Lightweight Authentic Wireless Communications for Micro: Bit IoT Device

UFCF8P-15-M - Iot Systems Security

UWE Bristol | University of the West of England

Group Details: 03.6	
Student Name	**Student ID**
Manoj A Dammulla	22067303
Hansaka Ashen Hewawasam	22064325
Bhadraja Charitha Munasinghe	23046448

Contents

01.Introduction

Micro:bit is a small electronic equipment which mainly use for education purposes initially developed by British Broadcast Corporation (BBC) in 2015. Primarily this device can be considered as IOT equipment (*What is the micro:bit?*, n.d.) which can generally be used for sense, measure, and log various functions such as light, sound, temperature, movement and magnetism etc.

In this assessment we basically use a pair of Micro:bit devices to communicate with each other and demonstrate a wireless device authentication system using various encryption algorithms. Furthermore, we use nFR radio (uBit.radio) component and C/C++ programming languages to develop the system.

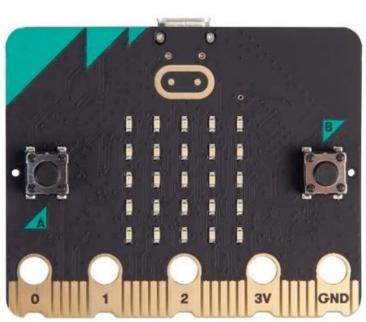
02.System Objectives

- Define three commands to communicate with and a secret code which is shared by the two Micro:bit devices.
- Using the Micro:bit 01 (Sender) will do the following functions.

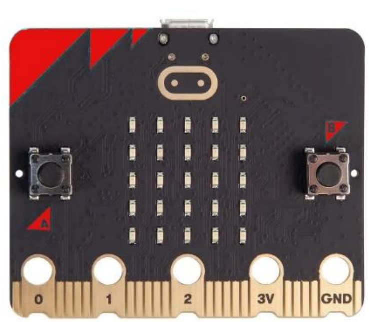
    1. Generate a random salt.
    2. Generate a data encryption key (dpk) using the shared secret, the salt, SHA256 and MD5 hash-function algorithms.
    3. Use AES to encrypt the commands.
    4. Send (cipher, salt) to the receiver Micro:bit via the radio.

- Using the Micro:bit 02 (Receiver) will do the following functions

    1. Receive the cipher text and salt)
    2. Generate the data encryption key (dpk)
    3. Decrypt the cipher.
    4. Run the command.



Sender (TX)

