Framework EdgeSec – Projeto e Arquitetura

Projeto Final de Programação – Gabriel Cantergiani

Visão geral

A linguagem de programação escolhida para este projeto é Kotlin, pois o contexto da implementação se dará com base no middleware Mobile Hub, que é um aplicativo mobile para dispositivos Android. Como Kotlin é uma das principais linguagens para desenvolvimento mobile em Android, e já é utilizada na versão atual do Mobile Hub, foi decidido realizar a implementação do framework também nesta linguagem, facilitando a adaptação e os testes.

A arquitetura do projeto a ser desenvolvido se dará em 3 partes principais: o código do framework, o código dos plugins, e o código do ContextNetCore. Quanto ao framework, o resultado esperado deve ser um módulo em Kotlin que possa ser importado e utilizado dentro do código do Mobile Hub. A parte dos plugins também seguirá o mesmo modelo, sendo desenvolvida como um módulo Kotlin a fim de ser utilizada da mesma forma, porém, de maneira isolada. Já a parte do ContextNetCore será uma parte isolada, que teoricamente seria executada em um servidor remoto, e se comunicaria com o framework através de uma rede. Porém, para fins de praticidade do projeto, este código será feito também como um módulo Kotlin isolado, que será utilizado como uma dependência do framework.

A incorporação do framework e dos plugins dentro do Mobile Hub, ou outro gateway que os utilize, não deve ser relacionada. Porém, o framework deve receber referências à classe dos Plugins na sua inicialização, para que possa utilizar os protocolos e mecanismos implementados por eles.

Um detalhe importante quanto a implementação dos plugins é que eles serão divididos em 3 tipos de plugins diferentes:

- Plugins de Transporte: encapsulam o uso de protocolos de troca de dados nas camadas mais baixas de comunicação, como o BLE, Bluetooth Classic ou Zigbee.

- Plugins de Criptografia: encapsulam o uso de algoritmos de criptografia de chave simétrica para encriptar e decriptar dados, além de algumas outras operações criptográficas. Alguns exemplos são RC4 e DES.
- Plugins de Autenticação: encapsulam o uso de protocolos de algoritmos de chave assimétrica e hashing para fins de autenticação e assinatura de mensagens. Alguns exemplos são HmacMD5, HmacSHA256 e RSA.

Outro ponto importante de se mencionar são os padrões de projeto que serão utilizados no desenvolvimento. O principal deles é o padrão *Observer*, que será muito utilizado, especialmente no plugin de transporte e no código central do framework. Isto porque este padrão é muito adequado para lidar com trocas de mensagens assíncronas, que é o cenário típico em todas as etapas de autenticação e também de trocas de dados segura entre o gateway e os dispositivos IoT.

Como já foi dito na especificação do projeto, o público-alvo deste projeto são desenvolvedores da área de sistemas distribuídos e IoT, e fabricantes de dispositivos IoT que desejam tornar seus dispositivos compatíveis com o framework EdgeSec. Portanto, se faz necessário definir interfaces genéricas que devem ser implementadas por um plugin para que os protocolos e algoritmos possam ser utilizados de maneiras correta pelo framework.

Seguindo os métodos definidos nas interfaces, o desenvolvedor ou fabricante estará seguro de que o dispositivo utilizado poderá se aproveitar dos benefícios de segurança trazidos pelo EdgeSec, sem precisar entender os mecanismos por trás da implementação do framework. Da mesma forma, não será necessário que o framework saiba de detalhes de implementação específicas de um protocolo ou tecnologia em particular. Usando os métodos da interface genérica, garantimos que o framework poderá invocar as funções de transporte, criptografia ou autenticação de forma genérica. Isso garante que o framework funcionará de forma igual com qualquer plugin que implemente a interface corretamente.

Interface para plugins

Como dito na seção anterior, os plugins foram divididos em 3 partes. Nesta seção, serão detalhados os métodos que fazem parte de cada interface e devem ser implementados por um plugin. Também será definida a interface implementada pelo código do framework, e que servirá como meio de comunicação entre o Mobile Hub e os mecanismos de segurança do EdgeSec. Uma observação é que a definição das interfaces foi extraída do código do programa, e, portanto, está com comentários em inglês.

