Practical 1

Q1

General Methodology

- Due to the syntax of English Natural language, some words are used more often than other words (eg. the, a, an, etc.), and some dictionary words contain letters at higher frequencies than others (eg. 't' occurs in more words, more frequently, than 'z'), and so the most frequent characters in one piece of text are more likely to match the most frequent characters in another.
- Given an arbitrary sample of English natural language text containing the same characters as the plaintext, we can therefore compare the relative frequencies of characters in the ciphertext with those in the sample to help determine the replacement characters.
- We know the plaintext has been made lowercase, with punctuation and non-printing characters except space removed, leaving just the 26 lowercase letters a-z, the digits 0-9, and the space character
- Numbers do not have the same feature as letters in natural language, and so we can just focus on the 26 letters and space character.
- With no information, there are 27 potential choices for the first replacement character, and then 27-1 more for the next. This gives a maximum of 27! options for replacement characters.
- With the frequency data, we can make it more likely to find a plausible solution as the most frequent ciphertext char is more likely replaced by the most frequent char in our sample. We can repeat this for each subsequent most frequent char until we either reach a plausible solution or find a word that isn't a valid English word.
- When an invalid word is reached, we go back one change, and try the next most frequent word.
- To make the process faster, we can use the knowledge of the English language to spot words that are one replacement off a common word and try those changes first

- We can also spot repeated patterns in the ciphertext, and try replacing their characters to form commonly repeated words

Frequency Analysis Methodology

```c
// frequencyAnalysis.c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

typedef struct CharFreq {
    char character;
    int count;
} CharFreq;

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("usage: ./freqAnalysis <string>\n");
        exit(1);
    }

    char* inputString = argv[1];
    int inputStringLength = strlen(inputString);

    CharFreq uniqueChars[inputStringLength];
    int uniqueCharCount = 0;

    // Count frequencies of all unique characters in the input string
    for (int position = 0; position < inputStringLength; position++) {

        char currentChar = inputString[position];

        // Check if the currentChar has already appeared
        bool isUnique = true;
        for (int i = 0; i < uniqueCharCount; i++) {

            // Increment matching unique char
            if (currentChar == uniqueChars[i].character) {
                uniqueChars[i].count++;
                isUnique = false;
                break;
            }
        }

        // If character hasn't appeared yet, append new CharFreq to array
        if (isUnique) {
           uniqueChars[uniqueCharCount++] = (CharFreq) {currentChar, 1};
        }
    }

```

```
48      // Print the frequency array
49      for (int i = 0; i < uniqueCharCount; i++) {
50          float freq = ((float) uniqueChars[i].count / inputStringLength) *
        100;
51          printf("%0.1f%%\t'%c'\n", freq, uniqueChars[i].character);
52      }
53  }
```

- I created a small C program which outputs the relative frequency of each unique character in a given piece of text (as a percentage of string length).
- After sorting this data with the UNIX sort command, I could then compare this to frequency data of an arbitrary sample.

```
0.4%  'G'
0.4%  'K'
0.4%  'O'
1.1%  'U'
1.4%  'B'
1.8%  'Q'
1.8%  'R'
2.2%  'C'
2.2%  'L'
2.5%  'H'
2.9%  'W'
3.3%  'T'
3.3%  'Z'
4.3%  'E'
5.1%  'N'
5.4%  'A'
5.8%  'P'
6.2%  'M'
6.9%  'Y'
7.2%  'D'
7.6%  'S'
9.8%  'J'
18.1% 'X'
```

Collecting an Arbitrary Sample

The challenge in finding a sample for frequency analysis is in finding an arbitrary sample representative of the type of text that will appear in the plaintext, as the goal is to match the reative frequencies to the plaintext.

There are letter frequency charts from various sources collected using huge text corpuses from various text formats, but I chose to find my own to show the methodology.

As the text is in English, I took a sample of the first 12,000 words from the COCA collection of the English Corpora, which in total contains 950 million words from 8 different categories of sources (spoken, academic, blog, fiction, magazine, news, tv, and web) (https://www.english-corpora.org/coca/)

- For each category, I removed the html symbols which appeared (anything in ¡¿ characters), and numbers (as they are not relevant).
- I then then made the same changes made to the plaintext: converting lowercase to uppercase, removing punctuation, and removing all non-printing characters except space.
- Finally, I took the first 12,500 characters from each category to create a 100,000 character sample.

