Einführung in kryptographische Methoden

Stefan Schumacher
<stefan@net-tex.de>, PGP: 0xB3FBAE33
http://www.net-tex.de/home/lect.html#krypto

20. September 2005

$Id: report.tex,v 1.17 2005/09/20 18:21:14 stefan Exp $

Inhaltsverzeichnis

1 Warum sollte man Kryptographie einsetzen? 2

2 Prüfsummen 2
2.1 md5(1) . . . 3
2.2 mtree . . . 4
2.3 verifiedexec(4) . . . 4

3 symmetrische Verfahren 5
3.1 mcrypt(1) . . . 5
3.2 cfs . . . 5
3.3 cgd . . . 6

4 asymmetrische Verfahren 6
4.1 Ein asymmetrisches Verfahren genauer betrachtet: RSA . . . 7

5 kryptographische Signaturen 7
5.1 Schlüsselsignatur . . . 9

6 weitere Links 10

Abstract

Dieser Artikel soll eine kleine Einführung in die Kryptographie bieten und mit den Prinzipien und einigen Anwendungsprogrammen vertraut machen. Der Vortrag dazu lief bereits auf den 6. Chemnitzer Linux-Tagen, dem 21. Chaos Communication Congress und dem Dresdner Linux Info-Tag 2005.
Die Programme beziehen sich auf NetBSD, sind aber fast alle auch auf anderen Unixsystemen wie FreeBSD, OpenBSD, HP/UX oder Solaris verfügbar.
Es liegt nicht in meiner Absicht ein mathematisches Papier zu verfassen - das können andere Leute wesentlich besser als ich. Ich werde lediglich einige kleinere mathematische Verfahren ansprechen, die zum Verständnis als *Anwender* notwendig sind. Weiterhin möchte ich in diesem Papier auch keine Kryptoalgorithmen analysieren oder bewerten.

1 Warum sollte man Kryptographie einsetzen?

Kryptographie ist z. Zt. das wohl einzige Mittel, um zuverlässig die Privatsphäre zu schützen oder die Integrität von Daten sicherzustellen. Jede eMail oder Datei, die man verschickt, wird im Klartext versand und ist somit - wie eine Postkarte - lesbar. Die Administratoren aller Mailserver oder Gateways die die eMail passiert, können diese lesen oder noch schlimmer - manipulieren. Es ist für den Empfänger sogar unmöglich nachzuweisen, daß die eMail auf dem Wege manipuliert wurde.

> **Das Briefgeheimnis sowie das Post- und Fernmeldegeheimnis sind unverletzlich.**
> Grundgesetz, Artikel 10, Abs 1.

Trotzdem versuchen insbesondere Politiker, die nicht den geringsten mathematischen oder gar technischen Sachverstand haben, kryptographische Methoden verbieten zu lassen, um somit der faschistoiden Überwachung der Bevölkerung weiter Vorschub zu leisten.

Kryptographische Methoden sind desweiteren auch hervorragende Mittel um Rechnersysteme vor Manipulation und Einbruch zu schützen bzw. diesen nachzuweisen. Man kann mittels kryptographischer Methoden wie z. B. der kryptographischen Signatur die Integrität von Dateien prüfen und so Manipulationen an diesen ausschließen oder gar das Ausführen manipulierter Binaries verhindern - und damit das Einspielen von Rootkits unmöglich machen.

2 Prüfsummen

Prüfsummen sind ein Verfahren, in der über eine Basiszahl, die die Information widergibt, nach einer bestimmten mathematischen Vorschrift eine

Prüfziffer gebildet wird.

Dies diente ursprünglich dazu, die Übertragung von Daten in Netzwerken sicherzustellen, da der Empfänger nach Erhalt der Daten und der Prüfziffer erneut die Prüfziffer berechnete und mit der Erhaltenen verglich. Stimmten beide überein verlief die Übertragung problemlos.

Prüfsummen begegnen uns in der heutigen Welt fast ständig, denn sie werden beispielsweise in der ISBN, EAN13, oder als Personalausweisnummer eingesetzt.

Als einfaches Beispiel möchte ich hier die EAN13 beschreiben, die EAN ist die Europäische Artikelnummmer, also die Zahl, die in der Regel unter dem Strichcode steht, der über Scannerkassen gezogen wird.

