桂林电子科技大学
GUILIN UNIVERSITY OF ELECTRONIC TECHNOLOGY

《应用密码学实验》
综合项目

题　　目：	**文件安全传输**	
学　　院：	计算机与信息安全学院	
组　　长：	2100300931	韦华升
组　　员：	2100300930	宋佳辉
	2100301011	邓凯文
	2100301010	邓晋直

2023 年 12 月 12 日

小组分工

（注：按照贡献大小写分工顺序，越靠前贡献越大）

学号：2100301011 姓名：邓凯文
个人在团队中的主要工作： 设计程序 UI 界面，参与报告写作，编写 TCP 传输文件，连接程序界面和代码。
学号：2100301010　　姓名：邓晋直
个人在团队中的主要工作： 设计程序 UI 界面，参与报告写作，编写加密明文，加密密钥和数字签名的代码。
学号：2100300930　　姓名：宋佳辉
个人在团队中的主要工作： 参与报告写作，设计程序基本流程图，编写对称密钥解密，验证数字签名代码。
学号：2100300931　　姓名：韦华升
个人在团队中的主要工作： 参与报告写作，设计程序 UI 界面，生成密钥对，生成文件摘要和对称密钥代码。

成绩评定标准及成绩

1、 能够按照格式要求、排版要求和字数要求等，有需求分析，系统设计与实现，系统测试和参考文献。（15 分）_____

2、 报告内容行文通畅，有条理性，无错别字，无抄袭等。（15 分）

3、 在验收过程中，能合理的回答问题（**30 分**）_____

4、 软件运行正常，实现所提出的功能（30 分）_____

5、 代码规范、具有自己的创新或特色（10 分）_____

总成绩：_____

摘要

本系统为基于网络的文件安全传输工具，实现了发送方和接收方对文件的加密发送和解密接收，其主要功能包括用户对文件的加密发送和接收解密，文件以数字信封进行封装，特点是用户可以验证文件是否被篡改。

系统实现了通过建立 socket 连接来实现发送方和接收方连接和传输数据的功能。通过 RSA 算法生成密钥对；通过 Fernet 算法生成随机对称密钥并加密明文；利用哈希算法中的 SHA-256 算法生成文件摘要参与消息验证的过程。

关键字：数字签名，数字信封，加密技术，安全传输。

目录

1. 绪论 1
   1.1 需求分析 1
   1.2 开发环境 1
2. 设计与实现 2
   2.1 概要设计 2
   2.2 详细设计与实现 2
      2.2.1 程序简要流程图 2
      2.2.2 导入所需模块和库 3
      2.2.3 RSA 算法生成密钥对 3
      2.2.4 Fernet 算法生成随机对称密钥并加密明文 4
      2.2.5 SHA-256 生成文件摘要 4
      2.2.6 数字签名 4
      2.2.7 Fernet 算法解密对称密钥和密文 5
      2.2.8 数字签名验证 5
      2.2.9 TCP 传输数据 6
3. 系统测试 9
4. 总结 14
   4.1 团队总结 14
   4.2 个人总结 17
   4.3 系统特点 17
   4.4 本项目设计实现中对社会法律影响方面的理解认知和应用实践 17
5. 参考文献 18

1. 绪论

当前网络信息安全主要是采用各种信息保密手段以及防护措施，为网络和信息系统提供保护，通过信息加密、网络防护以及安全管理等使信息更加安全。智能化手段在安全领域得到越来越广泛应用，但是目前仍然面临一些挑战与威胁。

在计算机网络日益扩大和普及的今天，计算机对安全的要求更高、涉及面更广。信息安全即数据安全，是指系统有能力抵抗外来非法入侵者对信息的恶意访问、泄露、修改和破坏等,即:机密性、完整性、可用性。

所以，如何实现计算机网络中数据传输安全，近年来一直是人们研究的课题之一。迄今为止，对网络和数据传输安全的最重要的自动工具是加密。数据在网络上传输时，其安全威胁主要来自于非法窃听，因此可将数据经加密算法加密成密文，然后在将密文发送到网络上进行传输，这是一种十分有效的安全保密手段。

1.1 需求分析

数字信封是一种用于传输文件的加密和安全技术，通过数字信封，文件可以在网络上进行安全传输，以防止未经授权的访问和窃取。这种技术在当今数字化时代的信息传输和数据隐私方面具有越来越重要的作用。以下是数字信封传输文件市场需求的一些分析因素：

