物联网安全第三次实验

- 实验名称：物联网设备加密通信设计与实现
- 小组：方沐华 秦泽斌
- 姓名：方沐华 秦泽斌
- 学号：2212288 2212005
- 专业：物联网工程
- 提交日期：2024.11.7

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（1-3人）完成实验内容，合作撰写实验报告（一组只交一份即可），回答问题，且报告内容至少包括如下**要点**:

- **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：**

    - 实验目标
    - 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）
    - 方案设计
    - 方案实现（包括网络拓扑，实现技术细节，功能效果演示，等等）
    - 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）
    - 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
    - 提交源代码

三、 实验内容

1. 实验目标

- 基于MQTT协议，编程模拟实现智能家居应用场景下用户控制物联网设备的原型程序。原型系统至少包含如下功能：
  - 支持至少2个用户，至少3个物联网设备（空调、灯泡 、插座不同类型）同时在线（注：物联网设备硬件功能仅用代码模拟即可，如输出“on”表示灯打开）
  - 在不同的实体计算机（或虚拟机、手机）上部署MQTT服务器与MQTT客户端
  - 利用密码学知识与工具，查询相关资料，使该系统能抵御指令重放攻击。
  - 模拟核心的智能家居功能：用户操控设备、查看设备状态、设备绑定（可选功能：设备分享、设备注册）
- 从攻击者的视角对本组的原型系统实现开展安全评估
  - 提示：请考虑不同的威胁模型，即不同的应用场景与敌手能力。例如，攻击者可能合法购买一个同款设备逆向分析，甚至可能共享使用过目标设备。

2. 组员分工

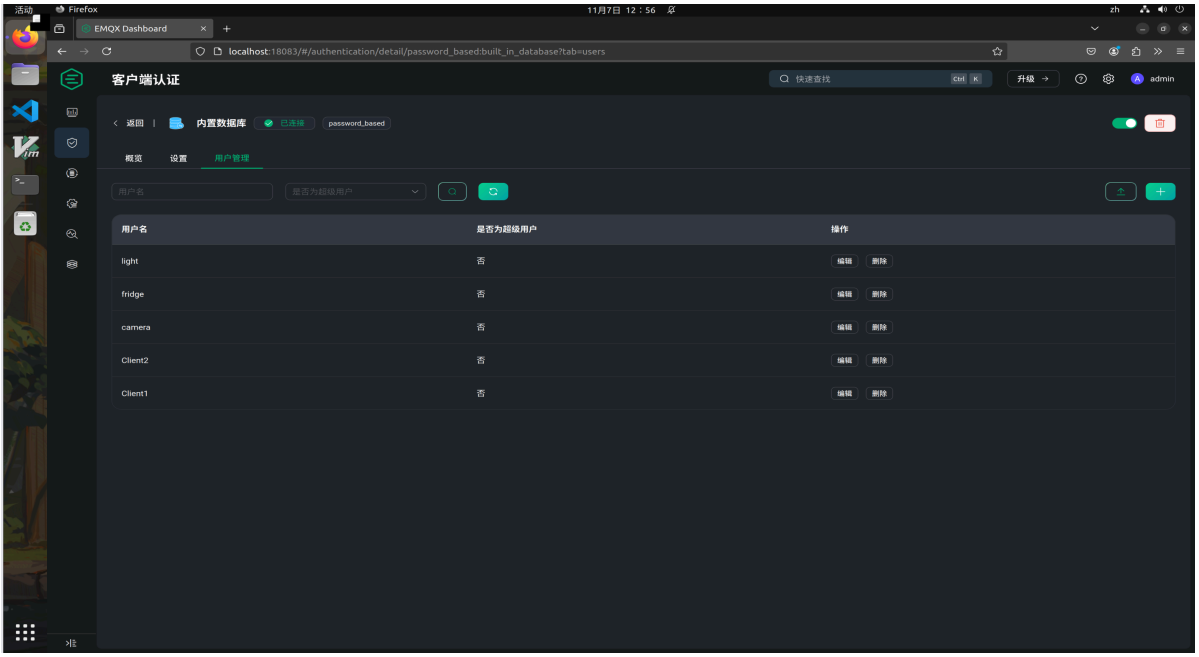
- **秦泽斌**：编写程序并进行检查，撰写报告方案设计与实现部分，针对不足进行修改。
- **方沐华**：负责从攻击者视角做安全评估，报告撰写,总结不足并提出改进思路。

