Homework 3

ECE4802

Nam Tran Ngoc

1. AES

a. Write the given input to Hexadecimal form.

The binary input was converted to hexadecimal by hand:

0xC1, 0x19, 0xCC, 0x10, 0x56, 0x50, 0x0A, 0x5C, 0x3D, 0x06, 0xB7, 0x38, 0xA7, 0x34, 0xAA, 0x00E

b. Write the input in a state diagram (4x4 matrix)

For this part I plugged the values in a 2D array in `C`, along with a print function.

```
uint8_t INPUT[4][4] = {
    {0xC1, 0x19, 0xCC, 0x10},
    {0x56, 0x50, 0x0A, 0x5C},
    {0x3D, 0x06, 0xB7, 0x38},
    {0xA7, 0x34, 0xAA, 0x0E}
};
```

Which outputs:

```
0xc1 0x19 0xcc 0x10
0x56 0x50 0x0a 0x5c
0x3d 0x06 0xb7 0x38
0xa7 0x34 0xaa 0x0e
```

c. Use AES S-box to substitute the given input.

For this part I recreated the AES S-box and wrote a function in `C` that substitutes elements in the `INPUT` array.

```
uint8_t SBOX[16][16] = {

{0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,0x30,0x01,0x67,0x2b,0xfe,0xd7,0xab,0x76
},

{0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,0xad,0xd4,0xa2,0xaf,0x9c,0xa4,0x72,0xc0
},

{0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,0x34,0xa5,0xe5,0xf1,0x71,0xd8,0x31,0x15
},

{0x04,0xc7,0x23,0xc3,0x18,0x96,0x05,0x9a,0x07,0x12,0x80,0xe2,0xeb,0x27,0xb2,0x75
},
```

```
{0x09,0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,0x52,0x3b,0xd6,0xb3,0x29,0xe3,0x2f,0x84
},

{0x53,0xd1,0x00,0xed,0x20,0xfc,0xb1,0x5b,0x6a,0xcb,0xbe,0x39,0x4a,0x4c,0x58,0xcf
},

{0xd0,0xef,0xaa,0xfb,0x43,0x4d,0x33,0x85,0x45,0xf9,0x02,0x7f,0x50,0x3c,0x9f,0xa8
},

{0x51,0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,0xbc,0xb6,0xda,0x21,0x10,0xff,0xf3,0xd2
},

{0xcd,0x0c,0x13,0xec,0x5f,0x97,0x44,0x17,0xc4,0xa7,0x7e,0x3d,0x64,0x5d,0x19,0x73
},

{0x60,0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,0x46,0xee,0xb8,0x14,0xde,0x5e,0x0b,0xdb
},

{0xe0,0x32,0x3a,0x0a,0x49,0x06,0x24,0x5c,0xc2,0xd3,0xac,0x62,0x91,0x95,0xe4,0x79
},

{0xe7,0xc8,0x37,0x6d,0x8d,0xd5,0x4e,0xa9,0x6c,0x56,0xf4,0xea,0x65,0x7a,0xae,0x08
},

{0xba,0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,0xe8,0xdd,0x74,0x1f,0x4b,0xbd,0x8b,0x8a
},

{0x70,0x3e,0xb5,0x66,0x48,0x03,0xf6,0x0e,0x61,0x35,0x57,0xb9,0x86,0xc1,0x1d,0x9e
},

{0xe1,0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,0x9b,0x1e,0x87,0xe9,0xce,0x55,0x28,0xdf
},

{0x8c,0xa1,0x89,0x0d,0xbf,0xe6,0x42,0x68,0x41,0x99,0x2d,0x0f,0xb0,0x54,0xbb,0x16
}
};
```

Which outputs:

```
0x78 0xd4 0x4b 0xca
0xb1 0x53 0x67 0x4a
0x27 0x6f 0xa9 0x07
0x5c 0x18 0xac 0xab
```

2. Find the followings using Extended Euclidean Algorithm

For this I wrote a couple of C functions: `gcd()` which finds the Greatest Common Denominator of two numbers, `swap()` which swaps two numbers, and `eea()` which uses those two functions to find the multiplicative inverse of two numbers.

The code for this is as follow:

```
// Find multiplicative input using Extended Euclidean Algorithm
int eea(int a, int b) {
    // Initialize variables to use in eea
```

```
    int q,
        t,
        x0 = 0,
        x1 = 1,
        b0 = b; // Save original b for later

    // Not coprime
    if(gcd(a,b) != 1) {
        printf("%d and %d are not coprime, there's no multiplicative inverse\n",
a, b);
        return -1;
    }

    // Fringe case
    if(b == 1) {
        return 1;
    }

    // Extended Euclidean
    while(a > 1) {
        q = a / b;
        swap(&a, &b);
        b = b % a;
        swap(&x0, &x1);
        x0 = x0 - q * x1;
    }

    // Wrap around in modular space
    if(x1 < 0) {
        x1+= b0;
    }

    return x1;
}
```

```
// Helper function to swap 2 numbers
void swap(int *a, int *b) {
    int tmp = *b;
    *b = *a;
    *a = tmp;
}
```

```
// Recursive function to find gcd()
int gcd(int a, int b) {
    int rem = b%a;
    // We found gcd!
    if(rem == 0) {
        return a;
    }
    gcd(rem, a);
}
```

The output of the program:

```
17^-1 mod 37: 24
13 and 91 are not coprime, there's no multiplicative inverse
13^-1 mod 91: -1
13^-1 mod 448: 69
16^-1 mod 4725: 886
```

As you can see, `eea()` returns `-1` since it can't find a multiplicative inverse.