Natural Language Sample Frequency Analysis

With my sample, I then ran the same frequency analysis code I ran on the ciphertext, and sorted the output.

```
0.1%  'q'
0.1%  'x'
0.1%  'z'
0.2%  'j'
0.8%  'k'
0.8%  'v'
1.3%  'b'
1.6%  'p'
1.7%  'g'
1.7%  'w'
1.9%  'f'
1.9%  'y'
2.1%  'm'
2.4%  'c'
2.4%  'u'
2.8%  'd'
3.4%  'l'
4.2%  'h'
5.1%  'r'
5.2%  's'
5.6%  'n'
5.9%  'i'
6.4%  'a'
6.6%  'o'
7.4%  't'
9.8%  'e'
18.4% ' '
```

Solving the Cipher

With the relative frequency data for both the ciphertext and arbitrary sample, we can begin solving the cipher by attempting to determine character replacements starting from the most frequent characters in the ciphertext to the least frequent.

For this, I created another small program which can replace 1 character in a given ciphertext with a given character.

For each replacement, we get a new branch of potential solutions

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[]) {

    if (argc != 4) {
        printf("usage: ./solver <ciphertext> <ciphertext_char> <plaintext_char>\n");
        exit(1);
    }

    char* cipherText = argv[1];
    int cipherLength = strlen(cipherText);

    char cipherChar = argv[2][0];
    char plainChar = argv[3][0];

    // Print ciphertext, replacing the cipherChar with plainChar
    for (int i = 0; i < cipherLength; i++) {
        if (cipherText[i] == cipherChar) {
            printf("%c", plainChar);
        } else {
            printf("%c", cipherText[i]);
        }
    }

    printf("\n");

}
```

- We begin with the initial ciphertext

  UEJNXUJXRMQBDPJXSEJXYNTYCYTHDWAXMZXSEJXADQJXCDPYJSLXMPXAHGCDPYJSLXM
  ZXMHPXMWTJPXRHWSYCDSJTXBWDNSAXDNTXDNYQDWAXMNJXMZXSEJXZYPASXBMYNSAXU
  EYREXASPYOJAXHAXYAXSEDSXSEJLXKJNJPDWWLXTYZZJPXQMPJXZPMQXJDREXMSEJPX
  SEDNXTMXSEJXYNTYCYTHDWAXMZXDNLXMNJXABJRYJAXMPXCDPYJSLXYNXDXASDSJXMZ
  XNDSHPJX

- Taking the most frequent ciphertext char 'X' (18.1%), we replace it with the most frequent sample character ' ' (18.4%), whose relative frequencies match almost perfectly, double the frequency of the next highest.

  UEJN UJ RMQBDPJ SEJ YNTYCYTHDWA MZ SEJ ADQJ CDPYJSL MP AHGCDPYJSL MZ
  MHP MWTJP RHWSYCDSJT BWDNSA DNT DNYQDWA MNJ MZ SEJ ZYPAS BMYNSA UEYRE
  ASPYOJA HA YA SEDS SEJL KJNJPDWWL TYZZJP QMPJ ZPMQ JDRE MSEJP SEDN TM
  SEJ YNTYCYTHDWA MZ DNL MNJ ABJRYJA MP CDPYJSL YN D ASDSJ MZ NDSHPJ

- As this is a plausible separation of words, we can continue the same process with the next most common characters 'J' (9.8%) and 'e' (9.8%).

```
UEeN Ue RMQBDPe SEe YNTYCYTHDWA MZ SEe ADQe CDPYeSL MP AHGCDPYeSL MZ
MHP MWTeP RHWSYCDSeT BWDNSA DNT DNYQDWA MNe MZ SEe ZYPAS BMYNSA UEYRE
ASPYOeA HA YA SEDS SEeL KeNePDWWL TYZZeP QMPe ZPMQ eDRE MSEeP SEDN TM
SEe YNTYCYTHDWA MZ DNL MNe ABeRYeA MP CDPYeSL YN D ASDSe MZ NDSHPe
```

- The next highest frequencies 'S' (7.6%) and 't' (7.4%) can then be used in the same way.