Interface do Framework

```
interface IEdgeSec {

    /*
       Set up EdgeSec framework initializing main variables and plugins

       Parameters:
           - gatewayID: string identifying MacAddress of gateway
           - transportPlugin: Object that implements ITransportPlugin interface
           - cryptoPlugin: Array of objects that implements ICryptographicPlugin
interface
           - authPlugin: Array of objects that implements IAuthenticationPlugin
interface

    */
    fun initialize(gatewayID: String, transportPlugin: ITransportPlugin,
cryptoPlugins: ArrayList<ICryptographicPlugin>, authPlugins:
ArrayList<IAuthenticationPlugin>);


    /*
        Search for devices nearby that are compatible with EdgeSec using
transport protocol

        Returns:
             - Observable that emits strings representing the MacAddress of
devices that are found
     */
    fun searchDevices(): Observable<String>;
```

```
    /*
    Tries to connect with device, perform handshake and start authentication
process. If succeeded, device will be connected securely.

    Parameters:
        - deviceID: String identifying MacAddress of device to connect

    Returns:
        - Observable that emits true if connection was succeeded and false
otherwise
     */
    fun secureConnect(deviceID: String): Single<Boolean>;

    /*
    Reads data securely from connected and authenticated device

    Parameters:
        - deviceID: String identifying MacAddress of device to connect

    Returns:
        - Observable that emits a ByteArray with the data read
     */
    fun secureRead(deviceID: String): Single<ByteArray>;

    /*
    Writes data securely to connected and authenticated device

    Parameters:
        - deviceID: String identifying MacAddress of device to connect

    Returns:
        - Observable that emits true if write was succeeded and false otherwise
     */
    fun secureWrite(deviceID: String, data: ByteArray): Single<Boolean>;
}
```

Interface do Plugin de Transporte

```
interface ITransportPlugin {

    /*
        Scan for nearby compatible devices using the BLE transport protocol

        Returns:
```

```
            - Observable that emits the MacAddress of devices that are found
     */
    fun scanForCompatibleDevices(): Observable<String>;

    /*
       Tries to connect with a device using BLE protocol

       Parameters:
           - device_id: string identifying MacAddress of device

       Returns:
           - Observable that emits true if connection was successful, false
otherwise
    */
    fun connect(deviceID: String): Single<Boolean>;

    /*
        Verifies if device is compatible with EdgeSec architecture

        Parameters:
            - device_id: string identifying MacAddress of device

        Returns:
            - Observable that emits true if it is successful, and false otherwise
     */
    fun verifyDeviceCompatibility(deviceID: String): Single<Boolean>;

    /*
       Sends the handshakeHelloMessage to device

       Parameters:
           - device_id: string identifying MacAddress of device
           - data: ByteArray with data to be sent

       Returns:
           - Observable that emits true if it is successful, false otherwise
    */
    fun sendHandshakeHello(deviceID: String, data: ByteArray): Single<Boolean>;

    /*
        Reads the handshakeResponse from device

        Parameters:
            - device_id: string identifying MacAddress of device
```

```
        Returns:
            - Observable that emits a ByteArray containing the data that was read
in case of success, and null in case of error.
     */
    fun readHandshakeResponse(deviceID: String): Single<ByteArray>;

    /*
        Send handshakeFinish message to device

        Parameters:
            - device_id: string identifying MacAddress of device
            - data: ByteArray with data to be sent

        Returns:
            - Observable that emits true in case of successful write, and false
otherwise.
     */
    fun sendHandshakeFinished(deviceID: String, data: ByteArray):
Single<Boolean>;

    /*
        Sends hello message to device

        Parameters:
            - device_id: string identifying MacAddress of device
            - data: ByteArray with data to be sent

        Returns:
            - Observable that emits true in case of successful write, and false
otherwise.
     */
    fun sendHelloMessage(deviceID: String, data: ByteArray): Single<Boolean>;

    /*
        Reads the HelloMessageResponse

        Parameters:
            - device_id: string identifying MacAddress of device

        Returns:
            - Observable that emits a ByteArray containing the message if read is
successful, and null otherwise.
     */
    fun readHelloMessageResponse(deviceID: String): Single<ByteArray>;
```

```
    /*
        Reads data from device

        Parameters:
            - device_id: string identifying MacAddress of device

        Returns:
            - Observable that emits a ByteArray containing the message if read is
successful, and null otherwise.
     */
    fun readData(deviceID: String): Single<ByteArray>;

    /*
        Writes data to device

        Parameters:
            - device_id: string identifying MacAddress of device
            - data: ByteArray with data to be sent

        Returns:
            - Observable that emits true if write is successful, and false
otherwise.
     */
    fun writeData(deviceID: String, data: ByteArray): Single<Boolean>;

}
```

