Assignment 1

Question 1

CIA:

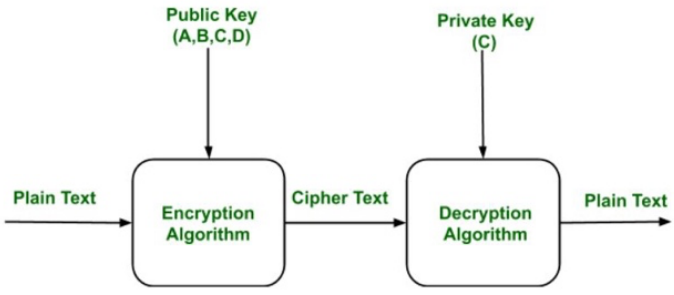
The CIA triad is one of the most core concepts to cybersecurity.
Confidentiality is often associated as being what cybersecurity is at its core, basically keeping a system private to ensure that only authorized people are able to view a certain asset or certain amount of data. For example customer credit card data, personal information leaking out to the public is illegal and a disaster and it calls for confidentiality. We therefore have to ask ourselves who should have access to this data, and are they able to share this with a 3rd part?, these are some basic questions within confidentiality that we have to consider.

Integrity is the ability to ensure that an asset has only been modified by an authorized person, the moment an unathourized entity modifies an asset integrity has been lost. For example if a "man in the middle" intercepts a connection between Alice and Bob, he's then able to receive messages from both ends, read them and send them off to the person it was intended for. In this scenario, confidentiality has obviously been lost, but so has integrity.

Availability applies to both data and services to ensure that an asset can only be used by an authorized user for an intended purpose, and availiblity ensures that we have timely responses to our requests, that resources are fairly allocated when we are trying to request something and that many users can use one service simultaneously without latency issues. On the surface this might not seem as critical as confidentiality and integrity, but consider AWS and their client base, but consider a ddos attack causing the server to not be available for 10 minutes, and considering the amount of companies that rely on this service we are talking millions of dollars within that short timespan.

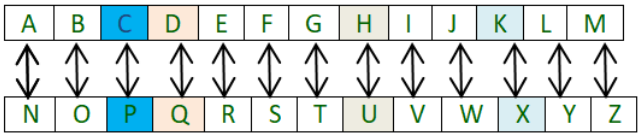
Now, with the CIA triad in mind we can take a practical example where Alice sends a message to Bob through the application "whatsapp". From the moment Alice sends the message "**Introduction to Data and Cyber-Security (DCS3101)"** to Bob receives it, there is a lot going on in order to ensure good security and in this example we will look at just that.

When Alice sends the message, there needs to be some sort of encryption/decryption implemented in order to have a secure line of communication and this is where cryptography comes in. Cryptography converts data into a format that is unreadable for an unauthorized user, allowing it to be transmitted without unauthorized entities decoding it back into a readable format, thus compromising the data".

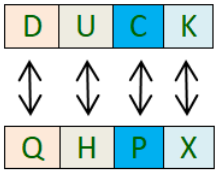


There are many different ways that the encryption and decryption can occur, with different algorithms for this. The keys, a secret input to the encryption/decryption algorithm can also be implemented in different ways, symmetric or asymmetric. In symmetric algorithms both Alice and Bob must have the same secret/private key and if the key is lost or stolen, the security of system is compromised. Asymmetric algorithms use two keys: a public key and a private key. The sender, Alice uses the public key to encrypt a message, and the receiver, Bob uses the private key to decrypt it. The real "magic" of these systems is that the public key cannot be used to decrypt a message. If Bob sends Alice a message encrypted with Alice's public key, it does not matter if everyone else on Earth has Alice's public key, since that key cannot decrypt the message.