Um den häufigsten Eingabefehler, nämlich Zahlendreher, zu verhindern werden die ersten 12 Ziffern, die die Basisinformationen enthalten, gewichtet und addiert:

$$a + 3b + c + 3d + e + 3f + g + 3h + i + 3j + k + 3l = I$$

Man erhält nun daraus die Summe I, deren Differenz zum nächsten Vielfachen von 10 berechnet und als Prüfziffer m angefügt wird.

Ein einfaches Beispiel an einer echten EAN13:

$$EAN = 4025839396595$$

$$4 + 3 * 0 + 2 + 3 * 5 + 8 + 3 * 3 + 9 + 3 * 3 + 9 + 3 * 6 + 5 + 3 * 9 = 115$$

$$120 - 115 = m = 5$$

Die Kassensysteme berechnen bei der Eingabe einer EAN nun diese Prüfziffer erneut und können so falsche Eingaben aufdecken.

2.1 `md5(1)`

`md5` ist in RFC1321 definiert und erzeugt nach einem mathematisch recht anspruchsvollen Verfahren 128 Bit Prüfsummen. Der Algorithmus ist als sogenannte Falltürfunktion konzipiert, d. h. man kann zwar vom Datum aus eine Prüfsumme erzeugen, zu einer gegebenen Prüfsumme das entsprechende Datum aber nicht konstruieren.

Dies prädestiniert `md5` unter anderem auch als Passworthash, indem man beim Anlegen eines Passworts mittels `passwd(1)` das übergebene Passwort mit `md5` in eine Prüfsumme überführt und diese Prüfsumme abspeichert. Selbige Prozedur wird angewand so man sich einloggen will und es wird der neu erzeugte Hash mit dem Abgespeicherten verglichen und bei entsprechender Übereinstimmung der Hashs das Login durchgeführt. Der Vorteil bei diesem Verfahren ist, daß das Passwort nicht im Klartext auf der Festplatte liegt. Sollte also ein Einbrecher die Passwortdatei (`/etc/master.passwd`) erbeuten, verfügt er nicht über die Passwörter der Benutzer, sondern lediglich über die entsprechenden Hashs.

Auch wenn es (zum jetzigen Zeitpunkt) keinen Weg gibt aus den erbeuteten Hashs die Passwörter zu rekonstruieren, ist es aber ohne weiteres möglich eine Wörterbuchattacke zu fahren, in dem man einfach ein Wörterbuch (bspw. /usr/share/dict/words) an `md5` verfüttert und die so erzeugten Hashs mit den erbeuteten aus der Passwortdatei vergleicht.

Eine weitere klassische Anwendung ist der Einsatz von `md5` als Prüfsumme insbesondere auf FTP-Servern, so wird in der Regel neben der eigentlichen Datei noch eine Datei mit Prüfsummen zur Verfügung gestellt, die eine Überprüfung der Datei nach dem Download ermöglicht.

2.2 mtree

`mtree` wurde als Dateisystemintegritätsprüfer entwickelt um zu prüfen ob Installation erfolgreich abgeschlossen wurden. Dazu wurde eine einfache Datenbank, in der Dateiname, Dateistatus und Prüfsumme über der Datei abgelegt werden, erzeugt. Diese Datenbank, eine einfache strukturierte Textdatei, wurde mit dem neuen System verglichen.

Sofern man diese initiale Datenbank entsprechend schützt (z. B. auf einem schreibgeschützten Medium oder verschlüsselt), kann man diese Datenbank mit dem aktuellen System Datenbank vergleichen. Weicht die Prüfsumme bzw. der Status einer Datei ab, wurde diese offensichtlich manipuliert.

Ziel des Tests ist es, insbesondere Rootkits zu enttarnen, diese Kits werden in der Regel nach einem erfolgreichen Einbruch installiert und dienen dazu die Anwesenheit eines Einbrechers zu verschleiern. Meist werden hierzu manipulierte Binaries wichtiger Systemprogramme installiert, um z. B. zu verhindern daß `ps -ax` die laufenden Prozesse des Einbrechers anzeigt. Nachteilig ist hierbei allerdings das `mtree` reaktiv funktioniert, d. h. es gelingt dem Einbrecher das Rootkit zu installieren und er wird erst beim nächsten Durchlauf von `mtree` enttarnt.

