Classical Ciphers

Cryptography

School of Engineering and Technology
CQUniversity Australia

Prepared by Steven Gordon on 04 Jan 2022,
classical.tex, r1964

Contents

Caesar Cipher (algorithm)

To encrypt with a key `k`, shift each letter of the plaintext `k` positions to the right in the alphabet, wrapping back to the start of the alphabet if necessary. To decrypt, shift each letter of the ciphertext `k` positions to the left (wrapping if necessary).

In the examples we will assume the Caesar cipher (and most other classical ciphers) operate on case-insenstive English plaintext. That is, the character set is a through to z. However it can also be applied to any language or character set, so long as the character set is agreed upon by the users.

Caesar Cipher Encryption (exercise)

Using the Caesar cipher, encrypt plaintext `hello` with key 3.

How many keys are possible in the Caesar cipher? (question)

If the Caesar cipher is operating on the characters a–z, then how many possible keys are there? Is a key of 0 possible? Is it a good choice? What about a key of 26?

Caesar Cipher Decryption (exercise)

You have received the ciphertext `TBBQOLR`. You know the Caesar cipher was used with key `n`. Find the plaintext.

Caesar Cipher, formal (algorithm)

$$C = E(K, P) = (P + K) \bmod 26 \quad (1)$$

$$P = D(K, C) = (C - K) \bmod 26 \quad (2)$$

In the equations, P is the numerical value of a plaintext letter. Letters are numbered in alphabetical order starting at 0. That is, a=0, b=1, . . . , z=25. Similarly, K and C are the numerical values of the key and ciphertext letter, respectively. Shifting to the right in encryption is addition, while shifting to the left in decryption is subtraction. To cater for the wrap around (e.g. when the letter z is reacher), the last step is to mod by the total number of characters in the alphabet.

Caesar Cipher, formal (exercise)

Consider the following mapping.

a	b	c	d	e	f	g	h	i	j	k	l	m
0	1	2	3	4	5	6	7	8	9	10	11	12
n	o	p	q	r	s	t	u	v	w	x	y	z
13	14	15	16	17	18	19	20	21	22	23	24	25

Use the the formal (mathematical) algorithm for Caesar cipher to decrypt SDV with key p.

Caesar Encrypt and Decrypt (python)

```
1  >>> pycipher.Caesar(3).encipher("hello")
2  'KHOOR'
3  >>> pycipher.Caesar(3).decipher("khoor")
4  'HELLO'
```


Note that the pycipher package needs to be installed and imported first (see Section **??**).

Brute Force Attack (definition)

Try all combinations (of keys) until the correct plaintext/key is found.

Caesar Brute Force (exercise)

The ciphertext `FRUURJVBCANNC` was obtained using the Caesar cipher. Find the plaintext using a brute force attack.

Caesar Brute Force (python)

```
1  for k in range(0,26):
2  pycipher.Caesar(k).decipher("FRUURJVBCANNC")
```


The `range` function in Python produces values inclusive of the lower limit and exclusive of the upper limit. That is, from 0 to 25.

Caesar Brute Force Results (text)

```
 0: FRUURJVBCANNC 13: SEHHEWIOPNAAP
 1: EQTTQIUABZMMB 14: RDGGDVHNOMZZO
 2: DPSSPHTZAYLLA 15: QCFFCUGMNLYYN
 3: CORROGSYZXKKZ 16: PBEEBTFLMKXXM
 4: BNQQNFRXYWJJY 17: OADDASEKLJWWL
 5: AMPPMEQWXVIIX 18: NZCCZRDJKIVVK
 6: ZLOOLDPVWUHHW 19: MYBBYQCIJHUUJ
 7: YKNNKCOUVTGGV 20: LXAAXPBHIGTTI
 8: XJMMJBNTUSFFU 21: KWZZWOAGHFSSH
 9: WILLIAMSTREET 22: JVYYVNZFGERRG
10: VHKKHZLRSQDDS 23: IUXXUMYEFDQQF
11: UGJJGYKQRPCCR 24: HTWWTLXDECPPE
12: TFIIFXJPQOBBQ 25: GSVVSKWCDBOOD
```


The results of the brute force are formatted to show the key (it is slightly different from the Python code output).

How many attempts for Caesar brute force? (question)

What is the worst, best and average case of number of attempts to brute force ciphertext obtained using the Caesar cipher?

