KAPITEL 12

# Ein Angriff auf MASC mit dem Computer

UNVOLLENDET

## 1. Einführung

Wir wiederholen kurz die MASC-Verschlüsselung.

**MASC-Verschlüsselung:** (**M**ono**a**lphabetische **S**ubstitutions**c**hiffrierung)

- Es werden nur Großbuchstaben A,B,C,...,Z berücksichtigt.
- **Schlüssel:**
  - Ein MASC-Schlüssel besteht aus einem Wort $k$, das aus 26 verschiedenen Großbuchstaben besteht. Den $i$-ten Buchstaben von $k$ bezeichnen wir mit $k[i]$. Sei
    $$\text{abc} = \text{ABCDEFGHIJKLMNOPQRSTUVWXYZ}.$$
  - Jedem Schlüsselwort $k$ ordnen wir eine Permutation $f_k$ von $\{\text{A}, \text{B}, \text{C}, \ldots, \text{Z}\}$ durch
    $$f_k(\text{abc}[i]) = k[i] \text{ für } i = 1, \ldots, 26$$
    zu. Man kann dies auch durch eine Tabelle angeben:

| $x$ | A | B | C | ... | Z |
|---|---|---|---|---|---|
| $f_k(x)$ | $k[1]$ | $k[2]$ | $k[3]$ | ... | $k[26]$ |

  - Die Menge der Schlüssel bildet den Schlüsselraum $\mathfrak{S}$. Es ist
    $$|\mathfrak{S}| = 26! = 403291461126605635584000000 \approx 0.4 \cdot 10^{27}$$
- **Verschlüsselung:** Ein Text $T = a_1a_2a_3\ldots$ wird mit dem Schlüssel $k$ zu $C = f_k(a_1)f_k(a_2)f_k(a_3)\ldots$ verschlüsselt. Wir schreiben dafür auch $f_k(T) = C$.
- **Entschlüsselung:** Ein mit dem Schlüssel $k$ verschlüsselter Chiffretext $C = b_1b_2b_3\ldots$ wird zu $T = f_k^{-1}(a_1)f_k^{-1}(a_2)f_k^{-1}(a_3)\ldots$ entschlüsselt. Wir schreiben dafür auch $f_k^{-1}(C) = T$.

**Überlegungen:**

- Wir haben einen MASC-Chiffretext $C$ und wollen ihn entschlüsseln (ohne den Schlüssel zu kennen), beispielsweise folgenden, aus 68 Zeichen bestehenden MASC-chiffrierten Text:
  $$C = \texttt{JPQVIAZQSMIAJQSNJLJWZJSLJAPIZFJPIZQBJAWSLFIZIPLAQZZJSPIJATEJNWAZRZIE}$$
- Wir können verschiedene Schlüssel $k$ wählen und damit entschlüsseln:

| Schlüssel $k$ | $f_k^{-1}(C)$ |
|---|---|
| POQFIEZCGDYBVWUHXJLMTKARNS | RACMEWGCZTEWRCZYRSRNGRZSRWAEGDRAEG |
| | CLRWNZSDEGEASWCGGRZAERWUFRYNWGXGEF |
| JOQYMTAZLBVUKXHEGPWFCNIRDS | ARCKWGHCZEWGACZVAIASHAZIAGRWHTARWH |
| | CJAGSZITWHWRIGCHHAZRWAGFPAVSGHXHWP |
| ZHXCMQABVDPLOFSYTJGRUKIENW | RKFIWGAFOEWGRFOYRLRZAROLRGKWANRKWA |
| | FHRGZOLNWAWKLGFAAROKWRGQXRYZGATAWX |

  Mit Computerhilfe können wir natürlich viele Schlüssel durchprobieren. Dabei müssen wir jedesmal nachschauen, ob der entschlüsselte Text nahe an einem deutschen Text ist. Dies ist bei vielen Schlüsseln praktisch unmöglich.
- Wir brauchen also eine **Bewertungsfunktion** $\mu$, die einem aus Großbuchstaben bestehenden Text eine Zahl zuordnet. Die Funktion $\mu$ sollte folgende Eigenschaft haben: Je näher ein Text an einem deutschen Text ist (bei vorgegebener Textlänge), desto größer sollte der Wert von $\mu$ sein.

