物联网安全课程实验报告

实验三

实验名称：物联网设备加密通信设计与实现

姓名：郭裕彬 杨雄峰 于洋淼

小组：郭裕彬 杨雄峰 于洋淼

学号：2114052 2113723 2113644

专业：____物联网工程____

提交日期：__2023.11.08____

一、实验目的

了解目前主流的基于云的物联网通信原理，学会使用基本的密码学工具，并在消费物联网应用场景下构建安全加密通信方案

二、实验要求及要点

分组完成实验内容，合作撰写实验报告，回答问题。

问题：

1）智能家居设备的使用与工业控制系统面临风险有何差异？

2）列举几种可用来实现加密通信的常见密码算法，并进行对比分析。

3）什么是虚拟机的 NAT 模式、桥接模式、Host-only 模式？

要点：

1）实验目标

2)组员分工情况(须有一位组员专门从攻击者视角做安全评估，无需参与设计)

3）方案设计

4）方案实现（包括网络拓扑，实现技术细节，功能效果演示，等等）

5）系统方案安全性评估（包括系统设计的不足和未来改进思路等）

6）每位成员的收获与感悟，体会“红蓝”对抗对构建安全系统的作用

7）提交源代码

组员分工情况

郭裕彬：实验初始环境搭建、辅助非安全部分代码编写、安全评估

杨雄峰：实验环境搭建、物联网设备模拟、协议设计

于洋淼：协议设计、文本处理、通信加解密实现

三、实验内容

方案设计

使用 EMQX CLOUD 在线部署 MQTT 服务器，在多台计算机上运行基于 paho-python 库编写的用户端程序和智能设备模拟程序，实现核心的智能家居功能和基础防御功能，并评估安全状况。

方案实现

采取 **AES + RAS** 的加密手段，其中 AES 用于加密数据，RAS 用于加密 AES 的密钥。

加密过程如下：

1. 参数是要加密的文本和已经得到的 RAS 公钥
2. 获取时间戳，用于防治重放攻击
3. 随机生成 AES 密钥
4. 使用 AES 密钥加密文本
5. 将 AES 密钥与时间戳合并
6. 使用 RAS 公钥加密合并后的 AES 密钥
7. 返回加密后的 AES 密钥和加密后的文本

```
def encryption(m, public_key_text):

    # 获取时间戳
    timestamp = int(time.time())
    timestamp_bytes = timestamp.to_bytes(8, byteorder="big")
    # 生成随机 aes 密钥
    aes_key = get_random_bytes(16)

    m = m.encode('ISO-8859-1') # 转换为字节流
    aes = AES.new(aes_key,AES.MODE_ECB)
    encry_m = aes.encrypt(pad(m,16)) # ECB 需要填充
```

```
    combined_key = timestamp_bytes + aes_key # 将时间戳和aes 密钥合并

    public_key = RSA.import_key(public_key_text)
    encry_key = PKCS1_OAEP.new(public_key).encrypt(combined_key)
    # 由于设计的数据包使用的是string，所以需要先由字节流转换为string
    encry_key = encry_key.decode('ISO-8859-1')
    encry_m = encry_m.decode('ISO-8859-1')

    return encry_key, encry_m
```

解密过程如下：

1. 参数是加密后的密文、加密后的 AES 密钥和 RAS 私钥
2. 使用 RAS 私钥解密 AES 密钥
3. 分离 AES 密钥和时间戳
4. 使用 AES 密钥解密密文
5. 返回解密后的文本和时间戳

```
def decryption(encry_key, encry_text, private_key_text):

    # string 转换为字节流
    encry_key = encry_key.encode('ISO-8859-1')
    encry_text = encry_text.encode('ISO-8859-1')

    combined_key = PKCS1_OAEP.new(private_key).decrypt(encry_key)

    # 获取时间戳和aes 密钥
    extracted_timestamp = int.from_bytes(combined_key[:8], byteorder=
"big")
    aes_key = combined_key[8:]

    aes = AES.new(aes_key,AES.MODE_ECB)
    m = unpad(aes.decrypt(encry_text),16) # 去除填充

    m = m.decode('ISO-8859-1')
    return m, extracted_timestamp
```

数据包设计

数据包格式如下：

- des 是目的客户端(用户或者是设备)，如果接收到不是自己的数据包就丢弃
- encry_key 是加密过的 aes 密钥
- encry_text 是由 aes 密钥加密过的数据

```
packet = {
    'des': des,
    'encry_key': public_key_test,
    'encry_text': message
}
```

