A Secure IoMT based Patient Health Monitoring System in Compliance with HIPAA Regulations

Shaik Shakeel Ahamad[1], and Majed Alowaidi[2]

[1]College of Computer and Information Sciences, Majmaah University, Al Majmaah, Kingdom of Saudi Arabia (e-mail: s.ahamad@mu.edu.sa)

[2]College of Computer and Information Sciences, Majmaah University, Al Majmaah, Kingdom of Saudi Arabia (e-mail: m.alowaidi@mu.edu.sa)

* Correspondence: s.ahamad@mu.edu.sa;

We implemented our proposed SIPHMS protocol using Kotlin language in Android Studio (Kotlin is designed to interoperate fully with Java). Shared symmetric key is agreed between the Patient MHA and Hospital server MHA using ECDH Key exchange. Digital signatures are generated using ECDSA (digest algorithm used is SHA-256) and messages are encrypted using AES encryption algorithm. We have created an EC key pair (NIST P-256 aka secp256r1) at patient and hospital server using ECDH, and created shared AES secret key which is shared between the patient and hospital, AES with GCM (Galois/Counter Mode) mode is used for encryption and decryption of patient data. Patient needs to authenticate himself to the Mobile Healthcare Application (MHA) to use the MHA as the MHA is protected using a PIN. Figure 7 depicts Encrypted patient data and signature to the Gateway, Figure 8 depicts Gateway sending Patient's data to Hospital Server and Figure 9 depicts Hospital verifying and confirming patient data at hospital server and informing the doctor and Table 8 shows the implementation details of the proposed system. Our implementation details and code can be found here: **https://webmah.com/security/sensor-healthcare.zip**

Table 1. Implementation Environment and Parameters

Environment	Parameters
Mobile	Asus ZenFone Max M2
	Snapdragon 632 (1.8GHz octa-core)
	3GB RAM
	Android v9.0 (minimum Android v6.0)
Gateway	Linux Ubuntu 20.04
	Intel i7 9700k
	4GB RAM
	40GB SSD
	Apache Server 2.4.43
	PHP 7.2.31
	MariaDB 10.4.11
Health Care server	Linux CentOS 7.8.2003
	Intel i7 9700k
	4GB RAM
	80GB SSD
	Nginx Server 1.18.0
	PHP 7.2.31
	MariaDB 5.5.65

	Java (OpenJDK 1.8.0_252)

Establishing a secure channel between Client (Mobile) and Server by using ECDH, ECDSA, and AES through Gateway.

Figure 1. shows the process of establishing a secure channel between the client and the server. There will be a gateway which acts as middleware between client and server. Both Client and Server creates EC Key pair (Private key and Public Key). Server and Client share Public keys and then by using Elliptic Curve Diffie-Hellman (ECDH) they create shared secret AES key. ECDSA used to create the signature and verify the signature at both ends.

