XorCrypt

Datei-Format und Verschlüsselungs-Tool

von Jochen Ertel

13.06.2018

Inhaltsverzeichnis

1 Überblick

XorCrypt ist in erster Linie ein minimalistisches Datei-Format für eine passwortbasierte Datei-Verschlüsselung, welche Vertraulichkeit und Integrität einer Datei sicherstellt. Als Algorithmen sind AES-256 im Counter-Modus (Verschlüsselung) und SHA-256 im HMAC-Modus (Integritätssicherung) festgelegt. Die zwei erforderlichen Schlüssel werden per SHA-256 basiertem PBKDF2 Algorithmus aus dem Passwort abgeleitet.

Unabhängig von der Datei-Format-Spezifikation wird in diesem Dokument eine Implementierung, das Tool XorCrypt, beschrieben. Dieses erlaubt es, Dateien im XorCrypt-Format zu ver- und entschlüsseln. Es steht als C-Quellcode für Debian-basierte Linux-Betriebssysteme zur Verfügung und kann dort als Kommandozeilen-Applikation gebaut und verwendet werden.

2 XorCrypt-Format

2.1 Zielstellung

Bei der Spezifikation von XorCrypt wurden folgende Ziele verfolgt:

- Es sollte ein möglichst einfaches Verfahren sein.
- Die Verschlüsselung sollte passwortbasiert erfolgen.
- Tools zur Ver- und Entschlüsselung sollten ohne großen Aufwand implementierbar sein.
- Es sollte aktuellen kryptografischen Sicherheitsanforderungen genügen, d.h. möglichst von niemandem auf dieser Welt zu brechen sein (unter der Vorraussetzung eines entsprechend lang gewählten Passwortes).
- Es sollte auf standardisierten kryptografischen Verfahren basieren.
- Die Entschlüsselung sollte nicht zu einer Fehlerfortpflanzung führen, d.h ein gekipptes Bit im Chiffrat sollte auch nur ein gekipptes Bit im entschlüsselten Klartext zur Folge haben.
- Eine verschlüsselte XorCrypt-Datei sollte als solche nicht identifizierbar sein, d.h. sie sollte sich nicht von einem Zufalls-Bytestrom unterscheiden.

2.2 Spezifikation

2.2.1 Datei-Struktur

Eine verschlüsselte XorCrypt-Datei besteht aus 3 Teilen:

- 32 Byte statistisch guter Zufall, der ausschliesslich für eine einzige Datei-Verschlüsselung verwendet wird.
- Das Chiffrat, d.h. die mittels AES-256 im Counter-Modus verschlüsselte Datei, in selber Länge wie die Orginal-Datei.
- Eine 32 Byte lange, mittels SHA-256 im HMAC-Modus über die ersten beiden Teile berechnete kryptografische Prüfsumme (Integritätscheckwert).

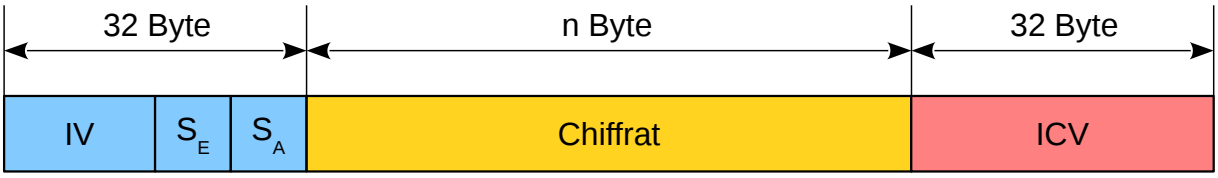

Abbildung 1: XorCrypt Datei-Struktur

2.2.2 Zufall

Für die XorCrypt-Verschlüsselung werden 32 Byte Zufall benötigt. Diese 32 Byte werden folgendermassen verwendet:

- Die ersten 16 Byte (128 Bit) stellen den Initialisierungsvektor für die Counter-Modus-Verschlüsselung dar (IV).
- Die nächsten 8 Byte (64 Bit) fungieren als Salt-Wert für die Ableitung des AES-Schlüssels aus dem Passwort (S_E).
- Die letzten 8 Byte (64 Bit) fungieren als Salt-Wert für die Ableitung des HMAC-Schlüssels aus dem Passwort (S_A).