2.3 `verifiedexec(4)`

Ähnlich wie AIDE, allerdings proaktiv, funktioniert `verifiedexec`. `verifiedexec` implementiert ab NetBSD 2.0 (bzw. 1.6-current) einen Prüfsummenvergleich in den `exec()`-Pfad des Kernels. Dazu wird ebenfalls eine initiale Datenbank (bspw. mit `/usr/share/examples/veriexecctl/gen_sha1`) erzeugt und im Bootprozess via `veriexecctl(8)` in den Kernelspeicher geladen. Ruft man nun ein Binary auf, wird die Prüfsumme erneut gebildet und mit der Datenbank verglichen. Im Securitylevel 1 warnt der Kernel bei abweichender Prüfsumme vor Ausführung, im Level 2 wird die Ausführung des Binaries verweigert.

3 symmetrische Verfahren

Symmetrische Verschlüsselungsverfahren verwenden sowohl zum Verschlüsseln als auch zum Entschlüsseln den selben Schlüssel. Dies bedeutet im Falle eines Nachrichtenaustauschs, das neben dem Chiffre, das die Nachricht enthält auch der entsprechende Schlüssel ausgetauscht werden muss.

Dies bedeutet im Falle einer einfachen eMail zwischen Alice und Bob, die bspw. mit `mcrypt(1)` `(siehe Kapitel 3.1)` und dem Passwort $GEHEIM$ verschlüsselt wurde, das dieses Passwort über einen sicheren Kanal ausgetauscht werden muss. In der Praxis bedeutet dies daß das unbefugte Abhören und Verändern des Schlüssels verhindert werden muss - was quasi unmöglich ist.

Ein weiteres Manko der symmetrischen Verfahren ist die Anzahl benötigter Schlüssel in der Kommunikation größerer Gruppen, man benötigt dort nämlich $n * (n-1)/2$ Schlüssel, was bei 10 Personen exakt 45 Schlüssel sind.

Sinnvoll ist die symmetrische Kryptographie jedoch, wenn man die Schlüssel nicht über unsichere Kanäle austauschen will, man z. B. das eigene Dateisystem oder Backups lokal verschlüsseln möchte.

3.1 `mcrypt(1)`

`mcrypt` umfasst ein Nutzerprogramm und eine entsprechende Bibliothek, die mehrere symmetrische Kryptoverfahren zur Verfügung stellt. `mcrypt` lässt sich recht einfach bedienen und bietet eine Vielzahl an Algorithmen und Modi. Die mitgelieferte Bibliothek `libmcrypt` bietet Entwicklerzugriff auf eine Vielzahl symmetrischer Kryptoalgorithmen von DES bis AES.

3.2 cfs

cfs ist das Cryptographic Filesystem von Matt Blaze, welches als erstes Dateisystemverschlüsselung implementierte und auf der Basis von NFS arbeitet. Dies bedeutet, das komplette Verzeichnisse und deren Inhalt verschlüsselt im Dateisystem abgelegt werden und bei Bedarf über den `cfsd`, der auf `localhost` lauscht, unverschlüsselt an das anfragende Programm übergeben. Soll die Datei zurückgeschrieben werden geschieht dies wieder über den `cfsd`, der die Datei entsprechend verschlüsselt in das Dateisystem schreibt.
Verzeichnisse, die mit cfsd verschlüsselt wurden, lassen sich auf Wechseldatenträger kopieren und so zwischen verschiedenen Systemen migrieren. Ebenso ist möglich die verschlüsselten Verzeichnisse per NFS von einem Server zu importieren und lokal zu entschlüsseln.

3.3 cgd

cgd ist der cryptographic devicedriver, der ab NetBSD 2.0 verschlüsselte Partitionen auf Blockebene ermöglicht. Hierbei wird der Zugriff auf die echte Partition über einen Pseudogerätetreiber abgewickelt, der die Daten entschlüsselt an das lesende Programm schickt und verschlüsselt auf die Festplatte schreibt.
cgd verwendet dazu symmetrische Kryptoverfahren mit einem einfachen Passwort, das zum Einbinden der echten Partition in den Pseudogerätetreiber übergeben werden muss. Zusätzlich kann cgd auch beim booten einen Zufallsschlüssel für eine Partition generieren, der nach dem shutdown verloren geht. Dieses Verfahren bietet sich für die Swappartition an, da deren Daten nach dem Herunterfahren des System nicht mehr von Bedeutung sind.