There are 26 letters in the English alphabet. The key can therefore be one of 26 values, 0 through to 25. The key of 26 is equivalent to a key of 0, since it will encrypt to the same ciphertext. The same applies for all values greater than 25. While a key of 0 is not very smart, let's assume it is a valid key.

The best case for the attacker is that the first key they try is the correct key (i.e. 1 attempt). The worst case is the attacker must try all the wrong keys until they finally try the correct key (i.e. 26 attempts). Assuming the encrypter chose the key randomly, there is equal probability that the attacker will find the correct key in 1 attempt (1/26), as in 2 attempts (1/26), as in 3 attempts (1/26), and as in 26 attempts (1/26). The average number of attempts can be calculated as $(26+1)/2 = 13.5$.

Recognisable Plaintext upon Decryption (assumption)

The decrypter will be able to recognise that the plaintext is correct (and therefore the key is correct). Decrypting ciphertext using the incorrect key will *not* produce the original plaintext. The decrypter will be able to recognise that the key is wrong, i.e. the decryption will produce unrecognisable output.

Is plaintext always recognisable? (question)

Caesar cipher is using recognisably correct plaintext, i.e. English words. But is the correct plaintext always recognisable? What if the plaintext was a different language? Or compressed? Or it was an image or video? Or binary file, e.g. .exe? Or a set of characters chosen randomly, e.g. a key or password?

The correct plaintext is recognisable if it contains some structure. That is, it does not appear random. It is common in practice to add structure to the plaintext, making it relatively easy to recognise the correct plaintext. For example, network packets have headers/trailers or error detecting codes. Later we will see cryptographic mechanisms that can be used to ensure that the correct plaintext will be recognised. For now, let's assume it can be.

How to improve upon the Caesar cipher?

1. Increase the key space so brute force is harder
2. Change the plaintext (e.g. compress it) so harder to recognise structure

Contents

Permutation (definition)

A permutation of a finite set of elements is an ordered sequence of all the elements of S, with each element appearing exactly once. In general, there are $n!$ permutations of a set with n elements.

The concept of permutation is used throughput cryptography, and shortly we will see in a monoalphabetic (substitution) cipher.

Permutation (example)

Consider the set $S = \{a, b, c\}$. There are six permutations of S:

`abc, acb, bac, bca, cab, cba`

This set has 3 elements. There are $3! = 3 \times 2 \times 1 = 6$ permutations.

Monoalphabetic (Substitution) Cipher (definition)

Given the set of possible plaintext letters (e.g. English alphabetc, a–z), a single permutation is chosen and used to determine the corresponding ciphertext letter.

This is a monoalphabetic cipher because only a single cipher alphabet is used per message.

Monoalphabetic (Substitution) Cipher (example)

In advance, the sender and receiver agree upon a permutation to use, e.g.:

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

To encrypt the plaintext `hello`, the agreed upon permutation (or mapping) is used to produce the ciphertext `ESCCF`.

Decrypt Monoalphabetic Cipher (exercise)

Decrypt the ciphertext QSWBSR using the permutation chosen in the previous example.

How many keys in English monoalphabetic cipher? (question)

How many possible keys are there for a monoalphabetic cipher that uses the English lowercase letters? What is the length of an actual key?

Consider the number of permutations possible. The example used a single permutation chosen by the two parties.

