Tiny Book of Simple Cryptography

Alfred C Thompson II

Contents

Introduction

Modern cryptography is filled with complicated algorithms, intricate mathematics, and sophisticated software. It was not always the case. Before the age of computers many, simpler, encryption methods were in use. In this booklet, we will look at some older methods and explore writing code to encrypt and decrypt messages.

Some of these are known as Substitution ciphers[1] which means that letters, individually or as a group are replaced with other letters. There are substitution ciphers where letters are replaced with other symbols or numbers as well. For example, ASCII [2]is a cipher that replaces letters and other characters with numbers.

Other simple ciphers are transposition ciphers[3]. In transposition ciphers letters are moved around according to a pre-determined pattern.

Throughout this booklet are footnotes that link to articles on Wikipedia that provide more information about the topics under discussion. These may be used as a starting point for more research and more details.

This booklet is intended as a starting point for experimentation and further learning. It is not intended as a definitive work.

1 https://en.wikipedia.org/wiki/Substitution_cipher
2 https://en.wikipedia.org/wiki/ASCII
3 https://en.wikipedia.org/wiki/Transposition_cipher

Caesar Cipher[4]

Introduction

The Caesar cipher is one of the oldest and most well-known cipher. While it may or may not have been invented by Caesar it became famous as something he used frequently. The cipher is simple in that each letter is replaced by a letter some number of positions after it in the alphabet. Caesar rotated the letters by 3 so that today we would replace the letter A with the letter D, the letter B with E, and so on.

The number used to encrypt a message is a shared secret. A shared secret is something that the sender and received both know but that no one else knows. Shared secrets must be shared before messages can be sent. This is a weakness of many ciphers.

Encrypting

Encrypting a message becomes a matter of knowing the ordinal position of a letter, adding the offset, and replacing the original letter with the letter at the indicated position. What if the position indicated is beyond the count of letters in the alphabet? In that case we start over at the beginning. Modulus arithmetic can help us there. The modulus operation returns the remainder of a division operation. For example, if we add 13 to position 16 (T) we get 29 which is beyond the number of letters in the English alphabet. 29 modulus 26 (the number of letters in our alphabet) returns 3 which leads us to D (A would be 0).

The formula is Location = (Position + Offset) % 26

Decrypting

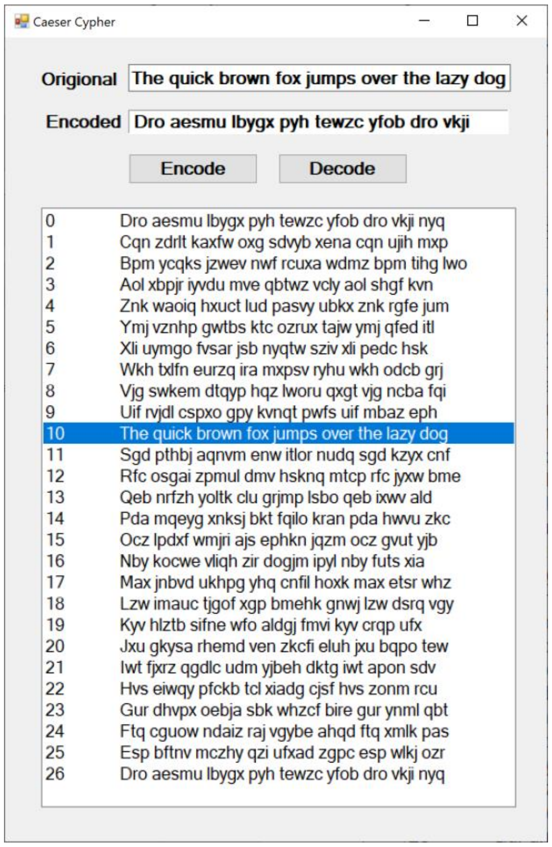
Decrypting an encrypted message is done by subtracting the offset from the encrypted value to get the location of the original letter. Our problem here is that our location may be before the beginning of the alphabet (less than zero). This means that we must check for the case of a negative location and use that value to subtract from 26 to get the correct location.

Cryptography Issues

There are several issues with this cipher than keep it from being secure in a modern world. The first problem, which it shares with many substitution ciphers, is that it is susceptible to breaking though letter frequency analysis. Some letters occur more frequently in writing than others. In English, E is the most used letter. The letter Z is used least frequently (there is a table showing frequency in the appendix). We can count the number of occurrences of each letter in an encrypted message. Given a large enough sample we can make some good assumptions about what letter is replacing what other letter. With the Caesar cipher we can deduce the rotation based on the letters we believe we have identified and so decrypt the rest of the message.

Modern computers give us our second easy way to break this encryption. There are only 25 possible rotations. Zero and 26 are the same as not encrypting at all. A simple program that implements all 25 possibilities will quickly show a clear plaintext result. The following image shows one implementation. We can easily see that the encrypting here used a rotation of 10 characters.