3. 方案设计

（1）EMQX服务器搭建

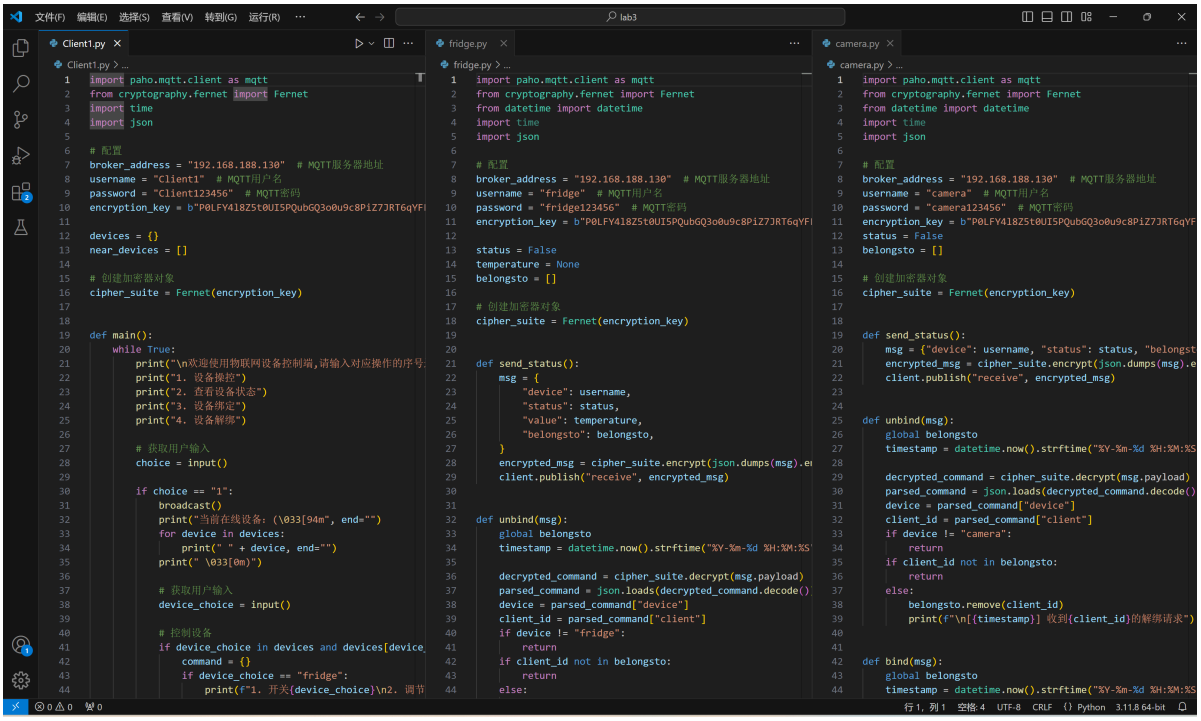
本次实验我们使用的**MQTT**服务器为**EMQX**服务器（EMQX 是一种高性能、分布式的 MQTT 消息中间件服务器，专为物联网 (IoT) 和实时消息传递场景设计。它支持 MQTT、MQTT-SN、CoAP、HTTP、WebSocket 等多种协议，并允许在设备间进行低延迟的实时数据传输）服务器架设在虚拟机内（Linux Ubuntu系统）。

- 下载EMQX，安装并进入`EMQX Dashboard 管理控制台`，完成物联网设备和用户控制端的账号添加。设备端包括light，fridge以及camera并且设有两个用户控制端。



（2）python代码模拟用户控制端和物联网设备端

python源代码详见**GitHub仓库**[lalaZZB/NKU_IoT-Security: 南开大学物联网安全课程实验仓库](https://github.com/lalaZZB/NKU_IoT-Security)，这里仅展示部分代码，并对核心部分进行讲解。



【1】用户控制端：（以Client1为例）

1. 配置与初始化

```
broker_address = "192.168.188.130"
username = "Client1"
password = "Client123456"
encryption_key = b"P0LFY4l8Z5t0UI5PQubGQ3oOu9c8PiZ7JRT6qYFLy38="

devices = {}  # 存储已绑定设备及其状态
near_devices = []  # 存储附近未绑定设备
cipher_suite = Fernet(encryption_key)  # 创建加密器对象
```

这里定义了 MQTT 服务器地址和客户端的用户名与密码。同时，还定义了 `devices` 和 `near_devices` 列表来管理设备。

2. `main` 函数：主菜单

```
def main():
    while True:
        # 打印操作菜单
        print("\n欢迎使用物联网设备控制端,请输入对应操作的序号进行操作：")
        print("1. 设备操控")
        print("2. 查看设备状态")
        print("3. 设备绑定")
        print("4. 设备解绑")

        choice = input()  # 获取用户选择
```

用户可以通过输入数字选择操作。之后的操作涉及四个主要功能：设备操控、状态查看、设备绑定和解绑。

- 设备操控

```
if choice == "1":
    broadcast()
    print("当前在线设备：(\033[94m", end="")
    for device in devices:
        print(" " + device, end="")
    print(" \033[0m)")
    # 控制设备
    device_choice = input()
    if device_choice in devices and devices[device_choice]:
        command = {}
        # 判断设备类型，并提供相应控制选项
```

广播在线设备：使用 `broadcast` 函数发送在线设备请求。

控制设备：如冰箱支持温度调节，选择开关控制等。每个命令会被加密后通过 MQTT 发送。

- 查看设备状态

```
elif choice == "2":
    broadcast()
    if not devices:
        print("\033[91m\n未绑定设备\033[0m")
    else:
        # 显示每个已绑定设备的状态（例如，开关状态和温度）
```

这里调用 `broadcast` 函数获取设备最新状态。若有设备绑定，将显示每台设备的状态。

- 设备绑定与解绑

```
elif choice == "3":
    # 绑定设备
    device = input()
    bind(device)
    broadcast()
elif choice == "4":
    # 解绑设备
    device = input()
    unbind(device)
    broadcast()
```