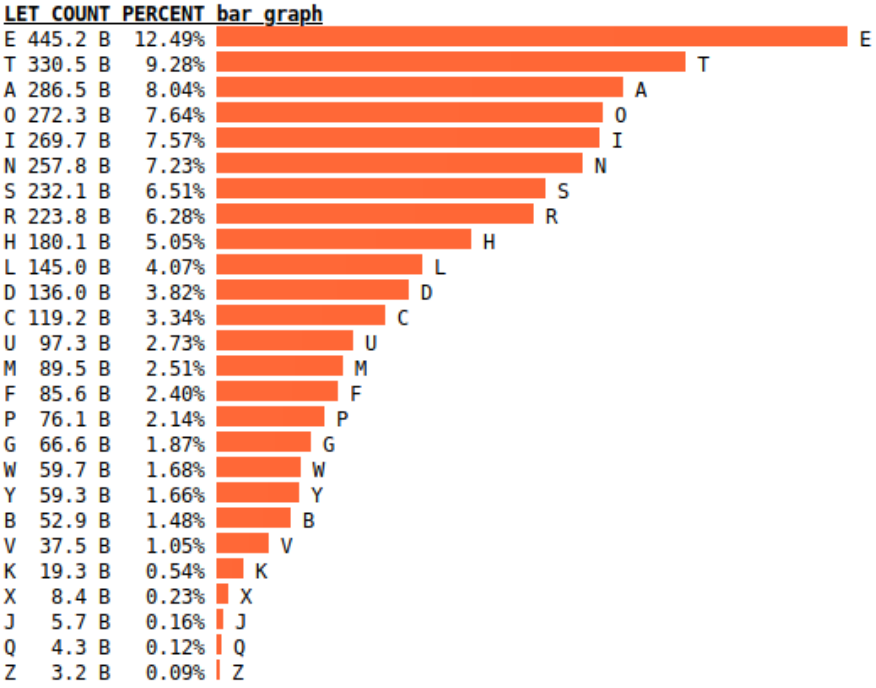
Brute Force on Monoalphabetic Cipher (exercise)

You have intercepted a ciphertext message that was obtained with an English monoalphabetic cipher. You have a Python function called:
`mono_decrypt_and_check(ciphertext,key)`
that decrypts the ciphertext with a key, and returns the plaintext if it is correct, otherwise returns false. You have tested the Python function in a while loop and the computer can apply the function at a rate of 1,000,000,000 times per second. Find the average time to perform a brute force on the ciphertext.

Frequency Analysis Attack (definition)

Find (portions of the) key and/or plaintext by using insights gained from comparing the actual frequency of letters in the ciphertext with the expected frequency of letters in the plaintext. Can be expanded to analyse sets of letters, e.g. digrams, trigrams, n-grams, words.

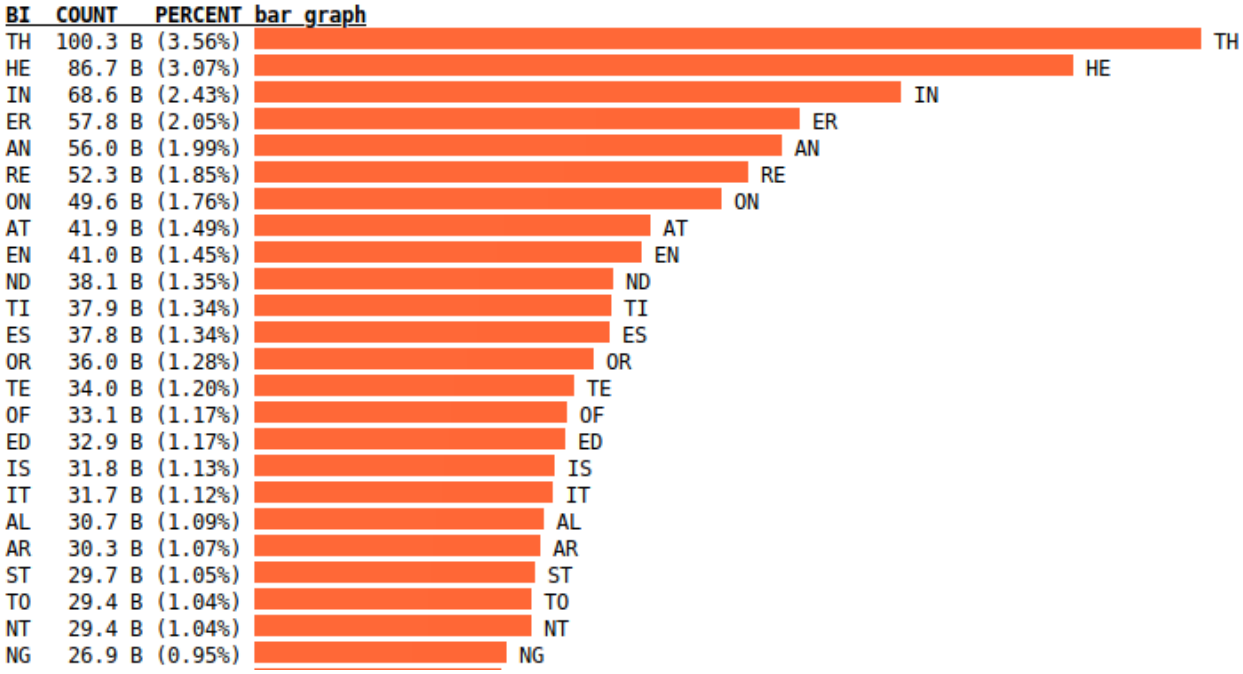
Relative Frequency of Letters by Norvig



Credit: *Letter Counts* by Peter Norvig

The letter frequencies of the figure above are based on Peter Norvig's analysis of Google Books N-Gram Dataset. Norvig is Director of Research at Google. His website has more details on the analysis.

