Assignment 3

Maxime CHAMBREUIL
maxime.chambreuil@mail.mcgill.ca

Contents

1 Exercises from Stinson's book **1**
1.1 Latin Square Cryptosystem . . . 1
1.2 Ciphertext equally probable . . . 2
1.3 Entropy calculi . . . 2
1.4 Vigenere cipher . . . 2

2 Visual One-Time Pad **3**
2.1 Superposition . . . 3
2.2 Redundancy . . . 3

3 Blum-Blum and Shub generator **3**
3.1 Distinguishing the Blum Blum and Shub generator . . . 3
3.2 Maple . . . 3

1 Exercises from Stinson's book

1.1 Latin Square Cryptosystem

A cryptosystem has a perfect secrecy if

$$\forall p \in P, c \in C, Pr_{P,C}(p/c) = Pr_P(p)$$

$$\forall p \in P, c \in C, Pr_{P,C}(p/c) = \frac{Pr(p).Pr(c/p)}{Pr(c)}$$

Each key is used with equal probability, so knowing j, there is only one key that encrypt j to a L(i,j) among the n keys (Each number appears once in a column). Concerning $Pr(c)$, each L(i,j) appears n times in the square among the n^2 possible cases.

$$Pr(p/c) = \frac{Pr(p).\frac{1}{n}}{\frac{n}{n^2}} = Pr(p)$$

In conclusion, the *Latin Square Cryptosystem* achieves perfect secrecy provided that every key is used with equal probability.

1.2 Ciphertext equally probable

If the cryptosystm has perfect secrecy, then we have :

$$\forall p \in P, c \in C, Pr_{P,C}(p/c) = Pr_P(p)$$

So, we can deduce with the Bayes Theorem that :

$$\forall c \in C, Pr(c) = \frac{Pr(p).Pr(c/p)}{Pr(p/c)} = \frac{Pr(p).Pr(c/p)}{Pr(p)} = Pr(c/p)$$

As $|P| = |C| = |K|$, we know that there is only one key among n that encrypts p to c. So

$$\forall c \in C, Pr(c/p) = \frac{1}{n}$$

We can conclude that every ciphertext is equally probable.

1.3 Entropy calculi

$$\begin{aligned}
H[P] &= -Pr(p=a)lgPr(p=a) - Pr(p=b)lgPr(p=b) - Pr(p=c)lgPr(p=c) \\
&= -\frac{1}{2}lg\frac{1}{2} - \frac{1}{3}lg\frac{1}{3} - \frac{1}{6}lg\frac{1}{6} \\
&= 1,4591 \\
H[K] &= -Pr(k=K_1)lgPr(k=K_1) - Pr(k=K_2)lgPr(k=K_2) - Pr(k=K_3)lgPr(k=K_3) \\
&= -\frac{1}{3}lg\frac{1}{3} - \frac{1}{3}lg\frac{1}{3} - \frac{1}{3}lg\frac{1}{3} \\
&= 1,5850 \\
H[C] &= -Pr(c=1)lgPr(c=1) - Pr(c=2)lgPr(c=2) - Pr(c=3)lgPr(c=3) - Pr(c=4)lgPr(c=4) \\
&= -\frac{2}{9}lg\frac{2}{9} - \frac{2}{9}lg\frac{2}{9} - \frac{1}{3}lg\frac{1}{3} - \frac{2}{9}lg\frac{2}{9} \\
&= 1,9749 \\
H[K/C] &= Pr(c=1)H[K/c=1] + Pr(c=2)H[K/c=2] + Pr(c=3)H[K/c=3] + Pr(c=4)H[K/c=4] \\
&= \frac{2}{9}[-Pr(K=K_1/c=1)lgPr(K=K_1/c=1) - Pr(K=K_2/c=1)lgPr(K=K_2/c=1) \\
&\quad -Pr(K=K_3/c=1)lgPr(K=K_3/c=1)] + \cdots \\
&= \frac{2}{9}[-\frac{1}{2}lg\frac{1}{2} - 0lg0 - \frac{1}{2}lg\frac{1}{2}] + \frac{2}{9}[-\frac{1}{2}lg\frac{1}{2} - \frac{1}{2}lg\frac{1}{2} - 0lg0] + \frac{1}{3}[-\frac{1}{3}lg\frac{1}{3} - \frac{1}{3}lg\frac{1}{3} - \frac{1}{3}lg\frac{1}{3}] + \cdots \\
&= 1,1950 \\
H[P/C] &= Pr(c=1)H[P/c=1] + Pr(c=2)H[P/c=2] + Pr(c=3)H[P/c=3] + Pr(c=4)H[P/c=4] \\
&= \frac{2}{9}[-Pr(P=a/c=1)lgPr(P=a/c=1) - Pr(P=b/c=1)lgPr(P=b/c=1) \\
&\quad -Pr(P=c/c=1)lgPr(P=c/c=1)] + \cdots \\
&= \frac{2}{9}[-\frac{1}{2}lg\frac{1}{2} - 0lg0 - \frac{1}{2}lg\frac{1}{2}] + \frac{2}{9}[-\frac{1}{2}lg\frac{1}{2} - \frac{1}{2}lg\frac{1}{2} - 0lg0] + \frac{1}{3}[-\frac{1}{3}lg\frac{1}{3} - \frac{1}{3}lg\frac{1}{3} - \frac{1}{3}lg\frac{1}{3}] + \cdots \\
&= 1,1950
\end{aligned}$$

