2012

Universität Hamburg

Florian Wilkens

[VERSCHLÜSSELUNG – PROSEMINAR: SPEICHER UND DATEISYSTEME]

Ein Ausarbeitung von Florian Wilkens im Rahmen des ProSeminars Speicher- und Dateisysteme 2012
an der Universität Hamburg

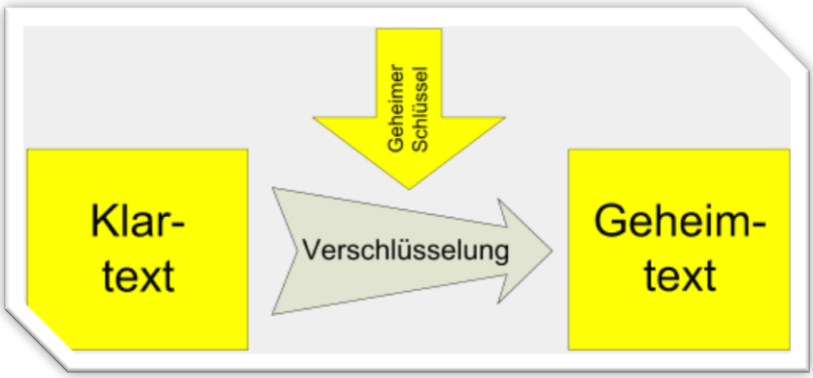
Inhaltsverzeichnis

Allgemeines zur Verschlüsselung

Einleitung

Im Folgenden werde ich unter verschiedenen Gesichtspunkten auf Verschlüsselung eingehen. Aber wie ist sie eigentlich formal definiert? Verschlüsselung bezeichnet grundsätzlich nur die Informationsumwandlung von Texten oder weitergehend auch allgemeinen Daten.

Aus einem Eingabetext (auch Klartext genannt) wird mithilfe von nicht näher definierten Methoden und Vorgehensweisen eine nicht mehr zu interpretierende Zeichenfolge (auch Geheimtext) generiert. Wichtig ist hierbei natürlich, dass das Verfahren in irgendeiner Weise wieder umgekehrt werden kann, sodass der Originaltext bei Bedarf wieder entschlüsselt werden kann. Während der Verschlüsselung wird hierbei meist eine je nach Implikation verschiedene Anzahl an sogenannten Schlüsseln verwendet. Dies sind im Wesentlichen Werte, die meist in Textform vorliegen und für die Verschlüsselung beziehungsweise Entschlüsselung benötigt werden. Es werden jedoch in der Regel nicht die Eigentlichen Zeichen verwendet, sondern (meist positive) Ganzzahlwerte, die dann in den zugrundeliegenden mathematischen Gleichungen in irgendeiner Form als Variablen eingesetzt werden können.



Motivation

Gründe für Verschlüsselung lassen sich gerade im Zeitalter fortschreitender Digitalisierung in nahezu allen Lebensbereichen finden. Mit der Verbreitung von sozialen Netzwerken (Facebook, Twitter, mySpace etc.), Synchronisationsdiensten für E-Mails und Kalenderdaten und auch der privaten Anwendung von FileHostern (Rapidshare, Mediafire etc.) verlieren die Nutzer immer mehr die direkte Kontrolle über die eigenen Daten bewusst oder unbewusst. Es wird also nach Wegen gesucht beispielsweise die Urlaubsfotos für die Familie oder auch den Termin des Beförderungsgespräches mit dem Chef im Netz entsprechend gegen unbefugten Zugriff zu schützen. Schließlich kann selbst bei gesichertem Zugriff niemals die vollständige Sicherheit der Daten gegenüber Einbrüchen in die Serverfarmen der Anbieter garantiert werden.

Hierbei muss aber auch beachtet werden, dass selbst die „beste“ Verschlüsselungsmethode aufgrund ihrer Natur immer umkehrbar ist und der Originaltext wiederhergestellt werden kann. Dies ist aber meist mathematisch so kompliziert, dass selbst modernste Computer dafür mehrere Jahre brauchen. Diese Algorithmen können daher durchaus als „sicher“ eingestuft werden. Da die Leistungsfähigkeit von Computern jedoch unaufhaltsam steigt, werden ständig neue Wege gesucht aufwendigere und damit sicherere Verschlüsselungsmethoden zu schaffen.

Verbreitete Konzepte

Zu Anfang sei hier zu erwähnen, dass hier wirklich Konzepte erklärt werden und keine konkreten Implementationen und natürlich gibt es für beide Ansätze diverse Umsetzungen digitaler und analoger Natur.