4 asymmetrische Verfahren

Ein asymmetrisches Verfahren wird durch die Verwendung eines Schlüsselpaares, bestehend aus öffentlichem und privaten Schlüssel, gekennzeichnet. Zur Erzeugung des Schlüsselpaares gibt es verschiedene Möglichkeiten, die verbreitetsten sind RSA und DSA, welche auf der Faktorisiserung des Produkts zweier Primzahlen basieren oder ElGamal bzw. Diffie/Hellman, die zwei große Zahlen potenzieren und den diskreten Logarithmus dieser Zahl verwenden.

Die so erzeugten Schlüssel ermöglichen eine komfortable asymmetrische Kommunikation, da ein Beteiligter nur den öffentlichen Schlüssel des jeweiligen Kommunikationspartners benötigt und dieser bedenkenlos über unsichere Kanäle verschickt werden kann.

Prinzipell ist die Verwendung asymmetrischer Kryptoverfahren recht einfach:
Alice und Bob haben je einen öffentlichen $(A_{\ddot{o}}, B_{\ddot{o}})$ und einen privaten (A_p, B_p) Schlüssel erstellt und möchten miteinander verschlüsselt kommunizieren:

$$Chiffrierung(Klartext + A_p + B_{\ddot{o}}) \rightsquigarrow Code$$

Alice verwendet ihren eigenen privaten Schlüssel und den öffentlichen Schlüssel von Bob um einen kryptographischen Algorithmus über den Daten anzuwenden. Nachdem sie das Chiffre, zusammen mit ihrem öffentlichen Schlüssel, an Bob geschickt hat, kann dieser das Chiffre mit seinem privaten und Alice öffentlichen Schlüssel wieder in Klartext wandeln:

$$Dechiffrierung(Code + B_p + A_{\ddot{o}}) \rightsquigarrow Klartext$$

4.1 Ein asymmetrisches Verfahren genauer betrachtet: RSA

Durch das entwickelte Prinzip des öffentlichen und privaten Schlüssels ist es möglich geworden verschlüsselte Daten und öffentlichen Schlüssel über ungesicherte Kanäle auszutauschen und so starke kryptographische Verfahren quasi für Jederman zugänglich zu machen. RSA wird unter anderem in `OpenSSH`, `GnuPG` oder dem `NetBSD-cgd` verwendet und ist offiziell in `RFC3447`[1] definiert.

Grundlage des Verfahrens ist das Erzeugen eines Schlüsselpaares welches dann zur Verschlüsselung der Daten verwendet wird. Erzeugt werden die Schlüssel in dem die zufälligen Primzahlen p und q generiert werden. Aus diesen Primzahlen wird die Modulzahl $modulus$ gemäß Vorschrift $modulus = p * q$ sowie phi mit $phi = (p-1) * (q-1)$ erzeugt. Danach generiert man eine große (empfohlen werden vom US DoD mindestens 1024 Bit) Primzahl e welche als privater Schlüssel genutzt wird und daher geheim zu halten ist. Abschließend wird der öffentliche Schlüssel d mittels Auflösen der Gleichung $(e * d) mod\ phi\ =\ 1$ nach d erzeugt. Da mit der Kenntnis von p und q die Zahlen $modulus$ und phi berechnet werden können ist es somit auch möglich den privaten Schlüssel zu generieren, daher ist es notwendig diese Zahlen entsprechend zu schützen bzw. nach Erzeugung des Systems zu vernichten. Die Verschlüsselung eines Textes erfolgt mittels

$$Chiffre\ =\ (Klartext^{\ öffentlicher\ Schlüssel}) mod\ modulus$$

Die Entschlüsselung vollzieht sich also folgendermaßen:

$$entschlüsselt\ =\ (Chiffre^{\ privater\ Schlüssel}) mod\ modulus$$

Die Implementierung des RSA Algorithmus ermöglicht es eine gewünschte Bitlänge zur Erzeugung von p und q zu übergeben. Dies ist wünschenswert, da nur eine entsprechend hohe Bitlänge (mindestens 1024 Bit, besser noch 2048) die Sicherheit des Schlüsselsystems ermöglicht, denn der Aufwand zur Zerlegung von phi in die entsprechenden Primfaktoren ist zu hoch.