Relative Frequency of Digrams by Norvig



Credit: *Two-Letter Sequence (Bigram) Counts* by Peter Norvig

Relative Frequency of N-Grams by Norvig

1	2grams	3grams	4-grams	5-grams	6-grams	7-grams	8-grams	9-grams
e	th	the	tion	ation	ations	present	differen	different
t	he	and	atio	tions	ration	ational	national	governmen
a	in	ing	that	which	tional	through	consider	overnment
o	er	ion	ther	ction	nation	between	position	formation
i	an	tio	with	other	ection	ication	ifferent	character
n	re	ent	ment	their	cation	differe	governme	velopment
s	on	ati	ions	there	lation	ifferen	vernment	developme
r	at	for	this	ition	though	general	overnmen	evelopmen
h	en	her	here	ement	presen	because	interest	condition
l	nd	ter	from	inter	tation	develop	importan	important
d	ti	hat	ould	ional	should	america	ormation	articular
c	es	tha	ting	ratio	resent	however	formatio	particula
u	or	ere	hich	would	genera	eration	relation	represent
m	te	ate	whic	tiona	dition	nationa	question	individua
f	of	his	ctio	these	ationa	conside	american	ndividual
p	ed	con	ence	state	produc	onsider	characte	relations
g	is	res	have	natio	throug	ference	haracter	political
w	it	ver	othe	thing	hrough	positio	articula	informati
y	al	all	ight	under	etween	osition	possible	nformatio
b	ar	ons	sion	ssion	betwee	ization	children	universit
v	st	nce	ever	ectio	differ	fferent	elopment	following
k	to	men	ical	catio	icatio	without	velopmen	experienc
x	nt	ith	they	latio	people	ernment	developm	stitution
j	ng	ted	inte	about	iffere	vernmen	evelopme	xperience
q	se	ers	ough	count	fferen	overnme	conditio	education
z	ha	pro	ance	ments	struct	governm	ondition	roduction

Credit: *N-Letter Sequences (N-grams)*" by Peter Norvig

Break a Monoalphabetic Cipher (exercise)

Ciphertext:

```
ziolegxkltqodlzgofzkgrxetngxzgzithkofeohs
tlqfrzteifojxtlgyltexkofuegdhxztklqfregd
hxztkftzvgkalvoziygexlgfofztkftzltexkoznz
itegxkltoltyytezoctsnlhsozofzgzvghqkzlyo
klzofzkgrxeofuzitzitgkngyeknhzgukqhinofes
xrofuigvdqfnesqlloeqsqfrhghxsqkqsugkozid
lvgkaturtlklqrouozqsloufqzxktlqfrltegfrhk
gcorofurtzqoslgyktqsofztkftzltexkoznhkgz
gegslqsugkozidlqfrziktqzltuohltecokxltlyo
ktvqsslitfetngxvossstqkfwgzizitgktzoeqsq
lhtezlgyegdhxztkqfrftzvgkaltexkoznqlvtssq
ligvziqzzitgknolqhhsotrofzitofztkftzziol
afgvstrutvossitshngxofrtloufofuqfrrtctsgh
ofultexktqhhsoeqzogflqfrftzvgkahkgzgegsl
qlvtssqlwxosrofultexktftzvgkal
```

Contents

Playfair Matrix Construction (algorithm)

Write the letters of keyword `k` row-by-row in a 5-by-5 matrix. Do not include duplicate letters. Fill the remainder of the matrix with the alphabet. Treat the letters *i* and *j* as the same (that is, they are combined in the same cell of the matrix).

Playfair Matrix Construction (exercise)

Construct the Playfair matrix using keyword `australia`.

Playfair Encryption (algorithm)

Split the plaintext into pairs of letters. If a pair has identical letters, then insert a special letter x in between. If the resulting set of letters is odd, then pad with a special letter x.