Bei einer Implementierung ist unbedingt sicherzustellen, dass jeder 32 Byte-Zufalls-Vektor nur einmal verwendet wird. Würden 2 Dateien mit dem selben Passwort und basierend auf dem selben Zufall verschlüsselt, wäre die Verschlüsselung aufgrund des verwendeten Counter-Modus (xor-Verknüpfung) quasi gebrochen.

2.2.3 Verschlüsselung

Die Verschlüsselung erfolgt mittels AES-256 (siehe [AES]) im Counter-Modus (siehe [CTR]). D.h. es wird zunächst durch Verschlüsselung des fortlaufend inkrementierten Initialisierungsvektors (IV) ein pseudozufälliger Bytestrom erzeugt. Mit diesem wird die Orginaldatei dann bitweise xor-verknüpft. Daher rührt auch der Name XorCrypt. Für die Verschlüsselung kommt ein 256 Bit langer AES-Schlüssel zum Einsatz.

Zu beachten ist, dass dabei die ersten 16 Byte (128 Bit) des pseudozufälligen Bytestromes durch AES-256 Verschlüsselung des IVs selbst erzeugt werden, alle weiteren 16 Byte-Blöcke durch Verschlüsselung der inkrementierten IV-Werte. Desweiteren sei erwähnt, dass ein Padding der zu verschlüsselnden Datei nicht nötig ist.

2.2.4 Integritätssicherung

Über die Verkettung der 32 Byte Zufall mit dem Chiffrat der Orginaldatei wird eine kryptografische Prüfsumme (Integritätscheckwert - ICV) mittels SHA-256 (siehe [SHA]) im HMAC-Modus (siehe [HMAC]) berechnet. Dabei kommt ein 256 Bit langer HMAC-Schlüssel zum Einsatz. Die Prüfsumme ist 32 Byte (256 Bit) groß.

2.2.5 Schlüsselableitung

Um Brute-Force-Angriffe auf das Passwort zu erschweren, wird das Passwort nicht direkt

als Schlüssel für AES-256 und HMAC-SHA-256 verwendet. Die beiden benötigten Schlüssel werden aus dem Passwort durch einen rechenintensiven Algorithmus abgeleitet. In die Ableitung fliesst neben dem Passwort ein individueller, nur einmalig zu verwendender 64 Bit großer Salt-Wert mit ein. Als Ableitungs-Algorithmus wird PBKDF2 (siehe [PBKDF2]) basierend auf SHA-256 verwendet.

Das Passwort darf aus maximal 63 ASCII-Zeichen bestehen, auch ein leeres Passwort (0 Byte groß) sei definiert. Diese Festlegung dient dazu, Implementierungen eindeutig sowie zueinander kompatibel zu gestalten. Die maximal mögliche Entropie eines Passwortes beträgt somit ca. 63 x 6 Bit = 378 Bit (Annahme: effektiv 6 Bit pro ASCII-Zeichen). Das Passwort wird als Bytestrom (1 Byte pro ASCII-Zeichen) an die Schlüsselableitung übergeben.

Algorithmenbeschreibung PBKDF2_XORCRYPT:

```
Eingabe:       P  (Passwort Bytestrom, Länge: 0 ... 63 Byte)
               S  (Salt-Zufallswert, Länge: 8 Byte)

Ausgabe:       K  (abgeleiteter Schlüssel, Länge: 32 Byte)

Algorithmus:   T = 00 00 00 ... 00          (32 Null-Bytes)
               U = S || 00 00 00 01         (8 Byte Salt
                                             + 4 Byte Integer-Wert 1)
               for i= 1 to 1.000.000
                 U = HMAC-SHA-256 (P, U)
                 T = T xor U                (bitweise xor-Verknüpfung,
                                             Länge T, U: je 32 Byte)
               K = T
```

