– Lösung der Praktikumsaufgabe 2 –

Thema: *Knacken historischer Chiffren, triviale XOR-Verschlüsselung*

1. Die Auftrittshäufigkeiten ermitteln wir mit dem `countchars`-Werkzeug:

```
robge@ipaetz2:~/is$ ./countchars geheimmono.txt | sort -r -n -k 3
Y : 36
B : 21
Z : 16
T : 12
H : 12
N : 11
G : 11
[...]
```

   Gemäß der Autrittswahrscheinlichkeit von Buchstaben in der deutschen Sprache können wir mit einiger Sicherheit annehmen, dass Y das 'e' kodiert[1], wahrscheinlich auch, dass B für 'n' steht und Z für 'i'. Probieren wir diese Substitution an der Kommandozeile mittels `tr`-Kommando einmal aus:

```
robge@ipaetz2:~/is$ tr A-Z AnCDEFGHIJKLMNOPQRSTUVWXei < geheimmono.txt
   eine einKJeNGJnO in Xie MiHHenHWNUKI LCA
 LeGHWNTJeHHeTn, LeGVeGOen JnX LeGNeiATiWNen
CNne UTTe OeNeiAniHRGUeAeGei, UVeG niWNI CNne
  NinIeGTiHIiOen HWNUTR, XUGOeHIeTTI SJA nJISen
    JnX eGOCeISen XeH UTTOeAeinen DJVTiRJAH.
```

   Nun kann man versuchsweise weitere Substitutionen vornehmen, bis man zunächst einzelne Worte und letztlich das gesamte Chiffrat entschlüsselt hat. Wir erraten X→d, J→u, C→o und N→h und erhalten:

```
robge@ipaetz2:~/is$ tr A-Z AnoDEFGHIuKLMhOPQRSTUVWdei < geheimmono.txt
   eine einKuehGunO in die MiHHenHWhUKI LoA
 LeGHWhTueHHeTn, LeGVeGOen und LeGheiATiWhen
ohne UTTe OeheiAniHRGUeAeGei, UVeG niWhI ohne
  hinIeGTiHIiOen HWhUTR, dUGOeHIeTTI SuA nuISen
    und eGOoeISen deH UTTOeAeinen DuVTiRuAH.
```

   Die finale Entschlüsselung lautet:

```
robge@ipaetz2:~/is$ tr A-Z mnopEFrstufvwhgPQkzlabcdei < geheimmono.txt
   eine einfuehrung in die wissenschaft vom
 verschluesseln, verbergen und verheimlichen
ohne alle geheimniskraemerei, aber nicht ohne
  hinterlistigen schalk, dargestellt zum nutzen
    und ergoetzen des allgemeinen publikums.
```

2. Ein Lösungsvorschlag ist das Programm in Listing 1.

Listing 1: Werkzeug für den Kasiski-Test

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

[1] Als Konvention soll gelten, dass Großbuchstaben zum Chiffrat und Kleinbuchstaben zum Klartext gehören.

```c
#include <ctype.h>

#define MINLENGTH 3
#define MAXLENGTH 10

char *cipherbuf;

void read_in_cipher(char *fname)
{
  FILE *fin;
  int c, pos;
  long length;

  fin = fopen(fname, "r");
  if (!fin) {
    perror("fopen()");
    exit(EXIT_FAILURE);
  }
  fseek(fin, 0, SEEK_END);
  length = ftell(fin);
  cipherbuf = malloc(length+1);
  if (!cipherbuf) {
    perror("malloc");
    exit(EXIT_FAILURE);
  }
  rewind(fin);

  pos = 0;
  while (!feof(fin)) {
    c = fgetc(fin);
    if (isalpha(c)) {
  cipherbuf[pos++] = c;
      }
  }
  fclose(fin);
  cipherbuf[pos] = 0;
}

void print_buffer(char *buf)
{
  int pos = 0;

  while (cipherbuf[pos]) {
    printf("%c", cipherbuf[pos]);
    pos++;
    if (pos%125 == 0) {
      printf("\n\n");
    }
    else {
      if (pos%25 == 0) {
  printf("\n");
      }
      else {
  if (pos%5 == 0) {
    printf(" ");
```

```
  }
      }
    }
  }
  printf("\n\n");
}

void search_ngrams(char *buf)
{
  char ngram[11];
  int ngram_lgth, startpos, dist;
  char *ret;

  for (ngram_lgth=MAXLENGTH; ngram_lgth>=MINLENGTH; ngram_lgth--) {
    printf("\nngram length = %d\n", ngram_lgth);

    startpos = 0;
    while(cipherbuf[startpos] != 0) {

      /* define ngram to search for */
      strncpy(ngram, &cipherbuf[startpos], ngram_lgth);
      ngram[ngram_lgth] = 0;

      /* do the search for ngram in (rest of) cipherbuf */
      ret = strstr(&cipherbuf[startpos]+ngram_lgth, ngram);
      while (ret) {
  dist = ret - &cipherbuf[startpos];
  printf("found %s: pos1 %d, pos2 %d, distance %d\n",
         ngram, startpos, startpos+dist, dist);
  ret = strstr(ret+1, ngram);
      }
      startpos++;
    }
  }
}

int main(int argc, char* argv[])
{
  read_in_cipher(argv[1]);
  print_buffer(cipherbuf);
  search_ngrams(cipherbuf);
  /* now we know with a little luck the length of the key */
  free(cipherbuf);
  exit(EXIT_SUCCESS);
}
```

