Kryptographische Dateisysteme im Detail

Stefan Schumacher
Magdeburger Institut für Sicherheitsforschung
Stefan.Schumacher@Magdeburger-Institut-fuer-Sicherheitsforschung.de
http://www.Magdeburger-Institut-fuer-Sicherheitsforschung.de

Kryptographische Dateisysteme sind ein wirksames Mittel, um Dateien vor fremden Zugriffen zu schützen. Aber gerade in einem größeren Umfeld werfen diese Dateisysteme einige Probleme auf, z. B. bei der Datensicherung oder bezüglich Schlüsselhinterlegung und Zugriffe über mehrere Benutzer. Dieser Artikel stellt einige grundlegende Probleme sowie Details kryptographischer Dateisysteme im Vergleich vor.

Arten der kryptografischen Dateisysteme

Um Daten effektiv vor unerlaubten Zugriffen zu schützen, wurden kryptographische Dateisysteme entwickelt. Diese sollen idealerweise ohne großen Aufwand Dateien, die auf einem Datenträger liegen zuverlässig verschlüsseln und dem Anwender bzw. den Anwendungsprogrammen möglichst transparent unverschlüsselt zur Verfügung stellen. Diese Anforderungen können auf verschiedene Arten und Weisen umgesetzt werden, die jeweils ihre spezifischen Vor- und Nachteile haben. Diese möchte ich im folgenden Artikel vorstellen.

Eine einfache Möglichkeit Dateien zu verschlüsseln ist die einzelne Verschlüsselung durch ein Verschlüsselungsprogramm. Dazu muss jede Datei einzeln verschlüsselt werden und die unverschlüsselte Datei sicher gelöscht werden. Möchte man auf eine Datei zugreifen, muss sie entschlüsselt werden. Nach Beendigung des Zugriffs muss sie wieder verschlüsselt und die Klartextdatei sicher gelöscht werden. Zur Verschlüsselung können verbreitete und damit portable Verschlüsselungsprogramme wie GnuPG, Mcrypt oder OpenSSL eingesetzt werden, die auf unterschiedlichen Betriebssystemen laufen. Allerdings ist es sehr aufwändig die Dateien zu ver- und entschlüsseln, da der Anwender die Aufgaben alle von Hand erledigen muss. Es funktioniert daher nur sinnvoll wenn es sich um wenige Dateien handelt, die selten benutzt werden bspw. speziell zu schützende Dateien. Oder wenn es zwingend erforderlich ist, von unterschiedlichen Betriebssystemen auf die Dateien zuzugreifen.

Wesentlich komfortabler ist es, Dateien von einem Programm automatisch ver- und entschlüsseln zu lassen, wenn auf diese zugegriffen werden muss. Dies kann entweder durch einen Dæmon geschehen, der Zugriffe auf Dateien im Dateisystem abfängt und auf einen verschlüsselnden Layer umlenkt, oder in dem man die Festplatte bzw. eine Partition davon auf Blockebene verschlüsselt und die Verschlüsselung unterhalb des Dateisystems durchschleift. Abb. 1 zeigt 3 Varianten der Verschlüsselung mittels CFS und CGD, wobei CGD einmal auf Blockebene und einmal als Container arbeitet.

Beide Varianten haben ihre spezifischen Vor- und Nachteile. Eine Verschlüsselung auf Blockebene, wie sie CGD, GBDE, Geli oder loop-aes einsetzen, verhält sich nach dem Einbinden für den Benutzer transparent. Außer für eine Passworteingabe beim einmounten merkt der Benutzer in der Regel nicht, dass seine Partition verschlüsselt ist.

Nachteilig ist hierbei die Granularität der Verschlüsselung. Es kann immer nur ein Block-Device, also eine Partition verschlüsselt werden. Möchte man die Zugriffe auf Dateien feiner granuliert steuern, reichen Block-Devices nicht aus.