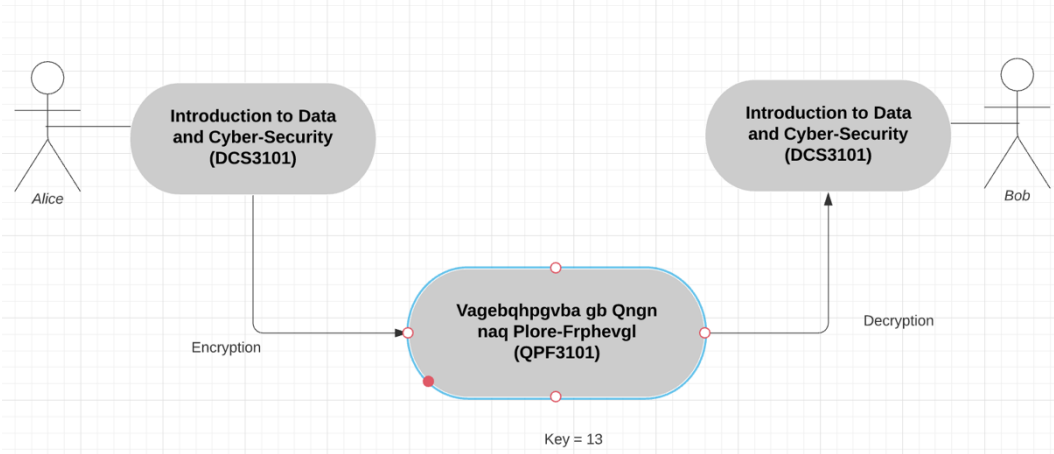
Now, lets take the example where Alice sends Bob a message. The algorithm I will use for this example is a RO13 cipher, which is a special case of the Caesar cipher in which the shift is always 13, as seen by its name RO13 (read as – "rotate by 13 places"). It is a simple letter substitution cipher that replaces a letter with the 13th letter after it, in the alphabet, to undo ROT13, the same algorithm is applied, so the same action can be used for encoding and decoding. The word "HELLO" becomes "URYYB". Obviously, this provides virtually no cryptographic security,but its good for demonstration purposes.



ROT13



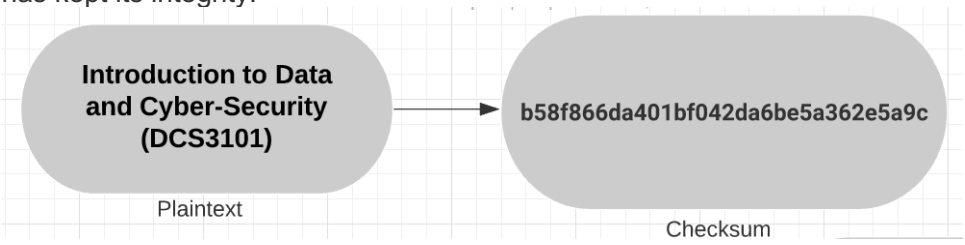
The first pillar of CIA is confidentiality and for the communication between Alice and Bob to have this we must encrypt the message to ensure that only authorized people (Alice and Bob) are able to view these resources. Alice obviously writes the message in plaintext, and the application then uses a ROT13 substitution algorithm to encrypt this message. Alice sends the message "**Introduction to Data and Cyber-Security (DCS3101)**". We shift it by 13 and obtain the encrypted ciphertext: **"Vagebqhpgvba gb Qngn naq Plore-Frphevgl (QPF3101) "**. Using the key, we can simply decode this by doing the inverse of the encryption.



All this happens after alice sends the message and before bob receives the message in order to obtain confidentiality because if we have an impostor Eve who picks up the communication line between Alice and Bob, she will only be able to view the encrypted message and can't decipher it unless she knows the private key, which in this instance isn't really much of a private key, but in a real life example obviously this is much more secure.

Confidentiality is only a part of it, when Alice sends the message to Bob we must ensure that the message has only been modified by an authorized person and that the message Alice sends is the exact one Bob receives, and this is where integrity comes in. If we have a "man in the middle" or in this case "woman in the middle" with Alice who intercepts the connection between Alice and Bob, she's then able to receive messages from both ends, read them and send them off to the person it was intended for. In this scenario, confidentiality has obviously been lost, but so has integrity. To prevent this, when Alice sends the message to Bob we can generate a checksum before and after transmission, this will be a value storing the number of bits in the message and it used to keep the integrity of our resources.

