Web Technology Coursework 1 Report

Jonathan Binns

40311703

Introduction

In this task the goal is to create a website that allows the user to encipher and decipher text in a choice of different ciphers. It should have a rewarding user experience that is easy to navigate and easy to both enter and retrieve the messages from. It should be achieved by having multiple files for each web page, a .HTML file which dictates the layout and text of the page, a .CSS file that dictates the styling of the page and finally a .js file that contains any JavaScript methods needed in the website.

The two ciphers I decided to implement are Bacon's cipher and the Rail Fence cipher. Bacon's cipher was invented by Sir Francis Bacon around 1576 to 1597 while he was in Paris (Dawkins. 2016). The main principle of this cipher is that each letter in the alphabet is given a 5 letter code made up of two values, for example 'b' becomes 'aaaab'. The main reason I chose this cipher is that it became the basis of important ciphers such as Morse Code and it was also the basis for how alphabetic characters can be represented in binary (Dawkins. 2016). The second cipher I chose was the rail fence cipher. It is an example of transposition or route cipher which was popular during the early history of cryptography (Simmons. 2011). The cipher works by splitting the message into a known number of parts, known as the key. The message is then split into these parts based on where each character is in the message, for example 'abcd' becomes 'acbd'. In order to decode the message the enciphered message would be written diagonally in a grid, revealing its message. I chose this cipher because it's doesn't follow a standard cipher structure, the letters are not replaced by another letter the entire string is rearranged. Testing my skills of working with strings and arrays of characters.

Software Design

The first place I started when designing this website was coming up with a list of requirements for how it will function and how it will look. This allows me to have a good idea of what my website will look like before I have written any code. My first requirement is that the website should be easy to navigate, I plan on ensuring this by making the website as minimal as possible, while still being functional. it should only have the features and layout necessary for enciphering and deciphering text. This stops the website from becoming too complex and hard to navigate. The second requirement for the website is that it should encipher and decipher text in two different ciphers. Adding this as a requirement ensures that the website is functional for its most basic of tasks and that it completes the main goal of this assignment. My final requirement for the website is that it should be pleasing to the eye. All text and information presented to the user should be easy to read and understand. This leans into making my website easy to navigate, as any links to different pages should be easy to find.

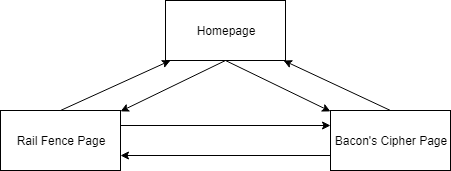


Figure 1: Navigation

As my navigation diagram shows I plan on having three pages in my website. A homepage, a page for Bacon's cipher and a page for the rail fence cipher. This keeps the site from having too much information on one page and thus keeps it simple to use. My home page should have links to the other two pages, this will come from having a area the user can click to take them to their desired cipher. The pages from both the rail fence cipher and Bacon's cipher should link to each other in order to let the user switch ciphers with one click. Finally both cipher pages should link to the home page in order to make sure all pages are accessible from each other.

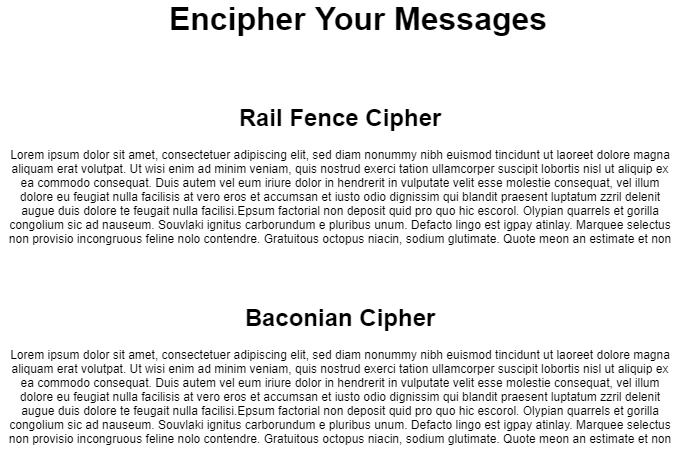


Figure 2: Wireframe for HomePage

I decided to keep the homepage simple and just use it to display information about the ciphers the user can choose. My plan is to have the background of the name of the cipher act as a link to the ciphers page. In order to make sure the user knows what they are about to click I will have the background turn a shade of grey while the mouse is over the link. The background for all pages will be an off white with the text being a dark grey to reduce overall contrast slightly and make the text easier to read.