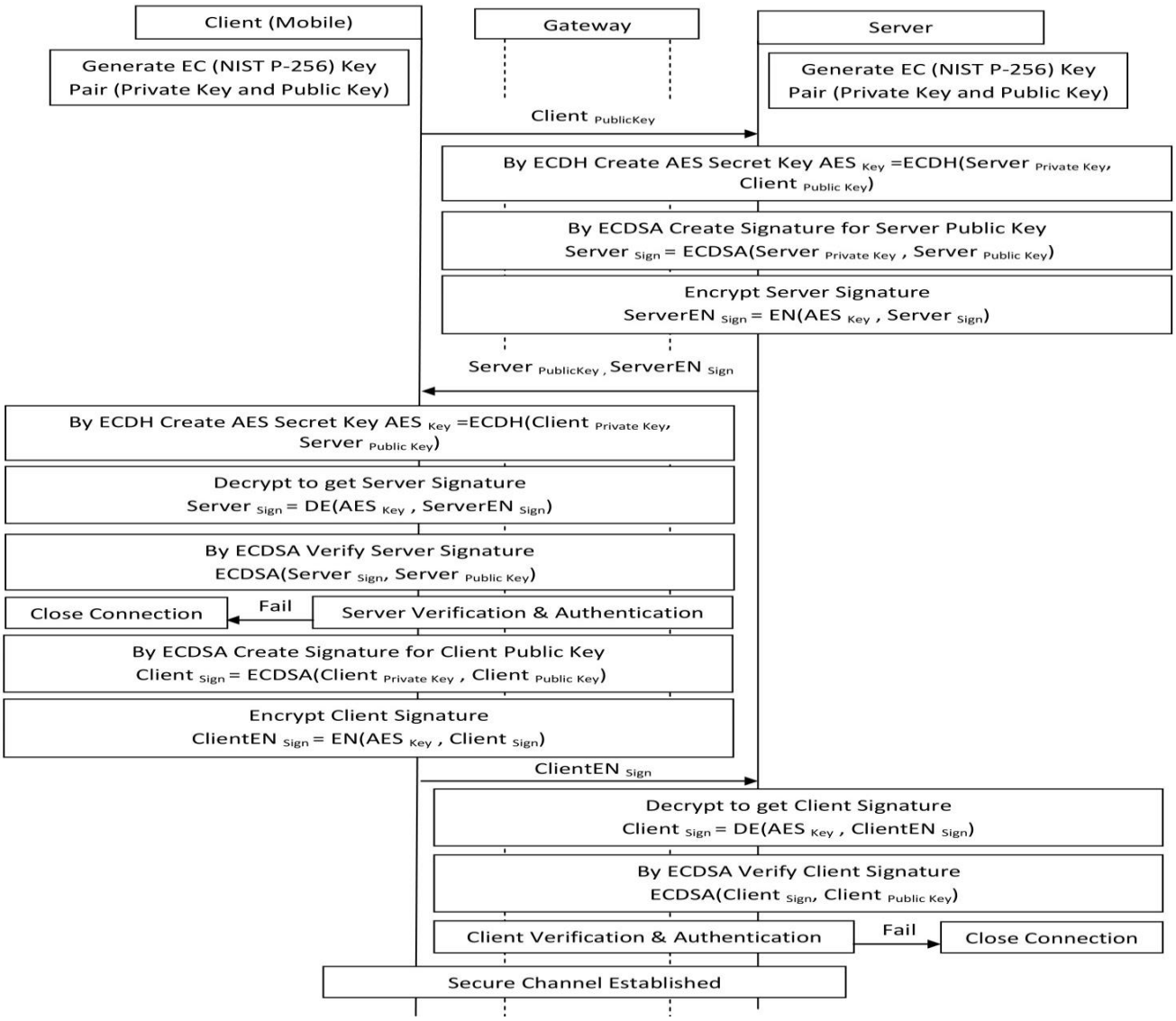

Figure 1. Establishing a secure channel between Client (Mobile) and Server and Authenticating Client and Server through Gateway

Process of Elliptic Curve Digital Signature Algorithm or ECDSA Implementation

Figure 2. shows the process of digital signature creation and verification. In ECDSA, client private key used to create a digital signature at client and client public key used to verify signature at the server. Similarly, server private key used to create a digital signature at server and server public key used to verify signature at the client.

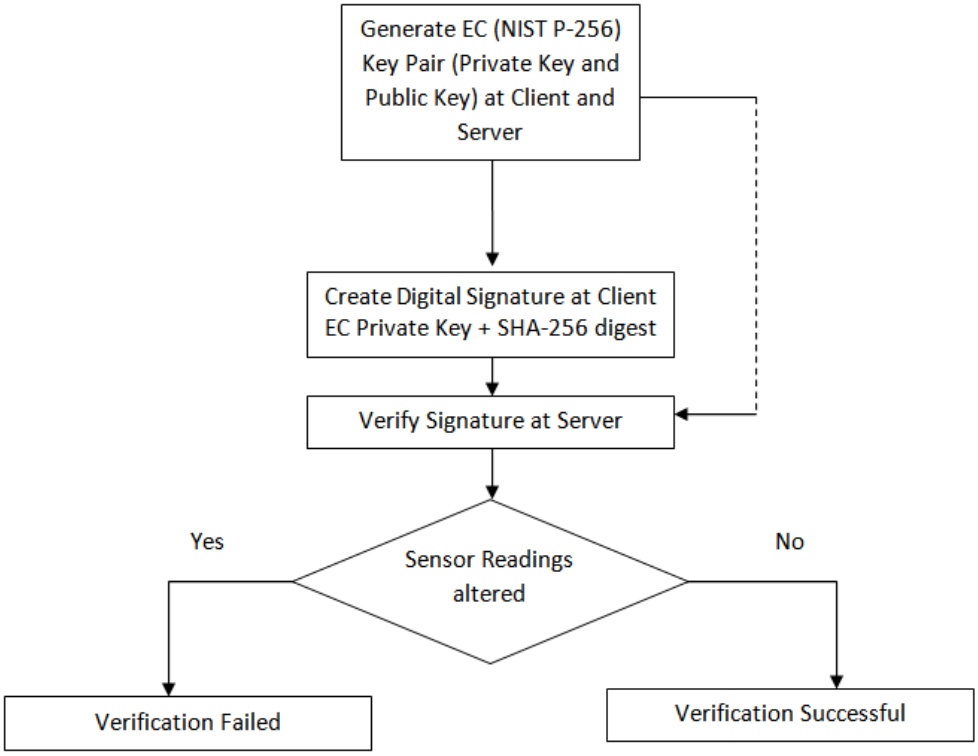

Figure 2. Process of Digital Signature Implementation in Android

AES encryption and decryption are shown in figure 3. Patient sensor readings like heart rate, blood pressure encrypted at the client and send to the server, then the server will decrypt Patient sensor readings and process received data. Every encryption or decryption involves initialization vector (IV) and this IV will be transferred between client and server or vice versa.

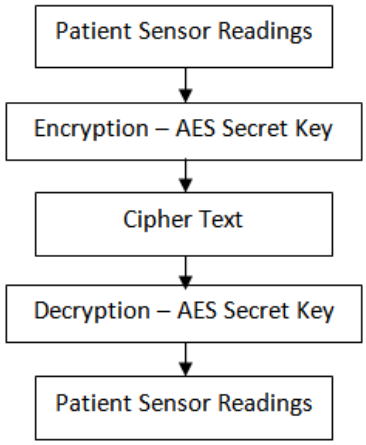

Figure 3. Encryption and Decryption of Patient Sensor readings

Figure 4 shows data transfer between client and server. Patient sensor readings are encrypted and a signature created at the client (mobile) and the server will decrypt the data and verify the signature.

