, Juni, September und Dezember. Sollten sich Artikel für mehr als vier Ausgaben anhäufen, sind wir flexibel genug, um weitere Ausgaben zu produzieren.

Neben den festen Ausgaben des ***PowerKey*** werden neu regelmässig aktuelle Kurzinformationen verfasst und als eine Art Info-Postkarte direkt an die Mitglieder verschickt.

Nun aber zum Inhalt: In der vorliegenden Ausgabe von ***PowerKey*** gibt es neben einem aktuellen Bericht zur CeBIT94, Schwergewichte in Richtung Backup- und Kryptosysteme.

Ich wünsche allen viel Vergnügen mit dieser Ausgabe von ***PowerKey***.

Dominik Moser, Redaktor

Aktuell

NEXTSTEP an der CeBIT94

Ein Kurzbericht von der CeBIT94, die vom 16. bis zum 23. März in Hannover stattfand.

NeXT

NeXT Computer Deutschland war mit einem 100m^2 Stand in Halle 2 vertreten. Ganz in schwarz, aber nur mit weissen Maschinen bestückt sah das ganz nett aus. Gezeigt wurde eine Pre-Beta des PA-RISC Ports, und wieder mal *NEXTIME*. Beide Produkte werden ab July lieferbar sein. Die PA-RISC Version wird übrigens gleichviel kosten wie die Intel Version. PDOs: HP/UX läuft, Solaris und DG/UX sollen ab Mai erhältlich sein und OSF/1 wird später dazukommen.
Im Kino vor dem Stand wurden abwechslungsweise Steve-Videos und Live-Demos gefahren. Im grossen und ganzen scheint's mit NeXT aufwärts zu gehen.

Sun

Der Sun-Stand in Halle 1 war klar dominiert von den neuen *Voyagers*. Echt heisse Kiste! Ein Sparc-Portable mit ColorLCD (Aktiv TFT) 1024×768 oder MonoLCD 1152×900. Wenn da drauf mal NEXTSTEP läuft. ich brauch's!!!
Marketingmässig war das WM-Sponsoring sehr gut sichtbar, ansonsten nicht viel neues. Zum Thema NEXTSTEP wird vorallem auf OpenStep verwiesen, scheint für Sun-Europe aber doch noch etwas früh zu sein. Sun kämpft immer noch mit ihren Altlasten: welches Fenstersystem darf's denn sein?!

hp

Am hp-Stand lief eine *Gecko* 712/60 mit einer NEXTSTEP Beta (Bootfahig, mit SCSI-Driver). Ich kann nur sagen: endlich wieder adäquate Hardware für NEXTSTEP. Colorrecovery ist erstaunlich gut. Die Abbildung Truecolor auf 8-Bit CLUT ist wohl das erstaunlichste, das mir je untergekommen ist. Die Preise für die 715'er Serie wurde nach unten angepasst (20–40%) und bei den 735'ern gibt's als neuestes Highend-Modell eine 125MHz Version.

Samsung/NetConsult

Bei Samsung gab's einen NEXTSTEP-PC der von NetConsult betrieben wurde: Sie konnten eine eigene Archivierungssoftware vorführen (*Archiv2000*, lieferbar auf CD).

NEC / Interpersonal-Computing

Bei NEC wurde der *Versa* (Laptop) mit NEXTSTEP gezeigt. Immer noch 2Bit, 640×480. Von diesem Laptop gibt's jetzt auch 'ne nette DX-4-Version mit 75MHz :-)
Zudem konnte man noch den Videograbber *Screenmachine* bestaunen.

tms

OneVision scheint (nun in der Version 1.99!) jetzt doch stabil zu laufen. Auch die Modulladezeiten sind erträglich.

d'ART

d'ART war vorallem am NeXT-Stand als Unterstützung eingesetzt. Sei zeigten nichts neues (Abgesehen von *VIVA*, das jetzt mit *Interbase* auch in reinen PC-Netzen läuft).

Borland

Am Borland-Stand zeigte ein d'ART-MA, was mit *DB-Kit* und *Interbase* so alles möglich ist (Standard-DBKit-Demo).

George Fankhauser

Die Kryptographie erobert den Alltag

In den USA erregt zur Zeit eine Diskussion über Kryptosysteme und die Rolle der National Security Agency (NSA) die Gemüter. Es geht darin um die Chiffrierung des privaten und geschäftlichen Telefon- und Datenverkehrs und um das «Vorrecht» der NSA über sämtliche Schlüssel zur Dechiffrierung zu verfügen, damit das «Abhören» auch in Zukunft noch möglich sein soll.

Die Kryptographie gewinnt als Werkzeug in der elektronischen Kommunikation zunehmend an Bedeutung. Wie in der geschichtlichen Entwicklung dieser Lehre vom Verschlüsseln und Entschlüsseln oft genug zu sehen ist, gedeihen Geheimschriften, Geheimsprachen und die ungezählten Ideen, Wissen und Informationen aller Art vor einer vermeindlichen oder realen Ausspähung zu schützen, besonders gut auf dem Boden des Krieges, des Wettbewerbs, überhaupt im Umfeld einer wie auch immer begründeten Abschottung nach aussen. So erinnert das sich bedingende Hinaufschaukeln von Chiffrieren und Kodebrechen an die gegenseitig abhängige Entwicklung von Angriffs- und Verteidigungswaffen.

Die Vorsilbe *krypto* (griech. kryptos – verborgen, versteckt) deutet an, worum es geht: um Methoden zum Verbergen von Informationen. Entsprechend dem Erfindungsreichtum menschlichen Geistes haben sich auch die Versuche gestaltet, entweder die Inhalte von Nachrichten oder diese selbst geheimzuhalten. Die Verfahren reichen von der unsichtbaren Tinte bis zum speziellen Verschlüsselungschip.

Dennoch ist es keinesfalls paradox, wenn am Ende der siebziger Jahre vorgeschlagen wurde, als Konsequenz der sog. *Public-Key*-Verfahren sogar das Geheimste selbst, die Schlüssel, zu enthüllen und sie den Augen einer breiten Öffentlichkeit preiszugeben. Menschliche Kombinationsgabe und Intuition scheinen seither nicht mehr gefragt, da, wie es aussieht, allein die geballte Kraft riesiger Rechenanlagen die heute gültigen Verschlüsselungen knacken könnte. Zugleich haben sich die Schwerpunkte der Kryptologie verschoben: vom militärischen zum privat-wirtschaftlichen Bereich. Im Zeitalter der Verarbeitung von Informationen wird neben einer Sicherung von Daten auch der Schutz vor unbefugten Zugriffen zunehmend bedeutsamer. Dass Passwörter keineswegs ausreichen, ist vielfach aufgezeigt worden. Was liegt näher, als sich nicht nur auf Abwehrmethoden gegen Unberechtigte zu konzentrieren, sondern die Daten selbst zu schützen, diese über Kryptosysteme so zu chiffrieren, dass sie ohne Kenntnis einer Schlüsselfunktion wertlos sind?

Unverschlossene Briefe

Für die meisten von uns ist es selbstverständlich, dass man einen (Papier-)Brief persönlich unterzeichnet und in einem Couvert verschliesst. Mit unsere Unterschrift besiegeln wir Verträge, heben Geld am Bankschalter ab, bestätigen den Empfang von Postsendungen und kennzeichnen Ausweise und Kreditkarten.
Nichts dergleichen findet man in der Computertechnik: Elektronische Post wandert in den meisten Fällen ungezeichnet und ungesichert über die Netzwerke. Wer elektronisch eine Mitteilung erhält, geht in der Regel davon aus, dass sie auch vom angegebenen Absender stammt. Dies obwohl heute etwa im Internet bereits mit wenig Fachwissen Absenderadressen gefälscht werden können. Gedankenlos erfolgt auch der Umgang mit den Mitteilungen selbst. Ihr Inhalt wird normalerweise offen und unverschlüsselt von einem Computer zum andern gesandt. Oftmals wandern sogar wichtige Geschäftsinformationen offen über Computernetzwerke, da sich die Benutzer der potentiellen Gefahr einfach nicht bewusst sind.

Leichtes Spiel für Codeknacker

Viele Anwender glauben auch, ihre Dokumente mit den Passwort- und Verschlüsselungsfunktionen ihrer Textverarbeitung, Tabellenkalkulation oder Datenbank vor neugierigen Blicken sicher geschützt zu haben. Im Ernstfall taugen solche und andere Schutzvorrichtungen

auch vieler kommerzieller Verschlüsselungssysteme kaum etwas. Wert ist dieser «Schutz» ganze 185 Dollar. Für diesen Betrag bietet die Firma Access Data in Orem, Utah, ein Programm an, das die Verschlüsselungssysteme populärer Softwarepakete wie Excel oder Wordperfect knackt — und zwar nicht durch blosses Erraten, sondern durch echte Kryptoanalyse. Und das geschieht so schnell, dass eine Verzögerungsschleife eingebaut werden musste, damit niemand merkt, wie einfach das geht!

Von der ENIGMA zum Data Encryption-Standard

Am Beginn der Geschichte elektronischer Rechenmaschinen steht ein Koloss aus Elektronenröhren, zum Zweck der Entschlüsselung militärischer Nachrichten konzipiert und konstruiert: COLOSSUS. Prototyp eines Grossrechners, 1942 für den britischen Geheimdienst gebaut, um die Verschlüsselungsparameter der elektromechanisch arbeitenden deutschen Chiffriermaschine ENIGMA (nach griech. ainigma – Rätsel) aufzudecken. Trotz anfänglicher Schwierigkeiten gelang es schliesslich den Kode zu knacken, zum einen, weil den Briten über verschiedene Quellen einiges über die ENIGMA bekannt war, zum andern half die analytische Fähigkeit der eingesetzten Mathematiker, darunter Alan Turing, die zum ersten Mal auf die Rechenkraft einer elektronischen Rechenmaschine zurückgreifen konnten. Wie anders auch als mit Hilfe von Rechenautomaten hätte die immense Zahl von Verschlüsselungsmöglichkeiten durchprobiert werden können.

Machen wir einen Sprung ins Jahr 1977, das Jahr, in dem erstmals eine Verschlüsselungsmethode als Standard anerkannt wurde: der *Data Encryption Standard* (DES). Die ENIGMA hatte längst ausgedient, ihre Aufgabe sollen Computerprogramme oder in spezielle Chips eingelagerte Verschlüsselungslogiken übernehmen. Gleichwohl hat sich das Prinzip der Verschlüsselung nicht grundlegend verändert, die Arbeitsweise des Algorithmus' jedoch, dem DES zugrunde liegt, ist weitaus komplizierter geworden. Klartext und Schlüssel werden blockweise in immer neuen Variationen durcheinandergeschüttelt, vertauscht und verknüpft, so dass am Ende auch der geschickteste Analysator verzweifelt aufgegeben haben sollte. Jedenfalls hofften dies die Entwickler, vornehmlich Programmierer der Firma IBM.
Will man den DES-Kode brechen, bedarf es einer Überprüfung aller möglichen Schlüssel. Das wären 2 hoch 56 verschiedene Konstellationen. In einer 1977 veröffentlichten Studie entwarfen die beiden Standford-Wissenschaftler Whitfield Diffie und Martin E. Hellman eine theoretische Maschine, die imstande wäre, einen Klartext über alle Schlüssel zu chiffrieren; und das innerhalb eines Tages. Auch wenn sich eine solche Maschine zur Zeit nicht realisieren lässt, sind doch zumindest Wege aufgezeichnet, die Festigkeit der DES-Methode zu erschüttern. Noch ein Punkt bleibt festzuhalten. Es fehlt eine schlüssige Begründung für den ausdrücklichen Wunsch des *National Bureau of Standards* (NBS) auf Verkürzung der Schlüssellänge auf 64 bzw. 56 Bits, schliesslich hatte IBM im Entwurf 128 Bit vorgeschlagen. Vielleicht, so meinen Kritiker, hat das NBS über die Schlüssellänge eine Hintertür geschaffen, um die DES-Methode zu knacken.