Datei: masc_angriff.tex. Version vom 6.2.2025

- Haben wir eine Bewertungsfunktion $\mu$, so können wir bei gegebenem Chiffretext $C$ die Funktion

$$f : \mathfrak{S} \to \mathbb{R} \text{ mit } f(k) = \mu(f_k^{-1}(C))$$

betrachten. Ist $k_0$ der bei der Verschlüsselung verwendete Schlüssel, d.h. $C = f_{k_0}(T)$, so ist

$$f(k_0) = \mu(f_{k_0}^{-1}(C)) = \mu(T).$$

Schön wäre es, wenn $f$ in $k_0$ ein Maximum hätte. Es stellen sich zwei Fragen:

- **Frage 1:** Wie findet man geeignete Bewertungsfunktionen $\mu$?
- **Frage 2:** Wie findet man bei gegebener Bewertungsfunktion $\mu$ Maximalstellen der Funktion

$$\mathfrak{S} \to \mathbb{R}, \quad k \mapsto \mu(f_k^{-1}(C))\,?$$

## 2. Eine Bewertungsfunktion mit 4-Grammen

Wir beschreiben hier eine Bewertungsfunktion, die nachfolgend benutzt wird. Wir berücksichtigen in einem Text nur die Großbuchstaben A,...,Z.
Auf der Seite http://practicalcryptography.com/cryptanalysis/letter-frequencies-various-languages/german-letter-frequencies/ gibt es Tabellen mit Häufigkeiten von $n$-Grammen, wobei wir uns auf 4-Gramme beschänken. Wenn ich es richtig verstanden habe, wurden dafür in einem deutschen Text mit ungefähr 950 Millionen Zeichen (nach Umwandlung in Großbuchstaben) die Häufigkeit von 4-Grammen gezählt und in der Datei `german_quadgrams.txt` angegeben. Eine Zeile enthält also ein 4-Gramm $a_1a_2a_3a_4$ und daneben die (absolute) Häufigkeit $H(a_1a_2a_3a_4)$. Anfang und Ende der Datei `german_quadgrams.txt` sehen so aus:

```
EINE 3692344
NDER 2624230
ICHT 2401684
CHEN 2159090
SCHE 2058078
ENDE 1808194
LICH 1795107
SICH 1669502
ERDE 1585800
INDE 1552055
...
...
ÄÄAU 1
ÄAAU 1
ÄÄÄÖ 1
AAÄN 1
ÄÄAN 1
ÄÄÄM 1
ÄÄAL 1
ÄÄÄK 1
ÄÄÄC 1
ÄAAB 1
```

(Wir berücksichtigen Umlaute nicht.) Beispielsweise ist $H(\texttt{EINE}) = 3692344$. Kommt das 4-Gramm nicht vor, setzen wir $H(a_1a_2a_3a_4) = 0$. Damit definieren wir

$$h_4(a_1a_2a_3a_4) = \begin{cases} \log H(a_1a_2a_3a_4), & \text{falls } H(a_1a_2a_3a_4) > 0, \\ 0, & \text{falls } H(a_1a_2a_3a_4) = 0. \end{cases}$$

Für einen aus $n$ Großbuchstaben bestehenden Text $T = a_1a_2a_3a_4 \dots a_n$ definieren wir die Bewertung $\mu(T)$ durch

$$\mu(T) = \mu(a_1a_2a_3 \dots a_n) = \sum_{i=0}^{n-4} h_4(a_{i+1}a_{i+2}a_{i+3}a_{i+4}).$$

**Beispiel:** Für das Wort FREITAG schauen wir in der Datei `german_quadgrams.txt` nach den vorkommenden 4-Grammen:

```
FREI 249822
REIT 493025
EITA 188117
ITAG 105346
```

