一、API接口 AK/SK 认证练习

1.1、描述API 接口认证技术中 AK/SK 认证的原理和基本流程。

基本原理

AK:Access Key Id,用于标示用户;
SK:Secret Access Key,是用户用于加密认证字符串和用来验证认证字符串的密钥，其中SK必须保密.
通过使用Access Key Id / Secret Access Key加密的方法来**验证某个请求的发送者身份。**

云主机接收到用户的请求后，系统将使用AK对应的相同的SK和**同样的认证机制生成认证字符串**，并与用户请求中包含的认证字符串**进行比对。**如果认证字符串**相同**，系统认为用户拥有指定的操作权限，并执行相关操作；如果认证字符串**不同**，系统将忽略该操作并返回错误码。

主要流程

判断用户请求中是否包含Authorization认证字符串。如果包含认证字符串，则执行下一步操作。
基于HTTP请求信息，使用相同的算法，生成Signature字符串。
使用服务器生成的Signature字符串与用户提供的字符串进行比对，如果内容不一致，则认为认证失败，拒绝该请求；如果内容一致，则表示认证成功，系统将按照用户的请求内容进行操作。

客户端：

1. 构建http请求（包含 access key）；
   使用请求内容和 使用secret access key计算的签名(signature)；
   3. 发送请求到服务端。

服务端：

1. 根据发送的access key 查找数据库得到对应的secret-key；
   使用同样的算法将请求内容和 secret-key一起计算签名（signature），与客户端步骤2相同；
   3. 对比用户发送的签名和服务端计算的签名，两者相同则认证通过，否则失败。

1.2、使用以下请求报文计算两种签名值，并分析优劣。

```
import hashlib
import uuid

def calculate_sign1(app_id, req_id, req_time, app_secret):
    sign_str = f"appId={app_id}&reqId={req_id}&reqTime={req_time}&appSecret=
{app_secret}"
    return hashlib.md5(sign_str.encode('utf-8')).hexdigest()

def calculate_sign2(app_id, req_id, req_time, app_secret, data):
    hash1 = hashlib.md5(data.encode('utf-8')).hexdigest()
    sign_str = f"appId={app_id}&data={hash1}&reqId={req_id}&reqTime=
{req_time}&appSecret={app_secret}"
    return hashlib.md5(sign_str.encode('utf-8')).hexdigest()

# data
```

```
app_id = "20301037"
req_id = f"API02_{str(uuid.uuid4())[:8]}"  # 使用UUID4生成req_id
req_time = "1682915696123"
app_secret = hashlib.md5(app_id.encode('utf-8')).hexdigest()
data = '{"name": "贺思超", "school": "软件学院", "course": "API设计与实现"}'


# 计算签名值
sign1 = calculate_sign1(app_id, req_id, req_time, app_secret)
sign2 = calculate_sign2(app_id, req_id, req_time, app_secret, data)

# 打印结果
print(f"appId: {app_id}, \nreqId: {req_id}, \nreqTime: {req_time}, \nappSecret:
{app_secret}, \ndata: {data}")
print("1) 签名逻辑 1:", sign1)
print("2) 签名逻辑 2:", sign2)
```

输出数据:

```
appId: 20301037,
reqId: API02_699db415,
reqTime: 1682915696123,
appSecret: 2b3b2cc1f9996c081530f3dcbc4733b4,
data: {"name": "贺思超", "school": "软件学院", "course": "API设计与实现"}
1) 签名逻辑 1: ed2c83cede8326f85212d24b2d2959e9
2) 签名逻辑 2: e598206fb6d582da9f6e83cbcee99ffe
```

3）简要分析两种签名逻辑的优劣

- 签名逻辑 1相对简单，只使用了少数几个参数进行签名计算。这种简单性可能会带来一定的效率优势，尤其在请求频繁的情况下。
- 签名逻辑 2在计算签名之前对data字段进行了哈希计算，可以提高签名值的安全性。哈希计算可以防止数据被篡改，并且可以确保签名值的长度一致。然而，这种计算也增加了计算量和复杂性。

二、Token 认证-JWT 练习

2.1、简要描述 JWT 中 JWS 和 JWE 的基本原理和使用场景

基本原理

JWS(JSON Web Signature)