1.安全性需求：

随着网络犯罪和数据泄露的不断增加，企业和个人对文件传输的安全性要求越来越高。数字信封提供了加密和认证等安全功能，满足了用户对数据保护的需求。

2.隐私保护：

法规和法律对个人数据和敏感信息的保护要求越来越严格。数字信封帮助确保在文件传输过程中的隐私保护，符合法规要求，降低了法律风险。

3.行业合规性：

一些行业，如医疗、金融和法律，对文件传输的合规性有着严格的规定。数字信封的使用可以帮助企业遵守相关行业法规和标准，保证数据传输的规范性。

4.便捷性和效率：

数字信封技术通常提供了便捷的文件传输方式，减少了传统方式中繁琐的步骤。这提高了工作效率，尤其对需要频繁传输大量文件的行业或组织而言更为重要。

5.实时性：

一些业务场景对实时文件传输有着更高的要求，数字信封的使用可以提供更快速的传输速度，以满足用户对实时性的期望。

6.成本效益：

数字信封的使用可能减少了物理传输文件的成本，如快递费用、打印成本等。在长期内，这对企业来说可能是一种经济效益。

7.技术创新：

技术的不断创新推动了数字信封的发展，新的功能和改进使得文件传输更加安全、方便和智能化，进一步推动市场需求。

1.2 开发环境

(1)Win10/win11

(2)PyCharm/vs code/python 3.10

(3)PyQt5 库：用于创建 GUI 界面。
(4) Qt Designer 工具:设计用户界面并将其转换为 Python 代码。
(5) socket 库：提供套接字编程的功能，用于网络通信。
(6) oc 库：提供与操作系统交互的功能，用于处理文件路径等操作。

2. 设计与实现

2.1 概要设计

代码实现了基于 RSA 算法、RSA-PSS 数字签名算法、Fernet 算法和 SHA-256 算法的文件传输系统，包括文件加密传输、签名验证两个功能。具体设计如下：

1.主要导入的库和模块：os、socket、sys 、Crypto.PublicKey.RSA、zipfile、cryptography.hazmat.backends、tcp_client、tcp_server 和 PyQt5。

2.创建 MainWindow 类，继承 QMainWindow 和 Ui_MainWindow 类，实现文件加密传输、签名验证等功能。其中，文件加密传输通过加密算法加密文件，再将加密后的对称密钥和签名值封装发送给接收方；签名验证由接收方通过发送方的公钥验证数字签名的有效性，判断文件传输过程有没有别篡改。