Damit ergibt sich

$$\mu(\text{FREITAG}) = \log(249822) + \log(493025) + \log(188117) + \log(105346) = 49.246643943430065$$

## 3. Der Schlüsselraum als topologischer Raum und Graph

Wir führen eine Metrik $d$ auf $\mathfrak{S}$ ein:

$$d(k_1, k_2) = |\{i : k_1[i] \neq k_2[i]\}|.$$

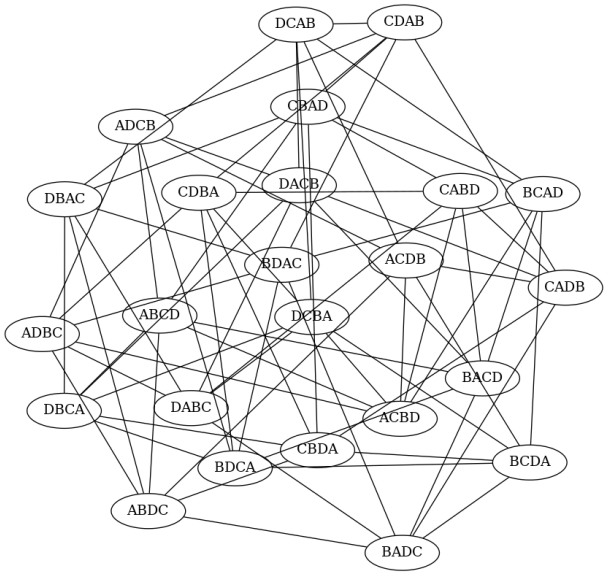
$d(k_1, k_2)$ gibt also an, an wievielen Stellen sich $k_1$ und $k_2$ unterscheiden.

Ist $k$ ein Schlüssel, der sich nur wenig vom zur Verschlüsselung verwendeten Schlüssel $k_0$ unterscheidet, so sollte $f_k^{-1}(C)$ nahe bei $f_{k_0}^{-1}(C) = T$ sein.

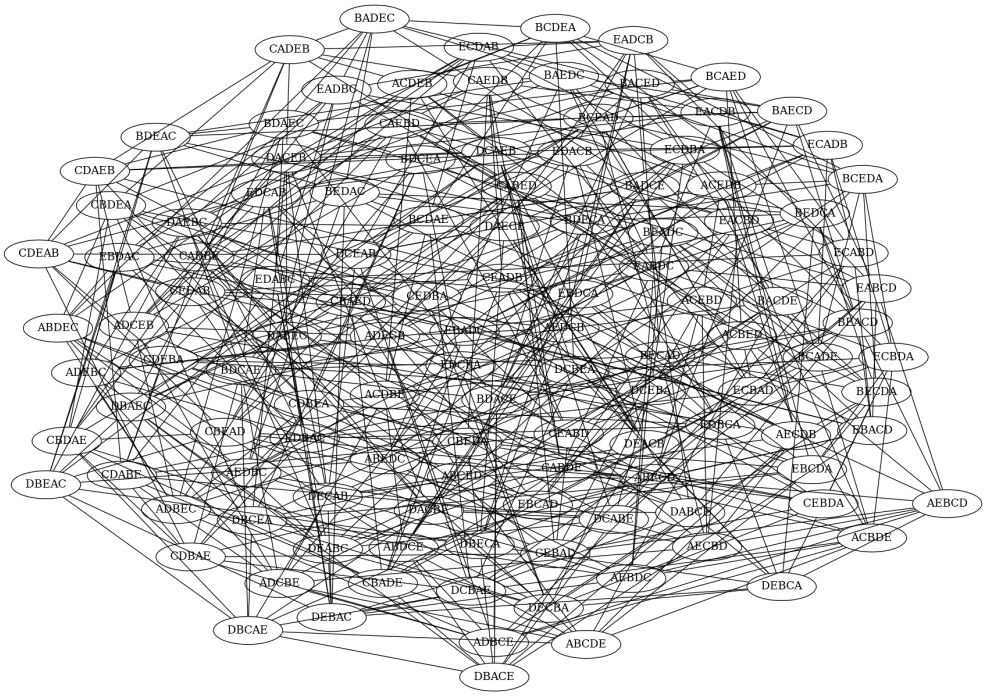
Im Folgenden nennen wir zwei Schlüssel $k_1$ und $k_2$ **benachbart**, wenn sie sich nur an zwei Stellen unterscheiden, d.h. wenn gilt $d(k_1, k_2) = 2$.

