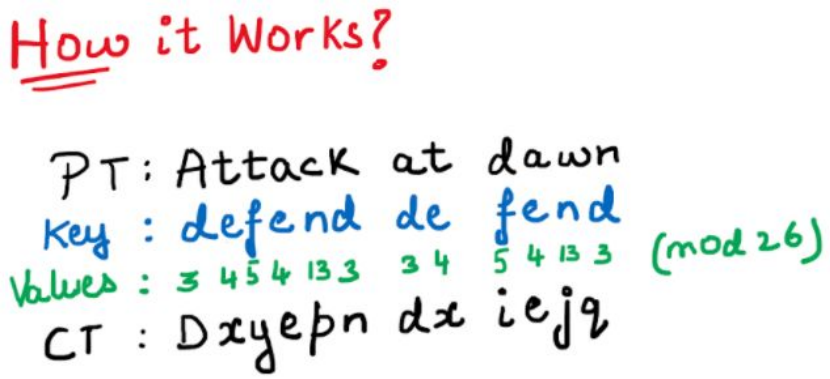
REPORT ON PROGRAMMING ASSIGNMENT NO-1

Project-1

By: Dheeraj (2017044) and Sehaj Singh (2017099)

We have developed executable programs to encrypt, decrypt, and more importantly use a brute-force attack to discover the key.
We used 5 Different classes for simplicity and neatness of the code. The main driver code file is VigenereCipher.java. This simulated the working of the whole scenario where a sender sends the ciphertext to a receiver (who decrypts it) and the hacker gets hold of that ciphertext from the exposed medium of communication.
Here is the rough idea of how the encryption algorithm works: -



Encryption algorithm used by the sender:

$C_i = (P_i + K_i) \bmod 26$

```
public String encode(){
    StringBuilder cipherText= new StringBuilder();

    int i=0; // counter for key
    for(char ch : plainText.toCharArray()){
        if(ch!=' ') {
            char ciph= shiftLetter(ch, key.charAt(i));
            cipherText.append(ciph);
            i=(i+1)%key.length();
        }
        else{
            cipherText.append(' ');
        }
    }

    return cipherText.toString();
}
```

```
private char shiftLetter(char ch, char k){
        char shift;
        if(ch<97){
            int pos=ch-65;
//              System.out.println(ch + " " +  pos);
            shift= Character.toUpperCase(arr[(pos+ (k-97))%26]);
        }
        else{
            int pos=ch-97;
            shift= arr[(pos+ (k-97))%26];
        }
        return shift;
    }
```

Decryption algorithm used by the Receiver:

$P_i = (C_i - K_i + 26) \bmod 26$

```
public String decode (){
    StringBuilder plainText= new StringBuilder();

    int i=0; // counter for key
    for(char ch : cipherText.toCharArray()){
        if(ch!=' ') {
            char plain= shiftLetter(ch, key.charAt(i));
            plainText.append(plain);
            i=(i+1)%key.length();
        }
        else{
            plainText.append(' ');
        }
    }

    return plainText.toString();
}

private char shiftLetter(char ch, char k){
        char shift;
        if(ch<97){
            int pos=ch-65;
            int computed=(pos - (k-97))%26;
//              System.out.println(ch + " " +  pos +  " " + computed);
            shift= Character.toUpperCase(arr[computed<0?computed+26:computed]);
        }
        else{
            int pos=ch-97;
            int computed=(pos - (k-97))%26;
            shift= arr[computed<0?26+computed:computed];
        }
```

```
        return shift;
    }
```

What does the hacker have?