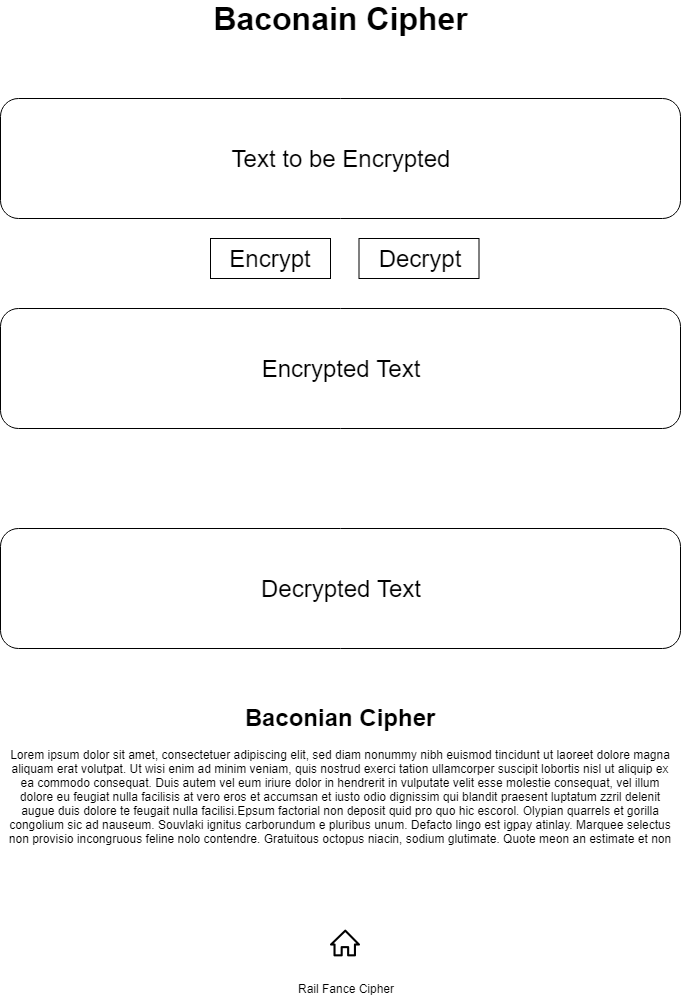


Figure 3:Bacon’s Cipher Wireframe

My layout for both Bacon's cipher and the Rail Fence Cipher are mostly the same, as they have to achieve the same function and keeping a website consistent is an good way to keep it user friendly. The first element is a big text box where the user can add text to be either enciphered or deciphered. Once a button is pressed the text will be displayed in a text box beneath the button denoting what function has been applied to the text. Beneath that is the same information as was presented on the homepage telling people a little bit about the cipher. In order to implement Bacon's cipher I plan on splitting the input string into its individual letters then replacing them with their corresponding code and adding them to an output string. For decoding this cipher my plan is to split the input string into substrings of five characters then take those strings and replace them with their corresponding letter of the alphabet. Version 2 over v1