3.程序作为主程序运行时，则创建 QApplication 实例和 MainWindow 实例，并显示 GUI 界面。

2.2 详细设计与实现

2.2.1 程序简要流程图

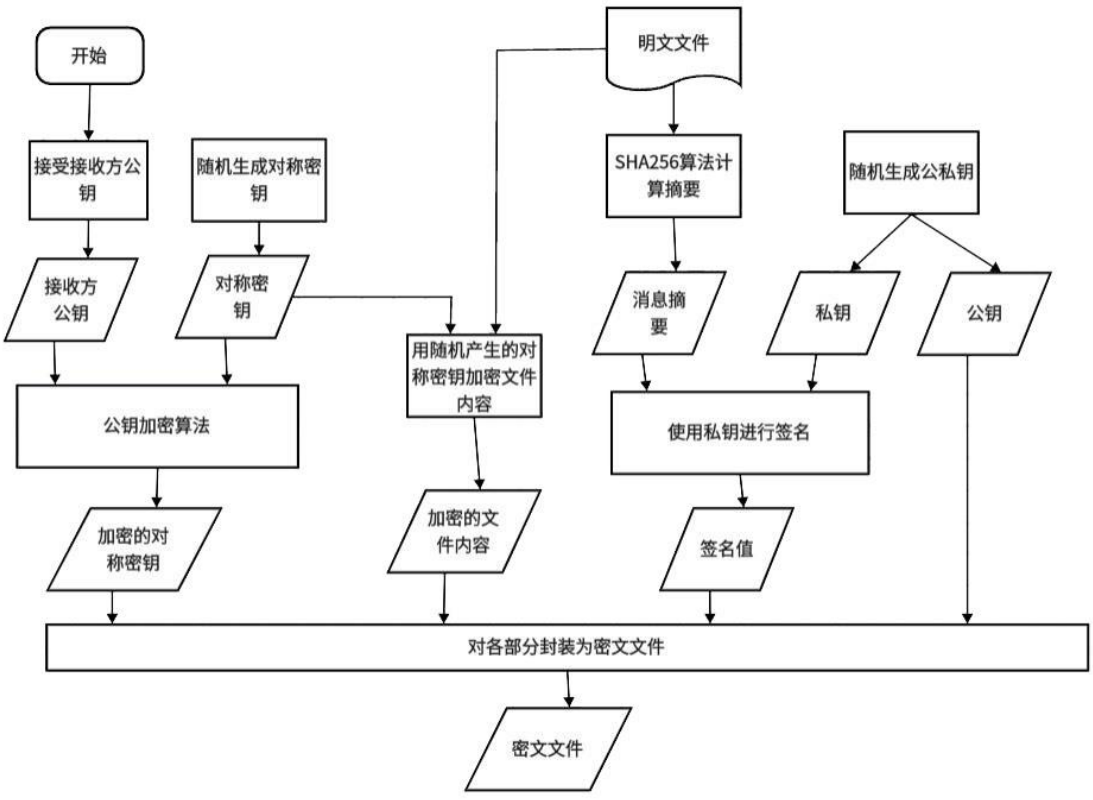

图 2-1 文件封装加密过程

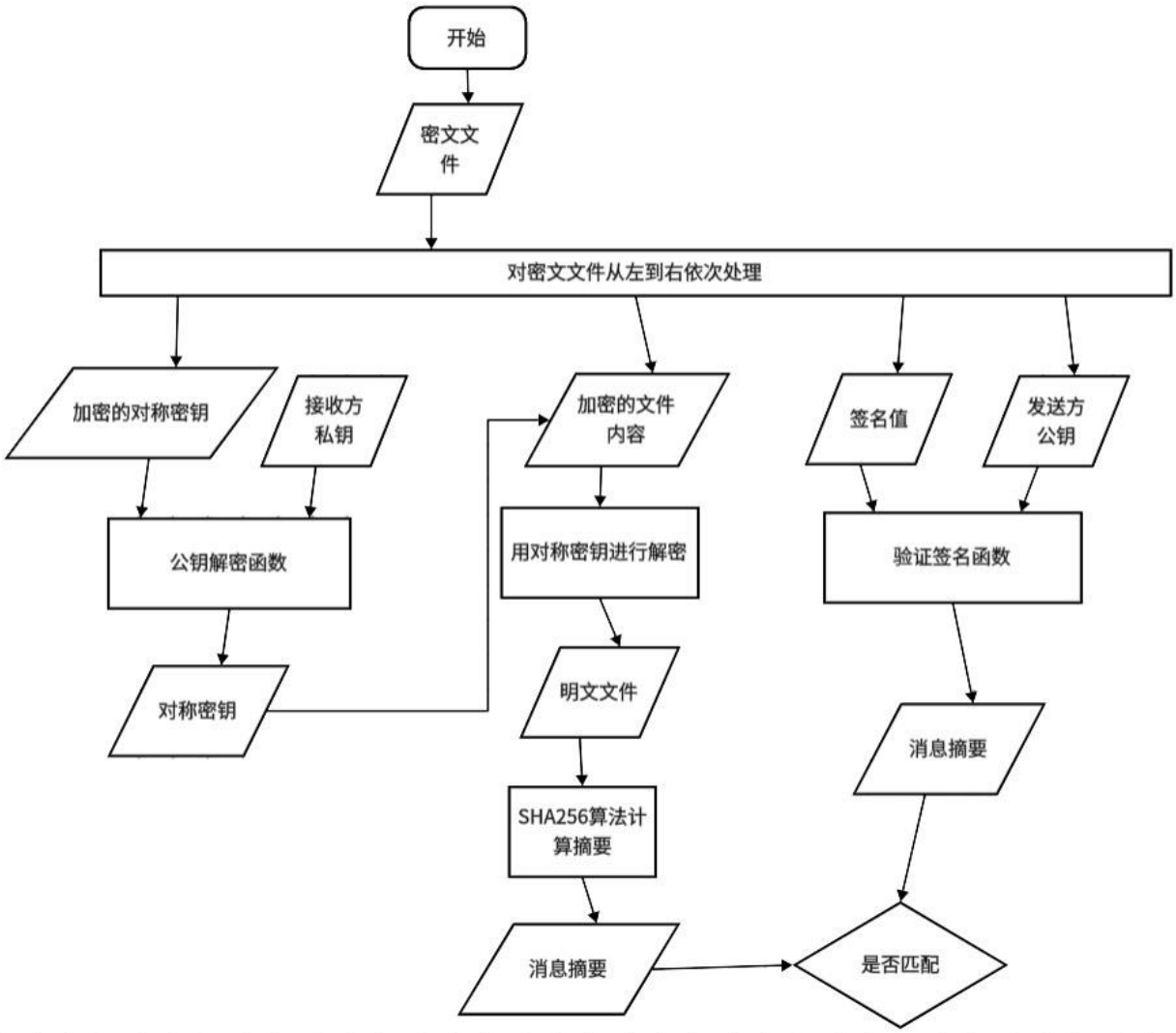

图 2-2 文件解封解密过程

2.2.2 导入所需模块和库

1.os：提供与操作系统交互的功能，用于处理文件路径等操作。
2.cryptography：版本 41.0.7， 用于加解密文本和密钥，进行数字签名和验签。
3.pycryptodome：版本 3.19.0，提供 RSA 算法的实现，用于生成密钥对。
4.PyQt5：版本号 5.15.9，用于创建图形用户界面 GUI。
5.socket：提供网络通信功能，实现 TCP 连接。
6.zipfile：用于创建和解压 zip 文件的库。
7.Ui_untitled：包含 GUI 设计的用户界面类，使用 Qt Designer 工具生成。