Konventionelle Kryptosysteme

Mit Kryptographie verbindet man auch heute noch aufwendige, teure und unhandliche Verschlüsselungslösungen, die bestenfalls in Banken oder anderen Orten mit empfindlichen Daten zum Einsatz kommen. Ein Grund dafür mag sein, dass vor allem die als *symmetrische* oder *Single Key* bezeichenten Verfahren bekannt sind.
Bei solchen Verfahren wird eine Botschaft oder eine Datei mit dem gleichen Code entschlüsselt, mit dem sie bereits verschlüsselt worden war. Alle beteiligten Personen benutzen denselben Schlüssel. Er muss dementsprechend geheim bleiben, was nur bei einer kleinen, vertrauten Gruppe von Personen funktioniert. Solche, auch *Private Key* genannte Verschlüsselungsverfahren eignen sich deshalb nur dort, wo verschlüsselte Daten in einem kleinen verlässlichen Personenkreis benutzt oder in Hardware fest installiert werden. Das Chiffrieren von Datenleitungen oder Festplatten sind zwei Beispiele dafür.

Wie alle konventionellen Kryptosysteme verliert auch der DES für einen Anwender augenblicklich an Wirksamkeit, wenn der geheime Schlüssel in die falschen Hände gerät.

Raffinierte Public-Key-Systeme

Von Diffie und Hellman, den Kritikern des DES, stammt auch die Idee, ein Kryptosystem zu schaffen, in dem der leidige geheime Schlüsselaustausch entfällt [1]. Mehr noch, sie führten sogenannte *Public-Keys* in die Diskussion ein. Das Prinzip ist einfach: Zum Chiffrieren wird ein anderer Schlüssel benutzt als zum Dechiffrieren. Dem Absender einer Mitteilung braucht nur der Code zum Verschlüsseln mitgeteilt werden. Das ist gefahrlos, denn mit jenem öffentlichen Schlüssel kann eine verschlüsselte Meldung nicht wieder dechiffriert werden. Das ist nur mit dem zweiten, geheimen Schlüssel möglich!

Der Vorteil von Public-Key-Verfahren liegt auf der Hand. Soll jemand eine verschlüsselte Botschaft erhalten, so muss dem Absender nicht mehr vertraut werden, da er mit dem Chiffriercode nur verschlüsseln kann. Der öffentliche Teil des Schlüssels kann also frei publiziert und zudem beliebig oft benutzt werden; geheim bleibt lediglich der andere, private Teil. Denkbar wäre zum Beispiel ein öffentliches, dem Telefonbuch ähnliches elektronisches Schlüsselverzeichnis. Der Absender muss darin bloss noch den Schlüssel des Empfängers heraussuchen und kann ihm eine chiffrierte Mitteilung zukommen lassen, ohne mit ihm vorher geheime Informationen über die zu verwendenden Schlüssel auszutauschen.

Diffies und Hellmans Vorschläge in ein praktikables Verfahren umgesetzt haben 1978 drei Wissenschaftler des MIT *Laboratory for Computer Science* in Cambridge, nach deren Nachnamen-Initialen auch die Methode benannt wird: *RSA-System* (von Rivest, Shamir und Adleman). Eine relativ einfache Überlegung liegt dem Verfahren zugrunde: Ohne Probleme lässt sich aus zwei grossen Primzahlen p und q ein Produkt n bilden, andererseits gibt es kein Verfahren, auf direktem Wege aus n die beiden Primfaktoren zurückzugewinnen. Um die notwendige Sicherheit zu gewährleisten, müssen die beiden Primzahlen mindestens in der Grössenordnung von 100 Dezimalstellen gewählt werden. Zur Verdeutlichung hier ein Beispiel, das aus dem Paper der drei RSA-Autoren stammt [2]. Die mathematischen Hintergründe werden bei der Wiedergabe vernachlässigt, nur der Vorgang des Ver- und Entschlüsselns soll erläutert werden. Die verwendeten Primzahlen sind unrealistisch klein gehalten, sie dienen nur dazu, den Ablauf nachvollziehbar zu machen.
Gegeben seien die Primzahlen

$p = 47, q = 59$

und das Produkt

$n = p \times q = 2773.$

Man berechnet dann die Eulersche Funktion *phi(n)*, die angibt, wieviele Zahlen unterhalb von n keinen gemeinsamen Teiler mit n haben. Der Rechenvorgang ist in diesem Fall einfach, da *phi(n)* gleich dem Produkt der Primfaktoren von n ist, wobei jeder Primfaktor um 1 vermindert wird. Da n per Definition aus den Primfaktoren p und q besteht, gewinnt man *phi(n)* nach der Formel

$phi(n) = (p-1) \times (q-1) = 2668.$

Im nächsten Schritt wird der Verschlüsselungsexponent e bestimmt. Dies ist eine beliebige Zahl zwischen 1 und n, für die gilt, dass sie zu *phi(n)* keinen gemeinsamen Teiler (ausser 1) besitzt: z.B. $e = 17$. Zuletzt braucht man die «magische Zahl» d, die die Gleichung

$e \times d \bmod phi(n) = 1$

erfüllt: also $d = 157$.

Zusammengefasst werden die folgenden Zahlen verwendet:

$p = 47$	(geheim)
$q = 59$	(geheim)
$n = 2773$	(öffentlich)
$phi(n) = 2668$	(geheim)
$e = 17$	(öffentlich)
$d = 157$	(geheim)

Jeder kann nun mit dem öffentlich bekanntgegebenen Exponenten e seine Botschaften chiffrieren. Dazu wird der Klartext in Zahlen umgewandelt, im nachfolgenden

Beispiel erhält das Leerzeichen die Zahl 00, A die Zahl 01, B = 02 usw.
Beispiel:

Klartext:	«MELDUNG»
Klartext in Zahlen:	13 05 12 04 21 14 07

Da n zwischen 10^3 und 10^4 liegt, können jeweils zwei Buchstaben zu einem Block der Länge 4 zusammengefasst werden. Gegebenenfalls müssen die Blöcke mit Leerzeichen (Nullen) aufgefüllt werden.

Klartext in Blöcken: 1305 1204 2114 0700

Die Verschlüsselungsfunktion *Vf* hat nun die Form

Vf = Klartext hoch e mod n

also

$1305^{17} \bmod 2773 = 0813$

$1204^{17} \bmod 2773 = 0232$ usw.

Der verschlüsselte Text erhält folgende Zahlenwerte:

0813 0232 1644 0700

Nur wer die «magische Zahl» d kennt, vermag den verschlüsselten Text mit folgender Funktion zu dechiffrieren:

Ef = Schlüsseltext hoch d mod n

also

$813^{157} \bmod 2773 = 1305$

$232^{157} \bmod 2773 = 1204$ usw.

Bereits an diesem einfachen Beispiel ist zu ersehen, dass im Verlauf der Rechnung schnell sehr grosse Zahlen entstehen, die nur mit speziellen Computerprogrammen (Stichwort: *stochastische Algorithmen*) ohne Überlauf zu bewältigen sind.

Prinzipiell ist der RSA-Kode einfach zu brechen, man muss nur die beiden Primfaktoren ermitteln, die der öffentlichen Zahl n zugrunde liegen. Allerdings – der schnellste bislang bekannte Algorithmus zum Zerlegen von Zahlen in ihre Primfaktoren benötigt für ein n mit 200 Dezimalstellen etwa 4×10^9 Jahre bei 1 Million Operationen pro Sekunde. Noch scheint das RSA-Verfahren das bislang einzige der Pubilc-Key-Systeme zu sein, das noch nicht zu Fall gebracht werden konnte.

Elektronische Unterschriften

Falls in Zukunft die elektronische Post den heutigen Papierbriefverkehr ablösen will, müssen von Anfang an zwei Voraussetzungen geschaffen werden. Erstens müssen elektronische Nachrichten in dem Sinne «persönlich» sein, als dass nur der rechtmässige Empfänger den Inhalt einer Botschaft zu sehen bekommt. Wie wir gesehen haben, wird dieser Punkt durch die Verwendung von Public-Key-Systemen gewährleistet, da nur der rechtmässige Empfänger über die entsprechende Entschlüsselungsfunktion verfügt. Zweitens müssen elektronische Nachrichten mit einer «Unterschrift» versehen werden können, damit sich der Empfänger sicher sein kann, dass die Botschaft direkt vom ursprünglichen Absender stammt. Wie wir gleich sehen werden, ist es mit Hilfe des gleichen Public-Key-Systems möglich elektronische Unterschriften zu generieren. Zuerst aber noch ein paar allgemeine Betrachtungen.

Eine elektronische Unterschrift muss sowohl von der Nachricht selbst als auch vom Absender der Nachricht abhängig sein. Wo dies nicht der Fall ist, könnte der Inhalt der Nachricht auf der Strecke zwischen Absender und Empfänger verändert oder sogar komplett ausgetauscht werden. Andererseits könnte eine elektronische Unterschrift an beliebige Nachrichten angehängt werden, da es nicht möglich ist elektronisches «cut and paste» nachzuweisen.

Um elektronische Unterschriften realisieren zu können braucht unser Public-Key-Kryptosystem sog. *trap-door one-way permutation.* Dahinter verbirgt sich folgendes Prinzip: Wenn wir auf unseren Klartext K zuerst die Entschlüsselungsfunktion *Ef* anwenden und anschliessend die Verschlüsselungsfunktion *Vf*, dann erhalten wir wieder den Klartext!
Also

$Vf(Ef(K)) = K$

Der Absender erzeugt zuerst mit Hilfe seiner (geheimen) Entschlüsselungsfunktion Ef_A eine «Unterschrift»

U des Klartexts:

$U = Ef_A(K)$

Diese verschlüsselt er nun mit der öffentlichen Verschlüsselungsfunktion Vf_E des Empfängers

$N = Vf_E(U)$

und erhält auf diese Weise die chiffrierte und unterschriebene Nachricht *N*, die er z.B. seinem Mailserver übergeben kann.

Der Empfänger wendet auf die erhaltene Nachricht zuerst seine persönliche Entschlüsselungsfunktion Ef_E an, was ihm die vom Absender «unterschriebene» Botschaft (aber noch nicht den Klartext) liefert. Den Klartext erhält er erst, wenn er in einem weiteren Schritt den öffentlichen Schlüssel Vf_A des Absenders anwendet. Also

$K = Vf_A(Ef_E(N))$

Das Verfahren lässt sich soweit automatisieren, dass der Mail-Anwender die zahlreichen Ver- und Entschlüsselungsvorgänge gar nicht bemerkt. (Vergleiche dazu den Bericht über das Public-Key-Kryptosystem PGP an andere Stelle in dieser Ausgabe)

Die Archillesferse

Von einem Schlüsselverrat mal abgesehen, sind elektronische Unterschriften gegenüber Papierunterschriften sogar wesentlich sicherer. Ein Nachahmen ist nicht möglich, und der Empfänger kann während der Dechiffrierung nicht nur die Echtheit des Absenders verifizieren, er merkt auch sofort, ob an der Meldung etwas verändert wurde.
Public-Key-Systeme haben freilich auch ihre Nachteile. So sind sie wesentlich langsamer als die symmetrischen Verfahren, bei denen nur ein meist kürzerer Schlüssel verwendet wird. Manche Krypto-Programme bedienen sich deshalb eines Tricks und setzen eine Kombination beider Verfahren ein: Die Datei oder Botschaft an sich wird mit einem zufällig ausgewählten Schlüssel nach einem symmetrischen Verfahren chiffriert. Der verwendete Schlüssel, der mit jeder Botschaft ändert, wird seinerseits mit einem asymmetrischen Algorithmus, also mit dem öffentlichen Schlüssel des Empfängers, chiffriert. Auf diese Weise erhält nur der berechtigte Empfänger den Code zum Dechiffrieren seiner Meldung.
Der eigentliche Schwachpunkt von Pubilc-Key-Kryptosystemen liegt paradoxerweise in ihrer Stärke, den öffentlichen Schlüsseln. Normalerweise führt ein Benutzer auf seinem Computer eine Art elektronischen Schlüsselbund. Darauf sind die öffentlichen Schlüssel aller Personen aufgeführt, von denen er chiffrierte Dokumente erhält. Soll ein solches Dokument dechiffriert werden, so wird die Krypto-Software automatisch nach dem passenden Schlüssel suchen und damit die Botschaft entschlüsseln oder die elektronische Unterschrift prüfen.
Befindet sich am Schlüsselbund ein Schlüssel auf den Namen Peter Muster, mit dem sich das Dokument, das Peter Muster geschickt hat, entschlüsseln lässt, dann besagt die Regel für elektronische Unterschriften, dass dieses Dokument tatsächlich von eben jenem Peter Muster stammt. Das Problem liegt nun aber darin, dass der Benutzer sich dessen nur dann ganz sicher sein kann, wenn er weiss, dass der Schlüssel an seinem Schlüsselbund auch wirklich echt ist, also wirklich von Peter Muster stammt. Jener Schlüssel könnte ihm ja auch von irgend einer anderen Person untergeschoben worden sein, die vorgab, sie sei Peter Muster.