Interface do Plugin de Criptografia

```
interface ICryptographicPlugin {

    /*
        Returns ID of protocol implemented by plugin

        Returns:
            - String representing protocol
     */
    fun getProtocolID(): String;

    /*
        Generate a random token using protocol implemented by plugin

        Parameters:
            - size: Integer defining size of token to be generated in bytes
```

```
        Returns:
            - ByteArray representing secure random token generated
     */
    fun generateSecureRandomToken(size: Int): ByteArray;

    /*
        Generate a secret key

        Parameters:
            - seed: ByteArray to be used as seed to generate key

        Returns:
            - Key object generated
     */
    fun generateSecretKey(seed: ByteArray): Key;


    /*
        Encrypt data using a provided key

        Parameters:
            - plainText: ByteArray representing data to be encrypted
            - key: ByteArray representing key value to be used in encryption

        Returns:
            - ByteArray representing encrypted data
     */
    fun encrypt(plainText: ByteArray, key: ByteArray): ByteArray;

    /*
        Decrypt data using a provided key

        Parameters:
            - cipher: ByteArray representing encrypted data
            - key: ByteArray representing key value to be used in decryption

        Returns:
            - ByteArray representing decrypted data
     */
    fun decrypt(cipher: ByteArray, key: ByteArray): ByteArray;
}
```

Interface do Plugin de Autenticação

```
interface IAuthenticationPlugin {

    /*
        Returns ID of protocol implemented by plugin

        Returns:
            - String representing protocol
     */
    fun getProtocolID(): String;

    /*
        Sign data using provided key with the protocol implemented by plugin

        Parameters:
            - data: ByteArray representing data to be signed
            - key: Key object used for signature

        Returns:
            - ByteArray representing generated signature
     */
    fun sign(data: ByteArray, key: Key): ByteArray;

    /*
        Verify a signature

        Parameters:
            - data: ByteArray representing data of the signature to be verified
            - key: Key object used for signature
            - signature: ByteArray representing signature to be verified

        Returns:
            - True if signature is valid and false otherwise
     */
    fun verifySignature(data: ByteArray, key: Key, signature: ByteArray):
Boolean;

    /*
        Generate a hash value using hashing function implemented by plugin

        Parameters:
            - payload: ByteArray representing payload to be hashed
```

```
        Returns:
            - ByteArray representing hash generated
     */
    fun generateHash(payload: ByteArray): ByteArray;

    /*
        Return size in bytes of the hash generated by hashing function of
protocol implemented by plugin

        Returns:
            - Integer representing size of hashes generated by plugin
     */
    fun getHashSize(): Int;
}
```

Organização e módulos

Nesta seção, serão descritos os arquivos e módulos que serão implementados durante o projeto. Assim como as interfaces, a implementação será dividida entre módulos relacionados ao framework e módulos relacionados a cada plugin. A seguir, é apresentada a lista dos módulos e classes assim como detalhes sobre cada um:

EdgeSec Framework

- **EdgeSec**: Esta será a classe principal do framework, que define o objeto importado e utilizado pelo Mobile Hub. Ela implementa a interface IEdgeSec descrita na seção anterior, e é responsável por inicializar e armazenar as principais variáveis relacionadas ao processo de autenticação, autorização e troca de dados criptografada, além de efetuar todas as operações lógicas de segurança.
- **Authorization:** Esta classe é responsável por abstrair a lógica de comunicação com o servidor de autorização, no caso, o módulo ContextNetCore, que posteriormente será substituído por uma chamada de rede à um servidor remoto.
- **AuthenticationPackage**: Classe de dados que representa o objeto de um pacote de autenticação, utilizando durante o processo de autenticação.

- **Utils**: Classe que implementa diversas funções auxiliares durante o processo de autenticação e autorização, como concatenar array de bytes e converter array de bytes em string de hexadecimais.
- **SecureConnection**: Classe de dados que representa uma conexão segura estabelecida entre o gateway e um dispositivo. É utilizada como um objeto que armazena variáveis e valores referentes à uma conexão já autenticada, e pronta para trocar dados, como a chave de sessão e ID do dispositivo. É utilizada toda vez que se faz necessário enviar ou receber dados criptografados e autenticados de um dispositivo.
- **Constants**: Classe que armazena alguns valores constantes utilizados em outras classes e operações.