交互流程

1. 用户和设备分别连接到 mqtt 服务器。设备一旦登录就订阅自己 user_list 中的所有 user(比如 air_condition 可以由 user1 和 user2 共同控制)
2. 用户(user) 登录，输入想要交互的设备(device)
3. 如果输入的 device 合法(存在且允许该用户访问)，使用 getRSAKey()得到一对 RSA 密钥，将公钥发送给 device(明文传输，这个数据包相当于请求连接)，将私钥保存在本地
4. device 收到用户发送的含有 RAS 公钥的数据包，保存对应用户的公钥(不同用户的公钥分开存储)；使用同样方法生成一对 RAS 密钥，将公钥发送给用户，将私钥保存在本地
5. 此时用户和设备都有了对方的公钥，可以开始加密通信了
6. 每次发送的数据包中都有加密过后的 aes 密钥+时间戳，假如小于上一次的时间戳就丢弃，防止重放攻击

连接云服务器

```
client = mqtt.Client(device_name)
client.username_pw_set(device_name, '123456')
client.on_connect = on_connect
client.on_message = on_message
client.on_publish = on_publish
client.on_disconnect = on_disconnect
client.on_unsubscribe = on_unsubscribe
client.on_subscribe = on_subscribe
client.connect('8.140.62.20', 1883, 600)
```

获取 RSA 密钥对

```
def getRSAKey():
    key = RSA.generate(1024)
    private_key = key.export_key().decode()
    public_key = key.publickey().export_key().decode()
    return private_key, public_key
```

生成数据包

```
def getPacket(des , message, public_key_test =''):
    if public_key_test == '': # 如果是请求连接的数据包
        packet = {
            'des': des,
            'encry_key': public_key_test,
            'encry_text': message
        }
    else :
        encry_key, encry_text = encryption(message, public_key_test)
        packet = {
            'des': des,
            'encry_key': encry_key,
            'encry_text': encry_text
        }
    return json.dumps(packet)
```

设备端处理用户请求

```
def on_message(client, userdata, msg):
    global user_list
    if(msg.topic in user_list): # 如果不是合法的用户就丢弃
        proc_message(msg.payload.decode('utf-8') , msg.topic)

def proc_message(s ,res):
    global public_key, private_key, device_name, last_time_stamp
    rec_data = json.loads(s)
    des_device = rec_data['des']

    if des_device == device_name: # 如果不是发给自己的数据包就丢弃
        if rec_data['encry_key'] == '': # 如果是请求连接的数据包
            public_key[res] = rec_data['encry_text'] # 保存对应用户的公
钥
```

```
            private_key[res], pub = getRSAKey()
            s = getPacket(res, pub)

            client.publish(device_name, s) # 将自己的公钥发送给用户
            return
        else :
            rec_message, ex_timestamp = decryption(rec_data['encry_key
'], rec_data['encry_text'], private_key[res])# 解密数据包
            if ex_timestamp < last_time_stamp: # 防治重放攻击
                print('消息过期')
                return
            last_time_stamp = ex_timestamp
    else:
        return
    message_handle(rec_message ,res) # 交给每个设备特有的处理函数
    return
```

用户端

```
while True:
    while True:
        des_device = input("请输入目标设备(输入 all-device 查看当前所有设
备):") # 输入目标设备
        if des_device == 'all-device':
            print(device_list)
            continue
        elif des_device in device_list:
            # 向目标设备发送连接请求
            private_key, pub= getRSAKey()
            cur_device = des_device
            client.subscribe(des_device, qos=0)
            s = getPacket(des_device, pub,'')
            client.publish(topic='user1', payload=s, qos=0, retain=Fal
se)
            time.sleep(2) # 防止还没接收到设备的公钥就发送数据包
            break
        else:
            print("设备不存在，请重新输入")
            continue

    while True:
        message = input("请输入指令(输入 help 获取帮助):")
```

```
        if message == 'help':
            print(help_string)
        elif message == 'quit':
            break
        else:
            send_pack = getPacket(des_device, message, public_key)
            client.publish(topic='user1', payload=send_pack, qos=0, re
tain=False)
            time.sleep(2)

def on_message(client, userdata, msg):
    global cur_device, private_key, public_key
    if(msg.topic != cur_device): return

    rec_data = json.loads(msg.payload.decode('utf-8'))
    if(rec_data['des'] != 'user1'): return
    if rec_data['encry_key'] == '':
        public_key = rec_data['encry_text'] # 保存设备的公钥
        return
    else:
        proc_message(rec_data)

def proc_message(data):
    global cur_device, private_key, public_key, last_time_stamp
    # print(private_key)
    message, ex_timestamp = decryption(data['encry_key'], data['encry
_text'], private_key)
    if ex_timestamp < last_time_stamp: # 防治重放攻击
        print('消息过期')
        return
    last_time_stamp = ex_timestamp
    print(message)
```