Fazit

Die Frage nach der totalen Sicherheit darf man sich auch im Zusammenhang mit Kryptosystemen nicht stellen. Die Sicherheit des RSA-Systems hängt wesentlich davon ab, ob in der Zukunft schnelle Algorithmen gefunden werden, um Zahlen in Primfaktoren zu zerlegen. Die Frage müsste schon eher lauten, wieviel Aufwand ist gerechtfertigt, um seine elektronischen Nachrichten vor einem möglichen Lauschangriff zu schützen?

Dominik Moser

Literaturhinweise:

[1] Diffie, Hellman: «New directions in cryptography», *IEEE Trans. Inform. Theory IT-22*, 6 (Nov. 1976), 644–654
[2] Rivest, Shamir, Adleman: «A Method for Obtaining Digital Signatures and Public-Key Cryptosystems», *Communications of the ACM*, Heft 21/1978, S. 120–126

Veranstaltungen

NiCE-Meetings

Wann: jeden zweiten Dienstag im Monat
Beginn 19.00 Uhr (bis ca. 21.30)
Wo: ETH Zürich, Hauptgebäude
Raum HG F5
Adresse: Rämistr. 101
Tram 6/9/10, Haltestelle ETH/Uni'spital
Themen: Immer aktuell! Werden nach Möglichkeit im ***PowerKey*** bekanntgegeben. Beiträge von Mitgliedern sind jederzeit willkommen.

talk & copy

Wann: jeden vierten Dienstag im Monat
Beginn 19.00 Uhr
Wo: Informatik-Gebäude (IFW) der ETH
Raum IFW A44
Adresse: Haldeneggsteig 4/Weinbergstr.
Tram 6/7/10/15, Haltestelle Haldenegg
Themen: Hier haben Mitglieder die Möglichkeit, Fragen zu stellen, Erfahrungen auszutauschen sowie die neuste Software aus unserem grossen Archiv zu kopieren. Bring doch einfach Deine Disketten oder besser Deine Festplatte mit! Die NiCE besitzt (fast) alle dazu erforderlichen SCSI-Kabel und -Terminatoren.

Falls auch Du gerne mal eines userer Meetings mitgestalten möchtest (zB. Vorstellen eines von Dir benutzten oder gar selber entwickelten Programmes usw.), so setze Dich doch bitte mit unserem Präsidenten in Verbindung!

NiCE-Agenda 1994

12. April:	talk & copy
26. April:	talk & copy
10. Mai:	Meeting Themen noch nicht bekannt
24. Mai:	talk & copy
14. Juni:	Meeting Themen noch nicht bekannt
28. Juni:	talk & copy
12. Juli:	Meeting Themen noch nicht bekannt
26. Juli:	talk & copy

Für 1994 ist geplant, besser über die Veranstaltungen zu informieren. Dies könnte, falls gewünscht, auch in schriftlicher Form geschehen. Anregungen nimmt der Vorstand gerne entgegen.

NiCE-Vorstand

Ende 1993 haben einige Vorstandsmitglieder (aus beruflichen Gründen) ihren Rücktritt bekanntgegeben. Um eine Kontinuität des Vereins gewährleisten zu können, suchen wir nun Nachfolger, die in erster Linie das weitere Erscheinen des ***PowerKeys*** ermöglichen und mithelfen, die User Meetings zu betreuen. – Falls Du gerne in unserem Verein mitarbeiten möchtest, so wende Dich bitte an den Vorstand.

The Joy of Backup oder: Hast Du Dein Backup heute schon gemacht?

Jeder weiss, dass man aktuelle Backups haben muss, denn je schneller die Computer werden, desto mehr Schaden können sie in immer kürzerer Zeit anrichten. (Murphy lässt grüssen…)

Aber Hand aufs Herz, wer könnte seinen NeXT, d.h. System und besonders eigene Daten vollständig wiederherstellen nach einem kleinen Unfall (versehentlich gelöschte Files) oder gar nach einem Daten-GAU (Headcrash der Harddisk)?

Eine Harddisk mag trotz Garantie teuer sein, aber wirklich wertvoll sind die Daten, die auf ihr gespeichert werden. Man denke nur an die Arbeitszeit die aufgewendet wurde um sie zu produzieren oder an die Kosten um sie vom Internet runterzuladen! Und trotzdem verzichten viele Leute — auch solche die es eigentlich besser wissen müssten — auf ein regelmässiges Backup.

1. Vorwort

Ich will nicht Horrorgeschichten aus zweiter oder dritter Hand aufwärmen, sondern nur ein Beispiel aus eigener Erfahrung kurz erzählen, wie sich sogar eine grosse Firma (mehrere hundert Millionen Umsatz täglich) angesichts des fast sicheren Daten-GAU's, um eine gute Backupstrategie drückte:
Ich arbeitete vor Jahren temporär bei einer Bank, die ein schlecht eingerichtetes Novell-Netz besass, bei dem sogar die elementarsten Sicherheitsregeln missachtet wurden. Als Beispiel sei zu nennen, dass der Loginname fast überall auch gleich dem Passwort entsprach, oder dass aus Platz- und Kostengründen auf eine Harddiskspiegelung verzichtet wurde. Mehr als 30 Leute arbeiteten mit einem mehr schlecht als recht funktionierenden Datenbankflickwerk auf dem Netz. Dieses System, das man treffend als Katastrophe bezeichnen konnte, stürzte im Monat etwa einmal schwer ab, mit Ausfallzeiten von mehreren Stunden bis zu eineinhalb Tagen (man berechne die Kosten der verlorenen Arbeitszeit, den Zinsausfall, etc.). Manchmal kam es sogar vor, dass Backups fehlerhaft waren und dass man auf ältere zurückgreifen musste. Neben der eigentlichen Ausfallzeit, musste also zudem bereits erledigte Arbeit von einem oder zwei Tagen nochmals gemacht werden.

Doch zurück zum eigentlichen Thema. Welches sind die Gründe um auf ein Backup zu verzichten:

1. Unwissenheit
2. Zeitaufwand
3. Kosten
4. Keine Lust / Vergessen

Ich möchte hier versuchen Punkt 1 zu beseitigen sowie die Punkte 2 bzw. 3 zu minimieren. Aber gegen Punkt 4 ist trotz *cron* noch kein Kraut gewachsen.

2. Einleitung

Natürlich gibt es die NeXT-Dokumentation (Stichworte: *backup*, *dump*, *restore*, *tar*, *cpio* im SysAdmin Bookshelf des Librarian), die im Notfall weiterhilft, aber sie ist etwas schwer verdauchlich und gibt nur sehr allgemeine Empfehlungen.

Ich habe fast alle von NeXT vorgeschlagenen Backup-Methoden ausprobiert. Auf die Netzwerkaspekte von *rdump/rrestore* will ich allerdings hier nicht eingehen. Dabei haben sich einige bemerkenswerte Erkenntnisse offenbart, die eine eindeutige Bevorzugung von *dump*/*restore* gegenüber von *tar*/*cpio* und den trivialen Methoden ergeben!

2.1. Vergleich der Methoden

Die trivialen Methoden (Kopieren mit der Maus im *Workspace.app* oder mit *cp* in einer Shell) sind sehr fehleranfällig! Denn man kann bei grösseren Datenbeständen kaum feststellen ob man im Backup alles berücksichtigt hat. Das gleiche Problem ergibt sich auch bei der Verwaltung verschiedener Backups. Ausserdem ist es ein sehr umständliches Verfahren dar, wenn man grössere Files auf ein paar Disketten verteilen muss! Es ist noch anzumerken, dass *cp* schneller ist, da vor dem Kopieren keine Grössenbestimmung der zu kopierenden Files vorgenommen wird und der Kopierprozess selbst weniger Resourcen verbraucht.

tar und *cpio* besitzen eine Längenbeschränkung bei Pfadangaben auf 100 bzw. 128 Zeichen, was auf einem NeXT zur Folge hat, dass kein Backup des kompletten Systems erstellt werden kann. Die Files mit zu langen Pfadnamen (etwa 3–7%) werden während des Backupvorgangs einfach nicht berücksichtigt. Ausserdem wird man häufig mit Meldungen konfrontiert, die das Backup, insbesondere aber den Restore-Vorgang, zusätzlich komplizieren:

```
tar: private/dev/xxx is not a file.
Not dumped
```

oder

```
tar: wgburri/.TeXview_Pipe is not a file.
Not dumped
```

Wirklich unangenehm ist aber der Umstand, dass die *tar*-Archive vollständig auf einen Datenträger passen müssen. Das heisst mit Disketten kann man allenfalls komprimierte Archive (Endung *.tar.Z* oder *.tar.gz*) zur Softwaredistribution verwenden, aber für Backups sind sie ungeeignet. Um dem ganzen die Krone aufzusetzen, sind die beiden Programme *tar* und *cpio* um mehr als den Faktor 4 langsamer als ein entsprechendes *dump* auf einen Streamer.

Eine letzte Anmerkung:
Ab System Release 3.2 ist auch *gnutar*, dass einige der Schwächen von *tar* ausbügelt, in einer stabilen Version verfügbar. Es sei noch kurz gesagt, dass *gnutar* zwar *tar*-Archive lesen kann aber nicht umgekehrt, da *gnutar* beliebig lange Pfad- bzw. Filenamen unterstützt und deshalb ein anderes Fileformat hat.

Zusammenfassend lässt sich sagen:
- *dump* (auf Band) ist die mit Abstand schnellste Backupmethode
- *dump* sichert ein ganzes Filesystem entweder voll oder inkrementell
- *dump* ist trivial einfach zu benutzen
- *restore* gestaltet sich sowohl für ein komplettes Filesystem wie auch für ein einzelnes File problemlos

2.2 Beschreibung von dump

Aufruf (Details unter: `man dump`):

```
/usr/etc/dump [Kommandos [Argumente] Filesystem]
```

dump erlaubt die Erstellung von Filesystem[1]-Backups auf zwei verschiedene Arten:
1. Volles Backup, d.h. alles im betreffenden Filesystem wird gesichert.
2. Inkrementelles Backup, d.h. nur die seit dem letzten Backup veränderte Files werden berücksichtigt.

Das volle Backup (*Dump der Stufe 0*) ist ein Spezialfall des inkrementellen Backups (*Dump der Stufe 1* bis maximal *9*). Das Konzept ist denkbar einfach über sogenannte Stufen implementiert, die zur Referenz im File */etc/dumpdates* abgespeichert werden.

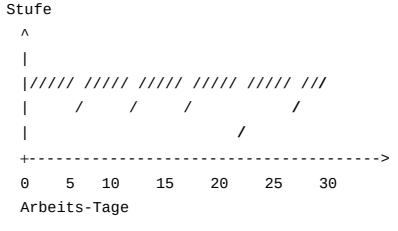
Stufe	Typ	akt. Datum	letztes dump-Datum
0	voll	1.1.1994	1.1.1970
1	inkr. monatl.	1.2.1994	1.1.1994
2	inkr. wöchentl.	7.2.1994	1.2.1994
3	inkr. täglich	9.2.1994	7.2.1994

Im Normalfall besteht das aktuelle Backup aus einem vollständigen und drei inkrementellen Dumps (jeweils der letzte monatliche, wöchentliche und tägliche). Wenn dann am 1.3.94 ein neuer monatlicher Dump erstellt wird, besteht das aktuelle Backup nur noch aus einem vollständigen und aus einem inkrementellen Dump, da sowohl der wöchentliche als auch der tägliche Dump obsolet geworden sind.

