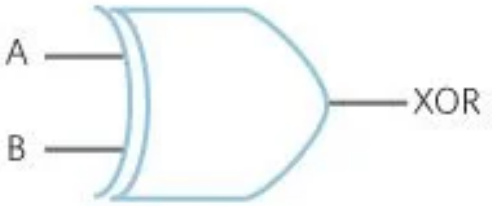
Basic Principles in Networking
Assignment 3 - Cryptography
Pair 29:
Nguyen Xuan Binh 887799
Nhut Cao 906939

Section 1: Goals of the experiment

In this experiment, we would create a symmetric encryption algorithm involving the logical operator XOR and a secret key, which is hardcoded into the Arduino program. The XOR operator is as follows

$$X = A \oplus B$$



A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

The XOR property is the symmetry between input and output. In other words, if A is the message and B is the key, then

- Original message xor key = encoded message
- Encoded message xor key = original message

The goal of this experiment is to input a message and receive an encoded message. Then the encoded message is fed to the input again and the original message is returned. This is called symmetric-key algorithm method in that the same secret key is used for both encoding and decoding the message

Section 2 : Experimental Setup (Details of the Experiment step by step)

Experiment setup has four parts:

- setup(), which runs only once when the program starts. This set up the Serial Monitor on the Arduino circuit.
- loop(), which runs indefinitely as it waits for input messages to be encoded/decoded. When the message is sent, the Serial Monitor encodes/decodes the message.
- establishContact(): when the program successfully starts, this will be run once and a message is sent to the Serial Monitor for program usage instructions.

- XORENC: helper function for loop() which carries out the actual encryption/decryption. The function works as follows: for each character in the string, it is xor-ed with each character in the key.

After setting up everything, we verify the sketch

```
void loop() {

  // put your main code here, to run repeatedly:

  // This is the XOR encrypt-decrypt symmetric algorithm

  // If you run the algorithm twice, you get back the original message

  // Example:    "Hello World" -> "^rxu};_Zlsh" -> "Hello World"
  String input = "";              // serial input character
  char key[10] = "secret";       // The key for xoring
  if (Serial.available() > 0){         // if you have data input
```

```
Done compiling.

Sketch uses 12108 bytes (4%) of program storage space. Maximum is 262144 bytes.
Global variables use 2400 bytes (7%) of dynamic memory, leaving 30368 bytes for local variables. Maximum is 32768 bytes.
```

=> Sketch verification is successful

After verification, we upload the sketch to the Arduino circuit.

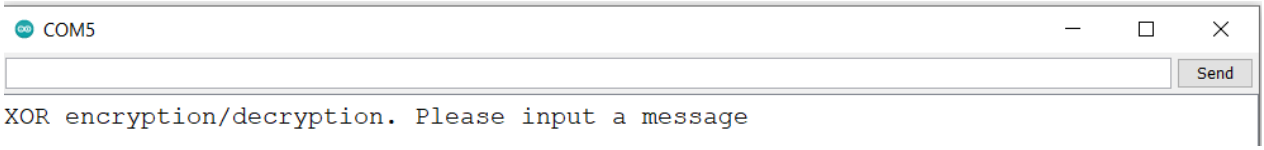
```
void loop() {

  // put your main code here, to run repeatedly:

  // This is the XOR encrypt-decrypt symmetric algorithm

  // If you run the algorithm twice, you get back the original message

  // Example:    "Hello World" -> "^rxu};_Zlsh" -> "Hello World"
  String input = "";              // serial input character
  char key[10] = "secret";       // The key for xoring
  if (Serial.available() > 0){         // if you have data input
```

```
Done uploading.

done in 0.092 seconds

Verify 12108 bytes of flash with checksum.
Verify successful
done in 0.011 seconds
CPU reset.
```

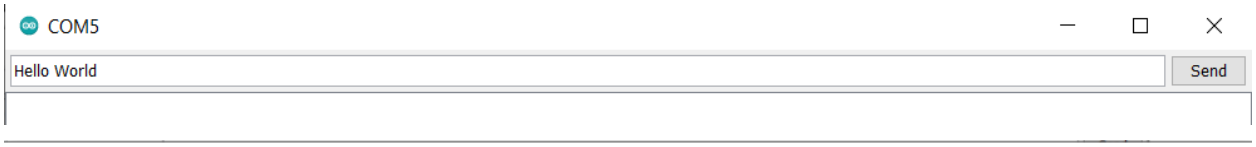
=> Sketch upload is successful

Section 3 : Results & Conclusion

Now we run the Serial Monitor and starts the encryption and decryption:

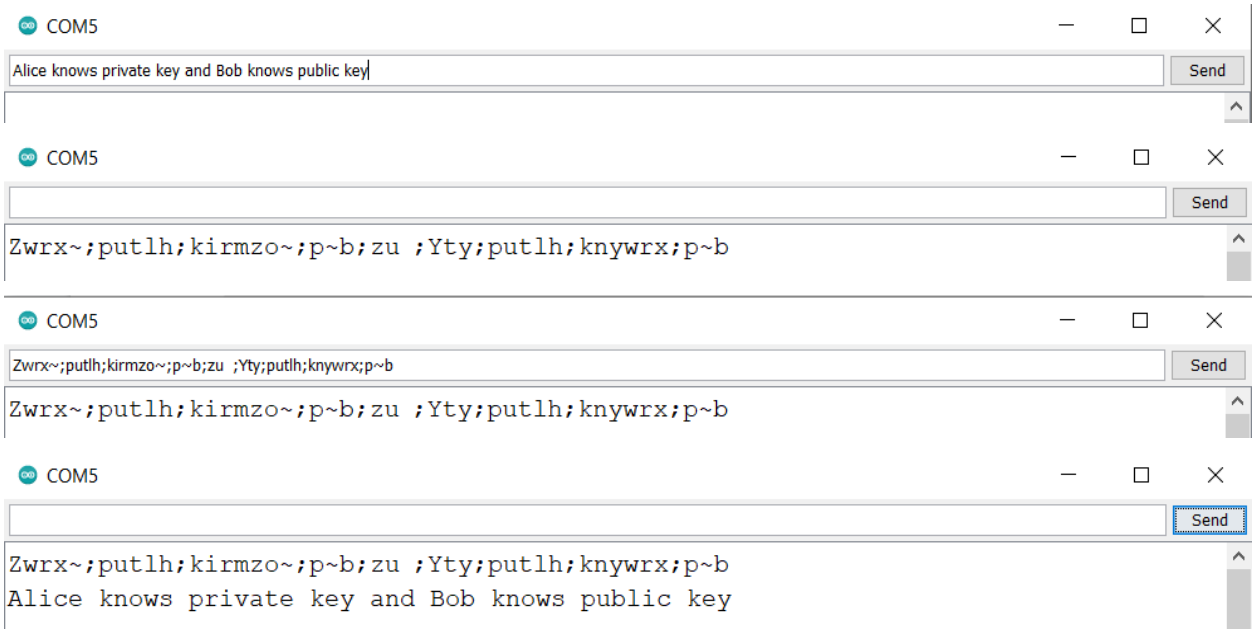


- Encrypt and decrypt the message “Hello World”



- Encrypt and decrypt the message “Computer Networks is fun”

- Encrypt and decrypt the message “Alice knows private key and Bob knows public key”



=> Conclusion: By a simple XOR logic and a key, we can implement a simple symmetric key encryption. Of course, the encrypted message will change depending on the secret key.

Section 4 : Annex of the symmetric key encryption sketch

```
// This is a XOR encryption sketch
// This sketch tests communication between your terminal program and the Arduino board.
// If you send a plain text message to the Arduino board, you get back an encrypted message.
// If you send the encrypted message back to the Arduino, you get the original plain text message back.
// Hence it is called the symmetric encryption

// Testing procedure:
// Connect your USB to serial cable to the Arduino.
// Compile and upload this sketch to the Arduino board.
// Go to tools -> serial monitor.
// Type in a plain text message and send it.
// The Arduino will reply with an encrypted message.

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);   // initialize the serial port at 9600 baud
  while (!Serial) {
        ;  // wait for serial port to connect
        } // wait for serial port to connect
  establishContact();  // wait for incoming data
  } /* setup */
```

