

Lightweight Authentic Wireless Communications for Micro: Bit IoT Device

UFCF8P-15-M - IoT Systems Security



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

Contents

01.	Introduction.....	3
02.	System Objectives.....	3
03.	System Design	4
04.	System Process.....	5
05.	System Functions	6
06.	Code Generation	7
06.1.	Using the Micro:bit 01 (Sender)	7
06.1.1.	Generate a random salt.....	7
06.1.2.	Md5 hash function	7
06.1.3.	XOR function.....	7
06.1.4.	Import sha256 function as external.....	8
06.1.5.	Generating the DPK.....	8
06.1.6.	Use AES to encrypt the commands.....	10
06.1.7.	Send (cipher, salt) to the receiver Micro:bit via the radio.....	10
06.1.8.	Main program.....	11
06.2.	Using the Micro: bit 02 (Receiver)	12
06.2.1.	Initialize the required pins.	12
06.2.2.	Receive the cipher text and salt.	12
06.2.3.	Generate the data decryption key (DPK).	13
06.2.4.	Decrypt the cipher.	14
06.2.5.	Run the command.	15
07.	Screen Display Information	16
08.	Conclusion	17
09.	Reference.	18

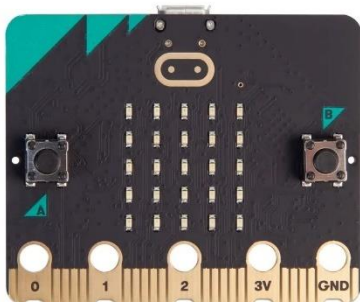
01.Introduction

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.

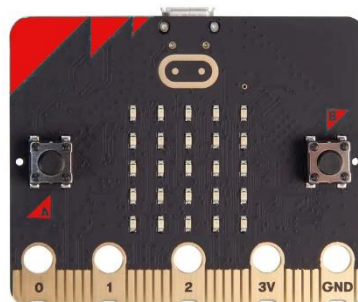
In this assessment we basically use a pair of Micro:bit devices to communicate with each other and demonstrate a wireless device authentication system using various encryption algorithms. Furthermore, we use nFR radio (uBit.radio) component and C/C++ programming languages to develop the system.

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.



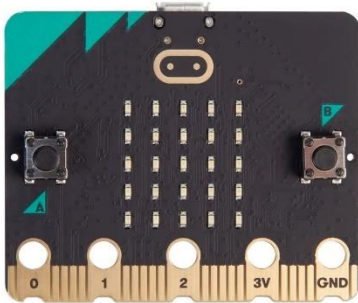
Sender (TX)



Receiver (RX)

03. System Design

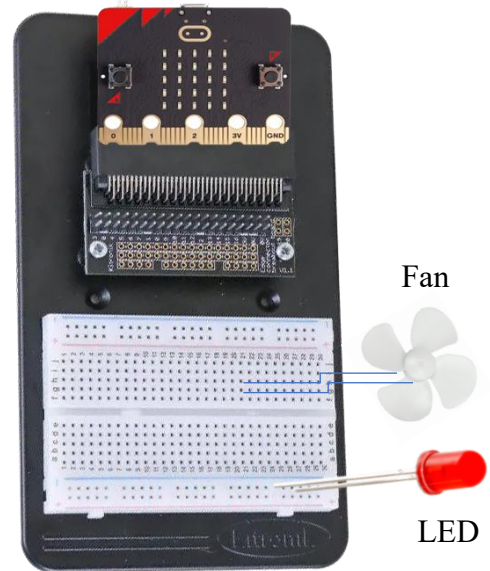
Sender (Micro:bit 01) -TX



Sender (TX)