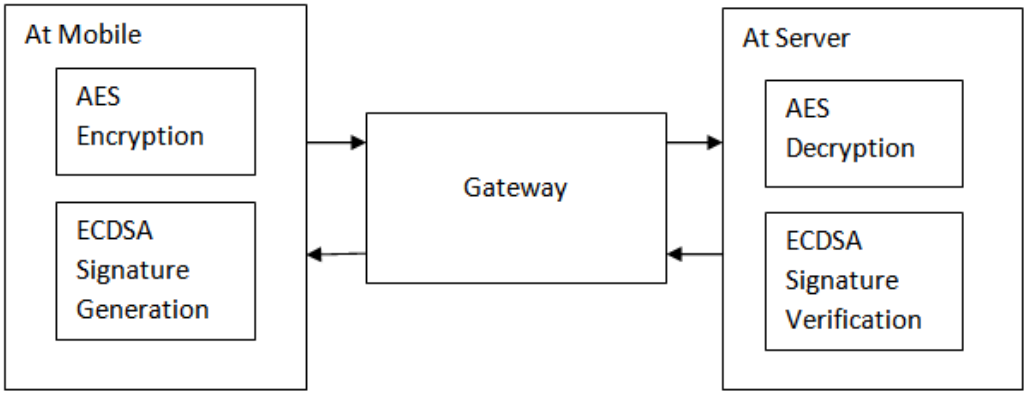

Figure 4. Mobile and Server communication through Gateway

To use the app used need to authenticate either through Pin or Fingerprint. Figure 5 shows the user authentication screenshot. Only after a successful authentication app starts communication with the gateway and server.

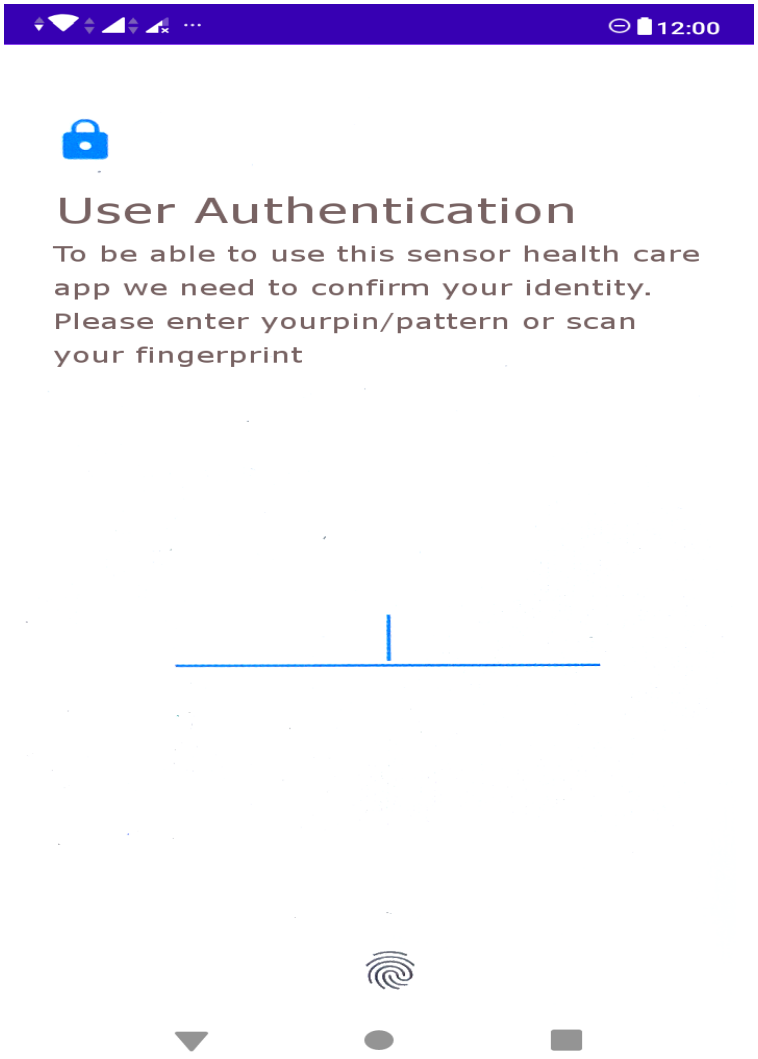

Figure 5. User authentication screen

Figure 6a shows establishing secure channel between mobile and server through Gateway by sharing keys and signatures. EC private key and public key at the client and received the server public key and server signature. Based on this data Mobile app and server establish a secure channel for communication through gateway.