Locate the plaintext pair in the Playfair matrix. If the pair is on the same column, then shift each letter down one cell to obtain the resulting ciphertext pair. Wrap when necessary. If the plaintext pair is on the same row, then shift to the right one cell. Otherwise, the first ciphertext letter is that on the same row as the first plaintext letter and same column as the second plaintext letter, and the second ciphertext letter is that on the same row as the second plaintext letter and same column as the first plaintext letter.

Repeat for all plaintext pairs.

Playfair decryption uses the same matrix and reverses the rules. That is, move up (instead of down) if on the same column, move left (instead of right) if on the same row. Finally, the padded special letters need to be removed. This can be done based upon knowledge of the langauge. For example, if the intermediate plaintext from decryption is `helxlo`, then as that word doesn't exist, the x is removed to produce `hello`.

Playfair Encryption (exercise)

Find the ciphertext if the Playfair cipher is used with keyword `australia` and plaintext `hello`.

Does Playfair cipher always map a letter to the same ciphertext letter? (question)

Using the Playfair cipher with keyword `australia`, encrypt the plaintext `hellolove`.

With the Playfair cipher, if a letter occurs multiple times in the plaintext, will that letter always encrypt to the same ciphertext letter?

If a pair of letters occurs multiple times, will that pair always encrypt to the same ciphertext pair?

Is the Playfair cipher subject to frequency analysis attacks?

Contents

Polyalphabetic (Substitution) Cipher (definition)

Use a different monoalphabetic substitution as proceeding through the plaintext. A key determines which monoalphabetic substitution is used for each transformation.

For example, when encrypting a set of plaintext letters with a polyalphabetic cipher, a monoalpabetic cipher with a particular key is used to encrypt the first letter, and then the same monoalphabetic cipher is used but with a different key to encrypt the second letter. They key used for the monoalphabetic cipher is determined by the key (or keyword) for the polyalphabetic cipher.

Examples of Polyalphabetic Ciphers

- Vigenère Cipher: uses Caesar cipher, but Caesar key changes each letter based on keyword
- Vernam Cipher: binary version of Vigenère, using XOR
- One Time Pad: same as Vigenère/Vernam, but random key as long as plaintext

Selected polyalphabetic ciphers are explained in depth in the following sections.

Contents

Vigenère Cipher (algorithm)

For each letter of plaintext, a Caesar cipher is used. The key for the Caesar cipher is taken from the Vigenère key(word), progressing for each letter and wrapping back to the first letter when necessary. Formally, encryption using a keyword of length m is:

$$c_i = (p_i + k_{i \bmod m}) \bmod 26$$

where p_i is letter i (starting at 0) of plaintext P, and so on.

Simply, Vigenère cipher is just the Caesar cipher, but changing the Caesar key for each letter encrypted/decrypted. The Caesar key is taken from the Vigenère key. The Vigenère key is not a single value/letter, but a set of values/letters, and hence referred to as a keyword. Encrypting the first letter of plaintext uses the first key from the keyword. Encrypting the second letter of plaintext uses the second key from the keyword. And so on. As the keyword (for convenience) is usually shorter than the plaintext, once the end of the keyword is reached, we return to the first letter, i.e. wrap around.

In the formal equation for encryption, i represents letter i (starting at 0) of the plaintext. For example, if the keyword is 6 letters, when encrypting letter 8 of the plaintext (that is the 9th), then k_2 is used, i.e. the 3rd letter from the keyword.

Vigenère Cipher Encryption (example)

Using the Vigenère cipher to encrypt the plaintext carparkbehindsupermarket with the keyword sydney produces the ciphertext UYUCEPCZHUMLVQXCIPEYUXIR. The keyword would be repeated when Caesar is applied:

```
P: carparkbehindsupermarket
K: sydneysydneysydneysydney
C: UYUCEPCZHUMLVQXCIPEYUXIR
```


Note that the first a in the plaintext transforms to Y, while the second a transforms to E. With polyalphabetic ciphers, the same plaintext letters do not necessarily always transform to the same ciphertext letters. Although they may: look at the third a.

Vigenère Cipher Encryption (exercise)

Use Python (or other software tools) to encrypt the plaintext `centralqueensland` with the following keys with the Vigenère cipher, and investigate any possible patterns in the ciphertext: `cat`, `dog`, `a`, `giraffe`.