用户可以输入设备名称绑定或解绑设备。`bind` 和 `unbind` 函数会加密相关命令并通过 MQTT 发布。

3. `broadcast`、`receive`、`bind`、`unbind` 函数

- **broadcast**：将查询请求广播给所有设备。
- **receive**：接收设备返回的状态信息并解密处理，更新设备状态。
- **bind/unbind**：分别发送绑定和解绑请求。

4. MQTT 事件处理

```
def on_connect(client, userdata, flags, rc):
    # MQTT连接成功后订阅消息
    client.subscribe("receive")
    command = {"client": username}
    encrypted_command = cipher_suite.encrypt(json.dumps(command).encode())
    client.publish("broadcast", encrypted_command)

def on_message(client, userdata, msg):
    if msg.topic == "receive":
        receive(msg)
```

- **on_connect**：连接到 MQTT 服务器时，订阅 `receive` 主题，用于接收设备的状态。
- **on_message**：处理接收到的消息，通过 `receive` 函数解析消息内容。

5. MQTT 客户端连接与启动

```
client = mqtt.Client(username)
client.username_pw_set(username, password)
client.on_connect = on_connect
client.on_message = on_message
client.connect(broker_address, 1883, 20)
client.loop_start()
time.sleep(0.5)
main()
client.loop_stop()
```

- **创建客户端对象**：设置 MQTT 连接参数，注册连接和消息处理函数。
- **启动消息循环**：使客户端保持连接并接收服务器发布的消息，执行用户操作。

【2】物联网设备端：（以camera为例）

1. 设备状态发送函数 `send_status`

```
def send_status():
    msg = {"device": username, "status": status, "belongsto": belongsto}
    encrypted_msg = cipher_suite.encrypt(json.dumps(msg).encode())
    client.publish("receive", encrypted_msg)
```

该函数将设备的状态消息加密并发布到 `receive` 主题，以便客户端获取设备的状态和绑定信息。

2. 解绑处理函数 `unbind`

```
def unbind(msg):
    global belongsto
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    decrypted_command = cipher_suite.decrypt(msg.payload)  # 解密消息
    parsed_command = json.loads(decrypted_command.decode())
    device = parsed_command["device"]
    client_id = parsed_command["client"]

    if device == "camera" and client_id in belongsto:
        belongsto.remove(client_id)
        print(f"\n[{timestamp}] 收到{client_id}的解绑请求")
```

- **解密消息**：解密并解析接收到的解绑请求。
- **检查并解绑**：如果请求解绑的设备是本设备且该客户端已绑定，则从 `belongsto` 列表中移除该客户端。

3. 绑定处理函数 `bind`

```
def bind(msg):
    global belongsto
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    decrypted_command = cipher_suite.decrypt(msg.payload)
    parsed_command = json.loads(decrypted_command.decode())
    device = parsed_command["device"]
    client_id = parsed_command["client"]

    if device == "camera":
        if client_id not in belongsto:
            if not belongsto:
                print(f"\n[{timestamp}] 收到{client_id}的绑定请求")
                belongsto.append(client_id)
            else:
                print(f"\n[{timestamp}] 已被其他设备绑定")
```

- **检查条件**：若设备为本设备且未绑定其他客户端，则接受绑定请求，否则提示已绑定其他客户端。

4. 控制命令处理函数 `command`

```
def command(msg):
    global status
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    decrypted_command = cipher_suite.decrypt(msg.payload)
    parsed_command = json.loads(decrypted_command.decode())

    device = parsed_command["device"]
    action = parsed_command["action"]
    client_id = parsed_command["client"]

    if device == "camera" and client_id in belongsto:
```

```
        print(f"\n[{timestamp}] 收到控制命令")
        if action == "switch":
            status = not status  # 切换开关状态
            print(f"\n[{timestamp}] 开关已{'开' if status else '关'}")
```

- **控制命令**：如果命令是 `switch`，则切换摄像头开关状态，并打印相应提示。

5. 连接和消息处理事件

```
def on_connect(client, userdata, flags, rc):
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    if rc == 0:
        print(f"\n[{timestamp}] {username}成功连接到MQTT服务器")
    else:
        print(f"\n[{timestamp}] {username}连接到MQTT服务器失败， 错误代码 %d\n", rc)

    client.subscribe("bind")
    client.subscribe("unbind")
    client.subscribe("broadcast")
    client.subscribe("command")
```

