

1

Proseminar: **Konzepte von
Betriebssystemkomponenten**

Kryptographische Dateisysteme

Übersicht über den Vortrag

- ❑ Braucht man überhaupt Verschlüsselung von Dateisystemen?
- ❑ Konkrete Zielsetzung
- ❑ Cryptographic File System (CFS), der “Urvater”
 - ◆ Benutzersicht
 - ◆ Implementierung
 - ◆ Sicherheit

Übersicht - Fortsetzung

- Weitere Systeme
 - ◆ Transparent und Enhanced CFS
 - ◆ CryptFS
 - ◆ Encrypting File System für Windows 2000

Warum Dateisysteme verschlüsseln?

- ❑ Vertrauliche Daten werden auf Server im Netzwerk gespeichert
- ❑ Angreifer erlangt *root*-Rechte auf meinem System
 - ◆ bei Verwendung von Dateisystemverschlüsselung erhält er nur Zugriff auf Ciphertext
 - ◆ Eventuelle weitere Maßnahmen des Angreifers erhöhen Entdeckungsrisiko

Warum verschlüsseln - Fortsetzung

- Angreifer erlangt *physischen* Zugang zum Datenträger
 - ◆ mounten des Datenträgers auf eigenem Rechner erlaubt ihm ebenfalls root-Zugriff
 - ◆ Auslesen von Rohdaten immer möglich, erlaubt alles von Volltextsuche bis Beauftragung von Datenrettungsunternehmen, um selbst vom Besitzer gelöscht geglaubte Daten wiederherzustellen

- Random Fact: Britisches Ministry of Defence verlor 594 Laptops in den letzten 5 Jahren . . .

Ziele der Verschlüsselung

- ❑ Hauptziel: Sicherung von Daten und vertraulichen Metadaten, etwa Dateinamen
- ❑ Transparenz: Benutzer soll möglichst wenig von der Verschlüsselung beeinträchtigt sein
 - ◆ Seltene zusätzliche Nutzerinteraktion z.B. für Schlüsseingabe
 - ◆ Kein übermäßiger Performanceeinbruch
 - ◆ Kompatibilität zu bereits existierenden Applikationen
 - ◆ Erhaltung der Semantik für nebenläufige Dateizugriffe

Ziele - Fortsetzung

- Konsistenz des zugrunde liegenden Dateisystems soll erhalten bleiben
- Entfernung des Schlüssels aus dem System nach Gebrauch
- Vertrauen des Benutzer nur in Systeme in seiner direkten Kontrolle notwendig

Nutzerinterface

- ❑ `cmkdir` erstellt Verzeichnisse, deren Inhalt verschlüsselt wird
- ❑ `cattach` für die Eingabe der Passphrase und Assoziation von Klartextverzeichnis, Schlüssel und Ciphertextverzeichnis
- ❑ `cdetach` für die Entfernung des Schlüssels
- ❑ `cname` und `ccat` für Zugriffe ohne installiertes CFS

Implementierung

- ❑ Schneller Dateizugriff an beliebige Position in eventuell grosse Dateien erforderlich
- ❑ blockbasierte Verschlüsselung unter Verwendung des DES Algorithmus
- ❑ Damit grosse Datenmengen nicht alle mit demselben Key verschlüsselt werden, XOR-Verknüpfung der Daten mit einem aus der Passphrase erzeugtem Bitfeld
- ❑ Versuch, hohe Sicherheit mit niedrigem Rechenaufwand zu erreichen