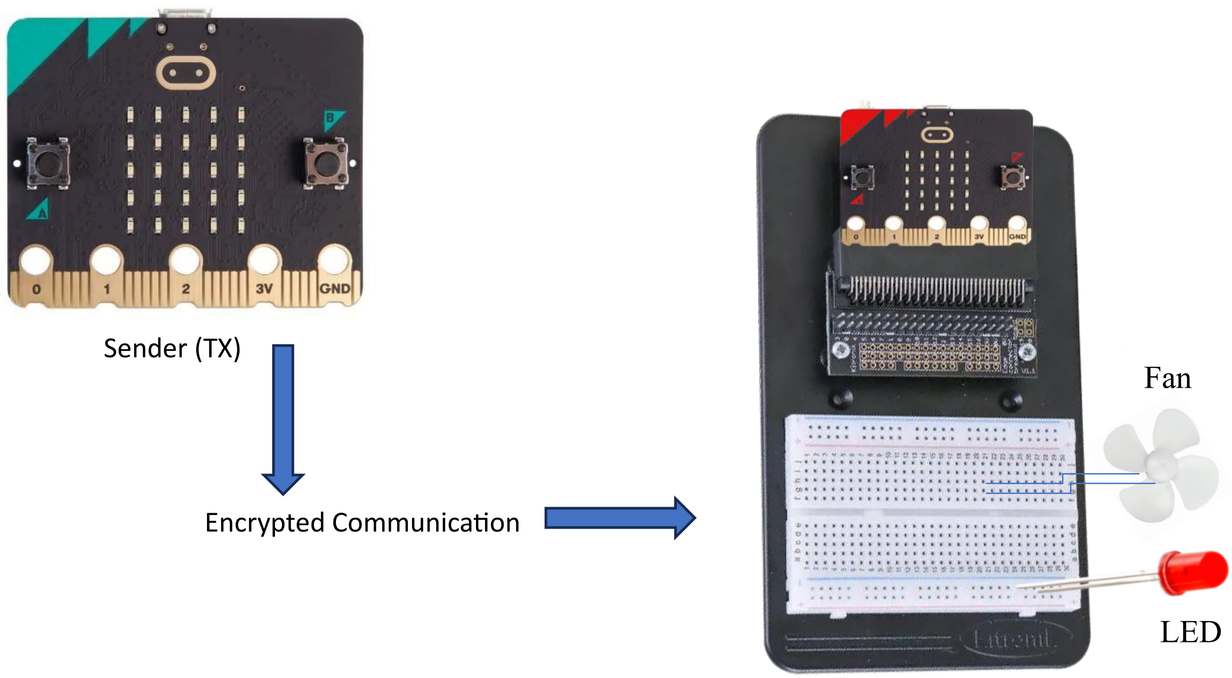


Receiver (RX)