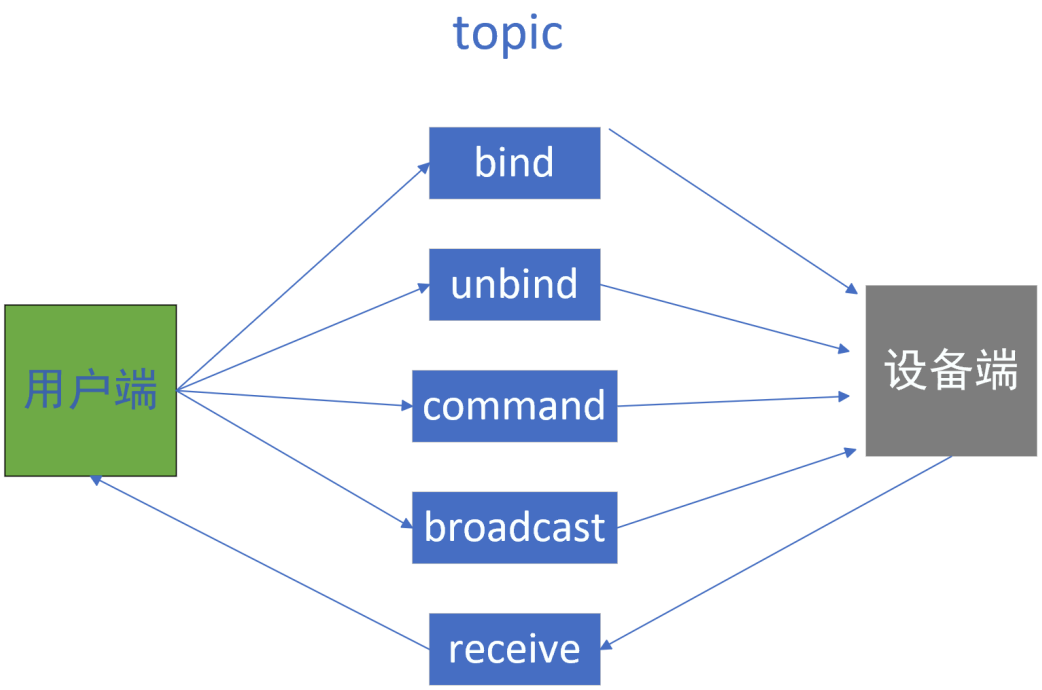
- **on_connect**：连接服务器后，订阅 `bind`、`unbind`、`broadcast` 和 `command` 主题，用于接收不同类型的控制消息。

```
def on_message(client, userdata, msg):
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    if msg.topic == "bind":
        bind(msg)
    if msg.topic == "unbind":
        unbind(msg)
    if msg.topic == "command":
        command(msg)
    if msg.topic == "broadcast":
        decrypted_command = cipher_suite.decrypt(msg.payload)
        parsed_command = json.loads(decrypted_command.decode())
        sender_id = parsed_command["client"]

        if sender_id in belongsto:
            send_status()
            print(f"\n[{timestamp}] 收到广播")
        else:
            print(f"\n[{timestamp}] 非法客户端尝试发送广播: {sender_id}")
            msg = {"device": username, "status": "NULL", "belongsto": "NULL"}
            encrypted_msg = cipher_suite.encrypt(json.dumps(msg).encode())
            client.publish("receive", encrypted_msg)
```

- **on_message**：根据主题调用相应函数处理消息。
    - **broadcast 处理**：若广播消息来自已绑定客户端，调用 `send_status` 发送设备状态；否则，返回 "非法客户端尝试" 信息。

（3）topic&subscribe设计



如上图所示，本实验**topic&subscribe设计**由5个topic组成，用户端通过向**bind，unbind，command，broadcast**发布消息（设备端会subscribe这些topic）来实现对应功能，包括绑定，解绑，控制，获取设备状态等。而**receive**主题则是用于用户端接收来自设备端的**状态信息**。至此，通过MQTT服务器，我们实现了用户端和设备端的通信。

（4）加密措施

- **对发送消息进行加密**

```
# 导入Fernet包
from cryptography.fernet import Fernet

# 加密密钥
encryption_key = b"09jyez3-73axU9OnTKKhT5DigEKqw2wutZ14z6MQwc8="

# 创建加密器对象
cipher_suite = Fernet(encryption_key)

# 加密操作
encrypted_msg = cipher_suite.encrypt(json.dumps(msg).encode())

# 解密操作
decrypted_msg = cipher_suite.decrypt(msg.payload)
```

利用Fernet加密算法对消息进行加密和解密

Fernet是一种对称密钥加密算法，它使用相同的密钥来加密和解密数据

加密原理

1. 生成密钥

   使用了一个密钥字节序列 encryption_key，这个密钥是用于加密和解密消息的关键。

2. 密钥加密

   当需要发送消息时，首先构建消息内容，然后将消息内容转换成字节序列。接下来，使用Fernet密钥（encryption_key）创建一个Fernet加密器对象 cipher_suite

   1. 将消息内容进行序列化，通常使用JSON格式。然后将序列化后的消息字节作为输入传递给Fernet加密器对象，调用 encrypt 方法进行加密
   2. 加密后的消息成为密文，可以通过MQTT协议发送给接收者