1 Ein Filesystem umfasst eine ganze Partition einer Harddisk. Im Normalfall besitzt eine Harddisk nur eine einzige Partition.

Abschliessend sei noch erwähnt, dass nur die Relation 0<1<2<3 wichtig ist und es absolut keine Rolle spielt, ob man für die Stufen unmittelbar aufeinanderfolgende natürliche Zahlen verwendet. Man kann also die Zahlen 1, 2, 3 der inkrementellen Stufen im obigen Beispiel durch ein Trippel 5<7<9 oder 1<5<9 ersetzen, ohne das Backup zu verändern.

Man kann sich das Modell von *dump* Backups auch als Baum oder Graph vorstellen:
4 Stufiges Backupverfahren (Stufen: 0=voll, 1=Monat, 2=Woche, 3=Tag) Das aktuelle Backup ist fett gezeichnet.



Hier sollte das Prinzip auf einen Blick klar werden. Zu einem Beliebigen Zeitpunkt (0) wird mit einem vollen Dump (Stufe 0) begonnen und dann täglich die Veränderungen gesichert (Dump Stufe 3) bis nach einer Woche (5 Arbeitstage) die Änderungen so zahlreich werden, dass der 6. Tagesdump am Wochende als wöchentlicher Dump (Stufe 2) gemacht wird. Die folgenden fünf täglichen Dumps entsprechen wieder den ersten fünf Dumps. Dies geschieht solange, bis nach vier Wochen der 4. Wochendump als monatlicher Dump (Stufe 1) abgespeichert wird, damit die folgenden wöchentlichen Dumps nicht zu umfangreich werden. Schliesslich werden auch die monatlichen Dumps (Stufe 1) sehr umfangreich und man macht einen neuen Stufe-0-Dump und das Ganze beginnt von vorne.

Dies ist die allgemeine Regel, doch nun zu den Ausnahmen und ein paar allgemeinen Tips:

a) Niemals den Datenträger mit dem aktuellen Backup für einen neuen Backup benutzen! Wenn etwas schiefgeht (Stromausfall, Bugs, Tippfehler, etc.), hat man keinen aktuellen Backup mehr. Man benutzt im Minimum Datenträger für 3 volle Backups (1. alter, 2. aktueller, 3. neuer), die man zyklisch (2→1, 3→2, 1→3) überschreibt. Je nach Wichtigkeit der Daten wird der Zyklus von 3 auf 5, 10 oder mehr Backups ausgedehnt. Unter Umständen werden gewisse volle Backups nie gelöscht, um ein Archiv aufzubauen.

b) Bevor grössere oder riskante Änderungen (neuer Release, neue Disk, etc.) am System gemacht werden, sollte auch ein vollständiges Backup durchgeführt werden. Nach einem eventuellen Fehlschlag ist es leichter das System in den Ausgangszustand zurückzusetzen. Ausserdem wird durch dieses zweite Backup die Sicherheit erhöht.

c) Nach grösseren Änderungen wird der tägliche inkrementelle Backup so umfangreich sein, dass man gleich einen neuen, vollen Backup machen kann.

d) Niemals den menschlichen Faktor vergessen! Alle gemachten Backups müssen sauber beschriftet sein und in einem Plan auf Papier vermerkt werden. Denn wenn ein Unglück geschieht, sollte man nicht stundenlang nach dem aktuellen Backup suchen müssen, nur um festzustellen, dass man es soeben bei der verzweifelten Suche durch einen Tippfehler zerstört hat.

e) Warnungen aus der Dokumentation: Das Filesystem muss sauber sein (Überprüfung am besten manuell mit *fsck*), und nach Möglichkeit nicht gemountet, um eine Änderung während des Backups auszuschliessen. Sicherheitsfanatiker machen deshalb einen 'Single-User' Boot, lassen manuell *fsck* laufen und machen dann den Dump. Meiner Erfahrung nach genügt es bei einem unvernetztem NeXT, über die einzige aktive Applikation *Terminal.app* den Dump laufen zu lassen. Das Filesystem ist sowieso ok, da nur eine Person eingeloggt ist und kein weiteres Programm schreibt oder verändert Daten auf der Disk. Vorsicht mit *cron* und gewissen daemons!

f) Zugriffsrechte: Um einen Dump zu machen braucht man '*root*'-Privilegien oder man muss Mitglied der '*operator*'-Group sein (d.h. alle Devices lesen können) und Zugriff auf */etc/dumpdates* haben. Der zweite Ansatz ist sicherer. Man sollte generell darauf achten, wenn immer möglich auf '*root*'-Privilegien zu verzichten, da man damit bei kleinsten Fehlern ungewollt grossen Schaden anrichten kann! (Denn es gibt kein '*Permission denied*' mehr…)

Bugs: Mit dem recompile unter 3.2 hat sich ein kleiner Bug eingeschlichen: Inkrementelle Dumps sind manchmal nicht mehr möglich weil sie wie ein voller Dump alles sichern!
Korrektur: *dump* aus dem Release 3.1 verwenden oder eine eigene Version patchen indem man den Inhalt des Bytes Nr. 10510 (von *dump)* von 246 auf 244 setzt.

2.3. Beschreibung von restore

Aufruf (Details unter: `man restore`):

```
/usr/etc/restore Kommandos [Argumente] [Name]
```

Wenn einmal auf ein Backup zurückgegriffen werden muss, wird mit dem Programm *restore* gearbeitet.
Es können dabei zwei Fälle auftreten:

- Das ganze Filesystem soll komplett wiederhergestellt werden.
- Nur gewisse Files sollen zurückgeholt werden.

Eine volle Restauration beginnt mit dem Restore des neuesten Stufe-0-Dumps, dann folgen, falls vorhanden, der Reihe nach von 1 bis 9 die neuesten inkrementellen Dumps. Konkreter sieht dies vielleicht so aus: Booten ab alternativem Bootdevice, initialisieren der Harddisk, mounten unter */mnt* und *restore*:

```
/usr/etc/disk -i /dev/rsd1a
/usr/etc/mount /dev/sd1a /mnt
cd /mnt
restore r
```

Ein einzelnes File kann im interaktiven Modus gesucht und zurückkopiert werden. Aus Sicherheitsgründen arbeitet man mit Vorteil im Verzeichnis */tmp*, um ein versehentliches Überschreiben zu verhindern. Mit *ls* kann der Inhalt des Dumps angeschaut werden, mit *cd* in die einzelnen Directories gewechselt werden. Anschliessend werden mit *add* die gewünschten Files oder Directories markiert und mit *extract* wieder auf die Harddisk kopiert.

```
cd /tmp
restore i
   add bin/awk
   extract
mv /tmp/bin/awk /bin/awk
rm /restoresymtab
```

3. Beispiele

Der Einfachheit halber und um ganz konkret zu bleiben, nehme ich 4 Szenarien an, die die Situation der meisten Leser abdecken sollte. Nach einer kurzen Beschreibung der Situation, wird eine passende Backup-Strategie mit konkretem Vorgehen beschrieben.

3.1. Der «arme Leute» - Fall

Situation: Eine unvernetzte NeXTStation mit einer kleinen Festplatte aber ohne DAT, MO oder sonstige Massenspeicher. Es müssen also alle Backups ausschliesslich mit Disketten gemacht werden.

Aus diesem Grund ist es nur unter grossem Zeitaufwand (Disketten wechseln) und Kosten (bei ~300 MByte rund 200 HD- oder 100 ED-Disketten) möglich ein vollständiges Backup zu machen. Es ist also eine Strategie gefragt, die ein vollständiges Backup umgeht.

Die Lösung beruht auf der Annahme, dass ein vollständiges Backup bzw. eine vollständige Restauration länger dauert als eine Neuinstallation ab CD-ROM:
Wenn das System neu installiert aber noch nicht konfiguriert ist, sollte ein leerer Stufe-0-Dump gemacht werden. Das System wird in allen weiteren Dumps höherer Stufe nur noch die Änderungen zur Standardinstallation sichern, also Platz und Zeit sparen.

```
dump 0uf /dev/null /dev/rsd0a
```

Im weiteren sieht ein pseudo-'vollständiger' Dump wie folgt aus (z.B. monatliches grosses Backup):

```
dump 1uOf 2.6 /dev/fd0a /dev/rsd0a
```

Bei einem solchen Dump werden vielleicht noch ganze 40–60 MByte geschrieben was 15–25 ED-Disketten entspricht.

Ein wöchentlicher Dump sieht dann etwa so aus (Es werden nur die Änderungen im laufenden Monat, also vielleicht rund 10 MByte geschrieben):

```
dump 5uOf 2.6 /dev/fd0a /dev/rsd0a
```

Ein möglichst täglicher inkrementeller Dump sieht dann so aus (Spätestens auf dieser Stufe passt das Backup auf eine einzelne Disk):

```
dump 9uOf 2.6 /dev/fd0a /dev/rsd0a
```

Oder mit Komprimierung (*gzip* ist effizienter) auf die Diskette '*/DumpDisk*'

```
dump 9uf - /dev/rsd0a |
  compress > /DumpDisk/sd0a_9.dump.Z
dump 9uf - /dev/rsd0a |
  gzip -9 > /DumpDisk/sd0a_9.dump.gz
```

Eine vollständige Restauration beginnt mit der Neuinstallation von NEXTSTEP, da ja der Stufe-0-Dump nicht existiert. Anschliessend folgt ein Restore des letzten monatlichen Dumps, dann des letzten wöchentlichen Dumps und zuletzt des aktuellen täglichen Dumps.

```
cd /
restore rvf /dev/fd0a
```

3.2. Der Normalfall

Situation: Ein unvernetzter NeXT mit entweder einer zweiten HD, einer MO oder Wechselplatte.

Da ein zweiter grosser Speicher zur Verfügung steht, von dem gebootet werden kann (zwei bootfähige HD's, eine HD und eine bootfähige Optical oder eine Wechselplatte), verringert sich das Risiko drastisch gleich beide Bootmöglichkeiten zu verlieren. Ausserdem vereinfacht sich die Backupstrategie:

Bei 2 HD's brauchen nur noch die vollenständigen Stufe-1-Dumps (siehe Fall 3.1) auf Diskette, die kleineren inkrementellen gehen mit entsprechend gewählter Komprimierung auf die jeweils andere HD in ein */dump* Directory, bis sie zyklisch wieder gelöscht werden.

Zuerst der leere Stufe-0-Dump für beide Disks

```
dump 0uf /dev/null /dev/rsd0a
dump 0uf /dev/null /dev/rsd1a
```

und der 'volle' Stufe-1-Dump

```
dump 1uOf 2.6 /dev/fd0a /dev/rsd0a
dump 1uOf 2.6 /dev/fd0a /dev/rsd1a
```

Im weiteren werden die HD's auf die jeweils andere gedumpt (man beachte den zweifachen *mv*, so wird verhindert dass der Dump der ersten Disk als neues File im Dump der 2. Disk berücksichtigt wird)

```
dump 9uf - /dev/rsd0a |
  gzip -9 > /DumpHD2/sd0a_9.dump.gz
mv /DumpHD2/sd0a_9.dump.gz
  /DumpHD1/sd0a_9.dump.gz
dump 9uf - /dev/rsd1a |
  gzip -9 > /DumpHD1/sd1a_9.dump.gz
mv /DumpHD1/sd0a_9.dump.gz
  /DumpHD2/sd0a_9.dump.gz
```

Möglicherweise kann man sich dazu durchringen nur auf einer Harddisk zu arbeiten, so dass nur diese eine Disk eines inkrementellen Backups bedarf. Dann vereinfacht sich das Vorgehen wie folgt:

```
dump 9uf - /dev/rsd0a |
  gzip -9 > /DumpHD2/sd0a_9.dump.gz
```

Wenn OD oder Wechselplatte gross genug für einen echten Stufe-0-Dump sind, kann nach Fall 3.3 verfahren werden. Andernfalls wird nach Fall 3.1 vorgegangen mit dem Vorteil dass man keine Disketten schaufeln muss und dass man sogar mehrere komprimierte, inkrementelle Dumps auf einen einzigen Datenträger bringt.

```
dump 0uo /dev/rsd0a
```

oder

```
dump 0uf /dev/null /dev/rsd0a
```

anschliessend inkrementelle Dumps

```
dump 9uf - /dev/rsd0a | gzip -9 >
   /DumpOD/level9/sd0a_nr001.dump.gz
```

3.3. Der Idealfall

Situation: Ein unvernetzter NeXT mit DAT.

