2024年臺灣國際科學展覽會
優勝作品專輯

作品編號　190037

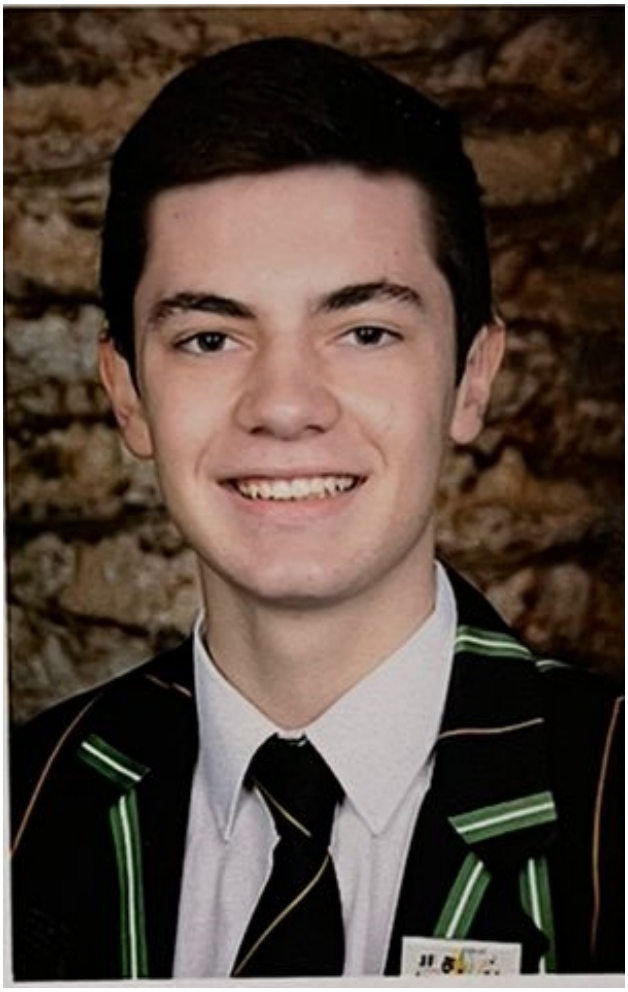
參展科別　電腦科學與資訊工程

作品名稱　Breaking a Caesar Cipher / Vigenère Cipher Encryption for Secure Data Communication

得獎獎項　四等獎

就讀學校　High School Jim Fouche
指導教師　Dibata Johannes Lephondo
作者姓名　Johannes Jacobus Deysel

作者照片



Abstract

This project had one purpose: creating almost unbreakable encryption by breaking a Caesar – and Vigenère Cipher and getting familiar with how they work. Created a program to encrypt and decrypt messages with a Caesar Cipher and Vigenère Cipher encryption. Breaking these encryptions in these programs will help to identify the factors that contribute to strong and weak encryption systems.

A program was created to encrypt messages using Caesar Cipher with a key from 1 to 25 and decrypt messages without knowing the original key by doing different types of “attacks” on the system: a brute force and frequency analysis attack. Created another program to encrypt messages using Vigenère Cipher with a keyword or keyphrase and decrypted messages whilst knowing that original keyword. Tested and compared the two different cyphers when being attacked. This helped identify factors that influenced the strength of encryption and identified the advantages and disadvantages of each Cipher as well as the weaknesses in each attack.

Through testing and breaking a Caesar and Vigenère Cipher successfully, multiple factors were identified that influenced the strength of the encryption system. These were used to ensure the new encryption created will be as strong as can be. Comparing the success rate of the different attacks on each Cipher, the similarities, weaknesses and strengths in the Brute Force and Frequency Analysis attacks were found.

Introduction

Literature Review

("Biggest Data Breaches in US History", 2023)

The software company, UpGuard, which uses cybersecurity risk management software to help organizations across the globe prevent data breaches, says that the biggest data breaches in US history happened between 2019 and 2021 and included some of the biggest companies like Microsoft, Facebook, LinkedIn, etc.

These colossal data breaches had several negative consequences. Identity theft was one of the biggest consequences where hackers gained that sensitive information and used it to commit fraudulent activities, such as opening new accounts or making unauthorized purchases. The answer to reducing this large amount of data breaches is to strengthen modern encryptions. Companies must ensure that their encryption techniques evolve and strengthen at the same pace as the fast evolution of technology, especially AI. The gap within the majority of companies' encryption is that it's not evolving fast enough to withstand hackers and AI.

Problem Statement

Technology is evolving rapidly; thus, hackers are also evolving their methods to decrypt these systems. To ensure the safety of users' data online, there is a constant need to evaluate what hackers know and to strengthen your encryption.

The first step to beat a hacker is for companies to think like hackers to protect themselves. The problem is that hackers are catching up and there is a need for new, different, and innovative encryptions.

Aim

The aim of this project is to create an encryption by breaking a Caesar – and Vigenère Cipher by creating a program to encrypt messages with a Caesar Cipher and Vigenère Cipher encryption and then break them without the original key.

Engineering goals or Design goals

Develop a computer program that would be able to encrypt messages in Caesar Cipher and break those encrypted messages with two different methods. *("Crack the Code: Breaking a Caesar Cipher", 2020)*

Create encryption and decryption process called Vigenère Cipher to compare to the Caesar Cipher encryption get familiar with different encryption techniques and identify advantages or similarities.

Method

Materials

- Windows operated computer
- Application: Delphi version 11 was used as the programming language

Procedure and Developing

Planning

An IPO table was used to plan this. An IPO table lists the input that the program needed to get from the user, the processing that had to be done to that input, and then what information had to be put out to the user.

- Table 1: The encryption section

Input	Process	Output
• Plaintext to encrypt • A key to encrypt in	• Shift each of the characters in the plaintext provided an X number of times *forward* in the alphabet where X = the user's key.	• Give feedback to the user and output the encrypted message with the characters already shifted.

- Table 2.1: The decryption section (Brute force)

Input	Process	Output
• An already encrypted message	• Decrypt the given message by shifting each of the characters X times *backward* in the alphabet • No provided key. Thus try all 25 possible keys (1 to 25)	• Output the key used with its corresponding decrypted output message. (There will be 25 outputs, one for every key).

- Table 2.2: The decryption section (Data analysis)

Input	Process	Output
• An already encrypted message (original English)	• Analyze the given encrypted message by counting how many times a character occurs. • Get the characters occurring the most in the message and use it as a safe guess to represent the character "e". • Use that safe guess as the key used to get an output.	• Output the key guessed with its corresponding decrypted message, which if correct, should be English.

Method: 5 Steps

1. ***Identified the existing encryption techniques***
   - Caesar Cipher
   - Vigenère Cipher

2. ***Created a program with Caesar Cipher***
   - Encrypted message with a key from 1 to 25
   - Decrypted message without knowing the key

- Brute Force
- Frequency Analysis

3. ***Created a program with Vigenère Cipher***
   - Encrypted a message with a keyword
   - Decrypted a message whilst knowing the original keyword

4. ***Tested and compared the two***
   - Identifed factors that influence the strength of an encryption
   - Identifed the advantages and disadvantages of Caesar – and Vigenère Cipher

5. ***Creted own encryption with a twist***

Pre-knowledge to encrypt or decrypt

- Encryption

The ASCII (American Standard Code for Information Interchange) table (Table 1) assigns standard numeric values to letters, numerals, punctuation marks, and other characters used in computers.

Every character and alphabet letter has a corresponding value on the ASCII table:

Table 1: The ASCII table
Source: (Adapted from Usha, 2010)

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

The ASCII table is case-sensitive; thus, input messages need to be immediately converted to uppercase before starting processing.

Every character in the input message was converted into its value in the ASCII table. The uppercase characters of the alphabet were all increased with a difference of 1, the key was added to the ASCII value. This shifts each character forward, in the alphabet, the number of times as the key.

Problem: However, it is important to note that when the value is added to the ASCII value of a character at the end of the alphabet, like “Z”, special characters are the output, like “[“.

In that case, the program was coded to loop back by using “mod 26” which limits the characters to only the letters in the alphabet.

The new values were then reconverted back to alphabet characters and displayed as the new encrypted message.

- Decryption

The program was decrypted by an already encrypted message with the button above.

When breaking into the encryption, there are two ways in which the code would be cracked without knowing the original key:

- A brute force attack
- Using frequency analyzation
- In the result section of my report I will be discussing which one is more effective aswell as each one's advantages and disadvantages.

Prototype 1:

- To encrypt the message the user inputs, the following was done:
  ("*DELPHI: Encryption examples", 2008)*

1. Split the plaintext message into individual characters.
2. Convert the characters into their corresponding value in the ASCII table with the function Ord(x).
3. Add the key to the ASCII value to get a new value.
4. Convert the new value to a character using the Chr(x) function.
5. Repeat this for each character in the message, get the number of characters by using the length(x) function.
6. Do the repeating using a loop.

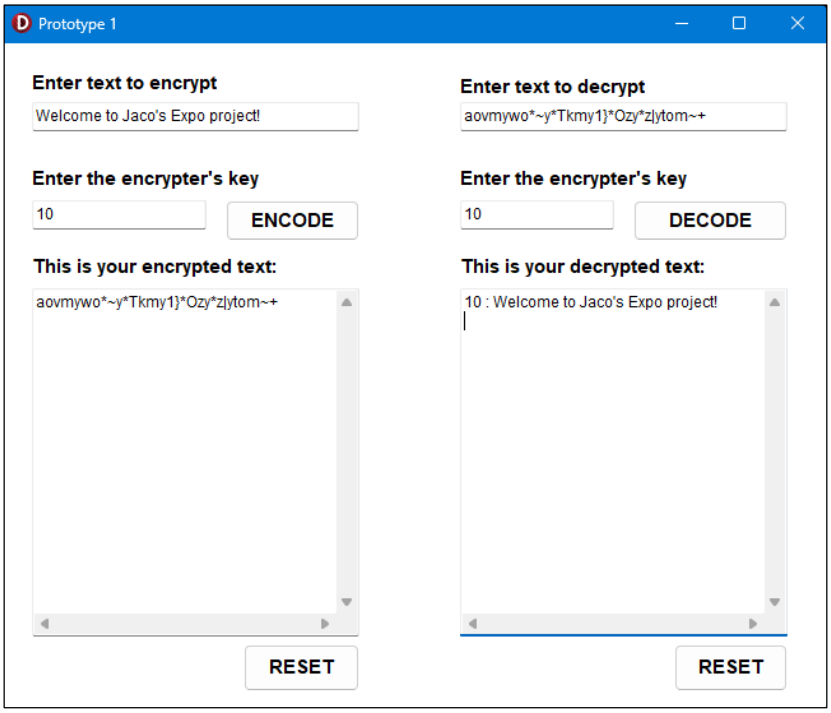


Figure 1: Testing prototype 1
Source: Screenshot of my own program

- When tested, the following problems with this prototype were identified:
  - This project was inspired by a Caesar Cipher encryption (which only uses alphabet letters) and to prove its weakness. This prototype is not limited to alphabet letters and includes special characters (See Figure 2: The encrypted message output).