Es sei denn, man verwendet als Workaround Dateisystem-Container, die man mit einem Block-Device verschlüsselt. Dazu erstellt man eine Datei, die als Container dient und über ein Loopback-Device wie vnconfig(8) eingemountet wird. In diesem Container wird mit dem Verschlüsselungssystem die »Pseudo-Block-Ebene« verschlüsselt und entsprechend eingemountet, siehe Abb. 1(b)

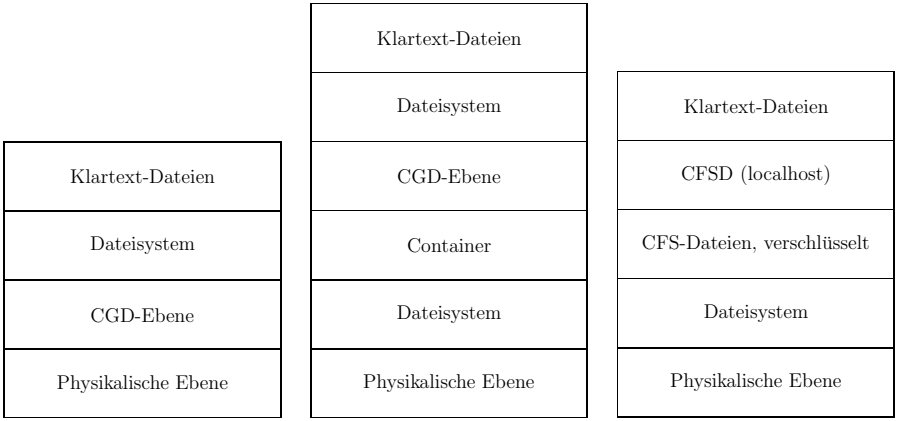
Vorteil dieser Variante ist die feinere Granularität: man kann bspw. jedem Benutzer einen eigenen Container in seinem Home-Verzeichnis zur Verfügung stellen oder auf einem Laptop verschiedene Container für verschiedene Daten einbinden. Beispielsweise einen Container für private E-Mails und den privaten GnuPG-Schlüsselring und einen Container für die beruflichen E-Mails und den beruflichen GnuPG-Schlüsselring. Je nach Situation bindet man nur den entsprechenden Container ein.

Ähnlich arbeiten Systeme, bei denen ein Dæmon die Zugriffe auf die Dateien im Dateisystem abfängt und die entschlüsselte Dateivariante dem Anwendungsprogramm zur Verfügung stellt. Dabei liegen die Quelldateien verschlüsselt im normalen Dateisystem und können theoretisch auch mit normalen Programmen wie vi bearbeitet werden. Dies ist aber nicht sinnvoll, da die Dateien verschlüsselt sind. Möchte der Anwender auf die Dateien unverschlüsselt zugreifen, fängt der entsprechende Dæmon den Zugriff ab und leitet ihn transparent auf das Entschlüsselungssystem um. Das Anwendungsprogramm erhält also die unverschlüsselten Daten ausgeliefert.

Vorteil dieser Variante ist die noch höhere Granularität gegenüber der Container-Variante, da hier im Prinzip jeder einzelnen Datei ein eigener Schlüssel zugeordnet werden kann. In der Praxis arbeiten die die meisten dieser Systeme aber auf Verzeichnisniveau, d.h. ein Verzeichnis wird rekursiv samt aller darin enthaltenen Dateien und Unterverzeichnisse ver- und entschlüsselt.

Alle drei Varianten haben verschiedene Vor- und Nachteile. Die Verschlüsselung auf Blockebene läuft weitestgehend transparent ab. Außer der Eingabe des Passwortes, die natürlich bei allen Verschlüsselungsprogrammen anfällt, bemerkt der Benutzer in der Regel nicht, dass die Partition mit seinen Daten verschlüsselt ist.

Umständlich kann es sein, die entsprechenden Partition im laufenden Rechner auszumounten, bspw. wenn man ihn per ACPI schlafen legen möchte. Es kann dann unter Umständen etwas umständlich sein, die Partition auszumounten, bspw. wenn Dienste wie PostgreSQL oder Apache auf sie zugreifen. Dies geht mit Containern



(a) CGD auf Blockebene einer Partition

(b) CGD auf Pseudo-Blockebene eines Container

(c) CFS im Dateisystem

Abbildung 1: Verschlüsselung mit CGD und CFS auf Block- bzw. Container-Ebene

oder Dæmonen einfacher, da man hier nur die entsprechenden Verzeichnisse bzw. Dateien schützen und ggf. ausmounten muss.