3. Computing RSA by hand

Alice wants to send Bob a message. Bob picks $p = 17$; $q = 29$; $b = 17$ as his initial parameters. Show all intermediate results for parts a, b, and c. You may use a calculator

a. Key generation

First, Bob must create his public and private keys. Compute N and $\varphi(N)$. Compute $a = b$ −1 mod $\varphi(N)$ using the extended Euclidean algorithm. What are Bob's public key (N; b) and private key(p; q; a)?

```
N    =  p * q         = 17 * 29 = 493
φ(N) = (p-1) * (q-1)  = 16 * 28 = 448
```

```
a = b^-1  mod φ(N)
a = 17^-1 mod 448
```

First we find the gcd(17,448)

```
448 = 17 * 26 + 6
17  = 6  *  2 + 5
6   = 5  *  1 + 1
5   = 1  *  5 + 0
```

We have `gcd(17, 448)` = 1

Now on to the Extended Euclidean Algorithm:

```
1 = 6 + 5(-1)
1 = 6 + (17 + 6(-2))(-1)
1 = 6 - 17 + 6(2)
1 = -17 + 6(3)
1 = -17 + (448 + 17(-26))(3)
1 = -17 + 448(3) + 17(-78)
1 = 448(3) + 17(-79)
```

We have `a = -79`, but since `-79 < 448`, we have `a = -79 + 448 = 369`

N **= 493**

$\varphi(N)$ **= 448**

Bob's public key : $(N; b)$ = (493, 17)

Bob's private key: $(p; q; a)$ = (17, 29, 369)

b. Encryption

Alice encrypts the message $X = 31$ using Bob's public key. Calculate the encrypted message by applying the square and multiply algorithm (first, transform the exponent to binary representation).

We have the encrypted message following the function `c = m^b mod N`, where `m` being the message and `c` being the encrypted message.

`c = 31^17 mod 493`

Applying the square and multiply algorithm

17 in binary is `0001 0001`

```
1       31
0       31^2         = 961       mod 493 = 468
0       468^2        = 219024    mod 493 = 132
0       132^2        = 17424     mod 493 = 169
1       169^2 * 31   = 885391    mod 493 = 456
```

So we have **c = 456**

c. Decryption

Bob decrypts Alice's encrypted message. Decrypt the ciphertext Y computed above by applying the square and multiply algorithm.

We have the decrypted message following the function `m = c^a mod N`, where `c` being the encrypted message and `m` being the decrypted message.

`m = 456^369 mod 493`

Applying the square and multiply algorithm

369 in binary is `0001 0111 0001`

```
1       456
0       456^2        = 207936    mod 493 = 383
1       383^2 * 456 = 66890184  mod 493 = 437
1       437^2 * 456 = 87081864  mod 493 = 316
1       316^2 * 456 = 45534336  mod 493 = 363
0       363^2        = 131769    mod 493 = 138
0       138^2        = 19044     mod 493 = 310
0       310^2        = 96100     mod 493 = 458
1       458^2 * 456 = 95652384  mod 493 = 31
```

So we have **m = 31**

d. Attack

Eve records the transmission of an RSA-encrypted message Y from Alice to Bob. Eve also knows the public key to be kpub = (493; 17). Your goal is to recover the message X that has been encrypted with RSA in part b.

- **i. Give the equation for the decryption of Y. Which variables are not known to Eve? Can Eve recover X? If so, how? If not, what would allow her to recover X?**

To recover message `m` from `Y` she will need `a`, a part of Bob's private key, following the function `m = Y^a mod 493`

Eve knows the public key, and values of `N = p * q`. Given enough resources, she could try a combination of all factors of 493 and recover `p` and `q` and solve for `a`.

- **ii. To recover the private key a, Eve has to compute a = b^−1 mod φ(N). Can Eve recover φ(N)?**

Yes. Eve can recover φ(N) by recovering `p` and `q` as factors of `N`, and find `ϕ(N) = (p-1)(q-1)`

- **iii. Compute the message X. (Hint: Start by factoring $N = p \cdot q$. Then use $\varphi(N)$ to compute a)**

Factorize `N`

```
N = 493

493/2  = 246.5
493/3  = 164.33
493/5  = 98.6
493/7  = 70.43
493/11 = 44.8
493/13 = 37.9
493/17 = 29
```

Therefore we get `p` and `q` to be 17 and 29.

We can then recover φ(N) to be 16 * 28 = 448.

We can then recover `a = b^-1 mod ϕ(N)`. `a = 369` as computed earlier.

- **iv. Can Eve do the same message recovery attack (as in (iii)) for large N, e.g., |N| = 1024 bit?**

At N = 1024 bit, the maximum value of N is 2^1024, which is not a feasible number to calculate and factorize with our current computing power. Other attack vectors must be considered.

- **v. Eve recovers a message-ciphertext pair (X; Y). Can she recover the private key a? If so, describe how. If not, why not?**

No. Eve cannot recover the private key A from the message-cipher text pair since she does not know N or φ(N), either of which is necessary to recover the private key `a`.