[4] https://en.wikipedia.org/wiki/Caesar_cipher



Origional: The quick brown fox jumps over the lazy dog

Encoded: Dro aesmu lbygx pyh tewzc yfob dro vkji

```
0         Dro aesmu lbygx pyh tewzc yfob dro vkji nyq
1         Cqn zdrlt kaxfw oxg sdvyb xena cqn ujih mxp
2         Bpm ycqks jzwev nwf rcuxa wdmz bpm tihg lwo
3         Aol xbpjr iyvdu mve qbtwz vcly aol shgf kvn
4         Znk waoiq hxuct lud pasvy ubkx znk rgfe jum
5         Ymj vznhp gwtbs ktc ozrux tajw ymj qfed itl
6         Xli uymgo fvsar jsb nyqtw sziv xli pedc hsk
7         Wkh txlfn eurzq ira mxpsv ryhu wkh odcb grj
8         Vjg swkem dtqyp hqz lworu qxgt vjg ncba fqi
9         Uif rvjdl cspxo gpy kvnqt pwfs uif mbaz eph
10        The quick brown fox jumps over the lazy dog
11        Sgd pthbj aqnvm enw itlor nudq sgd kzyx cnf
12        Rfc osgai zpmul dmv hsknq mtcp rfc jyxw bme
13        Qeb nrfzh yoltk clu grjmp lsbo qeb ixwv ald
14        Pda mqeyg xnksj bkt fqilo kran pda hwvu zkc
15        Ocz lpdxf wmjri ajs ephkn jqzm ocz gvut yjb
16        Nby kocwe vliqh zir dogjm ipyl nby futs xia
17        Max jnbvd ukhpg yhq cnfil hoxk max etsr whz
18        Lzw imauc tjgof xgp bmehk gnwj lzw dsrq vgy
19        Kyv hlztb sifne wfo aldgj fmvi kyv crqp ufx
20        Jxu gkysa rhemd ven zkcfi eluh jxu bqpo tew
21        Iwt fjxrz qgdlc udm yjbeh dktg iwt apon sdv
22        Hvs eiwqy pfckb tcl xiadg cjsf hvs zonm rcu
23        Gur dhvpx oebja sbk whzcf bire gur ynml qbt
24        Ftq cguow ndaiz raj vgybe ahqd ftq xmlk pas
25        Esp bftnv mczhy qzi ufxad zgpc esp wlkj ozr
26        Dro aesmu lbygx pyh tewzc yfob dro vkji nyq
```

The fact that secrets must be shared and that someone may discover the secret is also an issue as it is with any cipher that requires shared secrets.

Project Suggestions

1. Write a program to encrypt and decrypt using the Caesar cipher.
    a. Include options to enter either plaintext or encrypted messages
    b. Include an option to set the number of characters to rotate letters
2. Write a program to count the letters in the text in a data file. Compare those counts with the chart in the appendix.

Vigenère cipher[5]

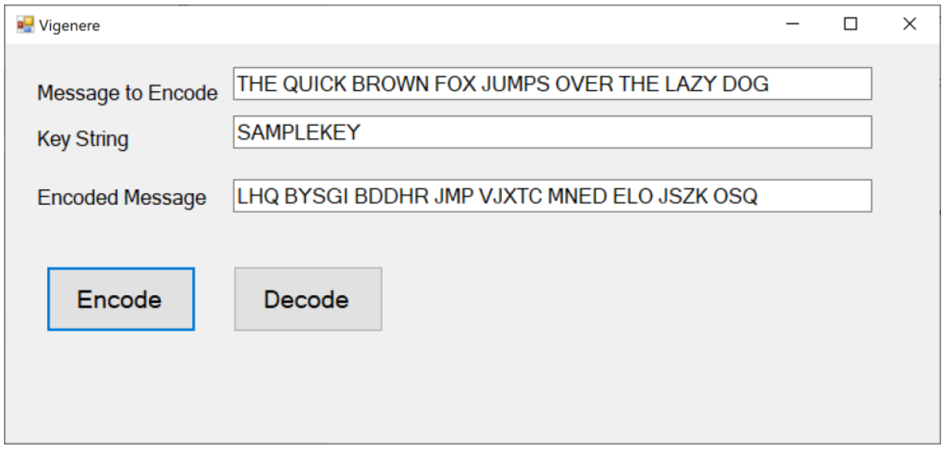
Introduction

As we have seen, the Caesar cipher is both easy to use and easy to break. The Vigenère cipher takes the Caesar cipher a big step further and was considered unbreakable for hundreds of years. This cipher rotates each letter by a different number of places. The rotation values are determined by a key, normally a phrase, that both the sender and receiver agree to in advance. Another shared secret.

Encrypting

The ordinal position for each letter is used as the rotation factor for the corresponding letter in the plain text. For example, in the figure below, S or 18 is the offset for the letter T in the plaintext which gives us the letter L. The letter A, the second letter in the key string, is in position zero so the second letter in the plaintext, H, is not changed at all. This is not considered a problem as anyone intercepting the message would assume that the letter had been changed.

If the key phrase has fewer letters than the plaintext being encrypted, the phrase is used repeatedly until the whole message is encrypted. Longer keys create messages that are harder to break.



Message to Encode	THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
Key String	SAMPLEKEY
Encoded Message	LHQ BYSGI BDDHR JMP VJXTC MNED ELO JSZK OSQ

Decrypting

The key phrase is used to decrypt the message once it has been received. Therefore, both sender and receiver must know the key.

Cryptography Issues

These messages are usually written with all letters in either upper or lower case. Having mixed case letters gives information about which letters may indicate the start of words such as the first word in a sentence or the start of a proper noun. Spaces also give away information about the size and number of words in a message. Encrypted messages are more secure if spaces are removed from the message.

Keys must be shared in advance or transmitted in some other secure manner. This can complicate the use of the code because it makes it hard to use the code with new senders or receivers.