不同设备的功能实现

空调

```
def message_handle(s, res):
    global state, model, temperature
    # print('here'+s)
    if(s!='on' and state =='off'): # 如果空调没开就不处理
        return
    if s == 'on': # 打开空调
```

```
        state = 'on'
    elif s == 'off': # 关闭空调
        state = 'off'
    elif s == 'cold': # 设置为制冷
        model = 'cold'
        temperature = 24
    elif s == 'warm': # 设置为制热
        model = 'warm'
        temperature = 24
    elif s == 'up': # 温度加一
        temperature += 1
    elif s == 'down': # 温度减一
        temperature -= 1
    elif re.match(r'set \d+', s): # 设置温度
        match = re.match(r'set (\d+)', s)
        temperature = int(match.group(1))
    elif s == 'get':
        send_state(res)

    # 温度范围限制
    if temperature > 30:
        temperature = 30
    elif temperature < 16:
        temperature = 16
```

安全评估

- 为了避免重放攻击，设计方使用了时间戳确认机制，但使用机制过于简单，只确认时间戳是否小于前一条指令来判断，对于同一条指令如果攻击者截获了该条报文重发，消息仍会被通过并执行。
- 设计方使用了 AES 密钥进行加密，但双方公钥的传输是公开的，同时没有增加额外的确认过程，当攻击者截获双方交互的公钥后，便存在伪装成其中一方与另一方交互的可能。
- 最开始用户端登录到云的账号密码验证是明文传输，攻击者可以很容易地获取到用户的账号和密码。
- 设计方使用的结构简单、采用简单的字符串拼接，没有增加伪装用的无用数据，通过观察分析指令交互的报文，攻击者可能可以发现报文的大

致结构。如果攻击者能够获得相同型号的设备，或者能够知道每一次报文对应了什么操作，就能够观察到同一种指令中某些位置数据变化的规律，从而推导出时间戳的存在。攻击者获取到了公钥，就能够通过同一类型的设备获取到使用同一公钥加密后的时间戳数据，伪装成攻击的用户端与设备进行交互。

- 攻击者如果能够插入自己作为攻击方，通过上述测试发现报文的结构后，就可以实现截获和篡改通信的目的。设计方没有使用方法来保证数据包的完整性，也没有使用 SSL/TLS 等传输层安全协议，因此存在着篡改部分内容如指令类型的可能。

- 如果攻击者能够使用用户的设备和客户端，那么由于设计者没有实现私钥在本地的加密存储，攻击者能够轻易获取并得到内容，此时设备和用户之间的交互对攻击者来说就是透明的了。

四、回答问题

1. 智能家居设备的使用与工业控制系统面临风险有何差异？

1）智能家居一般通过连接到云进行用户端和设备端的交互，云端服务器和消息传输时存在风险；工业控制系统一般部署在本地，使用专用网络和通信协议，采用更强的网络隔离和安全措施，主要风险关注点在可靠性和稳定性上。

2）智能家居服务于人的生活，安全方面重点在于防范隐私侵犯、数据泄露和未经授权的访问以及一些人身环境安全的保证，关注数据的保护，面临的攻击更多处在“窥探”层面；工业控制系统服务于社会的生产，安全方面的重点在于保障生产过程的正常运作，需要面对对各种设备的控制攻击、篡改攻击等。

2. 列举几种可用来实现加密通信的常见密码算法，并进行对比分析。

1）对称加密算法：

常见算法：AES、DES、3DES。

优点：速度快，适用于大数据量的加密，适合网络通信中的数据保护。

缺点：需要共享密钥，密钥管理可能会变得复杂。不提供身份验证或密钥交

换功能。

2）非对称加密算法：

常见算法：RSA、ECC、DSA。

优点：提供密钥交换和身份验证功能，不需要共享密钥。安全性较高。

缺点：加密和解密速度较慢，不适用于大数据量的加密。

3）哈希函数：

哈希函数通常用于验证数据完整性，而不是加密数据。它们将数据映射到固定长度的哈希值，并且相同的输入始终产生相同的哈希值，常见的算法有 SHA-256、SHA-3、MD5。