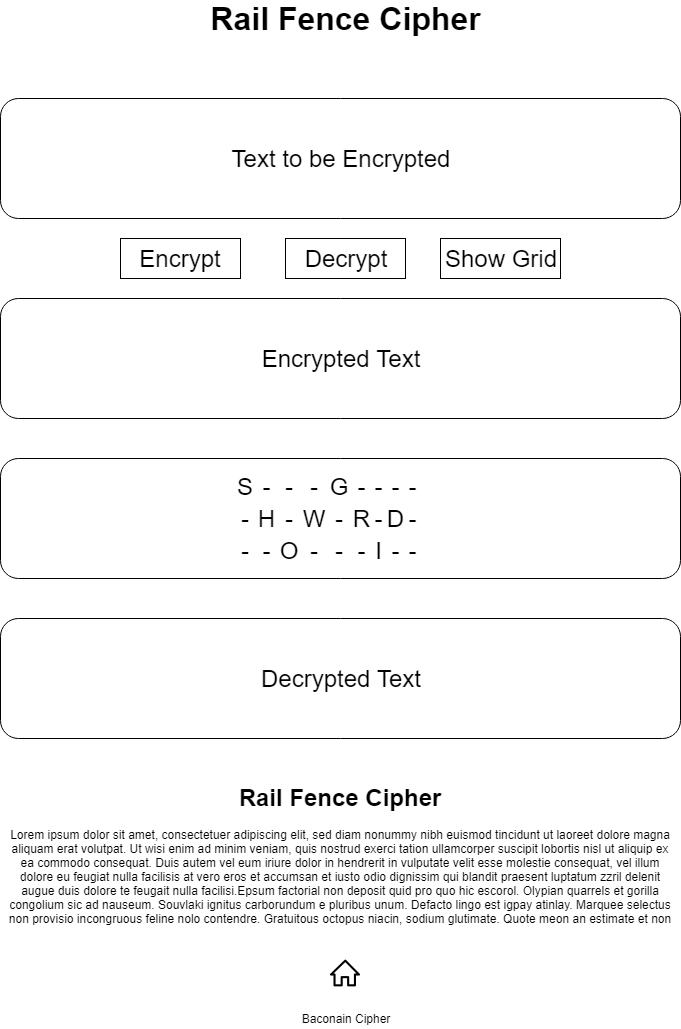


Figure 4: Rail Fence Cipher Wireframe

My design for the Rail Fence cipher is mostly similar to my design for Bacon’s cipher, with the addition of another text output area where the enciphered text can be displayed in a zig zag pattern showing how the cipher can be read. My implementation of this cipher will split the input string into three substrings. The middle string will have around twice as many letters as the top and bottom as it naturally occurs more in the cipher. Ideally, I can do this inside of one loop, taking in the input string and sending the characters to one of 3 others. For decoding on the other hand, I will need to calculate a formula that will tell me how many characters appear in all of the strings so I can ‘rebuild’ them back into one string in the correct order.