Prototype 2:

- To limit the encryption to alphabet letters only, the following was done:
  1. Created an array,
  2. Populated the array with the ASCII values of the alphabet characters from A to Z. Thus, there were 26 slots with values 65 to 90.
  3. Added the key to each slot in the array using a loop. A new alphabet was created using a unique key.
  4. Replaced the characters in the input message with the new corresponding characters in the new alphabet.
     Ex. If the message is "E", the letter was replaced with the 5th character in the array

The output is now limited to only letters in the alphabet. (See Figure 3: The encrypted message output).

- To decrypt an already encrypted message:
  1. Split the encrypted message into its individual characters.
  2. Convert the characters into their corresponding value in the ASCII table with the function Ord(x).
  3. Subtract the key from the ASCII value to get a new value.
  4. Convert the new value back to a character using the Chr(x) function.
  5. Repeat this for each character in the message, get the number of characters by using the length(x) function.
  6. Do the repeating using a loop.

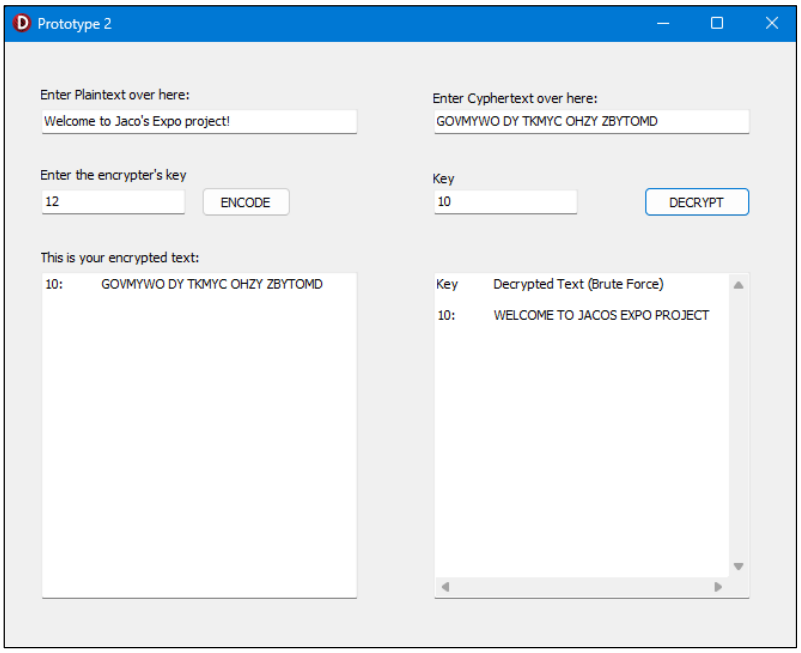


Figure 2: Prototype 2
Source: Screenshot of my own program.

- Problems with this prototype
  - In this decryption we need to enter a key, but in reality, we do not know what key the original message was encrypted in. Thus, the original key is unknown to us and this prototype relies on knowing the key.
  - Because the original key is unknown to us, this decryption requires you to manually enter all keys from 1 to 25 to get all the possible outputs which takes time.
  - This prototype only has one way of decrypting the message (a brute force technique), which requires you to distinguish between 25 outputs which one looks acceptable which could be a problem if the original message is not English or another identifiable language.

Prototype 3:

- Introduction to the "brute force" technique:

("Classical-ciphers-frequency-analysis-examples", 2015)

A brute force attack tries every key from 1 to 25 and outputs each's corresponding decrypted message. When done, 25 different decrypted messages are left and thus it is required from the user to manually distinguish between the 25 messages and decide which one looks like the acceptable output.

To distinguish between messages is easy when we assume the original message was English (or another identifiable language). But if the original message were a randomly generated password, it would become impossible to identify the original message from the other 24, because all of them would look randomly generated.

To automate the brute force technique, all of this needs to be done in a single button (no key input needed):

1. Decrypt the message with a key of 1. Output it.
2. Decrypt the message with a key of 2. Output it.
3. Use a loop to repeat this decryption until you reach the key 25. Output every key's decrypted message.
4. 25 Different output messages will be left at the end (See Figure 3)

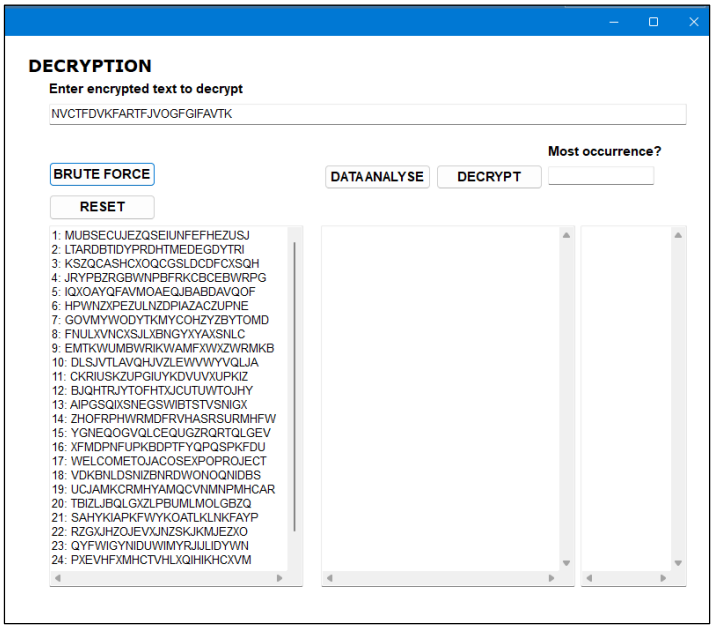


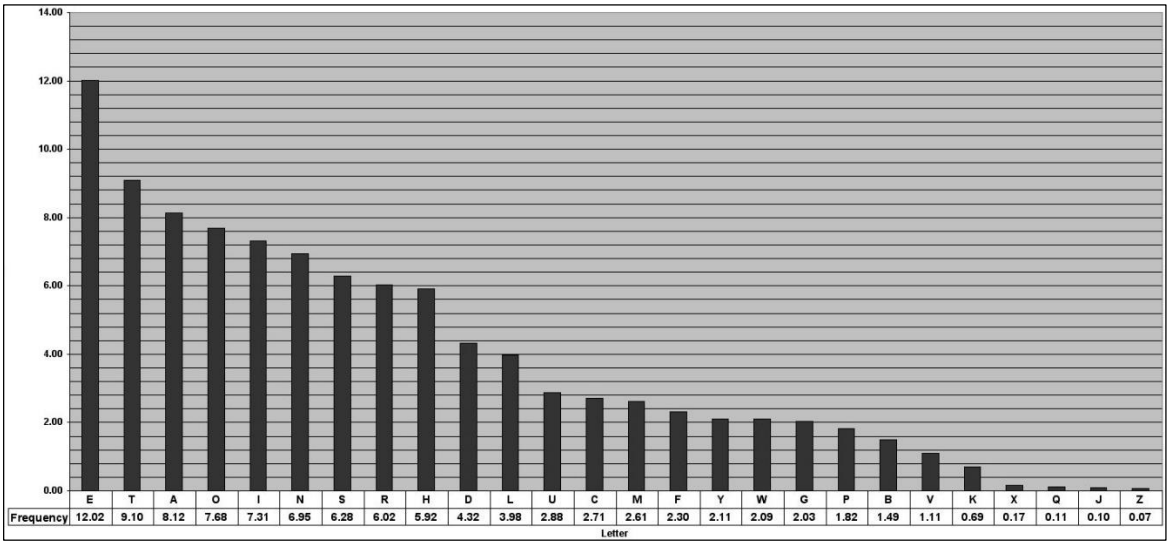
Figure 3: Prototype 3's Brute force

Source: Screenshot of my own program.

- Introduction to the “frequency analyzation” technique:

(“*Frequency Analysis: Breaking the Code”, 2017)*

To add another way of decrypting a message without a key, a new technique is introduced called frequency analysis. This analysis the number of times a single character occurs in a message. For example, the character “e” appears way more in English words than the character “Z” or “X”. In the English language the letter “E” is the letter that appears the most on average (see Figure 4).



	E	T	A	O	I	N	S	R	H	D	L	U	C	M	F	Y	W	G	P	B	V	K	X	Q	J	Z
Frequency	12.02	9.10	8.12	7.68	7.31	6.95	6.28	6.02	5.92	4.32	3.98	2.88	2.71	2.61	2.30	2.11	2.09	2.03	1.82	1.49	1.11	0.69	0.17	0.11	0.10	0.07

Table 2: Letter frequency in the English language

Source: Adapted from MEC and Cornell University, 2004

Assuming the original message was in English:

1. Use the built-in position function and a counting variable (which increases every time the position of a letter is found) along with a loop to continue finding the position of a given letter.
2. Repeat this process 26 times with another loop for all 26 letters of the alphabet.
3. When the frequency analyzation button is pressed, a list of the 26 letters of the alphabet should be displayed with their corresponding number of occurrences in the entered message. (See Figure 4)
4. Analyse that list to identify the most occurring letter.

When that list is studied, it would be clear which letter occurs the most in the message. It would be a safe assumption that the letter occurring the most in the encrypted message represents the letter “E” in the original message.

