POLITECNICO DI MILANO

USI/SUPSI

IDSIA





Classical Ciphers in SAGE

Cristina Emma Margherita Rottondi

E-mail: cristinaemma.rottondi@polimi.it; cristina.rottondi@supsi.ch

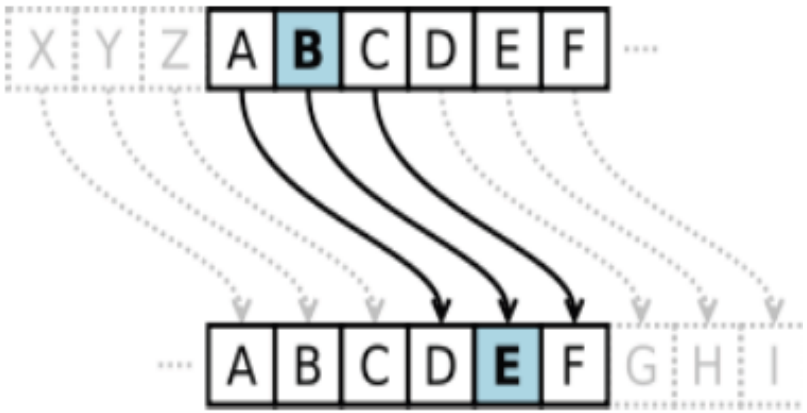
Website: http://people.idsia.ch/~rottondi/

The Shift Cipher – Origins (I)

- The shift cipher is also called the Caesar cipher, named afterJulius Caesar, who, according to Suetonius, used it over 2000 years ago with a shift of three to protect messages of military significance.

"If he had anything confidential to say, he wrote it in cipher, that is, by so changing the order of the letters of the alphabet, that not a word could be made out. If anyone wishes to decipher these, and get at their meaning, he must substitute the fourth letter of the alphabet, namely D, for A, and so with the others."

-Suetonius, Life of Julius Caesar -



The Shift Ciphers – Origins (II)

- The shift cipher was successful for Caesar because most people were illiterate at the time and those few who could read did not usually know the difference between different languages. If they got a message they could not read, they assumed it was in a foreign language they did not know.
- The fact that the "key space" (the set of all possible keys) has only 26 elements makes it a very weak cipher.
- Caesar's nephew Augustus reportedly used a similar cipher, but with a right shift of 1.
- There are also several instances (incredibly enough) of modern uses of this cipher. For example, in April 2006, fugitive Mafia boss Bernardo Provenzano was captured in Sicily partly because some of his messages, written in a variation of the Caesar cipher, were broken. Provenzano's cipher used numbers, so that "A" would be written as "4", "B" as "5", and so on.