ContextNetCore

- **IAuthorizationProvider:** Interface que define a comunicação entre o Framework EdgeSec e o servidor de autorização
- **ContextNetCore:** Classe estática que simula todas as operações de autorização executadas no servidor de autorização. Isto inclui a verificação se um gateway e um dispositivo podem se comunicar, e a geração das chaves utilizadas na comunicação segura, como OTP e chave de sessão. Também guarda alguns Mappings que simulam uma base de dados, com a relação de quais os protocolos suportados por cada dispositivo registrado.
- **Constants:** Objeto que armazena alguns valores constantes utilizados pelas operações de autorização.
- **AuthorizationResponse:** Classe de dados que representa a resposta enviada pelo ContextNetCore ao framework, com os valores e variáveis gerados.

HMAC MD5 Authentication

- **HmacMD5**: Classe que implementa a interface IAuthenticationPlugin, utilizando o protocolo HMAC e a função de hashing MD5 como forma de autenticação das mensagens.

RC4 Cryptography

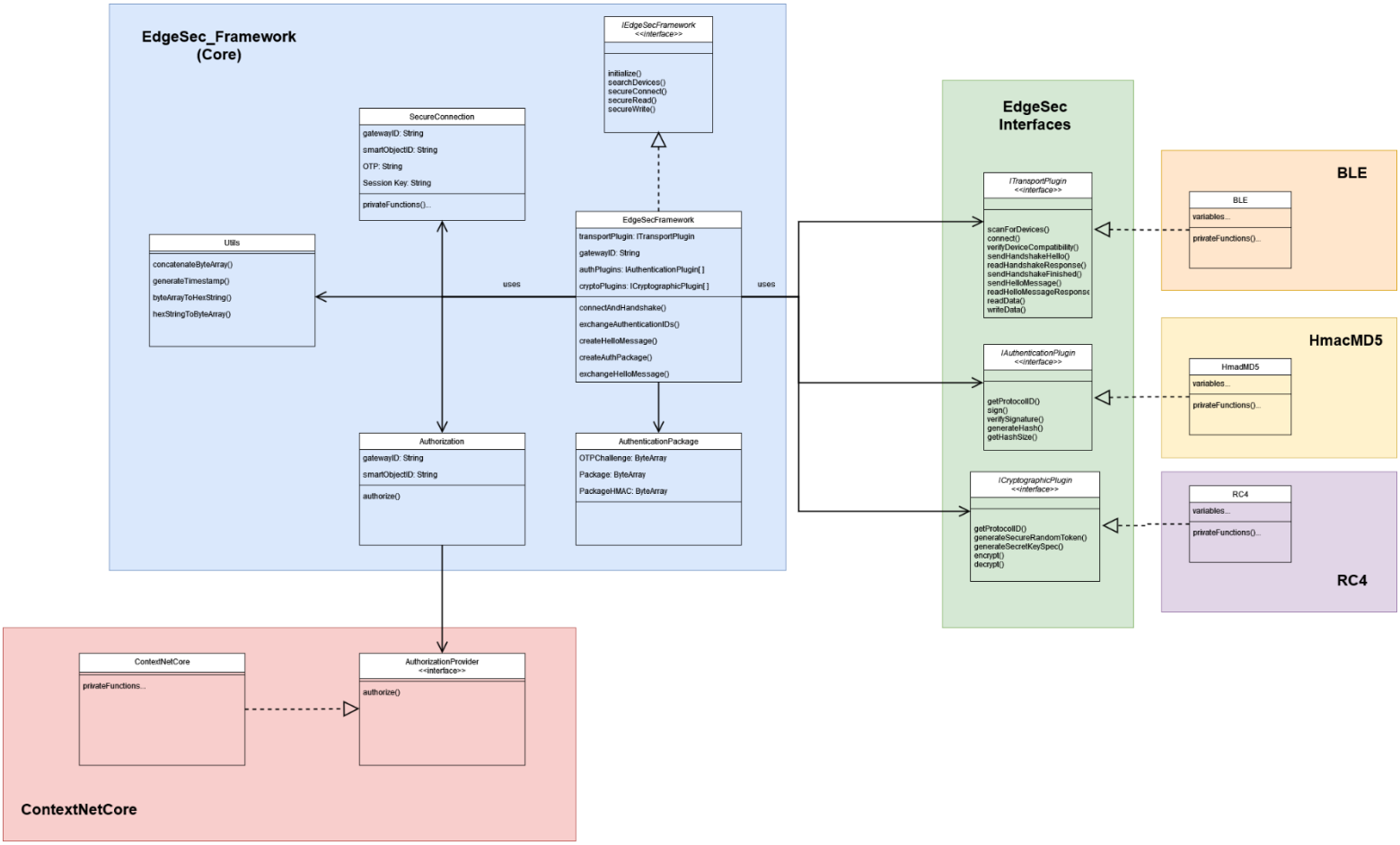
- **RC4**: Classe que implementa a interface ICryptographicPlugin, utilizando o algoritmo de chave simétrica RC4 como forma de criptografia dos dados trocados, e durante o processo de autenticação.