To perform this assumption, the program requires a manual input on which letter occurs most after the list is studied:

Assuming the original message was English:

1. Convert the letter occurring the most into its ASCII value using the built-in ord(x) function.
2. Subtract that value from the value 69, which is the ASCII value for the letter “E”.
3. Take the absolute value of that answer (remove the negative sign).
4. That answer will be the first guess to which key the original message was encrypted with.
5. Use that key to do the normal decryption. (See Figure 5)

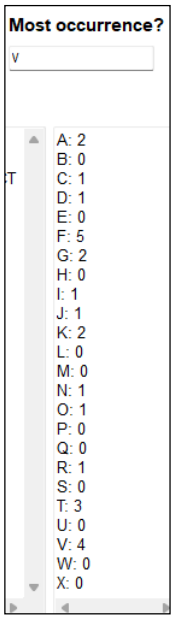


Figure 4: Character frequency list
Source: Screenshot of my own program

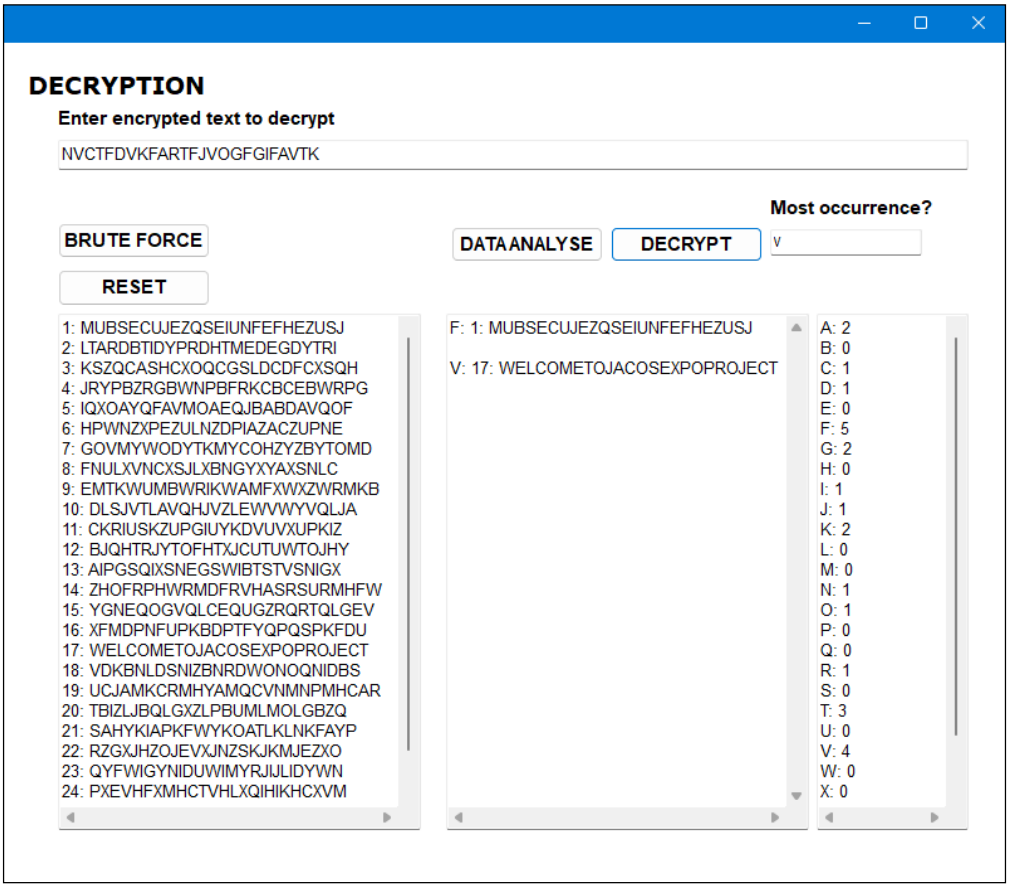


Figure 5: Prototype 3’s Frequency Analysis
Source: Screenshot of my own program.

- Problems with frequency analysis:
  - In some cases, the first try does not work and gives an incorrect decryption (See Figure 5). This is because the letter occurring the most in the original message was NOT the letter “E”, and thus not an average message. This process would have to be repeated but this time with the character occurring the second most. However, this is still more

effective than the brute force method because it significantly reduces the amount of output messages you have to distinguish from.
  - In the case that the original message is not English, this method will most likely not work. The character occurring the most differs from language to language.
  - In the case that the original message is randomly generated like a password, frequency analysis would be absolutely useless because analysing the frequency of something random would also just give random results and not something useful.
  - There are other substitution ciphers that are designed to defeat frequency analysis.

- Problems with this prototype

  - Frequency analysis requires the user to sift through a list of each character's number of occurrences and pick the one occurring the most for the program to guess a key and decrypt the message according to that key.
  - If it is the case that the first decryption of frequency analysis does not work, the user again needs to sift through the list of number of characters occurrences to choose the character occurring the second most. This slows down the speed of which a message can be cracked with frequency analysis and is very tedious.
  - Frequency analysis is divided into 2 separate buttons/processes. (See Figure 5)

Prototype 1 and 2 thus had a great significance because prototype 1 helped identify problem areas and in prototype 2 those problems could be approached differently and be fixed in prototype 3.

Prototype 4

- To automate/fasten the frequency analysis process:

For faster or automatic decryption with frequency analysis, the two different frequency analysis buttons on prototype 3 needs to be combined into one single process or button. This can be done by coding the program to automatically sift through the list we generated for the most occurring character, and in the case of that not being the correct encryption, also be able to identify the second most occurring character.

Assuming the original message was English:

1. Create 2 arrays with 26 slots each, one for unsorted data and one for sorted data.
2. Populate both arrays with the list of character occurrences frequencies alphabetically.

3. Using 2 loops, an inner and an outer loop, sort one of the arrays in descending order.
4. The value in the first slot in the sorted array will now be the greatest number of times a character occurs.
5. Compare that value with the first value in the unsorted array.
6. If they are the same, it is known that “A” is a letter occurring the most because the unsorted array is in alphabetical order. So, if they are the same, take the chr(x) of the slot number + 64 and store that character.
7. It is possible for there to be more than one character occurring the most in a message, thus this process of comparing and storing values between arrays needs to be repeated 26 times for all the number of slots by using a loop.
8. At the end one or more characters are left that can be used to create a key guess for the decryption. (See Figure 6)

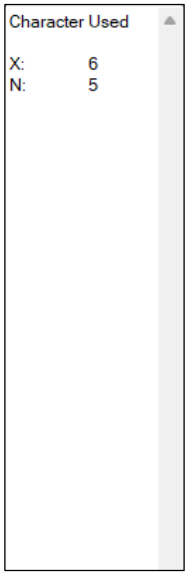


Figure 6: Character frequency list
Source: Screenshot of my own program

- In the case of the first decryption not being correct (Figure 7)

The button will be pressed a second time and have to automatically identify and use the character occurring the second most like shown below:

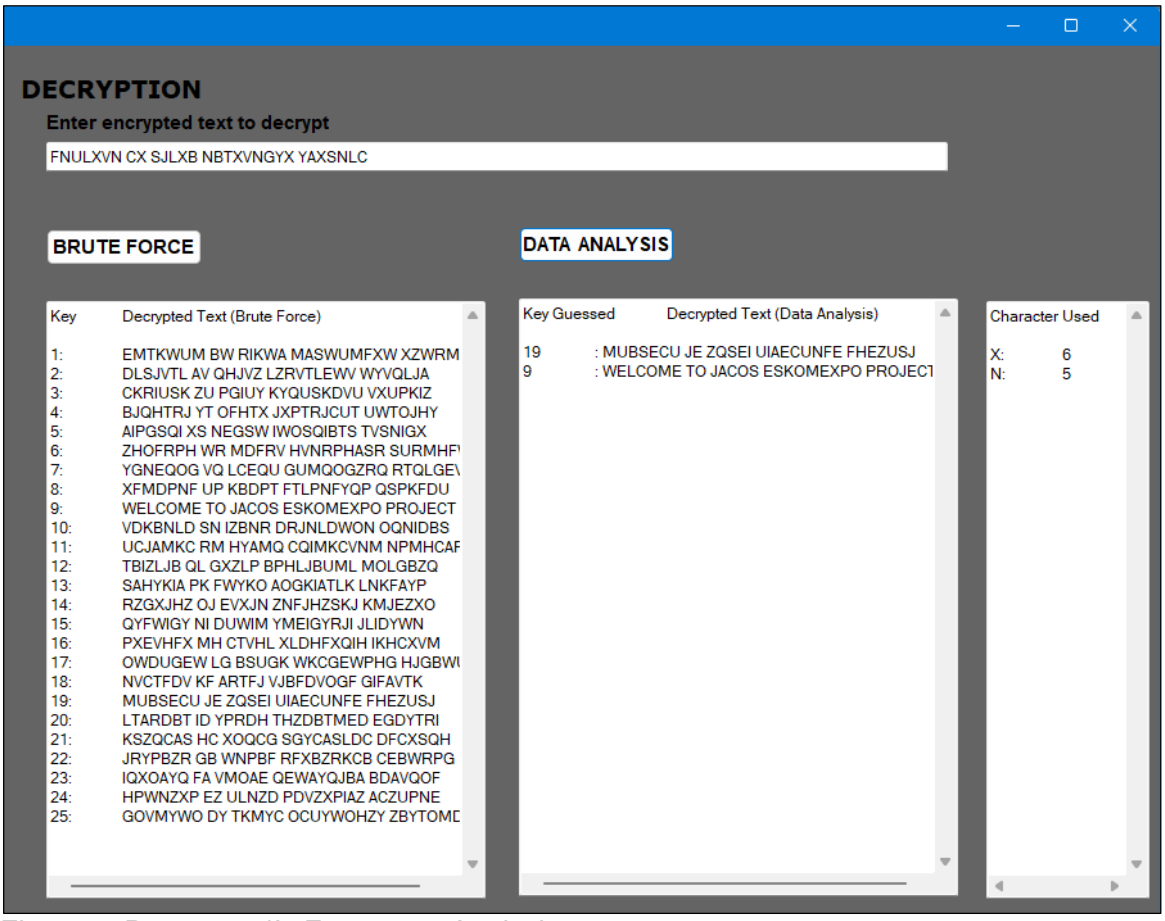


Figure 7: Prototype 4’s Frequency Analysis
Source: Screenshot of my own program.