```
UEeN Ue RMQBDPe tEe YNTYCYTHDWA MZ tEe ADQe CDPYetL MP AHGCDPYetL MZ
MHP MWTeP RHWtYCDteT BWDNtA DNT DNYQDWA MNe MZ tEe ZYPAt BMYNtA UEYRE
AtPYOeA HA YA tEDt tEeL KeNePDWWL TYZZeP QMPe ZPMQ eDRE MtEeP tEDN TM
tEe YNTYCYTHDWA MZ DNL MNe ABeRYeA MP CDPYetL YN D AtDte MZ NDtHPe A
```

- Here, I noticed that the pattern "tEe" appears three times. This points to 'E' (4.3%) being replaced with 'h' (4.2%) to make the common word "the". Their relative frequencies also supports this replacement.

```
UheN Ue RMQBDPe the YNTYCYTHDWA MZ the ADQe CDPYetL MP AHGCDPYetL MZ
MHP MWTeP RHWtYCDteT BWDNtA DNT DNYQDWA MNe MZ the ZYPAt BMYNtA UhYRh
AtPYOeA HA YA thDt theL KeNePDWWL TYZZeP QMPe ZPMQ eDRh MtheP thDN TM
the YNTYCYTHDWA MZ DNL MNe ABeRYeA MP CDPYetL YN D AtDte MZ NDtHPe
```

- This appears plausible so far. We can then spot the pattern 'thDt'. The most common word with this pattern is 'that', and so we can try replacing 'D' (7.2%) with 'a' (6.4%).

```
UheN Ue RMQBaPe the YNTYCYTHaWA MZ the AaQe CaPYetL MP AHGCaPYetL MZ
MHP MWTeP RHWtYCateT BWaNtA aNT aNYQaWA MNe MZ the ZYPAt BMYNtA UhYRh
AtPYOeA HA YA that theL KeNePaWWL TYZZeP QMPe ZPMQ eaRh MtheP thaN TM
the YNTYCYTHaWA MZ aNL MNe ABeRYeA MP CaPYetL YN a Atate MZ NatHPe
```

- Continuing to be plausible, we can see the pattern 'Atate', which has only 1 dictionary word it could be 'state', so we can try replacing 'A' (5.4%) with 's' (5.2%) - their frequencies also supporting this attempt.

```
UheN Ue RMQBaPe the YNTYCYTHaWs MZ the saQe CaPYetL MP sHGCaPYetL MZ
MHP MWTeP RHWtYCateT BWaNts aNT aNYQaWs MNe MZ the ZYPst BMYNts UhYRh
stPYOes Hs Ys that theL KeNePaWWL TYZZeP QMPe ZPMQ eaRh MtheP thaN TM
the YNTYCYTHaWs MZ aNL MNe sBeRYes MP CaPYetL YN a state MZ NatHPe
```

- Still being plausible, we can go to the next highest frequency chars left 'Y' (6.9%) and 'o' (6.6%).

```
UheN Ue RMQBaPe the oNToCoTHaWs MZ the saQe CaPoetL MP sHGCaPoetL MZ
MHP MWTeP RHWtoCateT BWaNts aNT aNoQaWs MNe MZ the ZoPst BMoNts UhoRh
stPoOes Hs os that theL KeNePaWWL ToZZeP QMPe ZPMQ eaRh MtheP thaN TM
the oNToCoTHaWs MZ aNL MNe sBeRoes MP CaPoetL oN a state MZ NatHPe
```

- There are no completed invalid words here, so we can continue to 'M' (6.2%) and 'i' (5.9%)

  ```
  UheN Ue RiQBaPe the oNToCoTHaWs iZ the saQe CaPoetL iP sHGCaPoetL iZ
  iHP iWTeP RHWtoCateT BWaNts aNT aNoQaWs iNe iZ the ZoPst BioNts UhoRh
  stPoOes Hs os that theL KeNePaWWL ToZZeP QiPe ZPiQ eaRh itheP thaN Ti
  the oNToCoTHaWs iZ aNL iNe sBeRoes iP CaPoetL oN a state iZ NatHPe
  ```

- Here, we have a problem, as there are no sample chars left that replace with 'P' to make a plausible word in 'itherP'. Therefore, we go back to the latest change, and try the next most frequent character.

- However, repeating this process runs into an issue. All character combinations of 'M' and 'P' remaining create implausible words for 'MtherP', and so we must go back again to the previous change.

- Therefore, instead of replacing 'Y' with 'o', we replace 'Y' (6.9%) with the next most frequent sample character 'i' (5.9%).

