

Linnaeus University

1DV700 - Computer Security

Assignment 1

Student: Evan Huynh

Personal number: 970330-4130

Student ID: eh223im@student.lnu.se



Setup Premises

Explain your setup such as, OS, web browser, tools being used, development environment, and whatever else is necessary…

OS: OS X 10.13 on MacBook Pro.

IDE: IntelliJ IDEA 2019.3

Java version: 13

Web browser: Google Chrome (ver 79 as the time of writing)

Specific tool used:

Task 2: <https://hexed.it>

Task 3: <https://quipqiup.com/>

Task 6: <https://www.dcode.fr/monoalphabetic-substitution>

# Task 1

1. The first task is to investigate different terms within cryptography and related areas.
   1. Different between pairs of methods:

Symmetric encryption: Alice wants to send Bob the message P. She encrypts the message using key K which results . If Bob wants to read P, he has to decrypt using the decryption process which results We can see that the same key is used to lock and unlock message. The algorithm to obtain the plaintext is because Decryption and Encryption are the mirror image of the process using the same key K. [1]

Advantages: Using the same key K both to encrypt and decrypt, the algorithm is faster since only one shared key is used.

Disadvantages: If the key K is lost then all the messages are available to the attackers. The transmission of key K which can be both hard and dangerous, due to physical constrain or the communication method.

Asymmetric encryption: different keys are used to encrypt and decrypt the message. The process can be described using [1]. This means that the encryption uses key and the decryption process use key , which is totally different than each other.

Advantages: Alice and Bob do not need to know each other private key in order to read the message. If the algorithm is lost, future messages would not be read since the interceptor will not know the key value. [1]

Disadvantages: The need to keep track of more keys since one key is used to encrypt the message and one key is used to decrypt it.

Encryption algorithms: the encryption algorithm is the process of encoding the message so that the message does not become obvious. [1] However, encryption algorithm is a two-way function since the message has to be readable by both party. If Alice send Bob something, Bob should be able to read it.

Hash algorithms: the hash algorithm digest the message and provide the value, doing the similar thing with the encryption. However, the algorithm should be a one-way function since there should be no way that the attacker can work backward the original message from the hashed value. This mean that once Alice hash something, Bob should NOT have the original message from Alice, just a value to ensure that the message has not been modified.

Compression:

Hashing:

# Task 2

1. Steganography
   1. According to the website, they used the classical technique of hiding the image within image using the least significant bit to achieve it.

Limitation: the hidden image quality is losses due to loss of information. Storing picture within pictures will results the recovered image quality is loss since there are not enough information that could be recovered from the stego file. Also, if the message is too large, the original picture quality might be modified too much that the attacker can notice something wrong with it. For example, hiding 7 bits in the original image might results the low-quality stego image, which indicates something is wrong with the data being sent. [2]

* 1. Since the image is mostly black and white, those white regions are represented in 00 bit. However, I noticed there are a lot of irregularity starting from bit #54 (reference to my Java program). In this image, most of the black pixels are represented by FE, however there are some that are represented by FF which caught my attention. Most of the black are represented by FE, throughout the rest of the file.  
     By running this program, I was able to extract specific region that contain the message. The algorithm was pretty simple, take the bit number, convert to integer and take the modulus 2. If it is even, that bit has a 0 hidden in the LSB, otherwise 1.  
     The rest of the program is just for decorating and house-keeping purpose. I just simply split the string every 8th bit and convert binary string to ASCII.

The final message is “Congratulation!”

# Task 3:

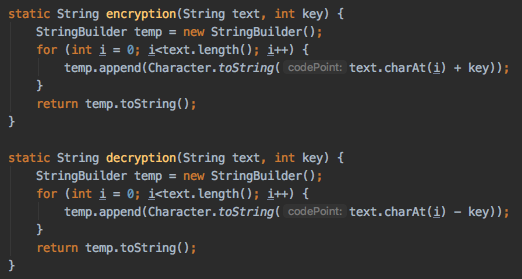
1. Message decryption
2. With pen and paper: “encrypted message”.
3. It could be decrypted without the key. Since this is a monoalphabetic cipher, it could be decrypted with ease. Although frequency analysis of single character might not work 100%, there are also some more popular bigrams (th, he, …), trigrams (the, and, …), not to mention double letters (ss, ee, ll, …) in English.

Knowing I used the tool [3] (mentioned in the beginning) to help me to decrypt the message without the alphabet. It gave ‘exploited message’ as the highest possible result and ‘encrypted message’ as the second possible one.