Implementation

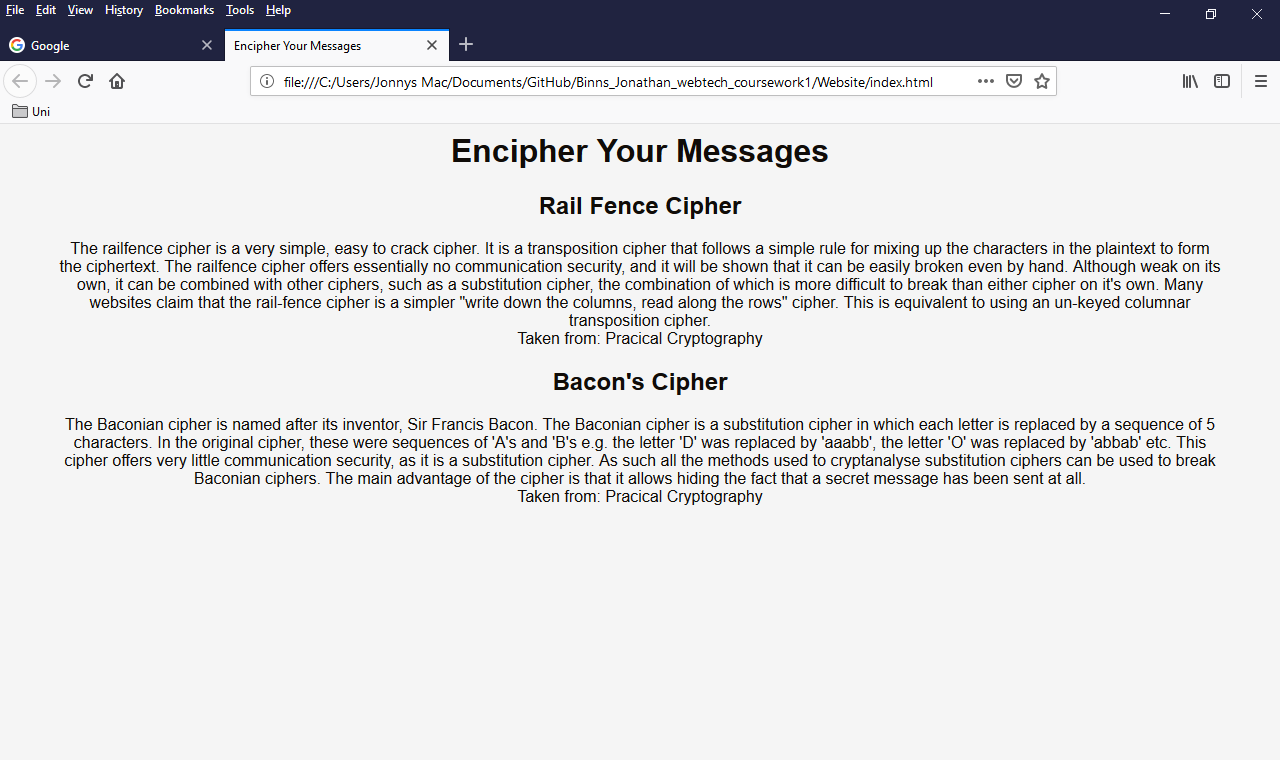


Figure 5: Home Page Screenshot

Implementing the home page was very simple as it only required a .html file and a .css as it only displays text and links to the other pages. Most of the implementation of this website was needed to make it look as close to my wireframe as possible. I went about this first by adding the layout and any information needed in the .html file, with each cipher needing both a title, a paragraph of information and a link to both the page to decode and the page I retrieved the information from. Formatting the links was probably the most complicated aspect of this page as I needed to set each link to stay the same colour as the rest of the font before and after being clicked and also to grey out the background when the user’s mouse is over the link. I did this by using these methods in the index.css file;

**#RailFenceLink** **{text-decoration: none;}**

**#RailFenceLink:link** **{color: #100c08;}**

**#RailFenceLink:hover{background-color:#808080;}**

**#RailFenceLink:visited{color:#100c08;}**

The first line dictates that the link has no text decoration and will display the same as any other piece of text. The second line changes the links colour before it has been clicked, this colour is the same as the rest of the text. The third line sets the background colour to a grey when the mouse is hovering over the link, indicating to the user that it is clickable. Finally the fourth line sets the links colour to be in line with the other text after it has been clicked. (W3.CSS, 2019)

Bacons implementation (show code), explanation

The design of this page is quite simple with many elements being added twice. This was done to keep the site simple and consistent in looks. There’s a text box to enter your message then two buttons to either encode the message or decode the message. The CSS is also very similar to the home page’s CSS in order to keep the look of the site consistent. My implementation to encode Bacon’s cipher is very simple, first I take in the text entered into the input text box and set it equal to the variable ‘input’. I then create a variable ‘output’ to which the encoded message will be added to. After that I loop through each character in the input string, depending on the character add its corresponding code to the output string. I decided to do this as a series of if-else statements in order to keep the code running quickly and to make it easy to correct and understand if any other programmer were to have a look at it. Finally I have the string print out to the corresponding text box. VALIDATION

To decode this cipher, first I take the text entered into the text box and set it equal to ‘input’. Then I remove any non-alphanumeric characters, using ‘string.replace()’, to replace them with nothing and I set the result of that to the string ‘input\_alphanumeric’. After that I split ‘input\_alphanumeric’ into an array of strings which each contain a five-character code. I also create a variable for the length of the array and one for the output string. Next, I loop through each element to the array and compare it to each combination of five-character codes, through another series of if-else statements. Once the code matches one of the if statements its corresponding letter of the alphabet is added to the output string. Finally, the output string is displayed in the corresponding text box. VALIDATION

Rail Fence implementation (show code), explanation

The design of this page is near identical to the page for bacons cipher except there is one more text box to display the enciphered text in its grid so the user can see how the cipher works. Alongside this there is another text box for the message to be shown in its grid. To encode the Rail Fence Cipher I first take in the text input in the corresponding text box and set it equal to the variable ‘input’. I then create three output strings, output 1,2 and 3. Each of these correspond to a ‘row’ of the rail fence. I also create another output variable called ‘output’ so I can combine each of the rows into one string. I then loop through all the characters in the input string and add every 4th character, including the character at position 0 to the first output string. Next I add every other character, starting at input position 1 to the second output string. After that I add every 4th character starting at position 2 to the third output string. Doing this gives me each row of the rail fence so in order to output the encoded string I add output1, output2 and output 3 to each other and set that equal to output so the output is in the correct order. Finally I display output in the corresponding text box. VALIDATION