[5] https://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher

Modern super computers can break this cipher though several proven attacks. It will probably be safe from casual readers but not from a major organization or government.

Project Suggestions

1. Write a program to encrypt and decrypt messages using the Vigenère cipher.

One Time Pad

Introduction

The one-time pad[6] is generally considered the unbreakable code. At least within some given constraints discussed below in Cryptography Issues. The concept is that the message is encrypted using a set of random numbers with the set being the same length as the plain text message. A set of random numbers is used only once which is what gives this cypher its name.

Encrypting

Encrypting using a one-time pad is very similar to encoding using a Vigenère cipher except that the rotations are determined by the numbers in the one-time pad and not from a keyword.

Decrypting

Decrypting a crypto text that was encoded using the one-time pad requires that the same one-time pad that was used to encode the message is used to determine the rotation of letters to recover the plaintext.

Cryptography Issues

This cipher is useful only of both the sender and receiver have the same set of numbers or pad. This means that some way of passing this information along securely is possible. If a third party were to get a copy of the pad the code is insecure and can not be used. Also, the set of numbers has to be truly random. This is a difficult problem[7]. Computer generated lists of random numbers are not truly random. They are technically referred to as pseudo random. They are random enough for many uses but not for cryptography or other applications like lotteries where true random numbers are necessary.

There are several websites on the internet that promise to provide truly random sets of numbers and two of them are listed here.

- https://www.random.org/integers/
- https://numbergenerator.org/true-random-numbers

Trusting internet websites for random numbers for cryptography is a risky venture and companies and governments who depend on security often have their own sources of random numbers. These websites provide lists that can be used for testing program code and for casual use. There are ways of verifying the randomness of numbers that can be used.[8] That is beyond the scope of this book.

Project Suggestions

Create a program that encodes a plain text message using a list of random number read from a file. Strip spaces from the plaintext before encoding. The program must read at least as many numbers as the plaintext is long. The program must also be able to decode a message using the same list of random numbers.

Option: Most pseudo random number routines will generate a list of numbers based on a seed value. Calling the routine twice with the same seed will result in duplicate lists. This is a possible way of

[6] https://en.wikipedia.org/wiki/One-time_pad
[7] https://en.wikipedia.org/wiki/Random_number_generation
[8] https://en.wikipedia.org/wiki/Random_number_generation#Post-processing_and_statistical_checks

creating a program if you don’t have easy access to files with lists of random numbers. It is not a secure method and should not be used for serious cryptography.

Polybius Square

Introduction

The Polybius[9] Square[10] was discovered by Greek scholars named Cleoxenus and Democleitus and made famous by the historian Polybius. It was used to send messages more easily. The basic idea is a 5 by 5 grid holding the letters of the alphabet. The letters were identified by the x and y coordinate numbers. Polybius used the Greek alphabet with 24 letters which fit nicely in a 5 by 5 grid. A (alpha) was 11. Ω (Omega) was 54.

	1	2	3	4	5
1	Α	Β	Γ	Δ	Ε
2	Ζ	Η	Θ	Ι	Κ
3	Λ	Μ	Ν	Ξ	Ο
4	Π	Ρ	Σ	Τ	Υ
5	Φ	Χ	Ψ	Ω	

The English alphabet has 26 letters which doesn't fit so the same box is often used for the letters "I" and "j". Each letter is then represented by a two-digit number that identifies its location in the square.

	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I/J	K
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

Alternatives to "I" and "j" are the letters "C" and "K" sharing a box, but other pairs can be used as well by prior agreement.

[9] https://en.wikipedia.org/wiki/Polybius_square
[10] Phttps://crypto.interactive-maths.com/polybius-square.html

In some cases, a 6 by 6 square that includes the digits zero through nine is used. This also avoids the need for two letters to share a box as well as avoiding the need to spell out numbers.

	1	2	3	4	5	6
1	A	B	C	D	E	F
2	G	H	I	J	K	L
3	M	N	O	P	Q	R
4	S	T	U	V	W	X
5	Y	Z	0	1	2	3
6	4	5	6	7	8	9

Six by six grids are also used for alphabets that have more than 25 or 26 letters. Russian has 33 letters for example.

In the case of the 6 by 6 English square the letter "A" is identified by 11. The number "1" is identified by 54. The letter "O" is identified by the number 33.

Having the square in a standard order is easy and was the originally way squares were used but to hide a message we want something different.[11] Using a keyword and ignoring any duplicate letters is one possible way and probably the most common. Other potential ways to build a square include reversing the alphabet or sharing a randomly generated square. The figure below uses the keyword "Polybius" as the beginning of the square with the rest of the alphabet following without duplicating any letters from the keyword. As shown in the next figure.

	1	2	3	4	5	6
1	P	O	L	Y	B	I
2	U	S	A	C	D	E
3	F	G	H	J	K	M
4	N	Q	R	T	V	W
5	X	Z	0	1	2	3
6	4	5	6	7	8	9

Encrypting

The first step in encrypting a message is to build the Polybius Square. The letters of the keyword are entered in to grid first followed by the rest of the alphabet, being sure not to duplicate letters. Each

[11] https://en.wikipedia.org/wiki/Polybius#Cryptography

letter in the plaintext is located in the square and the x and y coordinates of that location are used to represent the letter.

Using the square above the plaintext “SAMPLETEXT” becomes “11215200205133510433”.