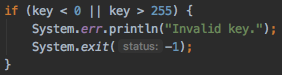
# Task 4

1. Substitution and Transposition cipher

## Substitution cipher



For the substitution cipher program, it is fairly easy to understand. I just take every character, plus the key, which is an integer from 0 to 255 (exactly 256 possible keys). This function will perform the encryption on every letter and then return everything as a string. Since StringBuilder class is faster than string concatenation, I decide to use it as a way to optimize my program. However, to also save time, I will force exit the program in case of invalid key number (smaller than 0 or larger than 255), which is shown below.



Decrypting the cipher text is just a matter of take it and subtract with the provided key.

## Transposition cipher

For transposition cipher, the problem becomes a little bit harder. There are two problems that need to be solved: the key and the encryption algorithm.

For the key, below are the invalid reasons for the key that I have come up:

|  |  |  |
| --- | --- | --- |
| Key | Invalid reason | Note |
| 2013 | Contain 0 | If 0 is in the beginning, it is still invalid since I treat the user input key as a String and read every digit. |
| 214 | Maximum digit is larger than the key | Should not work because we do not know where should the missing letter go |
| 234 | Does not contain 1 | Same reason as above |
| 2113 | Contain duplicate | This exists only to save time |
| -1234 | Contain digit smaller than 0 or larger than 9 | This should not be possible if user enter such an integer since I will convert it into String and split every character into integer array. |

The program, of course will exit if the user enter the invalid key or no input file found.

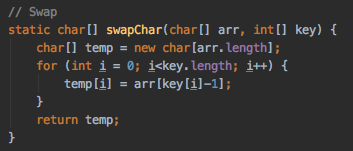
For the encryption/decryption algorithm, since they both works the same, I will just explain one of them.

At first, the algorithm checks for the length of the key (says length) and length of the string ( length). If , then the original string will be appended whitespaces. Algorithmically speaking, this ensures that we have enough characters to swap around.

The second step, I will split the string into arrays, casting both of and into double then cast the ceiling result into integer again. For this, the string is now split into array with equal length, split each of those strings into subarrays containing characters and then the swapping begins.

For the swapping algorithm, I just get ever right character into their right position by just make an new empty array, read both the each character array and key array. Since the key array is the position of the original character array, every character in the new array is the character of the original array of their respected position. Do this for all characters in each subarray and then using StringBuilder to append all the characters in every subarray in ever array.

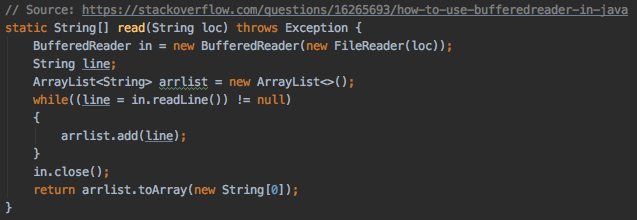
Here is the implementation of the swapping algorithm:



Decrypting the cipher text is just a matter of swapping back the encrypted one since swapping them twice will yield the original result. This algorithm is symmetric.

## File handling

Reading file is just a matter of putting every line into a String array. Below is the algorithm I used [4]:



For writing, I just write every line in the String array of to a new line.

## Main program



Main program is just a method to ask the user which algorithm they want to run and then call the main method of the respected algorithm. The rest of each program is just asking users if they want encrypt or decrypt, key, input file, output file and then perform the main algorithm. Please note that the encryption/decryption in the transposition is just for decoration since both of them work the same no matter what they would enter.

I gave default options for every question since dealing other user inputs is not really interesting compared to the algorithm. Of course, this is not an option for the key or for the in/output file since there is no way that I would know which one of them they want to perform algorithm on or the key.

Calling the main program in each class is not a good practice. This is not a really big deal in this case.

# Task 5

1. Encrypted file

In this task I added the soliloquy by Prince Hamlet in act 3, scene 1 of William Shakespeare’s play Hamlet, mostly known as “To be or not to be” [5]. This file is encrypted using my substitution method with the key ‘123’ and uploaded in my name. Below are the images showing what the file should looks like on my machine before and after encrypting it.