Encrypted Communication

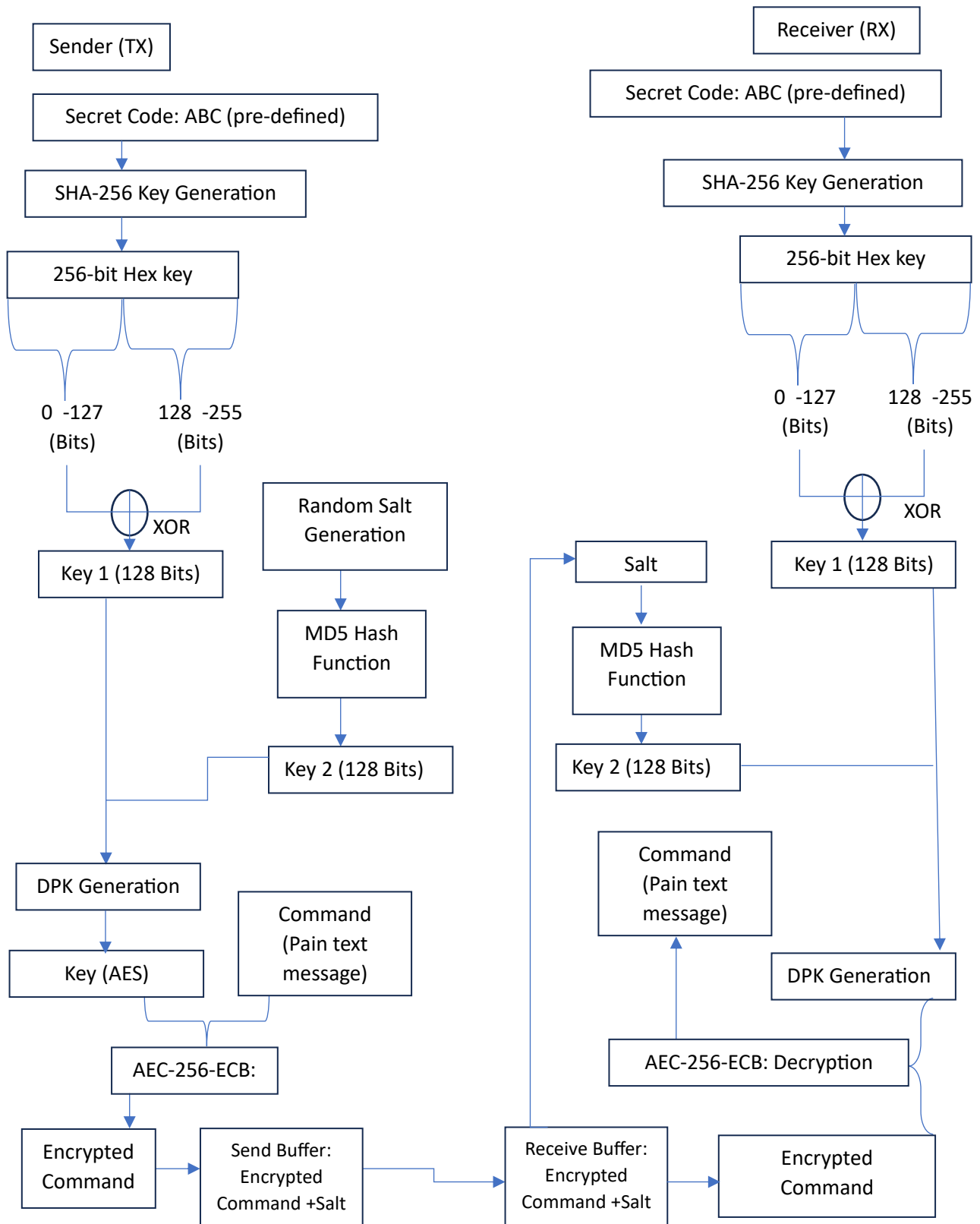
Receiver (Micro:bit 02) -RX