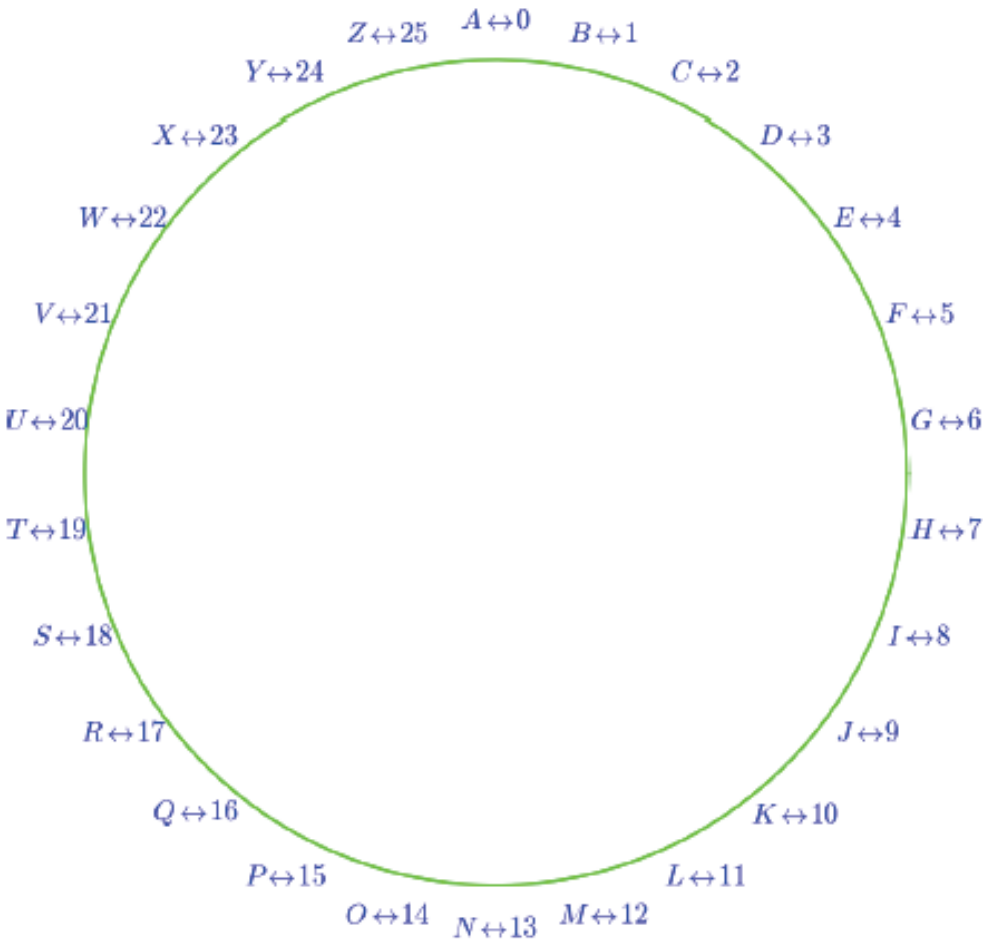
The Shift Cipher - Definition

- Encryption of a letter m by a shift *k* can be described mathematically as:

$$c = Enc_k(m) = m + k \bmod 26$$

- Decryption is performed as follows:

$$m = Dec_k(c) = c - k \bmod 26$$



The Shift Cipher – Encryption Algorithm

- INPUT:
  - k - an integer from 0 to 25 (the secret "key")
  - m - string of upper-case letters (the "plaintext" message)
- OUTPUT:
  - c - string of upper-case letters (the "ciphertext" message)
- STEPS:
  1. Identify the alphabet A, ..., Z with the integers 0, ..., 25.
  2. Compute from the string m a list L1 of corresponding integers.
  3. Compute from the list L1 a new list L2, given by adding k (mod 26) to each element in L1.
  4. Compute from the list L2 a string c of corresponding letters.

The Shift Cipher – Sage Implementation

```
AS = AlphabeticStrings()
CS = ShiftCryptosystem(AS)
s = "Go Navy! Beat Army!"
m = CS.encoding(s) # the plaintext message
c = CS.enciphering(1, m) # the ciphertext with k=1
print m; print c
GONAVYBEATARMY
HPOBWZCFBUBSNZ
```

- Deciphering this message can be accomplished by either shifting right by 25 (enciphering) or to the left by -1 (deciphering):

```
print CS.enciphering(25, c)
print CS.deciphering(1, c)
GONAVYBEATARMY
```

Encrypting Multiple Times

- By encrypting the same message *m* two times with two different keys *k1* and *k2* we obtain:

$$c = Enc_{k2}(Enc_{k1}(m)) = m + k_1 + k_2 \bmod 26$$

- Which is equivlent to encrypting only one time with *k3=k1+k2.*
- Therefore, encrypting multiple times does not bring any advantage
- This is NOT a general rule (some cryptosystems behave differently)

Exercise 1

Crack this ciphertext: "LRZZOACZZQTDZYPESLEXLVPDFDHTDPC“

SOLUTION

```
ct = "LRZZOACZZQTDZYPESLEXLVPDFDHTDPC"
c = CS.encoding(ct)
for i in range(26):
      print i, CS.deciphering(i, c)
0 LRZZOACZZQTDZYPESLEXLVPDFDHTDPC
1 KQYYNZBYYPSCYXODRKDWKUOCECGSCOB
…
9 CIQQFRTQQHKUQPGVJCVOCMGUWUYKUGT
10 BHPPEQSPPGJTPOFUIBUNBLFTVTXJTFS
11 AGOODPROOFISONETHATMAKESUSWISER
12 ZFNNCOQNNEHRNMDSGZSLZJDRTRVHRDQ
…
```