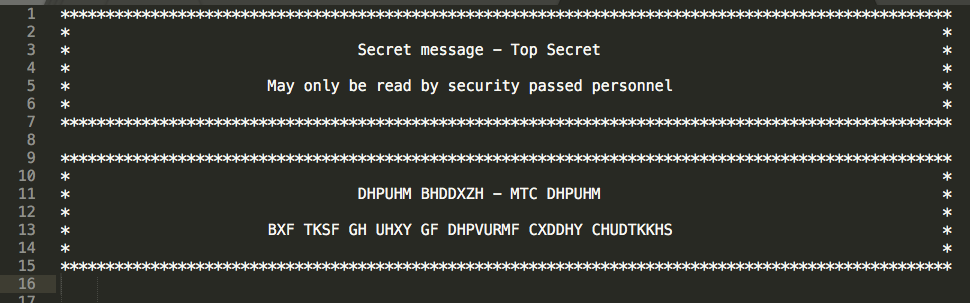
Of course, the encrypted file does contain some special characters and non-printable characters as well. Here is the result.

|  |  |
| --- | --- |
| Original | Encrypted |
|  |  |

# Task 6

1. Cryptanalysis

In this task I picked the easy monoalphabetic cipher by Haofei Yan [6] (given from the name of the file). I immediately noticed that the first two lines give the most of information.



Which gives me this alphabet.

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

|  |  |
| --- | --- |
| **mih** hxusrhdm **jktnk** cutzuxbbxgsh bxpirkh (**mixm** rd x **bxpirkh** **nitdh** **ghixertu** pxk gh ptkmutsshy gf **pixkzhd** mt x "cutzuxb") nxd xs-axoxur'd cutzuxbbxgsh ivbxktry utgtm rk 1206. xs-axoxur'd utgtm nxd turzrkxssf x gtxm nrmi **qtvu** **xvmtbxmrp** **bvdrprxkd** **mixm** qstxmhy tk x sxjh mt hkmhumxrk zvhdmd xm utfxs **yurkjrkz** cxumrhd. **ird** bhpixkrdb ixy x cutzuxbbxgsh yuvb bxpirkh nrmi chzd (pxbd) mixm gvbc rkmt srmmsh **shehud** mixm tchuxmh mih chupvddrtk. mih yuvbbhu ptvsy gh bxyh mt csxf yrqqhuhkm uifmibd xky yrqqhuhkm yuvb cxmmhukd gf bterkz mih chzd mt yrqqhuhkm stpxmrtkd. **xktmihu** **dtcirdmrpxmhy** cutzuxbbxgsh **bxpirkh** gf **xs-axoxur** **nxd** **mih** pxdmsh **pstpj**. | **t?e** earliest **?no?n** programmable mac?ine (**t?at** is a **mac?ine** **??ose** **be?a?ior** can be controlled by **c?anges** to a "program") ?as al-?a?ari's programmable ?umanoid robot in 1206. al-?a?ari's robot ?as originally a boat ?it? **?our automatic musicians** **t?at** ?loated on a la?e to entertain guests at royal **drin?ing** parties. **?is** **mec?anism** ?ad a programmable drum mac?ine ?it? pegs (cams) t?at bump into little **le?ers** t?at operate t?e percussion. t?e drummer could be made to play di??erent r?yt?ms and different drum patterns by mo?ing t?e pegs to different locations. **anot?er** **sop?isticated** programmable **mac?ine** by **al-?a?ari** **?as** **t?e** castle **cloc?**. |

The next step is taking the first paragraph, replaced all the unknown with ? using MS Word ‘Find and Replace’ tool. This step ensures that later on I will not accidentally replace unsolved clues with another one. Then, I started using the same ‘Find and Replace’ tool to solve for the unknowns, which gives the following paragaph on the right hand side.

Because this text is in English (obviously), I underline and bolden some of the words which gives me some informations and the corresponding one on the left, like machine, another, sophisticated, clock, the, etc. This one also mentioned the castle clock machine, a quick Google search gives the name Al-Jazari, which is the inventor of such machine. This results in the bold and underlined text on the alphabet. By this state, only two clues left.

The next step is to search for paragraphs which contain Q and X, which is to search for words contain L, W in the cipher text. So I look at the first sentence in the second paragraph.

|  |  |
| --- | --- |
| mih axplvxuy sttb, yhehstchy rk 1801, rd tqmhk lvtmhy xd x dtvuph tq curtu xum. | the jac?uard look, developed in 1801, is often ?uoted as a source of prior art. |

This immediately gives Q encrypted to L, so X only encrypted to W. This is noted by the bold and underline text with background.

