TP prise TPLink HS110 en java

La prise HS110 est une prise qu'on peut contrôler via le WIFI. On peut en particulier mettre en marche ou arrêter la prise, récupérer la consommation, …

1. Préambule : client/serveur TCP

On donne un exemple basique d'échange TCP en java (https://systembash.com/a-simple-java-tcp-server-and-tcp-client/)

// serveur TCP simple

```
ServerSocket socketEcoute = new ServerSocket(9999); // socket d'écoute port 9999
Socket SocketDialogue = socketEcoute.accept();//attente connexion,récup socket dialogue
BufferedReader inFromClient =
new BufferedReader(new InputStreamReader(SocketDialogue.getInputStream()));
DataOutputStream outToClient =
new DataOutputStream(SocketDialogue.getOutputStream());
String messRecu = inFromClient.readLine();
System.out.println("Received: " + messRecu);
outToClient.writeBytes("Réponse serveur ");
outToClient.Close() ;
inFromClient.Close() ;
SocketDialogue.Close() ;
socketEcoute.Close() ;
```

//Client TCP Simple

```
Socket clientSocket = new Socket("IP_Serveur", 9999); //connect serveur port9999
DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());
BufferedReader inFromServer = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
outToServer.writeBytes(("salute \n");
String reponse= inFromServer.readLine();
System.out.println("FROM SERVER: " + reponse);
outToServer.Close();
inFromServer.Close();
clientSocket.close();
```

rem : Pour info ,comparer avec un échange UDP

```
//attente message et envoi réponse en UDP
int port = 9998;
byte buffer[] = new byte[1000];
DatagramSocket socket = new DatagramSocket(port);
DatagramPacket data = new DatagramPacket(buffer,buffer.length);
socket.receive(data);
System.out.println(data.getAddress()); //IP de l'expéditeur
socket.send(("reponse");

// envoi message et attente réponse
int port = 9998;
InetAddress serveur = InetAddress.getByName("172.16.64.12");
int length = argv[1].length();
String mess = "coucou";
byte buffer[] = mess.getBytes();
DatagramPacket dataSent = new DatagramPacket(buffer,buffer.length(),serveur,port);
DatagramSocket socket = new DatagramSocket();
socket.send(dataSent); // Envoi message
DatagramPacket dataRecieved = new DatagramPacket(new byte[length],length);
socket.receive(dataRecieved); //attente réponse
System.out.println("Data recieved : " + new String(dataRecieved.getData()));
System.out.println("From : " + dataRecieved.getAddress() + ":" + dataRecieved.getPort());
```

En vous inspirant des exécutables fournis :

Ecrire une application ClientTCP simple permettant de se connecter sur un serveur , lui envoyer un message, recevoir un message.

Ecrire un serveurTCP (mono Client) qui permettant d'attendre un client, afficher l'IP, recevoir ou envoyer un message.

2. Installation de la prise

(Normalement prise déjà installée avant le TP, utiliser l'appli pour tester la prise)

- Télécharger l'appli mobile TPLink Kasa
- Brancher la prise (clignotement vert-ambre au bout de qq secondes)
- Lancer l'appli, ajouter la prise et suivre les instruction (en particulier donner la clé WIFI)
- Si la prise est connectée : la led passe au vert

Sur l'appli mobile on voit l'adresse MAC de la prise , mais pas son IP.
Utiliser « Advance IP Scanner » pour retrouver l'adresse IP de la prise.

3. Principe protocole

La prise attend une connexion TCP sur le port 9999.
Il faut ensuite envoyer une requête et la prise répond.
Les requêtes utilisent un balisage JSON. La requête doit être cryptée avant l'envoie.
La réponse reçue utilise aussi le balisage JSON, et elle est cryptée (il faut donc décrypter !).

4. Principales requêtes

a. Demande informations

Requête :

```
{"system":{"get_sysinfo":{}}}
```

La réponse contient les caractéristiques :

```
{"system":{"get_sysinfo":{"err_code":0,"sw_ver":"1.0.10 Build 160304 Rel.082646","
hw_ver":"1.0","type":"IOT.SMARTPLUGSWITCH","model":"HS110(FR)","mac":"50:C7:BF:08:
6F:EF","deviceId":"8006D98E5F3A8C3C166B8B63D8C3F7B01782332C","hwId":"797CB3D09D9A8
3B1C43A623C6DD103C2","fwId":"848F75775BF7C417E48DA916C0B3FD58","oemId":"0D13F703F0
4C040DC3D27DB248C01A1A","alias":"Nsa Plug","dev_name":"Wi-Fi Smart Plug With Energ
y Monitoring","icon_hash":"","relay_state":1,"on_time":1339,"active_mode":"schedul
e","f ……
```