Im Normalfall ist ein DAT-Tape (≥ 2 GByte) um ein mehrfaches grösser als eine Harddiskpartition (maximal 2 GByte), insbesondere dann, wenn Kompression verwendet wird. Anstelle des Diskjockey-Problems tritt die Frage, wie ich ein DAT-Tape optimal ausnütze.

Zuerst etwas Hintergrundinformation zum Thema DAT, da diese Geräte noch nicht so weit verbreitet sind. Man unterscheidet grundsätzlich zwei Typen: DAT und ExaByte, wobei sowohl DAT (*digital audio tape*) als auch CD-ROM ursprünglich aus der Musikindustrie stammen, wohingegen ExaByte auf dem Videomarkt beheimatet ist. Es gibt einige subtile Unterschiede zwischen den beiden Technologien: DAT hat eine einfachere Kopfgeometrie, es beansprucht deshalb die Bänder weniger stark und kann auch viel billiger aufgebaut werden. ExaByte hat generell die grössere Kapazität als DAT aber verschwendet auch mehr Platz durch ein anderes Verfahren, was sich bei vielen kleinen Dateien durchaus negativ bemerkbar machen kann. Ich bevorzuge deshalb die DAT Technologie!

Diese SCSI Geräte werden über die Devicenamen */dev/rst0* bzw. */dev/rst1* bei DAT und */dev/rxt0* bzw. *dev/rxt1* bei ExaByte angesprochen (es gibt aber auch Fälle, wo ein DAT über */dev/rxt0* angesteuert werden muss, weil es mit */dev/rst0* nicht funktioniert!). Es gilt noch eine Besonderheit zu beachten: Wenn man dem Devicenamen ein 'n' voranstellt (also */dev/nrst0* bzw. */dev/nrxt0*), wird dies als Rückspulverbot verstanden ('*no rewind on close*').

Mit einem DAT ist die Backupstrategie sehr einfach: Es lohnt sich kaum inkrementelle Backups zu machen, da ein vollständiges Backup wirklich schnell ist. Ein Dump von 1 GByte Umfang dauert nur rund 30 Minuten!

Ich bin deshalb dazu übergegangen wöchentlich einen Dump der Stufe 0 zu machen. Inkrementelle tägliche Dumps mache ich nach Bedarf und Laune auf die alte Harddisk, auf der zwar ein Minimalsystem installiert ist (2. Bootmöglichkeit aus Sicherheitsgründen) aber nur wenige aktuelle Daten vorkommen, die eines regelmässigen inkrementellen Backups bedürfen.

Meine Arbeit mit dem Backup beschränkt sich deshalb auf einen samstäglichen Dump der Stufe 0 von allen drei Filesysteme über Mittag. Und alle ein bis zwei Tage auf einen inkrementellen Dump auf die alte HD, der 2–3 Minuten dauert. Im Moment verwende ich zyklisch fünf DAT-Tapes. Wie oft und wie lange ich sie benutzen kann weiss ich nicht genau — laut Herstellerangabe sollen sie Daten während 10 Jahren sicher speichern.

Die Befehle für den vollständigen Stufe-0-Dump von drei Partitionen hintereinander auf ein DAT-Tape, mit anschliessendem Zurückspulen und Auswerfen des Tapes:

```
dump 0ufsd /dev/nrst0 4096 1048576 /dev/rsd1a
dump 0ufsd /dev/nrst0 4096 1048576 /dev/rsd0b
dump 0ufsd /dev/rst0  4096 1048576 /dev/rsd0a
mt -f /dev/rst0 rewoffl
```

Man kann beliebig viele Dumps hintereinander auf ein DAT-Tape speichern, solange man Platz hat und stets einen 'n'-Devicenamen verwendet wie etwa */dev/nrst0* und erst beim letzten Dump darauf verzichtet. Dies ist ganz praktisch wenn man ein Backup von 1–2 Dutzend Originaldisketten erstellen will oder auch einige Public-Domain oder Daten-CD's kopieren möchte.

In diesem Zusammenhang habe ich noch eine Frage an den geneigten Leser: Ist es möglich, das DAT-Tape im Audio-Format mit Soundfiles zu beschreiben, um sie später auf der Stereoanlage oder im DAT-Walkman zu spielen?

Hier die entsprechenden inkrementellen Dumps der externen Harddisk auf die alte interne Harddisk unter */mnt*:

```
dump 5uf - /dev/rsd0b | gzip >
/mnt/Dump/eb_l5_n038.dump.gz
dump 5uf - /dev/rsd0a | gzip >
/mnt/Dump/ea_l5_n038.dump.gz
```

Noch eine letzte Bemerkung zu *restore*. Da nun mehrere Dumps hintereinander auf dem Tape sind, muss bei einer Restaurierung auch angegeben werden, welcher Dump gemeint ist. Dies geschieht über die Option '*s*' (gleiches Beispiel wie oben, es wird der dritte Dump benutzt):

```
cd /tmp
restore isf 3 /dev/rst0
   add bin/awk
   extract
```

3.4. Der Spezialfall

Situation: Ein NeXT der mit anderen Unixrechnern Daten via Datenträger tauscht.

Hier kann es vorkommen, dass eine reine Datenpartition eingerichtet ist. Nun macht es plötzlich Sinn das Backup eines NeXT auf einer SUN wieder zu installieren (oder umgekehrt), weil beide Rechner mit den gleichen Daten arbeiten sollen. In einem solchen Fall tritt leider ein Kompatibilitätsproblem auf: NeXT hat Modifikationen am Datenformat von *dump* vorgenommen, sodass es nur mit *dump.old* und *restore.old* möglich ist, eine solche Aufgabe zu lösen.

Wenn aber kein eigenes Filesystem ausschliesslich für Daten existiert oder Daten mehrerer Projekte sich ein Filesystem teilen, kann *dump* nicht verwendet werden. In einem solchen Fall muss man, auf die Geschwindigkeit von *dump.old* und *restore.old* verzichten und statt dessen *tar* oder *cpio* verwenden, die es erlauben einzelne Directories inklusive Subdirectories zu kopieren. Dabei kann es leicht vorkommen, dass man in vorher beschriebene Probleme läuft. In diesem Fall muss man auf *gnutar* ausweichen, was zum Glück einigermassen unproblematisch ist, da es auf viele Systeme portiert wurde und recht verbreitet ist.

4. Schlusswort

Hoffentlich hat jeder ein Beispiel gefunden, das seiner Backup-Situation einigermassen gerecht wird (Ohne vom Umfang des Artikels erschlagen worden zu sein). Falls weitere Fragen auftreten, findet man in der NeXT-Dokumentation Hilfe oder man kann sich auch an mich wenden.

Werner Burri

E-mail:
nice!Werner.Burri
burri@akasha.nice.ch

Jahres-Inhaltsverzeichnis

Hier eine Übersicht über alle im Jahr 1993 erschienenen Artikel. Die Angaben beziehen sich jeweils auf Ausgabe und Seite.

Pretty Good Privacy

Allen Ausfuhrbeschränkungen des US-Exportgesetzes zum Trotz, befindet sich seit geraumer Zeit auf vielen ftp-Servern im Internet ein frei verfügbares Public-Key-Kryptosystem namens Pretty Good Privacy (PGP).

Die US-Exportgesetze behandeln kryptographische Systeme wie Kriegswaffen. Sie dürfen ab einer Schlüssellänge von 40 Bit nicht exportiert werden, da sie «feindlichen Drittstaaten» oder Terroristen in die Hände fallen könnten. Da die *National Security Agency* (NSA) ihre weltweiten Lauschaktionen dann aber kaum noch durchführen könnten, gilt der Export als Bedrohung der nationalen Sicherheit.
In den USA selbst sind viele solcher Systeme in jedem Computerladen erhältlich. Für die Exportversion der gleichen Software werden dann die Verschlüsselungsfunktionen durch einfachere Algorithmen ersetzt, die oftmals leichter zu knacken sind.

Zum Nulltarif

Seit kurzem gibt es leistungsfähige Kryptographie zum Nulltarif: *Pretty Good Privacy*, kurz PGP, nennt sich ein amerikanisches Freeware-Paket, das neben konventioneller, symmetrischer Verschlüsselungstechnik auch ein ausgereiftes Public-Key-Kryptosystem enthält. Entwickelt wurde es von Philip Zimmermann, einem Programmierer aus Boulder, Colorado. Er baute dazu das bekannte Public-Key-System RSA nach und stattete PGP mit umfachreichen Schlüsselverwaltungsfunktionen aus. Neben der reinen Chiffrierung ermöglicht es auch die Generierung elektronischer Unterschriften.

PGP enthält zudem den Komprimierungsalgorithmus PKZIP, was nicht nur Platz spart, sondern ein Knacken der Chiffrierung zusätzlich erschwert. PGP gibt es heute für diverse Plattformen; der Quellcode ist frei verfügbar. Die Bedienung ist zwar nicht besonders komfortabel, doch entsprechende Frontend-Programme werden bereits angeboten.

Lasst uns verschlüsseln!

Die von mir geteste Version von PGP trägt die Versionsnummer 2.3a (Release July 2, 1993) und beinhaltet sämtliche Source- und Makefiles für die diversen Plattformen auf denen PGP läuft. Nach dem Entpacken wechselt man ins *Source*-Verzeichnis und startet über eine Shell das entsprechende Makefile:

```
localhost> make next
```

Wenn alles vorbei ist erhält man ein Programm namens *pgp*. Vor der endgültigen Installation, sollte ein kleiner Testlauf durchgeführt werden, der in der mitgelieferten Dokumentation beschrieben ist.

Wie verwendet man PGP?

Um eine kurze Befehlsübersicht zu sehen, ruft man PGP mit der Option '*h*' (wie *help*) auf:

```
pgp -h
```

Um einem Klartext mit den öffentlichen Schlüssel des Empfängers zu verschlüsseln, tippt man

```
pgp -e textfile userid [other userids]
```

Dieses Kommando erzeugt einen Schlüsseltext namens *textfile.pgp*, allerdings nicht in einer ASCII-Repräsentation! Soll die verschlüsselte Datei per Mail versandt werden, so muss eine ASCII-Konvertierung durchgeführt werden, was man durch die Option `-a` erreicht. PGP versucht den Klartext vor der Verschlüsselung zu Komprimieren. Somit kann es durchaus sein, dass *textfile.pgp* kleiner ist als das ursprüngliche File.

PGP versucht den String *userid* als Teil einer Benutzer-ID im öffentlichen Schlüsselbund des Absenders zu lokalisieren. Es können auch Leerzeichen verwendet werden, allerdings sollte man dann die ganze *userid* in Hochkommas einschliessen. PGP verwendet den ersten gefundenen Public-Key, um den Klartext zu verarbeiten. Soll die Nachricht an mehr als nur einen Empfänger geschickt werden, so gibt man alle Empfänger in der Kommandozeile an.

Beispiel:

```
pgp -e brief.txt Alice Bob Carol
```

Diese Zeile erzeugt ein File namens *brief.pgp*, dass von Alice oder Bob oder Carol entschlüsselt werden kann!

Um ein Dokument mit dem eigenen geheimen Schlüssel zu unterzeichnen tippt man:

```
pgp -s textfile [-u your_userid]
```

Die Option `-u` ist nur für den Fall, dass ein Benutzer verschiedene IDs und somit auch verschiedene Schlüssel besitzt. Wird der Userid-Teil weggelassen, sucht PGP nach dem ersten Schlüssel am geheimen Schlüsselbund (*secring.pgp*) des Benutzers. Um eine Nachricht oder ein Dokument erst zu unterschreiben und dann zu verschlüsseln, tippt man

```
pgp -es textfile userid [-u your_userid]
```

Will man schliesslich aus einer erhaltenen Nachricht den Klartext extrahieren bzw. die Unterschrift prüfen lassen, so tippt man

```
pgp ciphertext [-o plaintext]
```

Der Schlüsseltext (*ciphertext*) sollte dabei die Extension *.pgp* besitzen. Wird über die Option `-o` kein Name für den zu entschlüsselten Klartext angegeben, wird der Name des Schlüsseltext ohne Extension verwendet. Es ist zu beachten, dass der Vorgang der Entschlüsselung vollständig automatisch abläuft, unabhängig davon, ob die erhaltene Nachricht nur unterschrieben, nur verschlüsselt, oder beides ist!