With the completed alphabet, the final step is to decode using tool. I used the website dcode.fr [7] and below is what I get. I copied all the note with no edit on spacing.

|  |
| --- |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* \*  \* SECRET MESSAGE - TOP SECRET \*  \* \*  \* MAY ONLY BE READ BY SECURITY PASSED PERSONNEL \*  \* \*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  COMPUTER PROGRAMMING, HISTORY OF PROGRAMMING  FROM WIKIPEDIA, THE FREE ENCYCLOPEDIA (081110)  THE EARLIEST KNOWN PROGRAMMABLE MACHINE (THAT IS A MACHINE WHOSE BEHAVIOR CAN BE CONTROLLED BY CHANGES TO A "PROGRAM" WAS AL-JAZARI'S PROGRAMMABLE HUMANOID ROBOT IN 1206. AL-JAZARI'S ROBOT WAS ORIGINALLY A BOAT WITH FOUR AUTOMATIC MUSICIANS THAT FLOATED ON A LAKE TO ENTERTAIN GUESTS AT ROYAL DRINKING PARTIES. HIS MECHANISM HAD A PROGRAMMABLE DRUM MACHINE WITH PEGS (CAMS) THAT BUMP INTO LITTLE LEVERS THAT OPERATE THE PERCUSSION. THE DRUMMER COULD BE MADE TO PLAY DIFFERENT RHYTHMS AND DIFFERENT DRUM PATTERNS BY MOVING THE PEGS TO DIFFERENT LOCATIONS. ANOTHER SOPHISTICATED PROGRAMMABLE MACHINE BY AL-JAZARI WAS THE CASTLE CLOCK.  THE JACQUARD LOOM, DEVELOPED IN 1801, IS OFTEN QUOTED AS A SOURCE OF PRIOR ART. THE MACHINE USED A SERIES OF PASTEBOARD CARDS WITH HOLES PUNCHED IN THEM. THE HOLE PATTERN REPRESENTED THE PATTERN THAT THE LOOM HAD TO FOLLOW IN WEAVING CLOTH. THE LOOM COULD PRODUCE ENTIRELY DIFFERENT WEAVES USING DIFFERENT SETS OF CARDS. THE USE OF PUNCHED CARDS WAS ALSO ADOPTED BY CHARLES BABBAGE AROUND 1830, TO CONTROL HIS ANALYTICAL ENGINE.  THIS INNOVATION WAS LATER REFINED BY HERMAN HOLLERITH WHO, IN 1896 FOUNDED THE TABULATING MACHINE COMPANY (WHICH BECAME IBM). HE INVENTED THE HOLLERITH PUNCHED CARD, THE CARD READER, AND THE KEY PUNCH MACHINE. THESE INVENTIONS WERE THE FOUNDATION OF THE MODERN INFORMATION PROCESSING INDUSTRY. THE ADDITION OF A PLUG-BOARD TO HIS 1906 TYPE I TABULATOR ALLOWED IT TO DO DIFFERENT JOBS WITHOUT HAVING TO BE REBUILT (THE FIRST STEP TOWARD PROGRAMMING). BY THE LATE 1940S THERE WERE A VARIETY OF PLUG-BOARD PROGRAMMABLE MACHINES, CALLED UNIT RECORD EQUIPMENT, TO PERFORM DATA PROCESSING TASKS (CARD READING). THE EARLY COMPUTERS WERE ALSO PROGRAMMED USING PLUG-BOARDS.  A BOX OF PUNCH CARDS WITH SEVERAL PROGRAM DECKS.  THE INVENTION OF THE VON NEUMANN ARCHITECTURE ALLOWED COMPUTER PROGRAMS TO BE STORED IN COMPUTER MEMORY. EARLY PROGRAMS HAD TO BE PAINSTAKINGLY CRAFTED USING THE INSTRUCTIONS OF THE PARTICULAR MACHINE, OFTEN IN BINARY NOTATION. EVERY MODEL OF COMPUTER WOULD BE LIKELY TO NEED DIFFERENT INSTRUCTIONS TO DO THE SAME TASK. LATER ASSEMBLY LANGUAGES WERE DEVELOPED THAT LET THE PROGRAMMER SPECIFY EACH INSTRUCTION IN A TEXT FORMAT, ENTERING ABBREVIATIONS FOR EACH OPERATION CODE INSTEAD OF A NUMBER AND SPECIFYING ADDRESSES IN SYMBOLIC FORM (E.G. ADD X, TOTAL). IN 1954 FORTRAN, THE FIRST HIGHER LEVEL PROGRAMMING LANGUAGE, WAS INVENTED. THIS ALLOWED PROGRAMMERS TO SPECIFY CALCULATIONS BY ENTERING A FORMULA DIRECTLY (E.G. Y = X\*2 + 5\*X + 9). THE PROGRAM TEXT, OR SOURCE, WAS CONVERTED INTO MACHINE INSTRUCTIONS USING A SPECIAL PROGRAM CALLED A COMPILER. MANY OTHER LANGUAGES WERE DEVELOPED, INCLUDING ONES FOR COMMERCIAL PROGRAMMING, SUCH AS COBOL. PROGRAMS WERE MOSTLY STILL ENTERED USING PUNCH CARDS OR PAPER TAPE. (SEE COMPUTER PROGRAMMING IN THE PUNCH CARD ERA). BY THE LATE 1960S, DATA STORAGE DEVICES AND COMPUTER TERMINALS BECAME INEXPENSIVE ENOUGH SO PROGRAMS COULD BE CREATED BY TYPING DIRECTLY INTO THE COMPUTERS. TEXT EDITORS WERE DEVELOPED THAT ALLOWED CHANGES AND CORRECTIONS TO BE MADE MUCH MORE EASILY THAN WITH PUNCH CARDS.  AS TIME HAS PROGRESSED, COMPUTERS HAVE MADE GIANT LEAPS IN THE AREA OF PROCESSING POWER. THIS HAS BROUGHT ABOUT NEWER PROGRAMMING LANGUAGES THAT ARE MORE ABSTRACTED FROM THE UNDERLYING HARDWARE. ALTHOUGH THESE MORE ABSTRACTED LANGUAGES REQUIRE ADDITIONAL OVERHEAD, IN MOST CASES THE HUGE INCREASE IN SPEED OF MODERN COMPUTERS HAS BROUGHT ABOUT LITTLE PERFORMANCE DECREASE COMPARED TO EARLIER COUNTERPARTS. THE BENEFITS OF THESE MORE ABSTRACTED LANGUAGES IS THAT THEY ALLOW BOTH AN EASIER LEARNING CURVE FOR PEOPLE LESS FAMILIAR WITH THE OLDER LOWER-LEVEL PROGRAMMING LANGUAGES, AND THEY ALSO ALLOW A MORE EXPERIENCED PROGRAMMER TO DEVELOP SIMPLE APPLICATIONS QUICKLY. DESPITE THESE BENEFITS, LARGE COMPLICATED PROGRAMS, AND PROGRAMS THAT ARE MORE DEPENDENT ON SPEED STILL REQUIRE THE FASTER AND RELATIVELY LOWER-LEVEL LANGUAGES WITH TODAY'S HARDWARE. (THE SAME CONCERNS WERE RAISED ABOUT THE ORIGINAL FORTRAN LANGUAGE.)  THROUGHOUT THE SECOND HALF OF THE TWENTIETH CENTURY, PROGRAMMING WAS AN ATTRACTIVE CAREER IN MOST DEVELOPED COUNTRIES. SOME FORMS OF PROGRAMMING HAVE BEEN INCREASINGLY SUBJECT TO OFFSHORE OUTSOURCING (IMPORTING SOFTWARE AND SERVICES FROM OTHER COUNTRIES, USUALLY AT A LOWER WAGE), MAKING PROGRAMMING CAREER DECISIONS IN DEVELOPED COUNTRIES MORE COMPLICATED, WHILE INCREASING ECONOMIC OPPORTUNITIES IN LESS DEVELOPED AREAS. IT IS UNCLEAR HOW FAR THIS TREND WILL CONTINUE AND HOW DEEPLY IT WILL IMPACT PROGRAMMER WAGES AND OPPORTUNITIES.    HAOFEI YAN HY222AP    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