5 kryptographische Signaturen

Die kryptographischen Signaturen bilden einen Sonderfall der asymmetrischen Verfahren. Mit ihnen ist es möglich neben der Integrität der signierten Daten auch die Integrität und Authenzität der Prüfsumme sicherzustellen. Hierzu wird vom Absender eine Prüfsumme erzeugt und mit dem eigenen privaten Schlüssel verschlüsselt, so daß das Chiffre mithilfe des öffentlichen Schlüssels entschlüsselt werden kann.

[1] ftp://ftp.rfc-editor.org/in-notes/rfc3447.txt

Schematisch funktioniert eine Signatur, die von Alice durchgeführt wird so:
Alice bildet eine Prüfsumme über ihre Nachricht und erhält:

$$h(N)$$

Dieser Hash entspricht der in Kapitel 2 besprochenen einfachen Prüfsumme. Um diese nun zu schützen, verschlüsselt Alice ihre Prüfsumme mit ihrem privaten Schlüssel A_p:

$$S = \{h(N)\}_{A_p}$$

Nun kann ein Empfänger der Nachricht, beispielsweise Bob, mit Hilfe des öffentlichen Schlüssels von Alice die Signatur und die damit signierten Daten überprüfen. Hierzu bildet Bob zuerst selbst eine Prüfsumme über die Nachricht ($h'(N)$) und entschlüsselt anschließend die Signatur S mit

$$A_{\ddot{o}} \rightsquigarrow h(N)$$

und erhält somit offensichtlich die von Alice gebildete Prüfsumme $h(N)$. Nun ist lediglich zu prüfen ob $h'(N) = h(N)$ gilt. Wenn dem so ist, sind Nachricht und Signatur korrekt.

Als Beispiel ein von mir mit GnuPG mit `0xB3FBAE33` signierter Klartext: Der Block wird mit `-----BEGIN PGP SIGNED MESSAGE-----` ein- und `-----END PGP SIGNATURE-----` ausgeleitet. `Hash: SHA1` kennzeichnet das verwendete Prüfsummenverfahren als SHA1. Der Block beginnend mit `iD8D...` ist die Signatur und enthält auch die entsprechenden Informationen zu meinem Schlüssel.

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

Von Zeit zu Zeit seh ich den Alten gern,
Und hüte mich, mit ihm zu brechen.
Es ist gar hübsch von einem großen Herrn,
So mit dem Teufel selbst zu sprechen.
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.2.4 (NetBSD)