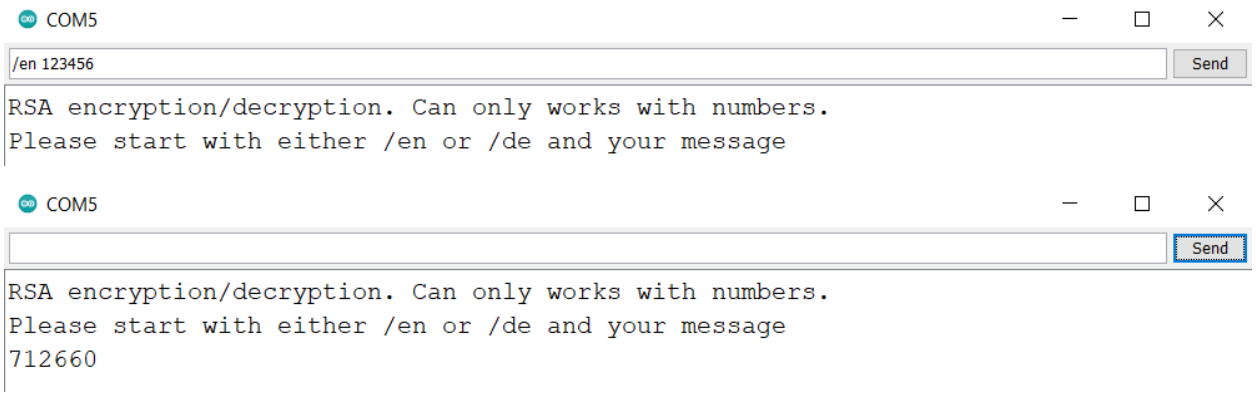
```
void loop() {
  // put your main code here, to run repeatedly:
  // This is the XOR encrypt-decrypt symmetric algorithm
  // If you run the algorithm twice, you get back the original message

  String input = "";              // serial input character
  char key[30] = "abcdefghijklmnopqrstuvwxyz";      // The key for xor function
  if (Serial.available() > 0) {        // if you have data input
        input = Serial.readString();        // read the whole input
        char charInput[input.length() + 1];
        input.toCharArray(charInput, input.length()); // adding the string input as char * type
        char * result = XORENC(charInput, key); // xor symmetric encryption
        Serial.print(result);
        Serial.println();
   }
}  /* loop */

// XOR symmetric encryption
char* XORENC(char* in, char* key){
  int insize = strlen(in);
  int keysize = strlen(key);
  for(int x=0; x < insize; x++){
    for(int i=0; i < keysize; i++){
      in[x]=(in[x]^key[i]);
    }
  }
  return in;
}

void establishContact() {
   if (Serial.available() <= 0) {
      Serial.print("XOR encryption/decryption. Please input a message");
   }
   Serial.println();
} // establishContact()
```

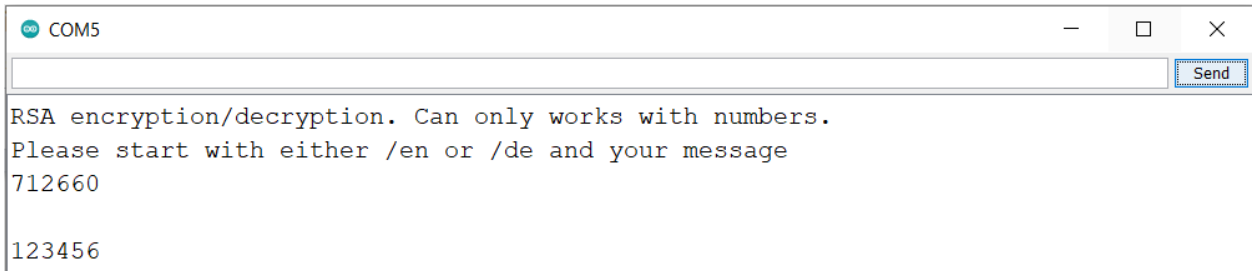
Section 5: RSA encryption and decryption (bonus)

In this assignment, I use hard-coded keys inside the program. The RSA algorithm works as follows:

- Choose two prime numbers p and q. In practice, these two prime numbers must be significantly big. For the sake of this assignment, I use small scale prime number to fit in integer type
- **int p = 997; // First prime number**
- **int q = 991; // Second prime number**
- The modulus n is calculated as a product of p and q, where n is the modulus for the public key and the private keys.
- **int n = p * q = 988027**
- The totient of n (phi(n)) is the number of co-prime numbers less than n. If n is has factorization of only 2 prime numbers, the totient has a formula of
- **int phi(n) = (p - 1)*(q - 1) = 986040**
- Choose an integer e such that 1 < e < phi(n), and e is co-prime to phi(n), then e is released as the public key exponent
- **int e = 877;**
- Compute d to satisfy the congruence relation: d * e congruent to 1 (mod phi(n)). In other words, d * e = 1 + x * phi(n), d and x are integers. Then d is kept as the private key exponent.
- **d = 3373; (In this case, 877 * 3373 = 1 + 3 * 986040)**
- Encrypted message and decrypted message will be calculated with the formulas:
- **encryptedMessage = (originalMessage ^ e) mod n;**
- **decryptedMessage = (encryptedMessage ^ d) mod n;**
- If the message and the exponent are large, then the exponentiation is infeasible to calculate. The modular exponentiation algorithm can be used to calculate the encrypt and decrypt formula, which is the function in the sketch:
  int modularExponentiation(long long m, int e, int n)
- This RSA however is now only limited to numbers. To encrypt plain text, a padding scheme is required to map all input strings to a unique number less than the modulus n. However this will not be included in the sketch and we will perform the RSA encryption/decryption only on numbers as demonstrated below.

RSA encryption and decryption for the number 123456789





Another example: The printable ASCII is ranged from 32 to 127 in byte data. For example, to encrypt the letter A, we can convert it to byte, which is 65. RSA encryption/decryption for 65 is:

Usage of the keys: The public key is the tuple (n, e) and private key is the tuple (n, d). For example, if Alice wants to send a message to Bob, she first sends Bob the public key (n, e). Alice keeps the private key “d” to herself only. Bob then crafts a message, decodes it with (n, e), and sends it back to Alice. Eve sniffs on the encrypted message and the public key (n, e) but they cannot decrypt the message because they don't have Alice’s private key “d”. Due to the exceptionally difficult factorization of n into p and q prime numbers, it is guaranteed Eve can never decrypt the message, given only n, without the knowledge of “d”. This is how RSA works.

Annex of the RSA encryption sketch

```
int p = 997; // First prime number
int q = 991; // Second prime number
int n = p * q; // Product of the two large number = 988027
// n is the modulus for the public key and the private keys.
int phi = (p - 1)*(q - 1); // The totient: phi(n) = (p-1)(q-1) = 986040
// e is released as the public key exponent
// Choose an integer e such that 1 < e < phi(n), and e is co-prime to phi(n)
int e = 877;

// Compute d to satisfy the congruence relation: d * e congruent to 1 (mod phi(n))
// d * e = 1 + x * phi(n). d and x are integers
// In this case, 877 * 3373 = 1 + 3 * 986040
// d is kept as the private key exponent.
int d = 3373;

// Public key is tuple (n, e) and private key is tuple (n, d)
void setup() {

  // put your setup code here, to run once:

  Serial.begin(9600);   // initialize the serial port at 9600 baud

  while (!Serial) {

        ;  // wait for serial port to connect

        } // wait for serial port to connect

  establishContact();  // wait for incoming data

  } /* setup */


void loop() {

  // put your main code here, to run repeatedly:

  // This is the RSA encrypt-decrypt algorithm

  String input = "";             // serial input character
  if (Serial.available() > 0){        // if you have data input
        input = Serial.readString();        // read the whole input
        if (input.startsWith("/en ")){
```

```
          int messageToEncrypt = input.substring(4).toInt();
          int encryptedMessage = modularExponentiation(messageToEncrypt, e, n);
          Serial.println(encryptedMessage);
         } else if (input.startsWith("/de ")){
          int messageToDecrypt = input.substring(4).toInt();
          int decryptedMessage = modularExponentiation(messageToDecrypt, d, n);
          Serial.println(decryptedMessage);
         } else {
            Serial.print("Unknown command. Please start the message with either /en or /de ");
          }
         Serial.println();

   } // if Serial.available() > 0

}  /* loop */

/* Efificent iterative modular exponentiation function to calculate (m^e)%n in O(log e) */
int modularExponentiation(long long m, int e, int n)
{
    long res = 1;     // Initialize result

    m = m % n; // Update m if it is more than or equal to n

    if (m == 0) return 0; // In case m is divisible by n;

    while (e > 0) {
        // If e is odd, multiply m with result
        if (e & 1)
            res = (res*m) % n;

        // e must be even now
        e = e>>1; // e = e/2
        m = (m*m) % n;
    }
    return res;
}

void establishContact(){
   if (Serial.available() <= 0) {
      Serial.print("RSA encryption/decryption. Can only works with numbers.\nPlease start with
either /en or /de and your message");
   }
   Serial.println();
} // establishContact()
```