Final note: it is actually easier to use quipqiup.com as in task 2. That also gives me the answer with no work at all.

# Task 7

1. Hash function

In this task I wrote the hash function that perform hashing on a byte array instead of a string. My hashing program takes the string, converts it into byte array and hashes it.

The hashing algorithm is as follow:

Step 1: With the initial vector is 8191, take the vector, XOR with its value shifted left two bits, XOR with its value shifted right 5 bits and XOR the first bit of the message and obtain the first hash value.

Step 2: repeat the whole process but the initial vector is now the previous hash value, repeat until the end of the message, obtaining the current value

Step 3: repeat step 1 but now the vector is the value from step 2.

Step 4: terminate if the whole procedure has been repeated 5 times, return the hash value AND 0xFFFF (16 bits).

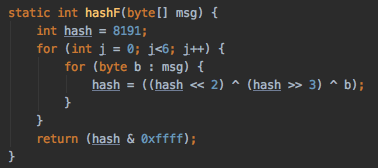


Figure 1 My hash function

b) Analyzing of my hash function

Figure 2 Hello World hash

1. For not so

# Bibliography

|  |  |
| --- | --- |
| [1] | C. P. Pfleeger, Security in Computing. |
| [2] | J. Stanley. |
| [3] | E. Olson. |
| [4] | M. Herlitzius. |
| [5] | W. Shakespeare. |
| [6] | H. Yan. |
| [7] | dCode. |

Bibliography

[1]

[2]

[3]