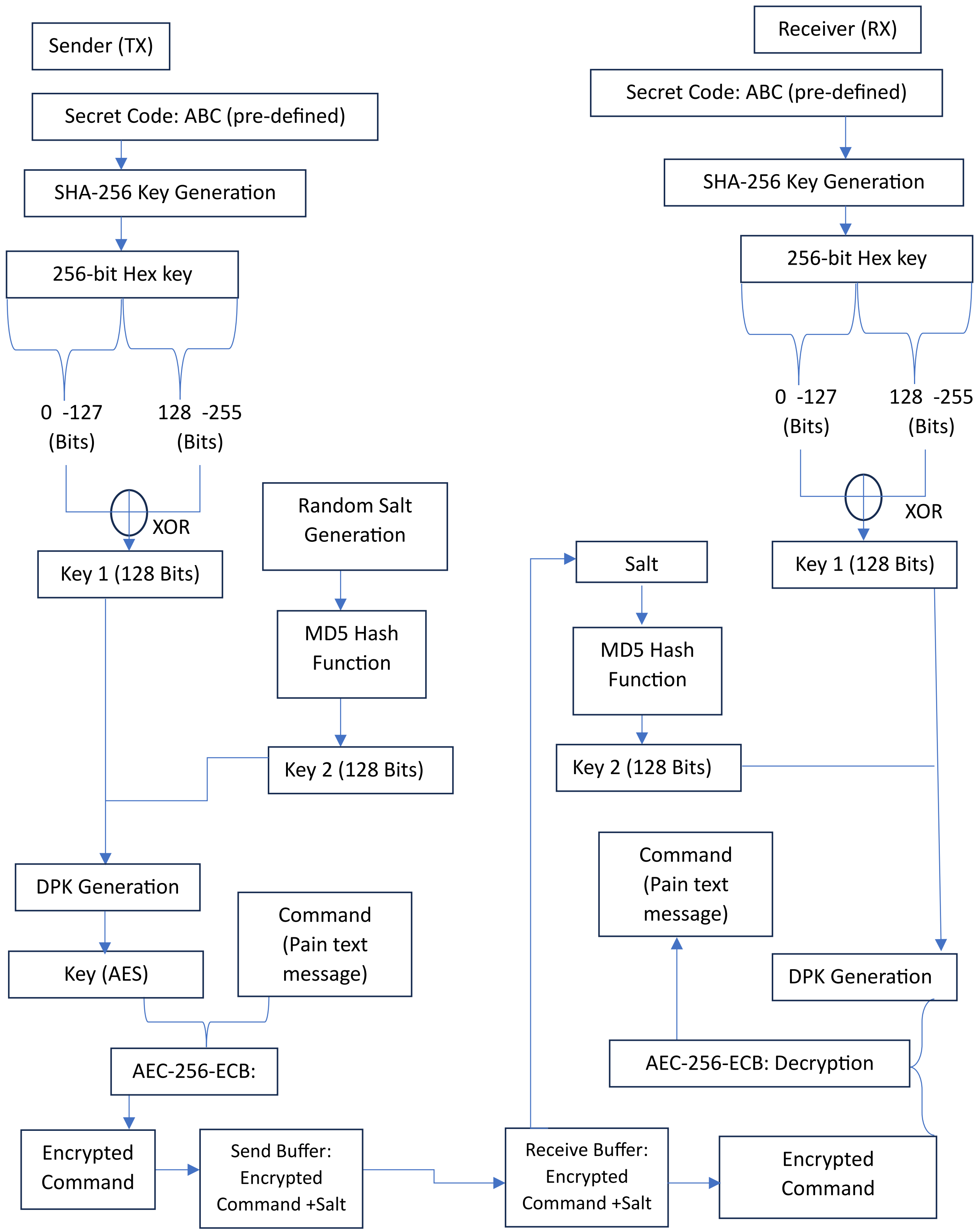
03.System Design

Sender (Micro:bit 01) -TX

Receiver (Micro:bit 02) -RX

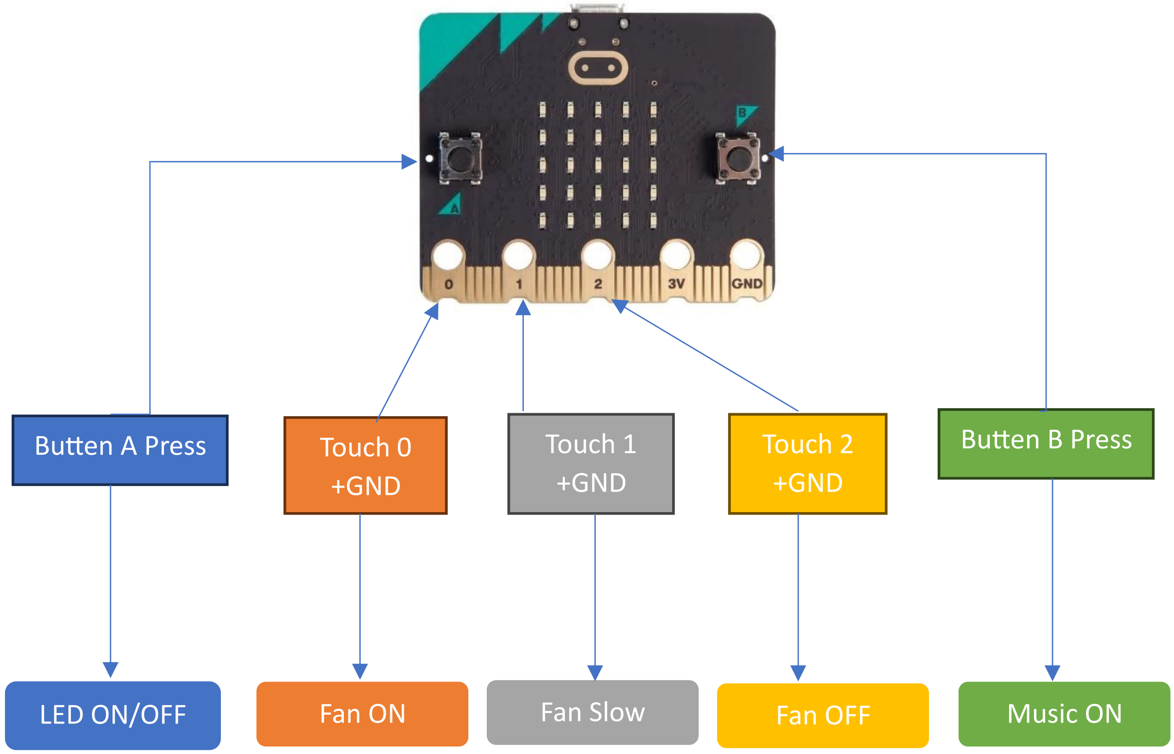


04.System Process

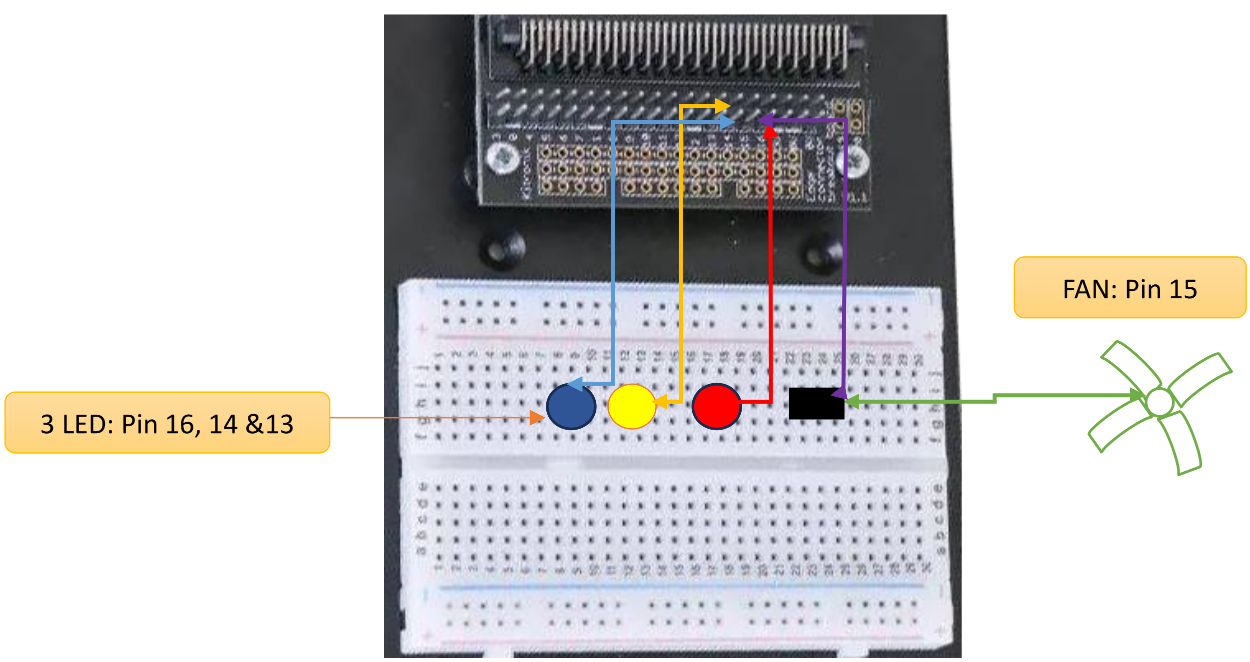


05.System Functions

Sender (TX)



Receiver (RX)



06.Code Generation

This section will cover the coding used for delivering the objectives which discussed under System Objectives.

06.1. Using the Micro:bit 01 (Sender)

06.1.1. Generate a random salt.

```
int salt = std::rand() % 255 + 1; // Generates a random number 1 byte
uBit.serial.printf("\rRandom Salt is: %d\n", salt); //print the md5 hash
uint8_t md5out[16]; // MD5 produces a 16 byte output
char str[20]; // Buffer to hold the string representation of the integer
sprintf(str, "%d", salt); // Convert integer to string