Problematisch ist es, verschlüsselnde Dateisysteme zu sichern. So ist es nicht möglich, verschlüsselte Partitionen auf Blockebene einfach mit dump(8) oder Bacula direkt zu sichern. Die verschlüsselte Partition muss immer von einem Anwender eingebunden werden, damit das eingebundene Pseudogerät gesichert werden kann. Es ist also immer der Eingriff eines Benutzers erforderlich, automatisierte Pull-Verfahren funktionieren hier nicht. Das Sicherungsarchiv ist dann gesondert zu schützen, also zu verschlüsseln.

Container bzw. Dæmonsysteme sind hier wesentlich einfacher zu handhaben, da die Container bzw. verschlüsselten Dateien direkt gesichert werden können. Sie sind dann bereits im Sicherungsarchiv verschlüsselt, eine gesonderte Verschlüsselung des Archivs ist daher nicht notwendig, aber empfohlen.

Nachteilig bei Containern ist hierbei die feste Größe, da der Container in der Regel weniger Nutzlast enthält, als er eigentlich groß ist. Wenn ein Container bspw. 700MB groß ist, aber nur 5MB Nutzdaten enthält, werden 695MB »Luft« gesichert. Bei einem System mag das tolerabel sein, in größeren Netzen mit hunderten von Benutzer in der Regel nicht mehr. Daher kann es auch hier erforderlich sein, die Container einzumounten und die unverschlüsselten Dateien direkt zu sichern. Praktischer sind in dieser Hinsicht Dæmonen-Systeme, da hier nur die Nutzdaten plus ein wenig Overhead anfallen.

Praktisch sind Container und Dæmonen auch bezüglich Zugriffe über Netz, so ist

es problemlos möglich, einen Container oder CFS-verschlüsselte Dateien von einem NFS-Server zu importieren und erst lokal zu entschlüsseln. Somit gehen die Daten verschlüsselt durchs Netz und sind auf dem Übertragungsweg geschützt. Außerdem lassen sich so Daten auf Wechseldatenträgern schützen.

Der Vorteil der Dæmonsysteme hinsichtlich Datenmüll ist aber auch gleichzeitig ein Nachteil bezüglich der Kryptoanalyse. Dæmonsysteme offenbaren durch die Verschlüsselung einzelner Dateien einige Metadaten, wie bspw. die Dateianzahl und -größe in einem Verzeichnis, sowie die mtime, ctime und atime. Dies ermöglicht eventuell Rückschlüsse auf den Dateiinhalt.

Betriebsarten des Kryptalgorithmus

Ein wichtiges Detail bei der Implementierung von Block-Geräten ist die Verwendung des Block-Modus als Betriebsart. Auf Block-Geräten werden Block-Chiffre eingesetzt, also Kryptoalgorithmen, die den Klartext in feste Blöcke zerlegen und diese Blöcke verschlüsseln. Alternativ gibt es noch sogenannte Stromchiffre, die den Klartext und den Schlüssel als Eingabestrom erwarten, da Festplatten etc. aber immer eine Blockstruktur aufweisen, wäre es unsinnig, einen Stromchiffre zu verwenden.

Zwei der bekanntesten Betriebsarten von Blockchiffren sind der Electronic Code Book Mode (ECB) und der Cipher Block Chaining Mode (CBC). Beim ECB wird jeder Block einzeln verschlüsselt, die Blöcke selbst werden nicht miteinander gekoppelt. Außerdem bildet ein Chiffrblock immer auf den selben Klartextblock ab. Dadurch ist es möglich, wahlfrei auf einen bestimmten Block zuzugreifen, in dem der gewünschte Block direkt ver- oder entschlüsselt wird.

Ein großer Nachteil ist aber, das im ECB statistische Kryptanalysen möglich sind, denn großflächig zusammenhängende Bereiche im Klartext bleiben auch im Chiffrat zusammenhängend. Außerdem ergeben identische Klartextblöcke immer identische Chiffrate. Daher ist der ECB definitiv als unsicher zu bezeichnen.