Decrypting

Decrypting also starts with creating a square. The encoded message is broken into two-digit codes which are further broken down to x and y coordinates and used to identify the original letter from the plain text.

Cryptography Issues

Like any other single transposition, the Polybius Square is susceptible to frequency analysis. Modern computers can also break it via brute force methods of trying all possible combinations. This encryption has very limited use if real security is needed.

Project Suggestions

Creating a program to create and use a Polybius Square is a good project. A simple 5 by 5 square without a keyword is the easiest first project.

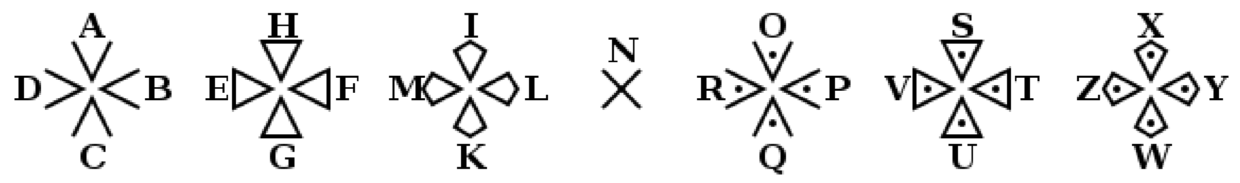
Creating a Polybius Square using a keyword is more complicated than not using a keyword in the square. This makes a good second project.

PigPen Cipher[12]

Introduction

One of the simplest and oldest methods of encoding messages is by replacing letters with other symbols. After all, letters are really symbols for sounds. The PigPen cipher is one well-known example. It has been called several other names over the years including the masonic cipher, Freemason's cipher, Napoleon cipher, Templar Cipher, and tic-tac-toe cipher. Versions of it have been used at least as early as the 16th century.

For example, the Knights Templar used a version based on the Templar Cross.

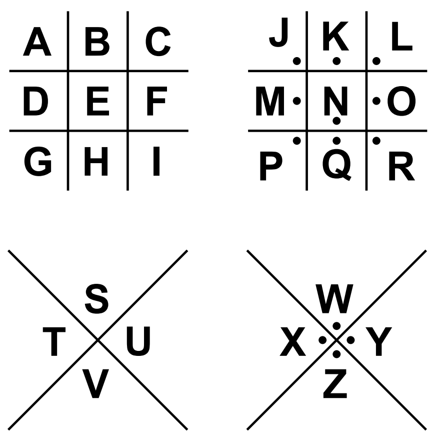


Templar Cipher

Other versions are based on grids or X shapes with additional items like dots.

Encrypting

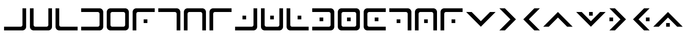
The figure below shows one popular version of the PigPen cipher. Each section of the grids defines a letter with a dot in the section differentiating the different grid.



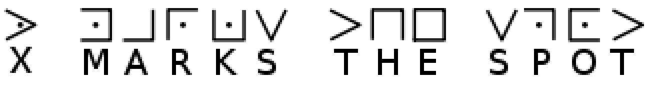
PigPen Grid

The message is encoded by using the grid segment, empty of letters but including a dot if necessary, to represent the letters that make up the message. For example, the letter A would be represented by the lines for the top left of the first grid – ⌟ and B would be the lines for the top center of the grid - ⊔. The letter W would be the lines and a dot for the top section of the last X grid – ⋁̇.

[12] https://en.wikipedia.org/wiki/Pigpen_cipher

The alphabet looks like this: 

A complete message would consist of the various symbols as displayed in the image below.



Coded Message with Decoded Result

Messages are encoded by referring to the diagrams, but some users may memorize the code. The memorization is made easy by the general organization of the grids.

Decrypting

Decrypting a message is a matter of comparing the symbols to the encoding grid. Each symbol can be compared to the grid and finding the corresponding part of the grid and reading the letter in that location.

Cryptography Issues

The use of simple grids makes it easy to share the code as well as to encrypt and decrypt messages. It is easy to reproduce a key for encrypting or decrypting messages if one knows the pattern used. For this reason, many different patterns can be used. For example, grid X grid X would give a different result than grid grid XX as was used in our example above. Other variations are moving the letter is a grid down first and then across or using only grids with different numbers of dots in sections. Of course, the pigpen method could be combines with other forms of substitution such as Caesar or Vigenère. In any case, the receiver must know the patterns used in advance. If the patterns are not changed from time to time the cipher becomes easier to decode (break).

Project Suggestions

There is a TrueType font of the PigPen symbols available at https://fontstruct.com/fontstructions/show/447940/cipher_code It can be used to display encoded messages. For example, the word “pigpen” could be displayed as . This font was used to display encoded messages in this document.

Create your own pattern for encrypting messages. Use that pattern to send a message to someone else and see if they can decrypt it.