3. 解密操作

   当接收到加密消息时，首先使用相同的密钥 encryption_key 创建一个Fernet解密器对象

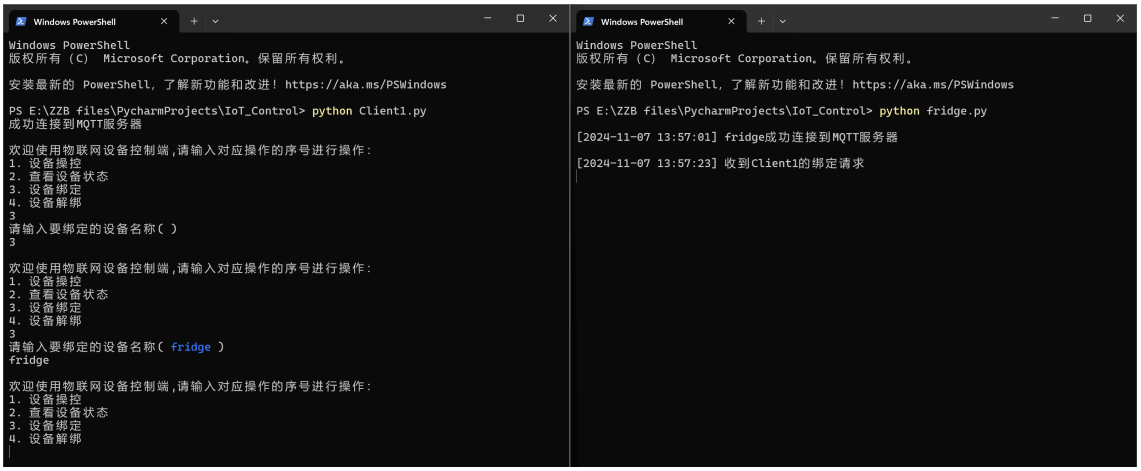
   1. 接收到的密文消息可以通过MQTT协议获取，并将其传递给Fernet解密器的 decrypt 方法，以获得原始的加密数据
   2. 解密器会使用密钥 encryption_key 对密文进行解密，将其还原为原始的序列化消息字节
   3. 将解密后的字节数据反序列化，通常使用JSON，以获得原始消息内容