2.2.3 RSA 算法生成密钥对

A_private_public.py 和 B_private_public.py 两个项目是发送方和接收方各自通过 RSA 算法生成 RSA 密钥对。

导入 Python 中的 os 模块,用于处理文件路径。从 PyCryptodome 库中导入 RSA 模块，用于生成 RSA 密钥对。

以 A（发送方）为例，生成一个 2048 位的 RSA 密钥对再获取生成的私钥和公钥。

通过 OS 模块获取当前文件的目录路径并构建公私钥文件的相对路径。以二进制写入模式打开公私钥文件，最后将公钥写入公钥文件，将私钥写入私钥文件。

2.2.4 Fernet 算法生成随机对称密钥并加密明文

使用 Fernet 用于生成一个随机的对称密钥。Fernet 是 cryptography 库中提供的对称加密工具，它使用 AES 算法进行加密。生成的密钥是一个二进制字符串，通常是 32 字节（256 位）长度。使用 with open(...)语句打开指定路径的文件，使用二进制写入模式('wb')。这样可以确保以二进制格式写入密钥，最后将生成的对称密钥写入打开的文件。

以二进制读取模式('rb')打开指定路径的文件，读取文件的内容，即明文。用给定的对称密钥 key 进行初始化 Fernet 对象，并且使用 Fernet 对象的 encrypt 方法将明文加密为密文。最后将加密后的密文写入打开的文件，生成一个加密文件。

2.2.5 SHA-256 生成文件摘要

使用 cryptography 库中的 hashes 模块来创建一个 SHA-256 哈希对象，将文件内容逐步更新到该对象中，然后通过 finalize 方法获取最终的摘要值。这种方式能够有效地处理大型文件，因为它允许分块更新消息，而不是一次性加载整个文件。

2.2.6 数字签名

打开 A 的私钥文件并读取私钥数据，使用 serialization.load_pem_private_key 方法加载 PEM 格式的私钥数据，得到私钥对象。接着打开指定文件路径读取原文。

```
# 计算摘要
hash_object = hashes.Hash(hashes.SHA256(), backend=default_backend())
hash_object.update(message)
hash_value = hash_object.finalize()

# 使用私钥进行签名
signature = private_key.sign(
    hash_value,
    padding.PSS(
        mgf=padding.MGF1(hashes.SHA256()),
        salt_length=padding.PSS.MAX_LENGTH
    ),
    utils.Prehashed(hashes.SHA256())
)
```

图 2-3 数字签名

创建 SHA-256 散列算法对象 hashes.Hash(hashes.SHA256(), backend=default_backend())。更新散列对象的内容，将原文数据添加到散列中。计算摘要值，得到 SHA-256 摘要。

使用 A 的私钥对摘要值进行数字签名。private_key.sign 方法接受摘要值、填充方案（PSS）、MGF1 哈希算法（SHA-256）、预期哈希算法（SHA-256）等参数，返回数字签名结果。

最后打开签名文件（signature.txt），以二进制写入模式将数字签名保存到文件中。

2.2.7 Fernet 算法解密对称密钥和密文

```
# 使用B的私钥解密密文获得对称密钥
symmetric_key = private_key.decrypt(
    encrypted_symmetric_key,
    padding.OAEP(
        mgf=padding.MGF1(algorithm=hashes.SHA256()),
        algorithm=hashes.SHA256(),
        label=None
    )
)
```

图 2-4 解密对称密钥

采用了 RSA 非对称加密算法中的 OAEP 填充方案，利用 B 的私钥对密文进行解密操作，最终得到原始的对称密钥。

```
def decrypt_file(encrypted_file_path, key):
    with open(encrypted_file_path, 'rb') as encrypted_file:
        ciphertext = encrypted_file.read()

    cipher_suite = Fernet(key)
    plaintext = cipher_suite.decrypt(ciphertext)

    decrypted_file_path = encrypted_file_path.replace('.encrypted', '_decrypted.txt')
```

图 2-5 解密文

从文件中加载对称密钥，读取加密后的密文，创建一个 Fernet 对象，用给定的对称密钥 key 进行初始化。使用 Fernet 对象的 decrypt 方法，将密文解密为明文。plaintext 包含了解密后的二进制数据。

接着构造解密后文件的路径，将加密文件路径中的.encrypted 替换为_decrypted.txt。最后将解密后的明文写入打开的文件，生成一个解密后的文件。

2.2.8 数字签名验证

数字签名验证通过计算消息的摘要并使用数字签名进行验证，确保在消息传输过程中没有被第三方篡改。用于确保通信安全性。

首先获取当前脚本文件所在的目录路径。从文件'A_public.txt'读取发送方的公钥。接着使用 serialization.load_pem_public_key 函数将读取到的公钥数据反序列化为公钥对象。读取解密后的文件'file.txt_decrypted.txt'。

```
# 计算摘要
hash_object = hashes.Hash(hashes.SHA256(), backend=default_backend())
hash_object.update(decrypted_message)
hash_value = hash_object.finalize()
# 读取数字签名
signature_path = os.path.join(current_directory, 'signature.txt')
with open(signature_path, 'rb') as file:
    signature = file.read()
# 使用公钥验证签名
try:
    public_key.verify(
        signature,
        hash_value,
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        utils.Prehashed(hashes.SHA256())
    )
    print("数字签名验证成功！")
except Exception as e:
    print(f"数字签名验证失败：{e}")
```