Columnar Transposition Cipher

Introduction

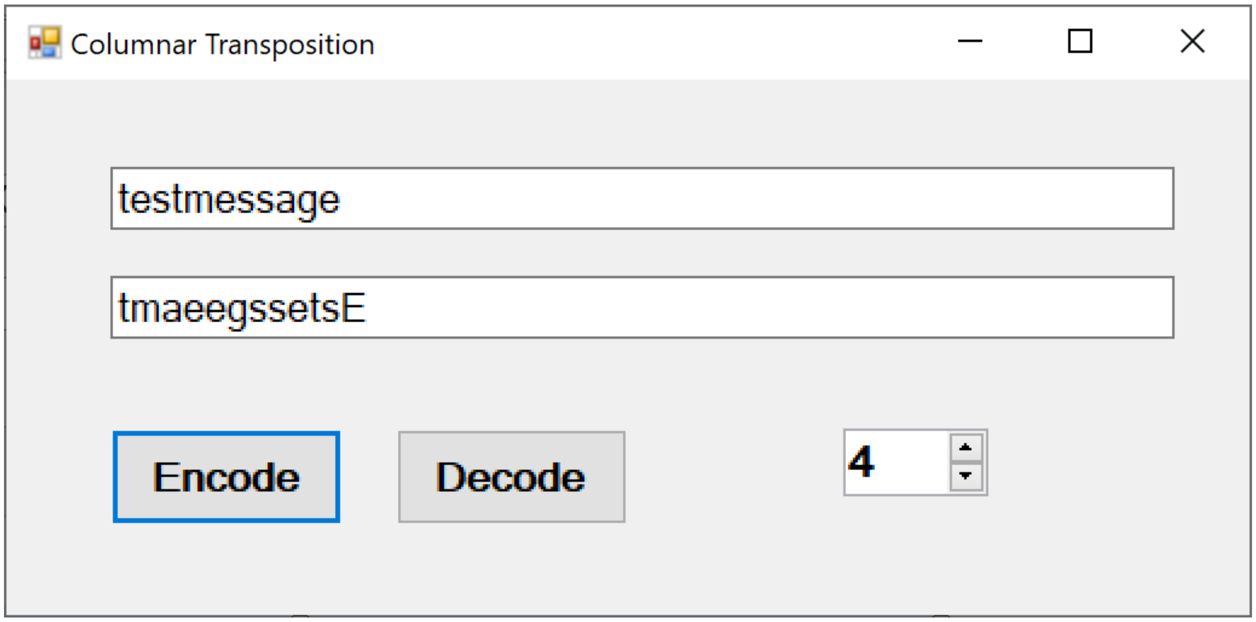
In a columnar transposition, plaintext letters are written out in rows and the encrypted message is built by taking the letters from columns. The simplest form of this is the take the columns in order but more secure ciphers process the columns in an order specified by a keyword or previously shares secret.

Encrypting

In this simple example (figure below) we are writing the message TESTMESSAGE in a four-by-four grid. We are padding trailing spaces with the letter Z. In some variations, the end of the message is padded with randomly selected letters. In other variations, not padding is used.

T	E	S	T
M	E	S	S
A	G	E	E

The coded message would be TMAEEGSSETSE if we move down from the first column to the last in order.



testmessage

tmaeegssetsE

4

In the figure above I have written the message in lowercase but padded the string with uppercase letters so that you could see the padded letters. Ideally you would use the same case for the padding as you would use in the message itself to avoid giving a code breaker more information.

There are any number of ways of specifying the number of columns and the order in which they are to be used to build the encrypted message. For example, a key word can be used where the number of letters in the word indicates the number of columns, and the alphabetical order of the letters indicates the order in which to process the columns. The number of columns and order could be shared in advance as well.

Decrypting

Decrypting is the opposite process. One writes into the columns and reads across the rows. Two dimensional arrays are ideal for programming this process. There are other ways of doing it and I leave that to your imagination.

Cryptography Issues

This method of encrypting messages was in common use until the 1950s when computers made it possible to try many combinations of columns in a relatively short period of time. As with other methods, spaces and mixed case letters provide a potential code breaker with useful information. If the message is prefixed with a key word that indicates the number of columns and the order in which to process them breaking one instance of a code will often give away future instances that use different key words.

Project Suggestions

1. Write a program to rearrange the letters in a message using a specified block size (number of columns).
2. Write a program to rearrange the letters in a message using a number of columns and an order to process them based on a key word.

Keyword Columnar Transposition Cipher

Introduction

A Keyword Columnar Transposition Cipher[13] isa columnar transposition that moves columns based on a shared key word or key phrase. Plaintext is written out under columns identified by a keyword or phrase. The length of the keyword is the number of columns. An initial square with the key phrase "amystery" might look like the figure below.

a	m	y	s	t	e	r	y
t	h	e	q	u	i	c	k
b	r	o	w	n	f	o	x
j	u	m	p	s	o	v	e
r	t	h	e	l	a	z	y
d	o	g					

The next step is to sort the columns based on the alphabetically sorted letters in the keyword. For example, the "t" in "amystery" indicates the 5th column (position 4 when counting from zero) but is the sixed letter in the work alphabetically. That means that the "t" column in the plaintext square becomes the sixed (5th position) in the encrypted square.

Note that squares that are not filled by letters from the plaintext may be either left as empty spaces, as here, or filled with random letters.

a	e	m	r	s	t	y	y
t	i	h	c	q	u	e	k
b	f	r	o	w	n	o	x
j	o	u	v	p	s	m	e
r	a	t	z	e	l	h	y
d		o				g	

The resulting grid can be read out either by rows or columns. Reading out by rows, the plaintext "thequickbrownfoxjumpsoverthelazydog" using the keyword "amystery" would be encoded as "tihcquekbfrownoxjouvpsmeratzelhyd o   g". Both the sender and the reciever have to know the keyword and to read by rows or columns.