Do the exact same process as in the previous steps, but replace step 4:

4. The value in the (first + 1) slot in the sorted array will now be the greatest number of times a character occurs.

If the second time does not work, replace step 4 again, but plus 2:

4. The value in the (first + 2) slot in the sorted array will now be the greatest number of times a character occurs.

Repeat these steps until the decryption is correct. (See Figure 7)

- Introduction to the “Random” button

This button generates a random 20-character long message. This feature will be used during testing to test the program’s encryption and decryption on a randomly generated message.

Final program (Prototype 5)

Another encryption technique was added to the existing program to provide more results to be able to compare these two techniques.

- Introduction to the Vigenère cipher

(“Vigenère cipher”, 2023)

The Vigenère cipher is a technique of encrypting alphabetic plaintext where each letter of the plaintext is encoded with a different Caesar cipher, whose key is determined by the corresponding letter of another text, called the keyword.

For example, if the plaintext is “attacking tonight” and the key is OCULORHINOLARINGOLOGY, then

- the first letter a of the plaintext is shifted by 14 positions in the alphabet (because the first letter O of the key is the 14th letter of the alphabet, counting from 0), yielding o;
- the second letter t is shifted by 2 (because the second letter C of the key means 2) yielding v;
- the third letter t is shifted by 20 (U) yielding n, with wrap-around;

and so on; yielding the message “ovnlqbpvt eoeqtnh”. If the recipient of the message knows the key, they can recover the plaintext by reversing this process.

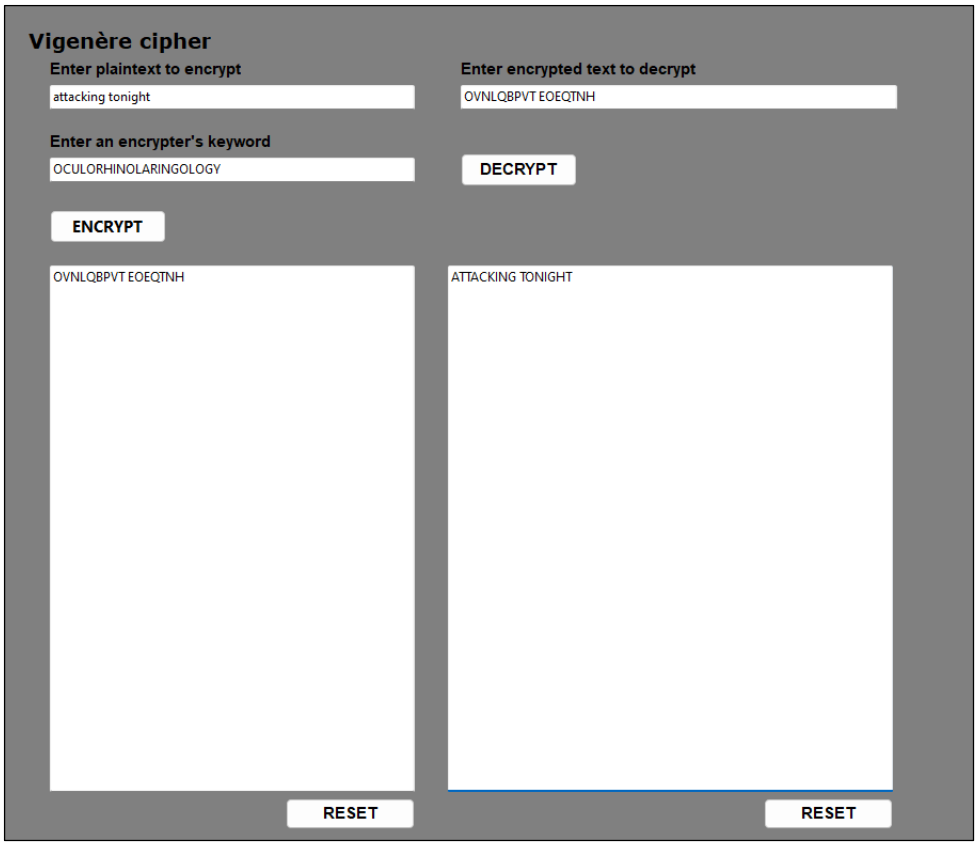


Figure 8: Final Program's Frequency Analysis
Source: Screenshot of my own program.

- What if the plaintext is larger than the keyword?

("VIGENERE CIPHER", 2023)

Because the encryption of the plaintext relies on the corresponding character of the keyword to provide a key, it would become a problem if the keyword is shorter than the plaintext. In that case, some letters would not have any corresponding character and thus no key.

To fix this, simply repeat the keyword with a loop until the new string is larger than the plaintext.

For example, if the keyword for the above plaintext were "KEYS":

- The plaintext is 17 characters long and the keyword 4
- Repeat the keyword 5 times, "KEYSKEYSKEYSKEYSKEYS"
- Now the key is 20 characters long and can be used

- Vigenère cipher vs. Frequency analysis

("Cryptanalysis", 2023)

The Vigenère cipher is unsuccessful to the decryption method of frequency analysis due to the fact that the cipher rotates through different shifts, so the same plaintext letter will not always be encrypted to the same ciphertext letter.

For example, let's say that "e" is the most common letter in English words. A decrypter using frequency analysis may think that the most common letter in an encoded message likely corresponds to "e". However, since a Vigenère cipher encodes the same letter in different ways, depending on the keyword, "e" could be encoded as many different letters, thus breaking the assumptions behind frequency analysis.

- Vigenère cipher vs. Brute force

(“Cryptanalysis”, 2023)

A Vigenère cipher is difficult to crack using brute-force because each letter in a message could be encoded as any of the 26 letters. Because the encoding of the message depends on the keyword used, a given message could be encoded in 26 to the power of *k* ways, where k is the length of the keyword.

For example, if we only know that a message is encoded with a word with a length of 7 letters, then it could be encoded in 8 billion ways!

- The vulnerability of Vigenère cipher

(“Cryptanalysis”, 2023)

The primary weakness of the Vigenère cipher is the repeating nature of its key. If a cryptanalyst correctly guesses the length of the key, then the ciphertext can be treated as interwoven Caesar ciphers, which, individually, can be easily broken.

Keyword: KIN GKI NGK ING KIN GK ING KING
Plaintext: THE SUN AND THE MAN IN THE MOON
Ciphertext: DPR YEV NTN BUK WIA OX BUK WWBT

Keyword: KIN GKI NGK ING KIN GK ING KING
Ciphertext: DPR YEV NTN BUK WIA OX BUK WWBT
Plaintext: THE SUN AND THE MAN IN THE MOON

For example, in the cryptogram above, the plaintext “THE” occurs twice in the message, and in both cases, it lines up perfectly with the first two letters of the keyword. Because of this, it produces the same ciphertext “BUK”.

Repetitions in the ciphertext indicate repetitions in the plaintext, and the space between such repetitions hint at the length of the keyword.

In fact, any message encrypted with a Vigènere cipher will produce many such repeated instances. Although not every repeated instance will be the result of the encryption of the same plaintext, many will be and this provides the basis for breaking the cipher. This method of analysis is called Kasiski examination.

- Introducing the “Jaco Cipher”: Encryption Steps

The following steps show the encryption process of the final encryption program.

1. Key: Keyword(characters) + Key (Integers) ex. UJGSTFMSNG23
2. Caesar Output = Message is encrypted with Caesar Cipher and key part
3. Vigenère Output = Message is encrypted with Vigenère Cipher and keyword
4. The Caesar Output is encrypted with Vigenère Output as the keyword for X number of times.
5. The twist: The number of times/layers encrypted stated above will be determined by the length of your name.
6. That output is then encrypted with the key but including special characters.

Breaking different types of messages

Testing was done by encoding different types of messages, then decoding them and see how they reacted to the process. This will give an idea to what influences the successfulness of an encryption.

English

- ✓ **Encrypted** an English type message successfully
- ✓ ***Brute force*** decrypted the encryption successfully at key 12
- ✓ ***Frequency analysis*** successfully decrypted the message with key 12 with 2 tries

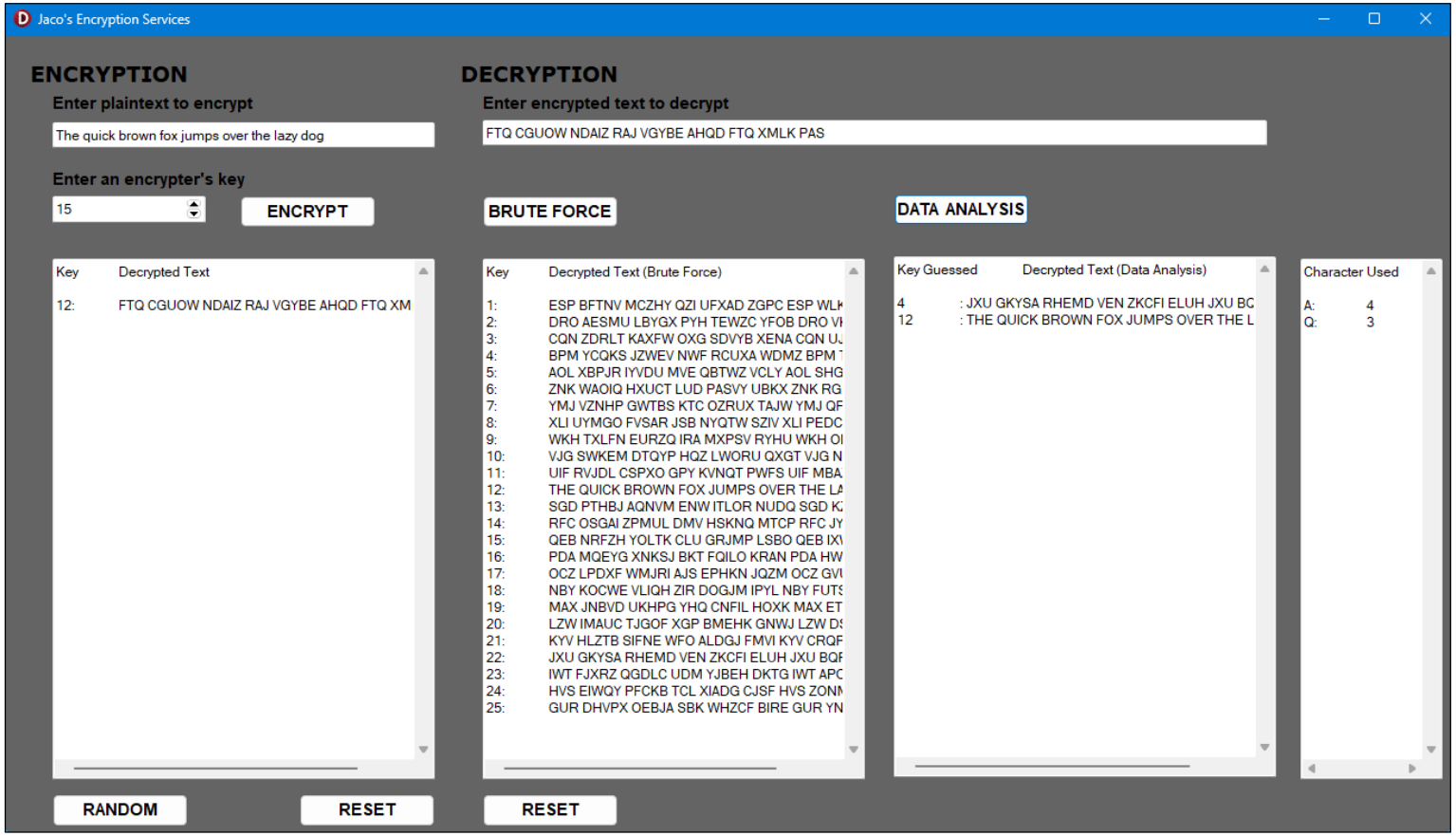


Figure 1: Testing an English message

Source: Screenshot of my own program.