We can generate a checksum using the MD5 algorithm, by taking a string of an any length and encoding it into a 128-bit fingerprint. MD5 hashes are used to ensure the data integrity of our resources. Because the MD5 hash algorithm always produces the same output for the same given input, one can for example compare a message before and after transmission to see if it has kept its integrity.

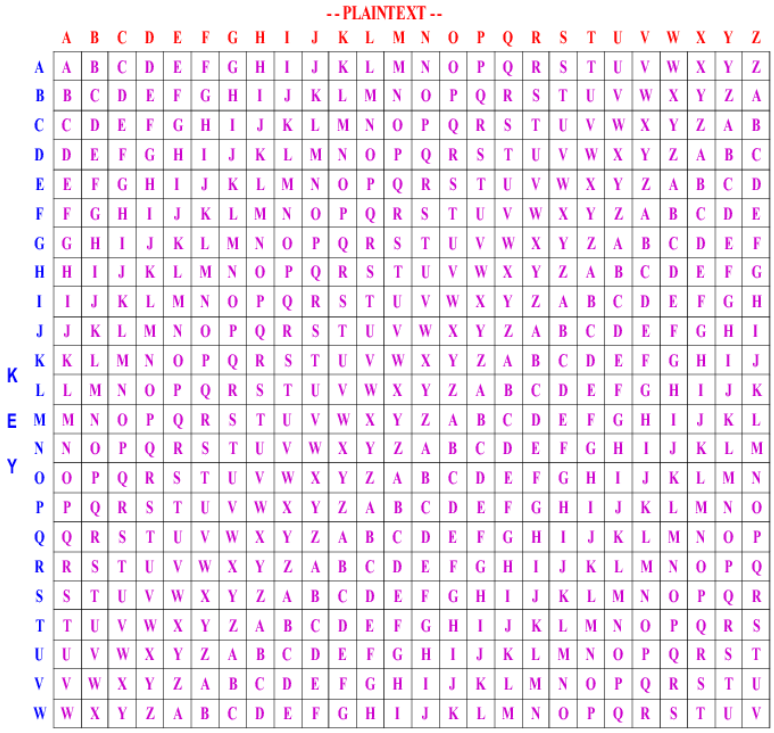


The last part of CIA availability might not seem as crucial as the other two, but if Alice sends a message to Bob and he can't read it because the servers are down, or the network connection is faulty then we have a problem. Availiblity ensures that we have timely responses to our requests, that resources are fairly allocated when we are trying to request something and that many users can use one service simultaneously without latency issues. A common attack here is a ddos attack (Distributed Denial-of-Service) where you flood the server in order to make it unavailable. In order to stop Eve from attack the network there are obviously preventative measures to ensure good availability such as developing a denial of service response plan, securing the network infrastructure or leverage the cloud.

Question 2

What is Vigenere ciphering?

The vigenere cipher is a method of encrypting alphabetic text, it is a poly-alphabetic cipher **meaning it's** based on substitution, using multiple substitution alphabets and it uses a private key containing letters from the alphabet. The cipher uses a clever way of representing possible mappings from one letter to another as a matrix.



We encrypt the plaintext by processing one letter at a time across the columns. We use the letters in a key to tell us which row to look at for the mapping. To illustrate the concepts of the Vigenere cipher further, we take an example with the **plaintext: "LIVERPOOL"** and the **key: "AZIM".** We also note that the key stream should be as long as the plaintext, so we must repeat the key till it meets that requirement, in this case the **key stream** will be:**"AZIMAZIMA"**. As mentioned, we encrypt the plaintext by processing one letter at a time across the columns. The first letter in the plaintext is "**L**", so we look at column "**L**", then we use the letter in the key, "**A**", to look at the row to decide the mapping of "**L**". The mapping then gives us the letter "**L**" which then becomes the first letter of our ciphertext. The next plaintext letter is "**i**", so we look at column "**i**", and we use the letter in the key, "**Z**" to look at row "**Z**" to decide the mapping. Column "**i**" and row "**Z**", gives us the next letter in our ciphertext which is "**H**". By repeating this processs we end up with the ciphertext "**LHDQROWAL**».