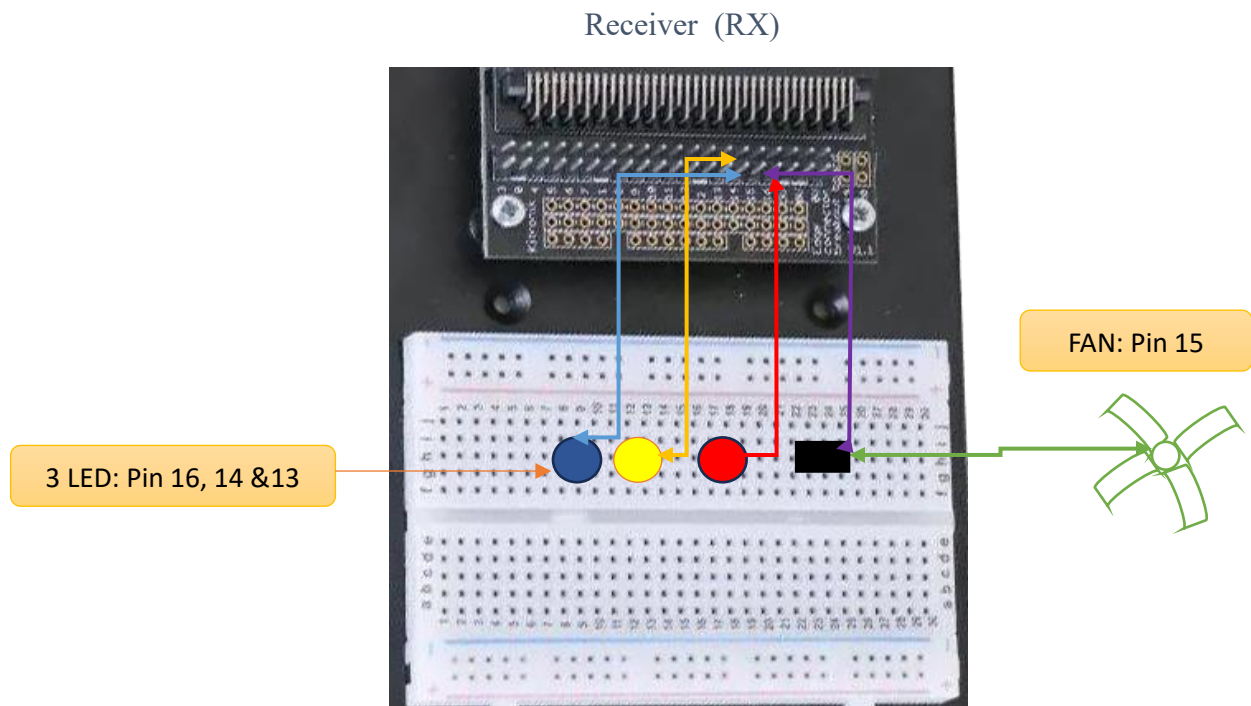
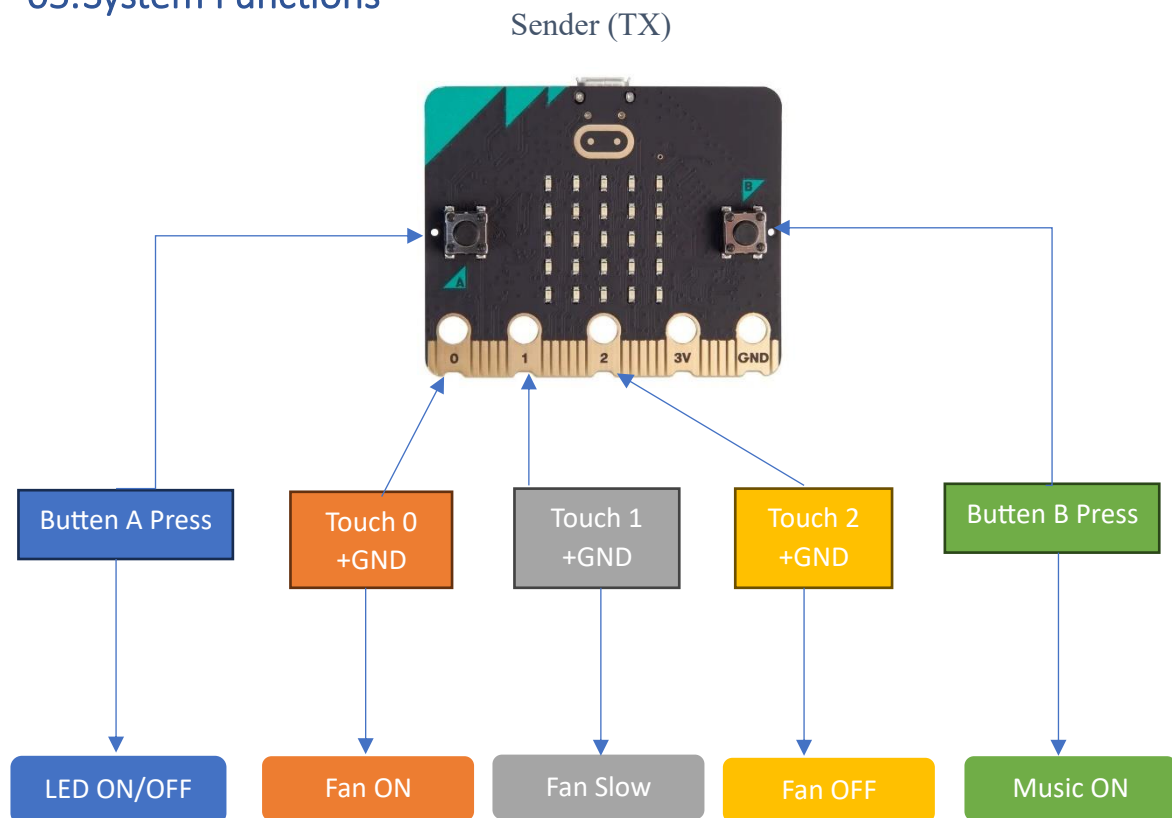
Fan