优点：快速、不可逆，适用于数据完整性验证。

缺点：不提供机密性，无法逆向计算原始数据。

3. 什么是虚拟机的 NAT 模式、桥接模式、Host-only 模式？

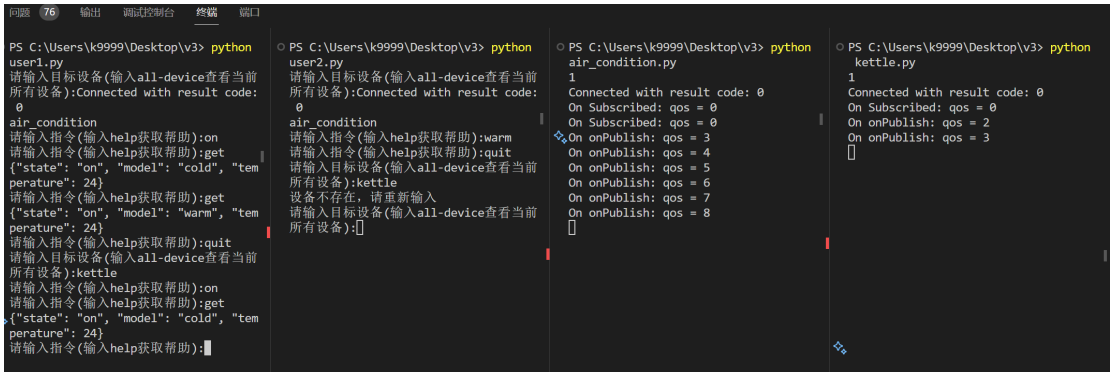
桥接模式就是将主机网卡与虚拟的网卡利用虚拟网桥进行通信。类似于把物理主机虚拟为一个交换机，所有桥接设置的虚拟机连接到这个交换机的一个接口上，物理主机也同样插在这个交换机当中，所以所有桥接下的网卡与网卡都是交换模式的，相互可以访问而不干扰。在桥接模式下，虚拟机 IP 地址需要与主机在同一网段，如果需要联网，则网关与 DNS 需要与主机网卡一致。

在 NAT 模式中，主机网卡直接与虚拟 NAT 设备相连，然后虚拟 NAT 设备与虚拟 DHCP 服务器一起连接在虚拟交换机 VMnet8 上，这样就实现了虚拟机联网。VMnet8 网卡是为了实现主机与虚拟机之间的通信。NAT 模式下主机通过 VMnet8 虚拟网卡为虚拟机分发地址。所以虚拟机和主机不在同一网段下，可以理解为主机是虚拟机的"上级"。

仅主机模式可有看成是 NAT 模式去除了虚拟 NAT 设备，然后使用 VMware Network Adapter VMnet1 虚拟网卡连接 VMnet1 虚拟交换机来与虚拟机通信的，Host-Only 模式将虚拟机与外网隔开，使得虚拟机成为一个独立的系统，只与主机相互通讯。

五、实验结果

创建两个用户 user1 和 user2，创建三个设备 air_condition、kettle 和 lamp。user1 可以控制 air_condition 和 kettle，user2 可以控制 air_condition 和 lamp。



上图中可以看到，user1 和 user2 均可连接到 air_condition，user2 发送修改的指令后，air_condition 做出响应，user1 再 get(获取设备状态)可以发现 model 已经由 cold 修改为 warm

查看 wireshark 抓包结果，可以看到传输的 aes 密钥和密文均为加密后的

六、收获感悟

于洋淼

本次实验中我学习了 MQTT 协议，了解了发布-订阅模型；学习了 RAS 和 AES 加密解密的原理及使用；学习了重放攻击及其防治手段；学习了如何设计物联网设备的交互方案。

郭裕彬

本次实验中参与了实验环境的搭建，主要作为蓝方对其余组员设计的系统进行评估，感受到了不同视角导致的对系统安全性能认知的 差异，认识到了综合多方视角来设计和完善系统功能的必要性。

杨雄峰

本次实验搭建了云服务器，建立了设备与服务器的初步连接，熟悉了 MQTT 协议的使用。设计物联网设备的基本功能和通信协议，对物联网设备的数据采集、数据传输、远程控制等核心功能有更深入的理解。