1.4 Vigenere cipher

We know that:

$$n \geq \frac{lg|K|}{R_L.lg|P|}$$

In our case:

$$\frac{lg|K|}{R_L.lg|P|} = \frac{lg(26)^m}{R_L.lg(26)^m} = \frac{1}{R_L}$$

So we can conclude that unicity distance for a vigenere cipher is $\frac{1}{R_L}$.

2 Visual One-Time Pad

2.1 Superposition

In a message, let us consider a pixel which can be black or white (1 or 0). We can construct the following truth tables:

M	K	M $\oplus$ K = c
0	0	0
1	0	1
0	1	1
1	1	0

C	K	C $\vee$ K = M'
0	0	0
1	0	1
1	1	1
0	1	1

$\Rightarrow$

M	M'
0	0
1	1
0	1
1	1

M and M' are pretty much the same, instead of having a white background, we have a black one though. Therefore we can distinguish the message M.

2.2 Redundancy

When we consider a letter, we can admit that it is represented inside a rectangle of approximately 20 $\times$ 20 pixels, so one letter of our alphabet is encoded on 400 bits. In fact we only need 5 bits for our 26 letters, so we have 400 - 5 = 395 useless bits. This amount of redundancy in the message allows Oscar to find it and break the cryptosystem.

If there is no redundancy in the plaintext, there is no chance for Oscar to retrieve the message. In our example, a letter would be represented by 5 pixels, so it is not enough to find out which letter is behind those pixels.

3 Blum-Blum and Shub generator

3.1 Distinguishing the Blum Blum and Shub generator

Knowing the algorithm of the generator, I will first take N = 1 and check if the second block of N bits is the square of the first block mod N. If it is not the case, I will increment N, until I find N (or N hit half the length of the string).

Knowing N, I would generate all the keystream and compare it with the output of the Blum Blum and Shub generator to be sure that's the right N. If they match, the string has been generated with the Blum-Blum and Shub Generator. If they do not match, I increment N and I look for another N, until I hit half the length of the original string.

3.2 Maple

```
isBBSG := proc(string)

  local lg,M,N,tmp,Next,generated,s,i;
  lg := length(string):
```

```
  M := 2;

  # Loop on N that match  s = s^2 mod N, but does not generate the right string
  while (M < 1+(lg/2)) do:

    N:=M:
    tmp := convert(string[1..N],decimal,10):
    Next := convert(string[N+1..2*N],decimal,10):

    # Look for an N that match the formulae
    while ((not (tmp^2=Next mod N)) and N < 1+(lg/2) ) do:

      N := N+1:
      tmp := convert(string[1..N],decimal,10):
      Next := convert(string[N+1..2*N],decimal,10):

    end do:

    # Test this N
    if (N < 1+(lg/2)) then:

      s := convert(string[1..N],decimal,10):
      generated := s:

      # Generate a string thanks to the found N
      for i from 1 to lg/N do:
        s := s^2 mod N:
tmp := convert(s,binary):
generated := cat(generated,tmp):
      end do:

      # Compare the generated string to the original
      if (generated[1..lg] = string) then:

        M := 2+(lg/2): # To stop the general loop
print("This string has been generated with the Blum-Blum and Shub Generator."):

      else:

        # To do another step of the general loop
        # from the found N + 1 (Useless to begin at N=0)
        N := N+1:
M := N:
      end if:

    else

      # We arrive to the middle of the original string, so no N was found
      M :=N:
      print("This string has NOT been generated with the
                                Blum-Blum and Shub Generator."):

    end if:
```

```
  end do:
  return N:

end proc:

isBBSG(W0):
  This string has NOT been generated with the Blum-Blum and Shub Generator.

isBBSG(W1):
  This string has NOT been generated with the Blum-Blum and Shub Generator.
```