In order to decrypt the ciphertext, we simply reverse the process. We work backwards using the same matrix, by first repeating the secret key so its length matches the cipher text.
"LHDQROWAL" (Ciphertext)
"**AZIMAZIMA" (Secret key)**
Then we use the matrix to find the row that corresponds to the first letter in the secret key text- in my case, A. In the A row, find the corresponding cipher text letter L. The vertical column where that cipher text letter is located reveals the plaintext letter L. By repeating this process we then get the plaintext "**LIVERPOOL**". Mathematically, we can describe the encryption and decryption as:

Encryption
The plaintext(P) and key(K) are added modulo 26.
$E_i = (P_i + K_i) \bmod 26$

Decryption

$$D_i = (E_i - K_i + 26) \bmod 26$$

Where P is the plaintext and K is the key. D_i and E_i denotes the offset of the i-th character of the plaintext and ciphertext.

Implementation with one key:
The code can also be found at:

```
ENCRYPTION:
Plaintext: The quick brown fox jumps over the lazy dog.
The keystream is: KONGSBERGKONGSBERGKONGSBERGKONGSBERGKONGSBER
Ciphertext: DVRWMJGBHBCJTXPBAAWDFUNFVKNOZNFQESX

DECRYPTION:
Ciphertext: DVRWMJGBHBCJTXPBAAWDFUNFVKNOZNFQESX
The keystream is: KONGSBERGKONGSBERGKONGSBERGKONGSBERGKONGSBER
Plaintext: THEQUICKBROWNFOXJUMPSOVERTHELAZYDOG
```

```cpp
#include <iostream>
#include <string>
using namespace std;

class VigCipher {
private:
    int counter;
public:
    string k;
    VigCipher(string k); //constructor
    string keyStream(string key, string message);
    string encryption(string plaintext);
    string decryption(string ciphertext);

};

VigCipher::VigCipher(string k){  //generate key

    for (int i = 0; i < k.size(); ++i) {
        if (k[i] >= 'A' && k[i] <= 'Z') //ignore what aren't letters, and capitalize everything.
            this->k += k[i];
        else if (k[i] >= 'a' && k[i] <= 'z')
            this->k += k[i] + 'A' - 'a';
    }
}
```

```
string VigCipher::keyStream(string key, string message){
    int temp = message.size();
    for (int i = 0; ; i++)
    {
        if (temp == i)
            i = 0;
        if (key.size() == message.size())
            break;
        key.push_back(key[i]);
    }
    return key;
}


string VigCipher::encryption(string plaintext){
    string encrypt;
    for (int i = 0, j = 0; i < plaintext.length(); ++i){
        char p = plaintext[i];
    if (p >= 'a' && p <= 'z')
        p += 'A' - 'a';
    else if (p < 'A' || p > 'Z')
        continue;
        encrypt += (p + k[j] - 2 * 'A') % 26 + 'A';
        j = (j + 1) % k.length();
    }
```

```
string VigCipher::decryption(string ciphertext){
    string decrypt;
    for (int i = 0, j = 0; i < ciphertext.length(); ++i)
    {
        char c = ciphertext[i];

        if (c >= 'a' && c <= 'z')  //ignore what aren't letters, and capitalize everything.
            c += 'A' - 'a';
        else if (c < 'A' || c > 'Z')
            continue;

        decrypt += (c - k[j] + 26) % 26 + 'A';  //the actual decryption happens here
        j = (j + 1) % k.length();
    }

    return decrypt;
}

int main()
{
    VigCipher vigenere( k: "Kongsberg");

    string plaintext = "The quick brown fox jumps over the lazy dog.";
    string encryption = vigenere.encryption(plaintext);
    string decryption = vigenere.decryption(encryption);
    string s = vigenere.keyStream(vigenere.k, plaintext);

    cout <<"ENCRYPTION:" <<endl;
    cout << "Plaintext: "<< plaintext << endl;
    cout << "The keystream is: " << s << endl;
    cout << "Ciphertext: " << encryption << endl << endl;

    cout <<"DECRYPTION:" <<endl;
    cout <<"Ciphertext: " << "DVRWMJGBHBCJTXPBAAWDFUNFVKNOZNFQESX" << endl;
    cout << "The keystream is: " << s << endl;
    cout << "Plaintext: " << decryption << endl;
}
```

