Lab 2

Allen Jiang
alljiang@utexas.edu
UT Austin, USA

ACM Reference Format:
Allen Jiang. 2023. Lab 2. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

In this lab, we implement AES-256 and RSA encryption and decryption in C. The goal of this lab is to familiarize ourselves with implementing cryptography algorithms.

We are using CTR block cipher mode for AES-256 encryption and decryption. The S-box is provided for us, along with a generated key and IV.

2 Our Architecture

The implementation of AES-256 followed the NIST standard for AES-256. In the aes.c file, there are a few functions that are used to implement AES-256:

- calculate_round_keys(): generates all keys needed given a key and the s-box.
- encode(): encodes an input given plaintext, a key, round keys, and the s-box.
- decode(): decodes an input given ciphertext, a key, round keys, and the s-box. While this function is not used in this lab due to the use of CTR mode, it is still implemented and tested.

The implementation of RSA mode is primarily based on the mod_exponentiation() function. This function takes in a base, exponent, and modulus and returns the result of the exponentiation. It calculates the result of the exponentiation by using the square-and-multiply algorithm.

To calculate the result, it uses a uint256 struct to store the intermediate results. The uint256 struct is constructed by using an array of 8 32-bit unsigned integers.

In addition to this new type, there are also a few functions that were developed to support operations on the uint256 struct:

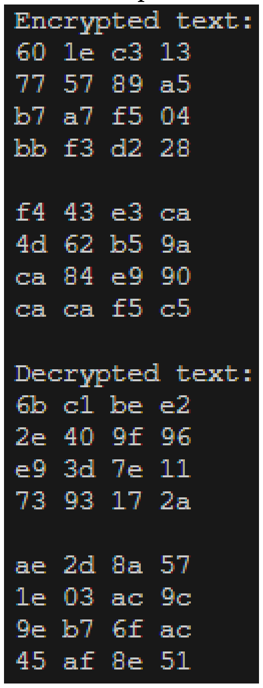
- compare(): compares two uint256 structs and returns a defined value based on the comparison.
- shift_left(): shifts a uint256 struct left by a given number of bits.
- shift_right(): shifts a uint256 struct right by a given number of bits.
- subtract(): subtracts two uint256 structs and returns the result.
- mod(): calculates the modulus of two uint256 structs and returns the result.
- multiply(): multiplies two uint256 structs and returns the result.
- convert_128_to_256(): converts a uint128 struct to a uint256 struct.
- convert_256_to_128(): converts a uint256 struct to a uint128 struct.

3 Experimental Results

The output of the aes.c program passes the given asserts:

- assert(memcmp(enc_buf, ciphertext[0], 32) == 0);
- assert(memcmp(decrypted_text, plaintext[0], 32) == 0);

The raw output of the aes.c program is:

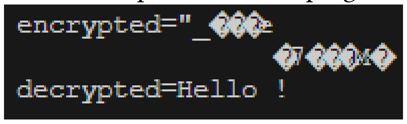


```
Encrypted text:
60 1e c3 13
77 57 89 a5
b7 a7 f5 04
bb f3 d2 28

f4 43 e3 ca
4d 62 b5 9a
ca 84 e9 90
ca ca f5 c5

Decrypted text:
6b c1 be e2
2e 40 9f 96
e9 3d 7e 11
73 93 17 2a

ae 2d 8a 57
1e 03 ac 9c
9e b7 6f ac
45 af 8e 51
```

The output of the rsa.c program passes the given assert:

- assert(strcmp(plaintext, decrypted_text) == 0);

The raw output of the rsa.c program is:



```
encrypted="_���e
                             �7���M�
decrypted=Hello !
```

4 Conclusions

The output of the aes.c program matches the expected output. This indicates that the AES implementation is correct.

One possible addition to this AES implementation is to integrate integrity checks. This can be done by computing the SHA-256 hash of the plaintext and including it as a header of the plaintext before encrypting the entire message. The receiver can then compute the SHA-256 hash of the decrypted plaintext without the header and compare it to the included hash header. If the hashes match, the integrity of the message is verified.

The decrypted output of the ciphertext output of the rsa.c program matches the original plaintext input. This indicates that the RSA implementation is correct.

To implement integrity checking for RSA, the sender can compute the SHA-256 hash of the plaintext and encrypt it with the private key. The receiver can then decrypt the hash with the public key and compare it to the SHA-256 hash of the plaintext. If the hashes match, the integrity of the message is verified.