  ```
  UheN Ue RMQBaPe the iNTiCiTHaWs MZ the saQe CaPietL MP sHGCaPietL MZ
  MHP MWTeP RHWtiCateT BWaNts aNT aNiQaWs MNe MZ the ZiPst BMiNts UhiRh
  stPiOes Hs is that theL KeNePaWWL TiZZeP QMPe ZPMQ eaRh MtheP thaN TM
  the iNTiCiTHaWs MZ aNL MNe sBeRies MP CaPietL iN a state MZ NatHPe
  ```

- This is plausible, and so we continue to the next most frequent chars left 'M' (6.2%) replaced with 'o' (6.6%)

  ```
  UheN Ue RoQBaPe the iNTiCiTHaWs oZ the saQe CaPietL oP sHGCaPietL oZ
  oHP oWTeP RHWtiCateT BWaNts aNT aNiQaWs oNe oZ the ZiPst BoiNts UhiRh
  stPiOes Hs is that theL KeNePaWWL TiZZeP QoPe ZPoQ eaRh otheP thaN To
  the iNTiCiTHaWs oZ aNL oNe sBeRies oP CaPietL iN a state oZ NatHPe
  ```

- Continuing to be plausible, there are a few words that are one char off being solved that appear to be common words, such as the pattern 'otheP'. We can try first replacing 'P' (5.8%) with 'r' (5.1%) (to make 'other'.

  ```
  UheN Ue RoQBare the iNTiCiTHaWs oZ the saQe CarietL or sHGCarietL oZ
  oHr oWTer RHWtiCateT BWaNts aNT aNiQaWs oNe oZ the Zirst BoiNts UhiRh
  striOes Hs is that theL KeNeraWWL TiZZer Qore ZroQ eaRh other thaN To
  the iNTiCiTHaWs oZ aNL oNe sBeRies or CarietL iN a state oZ NatHre
  ```

- We can continue this process, of trying to change characters which only have one plausible substitution, or picking the most frequent unsolved ciphertext char and trying the next highest frequency sample char. Repeating this, and going back one step when there are no plausible solutions with the previous level change and trying the next highest frequency sample char.

  ```
  when we compare the individuals of the same variety or subvariety of
  our older cultivated plants and animals one of the first points which
  ```