JSON Web Signature是一个有着简单的统一表达形式的字符串：

头部（Header）
头部用于描述关于该JWT的最基本的信息，例如其类型以及签名所用的算法等。
JSON内容要经Base64 编码生成字符串成为Header。

载荷（PayLoad）
payload的五个字段都是由JWT的标准所定义的。

1. iss: 该JWT的签发者
2. sub: 该JWT所面向的用户
3. aud: 接收该JWT的一方

4. exp(expires): 什么时候过期，这里是一个Unix时间戳
5. iat(issued at): 在什么时候签发的

后面的信息可以按需补充。
JSON内容要经Base64 编码生成字符串成为PayLoad。

签名（signature）
这个部分header与payload通过header中声明的加密方式，使用密钥secret进行加密，生成签名。、

具体生成步骤：

1. 准备 Payload 数据：将要传输的数据整理为 JSON 格式的 Payload。
2. 准备 Header 数据：构建包含算法和类型信息的 JSON 格式的 Header。通常，选择的签名算法包括 HMAC（例如 HS256）或 RSA（例如 RS256）。
3. 编码 Header 和 Payload：使用 Base64 编码将 Header 和 Payload 数据转换为字符串。将编码后的 Header 和 Payload 通过句点（.）连接起来形成一个字符串。
4. 选择和准备密钥：根据选择的签名算法，准备用于生成签名的密钥。例如，对于 HMAC 算法，密钥是一个共享的对称密钥；对于 RSA 算法，密钥是一个公钥-私钥对。
5. 计算签名：使用选定的签名算法和密钥，对编码后的 Header 和 Payload 进行签名计算。签名计算的结果是一个二进制字符串。
6. 编码签名：将计算得到的签名二进制字符串进行 Base64 编码，得到签名字符串。
7. 构建 JWS Token：将编码后的 Header、Payload 和签名通过句点（.）连接起来形成 JWS Token。最终的 JWS Token 结构为：Header.Base64_Payload.Base64_签名。
8. 返回 JWS Token：将生成的 JWS Token 返回给调用方。

JWE(JSON Web Encryption)

相对于JWS，JWE则同时保证了安全性与数据完整性。
JWE由五部分组成：

The protected header，类似于JWS的头部；
The encrypted key，用于加密密文和其他加密数据的对称密钥；
The initialization vector，初始IV值，有些加密方式需要额外的或者随机的数据；
The encrypted data (cipher text)，密文数据；
The authentication tag，由算法产生的附加数据，来防止密文被篡改。

具体生成步骤为：

1. JOSE含义与JWS头部相同。
2. 生成一个随机的Content Encryption Key （CEK）。
3. 使用RSAES-OAEP 加密算法，用公钥加密CEK，生成JWE Encrypted Key。
4. 生成JWE初始化向量。
5. 使用AES GCM加密算法对明文部分进行加密生成密文Ciphertext,算法会随之生成一个128位的认证标记Authentication Tag。
6. 对五个部分分别进行base64编码。

使用场景

JWS的主要目的是保证了数据在传输过程中不被修改，验证数据的完整性。但由于仅采用Base64对消息内容编码，因此不保证数据的不可泄露性。所以不适合用于传输敏感数据。JWS适用于身份验证、授权和信息传递等场景。

JWE的计算过程相对繁琐，不够轻量级，因此适合与数据传输而非token认证，但该协议也足够安全可靠，用简短字符串描述了传输内容，兼顾数据的安全性与完整性。所以JWE适用于保护敏感数据、安全传输和隐私保护等场景，通过数据加密确保数据的机密性和保密性。根据具体的需求和安全要求，选择适合的机制以满足数据保护和安全传输的需求。

2.2、使用以下配置计算一个 JWS Token 值

```
import hashlib
import json
import base64
import hmac

def calculate_jws_token(sub, name, school, course, iat, key):
    header = {
        "alg": "HS256",
        "typ": "JWT"
    }

    payload = {
        "sub": sub,
        "name": name,
        "school": school,
        "course": course,
        "iat": iat
    }

    # 序列化 Header 和 Payload
    encoded_header = base64.urlsafe_b64encode(json.dumps(header).encode('utf-
8')).decode('utf-8')
    encoded_payload = base64.urlsafe_b64encode(json.dumps(payload).encode('utf-
8')).decode('utf-8')

    # 构建待签名的数据
    data_to_sign = f"{encoded_header}.{encoded_payload}"

    # 使用密钥进行签名
    signature = hmac.new(key.encode('utf-8'), data_to_sign.encode('utf-8'),
hashlib.sha256).digest()
    encoded_signature = base64.urlsafe_b64encode(signature).decode('utf-8')

    # 构建完整的 JWS Token
    jws_token = f"{data_to_sign}.{encoded_signature}"

    return jws_token

# 示例数据
sub = "20301037"
name = "贺思超"
school = "软件学院"
course = "API设计与实现"
iat = 1516239022
key = hashlib.md5(sub.encode('utf-8')).hexdigest()

# 计算 JWS Token
```

```
jws_token = calculate_jws_token(sub, name, school, course, iat, key)

# 打印结果
print("JWS Token:", jws_token)
```

JWS Token:

```
eyJhbGciOiAiSFMyNTYiLCAidHlwIjogIkpXVCJ9.eyJzdWIiOiAiMjAzMDEwMzciLCAibmFtZSI6ICJ
cdThkM2FcdTYwMWRcdThkODUiLCAic2Nob29sIjogIlx1OGY2Zlx1NGVmNlx1NWI2Nlx1OTY2MiIsICJ
jb3Vyc2UiOiAiQVBJXHU4YmJlXHU4YmExXHUOZTBlXHU1YjllXHU3M2IwIiwgImlhdCI6IDE1MTYyMzk
wMjJ9.ha38CNwjnt9Pw4VpTZROyuiXEV2s46tKzcoWizeGgpY=
```