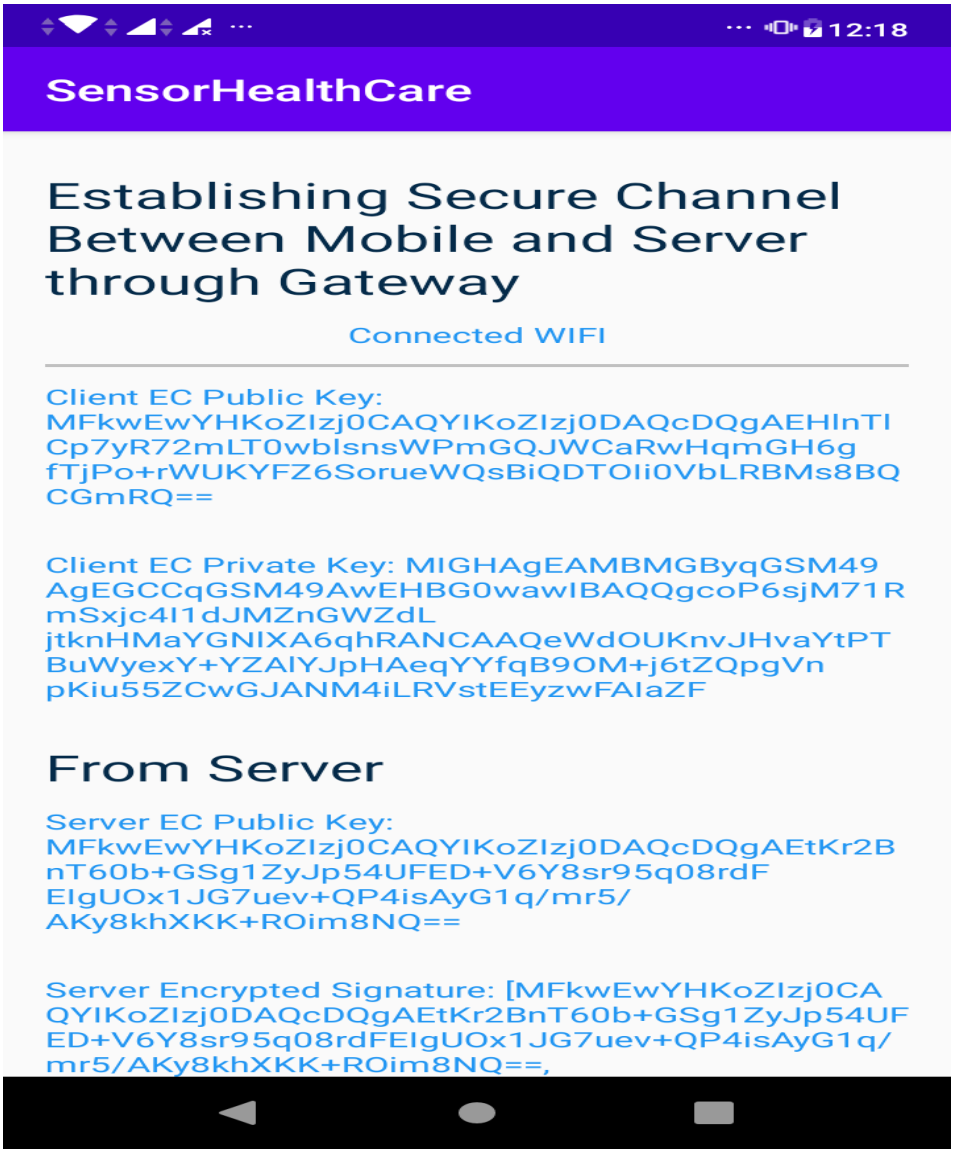

Figure 6a. Establishing Secure channel through Gateway – Client sends public key and receive server public key and signature

Figure 6b shows the client generated AES key by using ECDH. And client verifies server signature by using server public key. The client sends its signature to the server, at the server, the client's signature will be verified. After successful verifications of signatures at the client and server, the Access App option will be enabled.