Im Wesentlichen werden in der Verschlüsselung zwei konkurrierender Konzepte angewandt. Beide haben Vor- und Nachteile und werden in unterschiedlichen Szenarien angewandt. Einerseits gibt es die symmetrische Verschlüsselung. Das wichtigste Merkmal hier ist, dass nur ein Schlüssel sowohl zum Ver- als auch Entschlüsseln verwendet wird. Ein bekanntes Verfahren nach dieser Art wurde schon im antiken Rom von Caesar im Jahre 50 vor Christus verwendet um seine Marschbefehle für die Truppen vor den Feinden zu schützen. Das Verfahren war im Wesentlichen eine einfache Buchstabenverschiebung, wobei der Schlüssel hier eine Zahl zwischen 1 und 26 war und die Anzahl angab, um wie viele Buchstaben verschoben wurde. Dieses recht einfache Vorgehen war dennoch sehr effektiv, weil es eine der ersten größeren Anwendung von Verschlüsselung überhaupt war und die Gegner Caesars gar nicht an eine versteckte Botschaft in den scheinbar wirren, verschlüsselten Texten dachten.

Ganz nebenbei wird diese Tatsache auch in heutigen Verfahren angewandt, meist indem der Geheimtext immer noch ein sinnvoller Text ist (beziehungsweise sinnvoll interpretierbare Daten), sodass ein eventueller nicht autorisiert Empfänger unter Umständen gar keine versteckte Nachricht erwartet. Die Forschung rund um solche Verfahren wird auch Steganographie genannt.

Das zweite Konzept, das hier vorgestellt werden soll, ist entsprechend die asymmetrische Verschlüsselung. Auch hier ist der eigentlich zentrale Faktor die Anzahl und Verwendung der Schlüssel. Implementationen dieses Konzepts verwenden zwei verschiedene Schlüssel. Einen zur Verschlüsselung (oft öffentlicher Schlüssel genannt) und entsprechend einen zur Entschlüsselung (privater Schlüssel). Der öffentliche Schlüssel kann den Namen nach auch tatsächlich frei bekannt sein, da er im Prozess der Entschlüsselung ja keinerlei Bedeutung hat und somit nicht sicherheitskritisch ist. Im Gegenteil ist es sogar hilfreich, wenn der Schlüssel leicht einsehbar ist (beispielsweise auf der eigenen Webseite), denn dann ist es für neue Kommunikationspartner leichter eine Nachricht für den Empfänger zu verschlüsseln und der Schlüssel muss nicht erst im Voraus übermittelt werden. Da die zugrundeliegenden mathematischen Konzepte meist komplizierter und damit auch „jünger“ sind, gibt es an dieser Stelle kein so prominentes Beispiel wie die Caesar-Verschiebung.

Abschließend sei an dieser Stelle noch einmal darauf hingewiesen, dass es für beide Konzepte unzählige Umsetzungen gibt, die hier aber nicht im Fokus stehen sollen, sondern mehr die Gedanken hinter den beiden Ansätzen. Daher wird auch im weiteren Verlauf noch auf Vor- und Nachteile beider Seiten eingegangen.

Anwendungsbeispiel – Truecrypt

Über das Programm

Truecrypt ist ein Programm zur lokalen Verschlüsselung von Daten in Form von sogenannten Containern, ganzen Partitionen und sogar der Systempartition mittels on-the-fly Entschlüsselung. Dabei greift das Programm auf eine Vielzahl von symmetrischen Verschlüsselungsverfahren zurück. Namentlich sind das AES, Serpent und Twofish, als auch die Möglichkeit mehrere dieser Algorithmen zu kaskadieren also „hintereinander“ zu schalten (vereinfacht gesagt).

Warum symmetrische Verschlüsselung?

In der lokalen Verschlüsselung zeigen sich die Vorteile der symmetrischen Verschlüsselung. Der Nutzer muss sich nur einen Schlüssel merken, um an seine Daten zu kommen. Da die Daten auch nur an einer Stelle genutzt werden, müssen keine Schlüssel weitergeleitet werden und die Sicherheit der Daten hängt letztendlich nur von der Sicherheit des Algorithmus und des gewählten Schlüssels ab. Anders gesagt sind hier einfach keine weiteren Konzepte von Nöten. Der Nachteil der Symmetrischen Verschlüsselung, die Unsicherheit bei Schlüsselweitergabe, ist hier also vernachlässigbar.

Konkret sind die Daten bei einer Verschlüsselung durch Truecrypt als sicher zu betrachten, solange der Schlüssel ausreichend lang gewählt ist und den „allgemeinen Sicherheitskriterien“ (etwa keine Namen, Geburtstage, enthaltene Sonderzeichen etc.) genügt, da diese logischerweise recht einfach zu erraten sind und bei einem eventuellen Angriff vermutlich als erstes getestet würden. Alle der von Truecrypt unterstützten Algorithmen sind allgemein als „sicher“ eingestuft und waren alle Endrundenkandidaten bei der Wahl des AES-Standards.