err_code	0 = no error
sw_ver	Software Version, currently "1.0.8 Build 151101 Rel.24452"
hw_ver	Hardware Version, currently "1.0"
type	"smartplug"
model	"HS110(EU)"
mac	MAC address of WiFi interface
deviceID	Device ID, 40 char hex string
hwID	Hardware ID, 32 char hex string
fwID	Firmware ID, 32 char hex string
oemID	OEM ID, 32 char hex string
alias	Text description, e.g. "Basement Lights"
dev_name	"Wi-Fi Smart Plug With Energy Monitoring"
icon_hash	Hash for custom picture
relay_state	0 = off, 1 = on
on_time	0
active_mode	"schedule" for Schedule Mode
feature	"TIM:ENE" (Timer, Energy Monitor)
updating	0 = not updating
rssi	Signal Strength Indicator in dBm (e.g. -35)
led_off	0 = LED on (default), 1 = LED turned off
latitude	Optional geolocation information
logitude	Optional geolocation information

b. Commande M/A de la prise

Requête :

```
{“system":{"set_relay_state":{"state":1}}}
ou
{“system":{"set_relay_state":{"state":0}}}
```

c. Lire les mesures sur la prise (courant, tension,…)

```
{"emeter":{ "get_realtime":null } }
```

La réponse

```
{"emeter":{"get_realtime":{"current":0.013640,"voltage":246.750307,"power":0,"total":0.001000,"err_code":0}}}
```

Courant en ampères, tension en volt, puissance en watt, total= total puissance consommée depuis la dernière initialisation

5. fonctions de cryptage et décryptage

```
private  byte[] encrypt(String message) {
      byte[] data = message.getBytes(ASCII);
      byte[] enc = new byte[data.length + 4];
      ByteBuffer.wrap(enc).putInt(data.length);
      System.arraycopy(data, 0, enc, 4, data.length);
      byte key = KEY; // KEY vaut 0xAB
      for (int i = 4; i < enc.length; i++) {
         enc[i] = (byte) (enc[i] ^ key);
         key = enc[i];
      }
      return enc;
   }
```

```
private  String decrypt(byte[] data) {
   if (data == null) {
      return null;
   }
   byte key = KEY; // KEY vaut 0xAB
   byte nextKey = 0;
   for (int i = 4; i < data.length; i++) {
      nextKey = data[i];
      data[i] = (byte) (data[i] ^ key);
      key = nextKey;
   }
   return new String(data, 4, data.length - 4, ASCII);
}
```

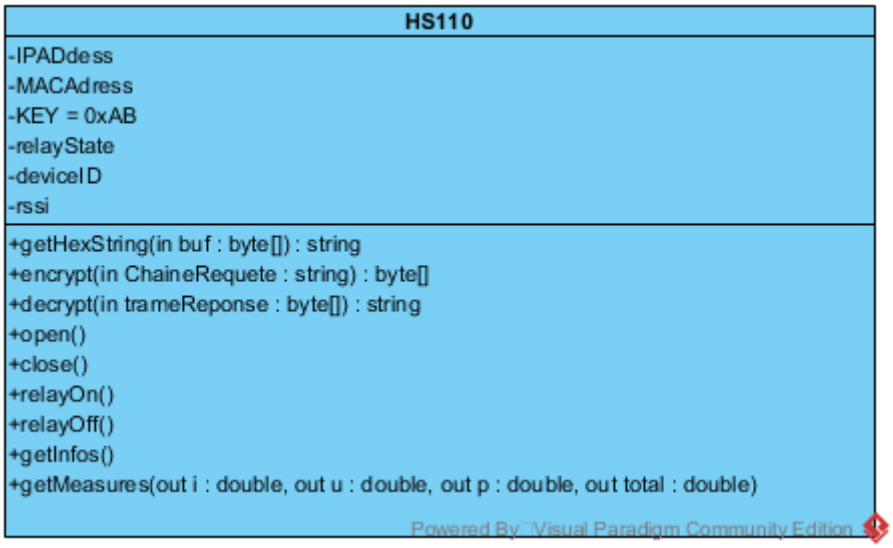
On donne aussi une méthode permettant de convertir un tableau de byte en String , afin de pouvoir l’afficher (utile pour la mise au point !)

```
public static String getHexString(byte[] b) {
      String result = "";
      for (int i = 0; i < b.length; i++) {
```

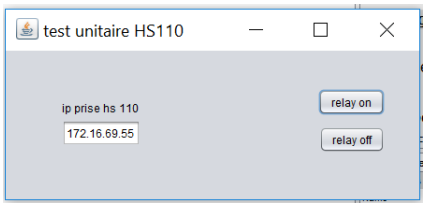
```
        result += Integer.toString((b[i] & 0xff) + 0x100, 16).substring(1);
    }
    return result;
}
```

6. Codage classe HS110

A partir d'un client TCP simple écrire une classe HS110 (sans getinfos() et getMeasures())



Ecrire un programme de test qui valide cette classe



7. JSON

Visualiser la réponse sur getInfos(). Chercher comment on peut extraire les données de la réponse.

Ecrire les méthodes getInfos() et getMeasures(). Compléter le test.