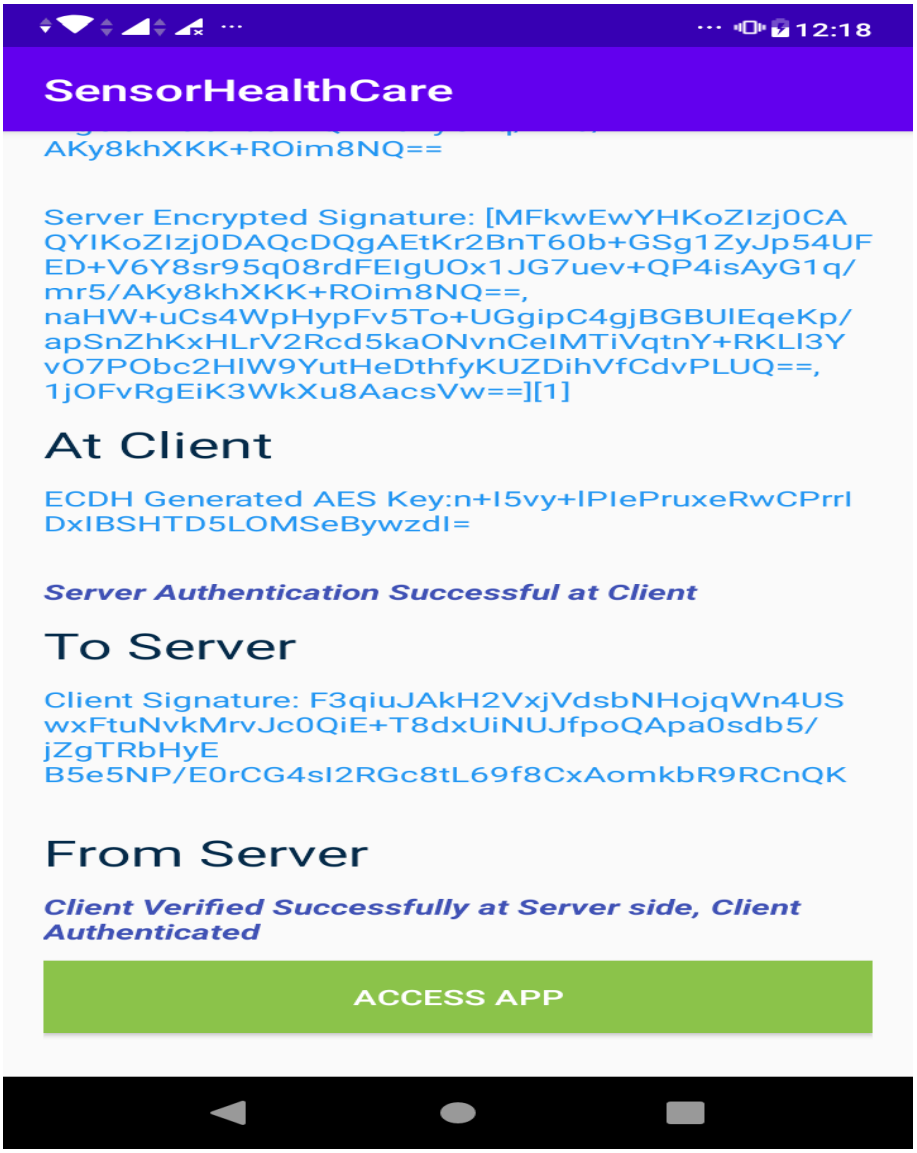

Figure 6b. Establishing Secure channel through Gateway – Client and server signatures verification and AES key generation.

Figure 7 shows patient information like Patient ID, Name, Age, Problem, Doctor, and Last visited date all this information is already registered at the server for the patient. On this screen, there is an options start monitoring sensor readings of the patient.

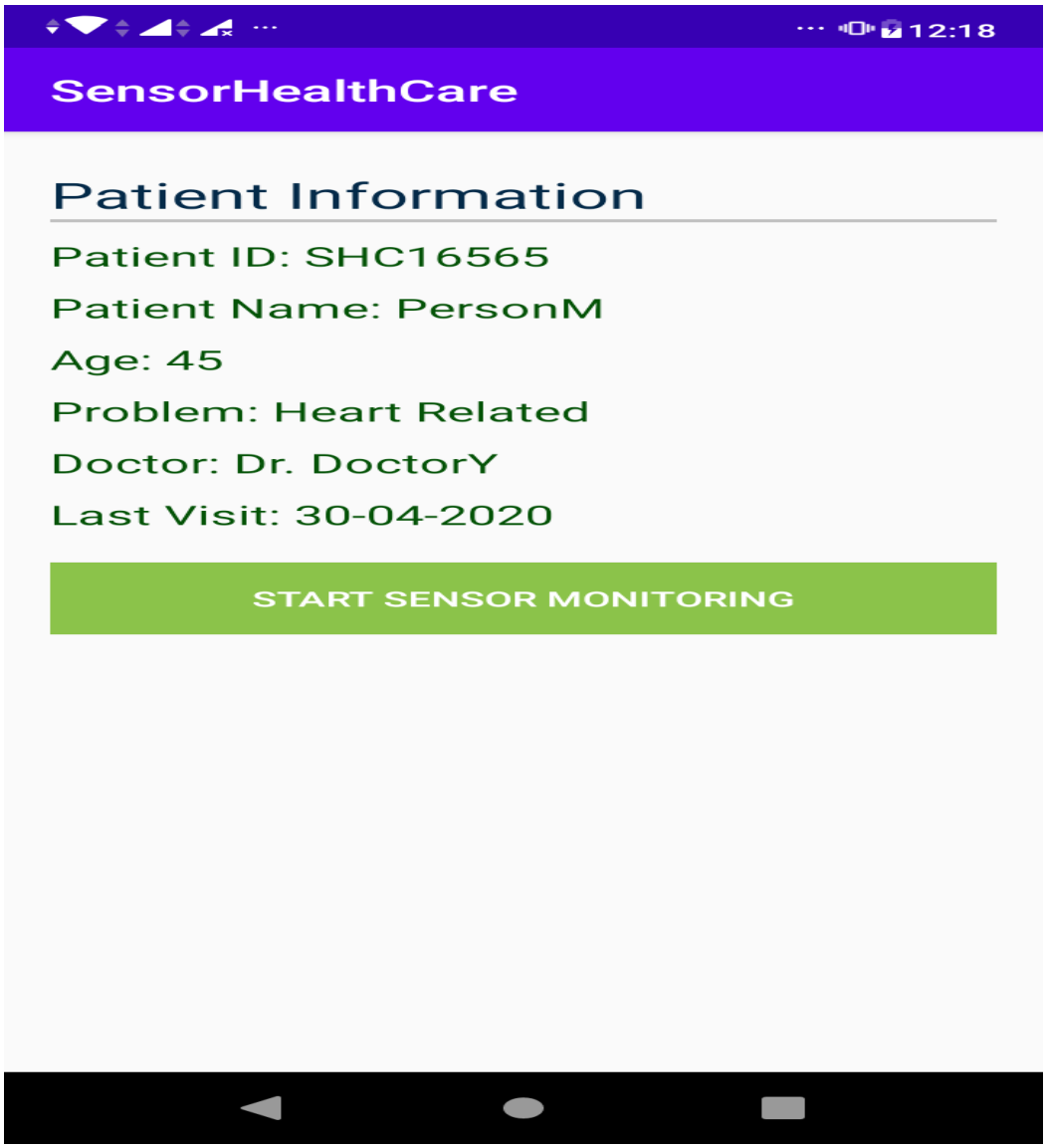

Figure 7. Patient Information.