iD8DBQFAA8ySEfTEHrP7rjMRAmLxAJ0ePbnStfzK4NVSZFMSpvhhIp2NqACdEuFA
NChplstKIDO3nVEDEDdOUmA=
=+lsL
-----END PGP SIGNATURE-----
```

Jeder Text, der vor der Ein- bzw. nach der Ausleitung steht, wurde nicht signiert und sollte demzufolge auch als nicht vertrauenswürdig betrachtet werden!

5.1 Schlüsselsignatur

Auch auf dem diesjährigen Chaos Communication Congress wird eine Keysigningparty durchgeführt. Nun kann man sich verständlicherweise Fragen warum man einen PGP Schlüssel signieren sollte, da dieser doch offensichtlich keine Nachricht enthält. Dies ist aber so nicht korrekt, denn immerhin enthält der Schlüssel natürlich die Informationen über seinen Besitzer wie Name, Kommentar, eMailadresse, Verfallsdatum etc. und kann eben auch manipuliert bzw. gefälscht werden.

Der große Vorteil der asymmetrischen Verfahren ist ja bekanntermaßen die Möglichkeit der Kommunikation über unsichere Kanäle, d. h. es ist eben auch problemlos möglich, mit Jemandem den man nicht persönlich kennt verschlüsselt zu kommunizieren. Dazu ist es notwendig den öffentlichen Schlüssel des Kommunikationspartners zu importieren und zu verwenden. Doch wie kann ich in diesem Falle sicher sein, das der importierte Schlüssel auch wirklich dem gewünschten Partner gehört? Eigentlich gar nicht, es sei denn man verwendet eben die Signatur einer anderen Person oder Authorität als Bestätigung der korrekten Identität eines Schlüssels. Dies heißt in der Praxis eines PGP Keysignings zwischen Bob und Charlie nichts weiter, als das Bob die korrekte Identität Charlies prüft, bspw. anhand des Bundespersonalausweises, Charlie anschließend seinen Schlüsselfingerabdruck als Ausdruck an Bob übergibt und Bob nun den öffentlichen Schlüssel mit seinem Privaten signiert. Der signierte Schlüssel wird nun von Bob an Charlie verschlüsselt per eMail verschickt, so das Charlie sowohl Zugriff auf den privaten Schlüssel als auch auf den genannten eMailaccount haben muss.

Betrachtet man nun die Signaturen eines öffentlichen Schlüssels, kann man dessen Vertrauenswürdigkeit festlegen. Hierbei ist es natürlich dem Nutzer selbst überlassen inwiefern er den Signaturen vertraut, man sollte trotzdem davon ausgehen können, das eine Anzahl von Signaturen vertrauenswürdig ist.

Die in PGP eingesetzte Methode der gegenseitigen Signaturen wird gemeinhin auch als Web of Trust bezeichnet, da die graphische Darstellung der Signaturen einem einfachen gerichteten Graphen entspricht, hierbei geben die inzidenten Kanten eines Knoten die erhaltenen und die adjazenten Kanten die gegebenen Signaturen wieder. Es existieren Programme, wie gpg2dot.pl, um das Web als Graphen zu zeichnen. Ein Beispiel findet sich unter http://www.uni-magdeburg.de/steschum/local-web.jpg in einer ca. 1,3 MB grossen JPEG-Datei, die mein persönlichers Web of Trust darstellt.

Eine andere Form der Signaturverteilung ist der Einsatz von Zertifizierungsinstanzen (auch CA - Certificate Authorities) in einer Hierarchie. Hierbei wird eine oberste CA festgelegt, die weitere CAs siginiert, welche wiederum Endnutzer zertifizieren. Hierbei bildet sich kein beliebiger Graph, sondern ein Baum. Da jeder Baum bekanntermaßen exakt eine Wurzel und nur einen Weg von dieser zu einem Blatt hat, wirkt sich ein Vertrauensbruch in einer

CA auf alle Knoten des entsprechenden Unterbaumes aus.

6 weitere Links

Die Links finden sich auf der Webseite zum Vortrag unter
http://www.net-tex.de/home/lect.html#krypto
AIDE auf NetBSD (deutsch) (english)
Encrypted filesystem with cfs on NetBSD
using verified executables (english)
NetBSD cgd
Regenechsen.de - Einführung Kryptographie
pruefziffernberechnung.de
Deutsche GnuPG [GPG] Anleitung / Inhalt
PGP Anleitung vom Datenschutzzentrum.de
RFC1321 (md5)
RFC3174 (SHA1)
Heise Kryptokampagne
mcrypt for PHP
DFN SSL Handbuch
Programme
GnuPG
AIDE - Advanced Intrusion Detection Environment
mcrypt
Jetico Downloadseite (bcwipe, bestcrypt)
Transparent Cryptographic File System
sig2dot - generiert Web Of Trust aus Schlüsselring
OpenSSL
OpenSSH
Politik und Hintergründe
The WTC Conspiracy XXVIII: Bürorechner überleben WTC-Crash, Flugschreiber nicht!
CCC Presseerklärung: Zentrale Speicherung biometrischer Daten gefährdet Grundrechte
C4: Ein Chip, sie zu knechten...
Hacktivismo veröffentlicht Six/Four
Italien: Zwangsdatenspeicherung bis zu fünf Jahren
Datenspeicherung für 100 Jahre
Grüner Abgeordneter will Hinterlegung von Kryptoschlüsseln prüfen
CCC: CrypTron
7 Thesen zur Kryptodebatte anno domini 1997