LED

04. System Process



05. System Functions



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.

```
96     int salt = std::rand() % 255 + 1; // Generates a random number 1 byte
97     uBit.serial.printf("\rRandom Salt is: %d\n", salt); //print the md5 hash
98     uint8_t md5out[16]; // MD5 produces a 16 byte output
99     char str[20]; // Buffer to hold the string representation of the integer
100     sprintf(str, "%d", salt); // Convert integer to string
101
```

06.1.2. Md5 hash function

```
37 void md5hash(const char* intext, uint8_t* outhash) { // function for md5 operation
38     size_t length = strlen(intext);
39
40     // Call the md5 function
41     md5((uint8_t*)intext, length);
42
43     // Copy the raw binary data of the hash into outhash
44     memcpy(outhash, &h0, 4);
45     memcpy(outhash + 4, &h1, 4);
46     memcpy(outhash + 8, &h2, 4);
47     memcpy(outhash + 12, &h3, 4);
48 }
49
```

06.1.3. XOR function

```
--
54 void xorsha256(uint8_t* shaKey, uint8_t* xorKey) { // function create for XOR operation
55     for (int i = 0; i < 16; ++i) {
56         xorKey[i] = shaKey[i] ^ shaKey[i + 16];
57     }
58 }
59
```

06.1.4. Import sha256 function as external.

```
--
16  /*****import prd-defined variable from sha256.cpp*****/
17  #ifndef SHA256_BYTES
18  #define SHA256_BYTES 32
19  #endif
20  unsigned char Sha_Key[SHA256_BYTES]; //define variable for store sha256 hash output
21  // Import sha256 function as a external function
22  extern void sha256(const void *data, size_t len, unsigned char *sha2digest);
23
24  /*****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.

```
62  void encryptdata(const std::string& cleartext)
63  {
64      /*****Perform the hashing*****/
65      sha256(secret, strlen(secret), Sha_Key);
66
67      std::stringstream sha_prt;    //create string to apend the sha output
68
69      for (int i = 0; i < SHA256_BYTES; ++i) {
70          // Print each byte as a two-digit hexadecimal number
71          sha_prt << std::hex << std::setfill('0') << std::setw(2) << static_cast<int>(Sha_Key[i]);
72      }
73
74      std::string shaKeyStr = sha_prt.str(); // format the output for print
75
76      sha_prt.str(""); // Clear stringstream
77
78      // Print the SHA-256 hash
79      uBit.serial.printf("\rSHA-256 hash is: %s\n", shaKeyStr.c_str());
80
81      /*****XOR*****/
82      uint8_t xorkey[16]; // XOR key will be 16 bytes
83      xorsha256(Sha_Key, xorkey);
84
85      // Print the XOR key
86      uBit.serial.printf("\r\nXOR Key is: ");
87      for (size_t i = 0; i < sizeof(xorkey); i++) {
88          uBit.serial.printf("%x ", xorkey[i]);
89          uBit.sleep(10);
90      }
91      uBit.serial.printf("\r\n");
92      uBit.serial.printf("Size of XOR Key is: %d ", sizeof(xorkey));
93
94      /*****End XOR*****/
--
```



