Aufgabenblatt #1: Symmetric Cryptography (15 Punkte)

Beachten Sie bitte die allgemeinen Hinweise am Ende des Aufgabenblattes.

1.1 Cryptanalysis (5 Punkte)

In dieser Aufgabe sollen Sie simple, historische Verschlüsselungen brechen:

- SPILUALY OVTPULZ PK XBVK CVSBUA JYLKBUA
  – Julius Caesar (1 Punkt)
- 44233443155223342533345223345244345224331142153245132332344 2
  153345321542344543442311334423344315522334253334522334524434
  32112515354234351542454315342144231524425124134434422415 43
  – Polybius (1.5 Punkte)
- `cgzno yjqgq voucu ,xnsh msgzo qgqvo ucucu jdsuj mwobx jdsxn shmsg zobxv sjjsn ubnen bmyub xvsjj snucg qzcyd snjsl j.jds tsjdb wcumu swquq gqcwj bansq fcgez vquuc zqvzc ydsnu .xnsh msgzo qgqvo ucucu aqusw bgjds xqzjj dqj,c gqgoe cisgu jnsjz dbxkn cjjsg vqgem qes,z snjqc gvsjj snuqg wzbta cgqjc bgubx vsjjs nubzz mnkcj diqno cgexn shmsg zcsu. tbnsb isn,j dsnsc uqzdq nqzjs ncujc zwcuj ncamj cbgbx vsjjs nujdq jcunb medvo jdsuq tsxbn qvtbu jqvvu qtyvs ubxjd qjvqg emqes .xbnc gujqg zs,ec isgqu szjcb gbxsg evcud vqgem qes,s ,j,qq gwbqn sjdst bujzb ttbg, kdcvs p,hqg wlqns nqns. vcfsk cus,j d,sn, bg,qg wqgqn sjdst bujzb ttbgz bttbg yqcnu bxvsj jsnu(jsnts wacen qtubn wcenq ydu), qgwuu ,ss,j j,qgw xxqns jdstb ujzbt tbgns ysqju .`

  Tipp: Englische Häufigkeitsanalyse (2.5 Punkte)

Geben Sie jeweils den Klartext, den Schlüssel und den Algorithmus an und beschreiben Sie kurz Ihren Lösungsweg. Schreiben Sie für den letzten Aufgabenteil ein Tool das die Häufigkeitsanalyse automatisch durchführt.

1.2 Unix Password Hashes (5 Punkte)

Unter Unix werden Benutzer-Informationen in der Datei `/etc/passwd` verwaltet. Ein Eintrag dieser Datei hat folgenden Aufbau:

```
User:Password-Hash:UID:GID:Comment:Home:Shell
```

Beispiel: `dummy:x:1001:1001::/home/dummy:/bin/bash`

- Aus Sicherheitsgründen wird statt dem Passwort-Hash heute meist nur ein "x" in der `/etc/passwd` gespeichert und der Passwort-Hash selbst in `/etc/shadow`. Warum? Welchen Aufbau hat ein Eintrag in der `/etc/shadow`? (0.5 Punkte)
- Ein Unix Passwort-Hash hat folgenden Aufbau:

  ```
  $Algorithm$Salt$Hash
  ```

  Beispiel: `$1$F7MoAVLF$rSng.D7mkCHszE3bXweRP0`

  Was versteht man unter einem "Salt"? Wieso erhöht dieser die Sicherheit? (0.5 Punkte)
- Die Kodierung `$1$` steht für einen MD5 basierten Algorithmus, `$5$` für SHA-256 und `$6$` für SHA-512. Sie können sich im Folgenden auf MD5 beschränken.

  Schreiben Sie ein C-Programm, das zu gegebenen Passwort und Salt einen Hash-String wie in der `/etc/shadow` berechnet:

  ```
  #./genpw F7MoAVLF
  Enter UNIX password: 0123
  $1$F7MoAVLF$rSng.D7mkCHszE3bXweRP0
  ```

  Nutzen Sie die C-Funktion *crypt(3)*; Sie brauchen den Hash-Algorithmus also nicht selbst zu implementieren. (1.5 Punkte)

- Schreiben Sie nun basierend auf der Lösung zum letzten Aufgabenteil einen Passwort-Cracker, der per Brute-force 4 Zeichen lange Passwörter knackt (alphanumerisch, keine Sonderzeichen, d.h. [a-zA-Z0-9]).

  Die Passwort-Hashes sollen aus einer `shadow`-Datei zeilenweise eingelesen und gecrackt werden. Systemnutzer ohne Passwort sollen übersprungen werden:

```
# ./crackpw shadow
Skipping daemon
Skipping bin
Skipping sys
Cracking password for root: XXXX
Password found.
Skipping mail
Skipping news
Cracking password for dummy: 0123
Password found.
Cracking password for student: XXXX
Password found.
```

  Unter `/proj/i4syssec/ex/1/2/` finden Sie eine `shadow`. Wie lauten die Passwörter von *root* und *student*? (2.5 Punkte)

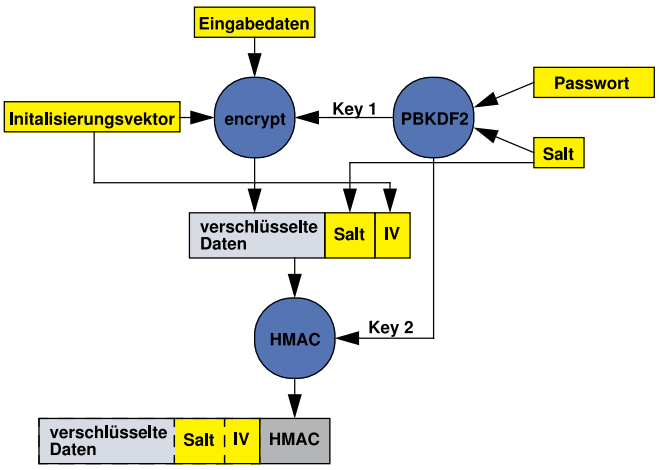
  Hinweis: *student* hat nur Buchstaben verwendet; *root* war vorsichtiger und hat auch Zahlen verwendet.