Figure 8 shows the sensor readings of the patient. This screen shows sensor readings like heart rate and blood pressure. In this screen there is an option to send this sensor readings to health care server through gateway.

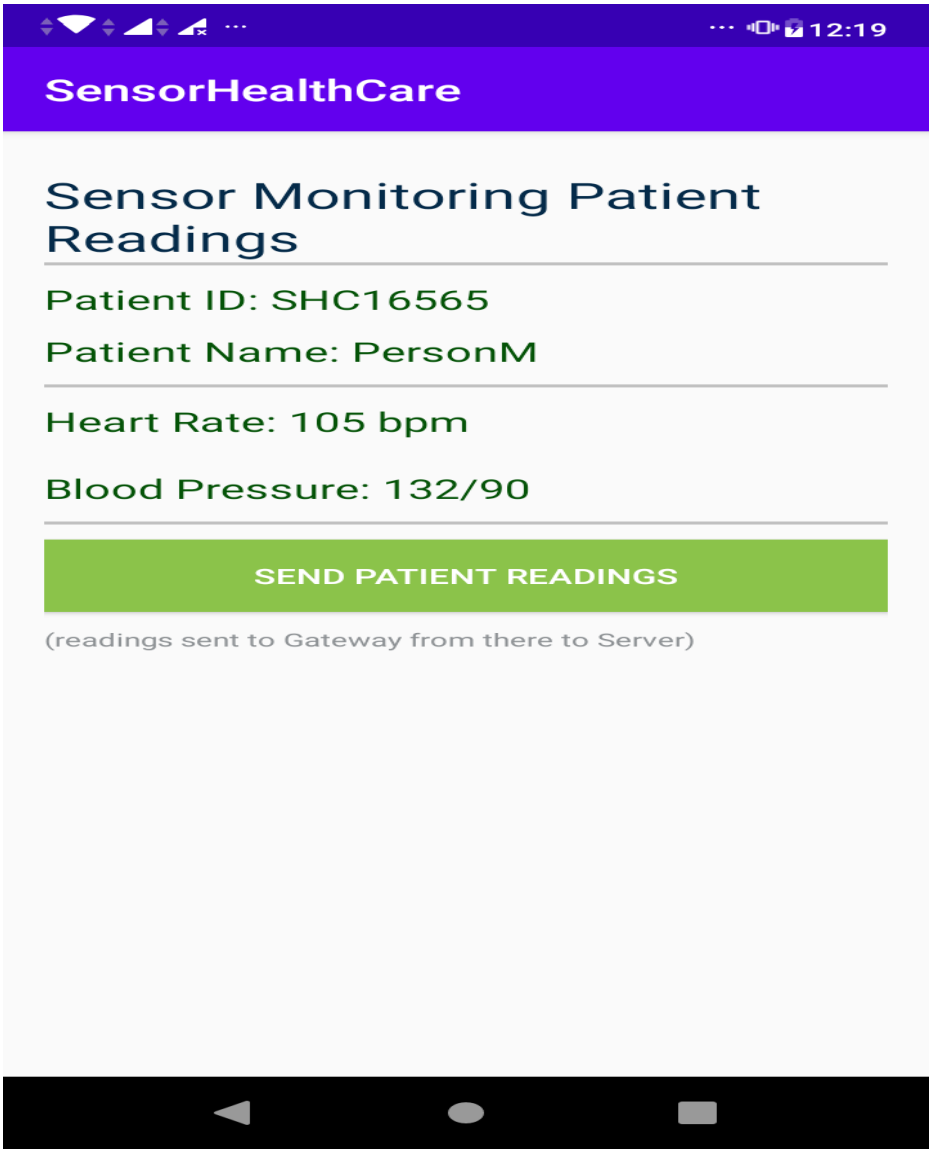

Figure 8. Send patient sensor readings to health care server through Gateway.

Figure 9a shows the screen with patient sensor readings, client encrypted patient sensor readings, and patient sensor readings signature. Encrypted patient sensor readings and the signature will be sent to the server through gateway, server verify the signature and decrypt patient sensor readings and add new readings record to the database.

Figure 9a. Send encrypted patient sensor readings and signature to the server through Gateway.

Figure 9b shows the successful confirmation of patient sensor readings records. Patient sensor readings saved at the server database and patient sensor readings sent to the doctor.