Afrikaans

- ✓ **Encrypted** an Afrikaans type message successfully
- ✓ ***Brute force*** decrypted the encryption successfully at key 8
- ✓ ***Frequency analysis*** successfully decrypted the message with key 8 with 3 tries

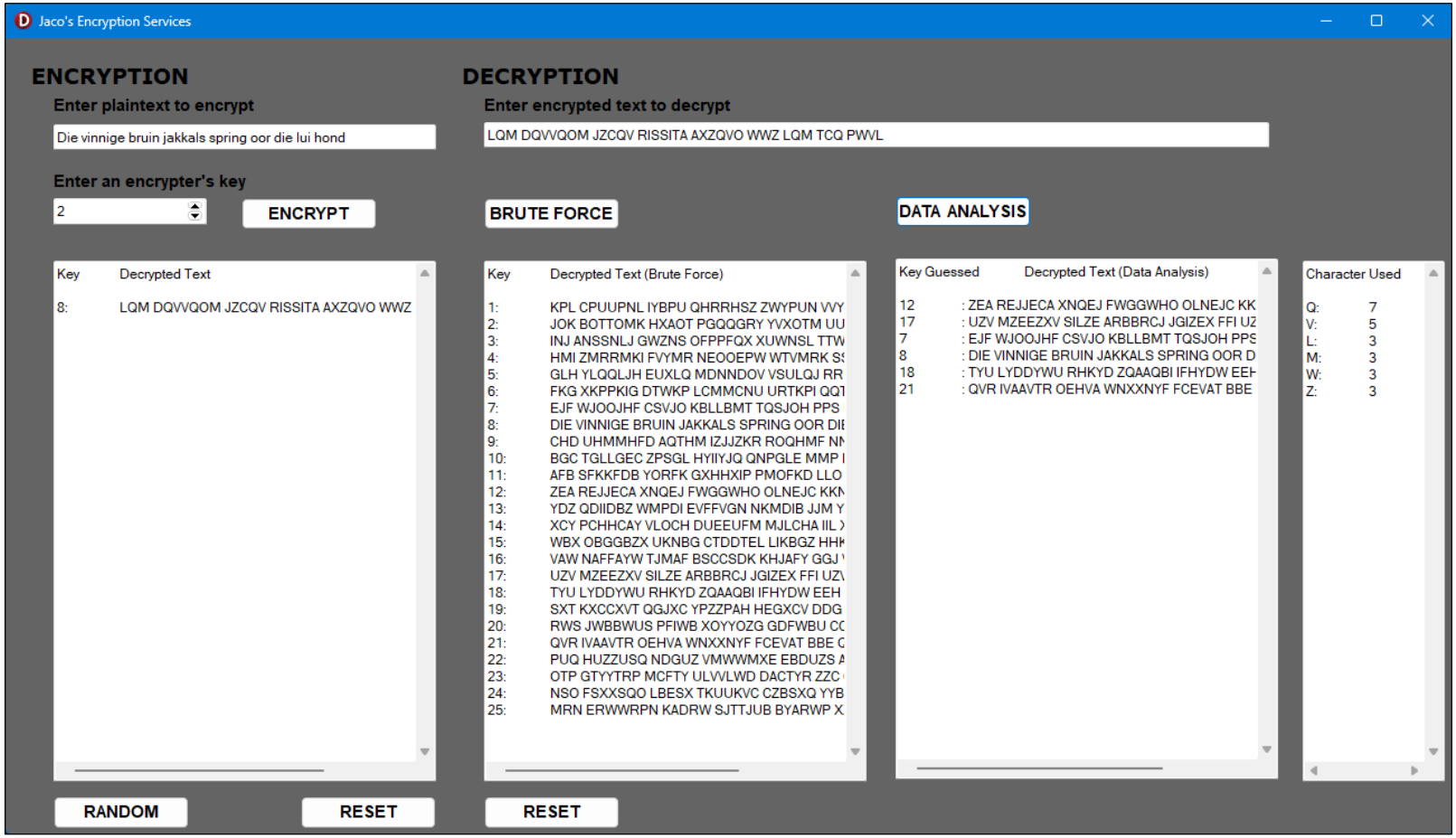


Figure 2: Testing an Afrikaans message

Source: Screenshot of my own program.

Password

- ✓ **Encrypted** a Password type message unsuccessfully (Capitalized every letter + left out digits)
- ✓ ***Brute force*** decrypted the encryption unsuccessfully
- ✓ **Frequency analysis** unsuccessfully decrypted the message

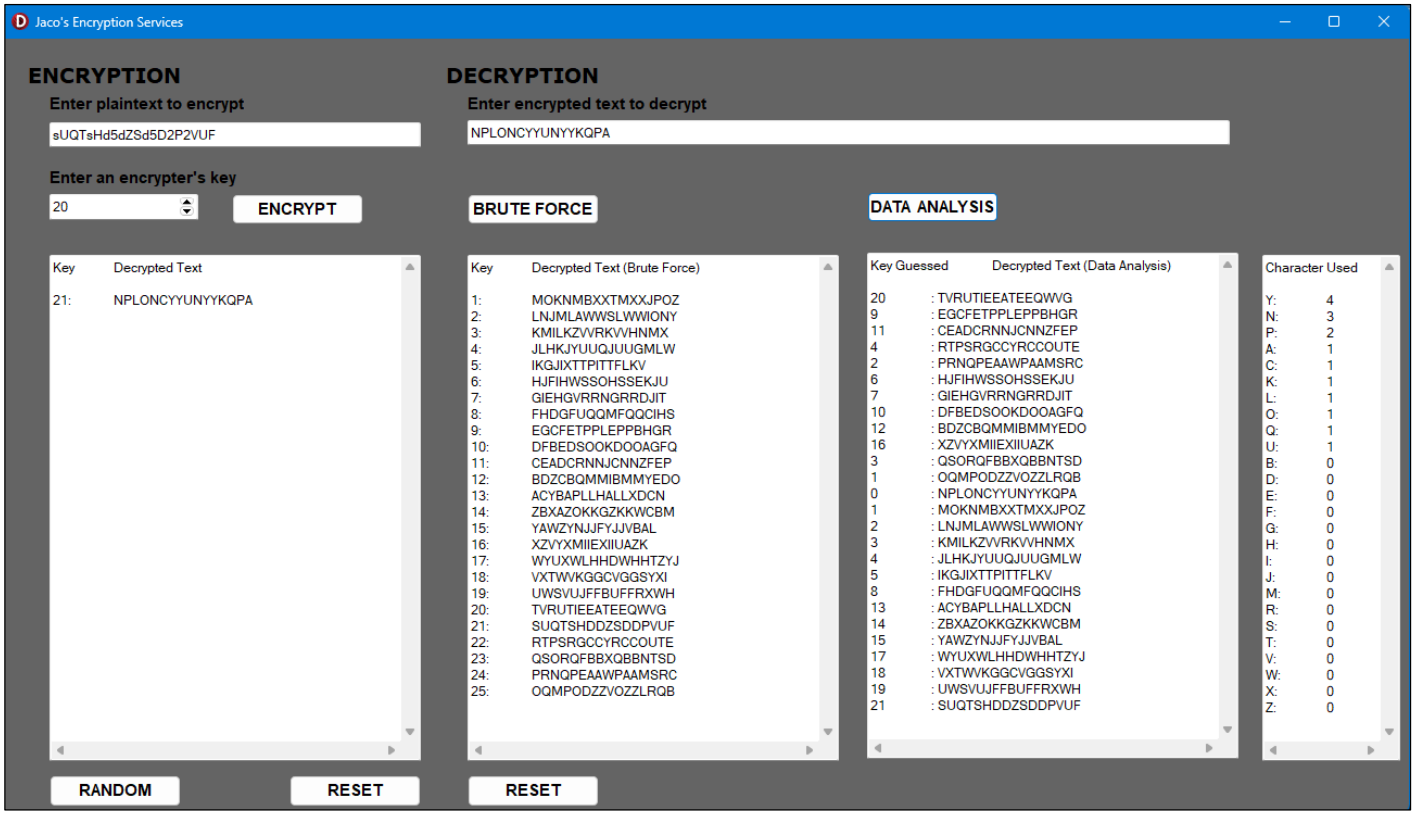


Figure 3: Testing a Password message
Source: Screenshot of my own program.