Der „Advanced Encryption Standard“ (kurz AES) ist der wohl bekannteste Algorithmus, den Truecrypt unterstützt. Der im Oktober 2000 als Nachfolger für DES und 3DES gewählte Standard wurde ursprünglich unter dem Name Rijndael-Algorithmus (nach dem Entwicklern Rijmen und Daemen) veröffentlicht und bietet Kern-Features wie variable Blockgrößen von 128, 192 oder 256 Bit, Lizenzfreiheit, „einfache“ Implementationen in Hard- und Software, sowie bisher praktische Berechnungssicherheit. Dies zeigt sich im bisher erfolgreichsten vollständigen Angriff auf AES aus dem Jahre 2011. Dabei gelang den Kryptologen Andrey Bogdanov, Dmitry Khovratovich und Christian Rechberger den Schlüssel eines mit AES-128 verschlüsselten Systems in $2^{126,1}$ Schritten zu berechnen. Dies stellt zwar einen theoretischen Durchbruch gegen die Sicherheit von AES da, ist aber für die praktische Sicherheit nicht relevant, da diese Anzahl von Schritten auf modernsten Computern noch viel zu viel Zeit beanspruchen würde, als dass dieses Verfahren für realistische Angriffe genutzt werden könnte.

Serpent war in oben genannter Wahl des AES-Standard ebenfalls einer der fünf Finalisten und wurde von vielen Experten als der sicherste dieser bezeichnet. Ein Grund der aber auch gegen seine Wahl zum Standard sprach, ist jedoch die im Vergleich sehr schlechte Performanz in reinen Softwareimplementationen. Dies spricht auch gegen die Nutzung in Truecrypt, da hier keinerlei Hardwareverschlüsselung unter Serpent unterstützt wird. Nichtsdestotrotz bietet Serpent eine wenig übertroffene Sicherheit und wird von diversen Kryptologen als derzeit unangreifbar betrachtet.

Twofish war ebenfalls ein AES-Finalist. Er wurde jedoch unter Anderem aufgrund der hohen Komplexität, die eine Kryptoanalyse erschwert, abgelehnt. Da aber nach der Wahl von AES deutlich weniger Forschung zu den nicht gewählten Algorithmen betrieben wurde, ist die Sicherheit von Twofish nicht einfach zu belegen. Ein Entwickler gibt aber auf seinem Blog an, dass Twofish nicht gebrochen werden kann.

Funktionsweise

Um Daten mittels Truecrypt zu verschlüsseln bieten sich dem Nutzer im Wesentlichen zwei Möglichkeiten, die sich in der Nutzung jedoch recht wenig unterscheiden. Lediglich der Erstellungsprozess variiert etwas.

Zum einen bietet Truecrypt die Möglichkeit einen sogenannten verschlüsselten Container zu erstellen. Hierbei wird eine Datei mit beliebiger Dateiendung an einem vom Nutzer gewählten Ort im Dateisystem generiert, die dann später die verschlüsselten Daten enthält. Wichtig ist, dass der Speicherplatz standardmäßig bereits zur Erstellzeit belegt wird, also dem Restsystem nach der Erstellung erst einmal nicht zur Verfügung steht. Es besteht zwar auch die Option einen dynamisch wachsenden Container zu erstellen, aber die Entwickler empfehlen zurzeit noch die statische Variante (zumindest bei Einsatz in einem Produktivsystem). Weiterhin bietet die Wahl der Dateiendung leichte zusätzliche Ansätze zur Verschleierung, falls zum Beispiel eine unscheinbare Dateiendung wie .jpg oder .mp3 genutzt wird. Allerdings sollte man dabei (besonders im Falle dynamischer Speicherplatzreservierung) die spätere Größe des Container nicht außer Acht lassen, denn ein beispielsweise 100 GB großes Bild würde vermutlich eher weitere Aufmerksamkeit erregen, als vom Container abzulenken.

Die Erstellung dieser Container ist weitgehend uninteressant. Ein Assistent führt den Benutzer durch den Vorgang bei der obige Einstellungen sowie Größe, Dateisystem, Algorithmus (sowohl Verschlüsselung, als auch Hashing) und Schlüssel gewählt werden. Abschließend wird die gewählte Konfiguration noch einmal dargelegt und der Nutzer bestimmt mittels Mausbewegung im Dialogfenster einen (hoffentlich zufälligen) sogenannten *seed* aus dem dann mithilfe des benutzergewählten Schlüssels die konkreten Schlüssel des Containers generiert werden. Je nach Größe folgt dann ein relativ langer Prozess des Verschlüsselns und danach ist der Container direkt einsatzbereit.

Die zweite Variante wäre dann die Verschlüsselung einer kompletten Partition. Dabei kann Truecrypt sowohl leere Partitionen verschlüsseln (beziehungsweise die Partition dabei formatieren) als auch bestehende Partitionen inklusive Daten in einen verschlüsselten Zustand überführen, wobei dies logischerweise deutlich weniger performant ist, da der gesamte Datenbestand erst Sektor für Sektor gelesen, verschlüsselt und geschrieben werden muss. Die Erstellung erfolgt ähnlich über den Assistenten wobei hier natürlich anstatt eines Dateinamens und der Containergröße, die zu verschlüsselnde Partition sowie der Umgang mit vorhandenen Daten (siehe weiter oben) gewählt werden müssen.