```

06.1.2. Md5 hash function

```
void md5hash(const char* intext, uint8_t* outhash) {  // function for md5 operation
    size_t length = strlen(intext);

    // Call the md5 function
    md5((uint8_t*)intext, length);

    // Copy the raw binary data of the hash into outhash
    memcpy(outhash, &h0, 4);
    memcpy(outhash + 4, &h1, 4);
    memcpy(outhash + 8, &h2, 4);
    memcpy(outhash + 12, &h3, 4);
}
```

06.1.3. XOR function

```
void xorsha256(uint8_t* shaKey, uint8_t* xorKey) {  // function create for XOR operation
    for (int i = 0; i < 16; ++i) {
        xorKey[i] = shaKey[i] ^ shaKey[i + 16];
    }
}

```

06.1.4. Import sha256 function as external.

```
/*******import prd-defined variable from sha256.cpp*********/
#ifndef SHA256_BYTES
#define SHA256_BYTES 32
#endif
unsigned char Sha_Key[SHA256_BYTES]; //define variable for store sha256 hash output
// Import sha256 function as a external function
extern void sha256(const void *data, size_t len, unsigned char *sha2digest);

/************************Import md5.c UWE file to use *****************/
```

06.1.5. Generating the DPK

Generate a data encryption key (dpk) using the shared secret, the salt, SHA256 and MD5 hash-function algorithms.

```
void encryptdata(const std::string& cleartext)
{
    /************Perform the hashing*******************************************/
    sha256(secret, strlen(secret), Sha_Key);

    std::stringstream sha_prt;    //create string to apend the sha output

     for (int i = 0; i < SHA256_BYTES; ++i) {
        // Print each byte as a two-digit hexadecimal number
        sha_prt << std::hex << std::setfill('0') << std::setw(2) << static_cast<int>(Sha_Key[i]);
    }

    std::string shaKeyStr = sha_prt.str();  // format the output for print

    sha_prt.str(""); // Clear stringstream

    // Print the SHA-256 hash
    uBit.serial.printf("\rSHA-256 hash is: %s\n", shaKeyStr.c_str());

     /*******************************XOR*************************************/
    uint8_t xorkey[16]; // XOR key will be 16 bytes
    xorsha256(Sha_Key, xorkey);

    // Print the XOR key
    uBit.serial.printf("\r\nXOR Key is: ");
    for (size_t i = 0; i < sizeof(xorkey); i++) {
        uBit.serial.printf("%x ", xorkey[i]);
        uBit.sleep(10);
    }
    uBit.serial.printf("\r\n");
    uBit.serial.printf("Size of XOR Key is: %d ", sizeof(xorkey));

    /********************************End XOR*************************************/
```

```

    int salt = std::rand() % 255 + 1; // Generates a random number 1 byte
    uBit.serial.printf("\rRandom Salt is: %d\n", salt); //print the md5 hash
    uint8_t md5out[16]; // MD5 produces a 16 byte output
    char str[20]; // Buffer to hold the string representation of the integer
    sprintf(str, "%d", salt); // Convert integer to string

    md5hash(str, md5out); // call MD5 hash function
     // Print the md5 hash key
    uBit.serial.printf("\r\nMD5 Hash Key is: ");
    for (size_t i = 0; i < sizeof(md5out); i++) {
        uBit.serial.printf("%x ", md5out[i]);
        uBit.sleep(10);
    }
    uBit.serial.printf("\r\n");

     /**********Concatanate xorKey with hashOutput*/

    uint8_t dpk[32]; // = xorKey + hashOutput;
    std::memcpy(dpk, xorkey, 16); // Copy first 32 bytes from xorKey to dpk
    std::memcpy(dpk + 16, md5out, 16); // Copy next 32 bytes from hashOutput to dpk
    uint8_t dpk_prt[32];
    strncpy((char*)dpk_prt,(char *)dpk,32);

    //print the dpk ********************
    uBit.serial.printf("\r\nDpk Key is: ");
    for (size_t i = 0; i < sizeof(dpk); i++) {
        uBit.serial.printf("%x ", dpk_prt[i]);
         uBit.sleep(10);
    }
    uBit.serial.printf("\r\n");
    uBit.serial.printf("\r\nSize of DPK is: %d\r\n",sizeof(dpk));
```