Randomly generated message (Password without digits)

- ✓ **Encrypted** a Randomly generated type message successfully
- ✓ ***Brute force*** decrypted the encryption unsuccessfully (impossible to distinguish)
- ✓ **Frequency analysis** unsuccessfully decrypted the message (impossible to distinguish)

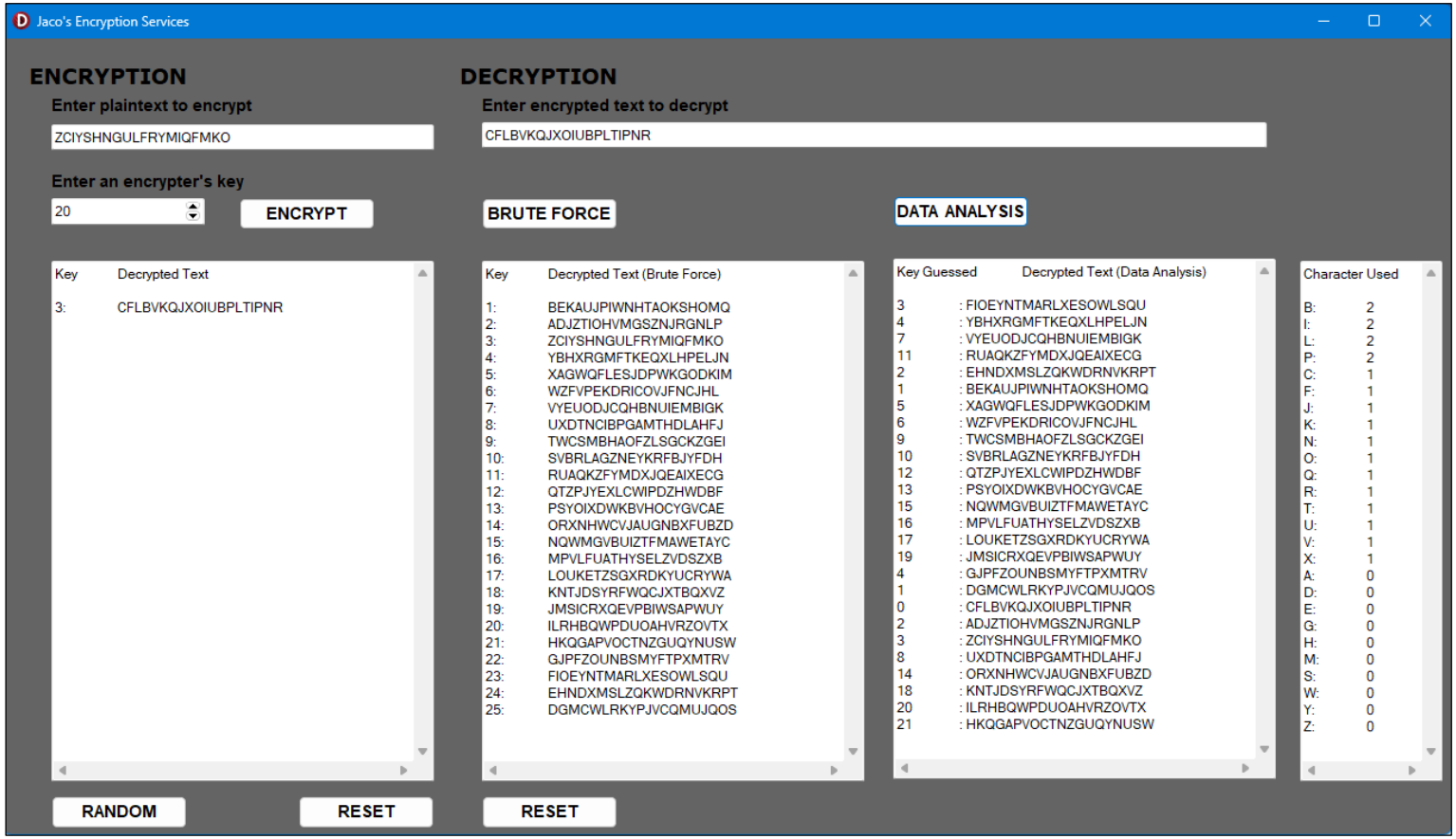


Figure 4: Testing a Randomly Generated message

Source: Screenshot of my own program.

Sentence with lots of e's

- ✓ **Encrypted** a sentence with lots of e's successfully
- ✓ ***Brute force*** decrypted the encryption successfully
- ✓ **Frequency analysis** successfully decrypted the message within the first try

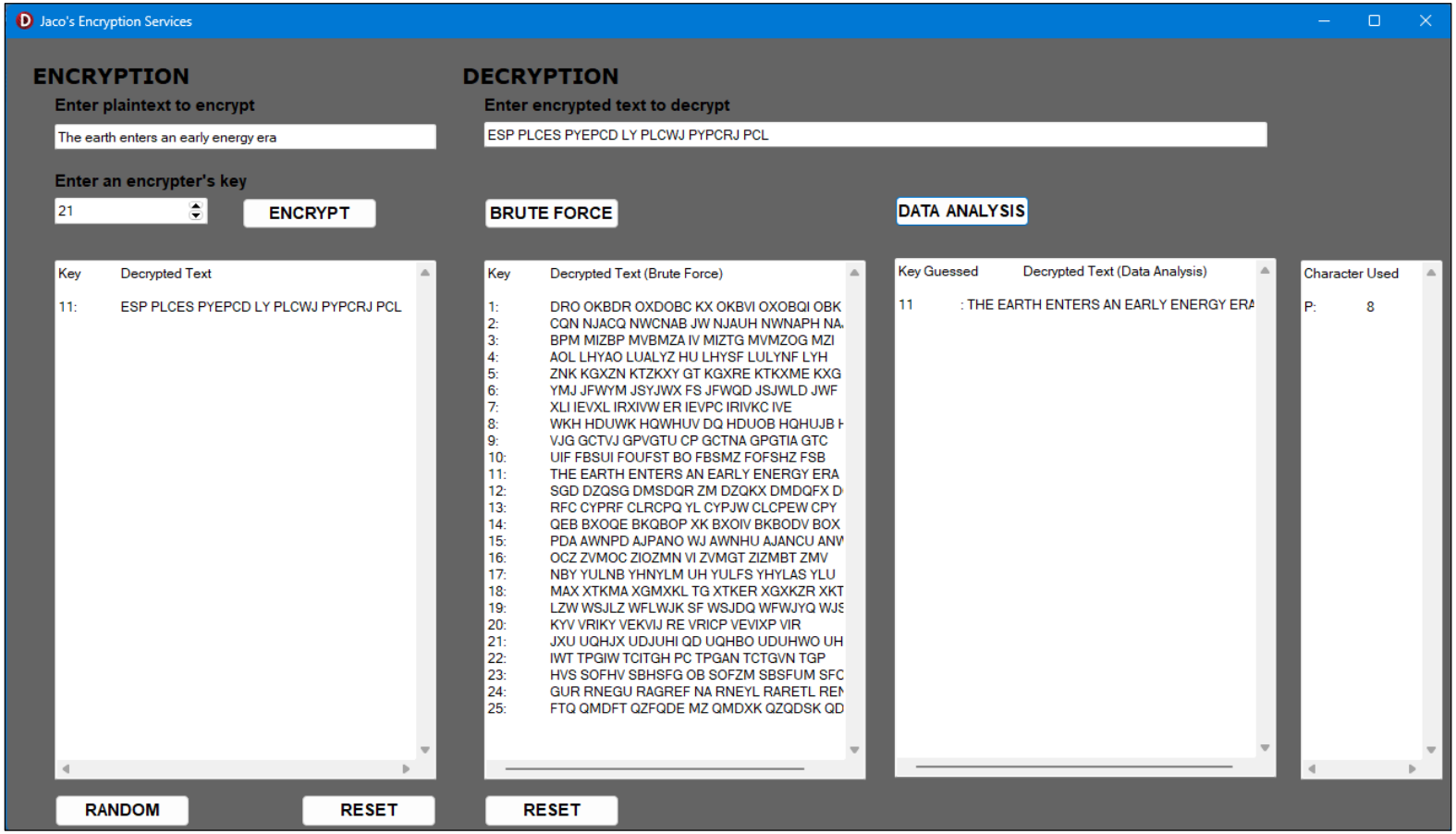


Figure 5: Testing a sentence with lots of e's

Source: Screenshot of my own program.

Sentence with no e's

- ✓ **Encrypted** a sentence with none e's successfully
- ✓ ***Brute force*** decrypted the encryption successfully
- ✓ **Frequency analysis** unsuccessfully decrypted the message even with all tries

