CypherMatrix® Verfahren

(Ernst Erich Schnoor)

Das „CypherMatrix-Verfahren" arbeitet im Kern in drei Stufen:

1. Generator
2. „one-time-chain"
3. „Bit-Konversion"

Als Zeichensatz dienen europäische Unicode-Zeichen im Umfang von 256 Bytes. Sie erhalten den Wert ihrer Position: 0 - 255

ǩOǬDZǾè½ə×ĬtaÕhjêɨVĐ+ȡǽƯ=ǼȖoiãiɣFŎ©ǴSNъÑY@ĭơɓgÔŊǺƐǹYRmdkÖJ5ƖpǢzĺğ
®èeƒűȀŭDzȚxXÉЂü(Θ℗ÝŏǰMP,njÀÂŬƨU|ƷƪọSqȦ>ÙBNJwlăȉǿvÓȁĂ6ƈKÊ€DɣÂ:îÇȈçǯǨ
0ÛÅ¾;ƶηØÜǻòűƢÚCTcƤÈБ-bƆÒLÚʒƇHƐəyΣǼɄʋưKdz<áǣÔùɧâà4ÎɁäÃÄɑ7ɌrɄÞgĐȔ${
ǴţGƎÏǭ%ƵæǟǫȄsE8¿ZöâɯßȒƱlj*3&ǞÁǖƲƠ9ÎǕ)WƬInfƊfƓÊQuƃȡ1Ë.AȌǸåô!ǡÌ¶ƐД
Umfang: 256 Zeichen

Testdatei ist „Ballin.txt". Als Startsequenz werden eingegeben:
„Schwarze Raben auf roten Felsen in der goldenen Abendsonne" (n=58)

In der ersten Runde der Verschlüsselung ergeben sich die folgenden Daten:

1. **Generator**

```
def generator(matrkey,breite,cd):

    # Der Generator erzeugt alle Parameter zur Steuerung des
      gesamten Ablaufs, sowohl bei der Ver-, als auch bei der
      Entschlüsselung.

      Parameter:

        cyphfolge
        alpha
        beta
        gamma
        theta

    return  rundkey,matrkey,alphabet,matrix,hs
```

Der Ablauf zeigt sich im Python 3.7.2 – Code auszugsweise wie folgt:
(ohne definierte Variablen):

```
def generator(matrkey,breite,cd):

    l = len(matrkey)                          #   l = 58, cd = 1
    ck = (l * (l-2)) + cd                     #   ck =  841
```

```
    for z in matrkey:
        n += 1
        s = (ord(z)+1) * (ck +n)
        hp = hp + s                                  #   hp =  22860702

    for z in matrkey:
        n += 1
        ss = ((ord(z)+1) * n * hp) +n+cd+1

        ergebnis = nachsys(77,ss)
        hashfolge = hashfolge + ergebnis

        hk = hk + ss                                 #    hk =  6512968278921

    hs = hp + hk                                     #    hs =  6512991139623
    folgehash = hashfolge[::-1]

    while hashfolge:

         a = hashfolge[i:i+1]
         b = folgehash[i:i+1]
         cyphfolge = cyphfolge + a + b

         if a == "":
            break
         i += 1
```

cyphfolge:
εWsnH80εκI11wkjHEμεjpA1dRκS6ξμvU4B3wP1JnQeTQPRηQiw21xliEκdehnβvd
QjWDwε8UbkEfQ1ΩδKLgFw9CYxq11fBP6xdgBSxrgt1Gz9FmζEO419λ0BβCDuPb
G8qlkmd4NfδλB2s1w1qiQJΩnyKu9αθκrYηFIBzNIκ9γMTarLβPιTnWHpVfRJvvJRfV
pHWnTιPβLraTMγ9κINzBIFηYrκθα9uKynΩJQiq1w1s2BλδfN4dmklq8GbPuDCβB0
λ914OEζmF9zG1tgrxSBgdx6PBf11qxYC9wFgLKδΩ1QfEkbU8εwDWjQdvβnhedκE
ilx12wiQηRPQTeQnJ1Pw3B4Uvμξ6SκRd1ApjεμEHjkw11Iκε08HnsWεP1

```
    alpha = ((hk + hp) % 126) +1                 #    alpha    =  4
    beta = (hk % 69) +1                          #    beta     =  113
    gamma = ((hp +cd) % 72) +1                   #    gamma =  12
    theta = (hk % 32) +1                         #    theta    =  10

    for z in cyphfolge:
        i += 1
        slip = cyphfolge[i:i+3]
        digit = nachdez(78,slip)
        digit = int(digit)
        zahl = (digit % 255)+1

        element = buchst[zahl]
        if element in matrix:
```

```
            element = ""
        else:
            matrix = matrix + element
```