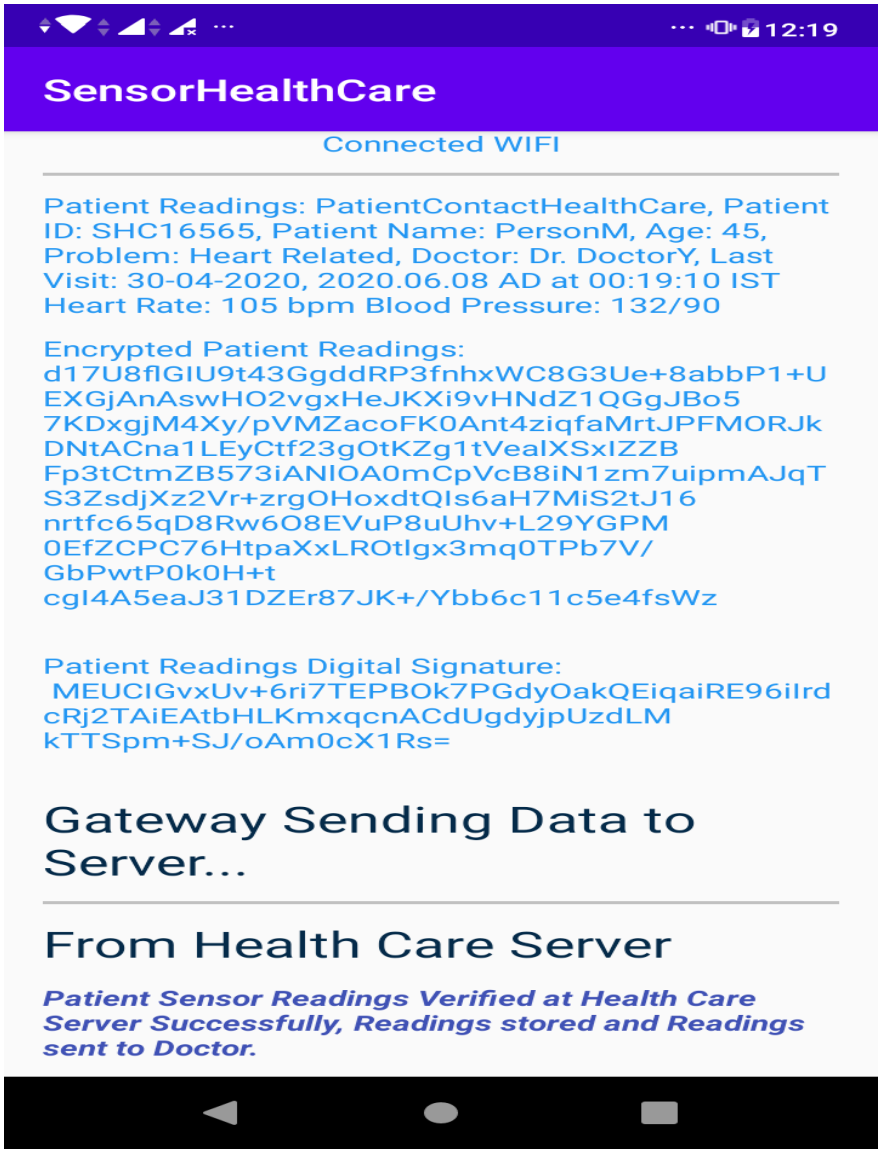

Figure 9b. Confirm and verify patient sensor readings at the server, readings stored and readings sent to the doctor.

Code to implement the proposed protocol

ECDH, ECDSA, and AES for Sensor Health care app involves lots of classes, so we provided the code of only SensorDataToGatewayAndConfirm class in this report.

