IT-Sicherheit
WS 2019/20
Prof. Dr. Elmar Böhler

Informatik Bachelor
Hochschule Kempten

Übungsblatt 3

Im Moodle finden Sie eine Datei `sha1.zip`, die die Quelldateien `sha1.h`, `sha1.c` und `crack.c` enthält. In `sha1.h` ist eine Struktur `bitBlock` enthalten mit der beliebige Daten in einem `uint32`-Array gespeichert werden können. Die Bibliothek bietet einige Funktionen an, die hoffentlich einigermaßen selbsterklärend sind. Für diese Übung benötigen Sie vor allem die folgenden Funktionen:

- `bitBlock *forChars(char *msg)`
  Eine Funktion, die einen 0-terminierten ASCII-String in einem `bitBlock` speichert. Die Funktion codiert jeden `char` als 8-Bit-Wert und reiht diese Bytes der Reihe nach in die `uint32`-Elemente des `data`-Arrays des `bitBlocks`. Die Funktion reserviert dazu genau so viel Speicher, wie nötig sind um `msg` unterzubringen. Die Verwaltungsvariablen `wordCount` und `usedBits` werden entsprechend initialisiert.

- `uint32 *sha1(bitBlock *message)`
  Die Funktion berechnet den Hashwert der gegebenen `message`, für den SHA-1 Hashalgorithmus. Dieser Algorithmus gilt als nicht mehr sicher, ist für unsere Übung aber völlig ausreichend. Hashwerte, die von SHA-1 produziert werden, haben immer 160 Bits, folglich hat das zurückgelieferte Array die Größe 5. Die Funktion gibt den Speicherplatz für `message` nicht frei, das müssten Sie selbst erledigen, falls Sie viele Messages hashen wollen (was Sie auch sollen ;)).

In `crack.c` finden Sie eine einfache Datenstruktur `wordVec` mit der eine Liste von Wörtern verwaltet werden kann. Sie bietet Funktionen zum Hinzufügen von Wörtern, so dass man sich nicht groß um die Speicherverwaltung kümmern muss, aber nicht viel mehr. Sie können diese Datenstruktur verwenden, müssen aber nicht. Stattdessen können Sie auch eine beliebige, geeignete Datenstruktur aus irgendwelchen Standardbibliotheken verwenden. (Letzteres gilt nur für die Datenstruktur zum Halten der Passwörter; externe Bibliotheken zum hacken von Passwörtern dürfen Sie z.B. explizit nicht verwenden.)

1. Schreiben Sie eine Funktion mit dem Kopf

   ```
   char* bruteForceCrack(uint32* sha1Hash, char* alphabet, uint8 alphabetSize)
   ```

   welche ein Passwort zu den in `sha1Hash` gegebenen Passwort sucht. Implementieren Sie dazu einen Algorithmus, der für alle möglichen Passwörter, die über das gegebene `alphabet` bildbar sind, jeweils einen SHA-1-Hashwert berechnet und mit dem Eingabehash vergleicht.

   (a) Probieren Sie Ihre Funktion zunächst an kürzeren Passwörtern mit einem kleinen Alphabet aus. (Beispiele für Alphabete: $\{0, 1\}$, $\{0, \ldots, 9\}$, $\{a, \ldots, z\}$) Probieren Sie sie dann an längeren Passwörtern aus und erhöhen Sie auch die Anzahl der Symbole im Alphabet.

   (b) Erstellen Sie eine Tabelle, in deren Zeilen Sie die Anzahl der Symbole im Alphabet und in deren Spalten Sie die Passwortlänge auftragen. Tragen Sie die Wartezeiten, bis ein Passwort gecrackt ist in die Tabelle ein. Wartezeiten über einer Minute müssen Sie nicht mehr eintragen.

   (c) Nutzen Sie Ihre Funktion um die Passwörter zu den Hashes

   ```
   65caa18f6f33d5e89493dc608eb0055126c34997
   d27eb55673c666c0c12873cc6ed592bfe59ff958
   fe5f81641092ee46c40e84c4493a757aadc3e65e
   ```

   zu finden.

2. In dieser Aufgabe sollen Sie eine Form des Wörterbuch-Angriffs durchführen. Im Moodle finden Sie dazu eine weitere Datei `dic-0294.zip`, die 869229 gängige mögliche Passwörter enthält
   (Quelle: `http://www.outpost9.com/files/WordLists.html`).

(a) Schreiben Sie eine Funktion mit dem Kopf

```
char *dictCrack1(uint32 *sha1Hash, wordVec *wv)
```

die ein Passwort zum gegebenen Hash zu finden versucht, indem die `pwList` durchgegangen, für jedes enthaltene Wort ein SHA-1-Hash gebildet und mit dem gegebenen Hash verglichen wird. Nutzen Sie Ihre Funktion um die Passwörter zu den Hashes

```
de145a203f49e2ef682b434d4e9d241dfe7c4b4f
c2ba266015990e8229267fbb1121568b4a5e981d
dd3deb7b9e98f6d189e5c36508b0c55e1697195a
```

zu finden.