matrix:
WljŏȄəăxƐȀ<ǍdƜƓȍâǞṅȉƂΣu¶g®ƈÎêÖÛǢДBÊƒÝRǬȗNÒtàîȪôѲǫ7ɡʅÄ)ѢȌğæ?
ãbȆƟɭţƮD8ǠGcǘʒȁƆ$ßUǾǡ℅ƶ½OP1Đ5DZĐȕf3aǯçǟȡNJF{dzǗÅKKöƚjmɱȒƱ-
ǨǗǕålǸȊwqƐȅY(Iɩ©fä=QƏÞHÏ,×ƯoⱤTrǢǣǦĭVSƍǾLȈØɿeEMÉÜ6ƇűǽáβÊnjƨǓ*ǺⱣ
%:pǺÔЪâƎƱǼÎǴ&ɄÇɣƵ0zǛDzɥƊǩĬÕh+iǑъÑY@ơɴƸkJÍèŭXü℗ǰÀÂ|
>Ȕǿv€ɣÂ¾;ɳǻCБÚǯyư4ƻÃǭs¿ZƠ9ƮnȡË.A!

```
    alphabet = matrix[alpha:alpha+128]
    rundkey = matrix[beta:beta+84]
    matrkey = matrix[gamma:gamma+72]
```

alphabet:
əăxƐȀ<ǍdƜƓȍâǞṅȉƂΣu¶g®ƈÎêÖÛǢДBÊƒÝRǬȗNÒtàîȪôѲǫ7ɡʅÄ)ѢȌğæ?
ãbȆƟɭţƮD8ǠGcǘʒȁƆ$ßUǾǡ℅ƶ½OP1Đ5DZĐȕf3aǯçǟȡNJF{dzǗÅKKöƚjmɱȒƱ-
ǨǗǕålǸȊwqƐȅY(Iɩ©fä=Q
rundkey:
ǗǕålǸȊwqƐȅY(Iɩ©fä=QƏÞHÏ,×ƯoⱤTrǢǣǦĭVSƍǾLȈØɿeEMÉÜ6ƇűǽáβÊnj
matrkey:
ƜƓȍâǞṅȉƂΣu¶g®ƈÎêÖÛǢДBÊƒÝRǬȗNÒtàîȪôѲǫ7ɡʅÄ)ѢȌğæ?ãbȆƟɭţƮD8ǠG

```
    return rundkey,matrkey,alphabet,matrix,hs
```

Das Ergebnis „matrkey" wird als Startsequenz auf die nächste Runde vorgetragen. Der „rundkey", in gleicher Länge wie der aktuelle „Klartext", wird als „one-time-pad" verarbeitet und das Ergebnis an die nächste Stufe weitergeleitet.

2. „one-time-chain"

```
# In jeder Runde wird aus „Textblock" und „rundkey" - beides in gleicher
  Länge – one-time-pad's erzeugt, die alle in serieller Folge zum
  one-time-chain verknüpft werden. Damit unbrechbar !

def XORgen(fall,eins,zwei,drei,vier):

    def exclusivoder(cypher,parbeta):

    return rohdata                    # Ergebnis der XOR-Verschlüsselung
```