```

95
96 int salt = std::rand() % 255 + 1; // Generates a random number 1 byte
97 uBit.serial.printf("\rRandom Salt is: %d\n", salt); //print the md5 hash
98 uint8_t md5out[16]; // MD5 produces a 16 byte output
99 char str[20]; // Buffer to hold the string representation of the integer
100 sprintf(str, "%d", salt); // Convert integer to string
101
102 md5hash(str, md5out); // call MD5 hash function
103 // Print the md5 hash key
104 uBit.serial.printf("\r\nMD5 Hash Key is: ");
105 for (size_t i = 0; i < sizeof(md5out); i++) {
106     uBit.serial.printf("%x ", md5out[i]);
107     uBit.sleep(10);
108 }
109 uBit.serial.printf("\r\n");
110
111 /******Concatenate xorKey with hashOutput*/
112
113 uint8_t dpk[32]; // = xorKey + hashOutput;
114 std::memcpy(dpk, xorkey, 16); // Copy first 16 bytes from xorKey to dpk
115 std::memcpy(dpk + 16, md5out, 16); // Copy next 16 bytes from hashOutput to dpk
116 uint8_t dpk_prt[32];
117 strncpy((char*)dpk_prt, (char *)dpk, 32);
118
119 //print the dpk *****
120 uBit.serial.printf("\r\nDpk Key is: ");
121 for (size_t i = 0; i < sizeof(dpk); i++) {
122     uBit.serial.printf("%x ", dpk_prt[i]);
123     uBit.sleep(10);
124 }
125 uBit.serial.printf("\r\n");
126 uBit.serial.printf("\r\nSize of DPK is: %d\r\n", sizeof(dpk));
127

```

06.1.6. Use AES to encrypt the commands.

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

```
128  /*****AES256-ECB Encryption*****/
129  struct AES_ctx ctx; //define instance of AES_ctx
130  AES_init_ctx(&ctx, dpk);
131  int num_blocks = cleartext.length() / 16;
132  if (cleartext.length() % 16 != 0)
133  {
134      num_blocks++;
135  }
136  for (int i = 0; i < num_blocks; i++)
137  {
138      std::fill(block, block + 16, 0);
139      for (int j = 0; j < 16; j++)
140      {
141          int index = i * 16 + j;
142          if (index < cleartext.length())
143          {
144              block[j] = cleartext[index];
145          }
146      }
147      /***** Encrypt the command using aes256 ECB*****/
148      AES_ECB_encrypt(&ctx, block);
149      uBit.serial.printf("\r\nEncrypted msg is: ");
150      uBit.serial.printf("%s\r\n", &ctx);
151  }
152
153  PacketBuffer ciphertxt(sizeof(block)+1); //Create ciphertxt packetbuffer with size of 16
154
155  for(uint8_t a=0;a<16;a++){ // 0 - 15 is encrypted message
156      ciphertxt[a]=block[a];
157  }
158  ciphertxt[16]= salt; /****16th position contain the salt
159  uBit.radio.datagram.send(ciphertxt); //sending message through radio
160  uBit.serial.printf("\r\n");
161
```

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.

```
153  PacketBuffer ciphertxt(sizeof(block)+1); //Create ciphertxt packetbuffer with size of 16
154
155  for(uint8_t a=0;a<16;a++){ // 0 - 15 is encrypted message
156      ciphertxt[a]=block[a];
157  }
158  ciphertxt[16]= salt; /****16th position contain the salt
159  uBit.radio.datagram.send(ciphertxt); //sending message through radio
160  uBit.serial.printf("\r\n");
161
```

06.1.8. Main program