BLE Transport

- **BLE**: Classe que implementa a interface ITransportPlugin, utilizando o protocolo BLE para envio e recebimento de dados. Internamente, esta classe irá utilizar a biblioteca BLE *com.polidea.rxandroidble2*, que encapsula acesso às funcionalidades do protocolo BLE do gateway, como escanear por dispositivos nas redondezas, ler e escrever em características BLE, e se conectar e desconectar de dispositivos.

Diagrama de Classes (UML)

Nesta seção é apresentado o diagrama de classe que define a divisão dos módulos e classes que serão desenvolvidos no projeto. Os retângulos coloridos definem cada módulo isolado, e as interfaces são representadas com a *tag* <<interface>>.



O diagrama também será compartilhado separadamente como arquivo pdf.

Testes e controle de qualidade

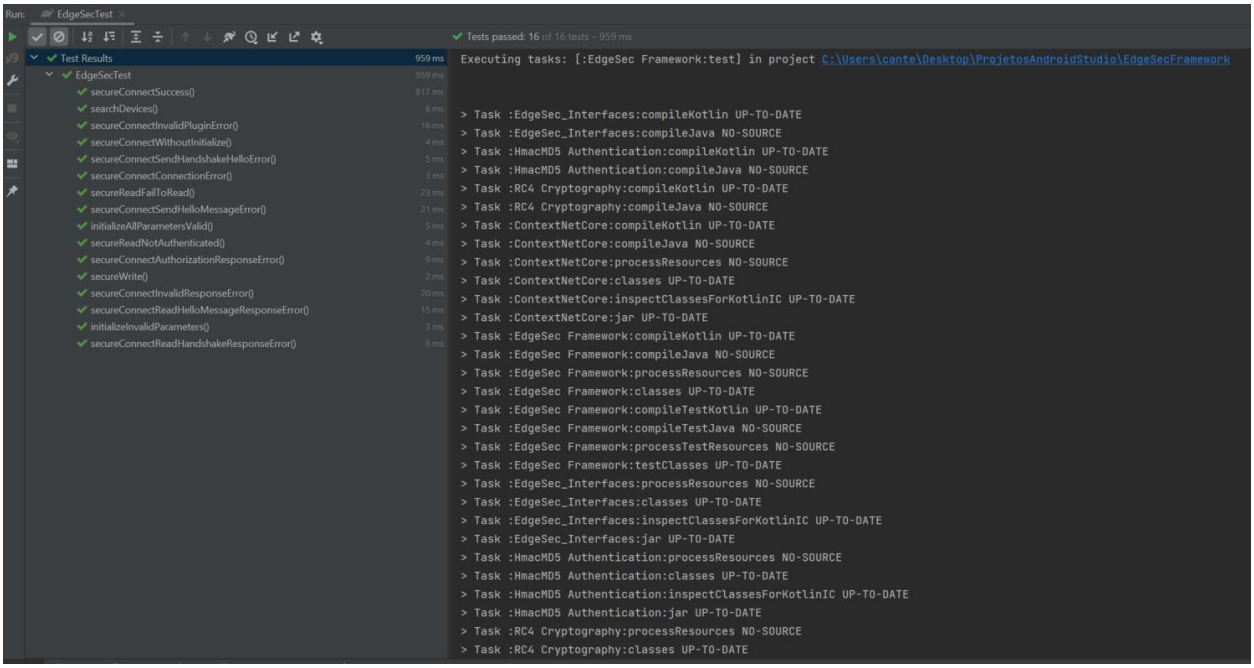
Nesta seção serão descritos os testes de controle de qualidade que serão feitos no programa, assim como o laudo de execução com os resultados. O critério de testes utilizado é o de testes unitários automatizados. Serão utilizadas bibliotecas Java/Kotlin para construção dos casos de teste, como JUnit e Mockito.

