Tristan Erney

September 8th, 2021

Intro to Cryptology

Hands On Exercise 2

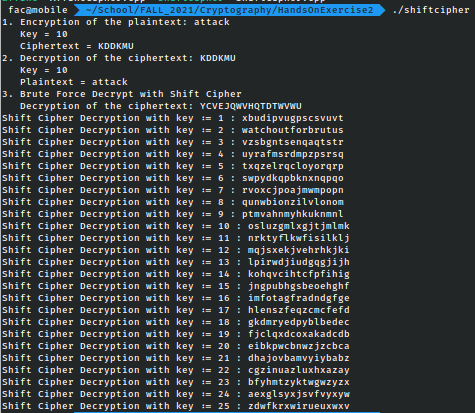
1)

a) “MTBID” => “howdy”

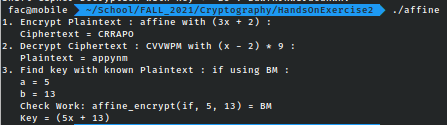
b) “MBIZDYVYQISCPEX” => “cryptologyisfun”

key = 10

2)



3)



Shiftcipher.cpp

#include <string>   
#include <iostream>   
#include <iomanip>   
  
// since C/C++ modulus doesn't handle negatives well, I have implemented a modulus function   
// such that negative numbers will be handled appropriately.   
int mod(int b, int m) {

if (b < 0) {   
 while (b < 0) b += m;   
 return b;   
 }   
  
 return b % m;   
}   
  
std::string shift\_encrypt(std::string, int);   
std::string shift\_decrypt(std::string, int);   
void shift\_brute\_force(std::string);   
  
int main() {   
 std::string plaintext = "attack";   
 std::string ciphertext = " ";   
 int key = 10;   
  
 ciphertext = shift\_encrypt(plaintext, key);   
 std::cout << "1. Encryption of the plaintext: " << plaintext << "\n";   
 std::cout << " Key = 10\n";   
 std::cout << " Ciphertext = " << ciphertext << "\n";

plaintext = shift\_decrypt(ciphertext, key);   
 std::cout << "2. Decryption of the ciphertext: " << ciphertext << "\n";   
 std::cout << " Key = " << key << "\n";   
 std::cout << " Plaintext = " << plaintext << "\n";   
  
 ciphertext = "YCVEJQWVHQTDTWVWU";   
 std::cout << "3. Brute Force Decrypt with Shift Cipher\n";   
 std::cout << " Decryption of the ciphertext: " << ciphertext << "\n";   
 shift\_brute\_force(ciphertext);   
  
 return 0;   
}   
  
// encrypt the given plaintext with the provided shift key.   
std::string shift\_encrypt(std::string plaintext, int shift) {   
 std::string e = "";   
  
 for (char c: plaintext) {   
 // c - 'a' will give us the value of the character in the range of [0..25] which is what we want.   
 // we then add our shift to the value found above.   
 // modulus by 26 to bring back the value to the range of [0..25] in case we exceeded it.   
 // we then finish off by adding 'A' back on to get our character in the range of ['A'..'Z'].   
 e += mod(((c - 'a') + shift), 26) + 'A';   
 }   
  
 return e;   
}   
  
// decrypt the given ciphertext with the provided shift key.   
std::string shift\_decrypt(std::string ciphertext, int shift) {   
 std::string p = "";   
  
 for (char c: ciphertext) {   
 // c - 'A' will give us the value of the character in the range of [0..25] which is what we want.   
 // we then subtract our shift to the value found above.   
 // modulus by 26 to bring the value back to the range of [0..25] in case we exceed it (or go negative).   
 // we then finish off by adding 'a' back on to get our character in the range of ['a'..'z'].   
 p += mod(((c - 'A') - shift), 26) + 'a';   
 }   
  
 return p;   
}   
  
// decrypt the cyphertext using all keys and display the new plaintext and key associated with it.   
void shift\_brute\_force(std::string ciphertext) {   
 for (int key = 1; key < 26; key += 1) {   
 std::cout << "Shift Cipher Decryption with key := " << key << " : ";   
 std::string p = shift\_decrypt(ciphertext, key);   
 std::cout << p << "\n";   
 }   
}

Affinecipher.cpp

#include <iostream>   
#include <string>   
  
// since C/C++ modulus doesn't handle negatives will, I have implemented a modulus function   
// in a way that handles negative modulus appropriately.   
int mod(int b, int m) {   
 if (b < 0) {   
 while (b < 0) b += m;   
 return b;   
 }   
  
 return b % m;   
}   
  
std::string affine\_encrypt(std::string, int, int);   
std::string affine\_decrypt(std::string, int, int);   
int\* affine\_known\_plaintext(std::string, std::string);   
  