Wir können daher $\mathfrak{S}$ auch als Graphen auffassen: Die Ecken/Knoten des Graphen sind die Elemente von $\mathfrak{S}$, zwei Knoten werden durch eine Kante verbunden, wenn sie benachbart sind, d.h. wenn $d(k_1, k_2) = 2$ gilt.

**Beispiel:** Da $\mathfrak{S}$ groß ist, können wir $\mathfrak{S}$ nicht zeichnen. Daher betrachten wir beispielhaft $\mathfrak{S}_4$, die Permutationen von ABCD. Der zugehörige Graph sieht so aus:



Für $\mathfrak{S}_5$, den Permutationen von ABCDE, erhalten wir folgendes Bild:




Im Fall $\mathfrak{S}$ hat jeder Schlüssel $\binom{26}{2} = 325$ benachbarte Schlüssel.

### 4. Wie findet man eine Maximalstelle der Funktion $f : \mathfrak{S} \to \mathbb{R}$, $k \mapsto \mu(f_k^{-1}(C))$?

**Idee:** Wir suchen zunächst nach einem lokalen Maximum.

- Wir starten mit einem beliebigen Schlüssel $k \in \mathfrak{S}$ und bestimmen $f(k)$.
- Wir bestimmen die 325 Nachbarschlüssel $k_1, \ldots, k_{325}$ von $k$ und berechnen $f(k_1), \ldots, f(k_{325})$.
- Wir berechnen das Maximum

$$w_{\max} = \max(f(k_1), \ldots, f(k_{325})) \text{ und zugehörig } k_{\max} \text{ mit } w_{\max} = f(k_{\max}).$$

- Gilt $f(k_{\max}) > f(k)$, so beginnen wir mit $k_{\max}$ statt $k$ von vorne. Andernfalls ist in $k$ ein lokales Maximum und wir hören auf.

---

**Algorithmus** Bestimmung eines lokalen Maximums

---

**Eingabe:** Chiffretext $C$
**Ausgabe:** $f(k), k, f_k^{-1}(C)$, wobei in $k$ ein lokales Maximum von $f(k) = \mu(f_k^{-1}(C))$ vorliegt.

1: Wähle einen zufälligen Schlüssel $k_{\text{zufällig}} \in \mathfrak{S}$.
2: $k_2 \leftarrow k_{\text{zufällig}}$, $w_2 \leftarrow f(k_2)$, $w_1 \leftarrow w_2 - 1$
3: **while** $w_2 > w_1$ **do**
4: $\quad k_1 \leftarrow k_2$, $w_1 \leftarrow w_2$
5: $\quad$ Bestimme alle Nachbarn $\ell_1, \ldots, \ell_{375}$ von $k_1$.
6: $\quad$ Berechne $f(\ell_1), \ldots, f(\ell_{375})$.
7: $\quad$ Berechne $w_2 \leftarrow \max(f(\ell_1), \ldots, f(\ell_{375}))$
8: $\quad$ Bestimme einen Index $i$ mit $w_2 = f(\ell_i)$.
9: $\quad k_2 \leftarrow \ell_i$
10: **end while**
11: **return** $w_1, k_1, f_{k_1}^{-1}(C)$

---

Es stellt sich heraus, dass wir oft nur ein lokales Maximum erreichen. Deswegen wiederholen wir die lokale Suche mit verschiedenen zufällig gewählten Schlüsseln:

---

**Eingabe:** Chiffretext $C$

$w_{\max} \leftarrow -100000$
**for** $i = 1, \ldots, 100$ **do**
    $w, k, \text{text} \leftarrow$ Ergebnis der Bestimmung eines lokalen Maximums
    **if** $w > w_{\max}$ **then**
        $w_{\max}, k_{\max} \leftarrow w, k$
        Gib aus $w, k, \text{text}$
    **end if**
**end for**

---

Beim vorangegangenen Verfahren muss man selbst noch die Qualität des Ergebnisses beurteilen. Dies sollte noch verbessert werden.