2.2.6 Bemerkungen zur Byte-Order

In diesem Dokument wird die heutzutage in kryptografischen Standards übliche Bit- bzw. Byte-Anordung in einem Bytestrom vorausgesetzt. D.h. das erste Bit in einem Byte ist das höchstwertigste, ebenso ist das erste Byte in einem Bytestrom / Codeblock / kryptografischen Wort immer das höchstwertigste, welches an linker Stelle geschrieben wird. Der Begriff "höchstwertig" hat natürlich nur im Falle einer numerischen Betrachtung eines Wertes Bedeutung, z.B. beim inkrementieren des 128 Bit großen Initialisierungsvektors im Counter-Modus. Ansonsten gilt wie bei der Schrift, Byteströme werden byteweise von links nach rechts geschrieben.

3 XorCrypt-Tool

3.1 Grundlegendes

Das XorCrypt-Tool von Jochen Ertel in der Version 1.0 ist eine simple Implementierung einer Datei-Ver- und Entschlüsselung basierend auf dem XorCrypt-Datei-Format bzw. Algorithmus (siehe Kapitel 2). Das Tool ist in C geschrieben und unter Debian-basierten

Linux-Systemen als Kommandozeilen-Applikation aus dem Quellcode baubar. Es bindet die unter Debian-Systemen vorhandene OpenSSL-Bibliothek ein (Algorithmen AES-256 und HMAC-SHA-256).

Es wird vorausgesetzt, dass der Computer, auf dem XorCrypt verwendet wird, vetrauenswürdig ist, d.h. dass kein potentieller Angreifer Zugriff darauf hat. Dies ist dadurch begründet, dass die Verarbeitung des Passwortes bei und nach der Eingabe durch den Anwender nicht besonders geschützt ist. Das Passwort wird im Klartext als Parameter übergeben und befindet sich nachher in der Bash-History sowie dem Arbeitsspeicher.

Ziel bei der Entwicklung war es ausschließlich, Dateien für die Übertragung durch das Internet sowie die Speicherung in der Cloud oder auf externen Datenträgern abzusichern. Es war nicht Ziel, lokale Dateien auf dem Computer vor dem Zugriff Dritter zu schützen!

3.1.1 Funktionalität

In der Version 1.0 kann man mit dem XorCrypt-Tool folgendes tun:

- Man kann einen Selbsttest der Krypto-Funktionen durchführen, um zu sehen, dass diese (AES und HMAC-SHA-256 eingebunden über OpenSSL) korrekt funktionieren.
- Man kann eine Datei verschlüsseln.
- Man kann eine verschlüsselte Datei ausschließlich auf Integrität prüfen.
- Man kann eine verschlüsselte Datei entschlüsseln.

Optional kann man mittels einer übergebenen Zeichenkette die 32 Byte Zufall, die bei der Verschlüsselung benötigt und generiert werden, beeinflussen (siehe folgender Abschnitt).

3.1.2 Zufalls-Generierung

Für jede Datei-Verschlüsselung werden initial 32 Byte Zufall benötigt (siehe Abschnitt 2.2.2). Ausgangspunkt für deren Erzeugung ist die aktuelle Unix-Zeit, ein Integerwert, der die vergangenen Sekunden seit dem 1. Januar 1970 anzeigt. Dieser Integerwert wird als ASCII-Zeichenkette (UnixTimeString) zum Schlüssel für eine HMAC-SHA-256 Operation, welche die 32 Byte Zufall zum Ergebnis hat:

```
Random = HMAC-SHA-256 (UnixTimeString, "abc")
```

Die gehashte Zeichenkette "abc" kann optional durch eine beliebige andere Zeichenkette ersetzt werden. Es wird davon ausgegangen, dass das XorCrypt-Tool niemals innerhalb einer einzigen Sekunde mehrfach aufgerufen wird. Damit ist die Bedingung, dass die erzeugten 32 Byte Zufall niemals wiederholt verwendet werden dürfen, erfüllt.