Implementierung - Schlüsselerzeugung

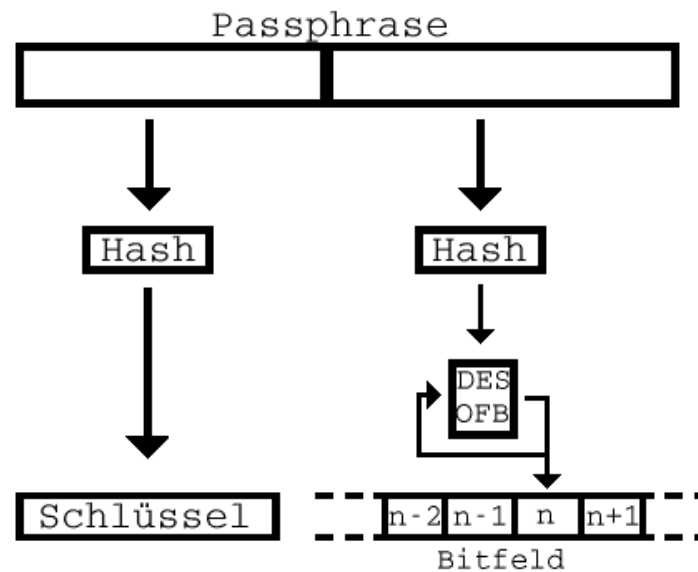


Abbildung 1: Schlüsselerzeugung

Aus der eingegebenen Passphrase wird ein DES-Schlüssel erzeugt, aus dem anderen Teil wird mit dem *DES output feedback* Modus das Bitfeld erzeugt.

Implementierung - Ver- und Entschlüsselung

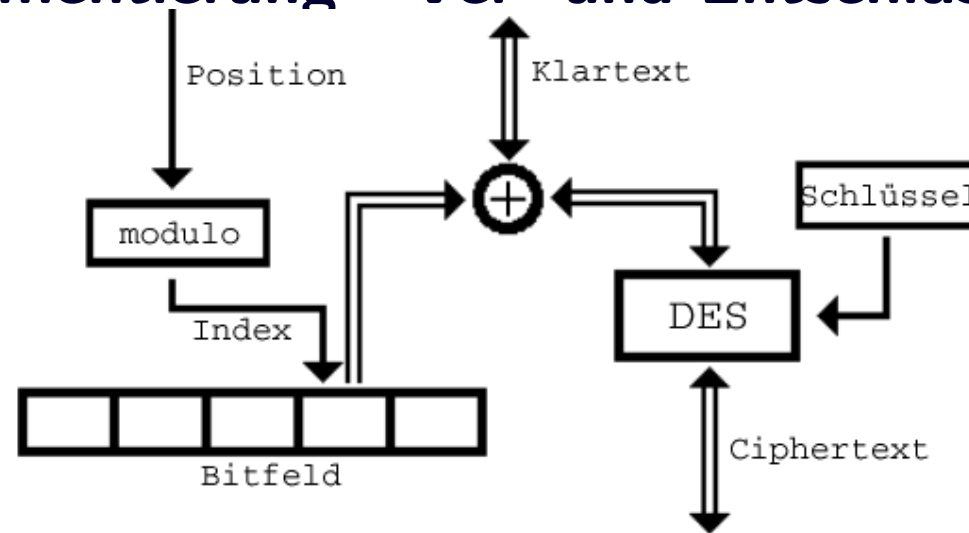


Abbildung 2: Ver- und Entschlüsselung

Aus der Position wird der dazugehörige Index im Bitfeld errechnet. Die Daten werden dann mit dieser Stelle des Bitfelds XOR-verknüpft, und das Ergebnis dann mit DES verschlüsselt, beim Lesen entsprechend in umgekehrter Reihenfolge.