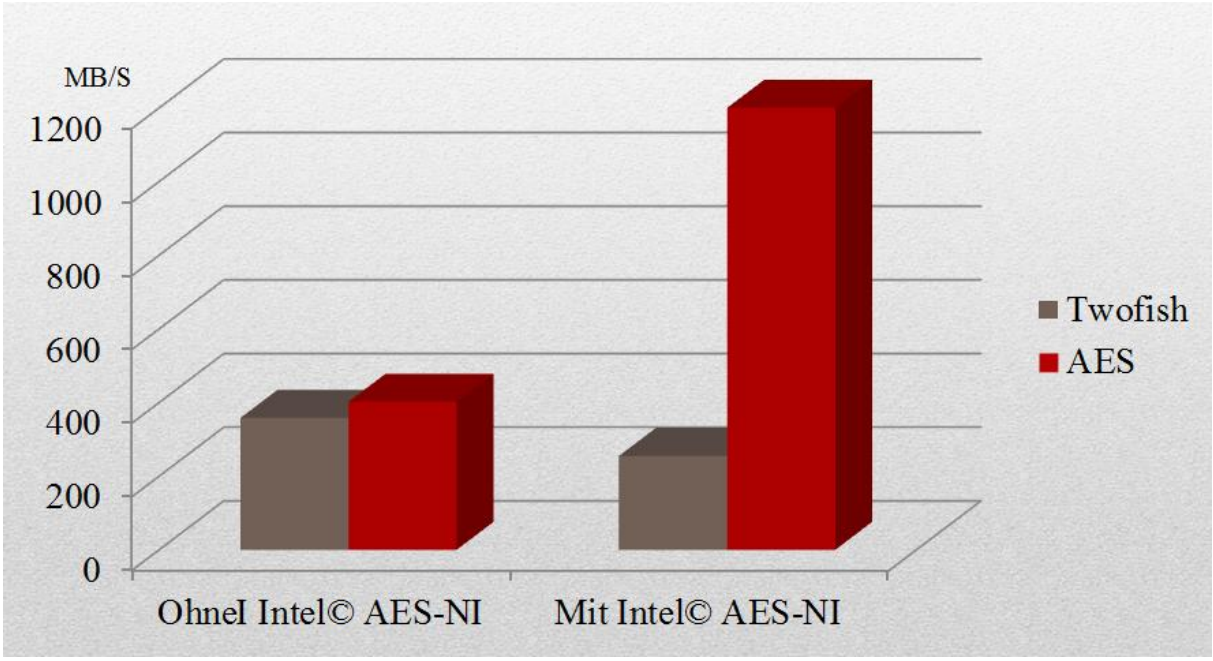
Eine Art Sonderfall stellt die Verschlüsselung der Systempartition da. Im Prinzip läuft die Erstellung auch hier wie bei einer normalen Partition ab, der Verschlüsselungsprozess jedoch kann natürlich nicht erfolgen, während das Betriebssystem läuft. Deshalb wird er nach einem Neustart aber vor dem Systemstart ausgeführt. Dabei wird ein Bootmanager in den Bootbereich der Festplatte (meist noch der Master Boot Record) geschrieben, der ab sofort das Betriebssystem startet. Nun muss der Nutzer

bei jedem Start das Passwort der Festplatte/Partition eingeben, um überhaupt „über das BIOS hinaus zu kommen“.

Hat der Benutzer dann einen verschlüsselten Container oder eine verschlüsselte Partition erfolgreich generiert, kann er diese „mounten“, um die Daten im laufenden System nutzbar zu machen. Dabei wird mittels Truecrypt ein Laufwerksbuchstabe (unter Windows) oder ein Verzeichnis (unter UNIX/Linux) gewählt und nach richtiger Eingabe des Schlüssels werden die entschlüsselten Daten an dem gewählten Ort bereitgestellt. Hierbei ist zu beachten, dass die Daten dabei zu keinem Zeitpunkt entschlüsselt auf das tatsächliche Speichermedium geschrieben werden, sondern nur „on-the-fly“ in den Speicher des Computers entschlüsselt werden. Daraus folgt auch ein kleiner Nachteil der Systemverschlüsselung. Da das ganze System ebenfalls konstant entschlüsselt wird, verzögert sich einerseits der Bootvorgang etwas, da dort sehr sequentiell viele Dateien gelesen werden, die nun natürlich erst entschlüsselt werden müssen und auch während des laufenden Systems kann während Speicherintensive Systemprozesse das System zusätzlich verlangsamen.

Wie aus dem vorgestellten Verfahren hervorgeht, sind vor dem Mounten des Container oder der Partition nur selbige sichtbar. Es ist also unmöglich Rückschluss auf die im Container/Partition enthaltenen Dateien zu ziehen. Man sagt auch, es wird das Konzept der „plausible deniabilty“ erfüllt. Demnach kann bei kritischen Daten der Besitz abgestritten werden. Andere Programme wie beispielsweise WinRAR verschlüsseln beispielsweise nur den Inhalt der Dateien, aber nicht deren Eigenschaften wie Dateiname, Erweiterung oder Größe. Dem Benutzer kann also im Zweifelsfall Besitz dieser Dateien vorgeworfen werden, auch wenn der Inhalt dieser zum aktuellen Zeitpunkt (noch) nicht bekannt ist. Durch die Verwendung einer einzelnen Datei als Container beziehungsweise einer (nach Verschlüsselung für das Betriebssystem unlesbaren) Partition ist die Sicherheit der gespeicherten Daten unter diesem Gesichtspunkt ebenfalls garantiert.