Ein weiteres Problem sind Watermarking- bzw. Wasserzeichen-Angriffe. Dabei sorgt ein Angreifer dafür, das eine speziell präparierte Datei bspw. als E-Mail auf dem System landet. Ziel des Wasserzeichen-Angriffs ist es, nachzuweisen, dass eben diese Datei auf dem Gerät liegt.

Der CBC verkettet die Blöcke hingegen, in dem er den Klartextblock vor der Verschlüsselung mit dem vorhergehenden Chiffreblock per XOR verknüpft. Dadurch werden statistische Zusammenhänge, die sich über mehrere Blöcke verteilen verwischt.

Mathematisch erfolgt die Verschlüsselung rekursiv im CBC wie folgt:

$$\forall \left(i \in \mathbb{N}^{+}\right) : \left(\begin{array}{rcl} C_0 & = & E_K(P_0 \oplus \mathrm{IV}) \\ C_i & = & E_K(P_i \oplus C_{i-1}) \end{array} \right)$$

Da die Verschlüsselung rekursiv definiert ist, erzeugt eine Änderung im n. Block einen Lawineneffekt auf alle Blöcke größer n, denn durch die Änderung des n. Blockes ändert sich auch das Chiffrat des n. Blockes und damit der Initialisierungsvektor für den Block $n-1$. Und damit das Chiffrat $n+1$, welches wiederum als Initialisierungsvektor für $n+2$ dient. Daher werden in der Praxis die Datenblöcke in logische »Partitiönchen« aufgeteilt, so dass sich der Lawineneffekt in Grenzen hält und nicht über die gesamte Platte erstreckt.

Da der 1. Block keinen Vorgängerblock hat, muss für ihn ein Initialisierungsvektor erst generiert werden. Im Prinzip muss dieser Initialisierungsvektor nicht geheimgehalten werden, allerdings ist bei schlechter Wahl des IV z.B. durch einen Zeitstempel oder die Wahl der Sektornummer, ein Wasserzeichen-Angriff gegen das System möglich. Abhilfe schafft das ESSIV-Verfahren, bei dem der IV aus der verschlüsselten Sektornummer und einem Hash aus dem Passwort generiert wird. Da das Passwort geheim bleibt, ist der IV nicht vorhersagbar.

Die Entschlüsselung von CBC ist nicht rekursiv folgendermaßen definiert:

$$\begin{array}{rcl} P_0 & = & D_K(C_0) \oplus \mathrm{IV} \\ \forall \left(i \in \mathbb{N}^+\right) : (\quad P_i & = & D_K(C_i) \oplus C_{i-1} \quad) \end{array}$$

Das heißt für die Entschlüsselung wird der jeweilige Chiffrat-Block, das Vorgänger-Chiffrat und der Schlüssel benötigt. Ist der Chiffrat-Block n nicht mehr lesbar, kann daher auch der Block $n+1$ nicht mehr dechiffriert werden. Ist der Block $n+1$ aber noch lesbar, kann er wieder als IV für den Block $n+2$ dienen und damit der Block $n+2$ dechiffriert werden.

Schlüsselverwaltung

Bisher existieren noch keine kryptographischen Dateisysteme mit funktionierender Schlüsselverwaltung für mehrere Benutzer. Alle produktiv einsetzbaren Systeme setzen einfache symmetrische Verschlüsselungsverfahren meist mit einem Passwort ein. Das heißt in der Konsequenz, dass alle Benutzer, die Zugriff auf die verschlüsselten Dateien haben woll das Passwort kennen müssen. Wenn das Passwort geändert werden muss oder einem Benutzer die Zugriffsrechte entzogen werden sollen, müssen alle Benutzer ein neues Passwort bekommen – dass ihnen dann auf einem sicheren Kanal mitgeteilt werden muss. Bisher eignen sich kryptographische Dateisysteme daher nur für einige wenige Benutzer, wenn nicht nur für genau einen.

Es ist daher notwendig, komplexere Schlüsselverwaltungssysteme aufzubauen, die unter anderem Schlüsselhinterlegung und -widerruf sowie eine Einbindung von Datensicherungsverfahren ermöglichen.