图 2-6 验证签名

使用 SHA-256 has 算法计算解密后的消息的摘要。从文件'signature.txt'中读取发送方对消息的数字签名。使用发送方的公钥对收到的数字签名进行验证。验证两个摘要是否一致，如果验证成功，说明消息的完整性得到保证，打印 "数字签名验证成功！"；否则，捕获异常并打印相应的失败消息。

2.2.9 TCP 传输数据

文件发送方：

```
# 创建一个 ZIP 文件
with zipfile.ZipFile(output_zip_path, 'w') as zip_file:
    # 将文件.txt添加到ZIP文件中
    zip_file.write(file_path, arcname='file.txt.encrypted')

    # 将A的公钥添加到ZIP文件中
    zip_file.write(Apub_key_path, arcname='A_public.txt')

    # 将加密后的对称密钥添加到ZIP文件中
    zip_file.write(encrypted_key_path, arcname='encrypted_symmetric_key.bin')

    # 将数字签名添加到ZIP文件中
    zip_file.write(signature_path, arcname='signature.txt')
```

图 2-7 压缩文件

将所需文件压缩：

获取相对路径后，使用相对路径构建其他文件（密文、发送方的公钥、加密后对称密钥、数字签名）的完整路径。然后创建一个 ZIP 文件对象，准备写入操作。在写入时，

向 ZIP 文件中逐个添加未加密的文件，并指定文件在 ZIP 中的名称，保存文件路径为 output_zip_path。

```
# 客户端连接服务器
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
    client_socket.connect((HOST, PORT))

    # 读取要传输的文件
    with open(output_zip_path, 'rb') as file:
        file_data = file.read()

    # 发送文件数据
    client_socket.sendall(file_data)
    print("文件发送完成")
```

图 2-8 发送加密包

发送加密包：

建立 socket 对象，根据给定的 ip 地址和端口连接到指定的服务器地址和端口。然后打开指定 zip，使用 client_socket.sendall 方法将文件数据通过 socket 发送给另一方。

```
def receive_Bpubkey(HOST,PORT):
    # 创建一个socket对象
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
        # 绑定IP和端口
        server_socket.bind((HOST, PORT))
        # 开始监听传入的连接
        server_socket.listen()

        print(f"等待客户端连接...")
        conn, addr = server_socket.accept()  # 接受客户端的连接
        print(f"已连接：{addr}")

        with conn:
            # 从客户端接收数据
            file_data = conn.recv(1024)  # 假设每次接收的数据大小为1024字节
            with open('B_public.txt', 'wb') as file:
                while file_data:
                    file.write(file_data)
                    file_data = conn.recv(1024)
            print("文件接收完成")
```

图 2-9 接收公钥

接收 B 的公钥：

首先创建一个 socket 对象，使用给定的 ip 地址和端口进行监听，等待另一方发送数据，当对方发送数据后，按 1024 字节写入 B_public.txt 文件保存。

文件接收方：

```
# 文件路径
Bpubkey_path = os.path.join(current_directory, 'B_public.txt')

# 客户端连接服务器
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
    client_socket.connect((HOST, PORT))

    # 读取要传输的文件
    with open(Bpubkey_path, 'rb') as file:
        file_data = file.read()

    # 发送文件数据
    client_socket.sendall(file_data)
    print("文件发送完成")
```

图 2-10 发送公钥

发送公钥给另一方：

首先 os.path.abspath(__file__)方法获取当前文件的绝对路径，再从绝对路径中提取目录路径。然后建立 socket 对象，根据给定的 ip 地址和端口连接到指定的服务器地址和端口。

打开指定文件，准备读取数据。使用 client_socket.sendall 方法将文件数据通过 socket 发送给服务器（另一方），文件发送完毕输出‘完成’，表明文件数据已经成功发送到指定的服务器。

```
# 创建一个socket对象
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
    # 绑定IP和端口
    server_socket.bind((HOST, PORT))
    # 开始监听传入的连接
    server_socket.listen()

    print(f"等待客户端连接...")
    conn, addr = server_socket.accept()  # 接受客户端的连接
    print(f"已连接：{addr}")
```

图 2-11 等待接收加密包

等待接收加密包：

首先使用 os.path.abspath(__file__)方法获取当前文件的绝对路径，并通过该绝对路径来获取当前文件夹路径。再使用 socket 模块创建了一个 TCP 类型的 Socket 对象，并绑定指定的 IP 地址和端口号，建立一个监听方便通信的另一方连接到此端口。

开始监听传入的连接请求，并等待客户端发起连接，一旦有客户端连接，返回一个连接对象 conn 和客户端地址 addr。

```
    with conn:
        # 从客户端接收数据
        encrypted_data_path = os.path.join(current_directory, 'encrypted_data.zip')
        with open(encrypted_data_path, 'wb') as file:
            while True:
                file_data = conn.recv(1024)
                if not file_data:
                    break
                file.write(file_data)

        print("文件接收完成")

# 解压缩 ZIP 文件
output_folder = os.path.join(current_directory)
os.makedirs(output_folder, exist_ok=True)
```