Wenn ein n-Gramm der Länge n mehrmals auftaucht, tauchen n-Gramme der Längen $n-1$, $n-2$ usw. natürlich auch mit dieser Distanz mehrfach auf. Für die gegebene Datei `vigenere.txt` ermittelt das Programm die folgenden einzigartigen Abstände für n-Gramme:

```
found MSYQHVBW: pos1 32, pos2 267, distance 235
found MZXJOAZ: pos1 149, pos2 314, distance 165
found CJMZQ: pos1 11, pos2 166, distance 155
```

```
found YQHVB: pos1 269, pos2 299, distance 30
found OYQH: pos1 288, pos2 298, distance 10
```

Da die Schlüssellänge alle diese Abstände teilen muss, ist eine ziemlich sichere Hypothese für die Schlüssellänge der Wert 5.

3. a)
   - Datei mittels `Datei` → `Öffnen` in ein neues Fenster laden
   - `Analyse` → `Symmetrische Verschlüsselung (klassisch)` → `Ciphertext-Only` → `Substitution`
   - `Verfahren 1` wählen

   Auf Grund der geringen Textlänge stimmen die erwarteten Auftrittswahrscheinlichkeiten nicht mit den tatsächlichen überein. Obwohl die meisten Substitutionen automatisiert korrekt ermittelt wurden, muss nun mit `Manuelle Analyse` fortgesetzt werden, um die restlichen Substitutionen aus dem Kontext zu erschließen.

   Der entschlüsselte Klartext lautet:

   EINE EINFUEHRUNG IN DIE WISSENSCHAFT VOM
   VERSCHLUESSELN, VERBERGEN UND VERHEIMLICHEN
   OHNE ALLE GEHEIMNISKRAEMEREI, ABER NICHT OHNE
   HINTERLISTIGEN SCHALK, DARGESTELLT ZUM NUTZEN
   UND ERGOETZEN DES ALLGEMEINEN PUBLIKUMS.

   b) `Analyse` → `Symmetrische Verschlüsselung (klassisch)` → `Ciphertext-Only` → `Vigenere`

   Das Brechen des Chiffrates funktioniert vollautomatisch: zunächst wird die Schlüssellänge von 5 ermittelt, dann der Schlüssel („MURKS“) und zuletzt der komplette Klartext:

   > „Der franzoesische Diplomat Blaise de Vigenere war sechsundzwanzig, als er waehrend einer zweijaehrigen Mission in Rom auf die Schriften Albertis stiess. Anfangs hatte er aufgrund seiner diplomatischen Taetigkeit nur praktisches Interesse an der Kryptographie. Dann lm Alter von neununddreissig Jahren beschloss Vigenere, dass er nun genuegend Vermoegen erworben habe, und sein Leben kuenftig der Wissenschaft widmen wolle.“

   c)* Bei einem unbekannten Chiffrat ist es immer sinnvoll, die Zeichenhäufigkeit sowie den Friedman-Index zu bestimmen.

```
robge@ipo51:~/txt/job/htw/is/src/prak02$ friedman unbekannt.txt
A : 38
B : 14
C : 21
D : 18
E : 73
F : 8
```

```
G : 18
H : 35
I : 35
J : 1
K : 8
L : 17
M : 16
N : 49
O : 14
P : 8
Q : 0
R : 28
S : 39
T : 45
U : 28
V : 7
W : 7
X : 0
Y : 0
Z : 8
n = 535.000000
kappa = 0.065050
```

Da $\kappa = 0.065$, handelt es sich offenbar um keine polyalphabetische Verschlüsselung (bzw. es sind nur sehr wenige Alphabete beteiligt). Da die Auftrittswahrscheinlichkeiten der beteiligten Buchstaben grob denen der deutschen Sprache entsprechen und im Chiffrat Umlaute zu finden sind, liegt der Schluss nahe, dass es sich um eine Transpositionschiffre eines deutschen Klartextes handelt.