Implementation with 2 keys:

```
ENCRYPTION:
Plaintext 1 is: The quick brown fox jumps over the lazy dog.
Keystream 1 is: NORWAYNORWAYNORWAYNORWAYNORWAYNORWAYNORWAYNO
Plaintext 2 is: GVVMUGPYSNOUATFTJSZDJKVCEHYALYMMUKG
Keystream 2 is: OSLOOSLOOSLOOSLOOSLOOSLOOSLOOSLOOSLOOSLOOSLO
Ciphertext is: HZPEIANYPJZKBXZLXMXDGGGSFLSSZSKMRGR

DECRYPTION:
Ciphertext 2 is: HZPEIANYPJZKBXZLXMXDGGGSFLSSZSKMRGR
Keystream 2 is: OSLOOSLOOSLOOSLOOSLOOSLOOSLOOSLOOSLOOSLOOSLO
Ciphertext 1: SDKYGOEKEVDGMBUFVAOPVSKOQPNMXGBYGSV
Keystream 1 is: NORWAYNORWAYNORWAYNORWAYNORWAYNORWAYNORWAYNO
Plaintext is: THEQUICKBROWNFOXJUMPSOVERTHELAZYDOG
```

```cpp
#include <iostream>
#include <string>
using namespace std;

class VigCipher {
private:
    int counter;
public:
    string k;
    VigCipher(string k); //constructor
    string keyStream(string key, string message); //increases the size of the key to match the length of pt
    string encryption(string plaintext);          //encrypts the plaintext
    string decryption(string ciphertext);         //decrypts the ciphertext

};

VigCipher::VigCipher(string k){  //generate key

    for (int i = 0; i < k.size(); ++i) {
        if (k[i] >= 'A' && k[i] <= 'Z') //ignore what aren't letters, and capitalize everything.
            this->k += k[i];
        else if (k[i] >= 'a' && k[i] <= 'z')
            this->k += k[i] + 'A' - 'a';
    }
}

string VigCipher::keyStream(string key, string message){
    int temp = message.size();
    for (int i = 0; ; i++)
    {
        if (temp == i)
            i = 0;
        if (key.size() == message.size())
            break;
        key.push_back(key[i]);
    }
```

```
        key.push_back(key[i]);
    }
    return key;
}


string VigCipher::encryption(string plaintext){
    string encrypt;
    for (int i = 0, j = 0; i < plaintext.length(); ++i){
        char p = plaintext[i];
    if (p >= 'a' && p <= 'z')
        p += 'A' - 'a';
    else if (p < 'A' || p > 'Z')
        continue;
        encrypt += (p + k[j] - 2 * 'A') % 26 + 'A';
        j = (j + 1) % k.length();
    }
    return encrypt;
}

    string VigCipher::decryption(string ciphertext){
        string decrypt;
        for (int i = 0, j = 0; i < ciphertext.length(); ++i)
        {
            char c = ciphertext[i];

            if (c >= 'a' && c <= 'z')  //ignore what aren't letters, and capitalize everything.
                c += 'A' - 'a';
            else if (c < 'A' || c > 'Z')
                continue;

            decrypt += (c - k[j] + 26) % 26 + 'A';  //the actual decryption happens here
            j = (j + 1) % k.length();
        }

        return decrypt;
```

```
int main()
{
    /*
    VigCipher vigenere("Kongsberg");
    string plaintext = "The quick brown fox jumps over the lazy dog.";
    string encryption = vigenere.encryption(plaintext);
    string decryption = vigenere.decryption(encryption);
    string s = vigenere.keyStream(vigenere.k, plaintext);

    cout <<"ENCRYPTION:" <<endl;
    cout << "Plaintext: "<< plaintext << endl;
    cout << "The keystream is: " << s << endl;
    cout << "Ciphertext: " << encryption << endl << endl;

    cout <<"DECRYPTION:" <<endl;
    cout <<"Ciphertext: " << "DVRWMJGBHBCJTXPBAAWDFUNFVKNOZNFQESX" << endl;
    cout << "The keystream is: " << s << endl;
    cout << "Plaintext: " << decryption << endl;
     */

    VigCipher vigenere ( k: "Norway");
    VigCipher vig( k: "Oslo");
    string plaintext = "The quick brown fox jumps over the lazy dog.";
    string encryption = vigenere.encryption(plaintext);
    string decryption = vigenere.decryption(encryption);
    string encrypt = vig.encryption(plaintext);
    string decrypt = vig.decryption(encryption);
    string s = vigenere.keyStream(vigenere.k, plaintext);
    string sa = vig.keyStream(vig.k, plaintext);

    cout <<"ENCRYPTION:" <<endl;
    cout << "Plaintext 1 is: "<< plaintext << endl;
    cout << "Keystream 1 is: " << s << endl;
    cout << "Plaintext 2 is: " << encryption << endl;
    cout << "Keystream 2 is: " << sa << endl;
    cout << "Ciphertext is: " << encrypt << endl << endl;
```

```
    cout <<"DECRYPTION:" <<endl;
    cout <<"Ciphertext 2 is: " << "HZPEIANYPJZKBXZLXMXDGGGSFLSSZSKMRGR" << endl;
    cout << "Keystream 2 is: " << sa << endl;
    cout << "Ciphertext 1: " << decrypt << endl;
    cout << "Keystream 1 is: " << s << endl;
    cout << "Plaintext is: " << decryption << endl;
}
```