Shift=11

Example - Frequency Analysis (I)


```
raven = 'Once upon a midnight dreary, while I
pondered, weak and weary, Over many a quaint and
curious volume of forgotten lore, While I nodded,
nearly napping, suddenly there came a tapping, As of
some one gently rapping, rapping at my chamber door.
“Tis some visiter,” I muttered, “tapping at my chamber
door -- Only this,and nothing more.“'
m = CS.encoding(raven)
m.frequency_distribution()
```

```
Discrete probability space defined by {A:0.0891472868217055,
   C:0.0193798449612403, B:0.00775193798449612, E:0.108527131782946,
   D:0.0581395348837209, G:0.0348837209302326, F:0.0116279069767442,
   I:0.0736434108527132, H:0.0310077519379845,K:0.00387596899224806,
   M:0.046511627906767, L: 0.0310077519379845, O:0.085271317829457,
   N:0.0930232558139535, Q:0.00387596899224806, P:0.046511627906767,
   S:0.0310077519379845, R: 0.0736434108527132, U:0.027131782945736,
   T:0.0620155038759690, W:0.0155038759689922, V:0.0116279069767442,
   Y:0.0348837209302326}
```

Example - Frequency Analysis (II)

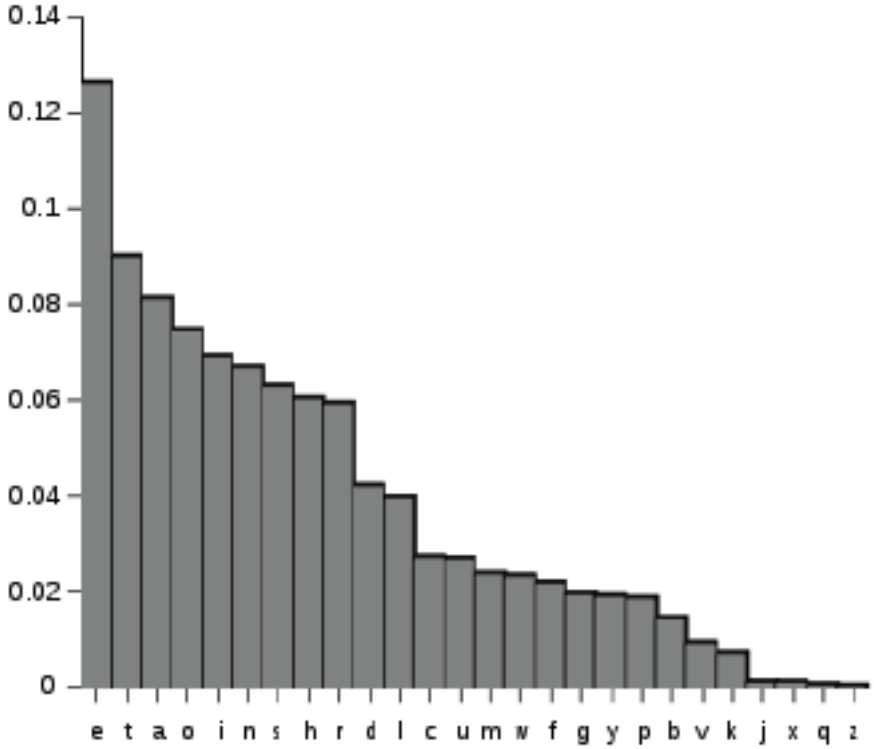
Let us use Sage to sort this of letter frequencies.