```
strikes us is that they generally differ more from each other than do
the individuals of any one species or variety in a state of nature
```

- This is plausible, and we have a final plausible replacement alphabet below

cipher	plain
X	' ' (space)
J	e
S	t
E	h
D	a
A	s
Y	i
M	o
P	r
N	n
Z	f
T	d
W	l
H	u
L	y
C	v
R	c
Q	m
B	p
U	w
O	k
K	g
G	b

Q2

The MiniChaCha20 double round consists of a Column Round, and Diagonal Round, where both then consist of four Quarter Rounds which in turn consist of four Basic Rounds which use simple addition $\mod 2^n$ (where n is the wordsize, in this case 4), XOR, and bitwise-rotation.

The quarter rounds rotation distances in ChaCha20 are 16,12,8,and 7, but with the wordsize of 4, rotating by a multiple of 4 will not change the value. I went with rotation distances of 3,2,2,1 which make sure the values are always changed. I also converted the HEX to BASE10 to make it easier to calculate the first quarter round values by hand.

15	14	13	12
11	10	9	8
7	6	5	4
3	2	1	0

Column Rounds

Here, I will give a detailed run through of the first quarter round, which affects the first column in the state.

$$QR(X_0, X_4, X_8, X_{12})$$

$$X_0 = 15, X_4 = 11, X_8 = 7, X_{12} = 3$$

1. Basic Round 1

   $B(X_{12}, X_0, X_4, 3)$
   $X_0 = (15 + 11) \mod 2^4 = 26 \mod 16 = 10$
   $X_{12} = 3(0011) \oplus 10(1010) = 9(1001)$
   $X_{12} = 9(1001) <<< 3 = 12(1100)$

   Changing the values to

   $X_0 = 10$
   $X_4 = 11$
   $X_8 = 7$
   $X_{12} = 12$

2. Basic Round 2

   $B(X_4, X_8, X_{12}, 2)$
   $X_8 = (7 + 12) \mod 2^4 = 19 \mod 16 = 3$
   $X_4 = 11(1011) \oplus 3(0011) = 8(1000)$
   $X_4 = 8(1000) <<< 2 = 2(0010)$

   Changing values to

   $X_0 = 10$
   $X_4 = 2$
   $X_8 = 3$
   $X_{12} = 12$

3. Basic Round 3

   $B(X_{12}, X_0, X_4, 2)$
   $X_0 = (10 + 2) \mod 2^4 = 12 \mod 16 = 12$

$X_{12} = 12(1100) \oplus 12(1100) = 0(0000)$
$X_{12} = 0(0000) <<< 2 = 0(0000)$

Giving

$X_0 = 12$
$X_4 = 2$
$X_8 = 3$
$X_{12} = 0$

4. Basic Round 4

   $B(X_4, X_8, X_{12}, 1)$
   $X_8 = (3 + 0) \mod 2^4 = 3 \mod 16 = 3$
   $X_4 = 2(0010) \oplus 3(0011) = 1(0001)$
   $X_4 = 1(0001) <<< 1 = 2(0010)$

   Giving

   $X_0 = 12$
   $X_4 = 2$
   $X_8 = 3$
   $X_{12} = 0$

The state at the end of the first column round is:

12	14	13	12
2	10	9	8
3	6	5	4
0	2	1	0

Which converts to HEX as:

C	E	D	C
2	A	9	8
3	6	5	4
0	2	1	0

The state after repeating the same process for the other 3 columns in HEX is:

C	C	C	F
2	A	7	C
3	1	D	D
0	6	D	7

0.1 Diagonal Rounds

- After the column rounds, we then run a quarter round for each of the "diagonals" in the state, each undergoing the same 4 basic rounds per diagonal as with the column rounds.
- Each diagnonal begins with one of the elements in the first row (0,1,2,3), and then forms a diagonal by moving down 1 row, and right 1 column, wrapping around to

the far left when moving off the far right, and wrapping to the top when reaching the bottom.