Q3

Confusion:
Confusion refers to making the relationship between the key and the ciphertext as complex and involved as possible. It is an encryption operation where the relationship between the key and ciphertext is obscured. Example: Today, a common way of achieving confusion is substitution - found in both AES and DES.
Here we have a look-up table with entries, numbers. Basically, for a certain input, you get a certain output.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

This is a good thing to do if you want to build a strong cipher, but it's not enough which makes sense because we know that Caesar cipher by itself isn't strong. From that reasoning, we need something else, and this is where diffusion comes in.

Diffusion:
Diffusion is an encryption operation where the influence of one plaintext symbol is spread of over many ciphertext symbols with the goal of hiding statistical properties of the plaintext. Example: A simple diffusion element is the bit permutation, which is frequently used within DES. The principles of confusion and diffusion are the most essential concepts in the design of modern block ciphers because they defend against statistical attacks

Along with the lecture notes I used these references for the assignment:

References for q1:
https://whatis.techtarget.com/definition/Confidentiality-integrity-and-availability-CIA
https://blog.netwrix.com/2019/03/26/the-cia-triad-and-its-real-world-application/
https://en.wikipedia.org/wiki/ROT13
https://www.geeksforgeeks.org/rot13-cipher/

References for q2:
https://www.geeksforgeeks.org/vigenere-cipher/
https://www.tutorialspoint.com/cplusplus-program-to-implement-the-vigenere-cypher
https://cryptii.com/pipes/vigenere-cipher

References for q3:
https://www.geeksforgeeks.org/difference-between-confusion-and-diffusion/
https://www.youtube.com/watch?v=aH6h52aMGT8

Link to code: https://github.com/azimiqbal/VigenereCipher/tree/main