Encrypting

The first step in encoding a message is to remove the spaces, as is typical for coded messages, and to build the grid. The message is written out row by row. The next step is to determine the alphabetical

[13] https://en.wikipedia.org/wiki/Transposition_cipher#Columnar_transposition

order of the letters in the keyword/phrase. Then the columns are reordered based on the order of the keyword letters. The message in the block is listed in order by row or column.

If programming, the most difficult part of the process may be tracking the order of the columns. There are probably many possible ways to do this. One way is to build an array of two-character symbols, the first character a letter from the keyword and the second an integer indication the original location of the letter. This array can be sorted and the position in the array indicating the destination column and the number indicating the source column.

Decrypting

Decoding is much the same as the encryption process. The grid in made using the encoded message. The original and sorted order of the letters is used to reorganize the columns back to their unencoded positions.

Cryptography Issues

This cipher can be broken by guessing at the number of columns, building squares, and looking for anagrams. This method also lends itself to brut force methods that are practical with powerful computers and good software.

Project Suggestions

Create a program that encodes and decodes a message using a keyword and associated columnar transposition.

For more advanced work, allow a choice between outputting the cipher text by row or by column and the user’s option.

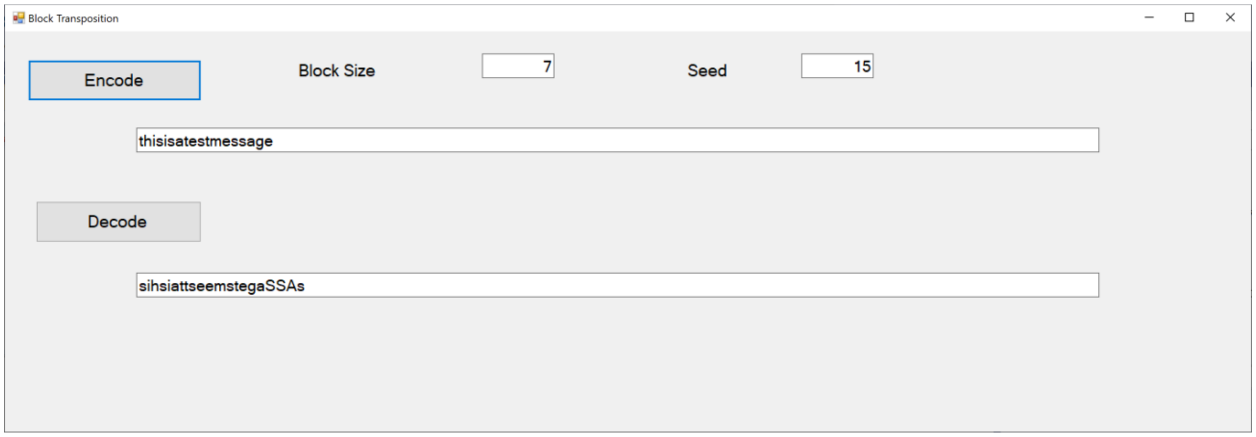
Random Block Transposition Cipher

Introduction

Random number generators as used in most programs are not truly random. They are known as pseudo random numbers[14]. While this has disadvantages for most cryptographic usage. They can be used for a simple block transposition cipher. That is because if the program provides the random number routine with a specified seed value the random number generator will calculate the same not quite random values.

Encrypting

For this method a block size is selected. The letters in each block are rearranged in a random order. The random order is determined by a specific pseudo random number generated with a specific seed. The seed and the block size are shared secrets. As with other transposition ciphers, the plaintext is padded with random letters to avoid spaces and other information that would provide extra information to potential cipher breakers.



Block Size 7

Seed 15

thisisatestmessage

sihsiattseemstegaSSAs

Decrypting

Decrypting involves using the same seed, shared secret, and the same pseudo random number generator to build the same order of transpositions in the same size block to move letters to their original position.

Cryptography Issues

Block sizes that are too small and which may be tried give too few possibilities for rearrangement. Larger block sizes are harder to crack as there are more possibilities. Brute force methods of cracking are still possible with fast computers.

Project Suggestions

Write a program that accepts a message with a block size and random number generator seed and encrypts and decrypts as message

[14] https://en.wikipedia.org/wiki/Pseudorandom_number_generator

Steganography

Steganography "is the practice of concealing a file, message, image, or video within another file, message, image, or video.[15]" Steganography hides the fact that there is a message. A file can be passed around in various forms without anyone, in theory at least, knowing that information is being passed along with it.

Encrypting

Introduction

There are many ways of hiding messages in an image. One way involves using the way that colors are stored in images. Colors are stored in a combination of red, green, and blue. Each color, in the most common form includes 255 levels of each of the three primary colors. The difference between 250 and 251 for one color is generally far too subtle for the casual viewer. We can take advantage of this to store information using one bit in each pixel. For example, we can use the last bit of the color blue and eight pixels to represent one ASCII character.

Our encoding routine would select a known location in the image to start the message. The first 8 pixels might be used to represent the total number of characters in the message.

For each character, including the letter count, our program would build a pattern of ones and zeros. The least bit in eight pixels would be set or left alone to match the pattern. Many bits would not be changed because they were already set the way our message requires them to be. The fact that the decryption program knows that the pixel is part of the message is all that is needed for this to work. Unchanged pixels make detection more difficult.

One thing to make note of is that many forms of compression will make changes to the file that make it almost impossible to recreate the hidden message. Image formats that use lossy compression[16] to save space will generally destroy the hidden message. Image formats like JPEG are examples of this. Using a JPEG image is not going to be reliable.