In der Fußnote steht, dass es sich um eine Geocache-Beschreibung handelt. Geocaches sind kleine Dosen, die an bestimmten Orten versteckt und durch die Teilnehmer mittels GPS-Gerät gesucht werden. Hier ist zuvor das Rätsel zu lösen, d.h., die Verschlüsselung zu brechen. Der Klartext muss die GPS-Koordinate enthalten, diese hat in unseren Breiten ungefähr folgendes Format:

```
N50°xx.yyy E012°mm.nnn
```

Dieses Wissen ist der Schlüssel zum Brechen der Chiffre. Die Koordinaten stehen abwechselnd an den Positionen 374ff. und 76ff., gezählt von 0:

Pos.	374	76	375	77	376	78	377	79	378	80	379	81	380	82	381	83	382	84	383	85	384
plain	N	5	0	°	4	8	.	1	0	7	E	0	1	2	°	3	8	.	3	2	1

Als nächstes „erweitern“ wir die Tabelle nach links bis zur Position 0 und schreiben den Text in 2 Zeilen:

```
HLOSHNDSDDNATNANJGRANHH2GKAKHS!AKNÖNIREHCEHLRCTATETR...
AL.CÖ,ASUELTEZU/ÄEZU(ÖE)ENCTATPREKNTHBIAKTAFUHSFKLEE...
```

Nun kann man den Text einfach spaltenweise lesen: ”Hallo. Schön, dass Du den Lattenzaun/Jägerzaun (Höhe 2) geknackt hast! [...]”

Wir haben es also mit der so genannten Jägerzaun-Verschlüsselung (auch: *Rail-fence* oder *Zigzag*) zu tun. Nähere Informationen finden Sie in der Literatur; die Sicherheit ist aufgrund der kleinen Menge an Schlüsseln äußerst gering. Die Schwierigkeit lag hier eher darin, das genutzte Verfahren zu identifizieren.

Die Entschlüsselung kann man noch in ein kleines C-Programm (vgl. Listing 2) gießen, muss jedoch zuvor das UTF-8-Chiffrat in die Latin1-Kodierung umwandeln, damit auch die Umlaute nur 1 Byte beanspruchen.

Listing 2: Dekoder für Transposition (Jägerzaun, Höhe 2)

```
/*
    decode-fence.c
    decode a fence of height 2
 */
#include <stdio.h>

char buf[1024*1024*1024];

int main(int argc, char *argv[])
{
  int lg = 0, i;

  while (!feof(stdin))
    buf[lg++] = getchar();
  printf("Ciphertext length: %ld\n", lg);

  for (i=0; i<lg/2; i++) {
    printf("%c%c", *(buf+i), *(buf+lg/2+i));
  }
  printf("\n");
}
```

```
robge@ipo51:~$ cp unbekannt.txt f
robge@ipo51:~$ recode 'u8..l1' f
robge@ipo51:~$ ./decode-fence < f > g
robge@ipo51:~$ recode 'l1..u8' g
robge@ipo51:~$ cat g
Ciphertext length: 599
HALLO.SCHÖN,DASSDUDENLATTENZAUN/JÄGERZAUN(HÖHE2)GEKNACKT
HAST!PARKENKÖNNTIHRBEIHACKETHALFRUCHTSAFTKELTEREI.DENRES
TMÜSSTIHRLEIDERZUFUSSGEHEN.BEGEBTEUCHZUN50°48.107E012°38
.321.PASSTBITTEAMMITTWOCHZWISCHEN17UND20UHR ...
```

Man kann den Text auch mittels CrypTool entschlüsseln , muss ihn aber genauso zuvor in die Latin1-Kodierung bringen. Danach Ver-/Entschlüsseln → Symmetrisch (klassisch) → Skytale / Gartenzaun, Anzahl Zeilen: 2, „Entschlüsseln“. Fertich, Meister.

Bei der Kryptoanalyse sucht man häufig nach 'Irregularitäten' im Chiffrat, so genannten 'Cribs', die einem kleine Hinweise auf die Verschlüsselung geben. Hier ist es das Wissen, dass eine GPS-Koordinate im Klartext enthalten sein muss, die zudem einen bestimmten, regulären Aufbau hat.

4. Wir besorgen uns zunächst `supercrypt` und überzeugen uns von dessen Funktionsfähigkeit.

```
$ wget http://www.informatik.informatik.htw-dresden.de/~robge/is/prak/src/sup
$ file supercrypt
supercrypt: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
dynamically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux
2.6.18, BuildID[sha1]=101e99b97897202346975044 2d4c72714ace1bd1, stripped
$ chmod +x supercrypt
$ echo "Das ist eine Testdatei, um dieses Supercrypt in die Knie zu \
zwingen." > plain.txt
$ ./supercrypt plain.txt cipher.txt
$ robge@ipo51:~/txt/job/htw/is/src/prak02/supercrypt$ ls -l
$ insgesamt 12
$ -rw-r--r-- 1 robge robge   69 Apr 10 11:48 cipher.txt
$ -rw-r--r-- 1 robge robge   69 Apr 10 11:48 plain.txt
$ -rwxr-xr-x 1 robge robge 3744 Mär 25  2013 supercrypt
$ robge@ipo51:~/txt/job/htw/is/src/prak02/supercrypt$ cat cipher.txt
(Binärdaten)
$ ./supercrypt cipher.txt plain2.txt
$ cat plain2.txt
Das ist eine Testdatei, um dieses Supercrypt in die Knie zu zwingen.
```