## 5. Umsetzung mit Python

```
# masc_angriff - 5.2.2025
# MASC - MonoAlphabetischeSubstitutionsChiffrierung
# Es werden nur die Grossbuchstaben A,...,Z beruecksichtigt.
# Schluessel sind hier Woerter aus 26 verschiedenen Buchstaben.

# Schluesselergaenzung: Aus einem beliebigen Wort wird ein gueltiger
# Schlussel gemacht, wobei nur Grossbuchstaben beruecksichtigt werden.
def schluessel_ergaenzung(k0):
  abc='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
  k1=[k0[i] for i in range(len(k0)) if k0[i] not in k0[:i] and k0[i] in abc]
  i=list(abc).index(k1[-1])
  k2=k1+list(abc[i:])+list(abc[:i])
  k3=''.join([k2[i] for i in range(len(k2)) if k2[i] not in k2[:i]])
  return k3

# Erzeugung eines zufaelligen Schluessels
def zufallspermutation():
  abc='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
  k1=list(abc)
  import random
  random.shuffle(k1)
  k=''.join(k1[i] for i in range(26))
  return k

# MASC-Verschluesselung
def masc_ver(text,k):
  k=schluessel_ergaenzung(k)
  abc='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
  f={}
  for i in range(26):
    f[abc[i]]=k[i]
  ctext=''.join([f[z] for z in text if z in abc])
```

```
  return ctext

# MASC-Entschluesselung: k muss ein gueltiger 26-Zeichen-Schluessel sein.
def masc_ent(ctext,k):
  abc='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
  g={}
  for i in range(26):
    g[k[i]]=abc[i]
  text=''.join([g[z] for z in ctext])
  return text

# Hier werden die 325 Nachbarn von k bestimmt. Benachbarte Schluessel
# unterscheiden sich an genau 2 Stellen.
def nachbar_woerter(k):
  N=[]
  for i in range(len(k)-1):
    for j in range(i+1,len(k)):
      N.append(''.join([k[i] if l==j else k[j] if l==i else k[l] for l in range(len(k))]))
  return N

#############################################################################
# 4-Gramm-Haeufigkeiten
datei=open("german_quadgrams.txt")
a=datei.read()
datei.close()
b=a.split("\n")
b.pop()

from math import log

h4_deutsch={}
for b_i in b:
  c=b_i.split(" ")
  h4_deutsch[c[0]]=log(int(c[1]))

def bewertung(text):
  s=0
  for i in range(len(text)-3):
    z=text[i:i+4]
    if z in h4_deutsch:
      s+=h4_deutsch[z]
  return s
#############################################################################

def suche_lokales_maximum(ctext,sichtbar=0):
  k2=zufallspermutation()
  w2=bewertung(masc_ent(ctext,k2))
  w1=w2-1
  while w2>w1:
    k1,w1=k2,w2
```

```
    nachbarn=nachbar_woerter(k1)
    werte=[bewertung(masc_ent(ctext,k)) for k in nachbarn]
    wert_max=max(werte)
    i=werte.index(wert_max)
    k2,w2=nachbarn[i],wert_max
    if sichtbar==1:
      print(w2,k2,masc_ent(ctext[:30],k2))
  return w1,k1,masc_ent(ctext,k1)

def suche(ctext,runden=100):
  w_max=-1000000
  for i in range(1,runden+1):
    w,k,text=suche_lokales_maximum(ctext)
    if w>w_max:
      w_max,k_max=w,k
      print("i=",i,"-w=",w,"-k=",k,"-",text[:30],"-",w/len(text),sep="")
  return k_max
```