Decrypting

Decrypting involves first knowing the message in the file and where to look for it. Once the location and formatting of the message is known the decoding program checks the values that are stored and constructs a legible message.

Cryptography Issues

Comparing a file with a message with the original file, before a message was encoded in it, will provide many clues to the message hidden it.

Project Suggestions

1. Create a program to hide a message in an image file (recommended BMP or PNG).

[15] https://en.wikipedia.org/wiki/Steganography

[16] Lossy compression - Wikipedia

a. Program should modify the bits in known locations of an image file and rewrite the new modified file.
b. Program should read a file with an encoded message and decode it.

Bacon’s Cipher

Introduction

Bacon's cipher[17] is a well-known and simple cipher. Its considered a form of steganography[18]. Unlike most steganography, the code is hidden in the text and not an image. The format is a 5-bit binary encoding where each letter of the plaintext is replaced by a group of five of the letters 'A' or 'B'. This binary code has A as 00000 or "aaaaa" and Z is 11010 or "bbaba". There are variations and the original version used the same codes for the letters "I" and "J" and for "U" and "V". The usual implementation is to use one type face for "a" and a different type face for "b".

Often this is done with a serif and a sans serif font. One option, and the first one I use below, uses italics and non-italics.

t*he*qu*ic*k*brown*fox*es*ju*m*p*edov*er*t*he*l*az*y*

One other option I have used is mixing upper-case and lower-case letters.

The QUicK brOwn FoXes JUmPed ovER tHe lAZy

One semi-famous example of the use of serif and Sans Serif fonts is the Cipher on the William and Elizebeth Friedman tombstone.[19] The biography of *Elizebeth Friedman,The Woman Who Smashed Codes: A True Story of Love, Spies, and the Unlikely Heroine Who Outwitted America's Enemies*[20],is a seriously recommended read.

There are also examples of using this encoding in pictures of groups of people. Some people facing straight ahead, and some turned to the side. Just about any possibility of two different options would work. It’s a flexible system.

Encrypting

Encrypting a message requires a plaintext message where the secret message will be hidden. Each letter in the secret message must be mapped to its A/B code. The plaintext message is then modified so that each group of 5 letters is set to the appropriate mix of A or B indictors. Secret messages traditionally do not include spaces because spaces make it easier to break encoded messages into words. The plaintext may have spaces, but they will be left unchanged by the encryption process. Spaces will be ignored during decoding as well.

The following table shows the various encoding using both A/B code and Binary codes.

[17]https://en.wikipedia.org/wiki/Bacon's_cipher
[18] https://en.wikipedia.org/wiki/Steganography
[19] https://www.elonka.com/friedman/
[20] https://smile.amazon.com/Woman-Who-Smashed-Codes-Outwitted-ebook/dp/B01M0EOI6I/

Letter	Code	Binary	Letter	Code	Binary
A	aaaaa	00000	N	abbab	01101
B	aaaab	00001	O	abbba	01110
C	aaaba	00010	P	abbbb	01111
D	aaabb	00011	Q	baaaa	10000
E	aabaa	00100	R	baaab	10001
F	aabab	00101	S	baaba	10010
G	aabba	00110	T	baabb	10011
H	aabbb	00111	U	babaa	10100
I	abaaa	01000	V	babab	10101
J	abaab	01001	W	babba	10110
K	ababa	01010	X	babbb	10111
L	ababb	01011	Y	bbaaa	11000
M	abbaa	01100	Z	bbaab	11001

Decrypting

Decrypting involves viewing the groups of five characters, determining the A/B or 0/1 values, and matching up with the appropriate letters. Spaces in the plaintext are ignored. Spaces may often be removed before processing the text for a message.

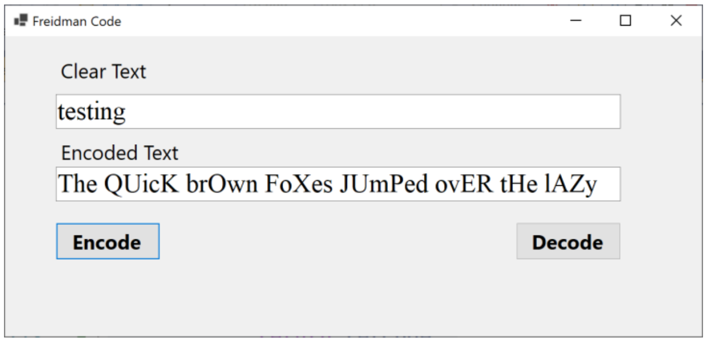
Cryptography Issues

A cipher like this works best if the differences are not too obvious to the casual observer. The big disadvantage of using mixed case letters is that it is not very subtle. It's almost obvious that there is something special going on. Italics are less noticeable. Serif and Sans Serif are even more subtle.

This is not the sort of encryption that holds up to serious attack. It relies almost completely on obscurity; of people not knowing there is a code at all.

Project Suggestions

Create a program to embed a message in a plaintext using a mix of uppercase and lowercase letters. The program should be able to decrypt the message as well. The program requires a plaintext message where the secret message will be hidden. The plaintext must have 5 times as many letters as the secret message. This is a sample program screen.



