这份代码演示了一个完整的 Diffie-Hellman (D-H) 密钥交换过程，包含了正常模式、中间人攻击模式以及基于口令的防御模式。

以下是针对你提出的三个问题的详细代码解释：

---

1. 什么是 D-H，它的工作原理是什么？

Diffie-Hellman (D-H) 是一种密钥交换协议，它允许双方在不安全的通信信道上（比如被监听的网络）协商出一个共同的秘密密钥，而不需要预先传输这个密钥。

工作原理（数学逻辑）：

D-H 的安全性基于**离散对数难题**。简单来说，计算 $g^a \mod p$ 很简单，但已知结果反推 a 极难。

1. **公共参数**：通信双方（Alice 和 Bob）约定两个大数：大素数 p 和生成元 g。这些在你的 `para.py` 中定义。
2. **生成私钥与公钥**：
   - Alice 生成私钥 a，计算公钥 $A = g^a \mod p$。
   - Bob 生成私钥 b，计算公钥 $B = g^b \mod p$。
3. **交换公钥**：Alice 把 A 发给 Bob，Bob 把 B 发给 Alice。（这就是网络传输的部分）。
4. **计算共享密钥**：
   - Alice 计算：$Key = B^a \mod p$
   - Bob 计算：$Key = A^b \mod p$
   - **数学原理**：$(g^b)^a \equiv g^{ab} \equiv (g^a)^b \pmod p$。双方算出了完全一样的数字。

代码对应（Mode 1）：

- **Alice.py**:
  - 生成私钥 `a = random.randint(...)`
  - 计算公钥 `A = pow(g, a, p)` 并发送给 Bob。
  - 收到 Bob 的 `B` 后，计算 `shared_key = pow(B, a, p)`。
- **Bob.py**:
  - 生成私钥 `b`，计算公钥 `B` 并发送。
  - 收到 Alice 的 `A` 后，计算 `shared_key = pow(A, b, p)`。
- **结果**：双方打印出来的 `shared_key` 是一样的，但并未在网络上直接传输这个 key。

---

2. 我的代码用到了中间人攻击（MITM），它的攻击原理是什么？怎么实现的？

中间人攻击 (Man-in-the-Middle, MITM) 的核心在于**欺骗**。攻击者 (Attacker) 位于 Alice 和 Bob 之间，拦截所有的通信。

攻击原理：

D-H 协议本身只保证了密钥交换的**保密性**，但没有验证**身份**。

- Alice 以为她在和 Bob 交换公钥，实际上她在和 Attacker 交换。
- Bob 以为他在和 Alice 交换公钥，实际上他在和 Attacker 交换。

代码实现 (`Attacker.py` 结合 Mode 2)：

攻击者需要生成自己的一套私钥 m 和公钥 M。

1. **拦截 Alice (handle_alice 函数)**：
   - **代码行为**：Attacker 监听 9999 端口，Alice 连接进来。Alice 发送她的公钥 A。
   - **攻击动作**：

```
self.A_alice = msg['public_key']  # 保存 Alice 的公钥
# ...
response = {'public_key': self.M, ...} # 关键点：发送攻击者自己的公钥 M 给
```

```
Alice
alice_conn.send(pickle.dumps(response))
```

- **结果**：Alice 计算出的共享密钥是基于 Attacker 的公钥算出来的（$K_{Alice} = M^a \mod p$）。Alice 以为这是她和 Bob 的密钥。

2. **拦截 Bob (handle_bob 函数)**：
   - **代码行为**：Attacker 主动连接 Bob 的 8888 端口。Attacker 接收 Bob 发来的公钥 B。
   - **攻击动作**：

```
self.B_bob = msg['public_key']    # 保存 Bob 的公钥
# ...
bob_conn.send(pickle.dumps(response)) # 关键点：发送攻击者自己的公钥 M 给 Bob
```

   - **结果**：Bob 计算出的共享密钥也是基于 Attacker 的公钥算出来的（$K_{Bob} = M^b \mod p$）。

3. **解密与篡改 (calculate_keys 与 AES 部分)**：
   - Attacker 现在拥有两个共享密钥：
     - `key_alice`: 与 Alice 通信专用。
     - `key_bob`: 与 Bob 通信专用。
   - **窃听/篡改流程**：
     - Alice 发送加密消息 -> Attacker 用 `key_alice` 解密 -> **看到明文**。
     - Attacker 修改明文（代码中加了 "ATTACKER: " 前缀）。
     - Attacker 用 `key_bob` 重新加密 -> 发送给 Bob。

总结：代码通过让 Attacker 充当“双面代理”，分别与两端建立独立的 D-H 连接，从而完全控制了加密通信。

---

3. 我在代码中实现了什么防御机制？它是怎么完成防御机制的？

你在代码中实现的是**基于预共享口令 (Pre-shared Password) 的身份验证**。

代码实现 (Mode 3)：

1. **发送方 (Alice.py)**： 在 Mode 3 下，Alice 在发送公钥的同时，将口令 `PASSWORD` ("123456") 放入数据包中。

```
if choice == '3':
    data['password'] = PASSWORD
    # ... 发送 data
```

2. **接收方/验证方 (Bob.py)**： Bob 收到数据包后，不仅提取公钥，还提取口令进行比对。

```
if mode == 3:
    if received_pwd == PASSWORD:
        print("[Bob] password match successful! ...")
    else:
        print("[Bob] password does not match! Refuse to communicate")
        conn.close() # 关键防御动作：断开连接
        return
```

防御原理分析：

- **防御机制如何生效**： D-H 的弱点在于它不知道“对方是谁”。加入口令校验后，Bob 只有在确认对方持有正确的口令时，才愿意继续进行密钥计算和后续通信。如果 Attacker 试图连接 Bob 但不知道口令，连接就会被 Bob 拒绝。
- **对中间人攻击的防御手段阐述**： 要防御中间人攻击，核心必须解决**身份认证 (Authentication)** 问题。
    1. **你的代码手段（简单认证）**：通过预先共享的秘密（Password）来确认身份。
        - *注意*：在你的代码逻辑中，如果 Attacker 仅仅是转发 Alice 的包（包含正确密码），Attacker 依然可以欺骗 Bob。真正的防御通常要求密码不仅仅是被传送，而是用于**加密**或**签名**公钥（例如：HMAC 或 Encrypted Key Exchange），这样攻击者因为没有密码，就无法伪造包含自己公钥 M 的合法数据包。
    2. **标准防御手段（PKI/证书）**： 在现实世界（如 HTTPS/TLS）中，我们不直接传密码，而是使用**数字证书**。
        - Alice 收到 Bob 的公钥 B 时，还要求 Bob 提供由权威机构（CA）签名的证书。
        - 证书证明了“公钥 B 确实属于 Bob”。
        - 如果 Attacker 发送公钥 M，但他拿不出属于 Bob 的合法证书，Alice 就会终止连接。

简而言之：你的代码通过检查“口令”来确保连接发起者是值得信任的（知道秘密的人），从而在一定程度上阻止了未经认证的第三方随意建立连接。