```
176 while(1) {
177     if (uBit.buttonA.isPressed()) {
178         uBit.sleep(100); // Debounce delay
179
180         if (isLedOn) {
181             encryptdata("LedOFF");
182             uBit.display.scroll("LED-OFF");
183             isLedOn = false;
184         } else {
185             encryptdata("LedON");
186             uBit.display.scroll("LED-ON");
187             isLedOn = true;
188         }
189         while(uBit.buttonA.isPressed()) {
190             uBit.sleep(100); // Wait until button A is released
191         }
192     }
193     if(uBit.io.P0.isTouched()){ //Fan ON
194         uBit.display.scroll("FAN-ON");
195         encryptdata("FanON");
196     }
197     if(uBit.io.P1.isTouched()){ //Fan slow
198         uBit.display.scroll("FAN-SLOW");
199         encryptdata("FanSLOW");
200     }
201     if(uBit.io.P2.isTouched()){ //Fan Off
202         uBit.display.scroll("FAN-OFF");
203         encryptdata("FanOFF");
204     }
205
206     if(uBit.buttonB.isPressed()){
207         encryptdata("MusicON");
208         uBit.display.scroll("MUSIC-ON");
209     }
210     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.

```
~
47 MicroBitPin P0(MICROBIT_ID_IO_P0, MICROBIT_PIN_P0, PIN_CAPABILITY_ALL); //Initialize Pin 0
48 MicroBitPin P13(MICROBIT_ID_IO_P13, MICROBIT_PIN_P13, PIN_CAPABILITY_ALL); //Initialize Pin 13
49 MicroBitPin P14(MICROBIT_ID_IO_P14, MICROBIT_PIN_P14, PIN_CAPABILITY_ALL); //Initialize Pin 14
50 MicroBitPin P15(MICROBIT_ID_IO_P15, MICROBIT_PIN_P15, PIN_CAPABILITY_ALL); //Initialize Pin 15
51 MicroBitPin P16(MICROBIT_ID_IO_P16, MICROBIT_PIN_P16, PIN_CAPABILITY_ALL); //Initialize Pin 16
52
```

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.

```
84 // call this function when receive data from radio
85 void onDataV2(MicroBitEvent)
86 {
87     char buffer[32];
88
89     // catch the received data from radio
90     PacketBuffer b = uBit.radio.datagram.recv();
91
92     //extract encrypted msg from receive buffer (0-15)
93     for(char i=0;i<16;i++){
94         block[i]=b[i];
95     }
96
97
98
99     uBit.serial.printf("\r\nReceived Encrypted message is: ");
100     for(uint8_t a=0;a<16;a++){
101         uBit.serial.printf(" %x ", block[a]);
102         uBit.sleep(10);
103     }
104
105     uBit.serial.printf("\r\nByte count %d \r\n",b.length());
106
107     //extract salt values from receive buffer b (16 )
108     salt =b[16];
109     uBit.serial.printf("\r\nReceived salt is: %d\n", salt); //print the salt
110
```

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.

```
111  /*****Perform the hashing*****/
112  sha256(secret, strlen(secret), Sha_Key);
113
114  std::stringstream sha_prt;    //create string to apend the sha output
115
116  for (int i = 0; i < SHA256_BYTES; ++i) {
117      // Format each byte as a two-digit hexadecimal number
118      sha_prt << std::hex << std::setfill('0') << std::setw(2) << static_cast<int>(Sha_Key[i]);
119  }
120
121  std::string shaKeyStr = sha_prt.str(); // format the output for print
122  sha_prt.str(""); // Clear stringstream
123
124  // Print the SHA-256 hash
125  uBit.serial.printf("\rSHA-256 hash is: %s\n", shaKeyStr.c_str());
126  /*****
127  | *****/
128  uint8_t xorkey[16]; // XOR key will be 16 bytes
129  xorsha256(Sha_Key, xorkey);
130
131  // Print the XOR key
132  uBit.serial.printf("\r\nXOR Key is: ");
133  for (size_t i = 0; i < sizeof(xorkey); i++) {
134      uBit.serial.printf("%x ", xorkey[i]);
135      uBit.sleep(10);
136  }
137  uBit.serial.printf("\r\n");
138  uBit.serial.printf("Size of XOR Key is: %d ", sizeof(xorkey));
139
```

```

145     uint8_t md5out[16];
146
147     char str[20]; // Buffer to hold the string representation of the integer
148     sprintf(str, "%d", salt); // Convert integer to string
149
150
151     md5hash(str, md5out); // call MD5 hash function
152
153     // Print the md5 hash key
154     uBit.serial.printf("\r\nMD5 Hash Key is: ");
155     for (size_t i = 0; i < sizeof(md5out); i++) {
156         uBit.serial.printf("%x ", md5out[i]);
157         uBit.sleep(10);
158     }
159     uBit.serial.printf("\r\n");
160
161     /*****Concatenate xorKey with hashOutput*/
162
163     uint8_t dpk[32]; // = xorKey + hashOutput;
164     std::memcpy(dpk, xorkey, 16); // Copy first 16 bytes from xorKey to dpk
165     std::memcpy(dpk + 16, md5out, 16); // Copy next 16 bytes from hashOutput to dpk
166     uint8_t dpk_prt[32]; // Define an array to print the generated DPK
167     strncpy((char*)dpk_prt, (char *)dpk, 32);
168
169     /*****print the DPK*****/
170     uBit.serial.printf("\r\nDpk Key is: ");
171     for (size_t i = 0; i < sizeof(dpk); i++) {
172         uBit.serial.printf("%x ", dpk_prt[i]);
173         uBit.sleep(10);
174     }
175     uBit.serial.printf("\r\n");
176     uBit.serial.printf("\r\nSize of DPK is: %d\r\n", sizeof(dpk));
177