Code of SensorDataToGatewayAndConfirm Class

```
package com.webmah.sensorhealthcare
import android.content.Context
import android.content.Intent
import android.net.ConnectivityManager
import android.net.Uri
import android.os.Bundle
import android.util.Base64
import android.util.Log
import android.view.View
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.lifecycle.lifecycleScope
import
kotlinx.android.synthetic.main.activity_sensordatatogatewayandconfirm.*
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext
import org.json.JSONException
import org.json.JSONObject
import java.io.*
import java.net.HttpURLConnection
import java.net.URL
import java.security.*
import java.security.spec.ECGenParameterSpec
import java.security.spec.PKCS8EncodedKeySpec
import java.security.spec.X509EncodedKeySpec
import javax.crypto.Cipher
import javax.crypto.SecretKey
```

```
import javax.crypto.spec.GCMParameterSpec
import javax.crypto.spec.SecretKeySpec
import javax.net.ssl.HttpsURLConnection
class SensorDataToGatewayAndConfirm : AppCompatActivity() {
    private lateinit var signatureResult: String
    private lateinit var enMessage: String
    private val TAG = "SMHEC"
    private val iv = "123456789abcdefh".toByteArray()
    private lateinit var clientPrivateKey: PrivateKey
    private lateinit var clientPublicKey: PublicKey
    private lateinit var clientAES: SecretKey
    private lateinit var serverResponse: String

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_sensordatatogatewayandconfirm)

        val SENDPATIENTPROBLEM = intent.getStringExtra("SENDPATIENTDATA")
        p_data.text = "Patient Readings: $SENDPATIENTPROBLEM"

        checkNetworkConnection()

        if (checkKeysExists()) {
            encryptAndSignAndSend()
        }

    }

    private fun checkNetworkConnection(): Boolean {
        val connMgr = getSystemService(Context.CONNECTIVITY_SERVICE) as
ConnectivityManager

        val networkInfo = connMgr.activeNetworkInfo
```

```
        val isConnected: Boolean = if(networkInfo != null)
networkInfo.isConnected() else false
        if (networkInfo != null && isConnected) {
            // show "Connected" & type of network "WIFI or MOBILE"
            howIsConnected.text = "Connected " + networkInfo.typeName
        } else {
            // show "Not Connected"
            howIsConnected.text = "Not Connected"
        }
        return isConnected
    }

    private fun checkKeysExists(): Boolean {
        val sharedPreference =  getSharedPreferences(SHAREDLOCATION,
Context.MODE_PRIVATE)
        if(sharedPreference.contains("clientPrivateKey") &&
sharedPreference.contains("clientPublicKey") &&
sharedPreference.contains("clientAES")){
            // decode the base64 encoded string
            val seck = sharedPreference.getString("clientAES", "no")
            if(seck == "no")
            {
                return false
            }
            System.out.println(seck)
            val secKey: ByteArray = Base64.decode(seck, Base64.DEFAULT)
            clientAES = SecretKeySpec(secKey, 0, secKey.size, "AES")

            // decode the base64 encoded string
            val pukey: ByteArray =
Base64.decode(sharedPreference.getString("clientPublicKey", "no"),
Base64.DEFAULT)
            val keySpec = X509EncodedKeySpec(pukey)
```

```
            val keyFactory = KeyFactory.getInstance("EC")
            clientPublicKey = keyFactory.generatePublic(keySpec)

            val prkey: ByteArray =
Base64.decode(sharedPreference.getString("clientPrivateKey", "no"),
Base64.DEFAULT)
            val keySpec1 = PKCS8EncodedKeySpec(prkey)
            val keyFactory1 = KeyFactory.getInstance("EC")
            clientPrivateKey = keyFactory1.generatePrivate(keySpec1)

            return true
        }
        return false
    }

    private fun encryptAndSignAndSend() {

        val pdata = p_data.text.toString()
        //encryption
        serverResponse = "no"
        val cipher = Cipher.getInstance(TRANSFORMATION)
        val parameterSpec = GCMParameterSpec(128, iv)
        cipher.init(Cipher.ENCRYPT_MODE, clientAES, parameterSpec)
        val bytes = cipher.doFinal(pdata.toByteArray())
        val clientENPdata = Base64.encodeToString(bytes, Base64.DEFAULT)
        pdata_encrypted.text = "Encrypted Patient Readings: $clientENPdata"
        val ivs = Base64.encodeToString(iv, Base64.DEFAULT)

        //sign

        //We sign the data with the private key. We use ECDAS algorithm along
SHA-256 digest algorithm
        val signature: ByteArray? =
```

```
Signature.getInstance("SHA256withECDSA").run {
            initSign(clientPrivateKey)
            update(pdata.toByteArray())
            sign()
        }

        if (signature != null) {
            //We encode and store in a variable the value of the signature
            signatureResult = Base64.encodeToString(signature,
Base64.DEFAULT)
            pdata_signature.text = "Patient Readings Digital Signature: \r\n
$signatureResult"
        }

        if (checkNetworkConnection()) {
            val jsonObject = JSONObject()
            jsonObject.accumulate("iv", ivs)
            jsonObject.accumulate("clientENPdata", clientENPdata)
            jsonObject.accumulate("clientSignature", signatureResult)
            lifecycleScope.launch {
                val result =
httpPost("https://webmah.com/sensorhealthcare/ConfirmSensorHealthCareContact.
php", jsonObject)
                server_msg.text = serverResponse
            }
        }
        else
            Toast.makeText(this, "Not Connected!", Toast.LENGTH_SHORT).show()

    }

    @Throws(IOException::class, JSONException::class)
    private suspend fun httpPost(myUrl: String, jsonObject: JSONObject):
```

```
String {

        val result = withContext(Dispatchers.IO) {
            val url = URL(myUrl)
            // 1. create HttpURLConnection
            val conn = url.openConnection() as HttpsURLConnection
            conn.requestMethod = "POST"
            conn.setRequestProperty("Content-Type", "application/json;
charset=utf-8")

            // 2. build JSON object
            //val jsonObject = buidJsonObject()

            // 3. add JSON content to POST request body
            setPostRequestContent(conn, jsonObject)

            // 4. make POST request to the given URL
            conn.connect()

            // 5. return response message
            conn.responseMessage + ""

            if (conn.responseCode == HttpsURLConnection.HTTP_OK) {
                val stream = BufferedInputStream(conn.inputStream)
                serverResponse = readStream(inputStream = stream)
            } else {
                serverResponse = "Problem in Getting Server Response"
            }

        }
        return result.toString()
    }
```

```
    @Throws(IOException::class)
    private fun setPostRequestContent(conn: HttpURLConnection, jsonObject:
JSONObject) {

        val os = conn.outputStream
        val writer = BufferedWriter(OutputStreamWriter(os, "UTF-8"))
        writer.write(jsonObject.toString())
        Log.i(TAG, jsonObject.toString())
        writer.flush()
        writer.close()
        os.close()
    }

    private fun readStream(inputStream: BufferedInputStream): String {
        val bufferedReader = BufferedReader(InputStreamReader(inputStream))
        val stringBuilder = StringBuilder()
        bufferedReader.forEachLine { stringBuilder.append(it) }
        return stringBuilder.toString()
    }

}

private const val TRANSFORMATION = "AES/GCM/NoPadding"
private const val SHAREDLOCATION = "SENSORMOBILEHEALTHCARE"
```