06.1.6. Use AES to encrypt the commands.

The following commands will show how the AES encryption has defined in the program.

```
/*************************AES256-ECB Encryption****************************/
 struct AES_ctx ctx; //define instance of AES_ctx
 AES_init_ctx(&ctx, dpk);
 int num_blocks = cleartext.length() / 16;
 if (cleartext.length() % 16 != 0)
 {
     num_blocks++;
 }
 for (int i = 0; i < num_blocks; i++)
 {
     std::fill(block, block + 16, 0);
     for (int j = 0; j < 16; j++)
     {
         int index = i * 16 + j;
         if (index < cleartext.length())
         {
             block[j] = cleartext[index];
         }
     }
 /*********** Encrypt the command using aes256 ECB**********/
 AES_ECB_encrypt(&ctx, block);
 uBit.serial.printf("\r\nEncrypted msg is: ");
 uBit.serial.printf("%s\r\n",&ctx);
 }

 PacketBuffer ciphertxt(sizeof(block)+1); //Create ciphertxt paketbuffer with size of 16

 for(uint8_t a=0;a<16;a++){   // 0 - 15 is encrypted message
     ciphertxt[a]=block[a];
 }
 ciphertxt[16]= salt; //****16th posission contain the salt
 uBit.radio.datagram.send(ciphertxt); //sending message through radio
 uBit.serial.printf("\r\n");
```

06.1.7. Send (cipher, salt) to the receiver Micro:bit via the radio.

Below section shows how the program send cypher text and salt to the receiver (RX) using nFR radio component.


```
 PacketBuffer ciphertxt(sizeof(block)+1); //Create ciphertxt paketbuffer with size of 16

 for(uint8_t a=0;a<16;a++){   // 0 - 15 is encrypted message
     ciphertxt[a]=block[a];
 }
 ciphertxt[16]= salt; //****16th posission contain the salt
 uBit.radio.datagram.send(ciphertxt); //sending message through radio
 uBit.serial.printf("\r\n");
```

06.1.8. Main program

```
    while(1) {
        if (uBit.buttonA.isPressed()) {
            uBit.sleep(100); // Debounce delay

            if (isLedOn) {
                encryptdata("LedOFF");
                uBit.display.scroll("LED-OFF");
                isLedOn = false;
            } else {
                encryptdata("LedON");
                uBit.display.scroll("LED-ON");
                isLedOn = true;
            }
            while(uBit.buttonA.isPressed()) {
                uBit.sleep(100); // Wait until button A is released
            }
        }
        if(uBit.io.P0.isTouched()){ //Fan ON
            uBit.display.scroll("FAN-ON");
            encryptdata("FanON");

        }if(uBit.io.P1.isTouched()){ //Fan slow
            uBit.display.scroll("FAN-SLOW");
            encryptdata("FanSLOW");

        }if(uBit.io.P2.isTouched()){ //Fan Off
            uBit.display.scroll("FAN-OFF");
            encryptdata("FanOFF");
        }

        if(uBit.buttonB.isPressed()){
            encryptdata("MusicON");
            uBit.display.scroll("MUSIC-ON");
        }
        uBit.sleep(500); //min time in ms between two transmissions
```

06.2. Using the Micro: bit 02 (Receiver)

06.2.1. Initialize the required pins.

```
MicroBitPin P0(MICROBIT_ID_IO_P0, MICROBIT_PIN_P0, PIN_CAPABILITY_ALL); //Initialize Pin 0
MicroBitPin P13(MICROBIT_ID_IO_P13, MICROBIT_PIN_P13, PIN_CAPABILITY_ALL); //Initialize Pin 13
MicroBitPin P14(MICROBIT_ID_IO_P14, MICROBIT_PIN_P14, PIN_CAPABILITY_ALL); //Initialize Pin 14
MicroBitPin P15(MICROBIT_ID_IO_P15, MICROBIT_PIN_P15, PIN_CAPABILITY_ALL); //Initialize Pin 15
MicroBitPin P16(MICROBIT_ID_IO_P16, MICROBIT_PIN_P16, PIN_CAPABILITY_ALL); //Initialize Pin 16