4. 方案实现

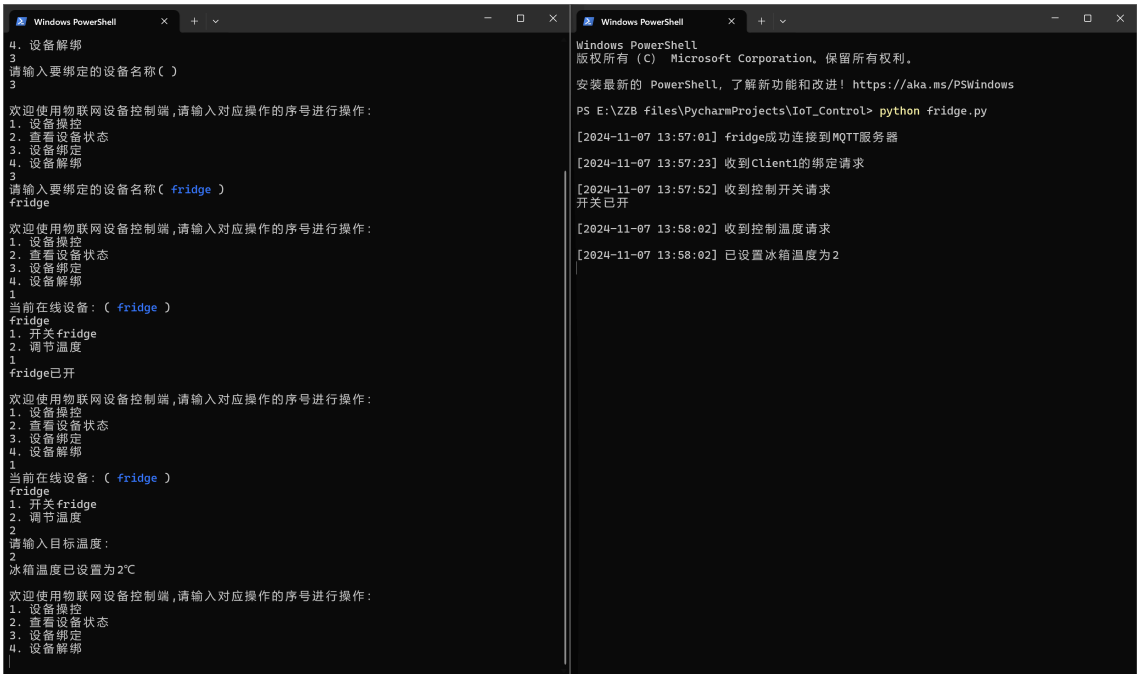
（1）控制功能测试：

以客户端Client1控制设备端fridge为例：

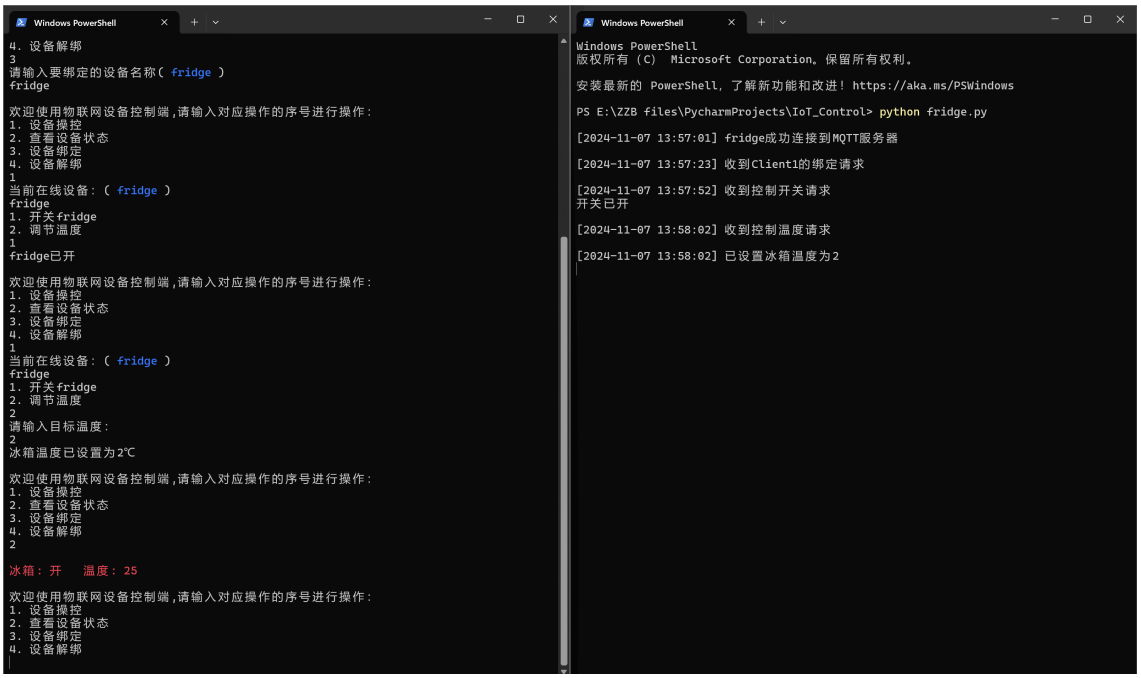
- **绑定设备端**



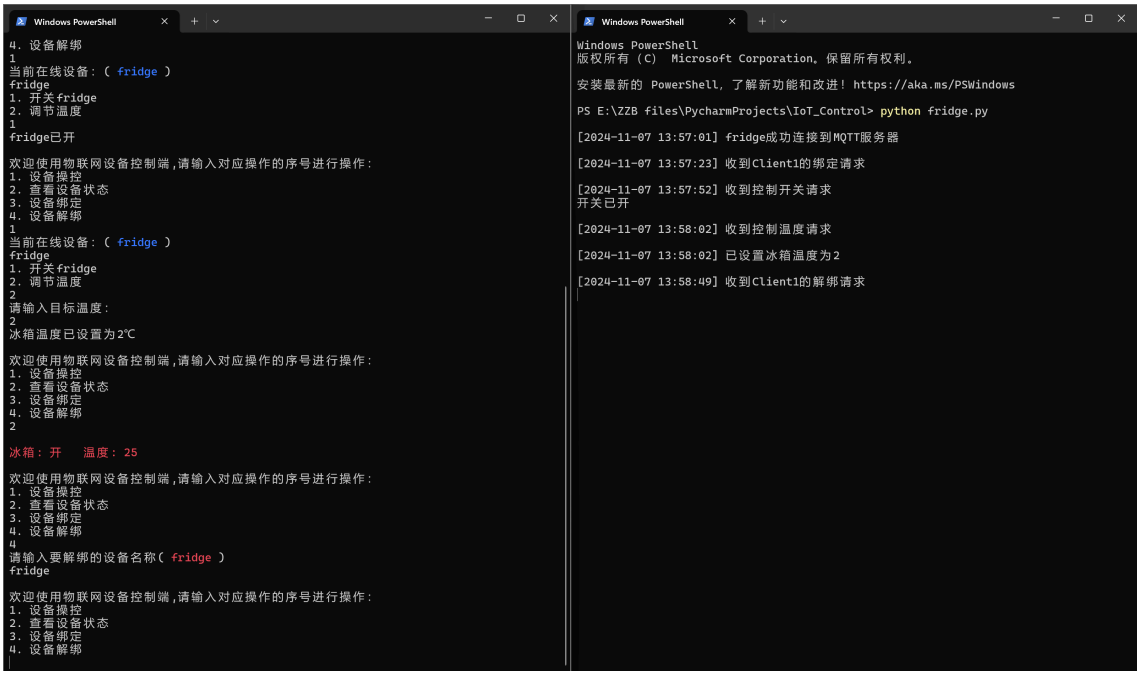
- **控制设备端**



- **查看设备端状态**



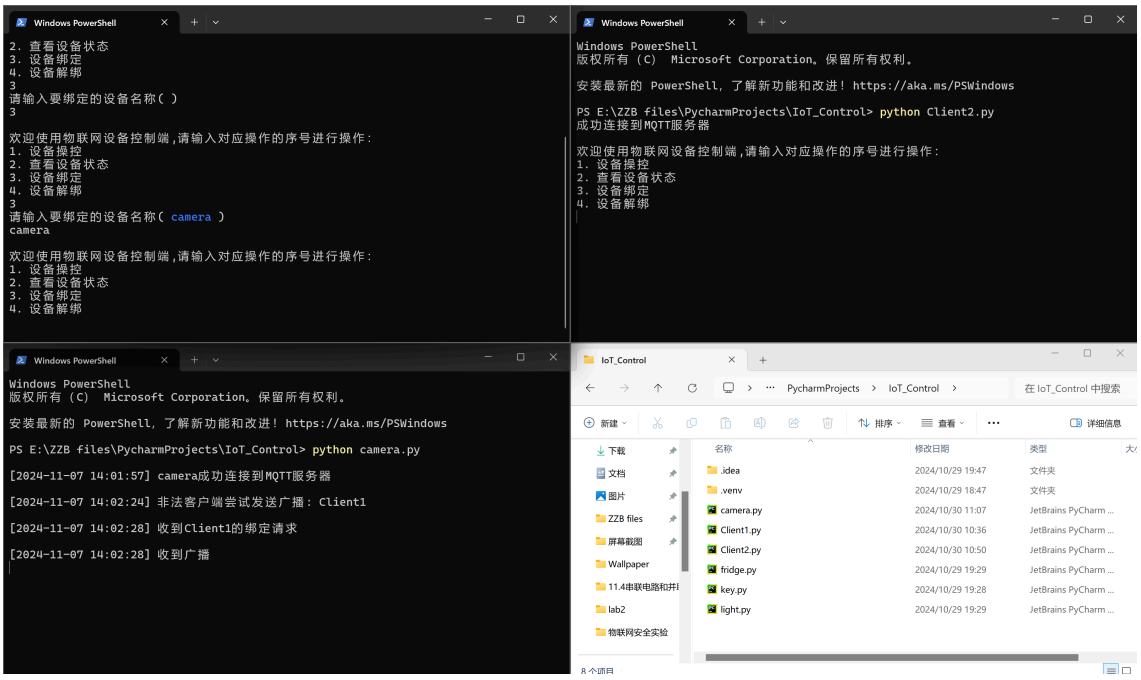
- **解绑设备**



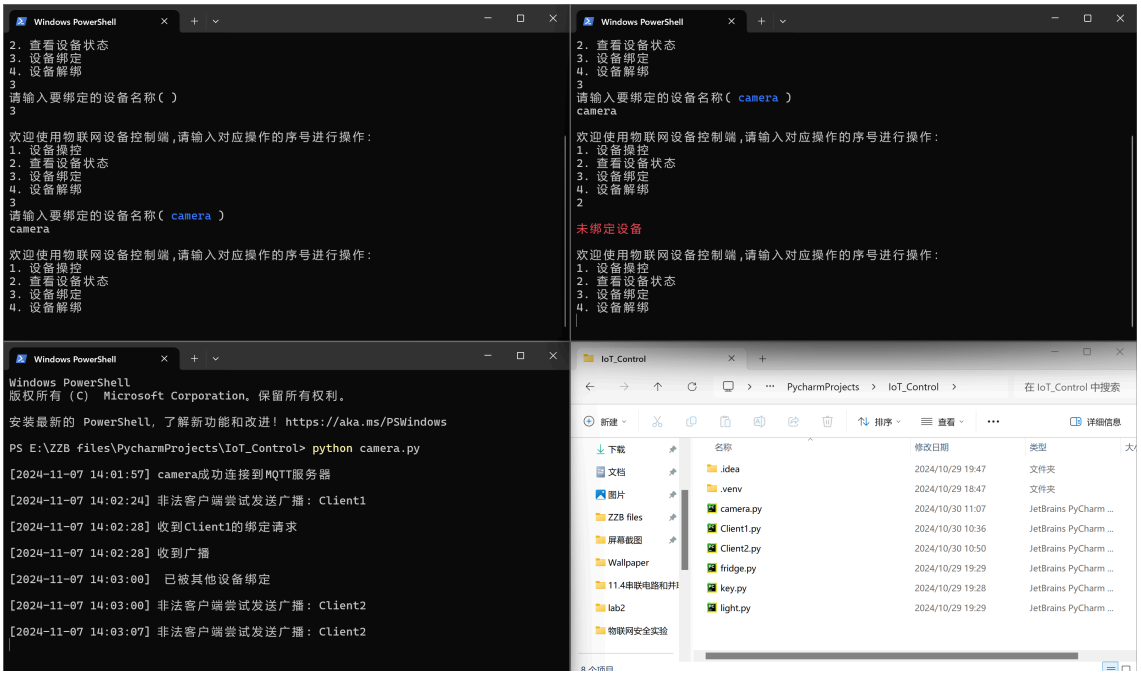
(2) 多用户功能测试

尝试用Client2绑定已经被Client1绑定过的camera设备端

- **Client1绑定camera设备端**



- **Client2尝试绑定camera设备端**



5. 系统方案安全性评估

（1）密码学加密效果不佳

当前代码中使用了 cryptography.fernet.Fernet 进行对设备控制命令的加密和解密。使用的密钥是静态的，并且直接在代码中定义。密钥硬编码在代码中是非常危险的。容易导致密钥泄露风险如果攻击者获得了代码，就能获取密钥，并解密所有消息。并且一个固定的静态加密密钥对于长期运行的系统而言并不安全，尤其是当密钥泄露时，所有通信都能被解密。

改进建议：

1. 使用动态密钥管理：密钥应该由设备或服务器动态生成，并且采用定期更换密钥的策略。可以考虑使用 对称加密 配合 公钥基础设施（PKI） 进行密钥交换。例如，在设备和服务器之间采用安全的认证和密钥交换协议（如使用 TLS/SSL 或 Diffie-Hellman）。
2. 改变密钥存储方式：密钥不应直接存储在代码中，可以考虑使用外部的密钥管理系统（如 HSM 或云平台提供的密钥管理服务）来存储密钥。