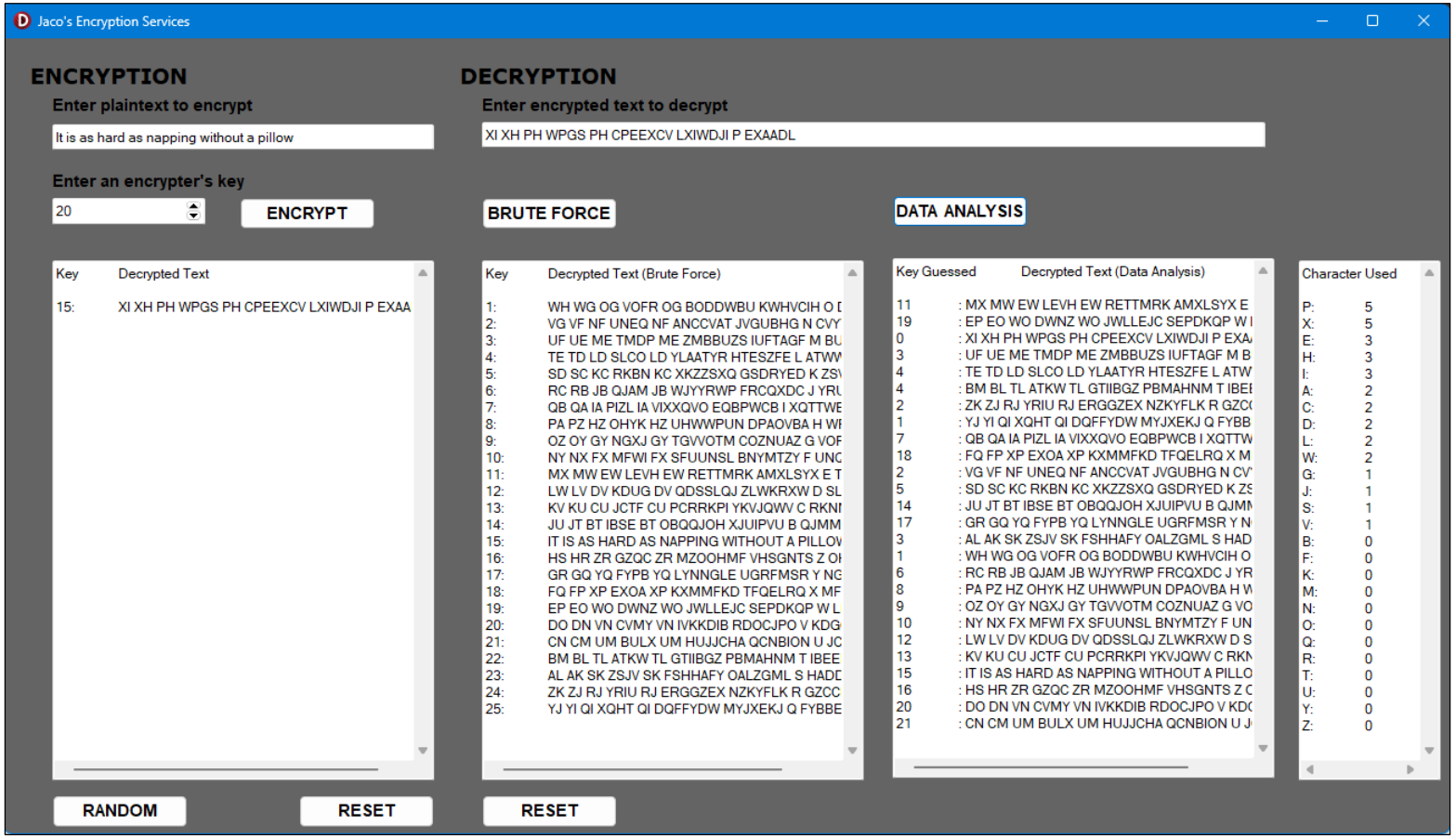


Figure 6: Testing a sentence with no e's

Source: Screenshot of my own program.

Results

The result of this program is that it was in fact able to break a Caesar Cipher encryption which aligns with the engineering and designing goals. There are also other results gathered.

Different types of messages

- **English:**
  - ✓ *Encrypted* an English type message successfully
  - ✓ *Brute force* decrypted the encryption successfully at key 12
  - ✓ *Frequency analysis* successfully decrypted the message with key 12 with 2 tries

- **Afrikaans:**
  - ✓ *Encrypted* an Afrikaans type message successfully
  - ✓ *Brute force* decrypted the encryption successfully at key 8
  - ✓ *Frequency analysis* successfully decrypted the message with key 8 with 3 tries

- **Passwords:**
  - ✓ *Encrypted* a Password type message unsuccessfully (Capitalized every letter + left out digits)
  - ✓ *Brute force* decrypted the encryption unsuccessfully
  - ✓ *Frequency analysis* unsuccessfully decrypted the message

- **Randomly generated message:**
  - ✓ *Encrypted* a Randomly generated type message successfully
  - ✓ *Brute force* decrypted the encryption unsuccessfully (impossible to distinguish)
  - ✓ *Frequency analysis* unsuccessfully decrypted the message (impossible to distinguish)

- **Sentence with lots of e's:**
  - ✓ *Encrypted* a sentence with lots of e's successfully
  - ✓ *Brute force* decrypted the encryption successfully
  - ✓ *Frequency analysis* successfully decrypted the message within the first try

- **Sentence with no e's:**
  - ✓ *Encrypted* a sentence with none e's successfully
  - ✓ *Brute force* decrypted the encryption successfully
  - ✓ *Frequency analysis* unsuccessfully decrypted the message even with all tries

Factors contributing to the strength of an encryption identified

- Size of the key
- Special characters
- Small and capital letters
- Encryption technique
- Length of encryption
- Language of message
- Layering

Discussion

There are loads of websites online encrypting a message with a Caesar Cipher, but none decrypt them using multiple types of decrypting types. Thus, the one created was able to gather information showing what variables to consider or improve when wanting to create a strong encryption for websites, messaging or other platforms.

The factors that contribute to a weak encryption

Factors Identified	Strong Encryption	Weak Encryption
Size of key	Key is not very limited (up to 1000+ options)	Key is limited (like my program with only 25 options)
Special characters	Special or unusual characters are allowed	Only alphabet characters are allowed
Small and capital letters	Encryption is case sensitive - small and capital letters have different corresponding ASCII values	Encryption is not case sensitive - characters are all capital or all small
Encryption technique	Techniques gets updated and modified ex. Hashing	Techniques are old and out of date like old Ciphers
Length of encryption	Length can be changed and set to whatever length you want - making more possibilities (Hashing - generates a unique signature of fixed length for a data set or message)	Length of plaintext input is the same as the encryption length
Language of message	Uses random generated characters like passwords with next to none patterns occurring	Uses known languages like English with occurring patterns that could be analyzed
Layering	Encrypts in layers	Encrypts a single time

These factors identified were interpreted in this situation to create the following ***characteristics of a strong encryption***

Factors identified that contribute to the strength of an encryption

Factors of Weak Encryption	Factors of Strong Encryption	Applying research to New Encryption
Key limited in length and type	Key not limited	Keyword + Integer Key
Strictly alphabet characters	Special Characters allowed	Alphabet + Special Characters
Not case sensitive – do not differentiate between small and capital	Case sensitivity – differentiate between small and capital	Not case sensitive (Beyond skillset at this moment)
Single technique	Different techniques	Combination of Caesar Cipher + Vigenère Cipher + Twist

Message length = Encryption length	Encryption length not the same as input length = more possibilities	Length of encryption = Length of keyword
Patterns occur + can be analyzed ex. encrypting English linear	Little to no visible patterns	Patters non visible = cannot be analyzed
Encrypts a single time	Encrypts in layers	Encrypts in multiple layers

After using two different techniques to decrypt messages, differences were noted and listed below:

Advantages and disadvantages to using a brute force technique

Pros	**Cons**
Button has to be only pressed once	Depends on the correct decryption being easily distinguishable
Guaranteed the correct decryption	Slow
	Creates a long list

Advantages and disadvantages to using a frequency analysis technique.

Pros	**Cons**
Shortens the list of possible decryptions	Not guaranteed the correct decryption (if original message did not contain an e)
Fast	Button in some cases needs to be pressed multiple times
Works within little tries if original message contained e's	Relies on the assumption that the original message was English
	Does not work at all if original message did not contain e's

Brute Force vs. Frequency Analysis: The differences

Brute Force	**Frequency Analysis**
Slow	Fast
Manually distinguish the correct decryption	Manually distinguish the correct decryption
Button pressed once	Button pressed more than once
Always guaranteed the correct decryption in the 26 options	Not always guaranteed the correct decryption (if original message did not contain an e)
Creates extensive list	Creates shorter list
Works whether original sentence contained an e or not	Does not work if original message did not contain an e

Knowing different attacks helps to prevent them

Brute Force Attack	***Frequency Analysis Attack***
Uses every possible key	*Analyses English to guess the most possible key*
Same key is used for every character	

- **How can these attacks be stopped?** *Encrypt every character with different key:*

1. Creates an impossibly long list of options during Brute Force (26^k ; k = length of keyword);
2. Every character with different key: Encryption cannot be analysed anymore because it is non-linear ex. An “E” can be encrypted as an “P” and “Y” in the same text;

Significance of each prototype leading to the final program

- Prototype 1:
  This prototype is significant because it identified the problem of the program including special characters and encrypting punctuation, making it difficult to break a normal Caesar Cipher like the aim was. This identification allowed for the problem to be fixed and the decryption of the final program to have a solid foundation and to run smoothly.

- Prototype 2:
  This prototype is significant because it identified the problem in which we cannot use the key to decrypt the message, because the key would be unknown to us which allowed for the final program to now use pre-known keys to decrypt messages. Secondly it identified that it would take too long to manually enter 25 keys to see which key worked, thus allowed for the final program to be automatic and only require the single press of one button. Thirdly it identified that a brute force technique on its own is not sufficient because the user has to distinguish between 25 options, thus allowed for the final program to have 2 distinctly different options of decrypting a message.

- Prototype 3:
  This prototype is significant because it firstly identified that frequency analysis was way too manual and took too long for it to be any faster and more productive than a brute force technique, this allowed for the final program to have frequency analysis automated and not require a user to si9ft through a large list for the most and second most occurring character. Secondly it allowed for the final program to automatically use the second or third most occurring character to keep on analysing until the correct decryption is reached, unlike before where the user had to manually sift through a large list again and again. Lastly, it allowed us to see the way 2 separate buttons in frequency analysis slowed down the process and made it possible for the final program to have only one button increasing the speed and productivity.

Caesar Cipher vs. Vigenère cipher

What happens when a simple adjustment of technique is changed?

Caesar Cipher	**Vigenère cipher**

One key used to encrypt entire message	Different key used for each individual character
Frequency analysis can be used to find the key	***Vulnerability:*** Repetitions in the ciphertext indicate repetitions in the plaintext, and the space between such repetitions hint at the length of the keyword.
Can be broken using a brute force technique	Cannot be decrypted using brute force, because each letter can be encrypted with 26 possible keys, and that is excluding special characters and spaces.

Limitations and errors

Programming Background

The overall result of this project was more the physical program that was built and less the results it gathered.

I was limited by not having an IT / programming background before Grade 10, I have only my Grade 10 IT knowledge of about 8 months that I had to use in order to build this program.

Because of my limitation in knowledge about other languages, I was limited to only the Delphi programming language and not able to use other languages with more features like Python to write this program.

This also affected the results my program helped me gathered:

This program was created using the programming language Delphi which is quite outdated and not frequently used by users. Information about factors that contribute to a weak encryption was gathered using a program that was designed with Delphi so thus the results could differ with other programming languages.

For example, Python has a function that could automatically detect an English word/sentence/phrase which would change one of my factors and thus my results.

Language

For this program to work successfully and to be able to get results, it had to rely on the assumption that the user's input message is English or Afrikaans.

This is because the brute force technique relied on the user to distinguish between 25 other options to find which one looks distinctly different than the rest (thus English or Afrikaans). If the input message had been a password or a randomly generated message, there would have been no way for a user to manually distinguish the original text from the other 24 options because they all would have looked random.

In the same way, the frequency analysis technique also relied on analysing the frequencies of different letters in each language and the predictable occurrences in languages. If the input message had been random characters or a password, the frequency analysis would just straight up not work because of the randomness and unpredictability in the text.

This goes the same for if the input message had been another language than Afrikaans and English because the character occurring the most in that language would have been different than the "e" for Afrikaans and English.