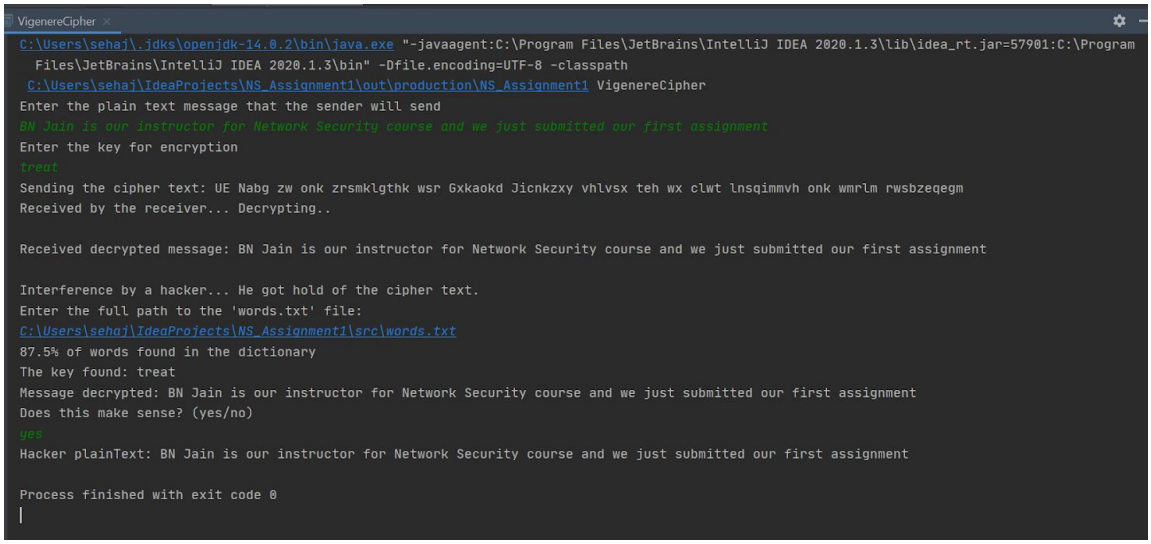
The hacker has access to a dictionary that contains all dictionary words and the ciphertext he got hold of in the exposed medium. These dictionary words can be extracted in the code through a text-file submitted (words.txt).

We have also put some additional efforts such as while the brute force attack executes, it checks all the key lengths from 1 to 6. There is a prompt that asks if the plaintext makes sense (grammatically) or not, if yes then the program exits and if not, it will further search for more keys.

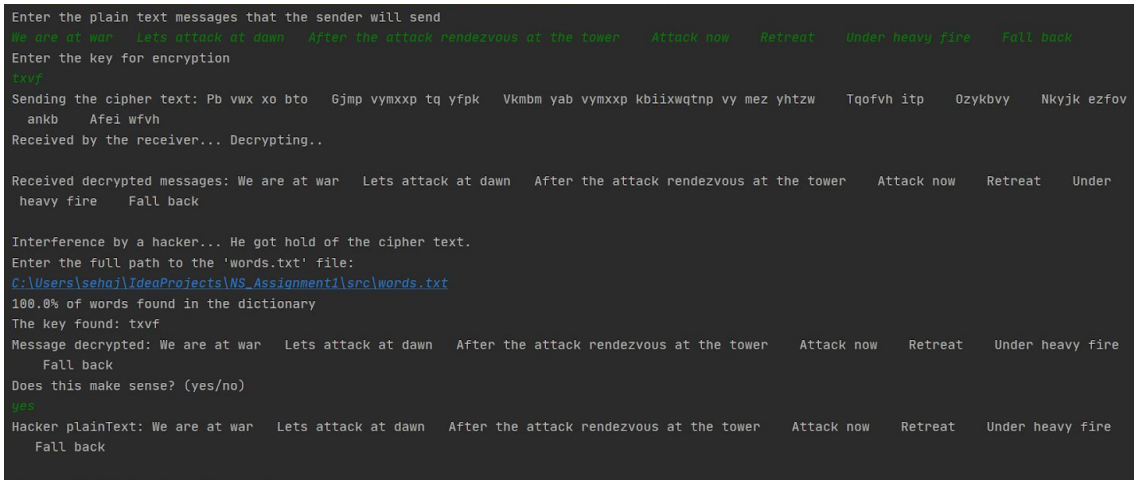
How does the hacker use his/her resources?

The hacker generates keys from length 1 to length 6 and checks all possible keys. The complexity of this attack is in the order of $O(26^6)$. He/she uses a dictionary to check if the words that are decrypted, by every generated key, are present in the English dictionary or not. Now, there is a corner case possibility that not all the words that are in the plaintext are also present in the dictionary. For example:

plaintext= “BN Jain is our instructor for the Network security course and we have just submitted our first assignment”. Now as we can see, words like “BN” and “Jain” may not be in our dictionary. Similarly, if this plaintext had our names, “Sehaj” and “Dheeraj”, they would not be in the English dictionary. So to counter this problem, there is a relaxation given to the percentage of words that are checked in the dictionary while decrypting. If more than 68% of words are present in the dictionary, we are going good and this deciphered plaintext has a high probability that it will be the correct plaintext and may have some names of individuals who are not in the hacker’s dictionary. Below is the sample run of this test-case:



More Sample Runs:



Plaintexts = We are at war Lets attack at dawn After the attack rendezvous at the tower Attack now Retreat Under heavy fire Fall back