Performancevergleich



Das obige Diagramm wurde mithilfe der Truecrypt internen Benchmarkfunktion erstellt und zeigt die Entschlüsselungsperformance bei voller Auslastung zweier halbwegs aktueller Intel Quadcoreprozessoren. Es ist deutlich erkennbar, dass Performanceverlust kein Argument gegen Verschlüsselung mit Truecrypt mehr ist. Selbst ohne AES-Hardwareunterstützung ist ein circa vier Jahre alter Quadcore in der Lage mit einer Bandbreite von ~400 MB/S zu entschlüsseln. Natürlich ist das System während des Benchmarks quasi zu nichts anderem zu gebrauchen, aber andererseits schafft eine traditionelle Festplatte lange nicht 400 MB/S. Wenn man nun von einer relativ hohen Festplattenbandbreite von 80 MB/S ausgeht und vereinfacht herunterrechnet ergibt sich eine Prozessorauslastung von 20% und selbst das wäre bei einer konstanten Vollauslastung der Speicherbandbreite. Rechnet man nun AES-Hardwareunterstützung mit ein, kann man theoretisch eine durchschnittliche SSD bei voller Bandbreite ver- / entschlüsseln ohne jemals an die Grenze einer CPU-Limitierung zu gelangen und auch wenn diese Tests hier auf einem relativ aktuellen (Notebook-) i7 ausgeführt wurden, besitzen selbst aktuelle „schwache“ i5-Prozessoren diese Feature, was selbst bei Systemverschlüsselung den Performanceverlust vernachlässigbar machen sollte.

Anwendungsbeispiel – PGP

Über das Programm:

„PGP (kurz für „Pretty Good Privacy“ auf Deutsch etwa: „Ziemlich Gute Privatsphäre“) ist ein [..] Programm zur Verschlüsselung und zum Unterschreiben von Daten.“ (Wikipedia-Artikel: Pretty Good Privacy, 2012) Dabei greift PGP auf Methoden der symmetrischen sowie asymmetrischen Verschlüsselung zurück. Im Wesentlichen wird der Klartext symmetrisch verschlüsselt und der verwendete Schlüssel asymmetrisch verschlüsselt der Nachricht vorangestellt. Dies minimiert den Rechenaufwand beim Entschlüsseln, da das asymmetrische Verfahren (meist RSA) deutlich rechenintensiver ist und der Empfänger so nur den Schlüssel des symmetrischen Verfahrens auf diese Art entschlüsseln muss. Dazu später noch genaueres.

Weiterhin ist festzuhalten, dass ein offener Standard (namentlich OpenPGP) existiert, der auf der Basis von PGP 5.X entwickelt wurde. Da die folgenden Erläuterungen nicht auf Details eingehen und nur Verfahren beschreiben, lassen sich diese nicht nur auf PGP, sondern ebenfalls auf Implementationen dieses Standards übertragen.

Warum asymmetrische Verschlüsselung?

PGP wird im hauptsächlich im Bereich der Internetkommunikation genutzt (obwohl es natürlich theoretisch jede Art von Daten verschlüsseln könnte). Ich werde mich im Folgenden also darauf konzentrieren. Wichtig sind vor Allem zwei Kriterien: Einerseits kennt der Absender die Identität(en) des/der Empfänger der Nachricht (mindestens die Emailadresse(n)). Andererseits muss diese Bekanntschaft aber nicht persönlich sein, sodass das Aushandeln eines traditionellen (symmetrisch verwendeten) Schlüssels Probleme bereiten kann und die Sicherheit der Kommunikation eventuell gefährdet ist. Abhilfe schafft hier ein asymmetrisches Verfahren, da dabei ja (wie oben beschrieben) der öffentliche Schlüssel frei bekannt sein kann, also in diesem Beispiel über unsichere Kanäle beispielweise das Internet weitergegeben werden darf, ohne die Sicherheit zu beeinträchtigen. Weiterhin kann dieses Verfahren quasi umgedreht werden (wird gleich gezeigt), sodass der Besitzer des privaten Schlüssels darüber seine Identität verifizieren kann.

Es muss aber angemerkt werden, dass das Verfahren lediglich den Schlüsselaustausch erleichtert. Die Sicherheit der Daten beziehungsweise der Nachricht hängen hier nur vom privaten Schlüssel ab. Wird dieser entwendet oder auf andere Weise bekannt, gelangt ein Angreifer in den Vollbesitz der Daten wie auch bei symmetrischer Verschlüsselung. Da aber wie geschildert die Schlüsselweitergabe oft eines der größeren Probleme bei Kommunikation zwischen mehreren Partnern (insbesondere über das Internet) darstellt, wird asymmetrische Verschlüsselung oft in diesen „verteilten Verschlüsselungen“ verwendet.