4 Test-Vektoren

In allen Beispielen dieses Kapitels wird von den folgenden 32 Byte initialem Zufall ausgegangen:

```
Random = {d8 bc 3e 25 b4 81 0c ee 08 65 99 c8 3c fe f4 75
          d2 1a bd 55 14 eb c0 70 74 9b 93 2e 72 0b 6d e8}

       = IV || SE || SA
```

4.1 Schlüsselableitung

Im Folgenden seien 4 Bespiele für die Schlüsselableitung nach Kapitel 2.2.5 gegeben.

Ableitung des AES-Schlüssels K_E vom leeren Passwort (der Länge 0) mittels Algorithmus PBKDF2_XORCRYPT:

```
P  = {}

SE = {d2 1a bd 55 14 eb c0 70}

KE = {c8 82 fc bd ce 3c ac de 3b dd 17 52 cb 5f ea f8
      93 84 c4 4a 85 2b e9 72 b8 b4 22 27 cc 14 75 d5}
```

Ableitung des AES-Schlüssels K_E vom Passwort „password“ mittels Algorithmus PBKDF2_XORCRYPT:

```
P  = {70 61 73 73 77 6f 72 64}   # "password"

SE = {d2 1a bd 55 14 eb c0 70}

KE = {41 7c 20 82 10 e4 bb bb 1c ba d3 af 5b 9f 59 57
      ee a5 26 99 ef 87 2c bc ec 95 89 dd e2 ba 2f da}
```

Ableitung des HMAC-Schlüssels K_A vom leeren Passwort (der Länge 0) mittels Algorithmus PBKDF2_XORCRYPT:

```
P  = {}

SA = {74 9b 93 2e 72 0b 6d e8}

KA = {25 dc cc ad 47 36 dd 47 a6 25 96 2c c6 25 27 49
      71 00 2c 16 68 e5 3b 9e 1e 66 44 4e e9 dc d4 8a}
```

Ableitung des HMAC-Schlüssels K_A vom Passwort „password“ mittels Algorithmus PBKDF2_XORCRYPT:

```
P  = {70 61 73 73 77 6f 72 64}   # "password"

SA = {74 9b 93 2e 72 0b 6d e8}
```

```
KA = {df 3e 6a 61 9c 78 48 96 d1 84 6e a1 53 2e a3 f6
      59 80 c6 aa e0 bf 7f 45 af a3 10 cb 52 e9 f3 87}
```

4.2 Verschlüsselung

Im Folgenden sei ein Beispiel für die Verschlüsselung und Integritätssicherung einer 25 Byte großen Datei mittels XorCrypt gegeben. Verwendung findet das leere Passwort der Länge 0. Die davon abgeleiteten Schlüssel für AES (K_E) und HMAC (K_A) entsprechen den Beispielen im vorangegangenen Kapitel.

```
DATEI_IN  = {44 69 65 73 20 69 73 74 20 65 69 6e 65 20 54 65
             73 74 2d 44 61 74 65 69 2e}

            # "Dies ist eine Test-Datei."

DATEI_OUT = {d8 bc 3e 25 b4 81 0c ee 08 65 99 c8 3c fe f4 75
             d2 1a bd 55 14 eb c0 70 74 9b 93 2e 72 0b 6d e8
             8f 32 b8 00 c0 7d 72 90 9a 2d b1 ee a0 29 9c 8b
             1d f2 1a 26 8f 49 b7 4d ca 2f ca fe 95 64 6c 8c
             84 99 42 26 3f ff 99 bc 8b 98 0a 76 6a 09 f4 63
             ed b3 60 fc fc 86 9c f3 fd}
```

5 Referenzen

[AES] FIPS PUB 197 "Announcing the ADVANCED ENCRYPTION STANDARD (AES)", 26.11.2001

[CTR] NIST Special Publication 800-38A "Recommendation for Block Cipher Modes of Operation, Methods and Techniques", 2001

[SHA] FIPS PUB 180-4 "Secure Hash Standard (SHS)", August 2015

[HMAC] FIPS PUB 198-1 "The Keyed-Hash Message Authentication Code (HMAC)", Juli 2008

[PBKDF2] NIST Special Publication 800-132 "Recommendation for Password-Based Key Derivation, Part 1: Storage Applications", Dezember 2010