- The first diagonal uses indexes 0,5,10,15. After completing, we get the state:

  5 C C F
  2 6 7 C
  3 1 C D
  0 6 D 7

- The second uses indexes 1,6,11,12. Afterwards, the state is:

  5 7 C F
  2 6 A C
  3 1 C 1
  B 6 D 7

- The third uses indexes 2,7,8,13. Afterwards, the state is:

  5 7 1 F
  2 6 A 5
  3 1 C 1
  B 9 D 7

- The fourth and final uses indexes 3,4,9,14. After, the state is:

  5 7 1 6
  4 6 A 5
  3 7 C 1
  B 9 0 7

Final State

5 7 1 6
4 6 A 5
3 7 C 1
B 9 0 7

Q3

Input: $x = 89, y = 55$ (because $89 > 55$)
Output: (h, s, t) such that $h = gcd(x, y)$, and $h = sx + ty$

1. $s_x = 1$
   $t_x = 0$

2. $s_y = 0$
   $t_y = 1$

3. Looping algorithm while $y \neq 0$
   NOTE: All division is integer divison

   3.1. $x = 55$
   $y = 89 \oplus 55 = 34$
   $s_x = 0$
   $t_x = 1$
   $s_y = 1 - (89 \div 55) * 0 = 1 - (1 * 0) = 1$
   $t_y = 0 - (89 \div 55) * 1 = 0 - (1 * 1) = -1$

   3.2. $x = 34$
   $y = 55 \oplus 34 = 21$
   $s_x = 1$
   $t_x = -1$
   $s_y = 0 - (55 \div 34) * 1 = 0 - (1 * 1) = -1$
   $t_y = 1 - (55 \div 34) * -1 = 1 - (1 * -1) = 1 - -1 = 2$

   3.3. $x = 21$
   $y = 34 \oplus 21 = 13$
   $s_x = -1$
   $t_x = 2$
   $s_y = 1 - (34 \div 21) * -1 = 1 - (1 * -1) = 1 - -1 = 2$
   $t_y = -1 - (34 \div 21) * 2 = -1 - (1 * 2) = -1 - 2 = -3$

   3.4. $x = 13$
   $y = 21 \oplus 13 = 8$
   $s_x = 2$
   $t_x = -3$
   $s_y = -1 - (21 \div 13) * 2 = -1 - (1 * 2) = -1 - 2 = -3$
   $t_y = 2 - (21 \div 13) * -3 = 2 - (1 * -3) = 2 - -3 = 5$

   3.5. $x = 8$
   $y = 13 \oplus 8 = 5$
   $s_x = -3$
   $t_x = 5$
   $s_y = 2 - (13 \div 8) * -3 = 2 - (1 * -3) = 2 - -3 = 5$
   $t_y = -3 - (13 \div 8) * 5 = -3 - (1 * 5) = -3 - 5 = -8$

   3.6. $x = 5$
   $y = 8 \oplus 5 = 3$
   $s_x = 5$
   $t_x = -8$
   $s_y = -3 - (8 \div 5) * 5 = -3 - (1 * 5) = -3 - 5 = -8$
   $t_y = 5 - (8 \div 5) * -8 = 5 - (1 * -8) = 5 - -8 = 13$

   3.7. $x = 3$
   $y = 5 \oplus 3 = 2$
   $s_x = -8$

$t_x = 13$
$s_y = 5 - (5 \div 3) * -8 = 5 - (1 * -8) = 5 - -8 = 13$
$t_y = -8 - (5 \div 3) * 13 = -8 - (1 * 13) = 5 - -8 = -21$

3.8. $x = 2$
$y = 3 \oplus 2 = 1$
$s_x = 13$
$t_x = -21$
$s_y = -8 - (3 \div 2) * 13 = -8 - (1 * 13) = -8 - 13 = -21$
$t_y = 13 - (3 \div 2) * -21 = 13 - (1 * -21) = 13 - -21 = 34$

3.9. $x = 1$
$y = 2 \oplus 1 = 0$
$s_x = -21$
$t_x = 34$
$s_y = 13 - (2 \div 1) * -21 = 13 - (2 * -21) = 13 - -42 = 55$
$t_y = -21 - (2 \div 1) * 34 = -21 - (2 * 34) = -21 - 68 = -89$

4. $h = x = 1$
$s = s_x = -21$
$t = t_x = 34$

verifying solution: $-21 * 89 + 34 * 55 = -1869 + 1870 = 1$

QED

Q4

·	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	0	2	4	6	8
3	0	3	6	9	2	5	8	1	4	7
4	0	4	8	2	6	0	4	8	2	6
5	0	5	0	5	0	5	0	5	0	5
6	0	6	2	8	4	0	6	2	8	4
7	0	7	4	1	8	5	2	9	6	3
8	0	8	6	4	2	0	8	6	4	2
9	0	9	8	7	6	5	4	3	2	1

The numbers 1, 3, 7, and 9 have multiplicative inverses mod 10.