Verfahren

Wie bereits oben angedeutet verwendet PGP zur eigentlichen Verschlüsselung der Daten einen symmetrischen Algorithmus, da dieses Verfahren jedoch bereits erklärt wurde, werde ich hier auf diesen nicht weiter eingehen, sondern hier erst einmal den verwendeten asymmetrischen Algorithmus RSA erläutern und dessen Verwendung während der Verschlüsselung der Nachricht.

Ein RSA-Schlüsselpaar besteht aus einem privaten Schlüssel, der wiederum aus einem geheimen Wert, dem Entschlüsselungsexponenten *d* und dem sogenannten RSA-Modul *N* und dem öffentlichen Schlüssel, analog bestehend aus dem Verschlüsselungsexponenten *e*, sowie dem genannten *N*. Um kommunizieren zu können, brauchen alle Teilnehmer also so ein Schlüsselpaar und diese können mit dem folgenden Verfahren erzeugt werden.

Zuerst wählt man zwei möglichst lange, ungleiche Primzahlen *p* und *q*, die eine Basis für das zukünftige Schlüsselpaar bilden. Sind diese gefunden, kann bereits *N* berechnet werden, denn das RSA-Modul ist nichts anderes als *p*q*. Nun wird mithilfe der Eulerschen ϕ-Funktion der Verschlüsselungsexponent *e* aus den beiden Primzahlen berechnet. Abschließend ist das multiplikativ Inverse von *e* bezüglich ϕ(*N*) das letzte gesuchte *d*.

Da die Primzahlen, sowie ϕ(*N*) nach Ende des Vorgangs gelöscht werden, kann ϕ(*N*) und damit auch *d* nicht aus dem öffentlichen Schlüssel allein rekonstruiert werden kann. Die Sicherheit des Verfahrens liegt jedoch darin bergründet, dass es für heutige, klassische Computer unmöglich ist „sehr große" *N* in polynomialer Zeit zu faktorisieren, also in seine beiden Faktoren, die Primzahlen *p* und *q* zu zerlegen.

Die Verschlüsselung einer Nachricht kann nun nach folgendem Schema durchgeführt werden.

$$C \equiv K^e \ mod \ N$$

C ist der herbei der zu erzeugende Geheimtext und K der vorhandene Klartext. Da wie beschrieben *e* und *N* Teil des öffentlichen Schlüssels sind, kann jeder, der im Besitz diesen ist, eine Nachricht an den Besitzer des Schlüsselpaars verschlüsseln. Die Entschlüsselung erfolgt analog.

$$\mathrm{K} \equiv C^d \ mod \ N$$

Daraus folgt, dass der Besitzer des privaten Schlüssels mithilfe von *d* und *N* einen nach der obigen Gleichung verschlüsselten Geheimtext wieder in seinen ursprünglichen Klartext überführen und damit die Nachricht entschlüsseln kann.

Soll nun eine Nachricht verschickt werden, wird ein zufälliger Schlüssel für den symmetrischen Algorithmus generiert und die Nachricht entsprechend verschlüsselt. Als nächstes wird dieser Schlüssel mit den öffentlichen Schlüsseln der Empfänger verschlüsselt und vor die verschlüsselte Nachricht gelegt. Dieser ganze Datenblock wird dann per base64-Verfahren in Zeichen umgewandelt und mit entsprechendem Header versehen. Der so entstandene Zeichenblock kann dann problemlos per Email verschickt werden. Optional kann der Versender noch einen Hash der Nachricht mit seinem privaten Schlüssel verschlüsseln, sodass eine Signatur entsteht. Empfänger können diesen Hash dann entschlüsseln mit einem selbst berechneten Hash vergleichen und so die Identität des Absender (oder zumindest des verwendeten Schlüssels) bestätigen.

Wie oben angedeutet verkürzt dieses Verfahren den Rechenaufwand beim Empfänger signifikant. Einerseits muss nur der symmetrische Schlüssel „teuer" – also asymmetrisch – entschlüsselt werden. Andererseits reduziert die Verwendung eines Hashs bei der Signatur weiterhin der Rechenaufwand, da nicht die komplette Nachricht ein weiteres Mal verschlüsselt werden muss, die Signatur aber trotzdem an diese gebunden ist und nicht für beliebige Nachrichten (unerlaubterweise) kopiert werden kann.

Verschlüsselnde Dateisysteme

In diesem letzten Abschnitt wende ich mich nun den verschlüsselnden Dateisystemen zu. Dies hat auch einen guten Grund, denn das grundlegende Konzept weist einige Ähnlichkeiten zu dem Verfahren aus PGP aus dem vorherigen Abschnitt auf.