```

06.2.4. Decrypt the cipher.

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

```

175
176     //call the required AES function from the library
177     struct AES_ctx ctx;
178     AES_init_ctx(&ctx, dpk);
179
180     //decrypt the recived data with generated AES ctx.
181     AES_ECB_decrypt(&ctx, block);
182     uBit.serial.printf("\r\nencrypted msg is: ");
183     uBit.serial.printf("%s\r\n", &ctx);
184     uBit.serial.printf("\r\nDecrypted msg is: ");
185     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

```
86
87 // compare received message with defined values and if its match execute the action.
88
89 if (strcmp((char*)block,"LedON") ==0) {
90     uBit.display.scroll("LED-ON");
91     P16.setDigitalValue(1); // P16 is high
92 }if (strcmp((char*)block,"LedOFF") ==0) {
93     uBit.display.scroll("LED-OFF");
94     P16.setDigitalValue(0); // P16 is low
95 }
96 /*****Rotete a FAN*****/
97 if (strcmp((char*)block,"FanON") ==0) {
98     uBit.display.scroll("FAN-ON");
99     P15.setAnalogValue(255);
100 }
101 if (strcmp((char*)block,"FanSLOW") ==0) {
102     uBit.display.scroll("FAN-SLOW");
103     P15.setAnalogValue(150);
104 }
105 if (strcmp((char*)block,"FanOFF") ==0) {
106     uBit.display.scroll("FAN-OFF");
107     P15.setAnalogValue(0);
108 }
109 /*****Play Music*****/
110 if (strcmp((char*)block,"MusicON") ==0) {
111     uBit.display.scroll("MUSIC-ON");
112     audio_virtual_pin_melody();
113 }
114
115
116 // Output the received encrypted message to the serial port.
117 uBit.serial.printf("%s\r\n",buffer);
```

```
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26 // call the function to play melody
27 void audio_virtual_pin_melody()
28 {
29     pin = &uBit.audio.virtualOutputPin;
30     playScale();
31 }
32
33
```

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

```
[01/03/24]STUDENT_ID-uwe@192.168.160.147: ~
[01/03/24]STUDENT_ID-uwe@192.168.160.147: ~ 181x44

XOR Key ls: 2a 7f dc 17 b7 e1 69 86 d5 5c 3d 68 d9 e2 a6 5e
Random Salt ls: 178 16