Schlüsselverwaltung

Die Schlüsselverwaltung ist der heikelste Teil der Kryptographie. PGP behauptet von sich, gerade in diesem Punkt einige Vorteile gegenüber anderen Systemen zu besitzen.

Um ein neues Schlüsselpaar zu generieren, tippt man

```
pgp -kg
```

Danach wird man, wie schon beim Testlauf nach der Compilation, nach der Schlüssellänge gefragt. Hier gilt die einfache Faustregel *«mehr Stellen = mehr Sicherheit»*, aber auch *«mehr Stellen = mehr Kaffeepausen»* (soll heissen: mehr Sichheit auf Kosten der Verarbeitungsgeschwindigkeit).
Als weitere Eingabe verlangt PGP eine Benutzer-ID, also im Normalfall den vollständigen Namen. Damit wird das Risiko verwechselt zu werden geringfügig kleiner. Verwechslung in einem Public-Key-Kryptosystem bedeutet, dass die Software des Absenders die Nachricht, die für mich bestimmt wäre, mit dem öffentlichen Schlüssel einer anderen Person verschlüsselt. Damit ist sie aber für mich nicht mehr lesbar! Aus diesem Grund schlägt Philip Zimmermann vor, zusätzlich zum vollständigen Namen die eigene Mail-Adresse zu verwenden, also zum Beispiel:

```
Douglas Adams <dna@dadams.demon.co.uk>
```

Aus den Angaben des Benuzters erstellt PGP zwei Files, *pubring.pgp* und *secring.pgp*, den öffentlichen und den geheimen Schlüsselbund des Benutzers. Will man nachträglich einen (oder mehrere) Schlüssel hinzufügen, so tippt man

```
pgp -ka keyfile [keyring]
```

Dabei kann es sich bei *keyfile* z.B. um den öffentlichen Schlüsselbund eines anderen Benutzers handeln.

Um einen Schlüssel nachträglich wieder zu entfernen, tippt man

```
pgp -kr userid [keyring]
```

Ähnlich ist die Syntax, wenn man einen Schlüssel aus seinem Keyring-File extrahieren will, z.B. um ihn einem anderen Benutzer zu übermitteln.

```
pgp -kx userid keyfile [keyring]
```

Dabei gibt *keyfile* den Namen desjenigen Files an, in das der extrahierte Schlüssel geschrieben werden soll.

Wie funktioniert's?

An anderer Stelle in dieser Ausgabe wurde bereits angesprochen, dass Public-Key-Algorithmen im allgemeinen langsamer sind als die entsprechenden symmetrischen Verfahren. Daher verwendet (nicht nur) PGP eine Kombination beider Verfahren. Für den Benutzer unsichtbar wird zuerst ein temporärer, zufälliger Schlüssel *ST* bestimmt, mit dessen Hilfe der Klartext konventionell verschlüsselt wird. Danach wird der öffentliche Schlüssel des Empfängers dazu verwendet, den temporären Zufallsschlüssel *ST* seinerseits zu verschlüsseln, um ihn zusammen mit der Nachricht z.B. einem Mailserver zu übergeben.
Der Empfänger erhält durch Anwendung seines geheimen Schlüssels Zugang zum temporären Zufallsschlüssel, der nicht nur die ursprüngliche Nachricht chiffriert hat, sondern wiederum in der Lage ist, die Nachricht zu entschlüsseln.

PGP verwaltet Public-Keys in sogenannten Schlüsselzertifikaten (*key certificates*). Neben einer Benutzer-ID und dem Datum der Generierung des Schlüsselpaares, enthalten sie den eigentlichen Schlüssel. Öffentliche Schlüsselzertifikate enthalten Informationen über die öffentlichen Schlüsselteile, während geheime Zertifikate den geheimen, privaten Teil enthalten. Jeder geheime Schlüssel ist zusätzlich mit einem Passwort geschützt, falls er einmal in falsche Hände fallen sollte. Ein Schlüsselfile, auch Schlüsselbund (*key ring*) genannt, enthält ein oder mehrere Zertifikate. Auch hier existiert wiederum eine öffentliche und eine geheime Variante des Schlüsselbunds.
Schlüssel werden intern durch eine Schlüssel-ID (*key-ID*) referenziert, was einer "Abkürzung" des Schlüssels entspricht. Dazu werden die *least significant* 64 Bit des Schlüssels verwendet, die ihrerseits auf 24 Bit gekürzt werden, wenn die Schlüssel-ID dargestellt wird. Obwohl viele Schlüssel sich eine gemeinsame Benutzer-ID teilen können, kommt es aus praktischen Gründen nie vor, dass zwei Schlüssel eine identische Schlüssel-ID besitzten.

PGP verwendet zur Generierung von elektronischen Unterschriften sogenannte *message digests*. Das sind, auf die ganze Nachricht angewandte, Einweg-Hashfunktionen mit 128 Bit Länge, vergleichbar mit einer Checksumme oder einem CRC Fehlercode. Vergleichbar in dem Sinne, als dass sie einer kompakten Repräsentation der ursprünglichen Nachricht entsprechen und zur Aufspürung einer nachträglichen Veränderung des Nachrichteninhalts verwendet werden können. Um die elektronische Unterschrift zu bilden, wird der *message digest* mit den geheimen Schlüssel des Absenders verschlüsselt.

Chiffrierte Nachrichten oder Dokumente enthalten als erstes die Schlüssel-ID des öffentlichen Schlüssels, der zur Verschlüsselung verwendet wurde. Damit kann die Software des Empfängers den korrespondierenden geheimen Schlüssel zur Entschlüsselung nachschlagen.

Nachrichten werden in PGP «unterschrieben», indem man ihnen ein Unterschriften-Zertifikat voranstellt. Darin enthalten sind neben der Schlüssel-ID des zur Unterschrift verwendeten Schlüssels, der oben beschriebene *message digest* und das Erstellungsdatum der Unterschrift. Die Schlüssel-ID wird von der Software des Empfängers dazu verwendet, den öffentlichen Schlüsselteil des Absender nachzuschlagen, um die Unterschrift zu prüfen.

Fazit

In der Fachwelt hat PGP für einige Aufregung gesorgt; dies nicht nur, weil der von PGP benutzte RSA-Algorithmus in den USA patentiert ist, sondern weil der Autor auch die Kühnheit besass, sein Programm inklusive Quellcode jedem kostenlos abzugeben, der es haben wollte. Das Programm verbreitete sich in Windeseile in der amerikanischen Mailbox-Szene und fand rasch seinen Weg in internationale Computernetzwerke wie Internet oder CompuServe. Es dauerte nicht lange, bis die US-Zollfahndung auftauchte und die Entfernung des Programms aus zahlreichen Mailboxen verfügte. Internet und CompuServe führten es aber weiterhin.

Dominik Moser

Was sie schon immer über TeX wissen wollten - Teil 4

Jeder, der schon einmal gezwungen war, mathematischen Text mit einem herkömmlichen Textverarbeitungssystem zu erstellen, hat wohl eine leidvolle Erfahrung machen müssen. Bereits das Einfügen einiger Leerzeichen kann das Layout einer mühsam eingegebenen Formel zunichte machen – die sauber gesetzten Indizes stehen irgendwo verloren im Raum herum.

Der erste Schritt

Mit diesen Problemen sollte Schluss sein, wenn man mit TeX arbeitet, denn gerade auf diesem Gebiet kann es seine wahren Qualitäten zum Ausdruck bringen. Der erste Schritt besteht in der Kennzeichnung. Mathematische Formeln im laufenden Text müssen grundsätzlich zwischen zwei Dollarzeichen eingeschlossen werden. Dadurch versetzt man den TeX-Compiler in den Mathematik-Modus, was das Umschalten auf die Schriftart *math italic* und eine Anpassung der Abstandsberechnung bewirkt.
Um eine Formel ausserhalb eines Textblocks darzustellen, dazu noch vergrössert und zentriert, muss man sie mit zwei Dollarzeichen (`$$`) einleiten und abschliessen. Diesen hervorgehobenen Mathematiksatz nennt man *Display-Style* (im Unterschied zum vorher beschriebenen *Text-Style*).

Hoch- und Tiefstapler

In beiden Mathematikmodi leitet man einen Index mit einem Underscore (_) ein. Um die chemische Formel für Wasser zu erzeugen, würde man also `H$_2$O` oder `${\rm H}_2{\rm O}$` eingeben. Benutzt wird der Modus hier allein zum Setzen des Index. Im längeren Ausdruck verwendet man den Mathematik-Modus als Basis und schaltet intern auf die Schriftart *roman* um, anderenfalls würden die Buchstaben in *math italic* ausgegeben; im kürzeren Ausdruck wird nur der Index im Mathematik-Modus gesetzt, wodurch die Buchstaben 'H' und 'O' in der gerade aktuellen Schrift verbleiben.

Exponenten leitet man durch einen Uparrow (^) ein; ansonsten werden sie genau wie Indizes behandelt. Ein besonders gebräuchlicher Exponent ist das "'"; der TeX-Befehl dazu ist `\prime`, man kann aber auch einfach ein Apostroph eingeben. Ein typisches Polynom sieht also in TeX folgendermassen aus:

```
$f'(x)= a_n x^n + a_{n-1} x^{n-1} +
\ldots + a_1 x + a_0$
```

Hier benötigt man die geschweiften Klammern, um dem Compiler mitzuteilen, dass der Index bzw. der Exponent aus mehr als einem Zeichen besteht. Der Befehl `\ldots` erzeugt eine Sequenz aus drei Punkten. Eine häufige Anwendung der Exponenten ausserhalb von mathematischen Formeln, stellt die Numerierung von Fussnoten dar. Dies kann man mittels der Anweisung `\footnote{$^1$}{`*Fussnotentext*`}` leicht in ansprechender Form erreichen.

Sowohl im normalen Mathematik-Modus als auch im Display-Style verkleinert TeX Indizes bis zur zweiten Ordnung, wie zum Beispiel in `$x^{a_i}$`. Erst Indizes dritter Ordnung werden wieder in der gleichen Grösse wie Indizes zweiter Ordnung ausgegeben, wie man in `$x_{a_{i_j}}$` sehen kann. Die hierbei verwendeten Schriftattribute sind über die beiden Befehle `\scriptstyle` und `\scriptscriptstyle` zugänglich.

Abstandsautomatik

Wie bereits erwähnt wird im Mathematik-Modus die Abstandsbestimmung anders gehandhabt als im Textmodus. Grundsätzlich gilt, dass Leerzeichen in der Eingabe ignoriert werden. Einzig am Ende eines Befehles muss ein Leerzeichen stehen.

Intern besitzt jedes im Mathematik-Modus zulässige Zeichen eine Kennung, die besagt, welcher Natur das

Zeichen ist. TeX unterscheidet dabei zwischen binäreren Operatoren (z.B. +, -, *), «grossen Operatoren» (z.B. Σ, ∫ oder Π), Relationen (z.B. < oder =), öffnenden bzw. schliessenden Klammern, Satzzeichen und sogenannten «normalen Zeichen». Diese Zeichenkennung verwendet TeX zur Abstandsberechnung, indem mit Hilfe einer Tabelle bestimmt wird, welcher Abstand zwischen den entsprechenden Gruppen zu setzen ist. Die Abstände variieren von keinem Abstand (z.B. zwischen zwei normalen Zeichen) über einen kleinen (zwischen einer öffnenden Klammer und einem normalen Zeichen) und einem mittleren (zwischen binären Operatoren und normalen Zeichen) bis zu einem grossen Abstand (zwischen einer Relation und normalen Zeichen).