```

06.2.2. Receive the cipher text and salt.

The following screen depicts how to capture the transmitted cipher text and salt in the program.

```
// call this function when recive data from radio
void onDataV2(MicroBitEvent)
{
    char buffer[32];

    // catch the recived data from radio
    PacketBuffer b = uBit.radio.datagram.recv();

    //extract encrypted msg from receiev buffer (0-15)
    for(char i=0;i<16;i++){
        block[i]=b[i];
    }



    uBit.serial.printf("\r\nReceived Encrypted message is: ");
    for(uint8_t a=0;a<16;a++){
        uBit.serial.printf(" %x ", block[a]);
        uBit.sleep(10);
    }

    uBit.serial.printf("\r\nByte count %d \r\n",b.length());

    //extract salt values from receiev buffer b (16 )
    salt =b[16];
    uBit.serial.printf("\r\nReceived salt is: %d\n", salt); //print the salt

```

06.2.3. Generate the data decryption key (DPK).

This screen capture will show how the data encryption key has been generated through the system.

```
/************Perform the hashing*******************************************/
sha256(secret, strlen(secret), Sha_Key);

std::stringstream sha_prt;    //create string to apend the sha output

 for (int i = 0; i < SHA256_BYTES; ++i) {
    // Format each byte as a two-digit hexadecimal number
    sha_prt << std::hex << std::setfill('0') << std::setw(2) << static_cast<int>(Sha_Key[i]);
}

std::string shaKeyStr = sha_prt.str();  // format the output for print
sha_prt.str(""); // Clear stringstream

// Print the SHA-256 hash
uBit.serial.printf("\rSHA-256 hash is: %s\n", shaKeyStr.c_str());
/*************************************************************************/
 /********************************XOR*************************************/
uint8_t xorkey[16]; // XOR key will be 16 bytes
xorsha256(Sha_Key, xorkey);

// Print the XOR key
uBit.serial.printf("\r\nXOR Key is: ");
for (size_t i = 0; i < sizeof(xorkey); i++) {
    uBit.serial.printf("%x ", xorkey[i]);
    uBit.sleep(10);
}
uBit.serial.printf("\r\n");
uBit.serial.printf("Size of XOR Key is: %d ", sizeof(xorkey));
```

```
uint8_t md5out[16];

char str[20]; // Buffer to hold the string representation of the integer
sprintf(str, "%d", salt); // Convert integer to string


md5hash(str, md5out); // call MD5 hash function

// Print the md5 hash key
uBit.serial.printf("\r\nMD5 Hash Key is: ");
for (size_t i = 0; i < sizeof(md5out); i++) {
    uBit.serial.printf("%x ", md5out[i]);
    uBit.sleep(10);
}
uBit.serial.printf("\r\n");

/**********Concatanate xorKey with hashOutput*/

uint8_t dpk[32]; // = xorKey + hashOutput;
std::memcpy(dpk, xorkey, 16); // Copy first 16 bytes from xorKey to dpk
std::memcpy(dpk + 16, md5out, 16); // Copy next 16 bytes from hashOutput to dpk
uint8_t dpk_prt[32];   // Define an array to print the generated DPK
strncpy((char*)dpk_prt,(char *)dpk,32);

/********print the DPK*****************/
uBit.serial.printf("\r\nDpk Key is: ");
for (size_t i = 0; i < sizeof(dpk); i++) {
    uBit.serial.printf("%x ", dpk_prt[i]);
     uBit.sleep(10);
}
uBit.serial.printf("\r\n");
uBit.serial.printf("\r\nSize of DPK is: %d\r\n",sizeof(dpk));
```