Weakness of Vigenère Cipher

- Determine the length of the keyword m
  - Repeated n-grams in the ciphertext may indicate repeated n-grams in the plaintext
  - Separation between repeated n-grams indicates possible keyword length m
  - If plaintext is long enough, multiple repetitions make it easier to find m
- Treat the ciphertext as that from m different monoalphabetic ciphers
  - E.g. Caesar cipher with m different keys
  - Break the monoalphabetic ciphers with frequency analysis
- With long plaintext, and repeating keyword, Vigenère can be broken

The following shows an example of breaking the Vigenère cipher, although it is not necessary to be able to do this yourself manually.

Breaking Vigenère Cipher (example)

Ciphertext ZICVTWQNGRZGVTWAVZHCQYGLMGJ has repetition of VTW. That suggests repetition in the plaintext at the same position, which would be true if the keyword repeated at the same position.

012345678901234567890123456
ZICVTWQNGRZGVTWAVZHCQYGLMGJ

That is, it is possible the key letter at position 3 is the repated at position 12. That in turn suggest a keyword length of 9 or 3.

```
ciphertext ZICVTWQNGRZGVTWAVZHCQYGLMGJ
length=3:  012012012012012012012012012
length=9:  012345678012345678012345678
```

An attacker would try both keyword lengths. With a keyword length of 9, the attacker then performs Caesar cipher frequency analysis on every 9th letter. Eventually they find plaintext is wearediscoveredsaveyourself and keyword is deceptive.

This attack may require some trial-and-error, and will be more likely to be successful when the plaintext is very long. See the Stallings textbook, from which the example is taken, for further explanation.

Contents

Vernam Cipher (algorithm)

Encryption is performed as:

$$c_i = p_i \oplus k_i$$

decryption is performed as:

$$p_i = c_i \oplus k_i$$

where p_i is the ith bit of plaintext, and so on. The key is repeated where necessary.

The Vernam cipher is essentially a binary form of the Vigenère cipher. The mathematical form of Vigenère encryption adds the plaintext and key and mods by 26 (where there are 26 possible charactersd). In binary, there are 2 possible characters, so the equivalnet is to add the plaintext and key and mod by 2. This identical to the XOR operation.

XOR (python)

```
1  >>> def xor(x, y):
2  ...  return '{1:0{0}b}'.format(len(x), int(x, 2) ^ int(y, 2))
3  ...
```


The Python code defines a function called `xor` that takes two strings representing bits, and returns a string represent the XOR of those bits. The actual XOR is performed on integers using the Python hat ôperator. The rest is formatting as strings.

Vernam Cipher Encryption (exercise)

Using the Vernam cipher, encrypt the plaintext 011101010101000011011001 with the key 01011.

Vernam Cipher Encryption (python)

```
1  >>> xor('011101010101000011011001','010110101101011010110101')
2  '001011111000011001101100'
```

Contents

One-Time Pad (algorithm)

Use polyalphabetic cipher (such as Vigenère or Vernam) but where the key must be: random, the same length as the plaintext, and not used multiple times.

Essentially, the Vigenère or Vernam become a OTP if the keys are chosen appropriately.

Properties of OTP

- Encrypting plaintext with random key means output ciphertext will be random
  - E.g. XOR plaintext with a random key produces random sequence of bits in ciphertext
- Random ciphertext contains no information about the structure of plaintext
  - Attacker cannot analyse ciphertext to determine plaintext
- Brute force attack on key is ineffective
  - Multiple different keys will produce recognisable plaintext
  - Attacker has no way to determine which of the plaintexts are correct
- OTP is only known unbreakable (unconditionally secure) cipher

Attacking OTP (example)

Consider a variant of Vigenère cipher that has 27 characters (including a space). An attacker has obtained the ciphertext:

`ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS`

Attacker tries all possible keys. Two examples:

```
k1:  pxlmvmsydofuyrvzwc tnlebnecvgdupahfzzlmnyih
p1:  mr mustard with the candlestick in the hall
k2:  pftgpmiydgaxgoufhkllmhsqdqogtewbqfgyovuhwt
p2:  miss scarlet with the knife in the library
```

There are many other legible plaintexts obtained with other keys. No way for attacker to know the correct plaintext

The example shows that even a brute force attack on a OTP is unsuccessful. Even if the attacker could try all possible keys—the plaintext is 43 characters long and so there are $27^{43} \approx 10^{61}$ keys—they would find many possible plaintext values that make sense. The example shows two such plaintext values that the attacker obtained. Which one is the correct plaintext? They both make sense (in English). The attacker has no way of knowing. In general, there will be many plaintext values that make sense from a brute force attack, and the attacker has no way of knowing which is the correct (original) plaintext. Therefore a brute force attack on a OTP is ineffective.