Verfahren / Konzept

Das grundlegende Konzept, das die Basis für die hier vorgestellten Implementationen bildet geht verfolgt im Wesentlichen zwei für Dateisysteme besonders relevante Ziele abseits von der offensichtlichen Sicherheit. Einerseits ist Performanz ein wichtiger Faktor, da besonders bei einer Systemverschlüsselung lange Wartezeiten zum Beispiel während des Bootvorgangs nicht akzeptabel sind. Andererseits muss in den meisten Fällen Mehrbenutzerfähigkeit garantiert werden, da bei heutigen Desktoprechnern lange nicht mehr nur ein Benutzer vorhanden sein muss.

Wie bereits oben erwähnt ähnelt die Methode also sehr dem Verfahren von PGP. Es werden also ebenfalls Varianten der symmetrischen und asymmetrischen Verschlüsselung verwendet. Im ersten Schritt verschlüsselt das Dateisystem den Block oder die Datei (je nach Implementation) symmetrisch mit einem zufälligen Schlüssel. Danach wird dieser Schlüssel für jeden Benutzer, dem die Datei/der Block zur Verfügung gestellt werden soll, mit deren öffentlichen Schlüsseln verschlüsselt in die Dateieigenschaften / ans Ende des Blocks geschrieben. Es entsteht also eine Art PGP-Nachricht mit dem Unterschied, dass die Schlüssel am Ende stehen. Dies garantiert einerseits Kompatibilität für mehrere Benutzer, da am Ende (theoretisch) beliebig viele Schlüssel angehängt werden können. Andererseits wird wie bei PGP Performanz gewonnen, indem nur der Schlüssel asymmetrisch verschlüsselt wird.

Grundlegende Probleme

Bereits nur aus diesem Konzept ergeben sich bereits einige Probleme, wenn man bedenkt, dass das Einsatzgebiet ein Dateisystem ist. Einerseits muss erwähnt werden, dass bei den meisten verschlüsselnden Dateisystemen im Gegensatz zu PGP der zufällige Schlüssel nicht für jeden Block zwangsweise neu gewählt wird, um die Performanz zu erhöhen. Daraus folgt aber auch, dass aus diesem Grund sehr leicht identische verschlüsselte Blöcke aus identischen Originalblöcken entstehen können, was eine Angriffsmöglichkeit darstellen kann, wenn zum Beispiel der Originalinhalt eines Blocks von einem potenziellen Angreifer beeinflusst werden kann. So kann dann Rückschluss auf andere Blöcke gezogen werden. Außerdem hat traditionell das Betriebssystem die Hoheit über das Dateisystem und auch wenn hier keine entschlüsselten Daten tatsächlich geschrieben werden, müssen dessen Berechtigungen gesichert sein, sodass einerseits ein reibungsloser Betrieb ermöglicht wird und andererseits ein korrumpiertes Betriebssystem nicht sicherheitsrelevante Daten weitergeben kann. Daraus folgt auch direkt das letzte Problem und zwar sind für den sicheren Betrieb mit einem verschlüsselnden Dateisystem sehr strenge Zugriffsrechte erforderlich. Denn was nützt benutzerspezifische Verschlüsselung, wenn beispielsweise bei oft veränderten Systemdateien aus Komfortgründen jeder Benutzer als berechtigt markiert ist?

Umsetzungen

Als Vertreter verschlüsselnder Dateisysteme habe ich hier „Encrypting File System“ (EFS) für Microsoft Windows und „dm-crypt“ mit der Erweiterung LUKS für GNU/Linux. Beide sind relativ simpel in das Betriebssystem integrieren und mit wenig Aufwand einzurichten.

Encrypting File System

EFS ist das mitgelieferte Verschlüsselungssystem von Microsoft Windows seit Windows 2000. Es ist eigentlich auch nur eine Erweiterung von NTFS, dies ermöglicht aber gerade die einfache nachträgliche Integration ohne eine neue Formatierung durchführen zu müssen, wie beispielsweise bei Truecrypt. EFS verschlüsselt Ordner und Dateien nach genau dem oben genannten Konzept, wobei als Algorithmen AES (bis Windows XP DES) als symmetrischer und RSA als asymmetrischer zum Einsatz kommt. Wichtig ist hierbei, dass nicht die ganze Partition oder gar Festplatte verschlüsselt wird, sondern der Nutzer selektiv einzelne Ordner oder Dateien auswählen muss.

Weiterhin gibt es eine potenziell unsichere Methode zur Datenwiederherstellung. So kann der Schlüssel verschlüsselt mit dem öffentlichen Schlüssel des Administrators zur Wiederherstellung genutzt werden. Dies ist natürlich problematisch, da gerade in größeren Firmen mehrere Personen administrativen Zugriff haben könnten, sodass dadurch eine Schwäche entstehen könnte.