Implementierung - Gesamtarchitektur

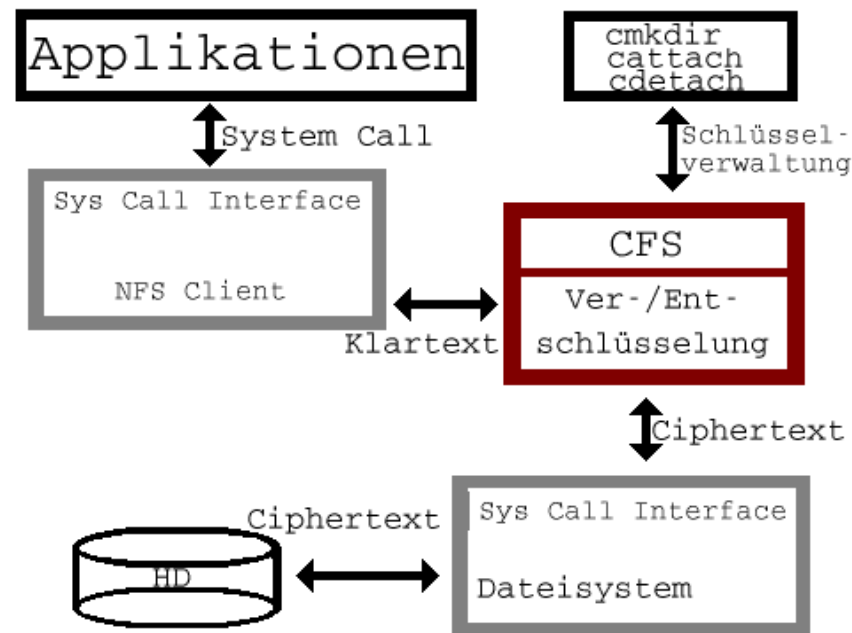


Abbildung 3: Gesamtsystem

CFS ist als eine erweiterte Version eines NFS-Servers konzipiert

Architektur - Fortsetzung

- cfsd läuft als User Prozess
 - ◆ kein spezieller Speicherschutz
 - ◆ häufige Kontextwechsel und Kopieren von Daten zwischen Kernel und User Space
 - ◆ keine Eingriffe am Betriebssystem selbst notwendig
- Portabilität auf alle Systeme, die NFS unterstützen

Architektur - Fortsetzung

- Speicherung des Ciphertexts durch System Calls
 - ◆ Schnelle Entwicklungszeit
 - ◆ Kompatibilität mit vielen Betriebssystemen
 - ◆ Dateien können mit normalen Systemtools bearbeitet werden, z.B Backups, Zugriffsrechte, fsck
 - ◆ allerdings: Informationen wie Dateilänge können nicht verschlüsselt werden

Sicherheit

- ❑ kein Schutz des Ciphertexts, der über OS Sicherung hinausgeht
 - ◆ Angreifer mit root-Rechten könnte nur schlecht am Löschen gehindert werden
 - ◆ Bei Verlust des Datenträgers sind Daten sowieso weg
 - ◆ Vorteil: ermöglicht Backups, die dann automatisch auch verschlüsselt sind
- ❑ Klartext und Passphrase nur so sicher, wie das System auf dem CFS läuft
- ❑ Alle weiteren Systeme sehen nur Ciphertext, müssen also nicht sicher sein

Sicherheit der Verschlüsselung

- ❑ Verschlüsselung wohl annähernd so stark wie 3DES mit zwei unabhängigen Subkeys
- ❑ Möglichkeit von *known plaintext* Angriffe, wenn bekannte Dateiformate verwendet werden
- ❑ Eventuell sehr viele Daten mit demselben Schlüssel verschlüsselt

Verbesserungen von CFS

- Transparent Cryptographic File System (TCFS)
 - ◆ *Transparent*, weil Schlüsseleingabe noch weiter automatisiert werden kann, bis hin zu automatisch beim Login, oder beim mounten eines Dateisystems
 - ◆ Geschwindigkeitsoptimierung durch Verschlüsselung im Betriebssystemkern
 - ◆ Unterstützung von DES, IDEA, und Blowfish, Auswahl eines Algorithmus je nach Geschwindigkeits- und Sicherheitsanforderungen
- Extended CFS erlaubt Signierung von Dateien und Metadaten zur Integritätssicherung

CryptFS

- ❑ Von vornherein für Integration in Solaris Kernel konzipiert, mittlerweile verfügbar auch für Linux und FreeBSD
- ❑ Operiert auf Blöcken in der Grösse von Speicherseiten, optimiert für *memory mapped* Dateioperationen
- ❑ Basierend auf einem Loopback-Dateisystem, Datei vom Kernel schreibgeschützt
- ❑ erreicht ca. 80% der Geschwindigkeit unverschlüsselten Zugriffs

NTFS Encrypted File System (EFS)

- ❑ Für jede Datei wird ein symmetrischer Schlüssel, der *File Encryption Key* erzeugt
- ❑ An Datei wird der mit dem public Key des Nutzers verschlüsselte FEK angehängt
- ❑ Dateisharing möglich, indem der FEK für jeden Nutzer einmal angehängt wird

EFS - Fortsetzung

- Datei immer mit dem *Recovery Agent* geshared
 - ◆ Administrator kann Datei immer wiederherstellen
 - ◆ Private Key des Recovery Agents besonders lohnendes Angriffsziel

This page intentionally left blank