Most other languages also use plenty special characters with unusual signs that would have made it harder to encrypt and decrypt and include a bunch of other factors as well.

The Insider Threat

("*Six Reasons why Encryption isn't working", 2015*)

This program only provided for two cases of decryption but did not provide for a case in which somebody could be getting keys from a person working inside of the system.

According to a reference, the "Insider Threat" is one of the biggest causes for data breaches and in a case like that, the whole decryption process will be totally unnecessary because the key is known and little to no work or effort has to be put into breaking the encryption.

One of the biggest causes of data breaches is insiders selling these secure keys or sensitive information to hackers in exchange for money. (See references)

Recommendations for Future Research

Make use of a new programming language used in the practice like Java or Python. Create a program using the research in this project to further strengthen and test the durability of the encryption. Add new types of encryption techniques like Transposition Ciphers. Add more layers like 2D or 3D encryption. By testing the durability in a new language, another perspective of where it can be improved can come to light.

Conclusion

Applying the bare basics of an encryption technique or Cipher will always be hackable in some kind of way, BUT by being innovative and combining techniques, applying the research in this project, and adding twists or information unknown to hackers, will result in an encryption system to be almost unbreakable.
CONSTANT IMPROVING and INNOVATION of these systems is the biggest step towards minimizing the huge number of data breaches.

Acknowledgments

- Mrs. Annelize van Rooyen
- Mr. Viwe, my mentor
- Mr. Elmar Henning, my IT teacher

References

- Computer science: Unit 2: Cryptography [Online]. 2023. Khan Academy, date unknown. Available from: https://www.khanacademy.org/computing/computer-science/cryptography/crypt/v/caesar-cipher [Accessed on 2 June 2023]

- Jak, O. 2008. DELPHI: Encryption examples [Online] Teachitza, date unknown. Available from: http://www.teachitza.com/delphi/encryption.htm [Accessed on 5 July 2023]

- Finio, B. 2020. Crack the Code: Breaking a Caesar Cipher. Science Buddies [Online] Science Buddies, 20 November. Available from: https://www.sciencebuddies.org/science-fair-projects/project-ideas/Cyber_p005/cybersecurity/crack-caesar-cipher [Accessed on 15 April 2023]

- Rubens, P. 2014. 6 Tips for Stronger Encryption [Online] Esecurityplanet, 23 January. Available from: https://www.esecurityplanet.com/networks/tips-for-stronger/ [Accessed on 24 April 2023]

- Gualt, M. 2015. Six Reasons why Encryption isn't working [Online] Guardtime, 15 March. Available from: https://guardtime.com/blog/6-reasons-why-encryption-isnt-working [Accessed on 16 July 2023]

- Chin, K. 2023. Biggest Data Breaches in US History [Online] Upguard, 18 July. Available from: https://www.upguard.com/blog/biggest-data-breaches-us [Accessed on 25 May 2023]

- Gordon, S. 2014. classical-ciphers-frequency-analysis-examples [Online] Sandilands, 3 November. Available from: https://sandilands.info/sgordon/classical-ciphers-frequency-analysis-examples [Accessed on 30 July 2023]

- Rodriguez-Clark, D. 2017. Frequency Analysis: Breaking the Code [Online] Crypto Corner, date unknown. Available from: https://crypto.interactive-maths.com/frequency-analysis-breaking-the-code.html [Accessed on 28 July 2023]

- Unknown, 2023. VIGENERE CIPHER [Online] thedetectivesociety, date unknown. Available from: https://thedetectivesociety.com/how-to-solve-ciphers/vigenere-cipher/#:~:text=Decoding%20a%20Vigenere%20cipher%20without,then%20finally%20deciphering%20the%20message. [Accessed on 27 August 2023]

- Beakal Tiliksew, Pavan Yadav, Karleigh Moore, 2023. Vigenère Cipher [Online] Brilliant, 27 August 2023. Available from: https://brilliant.org/wiki/vigenere-cipher/#:~:text=The%20primary%20weakness%20of%20the,individually%2C%20can%20be%20easily%20broken. [Accessed on 27 August 2023]

- Finio, B. 2020. Crack the Code: Breaking a Caesar Cipher. Science Buddies [Online] Science Buddies, 20 November. Available from: https://www.sciencebuddies.org/science-fair-projects/project-ideas/Cyber_p005/cybersecurity/crack-caesar-cipher [Accessed on 15 April 2023]

- Chin, K. 2023. Biggest Data Breaches in US History [Online] Upguard, 18 July. Available from: https://www.upguard.com/blog/biggest-data-breaches-us [Accessed on 25 May 2023]
- Gualt, M. 2015. Six Reasons why Encryption isn't working [Online] Guardtime, 15 March. Available from: https://guardtime.com/blog/6-reasons-why-encryption-isnt-working [Accessed on 16 July 2023]
- Further in references in Report File

Appendix

Code for “Encrypt” Button

```
procedure TForm1.btnEncodeClick(Sender: TObject);
Var
  arrAlphabet : Array[0 .. 25] of char;
  i : integer;
  sPlaintext, sCyphertext : string;
  iKey : Integer;
  j: Integer;

begin

  sPlaintext := edtPlaintext.text;
  sPlaintext := uppercase(sPlaintext);

  iKey := StrToInt(edtKey.text);

  //Populate the array with characters (uppercase) from
  //the ASCII table, A = 65, Z = 90

  for i:= 0 to 25 do
    begin
      arrAlphabet[i] := char(65+i);
    end;


   sCyphertext := '';
   for i := 1 to length(sPlaintext) do
   begin
     for j := 0 to 25 do
      begin
        if sPlaintext[i] = arrAlphabet[j] then
        begin
          sCyphertext := sCyphertext + arrAlphabet[(j + iKey) mod 26];
        end;

      end;

   end;

   memCyphertext.lines.add(sCyphertext);



end;
```

Trial Code for “Brute Force” Button

```
procedure TForm1.Button1Click(Sender: TObject);
Var
  arrAlphabet : Array[0 .. 25] of char;
  i : integer;
  sPlaintext2, sCyphertext2 : string;
  iKey2 : Integer;
  j: Integer;

begin

  sCyphertext2 := edtCyphertext2.text;
  sCyphertext2 := uppercase(sCyphertext2);

  iKey2 := StrToInt(edtKeyGuess.text);

  //Populate the array with characters (uppercase) from
  //the ASCII table, A = 65, Z = 90

  for i:= 0 to 25 do
    begin
      arrAlphabet[i] := char(65+i);
    end;


   sPlaintext2 := '';
   for i := 1 to length(sCyphertext2) do
   begin
     for j := 0 to 25 do
      begin
        if sCyphertext2[i] = arrAlphabet[j] then
        begin
          sPlaintext2 := sPlaintext2 + arrAlphabet[(j - iKey2) mod 26];
        end;

      end;

   end;

   memPlaintext2.lines.add(sPlaintext2);



end;
```

```
procedure TForm1.Button1Click(Sender: TObject);
Var
  arrAlphabet : Array[0 .. 25] of char;
  i : integer;
  sCyphertextInput, sPlaintextOutput : string;
  iKeyGuess : Integer;
  j: Integer;
begin
  sCyphertextInput := edtCyphertextInput.text;
  sCyphertextInput := uppercase(sCyphertextInput);

  for i:= 0 to 25 do
    begin
      arrAlphabet[i] := char(65+i);
    end;

    for iKeyGuess := 1 to 25 do
    Begin

        sPlaintextOutput := '';
        for i := 1 to length(sCyphertextInput) do
        begin
          for j := 0 to 25 do
           begin
             if sCyphertextInput[i] = arrAlphabet[j] then
             begin
               sPlaintextOutput := sPlaintextOutput + arrAlphabet[(j + ( 2
             end;

           end;

        end;

        memPlaintextOutput.Lines.Add( IntToStr(iKeyGuess) + ': ' + sPlainte
        memPlaintextOutput.Lines.Add(' ');
     End;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  memPlaintextOutput.Lines.Clear;
end;
```

```
procedure TForm1.Button4Click(Sender: TObject);
Var
   Count : Integer;
   a, Len, b : Integer;
   sCypherAna, sCheck : String;

begin
 // COUNT THE LETTERS....
  sCypherAna := edtCyphertextInput.Text;
  Count := 0;
  Len := Length(sCypherAna);

  for b := 65 to 90 do
  begin
   Count := 0;
      for a := 1 to Len do
        Begin
          if sCypherAna[a] = Char(b) then
            Begin
              inc(Count);
            End;

        End;

    memDataAna.Lines.Add( Char(b) + ': ' + IntToStr(Count));
  end;
```

Code for "Decrypt" Button

```
procedure TForm1.Button5Click(Sender: TObject);
Var
  arrAlphabet : Array[0 .. 25] of char;
  i : integer;
  sCyphertextInput, sPlaintextOutput : string;
  j: Integer;

  iGuess : Integer;
  sMost : String;
  cMost : Char ;
  iMost : Integer;
begin
//Getting the key guess

  sMost := uppercase(edtMost.Text);
  cMost := sMost[1] ;
  iMost := ord(cMost) ;

  iGuess := abs(69 - iMost);

//Decoding

  sCyphertextInput := edtCyphertextInput.text;
  sCyphertextInput := uppercase(sCyphertextInput);

  for i:= 0 to 25 do
    begin
      arrAlphabet[i] := char(65+i);
    end;
    sPlaintextOutput := '';
    for i := 1 to length(sCyphertextInput) do
      begin
        for j := 0 to 25 do
          begin
              if sCyphertextInput[i] = arrAlphabet[j] then
              begin
                sPlaintextOutput := sPlaintextOutput + arrAlphabet[(j + ( 26
              end;

          end;

      end;

    memDecode.Lines.Add( IntToStr(iGuess) + ': ' + sPlaintextOutput);
    memDecode.Lines.Add(' ');

end;
```

【評語】190037

This project attempts to decrypt the encrypted message in English using frequency analysis technique. The message is encrypted in Caesar Cipher and Vigenère Cipher schemes, respectively. The experiments were conducted over various types of messages other than English text with varying degrees of success and failure. The author developed programs to test the strength and weakness of the two schemes and came up with some factors that can affect the strength and weakness of the two schemes. Overall, this study is interesting but the results are quite basic.