dm-crypt mit LUKS

„dm-crypt ist ein Kryptographie-Modul des Linuxkernels. Es ermöglicht die Verschlüsselung von Festplatten, Partitionen oder logischen Laufwerken. (Wikipedia-Artikel: dm-crypt, 2012) Dabei bildet es eine für den Benutzer nicht sichtbare Zwischenschicht zwischen den verschlüsselten Daten auf dem Speichermedium und dem Dateisystem. Von sich aus stellt dm-crypt dabei eine Vielzahl von Algorithmen aus der Crypto-API des Linuxkernels zur Verfügung. Ich gehe hier jedoch besonders auf die Erweiterung LUKS (kurz für „Linux Unified Key Setup“ auf Deutsch in etwa „standardisierte Schlüsselverwaltung für Linux“) ein, die es ermöglicht den verschlüsselten Daten einen Header hinzuzufügen, der eine Klartextkennung, den verwendeten Verschlüsselungs- und Hashalgorithmus, die Größe des Masterschlüssels und bis zu 8 per öffentlichem Schlüssel verschlüsselte Versionen dessen. Dies ermöglicht eine deutlich einfachere Verwaltung der verschlüsselten Daten, da diese über den Header erkannt und einfach katalogisiert werden können. Es erleichtert allerdings auch eventuelle Angriffe, da einige Details über den verwendeten Algorithmus im Klartext verfügbar gemacht werden. Weiterhin erschwert die leichtere Erkennbarkeit das Leugnen des Vorhandenseins von verschlüsselten Daten (siehe oben „Plausible Deniability“) was einen starken Nachteil gegenüber der leichteren Verwaltbarkeit darstellt.

Probleme

Aus dem Konzept der verschlüsselnden Dateisysteme ergeben sich bereits einige Angriffsmöglichkeiten. Zum Beispiel kann ein Wasserzeichenangriff (englisch: „watermark attack“) erfolgen. Dieser zielt meist nicht auf das tatsächliche Entschlüsseln der Daten ab, sondern versucht die Anwesenheit von bestimmten Daten im verschlüsselten Teil eines Speichermediums nachzuweisen. So können bei bestimmten Verschlüsselungsverfahren, die auf Basis der Sektornummer einer Festplatte verschlüsseln, unabhängig vom verwendeten Algorithmus, mithilfe von präparierten Klartexten zwei gleiche, aufeinanderfolgende Blockinhalte generiert werden. Kann dieses sogenannte Wasserzeichen dann auf dem Computer des Opfers nachgewiesen werden, kann daraus der Nutzer das Vorhandensein dieser Daten auf seinem Computer nicht mehr leugnen und daraus können beispielweise in einem Strafverfahren unter Umständen weitere juristische Schritte legitimiert werden.

Wie aus der Funktionsweise verschlüsselnder Dateisysteme bereits erkennbar ist, zeigen diese nur Wirkung, solange der Computer nicht eingeschaltet ist und die Daten verschlüsselt auf dem Speichermedium geschrieben sind. Da die Daten während des Betriebs meist unverschlüsselt im Arbeitsspeicher vorhanden sind, greift die Verschlüsselung dort natürlich nicht. Daraus ergeben sich auch weitere Schwachstellen. Gelingt es einem Angreifer in den Besitz des laufenden Computers zu gelangen, so kann natürlich mithilfe der entschlüsselten Daten der Schlüssel rekonstruiert werden oder je nach Dateisystem direkt ausgelesen werden. Aus diesem allgemeinen Teil des Konzeptes werden auch die Einsatzgebiete verschlüsselnder Dateisysteme im Allgemeinen eingeschränkt. So muss beispielweise beim Schutz von Bankservern oder Kundendatenbanken eher auf den Schutz der Kommunikation geachtet werden, da dies meist den offensichtlichsten Angriffspunkt darstellt.

Ein allseits präsenter Angriffspunkt bei Verschlüsselungen aller Art ist natürlich auch hier vorhanden. Wie oben erwähnt muss natürlich auf die Sicherheit des Algorithmus garantiert werden können, damit die Sicherheit der Verschlüsselung auf lange Sicht erhalten bleibt.

Fazit

Mit Truecrypt und PGP hat der Nutzer zwei mächtige Werkzeuge in der Hand seine sensiblen Daten mit äußerst wenig Aufwand zu verschlüsseln. Beide verwenden als sicher einzustufenden Algorithmen und sichern bei korrekter Anwendung zuverlässig die privaten Inhalte.

Sollen die Daten im Wesentlichen bei Entwendung sicher sein, können auch verschlüsselnde Dateisysteme eingesetzt werden. Diese sind einfach ins Betriebssystem zu integrieren und sind (fast) ohne weitere Zusatzsoftware einsatzbereit.

Literaturverzeichnis

Wikipedia-Artikel: Advanced Encryption Standard. (2012). Von http://de.wikipedia.org/wiki/Advanced_Encryption_Standard abgerufen

Wikipedia-Artikel: dm-crypt. (2012). Von http://de.wikipedia.org/wiki/Dm-crypt abgerufen

Wikipedia-Artikel: Pretty Good Privacy. (2012). Von http://de.wikipedia.org/wiki/Pretty_Good_Privacy abgerufen

Wikipedia-Artikel: Truecrypt. (2012). Von http://de.wikipedia.org/wiki/Truecrypt abgerufen

Wikipedia-Artikel: Twofish. (2012). Von http://de.wikipedia.org/wiki/Twofish abgerufen