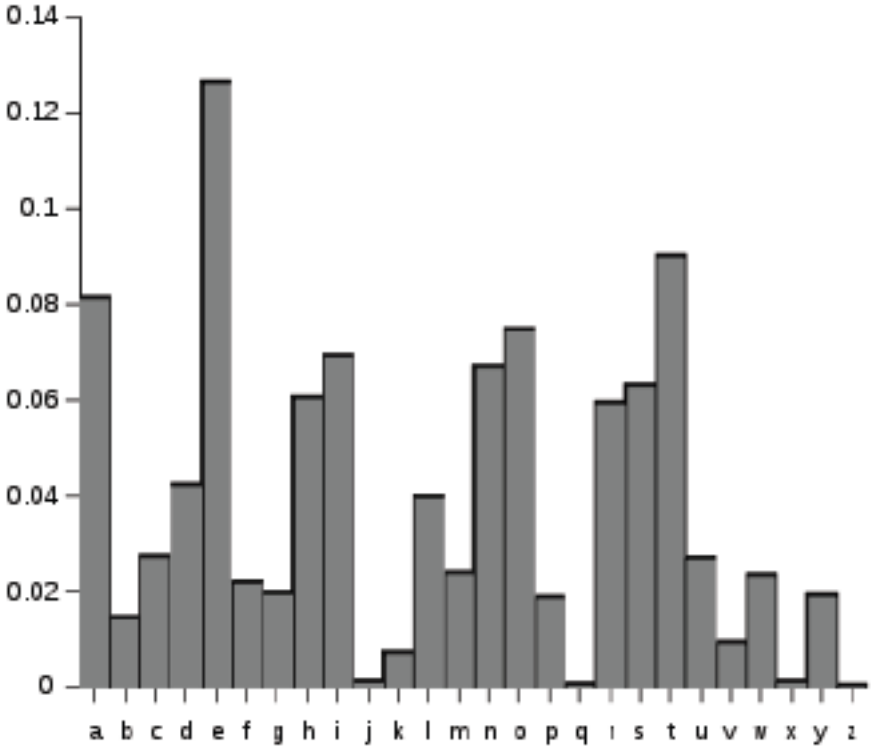
```
fd = m.frequency_distribution()
dict_fd = fd.function()
list_fd = [[dict_fd[x],x] for x in dict_fd.keys()]
list_fd.sort()
list_fd.reverse()
print list_fd
```

```
[[0.108527131782946, E], [0.0930232558139535, N],
[0.0891472868217055, A], [0.0852713178294574, O],
[0.0736434108527132, R], [0.0736434108527132, I],
[0.0620155038759690, T], [0.0581395348837209, D],
[0.0465116279069767, P], [0.0465116279069767, M],
[0.0348837209302326, Y], [0.0348837209302326, G],
[0.0310077519379845, S], [0.0310077519379845, L],
[0.0310077519379845, H], [0.0271317829457364, U],
[0.0193798449612403, C], [0.0155038759689922, W],
[0.0116279069767442, V], [0.0116279069767442, F],
[0.00775193798449612, B], [0.00387596899224806, Q],
[0.00387596899224806, K]]
```

Example – Frequency Analysis (III)

How does this compare with "typical" (whatever that means) English language use?





Not bad "E" is still in first place!

Exercise 2

Decipher this:

"VUJLBWVUHTPKUPNOAKYLHYFDOPSLPWVUKLYLKDLHRHUKD
LHYFVCLYTHUFHXBHPUAHUKJBYPVBZCVSBTLVMMVYNVAALU
SVYLDOPSLPUVKKLKULHYSFUHWWPUNZBKKLUSFAOLYLJHTL
HAHWWPUNHZVMZVTLVULNLUASFYHWWPUNYHWWPUNHATF
JOHTILYKVVYAPZZVTLCPZPALYPTBAALYLKAHWWPUNHATFJO
HTILYKVVYVUSFAOPZHUKUVAOPUNTVY

Use frequency analysis and *not brute force.*

SOLUTION

```
AS = AlphabeticStrings()
A = AS.alphabet()
CS = ShiftCryptosystem(AS)
ct="VUJLBWVUHTPKUPNOAKYLHYFDOPSLPWVUKLYLKDLHRHUKDLHYFV
   CLYTHUFHXBHPUAHUKJBYPVBZCVSBTLVMMVYNVAALUSVYLDOPSLP
   UVKKLKULHYSFUHWWPUNZBKKLUS”
c = CS.encoding(ct)
```