This is because 1 can be multiplied with 1 to get 1. 3 can be multiplied with 7 to get 1, 7 can be multiplied with 3 to get 1, and 9 can be multiplied with 9 to get 1.

All the other numbers: 2,4,5,6, and 8 do not have multiplicative inverses mod 10.

This feature is present in general for the numbers coprime with N, s.t. the numbers are all mod N, due to the feature of the extended euclidian algorithm, which shows that coprime numbers can be displayed in the form $1 = ax + by$, where a and b are integers.

As our field contains integers mod N, any value of a will give a number aN which will always be a multiple of N, with out modular addition, adding a multiple of the mod number N will not change the result, and so acts as adding 0.

This gives us, for these coprimes: $1 = by$, where b is an integer. Therefore, there exists an integer b, such that $b = 1/y$, which is the definition of a multiplicative inverse.

Q5

Polynomials

$0x + 0 = 0$
$0x + 1 = 1$
$0x + 2 = 2$
$1x + 0 = x$
$1x + 1 = x + 1$
$1x + 2 = x + 2$
$2x + 0 = 2x$
$2x + 1 = 2x + 1$
$2x + 2 = 2x + 2$

mod **3 Multiplication Table**

X	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

mod **3 Addition Table**

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

Multiplication Table (before $\mod (x^2+1)$)

$\cdot$	0	1	2	x	$2x$	$x+1$	$x+2$	$2x+1$	$2x+2$
0	0	0	0	0	0	0	0	0	0
1	0	1	2	x	$2x$	$x+1$	$x+2$	$2x+1$	$2x+2$
2	0	2	1	$2x$	x	$2x+2$	$2x+1$	$x+2$	$x+1$
x	0	x	$2x$	x^2	$2x^2$	x^2+x	x^2+2x	$2x^2+x$	$2x^2+2x$
$2x$	0	$2x$	x	$2x^2$	x^2	$2x^2+2x$	$2x^2+x$	x^2+2x	x^2+x
$x+1$	0	$x+1$	$2x+2$	x^2+x	$2x^2+2x$	x^2+2x+1	x^2+2x+2	$2x^2+1$	$2x^2+x+2$
$x+2$	0	$x+2$	$2x+1$	x^2+2x	$2x^2+x$	x^2+2x+2	x^2+x+1	$2x^2+2x+2$	$2x^2+1$
$2x+1$	0	$2x+1$	$x+2$	$2x^2+x$	x^2+2x	$2x^2+1$	$2x^2+2x+2$	x^2+x+1	x^2+2
$2x+2$	0	$2x+2$	$x+1$	$2x^2+2x$	x^2+x	$2x^2+x+2$	$2x^2+1$	x^2+2	x^2+2x+1

- $(2x+1)\cdot(x+2) = (2\cdot 1)x^2 + (2\cdot 2)x + x + 2 = 2x^2 + 1x + 1x + 2 = 2x^2 + 2x + 2$
- $(2x+1)\cdot(2x+2) = (2\cdot 2)x^2 + (2\cdot 1)x + (2\cdot 2)x + (1\cdot 2) = x^2 + 2x + 1x + 2 = x^2 + 0x + 2 = x^2 + 2$

Addition Table

$+$	0	1	2	x	$2x$	$x+1$	$x+2$	$2x+1$	$2x+2$
0	0	1	2	x	$2x$	$x+1$	$x+2$	$2x+1$	$2x+2$
1	1	2	0	$x+1$	$2x+1$	$x+2$	x	$2x+2$	$2x$
2	2	0	1	$x+2$	$2x+2$	x	$x+1$	$2x$	$2x+1$
x	x	$x+1$	$x+2$	$2x$	0	$2x+1$	$2x+2$	1	2
$2x$	$2x$	$2x+1$	$2x+2$	0	x	1	2	$x+1$	$x+2$
$x+1$	$x+1$	$x+2$	x	$2x+1$	1	$2x+2$	$2x$	2	0
$x+2$	$x+2$	x	$x+1$	$2x+2$	2	$2x$	$2x+1$	0	1
$2x+1$	$2x+1$	$2x+2$	$2x$	1	$x+1$	2	0	$x+2$	x
$2x+2$	$2x+2$	$2x$	$2x+1$	2	$x+2$	0	1	x	$x+1$

Multiplication with irreducible polynomials

To find multiplication modulo an irreducible polynomial, you can use polynomial long division, where the remainder is the same as the modulo.

$(2x^2+2) \mod (x^2+1) = 0$

	2		
x^2+1	$2x^2$		$+2$
-	$2x^2$		$+2$
	0	0	0

$(2x^2+2x) \mod (x^2+1) = x$

	2		
x^2+1	$2x^2$	$+x$	$+2$
-	$2x^2$		$+2$
	0	$+x$	0

$(2x^2+2x) \mod (x^2+1) = 2x^2+2x$

	2		-2
x^2+1	$2x^2$	$+2x$	
-	$2x^2$		$+2$
	0	$+2x$	-2
	$-2x^2$		-2
	$2x^2$	$+2x$	0