To decode this cipher I first had to figure out the ratio of how much of the string is taken by the first rail, second rail and third rail. I calculated it to be, the first rail is ¼ of the overall string, if there is a remainder you should round up to the next whole number. The second rail is ½ of the overall string, if there’s a remainder you should round down and the third rail is any of the letters left in the string. The variables I declared are, ‘input’ which takes the text input into the text box. ‘Input\_length\_1’ which stores the value for how many characters are in the first rail, to do this I used math.ceil() to force it to round up if there was a remainder (W3. 2019). ‘Input\_length\_2’ which stores the value for how many characters are in the second rail, I used math.floor() to force it to round any values down if there was a remainder (W3. 2019). ‘Input\_length\_3’, this stores ‘Input\_length\_1 + Input\_length\_2’ which tells me the position of the first character in the third rail. Then I have three input strings named ‘Input1’, ‘Input2’ and ‘Input3’ these store each of their respective rails. Then I have ‘output’ which stores the output string. The first operation I need to do to the input string is split it into its 3 component strings, representing the rails. I do this by looping through the characters in the input string and for each character in a position less than or equal to ‘input\_length\_1’ minus 1 the character gets added to the string ‘input1’ this is the first rail. Next I add all the characters in positions that are bigger than ‘input\_length\_1’ minus 1 but smaller than ‘input\_length\_2’ minus 1 and add them to the string ‘input2’, this is the second rail. Finally all the characters bigger than ‘input\_length\_3’ minus 1 are added to the string ‘input3’ this is the third and final rail. In order to take these three strings and put them back in the original order I decided to add null characters to them so they could be printed out as a grid, since in grid form there is only one letter per column, so I could just add all the characters at each position from the strings input 1,2 and 3. Then before I show the output I could use the string.replace() I used in bacons cipher to remove any non-alphanumeric characters leaving the original string(W3. C. 2019). For the first rail I just had to add three null characters, I chose ‘.’ after each letter. I did this by creating a temporary string called input\_1\_null and then looping through input1 and when a character at a position in the string wasn’t null adding the character and ‘…’ to the temporary string. Once the loop had finished I then created a substring that was equal in length to the original string and set that equal to input1. I did the same for input2 and input3 but for input 2 I needed to add one ‘.’ to the start of the string and one between each letter, for input3 I needed to add two ‘.’ to the start and three in between each letter. Once these strings were created I then just needed to loop through all of them and add each character at every position to the output string and remove any non-alphanumeric characters. All that is left to do after that is display the output in the corresponding text box. My method for displaying the grid is the same as this except instead of removing the null characters and combining the strings into one I display each of them on a new line. VALIDATION

Evaluation

comparison Against the Requirements

The requirements set out in the descriptor are;

‘The aim of this coursework is to design and implement a website that enables your user to encode simple text messages using at least two different classical cyphers or encoding schemes. On your page (index.html) you must provide at least:

1. an area where the user can type in a message
2. be an area where the encoded message can be displayed
3. be a way to select between different ciphers (for example a dropdown)
4. be some method to cause the cipher to be computed (such as a button)
5. be some mechanism for deciphering your messages to recover the plaintext’

I reached the first and second requirements by having text boxes for the user to input their message and text boxes to display the encoded message which are written to in the program on both cipher pages. This is the best method as it is the simplest to implement with both pages and since the text boxes are a different colour to the background it makes the text stand out to the user. The third requirement has been reached by having both ciphers on their own separate pages. I then have links to both ciphers on the home page and a link to both the other cipher and the home page on the individual cipher pages. I thought this would be the best method as both of the ciphers require a different number of text boxes and a slightly different layout overall meaning that if they were on the same page it would have become cluttered and confusing, which is against my plan for the sites design as I wanted to keep it as minimalistic as possible, while still being functional. The fourth requirement has been fulfilled as each function to encode, decode or show the cipher in grid form are controlled by buttons. Again, I thought this was the best method to complete this requirement as they are simple to implement and easy to understand. I fulfilled the final requirement by again using text boxes as I felt they were the simplest method to let the user see the decoded message. They are also in keeping with the design of the website, but to avoid any confusion I made sure to include a message above them saying what the text box displays.