Summary of OTP

- Only known unbreakable (unconditionally secure) cipher
  - Ciphertext has no statistical relationship with plaintext
  - Given two potential plaintext messages, attacker cannot identify the correct message
- But two significant practical limitations:
  1. Difficult to create large number of random keys
  2. Distributing unique long random keys is difficult
- Limited practical use

The practical limittions are significant. The requirement that the key must be as long as the plaintext, random and never repeated (if it is repeated then the same problems arise as in the original Vernam cipher) means large random values must be created. But creating a large amount of random data is actually difficult. Imagine you wanted to use a OTP for encrypting large data transfers (multiple gigabytes) across a network. Multiple gigabytes of random data must be generated for the key, which is time consuming (seconds to hours) for some computers. Also, the key must be exchanging, usually over a network, with the other party in advance. So to encrypt a 1GB file to need a 1GB random key. Both the key and file must be sent across the network, i.e. a total of 2GB. This is very inefficient use of the network: a maximum of 50% efficiency.

Later we will see real ciphers that work with a relatively small, fixed length key (e.g. 128 bits) and provide sufficient security.

Contents

Transposition vs Substitution

- Substitution: replace one (or more) character in plaintext with another from the entire possible character set
- Transposition: re-arrange the characters in the plaintext
  - The set of characters in the ciphertext is the same as in the plaintext
  - Problem: the plaintext frequency statistics are also in the ciphertext
- On their own, transposition techniques are easy to break
- Combining transposition with substitution makes ciphers stronger, and building block of modern ciphers

Rail Fence Cipher Encryption (definition)

Select a depth as a key. Write the plaintext in diagonals in a zig-zag manner to the selected depth. Read row-by-row to obtain the ciphertext.

The decryption process can easily be derived from the encryption algorithm.

Rail Fence Encryption (exercise)

Consider the plaintext `securityandcryptography` with key 4. Using the rail fence cipher, find the ciphertext.

Rows Columns Cipher Encryption (definition)

Select a number of columns m and permutate the integers from 1 to m to be the key. Write the plaintext row-by-row over m columns. Read column-by-column, in order of the columns determined by the key, to obtain the ciphertext.

Be careful with the decryption process; it is often confusing. Of course it must be the process such that the original plaintext is produced.

Rows Columns Encryption (exercise)

Consider the plaintext `securityandcryptography` with key `315624`. Using the rows columns cipher, find the ciphertext.

Rows Columns Multiple Encryption (example)

Assume the ciphertext from the previous example has been encrypted again with the same key. The resulting ciphertext is YYCPRRCTEOIPDRAHYSGUATXH. Now let's view how the cipher has "mixed up" the letters of the plaintext. If the plaintext letters are numbered by position from 01 to 24, their order (split across two rows) is:

01 02 03 04 05 06 07 08 09 10 11 12
13 14 15 16 17 18 19 20 21 22 23 24

After first encryption the order becomes:

02 08 14 20 05 11 17 23 01 07 13 19
06 12 18 24 03 09 15 21 04 10 16 22

After the second encryption the order comes:

08 23 12 21 05 13 03 16 02 17 06 15
11 19 09 20 14 01 18 04 20 07 24 10

Are there any obviously obversvable patterns?

After the first encryption, the numbers reveal a pattern: increasing by 6 within groups of 4. This is because of the 6 columns and 4 rows. After the second encryption, it is not so obvious to identify patterns.

The point is that while a single application of the transposition cipher did not seem to offer much security (in terms of hiding patterns), adding the second application of the cipher offers an improvement. This principle of repeated applications of simple operations is used in modern ciphers.

Summary of Transposition and Substitution Ciphers

- Transposition ciphers on their own offer no practical security
- But combining transposition ciphers with substitution ciphers, and repeated applications, practical security can be achieved
- Modern symmetric ciphers use multiple applications (rounds) of substitition and transposition (permutation) operations