MD5 Hash Key ls: 8f 85 51 79 67 79 5e ee f6 6c 22 5f 78 83 bd cb
Dpk Key ls: 2a 7f dc 17 b7 e1 69 86 d5 5c 3d 68 d9 e2 a6 5e 8f 85 51 79 67 79 5e ee f6 6c 22 5f 78 83 bd cb
Size of DPK ls: 32
Encrypted msg ls: *eeleee\=hee^eeQygy^ee\"_xeeeepe-eeeE[Z1e;eeB8.egv0leP^Re eeoeX^eQe7ee=ehoeed4lele5TerCqewebexM_3eyQp8-ee3rl3[8e4zeeee.ooooAtH4ZTKeSooVeeepcee
ee2e7eeeeleee]8
?e>9ee\"bMee$5= BeeeeMcV
SHA-256 hash ls: 671ff89ab7317e69f2371e42fb69f05a4d60248d00d017ef276b232a228b5604
XOR Key ls: 2a 7f dc 17 b7 e1 69 86 d5 5c 3d 68 d9 e2 a6 5e
Random Salt ls: 58: 16
MD5 Hash Key ls: 66 f0 41 e1 6a 60 92 8b 5 a7 e2 28 a8 9c 37 99
Dpk Key ls: 2a 7f dc 17 b7 e1 69 86 d5 5c 3d 68 d9 e2 a6 5e 60 f0 41 e1 6a 60 92 8b 5 a7 e2 28 a8 9c 37 99
Size of DPK ls: 32
Encrypted msg ls: *eeleee\=hee^feAe]\"eeee(ee7ee2eB[5exf;NeeC[e#dl'8Ee_eeBeeZeeec5ef_eC74e'S]eFeeeee]jeeSeYe!eeeeePRee0e:VeQ5';ee`1e^fh1
ee..*
SHA-256 hash ls: 671ff89ab7317e69f2371e42fb69f05a4d60248d00d017ef276b232a228b5604
XOR Key ls: 2a 7f dc 17 b7 e1 69 86 d5 5c 3d 68 d9 e2 a6 5e
Random Salt ls: 13: 16
MD5 Hash Key ls: c5 1c e4 10 c1 24 a1 e d b5 e4 b9 7f c2 af 39
Dpk Key ls: 2a 7f dc 17 b7 e1 69 86 d5 5c 3d 68 d9 e2 a6 5e c5 1c e4 10 c1 24 a1 e d b5 e4 b9 7f c2 af 39
Size of DPK ls: 32
ee`9eZeclee+Yc-UeeeeeFJ:eeS_8eeeeeBee`eIeBs,ee5fe)eeee3e0U
eIep%eSL#en
eeecReolefEke]^eeuee]b@eeK^deebSoo_eieede*IC(hte?AeeeeNeeN=ecDqee[ee8,$sne[L/eeeeDeece3
```

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


```
[01/03/24]22067303-uwe@192.168.160.144: ~
[01/03/24]22067303-uwe@192.168.160.144: ~ 144x43
(
Received Encrypted message is: f1 5c d7 37 ce 59 ef 4c de 8a a7 5 59 56 6 cf
Byte count 17
Received salt is: 58
SHA-256 hash is: 671ff89ab7317e69f2371e42fb69f05a4d60248d00d017ef276b232a228b5604
XOR Key is: 2a 7f dc 17 b7 e1 69 86 d5 5c 3d 68 d9 e2 a6 5e
Size of XOR Key is: 16
MD5 Hash Key is: 66 f0 41 e1 6a 60 92 8b 5 a7 e2 28 a8 9c 37 99
Dpk Key is: 2a 7f dc 17 b7 e1 69 86 d5 5c 3d 68 d9 e2 a6 5e 66 f0 41 e1 6a 60 92 8b 5 a7 e2 28 a8 9c 37 99
Size of DPK is: 32
encrypted msg is: *eelee\=hee^feAej'eeee(ee7ee2eB[SeXf;NeeIe#dl'&Ee_ee8eezeecgef_eC74e'S]eFeeeej]ee$eYe!eeeeePRee0e:veQs^;ee'1e*fh1
Decrypted msg is: FanSLOW
Received Encrypted message is: 49 9a 2e ad 6c ba e4 1e 31 ce 14 f5 3e 38 f2 2d
Byte count 17
Received salt is: 13
SHA-256 hash is: 671ff89ab7317e69f2371e42fb69f05a4d60248d00d017ef276b232a228b5604
XOR Key is: 2a 7f dc 17 b7 e1 69 86 d5 5c 3d 68 d9 e2 a6 5e
Size of XOR Key is: 16
MD5 Hash Key is: c5 1c e4 10 c1 24 a1 e d b5 e4 b9 7f c2 af 39
Dpk Key is: 2a 7f dc 17 b7 e1 69 86 d5 5c 3d 68 d9 e2 a6 5e c5 1c e4 10 c1 24 a1 e d b5 e4 b9 7f c2 af 39
Size of DPK is: 32
Decrypted msg is: FanOFF
```

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 way. 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].