Ether – Konzept

Felix Widmaier

Abstract Mit Ether soll zunächst ein „Konzeptprogramm“ zur sicheren Verschlüsselung von Dateien implementiert werden. Dabei soll das Programm ähnlich wie bei einer *One-time-Pad*-Verschlüsselung für jede Datei einen individuellen Schlüssel verwenden. Alle Schlüssel, welche Ether für das Verschlüsseln der Dateien verwendet sollen in einer *Schlüsseldatei* gespeichert werden. Die *Schlüsseldatei* soll dabei ebenfalls mittels Passwortabfrage gegen ungewolltes Auslesen der Schlüssel geschützt werden.

Ver-/Entschlüsselung

Zunächst werden alle Bytes mit den Zeichen des Schlüssels mittels XOR verschlüsselt. Allerdings wird der Schlüssel dabei „gespiegelt[1]“. Im Anschluss werden die bereits mit XOR verschlüsselten Bytes nochmals mittels Vigenére verschlüsselt.
Bei der regulären Verschlüsselung einer Datei wird ein zufällig generierter Schlüssel verwendet. Dieser besteht aus 150 Zeichen aus der ASCII-Tabelle (also aus 150 Bytes). Wird die Schlüsseldatei verschlüsselt, so wird als Schlüssel das zuvor festgelegte Passwort des Benutzers als Schlüssel verwendet. Seien b also die Bytes der zu verschlüsselnden Daten. b_i bezeichne den i-ten Byte. Weiterhin sei s die Länge der Daten. k sei der Schlüssel der Länge ℓ. k_i sei der i-te Byte des Schlüssels. Ether verschlüsselt die Daten dann mit der Funktion

$$v(b_i) = \left[b_i \oplus k_{(\ell - i) \mod s}\right] + k_{i \mod s} \mod 256. \tag{1}$$

Analog entschlüsselt man mit

$$g(b_i) = [b_i - k_{i \mod s} \mod 256] \oplus k_{(\ell - i) \mod s}. \tag{2}$$

Wobei wir mit $\oplus$ den XOR-Operator notieren.

Beweis. Es gilt

$$\begin{aligned}(g \circ v)(b_i) &= \left[\left(b_i \oplus k_{(\ell - i) \mod s}\right) + k_{i \mod s} - k_{i \mod s}\right] \oplus k_{(\ell - i) \mod s} \mod 256 \\ &= \left(b_i \oplus k_{(\ell - i) \mod s}\right) \oplus k_{(\ell - i) \mod s} \mod 256 \\ &= b_i \mod 256 = b_i\end{aligned}$$

Da $0 \leq b_i \leq 255$. □

[1] Aus `Hallo` wird bspw. `ollaH`

Die Schlüsseldatei

In der *Schlüsseldatei* werden alle Schlüssel zum Ver- und Entschlüsseln der einzelnen Dateien gespeichert. Weiterhin werden Prüfsummen[2] der jeweiligen Dateien gespeichert, um später sicherzustellen, dass die Datei richtig wiederhergestellt wurde. Weiterhin wird die Prüfsumme der verschlüsselten Datei gespeichert, um dieser später beim Entschlüsseln den korekten Schlüssel zuzuordnen.

Passwortsicherheit Wird eine neue Schlüsseldatei vom Benutzer angelegt, so wird er dazu aufgefordert ein Passwort einzugeben. Dieses Passwort dient der sicheren Speicherung der Schlüssel. Es soll sichergestellt werden, dass nur Personen, die über das Passwort verfügen, diese Datei korekt auslesen können. Weiterhin ist der letzte Byte der Schlüsseldatei ein CRC[3] des Passworts mit einem festgelegten Polynom. Das CRC-Polynom für den Passwort-CRC ist 10111011. Somit kann nach Eingabe des Passworts zunächst der Inhalt der Schlüsseldatei mittels der Eingabe „entschlüsselt“ werden. Im Anschluss wird mit dem Passwort-CRC ermittelt, ob ein Fehler vorliegt. Liegt ein Fehler vor, so wird dem Benutzer der Zugriff auf die Datei verwehrt, da das Passwort falsch ist. Dabei bietet CRC in diesem Falle mehrfachen Schutz: 1) ist die Eingabe inkorrekt, so wird der Passwort-CRC falsch entschlüsselt und CRC erkennt einen Fehler. 2) stimmt die Eingabe nicht mit dem gespeicherten CRC überein, so wird ein Fehler erkannt und dem Beutzer wird der Zugriff verwehrt.

Manipulationssicherheit Im vorletzten Byte der Schlüsseldatei wird ein weiterer CRC abgelegt. Der *sog.* Inhalts-CRC. Dieser CRC wird aus den abgelegten Daten und einem festen Polynom, dem Inhalts-Polynom, ermittelt. Das Inhalts-Polynom ist 10011010. Nach der erfolgreichen Eingabe des Passworts wird mit dem Inhalts-CRC nochmals der Inhalt auf Fehler geprüft. Somit können unerwünschte Manipulationen in der Datei erkannt werden. Wird erkannt, dass diese Datei manipuliert wurde, wird sie von Ether abgelehnt. Somit kann nochmals abgesichert werden, dass das Passwort korrekt eingegeben wurde.

Daten Die Daten liegen als `dict` (dictionary), also einem Python-Datentypen vor. Die Daten werden zunächst mit dem Modul `pickle` zu einem Bytestream konvertiert. Der Vorteil an `pickle` ist, dass aus ebendiesen Bytestreams ein Python-Objekt direkt geladen werden kann.
Also ist die Datei wie folgt aufgebaut:

...daten,daten,Inhalts-CRC,Passwort-CRC

Alle diese Bytes werden beim Speichern mit dem Passwort verschlüsselt.

[2] sha256 Prüfsummen
[3] Wird auch mitverschlüsselt.