****

**EE 4550L**

**IC Hardware Security and Trust LAB**

**SPRING 2024**

**TA: Kanchan Vissamsetty**

**Lab section: 01**

**Name: Alex Yeoh**

**“I have neither given nor received aid on this assignment, nor have I observed any violation of the Honor code”**

**Signature: Alex Yeoh Date: 21st March 2024**

**Report due date: 22nd March 2024**

1. **OBJECTIVE**

To understand the basic concepts of encryption.

1. **PROCEDURE**

Write code in a C coding language that encrypts a plaintext with the Caesar Cypher encryption algorithm. Take screenshots of the given code with inputs of various specified lengths.

1. **RESULT**

Part I

1. encryption and decryption for the example plaintext and key

A black background with white text

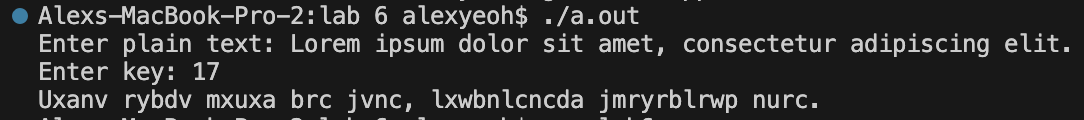
Description automatically generatedEncryption with the example plaintext and key

A black background with white text

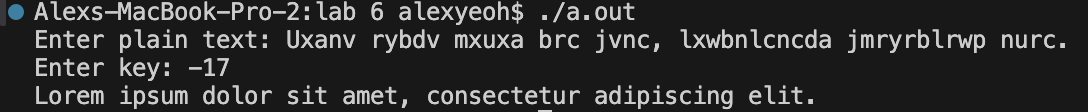
Description automatically generated

Decryption with the example plaintext and key

1. Uxanv rybdv mxuxa brc jvnc, lxwbnlcncda jmryrbirwp nurc.
2. encryption and decryption for the previous question



Encryption for the previous question



Decryption for the previous question

1. Code for the Caesar Cypher

|  |
| --- |
| 1. /\* 2. How to improve the Caeser Cypher 3. This could by done by having the key be used as a seed to generate random numbers to shift each character by. 4. We could also remove the limitation of needing the letters remain letters. 5. We could also allow the punctuation to also rotate with the letters. 6. \*/ 7. #include <iostream> 8. #include <string> 9. int main() 10. { 11. // reading in plain plainText from user 12. std::string plainText; 13. std::cout << "Enter plain text: "; 14. getline(std::cin, plainText); 15. char cypherText[plainText.length() + 1]; 16. // for (int i = 0; i < plainText.size(); i++) 17. // { 18. // cypherText[i] = plainText[i]; 19. // } 20. strcpy(cypherText, plainText.c\_str()); 21. // reading in key from user 22. int key; 23. std::cout << "Enter key: "; 24. std::cin >> key; 25. /\* 26. \* 0 = punctuation (default assumption) 27. \* 1 = upper case 28. \* 2 = lower case 29. \*/ 30. int changeArr[sizeof(cypherText) + 1]; 31. std::fill\_n(changeArr, sizeof(cypherText), 0); 32. for (int i = 0; i < sizeof(cypherText); i++) 33. { 34. char check = cypherText[i]; 35. if (check >= 65 && check <= 90) 36. changeArr[i] = 1; 37. else if (check >= 97 && check <= 122) 38. changeArr[i] = 2; 39. } 40. // makes all upper case 41. for (int i = 0; i < sizeof(cypherText); i++) 42. { 43. if (changeArr[i] == 2) 44. cypherText[i] -= 32; 45. } 46. int rotate = -key % 26; // reduce to minimum rotation amount 47. for (int i = 0; i < sizeof(cypherText); i++) 48. { 49. if (changeArr[i] == 0) 50. { 51. } 52. else 53. { 54. cypherText[i] += rotate; // rotate characters 55. char check = cypherText[i]; 56. // fix out of letter bounds 57. if (check < 'A') 58. cypherText[i] += 26; 59. else if (check > 'Z') 60. cypherText[i] -= 26; 61. } 62. } 63. // fixes uppercase 64. for (int i = 0; i < sizeof(cypherText); i++) 65. { 66. if (changeArr[i] == 2) 67. cypherText[i] += 32; 68. } 69. // printing out 70. std::cout << cypherText << "\n"; 71. return 0; 72. } |

C++ code for encrypting a given plaintext with Caesar Cypher encryption algorithm.

Part II

1. screenshot taken in AES Specific Challenge step 4  
   A screen shot of a computer

   Description automatically generated
   1. Cyphertext is the plaintext after it has been encrypted with an encryption algorithm.
2. screenshot taken in AES Specific Challenge step 7  
   A black screen with white text

   Description automatically generated
   1. The hex values of the plain text are also encrypted just like the plaintext is.
3. screenshot taken in AES Specific Challenge step 10  
   A black background with white text

   Description automatically generated
   1. The text does not match because there is a length limit of 16 characters in the C code.
   2. Yes, by increasing the number of rounds by changing various for loops to perform more rounds.
   3. Using 256-bit key is more secure than using a 192-bit key which is more secure than using a 128-bit key. Using longer keys, however, will make it take longer to encrypt and decrypt because more rounds will be required.
4. **CONCLUSION**

My results satisfy the requirements. It is likely possible to improve my C++ code to improve efficiency, but it is unlikely that it would result in better results other than running slightly faster. I have learned the basic concepts of encryption.