图 2-12 接收加密包并解压

接收加密包并解压：

使用 with conn 语句开启连接，并开始接收对方发送的数据，将对方发送的数据命名为 encrypted_data.zip。循环调用 recv(1024)以接收客户端发送的数据，每次接收 1024 字节数据。

以只读模式打开刚刚接收到的文件，并用 zipfile 模块将压缩包内容解压到指定的目标文件夹 output_folder 中。

3. 系统测试

演示系统截图：

图 3-1 发送方界面

图 3-2 接收方界面

功能测试：

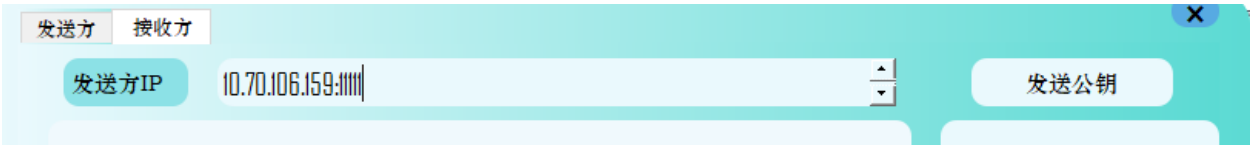

图 3-3 填写 ip 和端口

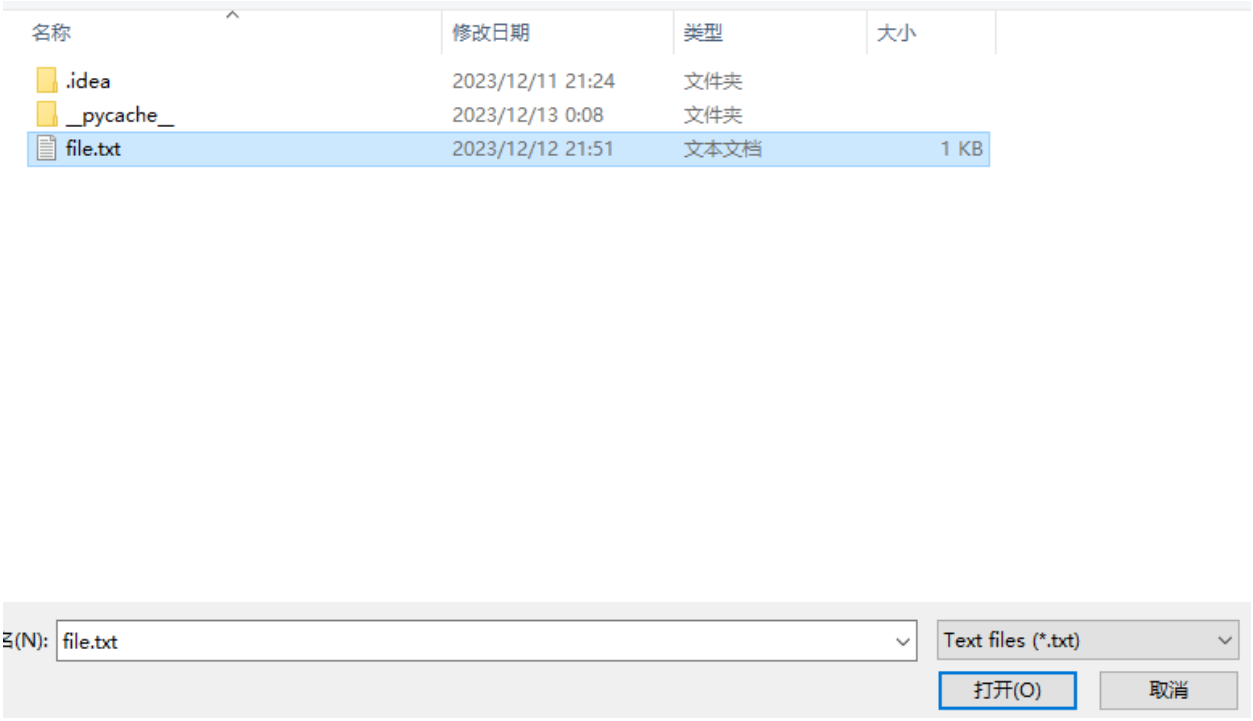

图 3-4 发送方选择所发送文件

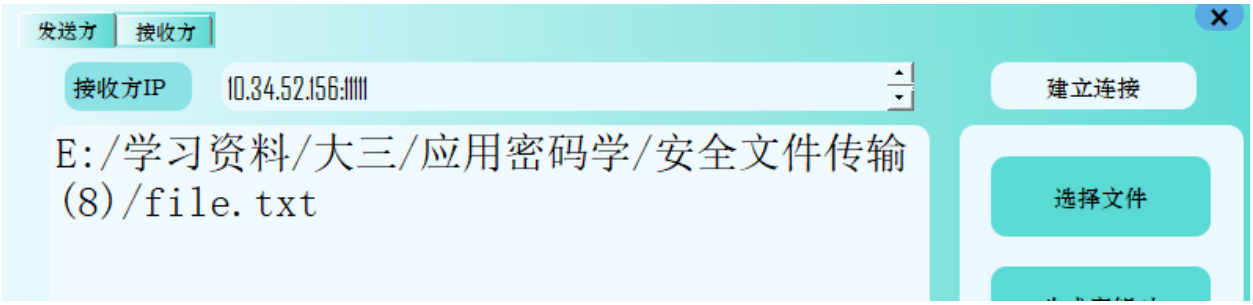

图 3-5 正常显示文件路径

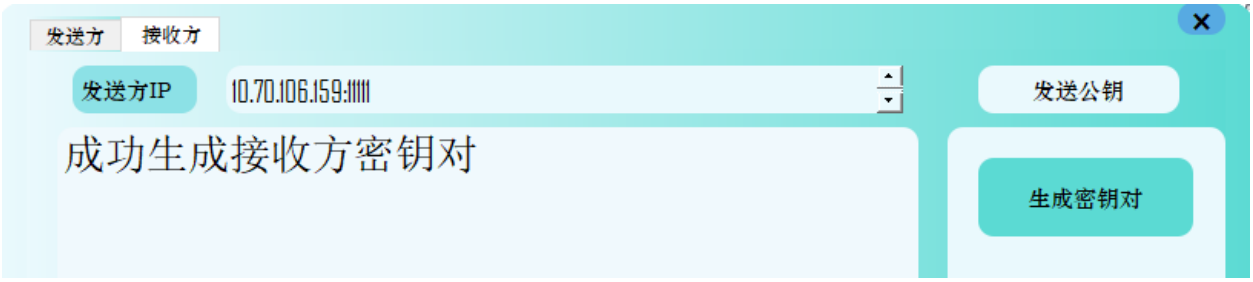

图 3-6 接收方生成密钥对

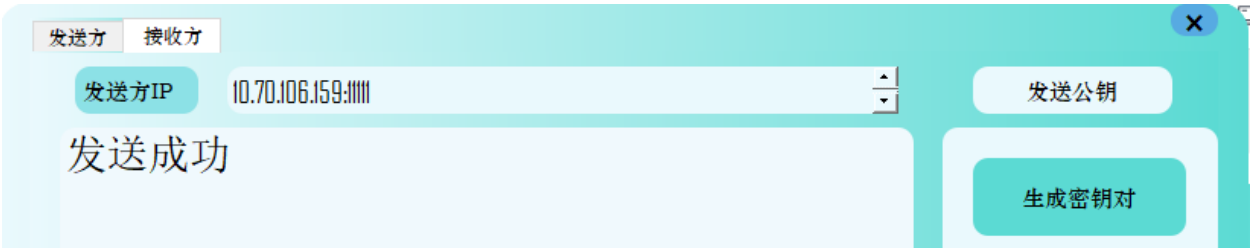

