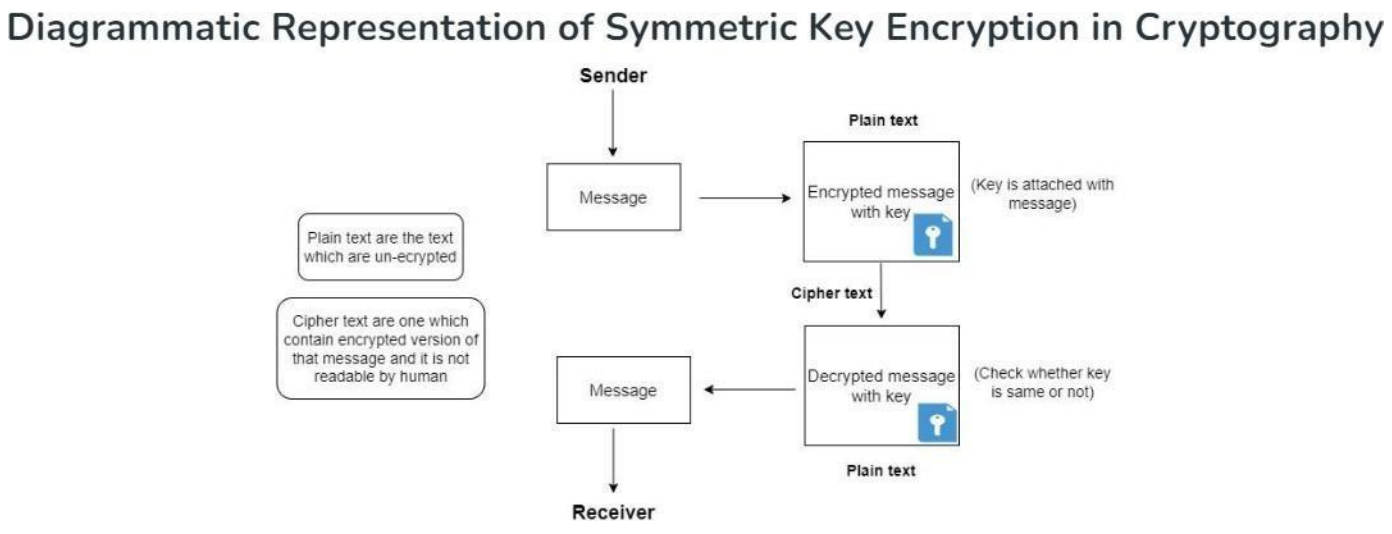
Ex. No: 6	Implementation of basic cryptography concepts in any IoT IDE.
23-08-2024	

Aim:

- To explore and apply foundational cryptographic techniques for ensuring data confidentiality and integrity in IoT applications.
- To analyze the implementation of hashing algorithms for secure data transmission and integrity verification.
- To evaluate the integration of cryptographic protocols for establishing secure communication channels between IoT devices and servers.

Introduction:

Cryptography is essential in IoT (Internet of Things) to secure data communication between devices. With the increasing connectivity of devices, the security of transmitted data becomes critical. This report details the implementation of basic cryptographic techniques, such as encryption, decryption, hashing, and secure communication protocols, using the Arduino IDE.


Diagrammatic Representation of Symmetric Key Encryption in Cryptography
Sender
Plain text
Message
Encrypted message with key
(Key is attached with message)
Plain text are the text which are un-ecrypted
Cipher text
Cipher text are one which contain encrypted version of that message and it is not readable by human
Message
Decrypted message with key
(Check whether key is same or not)
Plain text
Receiver

Overview of Cryptography in IoT:

Cryptography ensures that data is protected from unauthorized access, tampering, and eavesdropping. The key cryptographic concepts that can be implemented in an IoT environment include:

1. Symmetric Encryption: The same key is used for both encryption and decryption.

2. Asymmetric Encryption: Different keys are used for encryption (public key) and decryption (private key).

3. Hashing: A one-way function that converts data into a fixed length hash value, ensuring data integrity.

4. Secure Communication Protocols: Protocols such as TLS (Transport Layer Security) ensure secure data transmission.

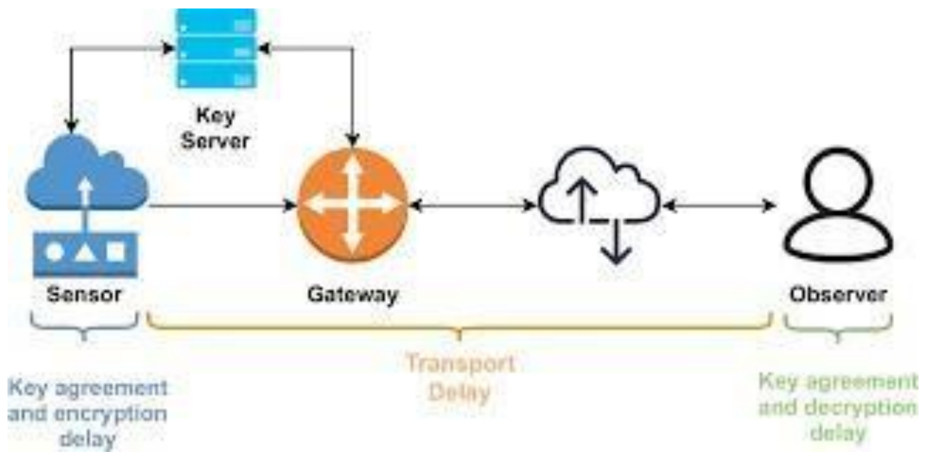
Cryptographic Libraries in Arduino IDE

Arduino provides several libraries that can be used to implement cryptographic functions:

1. Arduino Cryptography Library: Offers a range of cryptographic algorithms, including AES (Advanced Encryption Standard) and SHA (Secure Hash Algorithm).