Die Stufe „one-time-chain" muss sowohl die Verschlüsselung, als auch die Entschlüsselung durchführen (fall 1 und fall 2). Die Alternativen „eins, zwei, drei und vier" berücksichtigen den unterschiedlichen Verlauf im Einzelfall.

```
def XORgen(fall,eins,zwei,drei,vier):

    if eins:    #  [b'ALBERT BALLIN 1857 - 1918 Der Sohn eines j\xc3\xbcdis

        for x in eins:
            x = str(x)
            digit = ord(x)
            element = str(digit)
            cypheins.append(element)

    if zwei:
        for x in zwei:
            x = str(x)
            digit = alphabet.find(x)
            element = str(digit)
            cyphzwei.append(element)

    if drei:   #  ƃΘqÍǽʊ¿JhsçȔβâÜʡgdznj{DzAƢbÉ?ъ¶ËƕʅæÑ<Ǡ×ȅƐtãǞÈÅM-
                  ǖƬ:ǟəve¾ÚkƩ
        for z in drei:
            z = str(z)
            digit = alphabet.find(z)
            element = str(digit)
            parbeta.append(element)

    if vier:
        for x in vier:
            digit = alphabet.find(x)
            element = str(digit)
            cyphvier.append(element)

    if fall == 1:
        ergxor = exclusivoder(cypheins,parbeta)
        for x in ergxor:
            x = int(x)
            element = alphabet[x]
            rohdata = rohdata + element

    elif fall == 2:
        ergxor = exclusivoder (cyphvier,parbeta)
        for x in ergxor:
            x = int(x)
            element = chr(x)
            rohdata = rohdata + element

    return rohdata
```

rohdata: ÔṅèOnjƇБBnjÊpДʅƐGÌj-ʅAȈßǼ7î\$ƯÏâfǾÞ×ǡt3lǖv€ʡƌǸÊÌ0HǼôÌ=NƬДΘğ
Umfang: 56 Zeichen

Der nächste Schritt vollzieht die eigentliche „XOR-Verknüpfung“:

```
 cypher  =  [b'ALBERT BALLIN 1857 - 1918 Der Sohn eines j\xc3\xbcdis
 parbeta =  ƃƟqÍǽƲ¿JhsçǛβâÜƻʛdznj{DzAƆ̄bÉ?ъ¶ËƕʅæÑ<Ȧ×ȅƐtãǞÈÅM-

def exclusivoder(cypher,parbeta):

   for x in cypher:
        ax = cypher[m]
        ax = int(ax)
        bx = parbeta[m]
        bx = int(bx)

        xo = ax ^ bx
        ergxor.append(xo)
        m += 1

    return ergxor

XOR --> one-time-pad:  1
resultat:  ÔṅèOnjƇƂBnjÊpДʅƐGÌj-ʅAȉßǢ7ȋ₷ƯÏâfǾÞ×āt3lȕv€ƻđǸÊÌ0HǼôÌ=NTДƟğ
Umfang:  56  Zeichen
```

Als abschließende Stufe wird das Ergebnis der „XOR-Verknüpfung“ von 8-bit Elementen ohne Änderung der „Bits“ in 7-bit Sequenzen umgewandelt.

3. **„Bit-Konversion“**

```
def  bitsdown(original,matrix,alphabet):

     # für Konversion von 8-bit nach 7-bit (bzw. x-bit). Verschlüsselung

def  bitsup(original,matrix,alphabet):

    # für Konversion von 7-bit (bzw x-bit) nach 8-bit. Entschlüselung

    return ergebnis                    # Chiffretext / Klartext
```

Die Umwandlung gestaltet sich wie folgt:

```
def bitsdown(original,matrix,alphabet):

    lang = len(original)

    zugabe = "00000000"
    for x in original[:lang+1]:
        digit = buchst.find(x)
        i += 1
```

```
        xduo = nachsys(2,digit)
        xduo = str(xduo)
        xduo = zugabe + xduo
        l = len(xduo)
        xduo = xduo[l-8:]
        folge1 = folge1 + xduo

    umfang = len(folge1)

    for z in folge1[:umfang]:
        slip = folge1[i:i+7]
        i += 7
        if i > umfang:
            break
        digit = nachdez(2,slip)
        digit = int(digit)
        element = alphabet[digit]
        ergebnis = ergebnis + element

    return ergebnis
```

Chiffretext (8-bit --> 7-bit):
×-gǨKnjğ5ÒÖPƐszÚêbƎHƨBåÜ1ÜgqΘbÆÈìeƨWØūÑUqKǨơÂnjÂъ{JÂ¶?
ãÆĒƀāű1ŭDzç,ƶɳƐ
Umfang: 64 Zeichen

Das Ergebnis ist der Chiffretext (bei Verschlüsselung) bzw. der Klartext (bei Entschlüsselung).

Das Zusammenspiel von Generator / „one-time-chain" und „Bit-Konversion" führt im Ergebnis zu einer mit heutigen Mitteln nicht brechbaren Verschlüsselung.

München, den 25.07.2019
Ernst Erich Schnoor