（2）指令重放攻击防御

当前代码中的加密使用了对称密钥加密，但没有添加防止指令重放的机制。指令（如控制设备开关）通过 MQTT 发布，并且加密后发送。由于没有添加任何防重放机制，攻击者可以拦截并重放有效的控制指令。这意味着即使用户发送了“开灯”指令，攻击者也可以在之后的时间再次发送“开灯”指令，导致设备无效或被恶意控制。

改进建议：

使用时间戳：在每个消息中添加时间戳，并且让服务器和客户端对时间戳进行验证，确保指令不会被重放。可以为每个指令生成唯一的随机数（Nonce），并记录已经使用过的 Nonce 防止重放。也可以为每条消息分配一个递增的序列号，接收方可以根据序列号确保消息的唯一性和顺序性。

（3）设备认证与绑定

当前实现设备通过 bind 操作绑定到用户的控制中，且每个设备的命令都可以通过加密发布到 MQTT 服务器。设备绑定的过程并未进行额外的身份验证。如果攻击者控制了某个设备，便能直接通过 bind 操作绑定设备，甚至可能解绑用户已绑定的设备。攻击者也可以伪造一个设备，并通过相同的 MQTT topic 向服务器发送消息，这样就能伪装成合法设备。

改进建议：

1. 设备认证：在设备与服务器进行绑定时，应该加入认证机制。可以使用 证书 或者 双向TLS认证 来确保设备与服务器的身份是可信的。
2. 使用数字签名：在 bind 和 unbind 操作时，客户端和设备的身份可以通过数字签名进行验证。即设备发送 bind 请求时，必须附带数字签名，服务器使用预存的公钥来验证请求。

四、 回答问题

1. 智能家居设备的使用与工业控制系统面临的风险差异

智能家居设备主要面临的风险包括网络安全风险、设备脆弱性和用户行为风险。智能家居设备通常连接到家庭Wi-Fi网络，容易受到黑客攻击，这可能导致用户的隐私信息泄露或设备被远程控制。此外，许多智能家居设备在安全性设计上存在不足，软件更新不及时，使其容易受到攻击。用户在使用时，缺乏安全意识或操作不当也可能增加设备的风险。而工业控制系统则面临更为严重的风险，包括系统安全风险、操作失误和物理安全风险。由于这些系统通常与重要的基础设施相连，遭受攻击可能会导致重大经济损失或危害公共安全。此外，工业环境中的复杂操作可能因人为失误导致设备故障，影响生产流程。物理安全风险也值得关注，因为攻击者可以通过物理接入点对系统进行干扰。

2. 常见的加密通信密码算法对比分析

在加密通信中，常见的密码算法包括AES、RSA和ECC。

1. AES（高级加密标准）是一种对称加密算法，以其快速和高安全性（支持128位、192位和256位密钥）而广泛应用，但密钥管理较复杂，需要在通信双方之间安全传输密钥。
2. RSA（Rivest-Shamir-Adleman）是一种非对称加密算法，相对而言其优点在于安全性高和密钥分发简单，适合于数字签名，但加密速度较慢，通常用于小数据量的加密。

ECC（椭圆曲线密码学），是另一种非对称加密算法，提供了相同安全性所需的密钥长度更短，性能优于RSA，然而其实现复杂性较高，并且支持较少

3. 虚拟机的NAT模式、桥接模式、Host-only模式

1. NAT模式允许虚拟机通过主机的IP地址访问外部网络，而外部网络无法直接访问虚拟机，这种模式适合那些需要访问互联网但不希望暴露虚拟机的环境。
2. 桥接模式则使虚拟机直接连接到物理网络，获得一个独立的IP地址，这样其他网络设备可以直接访问虚拟机，适合需要与物理网络中的其他设备进行交互的情况。
3. Host-only模式仅允许虚拟机与主机之间的通信，无法访问外部网络，且外部网络也无法访问虚拟机，这种模式适合测试和开发环境，便于虚拟机与主机之间频繁交互而无需外部网络的支持。

五、 收获感悟

- **方沐华：**在本次基于MQTT协议的智能家居原型系统开发过程中，我深刻认识到安全性在物联网系统中的重要性。通过对系统的设计与实现进行反思，我意识到密钥管理、身份验证、消息加密以及权限控制等方面的安全问题，不仅会影响系统的可靠性，还可能成为攻击者利用的漏洞。为了保障系统的安全性，必须采取动态密钥管理、端到端加密、防重放机制以及细粒度的权限控制。同时，日志记录与审计功能的引入也为系统的安全性提供了有效的保障。这些思考与改进建议将有助于提升智能家居系统在实际应用中的安全性与稳定性，确保用户的数据隐私和设备控制的安全。

- **秦泽斌：**通过本次实验，大大提高了我对IoT网络编程的认识，对MQTT服务器的使用也增强了我对物联网网络技术的认识，同时，对本次实验的加密处理工作也让我认识到了安全性在物联网系统中的重要作用，尤其是与老师的讨论令我受益匪浅，老师指出多个关于我们实验设计中的安全漏洞，比如越权获取设备端信息和静态密钥的不足等，并且提出了一些很好的建议，例如通过服务器进行权限控制，为每个设备端分别分配一组topic等，将权限控制的压力和资源消耗从设备端转移到服务器端等，这些都是很好的建议，引导我们进行了一些良好的尝试。