Swap und Temporäre Verzeichnisse

Die Swappartition wird verwendet, um Daten aus dem Arbeitsspeicher auszulagern, bspw. wenn eine Anwendung mehr Speicher braucht als aktuell frei ist. Die kann natürlich die Sicherheit des Systems gefährden, wenn es sich dabei um eigentlich verschlüsselte Daten handelt, die unverschlüsselt im RAM liegen und ausgelagert werden. Bei einem Neustart des Systems wird zwar die /tmp-Partition bereinigt, die dort gelegenen Daten können aber problemlos rekonstruiert werden.

Damit dies nicht passiert, kann man auch die Swappartition entweder als RAM-Disk im Speicher anlegen oder mit auf Blockebene verschlüsseln. Im Prinzip geht man hier genauso vor wie bei einer normalen Datenpartition, lediglich die Schlüsselgenerierung wird abgeändert. CGD als Blocksystem unterstützt bspw. »urandomkey« als Schlüsselmethode. Hierbei werden einfach aus /dev/urandom, einem Pseudozufallsgenerator, Zeichen ausgelesen und als Schlüssel verwendet. Der Schlüssel wird dabei nicht gespeichert, so das beim Absturz des Systems oder nach dem Herunterfahren die Partition nicht mehr entschlüsselt werden kann. Dies ist aber bei einer Swappartition nicht notwendig, da sie bei jedem Systemstart neu initialisiert wird.

Das temporäre Verzeichnis /tmp legt man entweder in einer Ramdisk im Arbeitsspeicher an, oder verschlüsselt es ebenfalls auf Blockebene. Anders als bei der Swap-Partition muss in der Tmp-Partition aber ein Dateisystem angelegt werden. Dies kann man von einem Shellskript beim booten erledigen lassen. Da die Tmp-Partition in der Regel nicht besonders groß ist, dauert es auch nicht besonders lange.

Übersicht über verschlüsselnde Dateisysteme

CFS Cryptographic File System von Matt Blaze, veröffentlicht 1993. Der cfsd-Dæmon arbeitet ähnlich wie der NFS-Dæmon und entschlüsselt die verschlüsselt im Dateisystem liegenden Dateien. Ist für nahezu jedes Unix verfügbar und kann auch in Netzwerken bzw. auf Wechseldatenträgern eingesetzt werden. Mit Blowfish als Algorithmus hinreichend schnell und sicher.

CGD Cryptographic Disk Driver von Roland C. Dowdeswell. In NetBSD ab Version 2.0 verfügbar. Verschlüsselt ganze Partitionen auf Blockebene, kann Schlüssel aus /dev/urandom generieren und so Swap und Temp verschlüsseln. Unterstützt AES, 3DES und Blowfish jeweils in CBC. Schnell und transparent, nicht portabel. Kann via vnd auch in Containern eingesetzt werden.

Truecrypt Ursprünglich unter Windows entwickelt, arbeitet mit Containern oder auf Blockebene. Inzwischen auch auf Mac OS X und Linux lauffähig, daher zum Datenaustausch geeignet. Truecrypt arbeitet in der XEX-TCB-CTS-Betriebsart, welche in der NIST FIPS Pub 800-38E standardisiert ist.

GBDE steht für GEOM Based Disk Encryption, GEOM wiederum ist ein Plattenverwaltungsframework von FreeBSD. GBDE arbeitet auf Blockebene, verschlüsselt dabei aber jeden einzelnen Block mit einem *neuen* Zufallsschlüssel. Dadurch werden Watermarkingattacken erschwert, da Blöcke mit gleichem Inhalt durch den Zufallsschlüssel immer ein unterschiedliches Chiffrat ergeben.

GELI arbeitet wie GBDE im GEOM-Framework von FreeBSD auf Blockebene. Es bietet neben einem Zufallsschlüssel auch die Möglichkeit mehrere Schlüssel für eine Partition zu verwenden. Darüber hinaus verschlüsselt GELI Daten nicht nur, sondern kann auch mit verschiedenen Prüfsummen die Integrität der Dateien sicher stellen.

eCryptFS arbeitet ähnlich wie CFS im Dateisystem. Es ist ab Version 2.6.19. im Linux-Kernel enthalten. Enthält der Linux-Kernel Unterstützung für Public-Key Kryptographie, kann diese auch für eCryptFS eingesetzt werden. Das System legt alle Metadaten in der jeweiligen Datei ab, dadurch kann diese zwischen verschiedenen Dateisystemen ausgetauscht werden, außerdem unterstützt es mehrere Schlüssel für eine Datei bzw. Hierarchie. Damit können Daten bequem ausgetauscht werden.