Clear Text

testing

Encoded Text

The QUicK brOwn FoXes JUmPed ovER tHe lAZy

One important piece of description is determining if a letter is uppercase or lowercase. In some systems, Microsoft's .NET Framework for example, the is an IsUpper method that will return true for uppercase letters or false for lowercase letters. If you do not have such a method in the programming environment you are using there is some information in the sample code chapter that may help. See **Determine if a letter is uppercase or lowercase**.

Code Samples

Introduction

These samples were written and tested using the C# programming language and are provided for reference with no guarantee they will work in your program or programming language.

Padding a string with random letters

```
private string padString(string input, int size)
{
    /* Create a string padded the a requested length
     * Inputs: Original string & Length of desired output string
     * Assumes adding to a string
     * Returns: New string with random letters as padding to length requested
     */
    string alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    Random R = new Random();

    for (int i = input.Length - 1; i < size - 1; i++)
    {
        input += alphabet.Substring(R.Next(alphabet.Length), 1);
    }
    return input;
}
```

Creating an array of numbers in a random order

```
private int[] BuildKeys(int length, int seed)
{
    /*
   * Builds a key string
   * Inputs: Length of key to create & Seed for random number generator
   * Output: integer array to be used as a key for encoding and decoding
   */
    Random r = new Random(seed);         // Use seed value to start generator
    int[] working = new int[length];    // Create an array of the required length
    for (int index = 0; index < length; index++) // Initialize key array
    {
        working[index] = index;
    }
    // Shuffle the key array
    for (int index = 0; index < length; index++)
    {
        int temp = working[index];
        int swap = r.Next(length);
        working[index] = working[swap];
        working[swap] = temp;
    }
    return working;
}
```

Determining the ordinal position of a letter

For English letters, the position in the alphabet of a letter can be done easily by taking advantage of the ASCII codes. Using a Char data type the integer value of the char subtracted by the integer value of the letter 'a' or 'A' results in what position in the alphabet. This example assumes that c is a variable of type Char.

```
int x = (int)c - (int)'a';
```

An alternate way is to create a string of the letters in order and search for the position of your letter in that string. Assuming s is a string variable with a single letter stored in it.

```
string alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
int location = alphabet.IndexOf(s);
```

Building a String Representation of a Binary ASCII value

```
string OnesAndZeros(char c)
{
  /* builds as string of ones and zeros that represent the Binary ASCII value of a char
   * The Binary AND operator (&) is used to check the bit settings within the char
  */
    string output = "";
    if (((int)c & 128) != 0) output += "1"; else output += "0";
    if (((int)c & 64) != 0) output += "1"; else output += "0"; ;
    if (((int)c & 32) != 0) output += "1"; else output += "0"; ;
    if (((int)c & 16) != 0) output += "1"; else output += "0"; ;
    if (((int)c & 8) != 0) output += "1"; else output += "0"; ;
    if (((int)c & 4) != 0) output += "1"; else output += "0"; ;
    if (((int)c & 2) != 0) output += "1"; else output += "0"; ;
    if (((int)c & 1) != 0) output += "1"; else output += "0"; ;
    return output;
}
```

Determine if a letter is uppercase or lowercase

If you do not have such a method available there is an easy way to determine if an ASCII value is uppercase or lowercase. In ASCII, there are 32 characters between an uppercase letter and a lowercase letter. For example, the uppercase A is 65 and the lowercase a is 97. This means that the 32 bit is set, or 1, for lowercase letters and unset, or 0, for uppercase letters. Many languages support operators that support bit operations. In many languages the & sign [single & not to be confused with && which is a Boolean AND] that lets a programmer compare values at the binary level. The following code uses the & operator to see if the 32 bit is set.

```
private Boolean isUpper(char c)
{
    Boolean retCode = false;
    int check = c & 32;
    if (check == 0) retCode = true;
    return retCode;
}
```

Appendix

Frequency of Letters in English[21]

e	13.0%	m	2.4%
t	9.1%	w	2.4%
a	8.2%	f	2.2%
o	7.5%	g	2.0%
i	7.0%	y	2.0%
n	6.7%	p	1.9%
s	6.3%	b	1.5%
h	6.1%	v	1.0%
r	6.0%	k	0.8%
d	4.3%	j	0.2%
l	4.0%	x	0.2%
c	2.8%	q	0.1%
u	2.8%	z	0.1%

Non-English Alphabets

https://en.wikipedia.org/wiki/Letter_frequency#Relative_frequencies_of_letters_in_other_languages

Greek Alphabet[22]

Α α, Β β, Γ γ, Δ δ, Ε ε, Ζ ζ, Η η, Θ θ, Ι ι, Κ κ, Λ λ, Μ μ, Ν ν, Ξ ξ, Ο ο, Π π, Ρ ρ, Σ σ/ς, Τ τ, Υ υ, Φ φ, Χ χ, Ψ ψ, and Ω ω.

Norwegian Alphabet[23]

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	Æ	Ø	Å

Turkish Alphabet[24]

A	B	C	Ç	D	E	F	G	Ğ	H	I	İ	J	K	L	M	N	O	Ö	P	R	S	Ş	T	U	Ü	V	Y	Z

[21] https://en.wikipedia.org/wiki/Letter_frequency#Relative_frequencies_of_letters_in_the_English_language

[22] https://en.wikipedia.org/wiki/Greek_alphabet#Letter_shapes
[23] https://en.wikipedia.org/wiki/Danish_and_Norwegian_alphabet
[24] https://en.wikipedia.org/wiki/Turkish_alphabet#Letters