Exercise 2 – Solution (I)


```
fd = c.frequency_distribution()
dict_fd = fd.function()
list_fd = [[dict_fd[x],x] for x in dict_fd.keys()]
list_fd.sort()
list_fd.reverse()
list_fd
```

```
[[0.108527131782946, L], [0.0930232558139535, U],
[0.0891472868217055, H], [0.085271317829457, V],
[0.0736434108527132, Y], [0.0736434108527132, P],
[0.0620155038759690, A], [0.0581395348837209, K],
[0.046511627906977, W], [0.046511627906977, T],
[0.0348837209302326, N], [0.0348837209302326, F],
[0.0310077519379845, Z], [0.0310077519379845, S],
[0.0310077519379845, O], [0.027131782945736, B],
[0.0193798449612403, J], [0.0155038759689922, D],
[0.0116279069767442, M], [0.0116279069767442, C],
[0.00775193798449612, I], [0.00387596899224806, X],
[0.00387596899224806, R]]
```

Maybe E is mapped to L?

Exercise 2 – Solution (II)


```
print A.index("E")
print A.index("L")
4
11
CS.deciphering(11-4, c)
ONCEUPONAMIDNIGHTDREARYWHILEIPONDEREDWEAKANDWEARYOVERM
   ANYAQUAINTANDC\
URIOUSVOLUMEOFFORGOTTENLOREWHILEINODDEDNEARLYNAPPINGSU
   DDENLYTHERECAM\
EATAPPINGASOFSOMEONEGENTLYRAPPINGRAPPINGATMYCHAMBERDOO
   RTISSOMEVISITE\
RIMUTTEREDTAPPINGATMYCHAMBERDOORONLYTHISANDNOTHINGMORE
```

Maybe the shift is 11-4=7?

The Affine Cipher - Definition

- In the affine cipher the letters of an alphabet of size *n* are first mapped to the integers in the range *0,…,n-1*.The encryption function for a single letter is:

$$c = Enc(m) = am + b \bmod n$$

  Where *n* is the modulo and *b* is the magnitude of the shift. The key *k* is the pair *(a,b).* The value of *a* must be chosen in such a way that *a* and *n* are coprime.

- The decryption function is:

$$m = Dec(c) = a^{-1}(c - b) \bmod n$$

  Where a^{-1} is the multiplicative inverse of *a* modulo *n*.

- Of course, with *a*=1, the affine cipher is the same as the shift cipher. However, the key space is larger, making this cipher slightly more secure than the shift cipher.

The Affine Cipher – Encryption Algorithm

INPUT:

a, b - a pair integers, where gcd(*a*, 26) = 1 (the secret "key")

m - string of upper-case letters (the "plaintext" message)

OUTPUT:

c - string of upper-case letters (the "ciphertext" message)

Identify the alphabet A, ..., Z with the integers 0, ..., 25.

STEPS:

1. Compute from the string m a list L1 of corresponding integers.
2. Compute from the list L1 a new list L2, given by replacing *x* by $ax+b \pmod{26}$, for each element *x* in L1.
3. Compute from the list L2 a string *c* of corresponding letters.

The Affine Cipher – SAGE Implementation


```
AS = AlphabeticStrings()
s = "Go Navy! Beat Army!"
ACS = AffineCryptosystem(AS)
m = ACS.encoding(s)
print m
GONAVYBEATARMY
c = ACS.enciphering(17, 3, m)
print c
BHQDWVUTDODGZV
print ACS.deciphering(17, 3, c)
GONAVYBEATARMY
```