2. Crypto: A lightweight library supporting AES, SHA1, and HMAC.

3. BearSSL: A library for secure SSL/TLS connections, commonly used for secure IoT communication.


Key Server
Sensor
Gateway
Observer
Key agreement and encryption delay
Transport Delay
Key agreement and decryption delay

Implementation of Symmetric Encryption using AES

AES (Advanced Encryption Standard) is widely used in IoT for secure data encryption. Here's how to implement AES encryption and decryption in Arduino IDE:

1. Setting Up the Environment

Install the `Crypto`library from the Arduino Library Manager. Include

the library in your sketch using `include <Crypto.h>`.

2. Writing the Code

```
include     <AES.h>

include <Crypto.h>


AES128 aes;


byte key[16] = { 0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7, 0x01, 0x2f, 0xbf, 0xd5, 0x5b,
0x6e };

byte plainText[16] = { 'E', 'x', 'a', 'm', 'p', 'l', 'e', 'T', 'e', 'x', 't', '1', '2', '3', '4' };
byte cipherText[16];
byte decryptedText[16];

void setup() {
  Serial.begin(9600);

  // Encrypt
  aes.setKey(key, sizeof(key));
  aes.encryptBlock(cipherText, plainText);

  // Decrypt
  aes.decryptBlock(decryptedText, cipherText);

  // Output
  Serial.println("Plaintext:");
  for(int i = 0; i < 16; i++) Serial.print((char)plainText[i]);

  Serial.println("\nCiphertext:");
  for(int i = 0; i < 16; i++) Serial.print(cipherText[i], HEX);

  Serial.println("\nDecrypted Text:");
  for(int i = 0; i < 16; i++) Serial.print((char)decryptedText[i]);
}

void loop() {}
```

3. Algorithm / Explanation of the Code

Key and Plaintext: A 16byte key is used for encryption, and a 16byte plaintext is the data to be encrypted. AES Encryption: The `aes.encryptBlock` function encrypts the plaintext using the key.

AES Decryption: The `aes.decryptBlock` function decrypts the ciphertext back into plaintext.

Hashing Data using SHA256

Hashing ensures data integrity by generating a fixedlength hash from input data. Here’s how to implement SHA256 hashing in Arduino IDE:

1. Writing the Code

```
include <SHA256.h>

SHA256 sha256;


byte message[] = "Hello, IoT!";

byte hash[32];


void         setup()         {
  Serial.begin(9600);


  //    Compute    SHA256    hash
  sha256.reset();
  sha256.update(message, sizeof(message)1);
  sha256.finalize(hash, sizeof(hash));


  //    Output    the    hash
  Serial.println("Hash:");
  for (int i = 0; i < 32; i++) {
    if    (hash[i]    <    16)    Serial.print("0");
    Serial.print(hash[i], HEX);
  }
}


void loop() {}
```

2. Algorithm / Explanation of the Code

Message: The input data to be hashed.

SHA256 Hashing: The `sha256.update` function processes the message, and `sha256.finalize` generates the hash.

Implementing Secure Communication with TLS

To ensure secure communication between IoT devices, TLS can be used. The `BearSSL` library in Arduino IDE can be used to establish a secure SSL/TLS connection.

1. Setting Up the Environment

Install the `BearSSL` library from the Arduino Library Manager.

Include the library in your sketch using `include <BearSSL.h>`.

2. Writing the Code for TLS Connection

```
include   <WiFiClientSecure.h>
include <BearSSL.h>

const char ssid = "yourSSID";
const char password = "yourPASSWORD";
const char host = "example.com";

WiFiClientSecure client;

void            setup()            {
  Serial.begin(9600);
  WiFi.begin(ssid, password);

  while    (WiFi.status()    !=    WL_CONNECTED)    {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  client.setInsecure(); // for testing purposes only, replace with proper certificate validation
  if (!client.connect(host, 443)) {
    Serial.println("Connection failed!");
    return;
  }
  client.println("GET   /   HTTP/1.1");
  client.println("Host: example.com");
  client.println("Connection:   close");
  client.println();

  while (client.connected() || client.available()) {

  if (client.available()) {
      String  line  =  client.readStringUntil('\n');
      Serial.println(line);
    }
  }
  client.stop();
}

void loop() {}
```

3. Algorithm / Explanation of the Code

WiFi Connection: The device connects to the specified WiFi network.

TLS Connection: The `client.connect` function establishes a secure connection to the server. Sending and Receiving Data: Data is sent to and received from the server over a secure connection.

Result:

Implementing basic cryptographic concepts in Arduino IDE enables secure IoT applications. By utilizing encryption, hashing, and secure communication protocols, the security of IoT devices and the data they transmit can be significantly enhanced.

Hence, this report we have implemented demonstrates how to use the Arduino IDE to apply these cryptographic techniques effectively.