作者： Zephyr369

Lab3 混合加密

问题一 实现混合加密

在 Lab2 中已经实现了CCA安全的图片内容加密方案，而且封装为一个类，这次只需要对类进行继承，然后在新的混合加密类里面添加上RSA的公钥与私钥生成，以及对对称加密秘钥的加解密即可

```
def __init__(self, *args, **kwargs):
        # 先顺手调用一下CCA的构造函数吧CCA初始化了
        super().__init__(*args, **kwargs)
        self._ensure_rsa_keys()
```

然后需要确保目录存在

```
 def _ensure_rsa_keys(self):
        os.makedirs(os.path.dirname(self.rsa_key_path), exist_ok=True)
        # 首先读取存不存在RSA的公钥或者私钥，如果存在，读取，否则，生成
        if not os.path.exists(f"{self.rsa_key_path}_private.pem") or not os.path.exists(f'{self
            self._generate_and_save_rsa_keys()
```

这里面需要对Crypto库进行说明

> 在cryptography库中，default_backend()函数的作用是提供一个默认的加密后端。加密后端（backend）是执行加密算法运算的底层实现，比如对密钥进行生成、数据的加密解密、散列计算等操作。cryptography库设计为后端无关的方式，意味着它可以使用不同的库来实际执行加密操作，而这些库就是所谓的后端。使用default_backend()函数，我们可以不必关心具体的加密算法是如何实现的，或者它背后使用了什么库。这个函数会根据你的环境自动选择最合适的后端。例如，在不同的操作系统上，它可能会选择不同的库作为后端来实现相同的加密功能。

```
def _generate_and_save_rsa_keys(self):
        private_key = rsa.generate_private_key(
            public_exponent=65537,
            key_size = 2048,
            backend = default_backend()
        )
        public_key = private_key.public_key()

        # 保存私钥和公钥
        with open(f"{self.rsa_key_path}_private.pem", 'w') as file:
            file.write(
                private_key.private_bytes(
                    encoding = serialization.Encoding.PEM, # 指定输出格式为PEM
                    format = serialization.PrivateFormat.PKCS8, # 指定私钥的格式为PKCS#8
                     encryption_algorithm = serialization.NoEncryption()
                ).decode('utf-8')
            )
        with open(f'{self.rsa_key_path}_public.pem', 'w') as file:
            file.write(
                public_key.public_bytes(
                    encoding = serialization.Encoding.PEM,
                    format = serialization.PublicFormat.SubjectPublicKeyInfo
                ).decode('utf-8')
            )
```

然后，为了保证加密和解密的接口不变，我们需要重写父类的加解密接口，首先利用公钥对对称密钥进行加密，然后利用对称密钥加密图片，将加密好的对称密钥传给Bob，Bob利用自己的私钥先对加密的对称密钥解密得到解密的对称密钥，然后利用解密的对称密钥解密图片即可

共因子攻击Rsa

欧拉定理

欧拉定理是数论中的一个重要定理，与费马小定理密切相关，它提供了一种计算模幂运算的有效方法。欧拉定理指出，若 nn 是一个正整数，aa 是任意与 nn 互质的整数，则 aa 的欧拉函数 $\varphi(n)φ(n)$ 次幂除以 nn 的余数为1。数学表达式为：

$a^{\phi(n)} \equiv 1 \pmod{n}$ $\phi(n)$是欧拉函数，表示小于或等于n的正整数中与n互质的数的数量。

RSA 加密原理

1. 选择两个大素数 p 和 q ，计算$n = pq$。
2. 计算欧拉函数$\phi(n) = (p-1)(q-1)$。
3. 选择一个公钥指数 e : e 与$\phi(n)$互质，一般65537
4. 计算私钥指数 d :$ed \equiv 1 \pmod{\phi(n)}$
5. 公钥是(e, n)，私钥是(d, n)

相同因子攻击

本实验中，两个公钥(n_1, e_1) 与(n_2, e_2)，中的模有相同因子，因此我们可以用GCD求出相同因子p,本实验的对称密钥是用公钥1加密的，因此我们拿到公钥1的其中一个大素数 p ,那我们就可以很轻易的利用$\frac{n_1}{p}$来求出另一个大素数q，然后就可以按照RSA的要求来构建公钥一对应的私钥。

解密对称密钥

根据题目中的加密方案

> ① 对称加密方法为密钥长度128位的AES-CBC，对明文采用PKCS #7填充，128位IV放在密文开头。 ② 对称加密的明文为RGBA四通道图像中的所有像素，为使密文图片尺寸合法，对密文进行了填充，以四字节（一个像素）为单位，与PKCS #7类似，即如果密文图像最后一个像素转换为四字节整数的值为k（大端序），说明密文图像的后k个像素是padding。由此填充方法产生的密文图片比明文图片多一行像素。 ③ 对称加密产生的密文图片为enc1.png ④ 128位对称密钥先进行Base64编码，再使用公钥1加密，加密方法为RSA-OAEP，密文的Base64编码在下面给出。

我们首先利用pillow库拿到密文的像素字节 ```python # 使用Pillow读取图像 img = Image.open(path).convert("RGBA") img_data = np.array(img)

```
# 将图像数据转换为字节序列
img_bytes = img_data.tobytes()
```

然后利用slice方法提取IV

```python
# 提取IV（前16个字节）
    iv = img_bytes[:16]
```

然后根据加密方案描述，我们需要获取最后一个像素，将他按照大端序解析出来填充了多少个像素，然后将这些像素去掉，掐头去尾之后，中间的就是需要加密的内容。

```
# 解析自定义填充
    # 最后一个像素（4字节）大端序表示填充长度（以像素为单位）
    padding_indicator = img_bytes[-4:]
    padding_length_pixels = int.from_bytes(padding_indicator, "big")

    # 计算去除自定义填充后的密文长度
    # 每个像素RGBA占用4字节
    encrypted_content_length = len(img_bytes) - 16 - padding_length_pixels * 4
    encrypted_content = img_bytes[16:16 + encrypted_content_length]
```

然后利用AES CBC解密，然后对明文去填充

```
def decrypt_image(encrypted_image, symmetric_key, iv):
    cipher = AES.new(symmetric_key, AES.MODE_CBC, iv)
    decrypted_data = cipher.decrypt(encrypted_image)
    # # 使用unpad去除PKCS#7填充
    decrypted_data_no_pkcs7_padding = unpad(decrypted_data, AES.block_size, style='pkcs7')
    return decrypted_data_no_pkcs7_padding
```

最后查看图片属性获得图片大小（1920*1080），然后保存文件

```
def save_decrypted_image(dec_data_no_padding, output_path, image_size):
    image = Image.frombytes("RGBA", image_size, dec_data_no_padding)
    image.save(output_path)
    # 查看一下图片信息 1920*1080
width = 1920
height = 1080
image_size = (width, height)
save_decrypted_image(decrypted_data, r"lab3\attacks/dec.png", image_size)
```