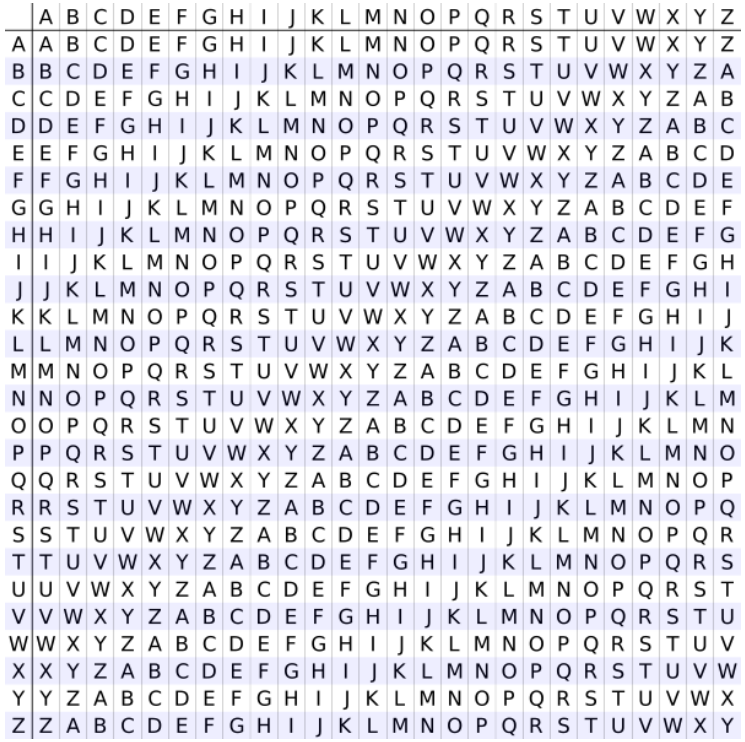
The Vigenère Cipher - Definition

- The Vigenère cipher is a method of encrypting alphabetic text by using a series of different Caesar ciphers based on the letters of a keyword
- Let $M=M_0, M_1, \ldots, M_n$ be the plaintext, $C=C_0, C_1, \ldots, C_n$ be the ciphertext and $K=K_0, K_1, \ldots, K_m$ be the encryption key (which is repeated until it matches the length *n* of the plaintext)
- The encryption works as follows:

$$C_i = Enc_K(M_i) = M_i + K_i \bmod 26$$

- Decryption can be performed as:

$$M_i = Dec_K(C_i) = C_i - K_i \bmod 26$$



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

The Vigenère Cipher – Encryption Algorithm

- INPUT:
  - k – a string of integers from 0 to 25 (the secret "key")
  - m - string of upper-case letters (the "plaintext" message)
- OUTPUT:
  - c - string of upper-case letters (the "ciphertext" message)
- STEPS:
  1. Identify the alphabet A, ..., Z with the integers 0, ..., 25.
  2. Compute from the string m a list L1 of corresponding integers.
  3. Compute from the string k a list K of corresponding integers and replicate it until it reaches the length of L1
  4. Compute from the list L1 a new list L2, given by adding K (mod 26) to each element in L1.
  5. Compute from the list L2 a string c of corresponding letters.

The Vigenère Cipher - Implementation


```
AS = AlphabeticStrings()
VC = VigenereCryptosystem(AS,14)
VC
Vigenere cryptosystem on Free alphabetic string monoid
   on A-Z of period 14

K = AS('ABCDEFGHIJKLMN')
m=VC.encoding("THECATINTHEHAT")
c=VC.enciphering(K,m)
print c
TIGFEYOUBQOSMG
VC.deciphering(K,c)
THECATINTHEHAT
```

Key length

Index of Coincidence Method

- Used to identify the key-length in the Vigenère Cryptosystem
- Basic principle:
  - if the key-length is t, then the ciphertext characters c_1, c_{1+t}, c_{1+2t},… are encrypted using the same shift.
  - Therefore, the frequencies of such characters are expected to be identical to the frequencies of standard English text, except in some shifted order.
  - Conversely, if we try a key-length $\tau \neq t$, we espect that all characters will occurr with roughly equal probability in the sequence c_1, $c_{1+\tau}$, $c_{1+2\tau}$
- Once t is known, each element i ($i=1, \ldots, t$) of the key can be obtained with the same methods used to break a shift cryptosystem, applied on the $i+nt$-th ($n=0, \ldots, \lceil(\text{length}(c)-i)/t\rceil-1$) elements of the cihpertext c

Example 3 – Index of Coincidence Method (I)

Decrypt the encrypted text in the attached file named “the black cat_encrypted.txt” using frequency analysis.