06.2.4. Decrypt the cipher.

Below output shows the how to decrypt the received chipper text using AES-256-ECB

```

//call the required AES function from the library
struct AES_ctx ctx;
AES_init_ctx(&ctx,dpk);

//decript the recived data with genarated AES ctx.
AES_ECB_decrypt(&ctx, block);
uBit.serial.printf("\r\nencrypted msg is: ");
uBit.serial.printf("%s\r\n",&ctx);
uBit.serial.printf("\r\nDecrepted msg is: ");
uBit.serial.printf("%s\r\n",block);
```

06.2.5. Run the command.

Below screen shows how to execute the command which received from the Micro:bit- TX

```
    // compare received message with defined values and if its match execute the action.

    if (strcmp((char*)block,"LedON") ==0) {
        uBit.display.scroll("LED-ON");
        P16.setDigitalValue(1); // P16 is high
    }if (strcmp((char*)block,"LedOFF") ==0) {
        uBit.display.scroll("LED-OFF");
        P16.setDigitalValue(0);  // P16 is low
    }
    /***************Rotete a FAN*************/
    if (strcmp((char*)block,"FanON") ==0) {
        uBit.display.scroll("FAN-ON");
        P15.setAnalogValue(255);
    }
    if (strcmp((char*)block,"FanSLOW") ==0) {
        uBit.display.scroll("FAN-SLOW");
        P15.setAnalogValue(150);
    }
    if (strcmp((char*)block,"FanOFF") ==0) {
        uBit.display.scroll("FAN-OFF");
        P15.setAnalogValue(0);
    }
    /************Play Music***************/
    if (strcmp((char*)block,"MusicON") ==0) {
        uBit.display.scroll("MUSIC-ON");
        audio_virtual_pin_melody();

    }

    // Output the received encrypted message to the serial port.
    uBit.serial.printf("%s\r\n",buffer);
```

```
// call the function to play melody
void audio_virtual_pin_melody()
{
    pin = &uBit.audio.virtualOutputPin;
    playScale();

}
```

07.Screen Display Information

Sender -TX: This screen will display following information which is generated by the sending Microbit device.

- ✓ SHA Hash value
- ✓ XOR Key
- ✓ Random Salt
- ✓ MD5 Hash Key
- ✓ DPK
- ✓ Encrypted message

Receiver -RX: This screen will display the following information which is generated by the receiving Microbit device

- ✓ Received Encrypted message.
- ✓ Received Salt
- ✓ SHA Hash value
- ✓ Generated XOR Key
- ✓ MD5 Hash Key
- ✓ DPK
- ✓ Decrypted message

08. Conclusion

This system will demonstrate the total process of sending a message through various encrypted methods and capturing those data and decrypt them to do the intended functions accurately. Here we have used MD5 hash function with AES-256-ECB encryption to encrypt the data and decrypt it using the same sway. These encryption processes are very vital when we are making communications between IOT or other devices since the data can be obtained by a third party using various techniques such as MIM attacks easily unless we are not considering the security of the setup. Adhering to such security measures are important in the real-world scenarios since those vulnerabilities are really exist on the systems.

09. Reference.

What is the micro:bit? (no date) [online]. Available from: https://microbit.org/get-started/what-is-the-microbit/ [Accessed 02 December 2023].

'microbit-IoT-project/source/sha2.cpp at master · SnoozyRests/microbit-IoT-project' (no date) *GitHub* [online]. Available from: https://github.com/SnoozyRests/microbit-IoT-project/blob/master/source/sha2.cpp [Accessed 04 December 2023].

'tiny-AES-c/aes.c at master · kokke/tiny-AES-c' (no date) *GitHub* [online]. Available from: https://github.com/kokke/tiny-AES-c/blob/master/aes.c [Accessed 07 December 2023].