// gcd table which is to be used in junction with the mod26\_inverse\_table to lookup inverses   
// without having to calculate.   
static int mod26\_gcd\_table[] = {   
 1, 3, 5, 7,   
 9, 11, 15, 17,   
 19, 21, 23, 25   
};   
  
// mod26 inverses of the mod26\_gcd\_table   
static int mod26\_inverse\_table[] = { \

1, 9, 21, 15,   
 3, 19, 7, 23,   
 11, 5, 17, 25   
};   
  
int main() {   
 int a = 3,   
 b = 2;   
  
 std::string plaintext = "affine";   
 std::string ciphertext = " ";   
  
 ciphertext = affine\_encrypt(plaintext, a, b);   
 std::cout << "1. Encrypt Plaintext : " << plaintext << " with (" << a << "x + " << b << ") : \n";   
 std::cout << " Ciphertext = " << ciphertext << "\n";   
  
 ciphertext = "CVVWPM";   
 int inv = -1;   
 for (int i = 0; i < 12; i += 1) {   
 if (mod26\_gcd\_table[i] == a) {   
 inv = mod26\_inverse\_table[i];   
 break;   
 }   
 }   
 plaintext = affine\_decrypt(ciphertext, inv, b);   
 std::cout << "2. Decrypt Ciphertext : " << ciphertext << " with (x - " << b << ") \* " << inv << " : \n";   
 std::cout << " Plaintext = " << plaintext << "\n";   
  
 int\* affine\_alpha\_beta = affine\_known\_plaintext("if", "BM");   
 std::cout << "3. Find key with known Plaintext : if using BM : \n";   
 std::cout << " a = " << affine\_alpha\_beta[0] << "\n";   
 std::cout << " b = " << affine\_alpha\_beta[1] << "\n";   
 std::cout << " Check Work: affine\_encrypt(if, " <<   
 affine\_alpha\_beta[0] <<   
 ", " <<   
 affine\_alpha\_beta[1] <<   
 ") = " <<   
 affine\_encrypt("if", affine\_alpha\_beta[0], affine\_alpha\_beta[1]) << "\n";   
 std::cout << " Key = (" << affine\_alpha\_beta[0] << "x + " << affine\_alpha\_beta[1] << ")\n";   
  
 return 0;   
}   
  
// affine encryption of plaintext which uses the function (ax + b) to encrypt plaintext.   
std::string affine\_encrypt(std::string plaintext, int a, int b) {   
 std::string e = "";   
  
 for (char c: plaintext) {   
 int x = (c - 'a');   
 int y = (a \* x) + b;   
 e += mod(y, 26) + 'A';   
 }   
  
 return e;   
}   
  
// affine decryption of ciphertext which uses the function y = (x - b) \* inv(a)   
std::string affine\_decrypt(std::string ciphertext, int inv, int b) {   
 std::string p = "";   
  
 for (char c: ciphertext) {   
 int x = (c - 'A');   
 int y = (x - b) \* inv;   
 p += mod(y, 26) + 'a';   
 }   
  
 return p;   
}   
  
// known plaintext decypher which uses a known plaintext and ciphertext pair to solve for   
// a and b.   
int\* affine\_known\_plaintext(std::string plaintext, std::string ciphertext) {   
 int alpha, beta;   
  
 // get the base values of ciphertext by subtracting 'A' from both.   
 int y = mod(((ciphertext[0] - 'A') - (ciphertext[1] - 'A')), 26);   
 // get the base value of plaintext by subtracting 'a' from both.   
 int x = ((plaintext[0] - 'a') - (plaintext[1] - 'a'));   
  
 // find alpha first   
 // get inverse from inverse\_table   
 int inv;   
 for (int i = 0; i < 12; i += 1) {   
 if (mod26\_gcd\_table[i] == x) {   
 inv = mod26\_inverse\_table[i];   
 break;   
 }   
 }   
  
 alpha = mod(inv \* y, 26);   
  
 // find beta   
 beta = mod((ciphertext[0] - 'A') - (alpha \* (plaintext[0] - 'a')), 26);   
  
 int\* ret = (int\*)malloc(sizeof(int) \* 2);   
 ret[0] = alpha;   
 ret[1] = beta;   
 return ret;   
}