- Load the encrypted text in the variable `ct`
- To find the length of the keyword which was used, the first step is to write a program to apply frequency analysis with different key lengths

```
def cshift(str,n):
    for i in range(1,10):
     trial=ct[0::i]
     p=AS.encoding(trial)
     fd = p.frequency_distribution()
     dict_fd = fd.function()
     list_fd = [[dict_fd[x],x] for x in dict_fd.keys()]
     list_fd.sort()
     list_fd.reverse()
     print list_fd
```

Key length ranges from 1 to 10

Example 3 – Index of Coincidence Method (II)

- Analyze the ouptut for key length 1:

```
[[0.065926317644985 2, S], [0.065573770491803 4, V],
  [0.063811034725894 8,W], [0.057112638815441 8, F],
  [0.056936365238850 9, L],[0.053058346553851 8, J],
  [0.050590516481579 6, K], [0.049180327868852 7,Z],
  [0.043363299841354 0, A], [0.042834479111581 3, H],
[0.041424290498854 4, Y], [0.034549621011810 5, G],
  [0.033844526705447 0,T], [0.031905517362947 4, O],
  [0.031200423056583 9, R],[0.029261413714084 4, M],
  [0.028380045831130 0, U], [0.028203772254539 1,D],
  [0.028027498677948 3, I], [0.027851225101357 4, C],
[0.025559668605676 1, N], [0.023973206416358 3, B],
  [0.023796932839767 4,P], [0.022563017803631 3, E],
  [0.021505376344086 1, Q],[0.019566367001586 5, X]]
```

Almost uniform distribution, the key length is most probably not 1!

Example 3 – Index of Coincidence Method (III)

- Analyze the ouptut for key length 5:

```
[[0.119823788546255, J], [0.080176211453744, Y],
  [0.075770925110132 1,F], [0.074008810572687 1, S],
  [0.074008810572687 1, N],[0.064317180616740 0, W],
  [0.059030837004405 2, T], [0.055506607929515 4,M],
  [0.054625550660792 9, X], [0.044052863436123 3, R],
[0.044052863436123 3, Q], [0.041409691629955 9, I],
  [0.037004405286343 6,Z], [0.031718061674008 8, K],
  [0.029074889867841 4, H],[0.020264317180616 7, D],
  [0.019383259911894 3, B], [0.018502202643171 8,U],
  [0.015859030837004 4, L], [0.015859030837004 4, G],
[0.014977973568281 9, A], [0.004405286343612 34, P],
  [0.001762114537444 93,V], [0.001762114537444 93, O],
  [0.001762114537444 93, E],[0.000881057268722 467, C]]
```

Very high variability in the percentage of occurrences, this must be the right key length!

Example 3 – Index of Coincidence Method (IV)

- We perform a frequency analysis attack on subsets of the ciphertext obtained by k-decimation on a block of length 5. Let's start with k=0

```
prova=ct[0::5]
p=AS.encoding(prova)
sorted(p.character_count().items())
[(A, 17), (B, 22), (C, 1), (D, 23), (E, 2), (F, 86),
   (G, 18), (H, 33),(I, 47), (J, 136), (K, 36), (L,
   18), (M, 63), (N, 84), (O, 2), (P, 5),(Q, 50), (R,
   50), (S, 84), (T, 67), (U, 21), (V, 2), (W, 73),
   (X, 62),(Y, 91), (Z, 42)]
```

- Repeating the decimation procedure starting from char k=2,…,5 we obtain the remaining letters of the keyword, which is "FHRSO"

Probably J corresponds to E, which means that the first letter of the keyword is F

Homework – Exercise 1

- The ciphertext “KDDKMU” has been encrypted with a Ceasar cipher. Find the plaintext using brute force attack.

SOLUTION:

- Use the SAGE code at slide 6 to obtain all the possible plaintexts.
- The only meaningful word is “ATTACK”, which is obtained with a shift of 17.

Homework – Exercise 2

- Encrypt the plaintext “CLEOPATRA” using an affine cipher with encryption function 7x+8.

SOLUTION:

- Use the code presented at slide 16 to obtain the ciphertext “WHKCJILXI”

Homework – Exercise 3

- The ciphertext “MZDVEZC” has been obtained using an affine cipher with encryption function y=5x+12. Find the plaintext.

SOLUTION:

- We have to invert the encryption function ad compute $x=5^{-1}(y-12)$

```
inv=5^(-1) %26
21
21*12 %26
8
```

- Therefore, the decryption function is x=21y+8
- Use the code presented at slide 16 to obtain the plaintext “ANTHONY”