Der Funktionstest ist erbracht; `supercrypt` scheint zu verschlüsseln. Nun haben wir in Form der Dateien `plain.txt` und `cipher.txt` einen Klartext mit zugehörigem Chiffrat und können die beiden miteinander per XOR verknüpfen.

```
$ wget http://www.informatik.htw-dresden.de/~robge/is/prak/src/xor.c
[...]
$ gcc -o xor xor.c
$./xor
$ Usage: ./xor <keyfile> <plaintextfile> <cipherfile>
$ ./xor plain.txt cipher.txt result.txt
$ cat result.txt
We are the Knights who say 'Ni'!We are the Knights who say 'Ni'!We ar
```

Das Ergebnis der Verknüpfung ist der Schlüssel[2] „We are the Knights who say 'Ni'“, der sich periodisch wiederholt.

Bei derartig primitiven Implementierungen lohnt es sich häufig, in das Binary zu schauen, ob der Schlüssel möglicherweise im Klartext als Zeichenkette abgelegt ist.

```
$ strings supercrypt
[...]
Usage: %s <plaintext> <cipher>
Plain file
Cipher file
;*3$"
```

[2]Zusatzaufgabe: Zitatquelle ermitteln und Film ansehen!

```
Xf!bsf!uif!Lojhiut!xip!tbz!(Oj("
[...]
```

Das ist offensichtlich nicht der Fall, aber die letzte Zeichenkette ist auffällig. Erkennen Sie, mit welch billiger „Verschlüsselung“ der Schlüssel gesichert wurde?

5.* Das folgende Programm leistet das gewünschte:

Listing 3: Werkzeug zur Bestimmung der Autokorrelation

```c
#include <stdio.h>
#include <stdlib.h>

#define MAXDATA 1024*1024*1024
#define MAXSHIFT 64

int main(int argc, char *argv[])
{
  char *buf, *pt;
  unsigned long lgth, pos1;
  int c, dist, pairs;

  buf = malloc(MAXDATA);
  if (!buf) {
    perror("What? Not a single GiB for the user? So sad. Exit.");
    exit(EXIT_FAILURE);
  }
  pt = buf;
  while (!feof(stdin)) {
    *pt++ = getchar();
  }
  lgth = pt-buf-1;

  for (dist=1; dist<MAXSHIFT; dist++) {
    for (pos1=pairs=0; pos1<lgth; pos1++) {
      if (buf[pos1] == buf[(pos1+dist)%lgth]) {
          pairs++;
      }
    }
    printf("dist=%i, pairs=%u\n", dist, pairs);
  }
  free(buf);
  exit(EXIT_SUCCESS);
}
```

Wir müssen das Chiffrat herunterladen, `autokorr` übersetzen und aufrufen:

```
$ wget http://www.informatik.htw-dresden.de/~robge/.../xorchiffrat.txt
$ gcc -o autokorr autokorr.c
$ ./autokorr < xorchiffrat.txt
dist=1, pairs=3943
dist=2, pairs=3108
dist=3, pairs=4070
dist=4, pairs=16698
```

```
dist=5, pairs=4076
dist=6, pairs=3511
dist=7, pairs=3771
dist=8, pairs=13060
dist=9, pairs=3830
dist=10, pairs=3578
dist=11, pairs=3902
dist=12, pairs=13005
dist=13, pairs=4107
dist=14, pairs=3369
dist=15, pairs=3898
dist=16, pairs=13400
dist=17, pairs=3886
dist=18, pairs=3556
[...]
```

Lokale Maxima der Buchstabenpaare sind bei 4 und deren Vielfachen zu finden; daher ist sie eine (gute) Hypothese für die Schlüssellänge. Im `cryptool` können wir diese Verschlüsselung leicht vollständig brechen:

- `Analyse` → `Symmetrische Verschlüsselung (klassisch)` → `Ciphertext-Only` → `XOR/Vernam`

Der Schlüssel lautet „thor“, der Klartext ist das Bibelkapitel *Genesis* mit vertauschten Groß-/Kleinbuchstaben, Nullbytes anstelle Leerzeichen und vermurksten Umlauten; wahrscheinlich das Produkt einer fehlerhaften Verschlüsselung.

6. „James Brown is dead“