Improvements

The first improvement I would have tried is to use version 1 of Bacon’s cipher rather than version 2. The main problem with this is that in version one the letters I and j, along with u and v share the same 5 letter code. To implement this with encoding the text would just mean changing the if-else statements, however to decode the text would be a lot more complex. One idea on how to implement it would be to translate the word asides from I and j, or u and v then use some form of autocorrect to ‘guess’ what the missing letter would be. There is an example online of a Bacon’s cipher version 1 encoder but that implementation omits the letters j and v, which I think loses functionality overall from version 2 and my implementation (Bynens. M. 2015). The second improvement I could make is to include punctuation and spaces in my ciphers. For the rail fence cipher this wouldn’t be too complex as I could just treat them the same as any other letter. The main problem with this is that I would have to chose a different null character to add to the strings when I format them to be shown as a grid and I would have to create exceptions when I remove the null characters to make sure I don’t remove any punctuation or spaces by mistake. Bacon’s cipher doesn’t have any code for punctuation so to encode with it would not be complicated. I could just add them to the output string with the else statement instead of throwing an error. Decoding however would be a lot more difficult as I would either have to remove the punctuation from the input string, store it in an 2 dimensional array along with the position in the string once it was decoded, something I would have to calculate. I would then add the punctuation back to the output string once decoded. Another method would be to take each of the 5 character codes as 6 characters and if there wasn’t any punctuation in the 6th character, add a null character. Then once the codes had been changed for their alphabet counterparts remove the null character that I set before displaying to the user. The third improvement I would make is to try and come up with a more concise method of implementing Bacon’s cipher. I could try using switch statements but with those I would still have to have the same number of statements so it would be just as long of a program. Ideally this would make my program a bit faster but because of all the different statements the actual time saved might be negligible.

Personal Evaluation

The main thing this coursework has taught me is an understanding of all the different languages that go into building a website, before the module I had a basic understanding of html an little understanding of either css or javascript. Now I have a good degree of understanding in all of them and can easily go on to learn more and implement more complex things in these languages. The biggest challenge I faced was working out the equation of the rail fence cipher when its decoding enciphered text. There were no implementations of a 3 fence cipher that I could find, that displayed their code so I had to sit for a long time and create a lot of ciphers with different sized strings in order to figure out how many individual letters are in each rail. Despite doing this I still haven’t managed to figure out the correct notation for the problem, I have it down to the first rail is the first quarter of the string with any remainders rounded up to include the next character. The second rail is the middle half of the string with any remainders rounded down to exclude that character. Finally the third rail is just the remaining letters in the string. The main method I used to come up with this answer was to use my maths skills and make it as easy to implement in javascript as possible instead of leaning on javascript functions that may not function exactly as I needed them to. I feel I performed well overall as I fulfilled the requirements set out in the descriptor, using two ciphers I researched and implemented on my own, both of which tested my ability to manipulate strings and one of which tested my maths skills aswell. These combined to make a functional website.

References

Dawkins, P. (2016). Baconian-Rosicrucian Ciphers, An introduction to the cryptography used by Francis Bacon and the Rosicrucian-Freemasonic fraternity. Accessed On: 03.03.19

Retrieved From: <https://www.fbrt.org.uk/pages/essays/Baconian-Rosicrucian_Ciphers.pdf>

Simmons, Gustavus J. (2011). Transposition cipher. Encyclopaedia Britannica. Accessed On: 03.03.19

Retrieved From: <https://www.britannica.com/topic/transposition-cipher>

W3.CSS (2019) W3.CSS Tutorial. Accessed on 06.03.19

Retrieved From: <https://www.w3schools.com/w3css/default.asp>

W3 (2019) JavaScript Math Reference. Accessed on 06.03.19

Retrieved From: <https://www.w3schools.com/jsref/jsref_obj_math.asp>

Bynans. M. (2015) Bacon-cipher. Accessed on 07.03.19

Retrieved From: <https://github.com/mathiasbynens/bacon-cipher>