Diese Berechnungen laufen für den Anwender völlig unsichtbar ab. Falls man dennoch einmal mit dem von TeX gewählten Abständen nicht zufrieden sein sollte, kann man diese in sehr einfacher Weise korrigieren. Am häufigsten tritt der Fall eines zu grossen Abstands ein, den man mittels `\!` verkleinern kann. Dabei wird der Abstand um ein Sechstel eines `\quad` verringert, ist also vom gewählten Font abhängig. Daneben gibt es die Möglichkeit, einen Abstand zu vergrössern, indem man `\,` für einen kleinen, `\>` für einen mittleren oder `\;` für einen grossen Abstand eingibt. Hier eine Anwendung des oben gesagten:

```
$$\int\!\!\int z\,dx\,dy$$
```

$$\int\!\!\int z\,dx\,dy$$

Die beiden Integralzeichen werden enger zusammengerückt, während man zwischen dem Integrationsausdruck *z* und den beiden Flächen- bzw. Volumenelementen *dx* und *dy* einen grösseren Abstand setzt.

Alle Besonderheiten des Schriftsatzes, die bei mathematischen Formeln sonst noch auftreten (griechische Buchstaben, Symbole, Brüche, Wurzelzeichen, Integrale, Summen und Produkte), sind in Form von Befehlen verfügbar.

Griechische Buchstaben

Die Eingabe griechischer Buchstaben geschieht einfach über den üblichen Namen des Buchstabens. Befehle für kleine griechische Buchstaben beginnen mit einem Kleinbuchstaben am Befehlsanfang, für grosse mit einem Grossbuchstaben. Für griechische Zeichen, die identisch zur Normalschrift sind – zum Beispiel das kleine *o* oder die meisten grossen Buchstaben – sind keine speziellen Befehle vorhanden. Dort reicht die Angabe des normalen Buchstabens, versehen mit der Schriftart `\rm`.

Hier ein kleiner Ausschnitt aus der Tabelle der griechischen Kleinbuchstaben:

α	`\alpha`	ι	`\iota`	σ	`\sigma`
β	`\beta`	κ	`\kappa`	τ	`\tau`
γ	`\gamma`	λ	`\lambda`	ϕ	`\phi`
δ	`\delta`	μ	`\mu`	χ	`\chi`
ε	`\epsilon`	ν	`\nu`	ω	`\omega`
…		…		…	

Brüche

Ein normaler Bruch wird mit dem Befehl `\over` eingeleitet. Dabei wirkt `\over` wie ein binärer Operator; was links steht, bis zur nächsten öffnenden Klammer oder bis zum Dollarzeichen kommt in den Zähler, was rechts steht bis zur nächsten schliessenden Klammer oder $ (bzw. $$) in den Nenner. Wie Brüche werden auch Binomial-Koeffizienten gesetzt. Der zugehörige Befehl heisst `\choose`. Praktisch gesehen handelt es sich um echte Brüche ohne Bruchstrich, jedoch mit Klammern. Die entsprechenden Befehle für Binomial-Koeffizienten mit eckigen bzw. geschweiften Klammern heissen `\brack` und `\brace`.
Beispiel:

```
$$1 \over {1 + {1 \over x+1}}$$
```

$$1 \over {1 + {1 \over x+1}}$$

Wurzelzeichen

Wurzelzeichen werden mit `\sqrt` eingegeben, die n-te Wurzel mit `\root n \of`. Dabei wird die Höhe des Wurzelzeichens automatisch bestimmt. Soll mehr als ein Zeichen unter der Wurzel stehen, sind wie üblich geschweifte Klammern zu setzen.
Beispiele:

```
$\sqrt x$
$\sqrt{x^3 + \sqrt\alpha}$
$\root n+1 \of {x^n + y^n}$
```

$$\sqrt x$$
$$\sqrt{x^3 + \sqrt\alpha}$$
$$\root n+1 \of {x^n + y^n}$$

Integrale, Summen und Produkte

Grosse Operatoren, wie Summenzeichen und Integrale, stehen meist für sich allein und dominieren durch ihre Grösse das Aussehen einer mathemaischen Formel. Neben Summenoperator (`\sum`), Integral (`\int`) und Produktzeichen (`\prod`) existieren auch noch grosse Operatoren für Durchschnitt (`\bigcap`) und Vereinigung (`\bigcup`) sowie Existenz- (`\bigvee`) und Allquantoren (`\bigwedge`). Häufig besitzen sie untere und obere Grenzen, die mittels '_' und '^' in der Form

```
$\<operator>_{untere Grenze}^{obere
Grenze}$
```

angegeben werden. Beispiele:

```
$\sum_{i=1}^n a_i$
$\int_0^1 \ln(x)\,dx$
```

Bei den grossen Operatoren wie Summe oder Integral werden die Indizes im Text-Style neben das Zeichen gesetzt, wodurch extreme Ober- und Unterlängen vermieden werden können. Gibt man hingegen ein Summenzeichen im Display-Style aus, so werden – wie allgemein üblich – die Indizes über und unter das Zeichen gesetzt. Im Gegensatz dazu werden die Integrationsgrenzen auch im Display-Style immer neben das Integralzeichen gesetzt.

Mathematische Funktionen

Es gibt eine Reihe mathematischer Funktionen, die in Veröffentlichungen üblicherweise nicht kursiv gesetzt werden, wie beispielsweise «sin» oder «cos» Diese Funktionen sind in TeX als Befehle vorhanden und werden gleich behandelt, wie die oben erwähnten grossen Operatoren. Anhand des folgenden Beispiels kann man den Unterschied zwischen der Zeichenfolge '`s`' '`i`' '`n`' und dem Befehl `\sin` sehen:

```
$$sin2\beta=2\sin\beta\cos\beta$$
```

$$sin2\beta=2\sin\beta\cos\beta$$

Bei Funktionen wie «max» oder «lim» werden oft zusätzliche Bedingungen bzw. Intervalle für die verwendeten Indizes unter den Funktionsnamen geschrieben. Verwendet man zur Darstellung die dafür vorgesehenen Befehle, kann man die gewünschten Grenzen mittels '_' unterhalb des Funktionsnamens plazieren. Beispiele:

```
$$\max_{1<i<n} \log_2 a_i$$
$$\lim_{n \to \infty} x_n=0$$
```

$$\max_{1<i<n} \log_2 a_i$$
$$\lim_{n \to \infty} x_n=0$$

Klammern, wie man will

Neben den runden, kommen auch eckige, geschweifte und spitze Klammern zur Anwendung. Zur Grössenangabe werwendet man die Schlüsselwörter `\bigl`, `\Bigl` (1.5 × `\bigl`), `\biggl` (2 × `\bigl`) und `\Biggl` (2.5 × `\bigl`) für linke Klammern und entsprechend `\bigr` bis `\Biggr` für rechte Klammern. Für Trennstriche existieren analog `\bigm` bis `\Biggm`, mit 'm' wie *middle.*
Beispiel:

```
$$p(x):=\sum_{i=1}^n \biggl(
\prod_{\scriptstyle j=1\atop \scriptstyle
j\ne i}^n {x-x_j \over x_i-x_j}\biggr) \cdot
f_i$$
```

Normale Zeichen können, falls die entsprechende Zeichengrösse verfügbar ist, mit `\big` bis `\Bigg` expandiert werden. Das Schlüsselwort wird einfach vor das zu verwendende Zeichen gestellt. Dabei ist es TeX gleichgültig, ob man dem Schlüsselwort eine öffnende oder schliessende Klammer folgen lässt. Wenn man z.B. `\bigl]` eingibt, wird diese Klammer mit den Abständen einer öffnenden Klammer gesetzt. Dies kann man sich bei offenen Intervallen zunutze machen, wenn man etwa]-a,a[anstelle von (-a,a) schreiben will.

Neben dieser expliziten Grössenangabe besteht noch die Möglichkeit, TeX selbständig die Grösse der Klammer bestimmen zu lassen. Dazu verwendet man die Schlüsselwörter `\left` und `\right`, gefolgt von der gewünschten Klammer. TeX setzt dann die am ehesten der Höhe und Tiefe des Ausdrucks entsprechende Klammer, was in den meisten Fällen zufriedenstellende Ergebnisse liefert. Dabei ist noch zu beachten, dass diese Schlüsselwörter immer paarweise auftreten müssen. Möchte man nur eine der beiden Klammern, so muss man das zweite Schlüsselwort durch einen Punkt gefolgt eingeben. Der Befehl `\cases`, der dazu dient, eine Fallunterscheidung zu erzeugen, benutzt eben diesen Trick.
Beispiel:

```
$$\left|x\right|=\cases{
   x &falls $x\ge 0$\cr
  -x &sonst}
$$
```

$$\left|x\right|=\cases{ x & falls $x \geq 0$ \cr -x & sonst}$$

Matrizensetzen leichtgemacht

Wer schon einmal mit einem WYSIWYG-System eine Matrix oder eine mehrzeilige Formel gesetzt hat, wird dabei auftretende Probleme kennen. Gerade in solchen Situationen beginnt man, TeX als ausgesprochen komfortables System zu geniessen. Man benötigt zur Eingabe einer Matrix nur den Befehl `\matrix{}`, dem man im Innern der geschweiften Klammern den Inhalt der Matrix folgen lässt. Die einzelnen Spalten werden dabei durch ein Tabulatorzeichen (`&`) gekennzeichnet; die Zeilenenden werden mit Hilfe des Befehls `\cr` markiert. Die zu verwendenden Klammern kann man beliebig wählen. Man gibt sie im allgemeinen mittels `\left` und `\right` ein. Will man runde Klammern verwenden, so kann man an Stelle von

```
$$\left(\matrix{…}\right)$$
```

kurz und bündig `$$\pmatrix{…}$$` schreiben, wobei das 'p' für *paranthesized* (geklammert) steht.

Aus der folgenden Matrix, die mittels `\pmatrix` eingegeben wurde,

```
$$A=\pmatrix{
x-\lambda &      1    &      0 \cr
     0    & x-\lambda &      1 \cr
     0    & 0         & x-\lambda \cr
}$$
```

entsteht durch Verwendung des Befehles `\matrix` und der expliziten Wahl der neuen Klammern eine Determinante:

```
$$\det A=\left|\matrix{…}\right|$$
```

$$\det A=\left|\matrix{x-\lambda & 1 & 0 \cr 0 & x-\lambda & 1 \cr 0 & 0 & x-\lambda \cr}\right|$$

Ausgerichtete Gleichungen

Häufig verlangt man zueinander ausgerichtete Formeln. Hierbei ist es zum Beispiel wünschenswert, dass alle Gleichheitszeichen übereinander zu stehen kommen. Man erreicht das erwünschte Ziel durch Verwendung des Befehles `\eqalign{}`, der für *equation alignment* steht. In geschweiften Klammern folgen die einzelnen Gleichungen, jeweils durch ein `\cr` getrennt. Jede der Gleichungen muss dabei genau ein Tabulatorzeichen (`&`) enthalten. TeX setzt dann die Gleichungen so, dass die Tabulatoren exakt übereinander zu stehen kommen.

Beispiel:

```
$$\eqalign{
  X_1 + \cdots + X_p &= m \cr
  Y_1 + \cdots + Y_q &= n \cr
}$$
```

Der Befehl ist nur im Display-Style sinnvoll und zulässig. Sollen mehrere Formelblöcke gegeneinander ausgerichtet werden, so kann dies durch mehrfache Anwendung von `\eqalign` im gleichen `$$`-Block geschehen.

Möchte man die Gleichungen zudem noch numerieren, so kann man `\eqalignno{}` für Numerierung auf der rechten Seite oder `\leqalignno{}` für Numerierung auf der linken Seite verwenden. Dabei werden die einzelnen Gleichungen wiederum durch `\cr` getrennt, aber zusätzlich zum Tabulator, der die vertikale Ausrichtung der Einzelgleichungen bestimmt, wird noch ein zweiter Tabulator, gefolgt von der zu verwendenden Zeilen- oder Formelnummer, eingegeben. Endet eine Gleichung mit '`&\cr`', so entfällt die Zeilennummer. Beispiel:

```
$$\eqalignno{
  X_1 + \cdots + X_p &= m &(1)\cr
  Y_1 + \cdots + Y_q &= n &(2)\cr
}$$
```

Die Numerierung von einzelnen Formelzeilen erfolgt durch `\eqno` rechtsseitig und durch `\leqno` linksseitig. Beispiel:

```
$$\sin 18^\circ = {1\over 4}\left(\sqrt5 - 1
\right) \eqno(1)$$
```

Heikle Formeltrennung

Ein grosses Problem stellt für TeX die automatische Trennung von Formeln dar. Diese erfolgt prinzipiell nur im Text-Style (nicht jedoch im Display-Style) und sollte durch den Anwender selbst vorgenommen werden. TeX teilt Formeln nur bei Relationen und binären Operatoren. Verhindern lässt sich eine Trennung leicht, indem man die Formel in geschweifte Klammern setzt.