Testes unitários da classe EdgeSec

- Teste de inicialização do framework:
    - Descrição: Deve verificar se o framework consegue ser importado e inicializado corretamente.
    - Resultado esperado: Inicialização deve ser feita sem erros, usando os plugins como parâmetro.
- Teste de busca de dispositivos:
    - Descrição: Deve verificar se a funcionalidade de busca de dispositivos nas redondezas através do framework retorna valores de IDs.
    - Resultado esperado: Um observável contendo uma lista de IDs como Strings.
- Teste de conexão segura:
    - Descrição: Deve verificar se a funcionalidade de conexão segura funciona como esperado. Deve testar diferentes possibilidades de erro, como erro de conexão, erro de autorização, erro de troca de mensagens handshake, etc.
    - Resultado esperado: Verdadeiro se a conexão foi estabelecida corretamente, e falso se a conexão não foi estabelecida, retornando no observável a mensagem de erro ocorrida.
- Teste de leitura segura:
    - Descrição: Deve verificar se a leitura de um dado de um dispositivo conectado é feita conforme esperado.

    - Resultado esperado: Um valor correto caso a leitura seja feita em um dispositivo existente e já conectado, e um valor nulo caso contrário.
- Teste de escrita segura:
    - Descrição: Deve verificar se a escrita de um dado de um dispositivo conectado é feita conforme esperado.
    - Resultado esperado: Verdadeiro caso a escrita seja feita em um dispositivo existente e já conectado, e falso caso contrário.
- Teste de handshake:
    - Descrição: Deve verificar se o handshake para troca de IDs entre o Mobile Hub e um dispositivo é feito conforme esperado.
    - Resultado esperado: O handshake deve ser bem-sucedido em um cenário onde o ID do dispositivo é existente. Deve falhar caso o ID do dispositivo não exista ou não haja protocolos compatíveis.
- Teste de criação da HelloMessage:
    - Descrição: Deve verificar se a HelloMessage é montada conforme esperado.
    - Resultado esperado: A HellloMessage retornada deve ser válida dado que os parâmetros também o sejam.
- Teste de criação do AuthPackage:
    - Descrição: Deve verificar se o pacote de autenticação é montado conforme esperado.
    - Resultado esperado: O pacote de autenticação retornado deve ser válido.
- Teste de troca da HelloMessage:
    - Descrição: Deve verificar se o envio e recebimento da resposta da HelloMessage funciona conforme esperado.
    - Resultado esperado: Caso o dispositivo no qual a mensagem esteja sendo trocada seja compatível, e as mensagens sejam válidas, deve finalizara troca da HelloMessage de forma bem-sucedida. Se alguma destas condições não for verdadeira, deve retornar erro.

A imagem abaixo mostra o laudo de execução dos testes acima, todos executados na IDE Android Studio.



As funções de teste estão todas dentro do arquivo EdgeSecTest.kt. Este arquivo está localizado no seguinte diretório (partindo do diretório raiz do projeto):

EdgeSec Framework/src/test/java/br/pucrio/inf/lac/edgesec/EdgeSecTest.kt

Documentação para o usuário

Conforme descrito anteriormente, os usuários alvo deste projeto são desenvolvedores de sistemas IoT e fabricantes de dispositivos IoT. Logo, são usuários de perfil técnico, que utilizarão o framework como uma biblioteca, incorporando em seus projetos os módulos desenvolvidos como dependências.

O passo a passo para se utilizar o framework em um gateway de um sistema IoT é descrito a seguir:

1. Obter o arquivo JAR compilado dos seguintes módulos:
    a. EdgeSec Interfaces
    b. EdgeSec Framework
    c. RC4

    d. HmacMD5
    e. BLE
2. Incorporar os JAR obtidos no passo anterior dentro do projeto do gateway.
3. Importar os módulos descritos no passo 1 dentro do código do gateway.
4. Instanciar as classes EdgeSec, RC4, HmacMD5 e BLE.
5. Chamar o método de inicialização do EdgeSec, passando como argumento as instâncias dos plugins criadas no passo anterior.
6. Utilizar os outros métodos providos pelo framework para escanear, se conectar e trocar dados de forma segura com dispositivos nas redondezas.