Experiment 2 Documentation

In this experiment we did not require the Haha Board. Since it needed a Linux system and I did not have any Linux system with me, I worked with a friend who had Ubuntu in his laptop.

In this experiment we learned how to encrypt data with standard encrypting algorithms, with a focus on using cryptography to protect sensitive data with a secret key.

We downloaded all of the required files in the Ubuntu system and started the experiment.

Part I: Caesar Cipher

Using the help of Caesar Cipher we did in CSC 101, we coded for it in C. Setting the plain text and Key (hardcoded).

A computer screen with text and images

Description automatically generated

Testing the code, we got the same result as mentioned in the PowerPoint so we knew we did the code in the correct way.

Few sentences for instruction on how to improve the Caesar Cipher.

To further improve the Caesar Cipher's security: Use a variable key that varies for every message rather than a constant shift value. As a result, it becomes more difficult for attackers to use frequency analysis to decode the code. Change the alphabet of the mainstream English language by moving or adding some letters. Because of this change, attackers are unable to depend solely on conventional letter frequencies, which complicates decryption attempts. Utilize several shifts in a single message. To develop a more complex encryption technique, combine the Caesar Cipher with additional substitution ciphers. Using a pseudorandom number generator or integrating a real random source, add a degree of unpredictability to the shift results.

Part II: AES Specific Challenge

After downloading the files as required for this, we used the terminal to execute the following commands given to us. At first we compiled the project and ran it.

A computer screen with a web browser

Description automatically generated

Difference between Plaintext and Ciphertext

Plain text refers to the original, unencrypted and readable form of the data. And Ciphertext in the encrypted form of the message given to us using a cryptographic algorithm or key given to us. Plaintext is transformed or encrypted into Ciphertext using the encryption menthod or key.

After that process we changed the data in unsigned char msg[] = Gainesville. Compiled the file and ran it like we previously did.

A computer screen with a black and white text

Description automatically generated

In the provided C program, AES encryption and decryption are performed on a plaintext message, and the program outputs the hexadecimal values of the characters both before and after encryption and decryption.

Before Encryption the program prints the hexadecimal values of each character in the plaintext message `msg`. This representation shows the ASCII values of the characters in hexadecimal format. Since `msg` is a string, these values correspond to the ASCII codes of the characters in `msg`.

After Encryption the `aes\_encrypt` function is called, the contents of `msg` are modified to represent the encrypted data. The program then prints these new values, which are the hexadecimal representation of the ciphertext. AES encryption transforms the plaintext into a seemingly random series of bytes, so these values will appear random and bear no resemblance to the original plaintext values.

After decrypting the ciphertext using the `aes\_decrypt` function, the program prints the hexadecimal values of the decrypted text. If the encryption and decryption are correctly implemented and the correct key is used, these values should match the original plaintext values. This would confirm that the decryption process successfully reverted the ciphertext back to the original plaintext.

After that process, I changed the data to “universityoffloridauniversityofflorida” and compiled the file again.

A computer screen with text on it

Description automatically generated

In the program, the text before and after encryption does not match, and this is expected and intentional.

The program prints the original plaintext, which is the message "universityoffloridauniversityofflorida". The plaintext is encrypted using the AES algorithm, resulting in ciphertext. This ciphertext is essentially a scrambled version of the original plaintext, generated using the AES encryption algorithm and the provided key. The ciphertext appears as a series of seemingly random hexadecimal values, which do not resemble the original plaintext's hexadecimal values. Ideally, after decryption, the text should match the original plaintext. However, in this program, there are several issues that may prevent successful decryption:

The size of the `msg` array and the `ci` array is not properly managed. AES operates on blocks of 16 bytes.Encrypted data might contain non-printable characters.

To change the key length in AES to 192 or 256 bits, we would need to modify several aspects of the implementation: Modify the `ikey` array to hold 24 bytes (192 bits) or 32 bytes (256 bits) instead of the current 16 bytes (128 bits). The key values should be appropriately set.

.

Advantages and Disadvantages of 192 and 256-bit Key Lengths

192-bit Keys Advantages: Offers a higher level of security compared to 128-bit keys. It's less computationally intensive than 256-bit encryption, making it faster while still providing robust security.

Disadvantages: Slower than 128-bit encryption. Offers less security than 256-bit keys. It is less commonly used, which might lead to compatibility issues in some systems.

256-bit Keys Advantages: Provides the highest level of security among the standard AES key sizes. It's more resilient against brute-force attacks due to the larger key size.

Disadvantages: More computationally intensive, leading to slower encryption/decryption processes compared to 128-bit and 192-bit keys. This might be a concern in environments where speed is critical or where resources are limited.

**Expanding on experiment 2**

To understand the problem properly of why and how the encryption works the way it does. I tried with various input lengths to test the experiment.

In simple terms, the input size that will be correctly encrypted and decrypted in the provided program should be exactly 16 bytes. This is because the AES algorithm, as used in this program, works on blocks of data that are 16 bytes (or 128 bits) in length. The program takes the first 16 bytes of the input message `msg` for encryption, and any data beyond that is not processed. If the input message is shorter than 16 bytes, the behavior is not defined in the provided code, as it does not include padding logic to handle messages that are not exactly 16 bytes long. So, to ensure correct encryption and decryption, your input message should be exactly 16 bytes in length.

Too Short (Less than 16 Bytes):

- If the input message is shorter than 16 bytes, the AES algorithm still processes a full 16-byte block. However, because the program doesn't implement any padding mechanism, the contents of the memory following the end of your message (up to 16 bytes) will be included in the encryption. This could be anything, as it's essentially uninitialized memory or residual data, leading to unpredictable results. Decrypting this data will also give unpredictable results because the extra bytes beyond the actual message were random or uninitialized.

Too Long (More than 16 Bytes):

- If the input message is longer than 16 bytes, only the first 16 bytes are processed by the AES algorithm in your current program. The rest of the message is ignored. This means it loses any data beyond the 16th byte, and only the first 16 bytes are encrypted and then decrypted.

In both cases, the integrity and security of the data are compromised:

Security Risks: Using uninitialized memory in encryption can lead to security risks, as it might inadvertently expose sensitive data in memory.

Data Loss: For messages longer than 16 bytes, losing part of the data makes the encryption/decryption process ineffective for practical use.

According to my research to handle these issues, we would typically need to implement a padding scheme for short messages and a mode of operation (like CBC or CFB) that allows processing longer data in multiple 16-byte blocks. Additionally, for any serious application, we would should ensure the proper and secure handling of keys and initialization vectors (IVs) if needed by the mode of operation.