NETWORK SECURITY
ASSIGNMENT - 1

Manavjeet Singh 2018295
Rhythm Gupta 2018082

Project 0: Encryption & decryption using monoalphabetic substitution of 2 characters at a time from
the set of symbols {A, B, C} using a table consisting of 3^2 rows of tuples of the kind <2 character plaintext, 2 character ciphertext>) , e.g. AA -> BA, AB -> BC, etc. of the kind discussed in class. Then
develop the software to launch

MONO-ALPHABETIC SUBSTITUTION CIPHER

Mono Alphabetic substitution cipher is a simple cipher where during encryption, we substitute alphabets with their respective substitution alphabets (as given in key) and for each occurrence of an alphabet in plaintext the same substitution alphabet is used in ciphertext.

However, such a system is easy to break using bruteforce since there are only 26! Possibilities of keys that too can be reduced using frequency analysis.

MULTIPLE-SYMBOL SUBSTITUTION CIPHER

This is similar to the above mentioned scheme but instead of replacing an alphabet by its substitution alphabet, we replace the combination of alphabets with a combination of substitution alphabets . So if combination size is n, the total number of possible keys are $(26^n)!$.

In this project, our symbol set is {A, B, C} and the combination size is 2 so we have a total 9! Possible keys.

GENERATION OF KEYS

First of all, we associate an index for each combination of symbols. Our Indices are as follows:

AA	0	BB	3	CC	6
AB	1	BA	4	CA	7
Ac	2	BC	5	CB	8

Next we make an array as follows
key = ['AA', 'AB', 'AC', 'BB', 'BA', 'BC', 'CC', 'CA', 'CB']
And then shuffle this array.
Lets assume shuffle is as follows:

key k = ['AC', 'BC', 'CC', 'AA', 'BA', 'CA', 'AB', 'BB', 'CB']

So, we will replace plaintext p with key [index(p)] that is a substitution present at index of p.

For example AB will be replaced by BC.

This key is used for encryption. For decryption purpose, we prepare key k2 which is inverse of k in following manner:

K2 [i] = p such that i = index(k [index (p)])

So K2 [5] = AB since 5 = index(k[1])

ENCRYPTION

Let p be a pair of characters and P be the plaintext.
E(p, k) = k [index(p)]

P = p1 || p2 || p3 ….. p_n
E(P, k) = E (p1, k) || E (p2, k) || ….. || E(p_n , k)

If P is of odd length, E(P, k) = E(P + ‘C’ , k)

DECRYPTION

Let p be a pair of characters and P be the plaintext.
D(c, k2) = k2 [index(p)]

C = c1 || c2 || c3 ….. c_n
D(C, k2) = D (c1, k2) || D (c2, k2) || ….. || D(c_n , k2)

By our encryption process, the encrypted text is always even length but it might be an encryption of some odd length text with an alphabet concatenated at last. So, both D(C,k2) and D(C , k2) [:-1] can be possible plaintexts.

VERIFYING DECRYPTION

We verify our decryption is correct by ensuring D(E(p)) = p

CRYPTANALYSIS (BRUTEFORCE)

a. Verifying a Key

For the purpose of verifying a key during bruteforce, we use a specific format while encryption. While encrypting a plaintext p , we use following function:

E2(p, k) = E(p, k) || Hash(p) (where || is the concatenation function)

We use MD5 for hashing.
MD5 hash of any string is 32 hexadecimal bytes long.

D2(c, k) = D(c[:32] , k) || c[32:]

b. Process

Let S be the set of ciphertexts we have for bruteforce and P be the set of all possible keys.

For k in P:

    For c in S:

        If (Hash(D2(c [: -32] , k)) = c[-32:] or Hash(D2(c [: -33] , k)) = c[-32:]) :

            Then key k is a candidate key

        Else:

            Key k can’t be a candidate key

    If key k works for all the ciphertexts c in S, then k is the key we were looking for.

Examples

Encryption Key: ['CA', 'CC', 'BA', 'BB', 'CB', 'AA', 'BC', 'AC', 'AB']
Decryption Key: ['BC', 'CB', 'CA', 'BB', 'AC', 'CC', 'AB', 'AA', 'BA']

Plaintext	**CipherText**
ABCABC	CCACAA6a1423d3711726df945da02a8ebf0f59
BACBBACBBBAC	CBABCBABBBBA6a28cfb93f10e2ace34eb735b844b599
CCCCABACCABC	BCBCCCBAACAAd1403053859f1b0723aa1d128161b955
BBACBAABCBAC	BBBACBCCABBAe466520675d48df362b5594e29676a08
CABCABCCAB	ACAACCBCCCf659551a9d9dbdc2ee8b679

	07958a7d5
AACBAABCACBC	CAABCAAABAAAd70fb06e2bbfe77f3466f0a3907e4792
ACBCBCBCBBAC	BAAAAAAABBBAd083b34703df056893227167c5d6ebfc

CipherText	**PlainText**
CCACAA6a1423d3711726df945da02a8ebf0f59	ABCABC
CCACAA6a1423d3711726df945da02a8ebf0f59	BACBBACBBBAC
CBABCBABBBBA6a28cfb93f10e2ace34eb735b844b599	CCCCABACCABC
BCBCCCBAACAAd1403053859f1b0723aa1d128161b955	BBACBAABCBAC
BBBACBCCABBAe466520675d48df362b5594e29676a08	CABCABCCAB
ACAACCBCCCf659551a9d9dbdc2ee8b67907958a7d5	AACBAABCACBC
BAAAAAAABBBAd083b34703df056893227167c5d6ebfc	ACBCBCBCBBAC

ABOUT CODE USAGE

Our code is based on python3.
We have following files:

File Name	Contents
encrypt.py	Contains Encryption Algorithm
decrypt.py	Contains Decryption Algorithm
model.py	Generates random key
bruteforce.py	Performs bruteforce on given ciphertexts
main.py	Contains ui for interacting with system

To run, enter python3 main.py on the terminal after navigating to the code's directory.