1.3 AES File Encryption (5 Punkte)

Implementieren Sie eine passwortbasierte Dateiverschüsselung, die wie folgt arbeitet:

```
$./filecrypt enc test.txt test.enc.txt
Enter Password: geheim
$./filecrypt dec test.enc.txt text.txt
Enter Password: geheim
```

Hierbei soll aus einem Passwort und einem zufälligen 160 Bit Salt durch Anwenden des in OpenSSL implementierten PBKDF2-Algorithmus mit 2000 Iterationen ein 512 Bit langer Schlüssel erzeugt werden, welcher zur weiteren Verwendung in zwei 256 Bit Schlüssel (*Key1* und *Key2*) aufgeteilt wird. *Key1* und ein zufälliger 128 Bit langer Initialisierungsvektor (IV) dienen als Eingabeparameter des aes-256-ofb Algorithmus, mit welchem der Dateiinhalt verschlüsselt wird.
Das Tripel verschlüsselte Daten, Salt, IV soll durch einen 160 Bit langen SHA-1 HMAC geschützt werden, wobei der durch PBKDF2 generierte *Key2* als Schlüssel dienen soll. Als Ausgabe soll das Programm das Quadtrupel verschlüsselte Daten, Salt, IV, HMAC in der Zieldatei ablegen.
Bei der Entschlüsselung soll das Passwort überprüft werden, bevor der Entschlüsselungsalgorithmus initiiert wird (es sollen also nur PBKDF2 und HMAC berechnet werden).



Parameter im Überblick:

```
#define AESKEYLEN 32          /* Key length of aes-256-ofb */
#define HMACKEYLEN 32         /* Key length for hmac */
#define SALTLEN 20            /* Salt for PBKDF2 to derive key from pw */
#define IVLEN 16              /* IV for aes-256-ofb, 128 bit */
#define HMACLEN 20            /* HMACLEN is 160 bit, as it shall base on SHA1 */
#define PBKDF2ITERATIONS 2000 /* Number of iterations */
```

1.4 Zusatzaufgabe: Advanced Password Cracker (bis zu 5 Punkte)

In dieser Aufgabe können Sie die Funktionalität Ihres Passwort Crackers aus Aufgabe 2 erweiteren. Einige Anregungen:

- Passwörter beliebiger Länge und mit beliebigem Zeichensatz
- Zeitmessung
- Wörterbuch-Attacke
- Muster-basierte Angriffe (bspw. Zahlen am Ende; "sicher2010")
- Andere Hash-Algorithmen (SHA-256 und SHA-512)
- etc.

Jedes neue Feature wird dem Aufwand entsprechend entlohnt (insgesamt max. 5 Punkte). Zeigen Sie Ihre neuen Features jeweils an einem Testlauf.

Allgemeine Hinweise

Die Übungen sollen in Gruppen von 2-3 Studierenden bearbeitet werden. Sobald Sie sich zu einer Gruppe zusammengeschlossen haben, melden Sie sich mit dem Programm `/proj/i4syssec/bin/syssecgroups` bei uns an. Daraufhin erhalten Sie eine URL, einen Benutzernamen und ein Kennwort mit denen Sie Zugriff auf Ihr SVN-Repository erhalten.
Die für diese Übungen relevante Dateien finden Sie im CIP-Pool unter `/proj/i4syssec/ex/$(BLATT)/$(AUFGABE)`, also bspw. unter `/proj/i4syssec/ex/1/2/`. In `/ex/` haben Sie nur Leserechte; kopieren Sie sich die Dateien in Ihr Home-Verzeichnis um mit Ihnen arbeiten zu können. Die Abgabe Ihrer Lösungen geschieht über Ihr SVN-Repository. Legen Sie die Lösungen zu Aufgabe 1.2 also bspw. in `$REPOSITORY/1/2/` ab. Dies betrifft nicht nur die Programmieraufgaben, sondern auch inhaltliche Fragen. Erstellen Sie in solchen Fällen bitte einfache Text-Dateien (nur falls nötig, bspw. wegen Bildern, werden auch PDFs akzeptiert).
Beachten Sie, dass alle von Ihnen abgegeben Programmieraufgaben im CIP-Pool übersetz- und ausführbar sein müssen. Falls Ihnen ein bestimmtes Paket fehlt, melden Sie sich bitte und wir werden es nachinstallieren. Die Programmiersprache in diesem Kurs ist üblicherweise C; nutzen Sie zur Übersetzung Ihrer Projekte das GNU Make System.
Fügen Sie jedem Projekt außerdem eine Datei README hinzu, in der Sie Ihr Programm dokumentieren, sowie einen beispielhaften Aufruf und Testlauf Ihres Programms zeigen.

Deadline: 26.11.2010