Beispielsweise könnte

```
$f(x)=3x^2+2x-1$
```

sowohl nach dem Gleichheitszeichen als auch nach den Operatoren '+' und '-', die Formel

```
$f(x)={3x^2+2x-1}$
```

hingegen nur nach dem Gleichheitszeichen getrennt werden.

Daneben kann man die Trennung von Formeln zulassen, indem man an die entsprechenden Stellen den Befehl `\allowbreak` einfügt. Wie im Textstyle verbietet `\nobreak` die Trennung an der gekennzeichneten Position. Für multiplikative Terme gibt es noch eine besondere Form der Trennhilfe (`\*`), die wie ein '`\`' im Textmodus wirkt. Als Trennsymbol wird dabei das ×-Zeichen (`\times`) eingesetzt. Eine Anwendung davon wäre z.B.:

```
$(x_1+1) \* (x_2+2) \* (x_3+3) … $
```

Formeln im Display-Style werden nicht automatisch getrennt. Die stilistisch angemessene Form der Trennung ist jeweils vor einem binären Operator mit Einrückung des zweiten Formelteils per `\qquad`.

Quod erat demonstrandum

Natürlich kann man in einem einzigen Artikel kaum auf die ganzen Feinheiten des Mathematiksatzes eingehen. Für die tägliche Arbeit mit TeX sollte die kleine Einführung aber allemal reichen. Dem geneigten TeXianer möchte ich noch die Kapitel 16–19 und 26 im TeXbook von Knuth ans Herz legen. Darin wird dem Mathematik-Modus der gebührende Platz eingeräumt.

In der nächsten Folge geht es um das Setzen von Tabellen. Bis dahin viel Spass mit TeX.

Dominik Moser

Externe Harddisk, Monitorprobleme und CompuServe

Aus Deutschland erreichte uns der folgende Leserbrief. Allfällige Anmerkungen oder Ratschläge anderer Mitglieder bitte an die Redaktion des *PowerKey* übermitteln oder aber direkt Kontakt mit Dr. Klaus Kunze aufnehmen.

Liebe NeXT-User

Ich bin Mitglied der NiCE – NeXT User Group und habe den ***PowerKey*** immer als nützliche Informationsquelle für Tips und Tricks rund um NeXT geschätzt. Insbesondere Infos über den Anschluss von externen SCSI-Festplatten wie z.B. in der Ausgabe 6/93 waren wertvoll, da hier vom Händler meist nur wenige gute Tips zu Anschlussproblemen zu erhalten sind.

Festplatten, die direkt vom NeXT-Händler in Deutschland angeboten werden, sind nach wie vor überteuert, der Weg über ein Versandhaus die günstige Alternative – jedoch nur dann, wenn sich die Festplatte ohne Probleme anschliessen lässt.

Externe Festplatten

Meine Erfahrungen beruhen auf dem Einsatz einer IBM 662-S12 und einer Fujitsu 2694 Festplatte (je 1 GB).

Zur IBM 662-S12: Trotz Änderungen in der Datei */etc/disktab* und vielem Ausprobieren von Jumpersettings ist es mir nicht gelungen, diese Platte zum Einsatz zu bringen. Sie lässt sich formatieren aber nicht initialisieren. Die in der Konsole auftauchenden Fehlermeldungen konnte mir keiner erklären, und auch ein Faxwechsel mit IBM Deutschland (die Leute vom Support waren sehr bemüht) hat nicht weitergeholfen. Schliesslich tauschte ich die Festplatte, deren Performance-Daten ja einiges versprachen, um gegen eine Fujitsu 2694.

Auch mit dieser Platte gab es anfangs grosse Probleme, da unser kleines «Netzwerk», bestehend aus zwei NeXTstations, komplett für einen Tag lahmlegten. Zunächst wurde die für Macintosh vorformatierte Platte, die als einziges externes SCSI-Device am NeXT-Client angeschlossen war, richtig erkannt. Sie wurde mit

```
disk -F /dev/rsd1a
```

richtig formatiert (20 Minuten) und danach für zwei Partitionen initialisiert. Auch ein neues Filesystem konnte kreiert werden.

So weit so gut. Ich beabsichtigte, die neue Festplatte als Boot-Platte (SCSI-ID=0) am NeXT-Server zu betreiben. Am Server befinden sich u.a. auch ein CD-ROM Laufwerk, ein Syquest Wechselplatten-Laufwerk und ein HSD Scanner mit jeweils verschiedenen SCSI IDs und eine interne Festplatte mit 105 MB (SCSI-ID=1). Nun versuchte ich, von der externen Fujitsu-Platte mit ID 0 zu booten, und das Drama nahm seinen Lauf. Beim Booten im *verbose mode* erschien immer wieder die gleiche SCSI Fehlermeldung. Der Bootvorgang war einfach nicht möglich. Nun tauschte ich SCSI-Kabel und externe Gerätereihenfolge in allen möglichen Kombinationen, zum Schluss verfing sich der NeXT in einem SCSI-Trap und irgendwelchen kodierten Adressziffern als Fehlermeldung.

Nachdem ich nun alle SCSI-Geräte entfernt hatte um den Rechner überhaupt nochmals zu starten (von der internen Festplatte), ging nichts mehr. Fehlermeldung: *«no SCSI disk»*.

Mir blieb nichts anderes übrig, als von CD-ROM das Filesystem auf der internen Festplatte neu aufzuspielen. Danach schloss ich als zusätzliches SCSI-Gerät die neue externe Festplatte (mit SCSI-ID=0) wieder an und nun konnte von dieser Platte gebootet werden. Ich fand letztlich heraus, dass für einen korrekten Bootvorgang nicht mehr als drei externe SCSI-Geräte in Verbindung mit der Fujitsu-Platte angeschlossen sein dürfen. Ich nehme an, dass andernfalls eine korrekte Terminierung (Impedanz-Wert) aus irgendwelchen Gründen nicht ermöglicht wird.

Weiterhin ist zu beachten, dass die Schalter am Dip-Block SW1 der Fujitsu-Festplatte folgende Konfiguration erhalten:

1	off	SCSI-2 Modus
2	off	
3	on	
4	on	
5	on	
6	off	Synchronous mode data transfer request
7	on	
8	on	

Dip-Schalter 6 muss in der Stellung *off* stehen, denn sonst erscheinen immer wieder SCSI Fehlermeldungen beim Bootvorgang.

Monitor-Probleme

Ja, hatte ich auch, nicht mit dem 17" MegaPixel Display, sondern mit dem Color-Monitor. Es passierte bis zum Juni vergangenen Jahres häufiger, dass der Monitor beim ersten Einschalten kein Bild erzeugte. Über neun Monate lang machte sich zudem auch ein leichtes Flackern unter horizontaler Stauchung des Bildes bemerkbar. Bis zu dem Tag, als das Flackern doch sehr heftig wurde und das Bild nur noch 5 cm breit war. Stunden später nur noch eine nette schwarze Grafik, danach war es um den Monitor geschehen. Die Firma d'Art in Hamburg nahm sich des Problems an und reparierte den Bildschirm (innerhalb von zwei Monaten!). Deren Diagnose: Netzteil defekt, keine HV, keine Synchronisation, vertikale und horizontale Ablenkung nicht gegeben. Folgende Reparaturen wurden vorgenommen: Austausch der RGB-Platine, Preis: runde 1000.- DM.

CompuServe

Schon lange verfolge ich auch von Zeit zu Zeit die Nachrichten im NeXT Forum von *CompuServe*. Schon lang begleitet mich der Wunsch, auch mal eine Datei downzuladen oder auch mal eine eigene Nachricht abzusetzen. Leider fehlt mir das entsprechende Kommunikationsprogramm. Ich besitze ein 2400 Baud Modem von DoveFax und arbeite mit dem Unix Programm *tip.* Entsprechende Kommunikationsprogramme gibt es ja bei *CompuServe* in der Library des NeXT-Forums, aber wie soll ich mir diese downloaden, wenn ich noch kein entsprechedes Programm besitze?

Es wäre schön, wenn ihr mir weiterhelfen könnt bei der Bereitstellung eines Public Domain Kommunikationsprogramms mit einem guten Übertragungsprotokoll.

Dr. Klaus Kunze
Am Kieselhumes 86
D-66123 Saarbrücken
Deutschland

PS: Den ***PowerKey*** wird es doch weiterhin geben, oder?

Anmerkung zum letzten Abschnitt des Leserbriefes von Neil Franklin, Präsident NiCE:
Mein Tip dazu ist, dass wenn Du es ernst meinst mit «on-line» sein, dass Du vom NiCE Internet-Angebot gebrauch machst. Es bietet Dir Email (ca. 80 NiCE-Mitglieder und 10 Millionen Personen weltweit), News (wie CompuServe-Foren, aber gut 3000 davon, 60MB am Tag), Files zum runterladen (einige Terabyte) und vieles mehr. Du wirst aber ein schnelleres Modem benötigen (mind. 9600Baud).

Anmerkung der Redaktion:
Zum Thema Weiterbestehen des ***PowerKey*** verweise ich auf das Editorial dieser Ausgabe.

Impressum

Herausgeber: NiCE – NeXT User Group

PowerKey ist das Magazin der NiCE und erscheint vier mal jährlich. Ein Abonnement ist in der Mitgliedschaft bei der NiCE enthalten. ***PowerKey*** wird vollständig auf NeXT-Computern mit *WriteNow* erstellt.
Auflage: 250 Exemplare • Einzelverkaufspreis: Fr. 7.–

Redaktion:
Verantwortlicher Redaktor: Dominik Moser
Mitarbeiter dieser Ausgabe: W. Burri, A. Gabaglio, G. Fankhauser
«Wir bemühen uns, sowohl die männliche als auch die weibliche Schreibform zu verwenden. Wo wir dies zugunsten einer besseren Lesbarkeit nicht tun, beziehen sich sämtliche Aussagen auf Männer und Frauen.»

Redaktionsadresse:
Dominik Moser, Dörflistrasse 41, 8942 Oberrieden
Anfragen und Inserate von Mitgliedern bitte nur schriftlich.
Adressänderungen bitte an den Aktuar.

Anzeigenpreise:
8 × 5 cm Fr. 60.–, 8 × 10 cm Fr. 100.–, 16 × 10 / 8 × 20 cm Fr. 175.–, 1 Seite A4 Fr. 300.–, Mengenrabatt bereits ab 2 Ausgaben!
Einmalige, nicht gewerbsmässige Inserate von Mitgliedern gratis.

NiCE – NeXT User Group

Vereinsadresse: NiCE – NeXT User Group
Rechenzentrum
ETH Zentrum
8092 Zürich

PC-Konto: 80-46102-05

Vorstand

Präsident: Neil Franklin
Morgenweg 8, 8404 Winterthur

Vizepräsident: Christian Limpach
Mainaustr. 44, 8008 Zürich

Aktuar: Albin Mächler
Jonas Furrer-Str. 97, 8400 Winterthur

Kassier: Werner Burri
Chisweg 17, 5313 Klingnau

Redaktor: Dominik Moser
Dörflistr. 41, 8942 Oberrieden

Beisitzer: Adriano Gabaglio
Brunnmattstr. 22a, 6010 Kriens

Beisitzer: Patrik Lori
Schumacherweg 44, 8046 Zürich

Verleger: vakant

e-mail Vorstand oder einzeln: Vorstand bzw. <Vorname>.<Name>@nice.usergroup.ethz.ch

Mailbox: Tel. 01 251 20 02
```
# call nice
nice login: mailbox
```

PowerKey 2/94 **erscheint Ende Juni 1994**
Redaktions- und Anzeigenschluss: 15. Juni 1994