EncFS basiert auf FUSE und verschlüsselt Dateien im Dateisystem. Es unterstützt die im Linux-Kernel vorhandenen Kryptalgorithmen und kann jede Datei mit einem eigenen Zufalls-IV verschlüsseln. Außerdem generiert EncFS eine 8-Bit-Prüfsumme für jede Datei, um Dateikorruptionen zu erkennen. EncFS ist auch für Mac OS X und NetBSD erhältlich, ließ sich aber zur Zeit auf einem NetBSD 5.1 nicht kompilieren.

CryptoFS basiert ebenfalls auf FUSE oder wahlweise LUFS und arbeitet ähnlich wie EncFS und CFS im Dateisystem, in dem es jede Datei einzeln verschlüsselt.

dm-crypt ist Teil des Linux-Kernels der 2.6er-Reihe. Es arbeitet im Device-Mapper-Framework und kann daher auf Blockebene einzelne Partitionen oder ganze Geräte, RAID-Volumes sowie einzelne Dateien verschlüsseln. Es ist eine weiterentwicklung von cryptoloop und unterstützt unter anderem XTS, LRW und ESSIV um Wasserzeichen-Angriffe zu vereiteln. Das System läuft neben Linux auch auf DragonFly BSD.

SD4L ScramDisk for Linux ist ein Linux-Programm, um ScramDisk-Container zu erzeugen und einzubinden. ScramDisk war ein Verschlüsselungsprogramm für Windows 9x. SD4L nutzt das Scramdisk-Containerformat und erzeugt bzw. mountet verschlüsselte Container fester Größe im Dateisystem.

BestCrypt ist ein kommerzielles Verschlüsselungsprogramm von Jetico für Windows und Linux. Es kann auf Blockebene komplette Datenträger oder Partitionen verschlüsseln oder Container im Dateisystem erzeugen.

Siehe auch:

- Stefan Schumacher (2006). „Verschlüsselte Dateisysteme für NetBSD“. Deutsch. In: *UpTimes* 4, S. 25–31. ISSN: 1860-7683. URL: http://kaishakunin.com/publ/guug-uptimes-cgd_cfs.pdf (besucht am 07. 11. 2009)
- Stefan Schumacher (2007). „Daten sicher löschen“. Deutsch. In: *UpTimes* 1, S. 7–16. ISSN: 1860-7683. URL: http://kaishakunin.com/publ/guug-uptimes-loeschen.pdf (besucht am 07. 11. 2009)
- Stefan Schumacher (2004). *Einführung in kryptographische Methoden*. URL: http://www.cryptomancer.de/21c3/21c3-kryptograhie-paper.pdf (besucht am 2005. 01. 06)

Über den Autor

Stefan Schumacher ist geschäftsführender Direktor des Magdeburger Instituts für Sicherheitsforschung und gibt zusammen mit Jan W. Meine das Magdeburger Journal zur Sicherheitsforschung heraus.

Er befasst sich seit über 15 Jahren mit Fragen der Informations- und Unternehmenssicherheit und erforscht Sicherheitsfragen aus pädagogisch/psychologischer Sicht. Seine Forschungsergebnisse stellt er regelmäßig auf internationalen Fachkongressen der Öffentlichkeit vor.

Zur Zeit organisiert er eine Ringvorlesung zur Informationstechnologie und Sicherheitspolitik an der Otto-von-Guericke-Universität Magdeburg und arbeitet an mehreren Publikationen zum Thema Cyber-War und Sicherheitsstrategien.

Darüber hinaus berät er Unternehmen bei der Umsetzung von Sicherheitsmaßnahmen und der Etablierung unternehmensweiter IT-Sicherheitsstrategien.