图 3-7 接收方发送公钥给发送方

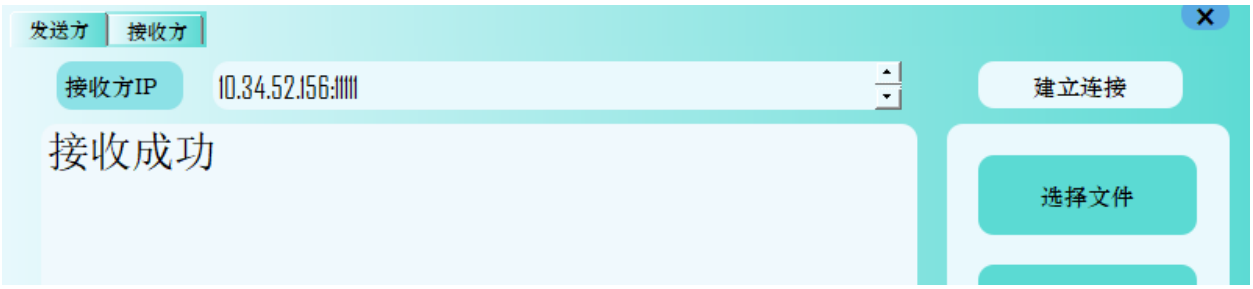

图 3-8 发送方接收成功

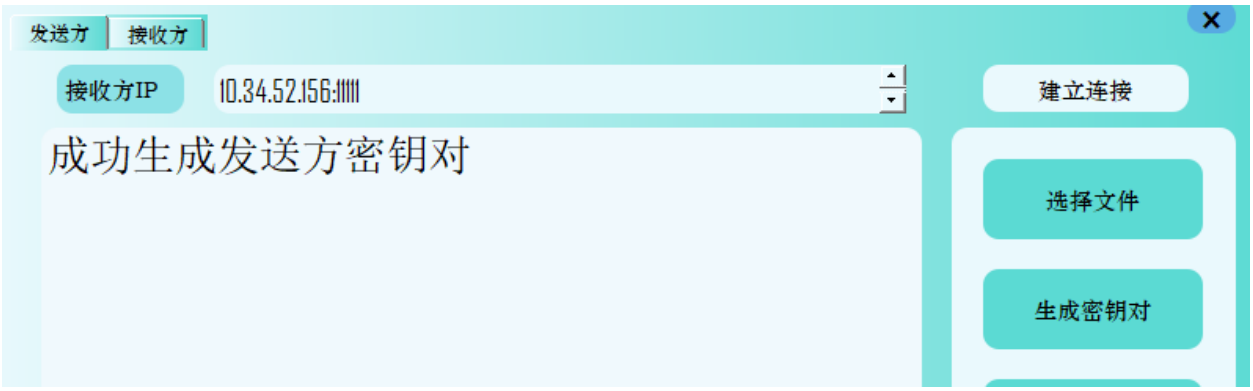

图 3-9 发送方输出密钥对

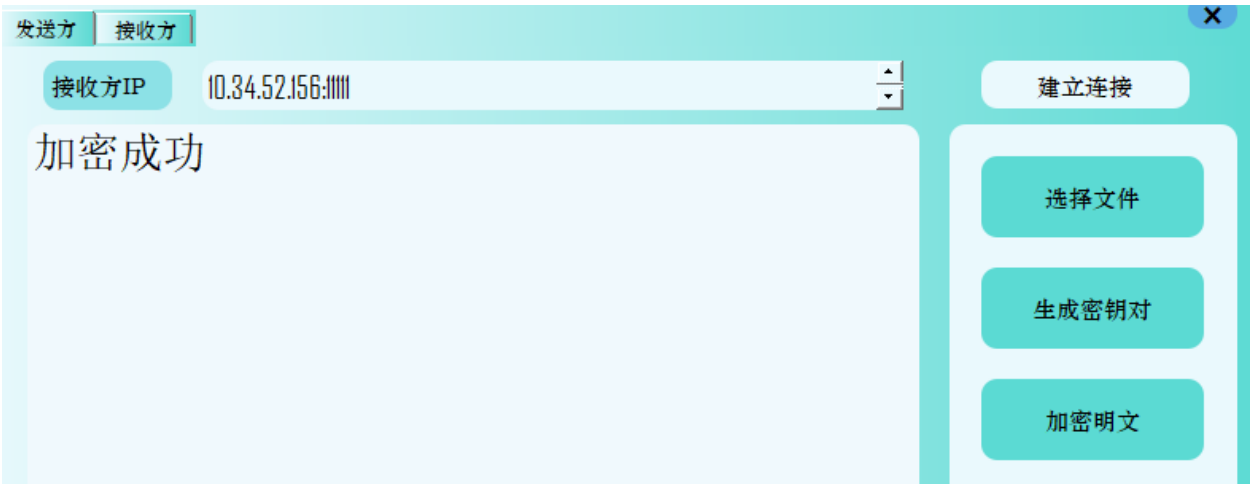

图 3-10 发送方加密密文

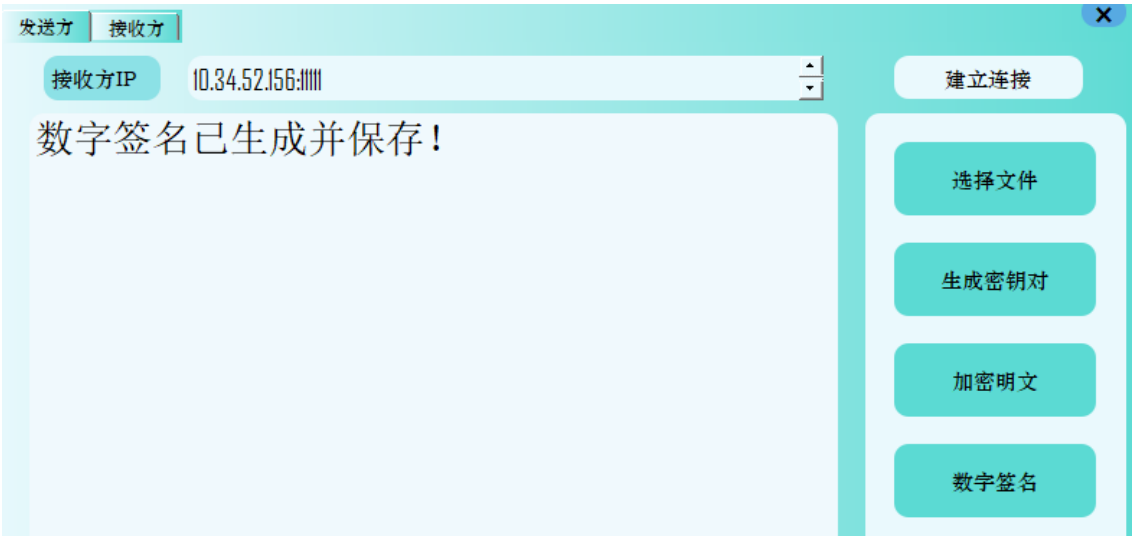

图 3-11 发送方进行数字签名

图 3-12 发送方加密对称密钥

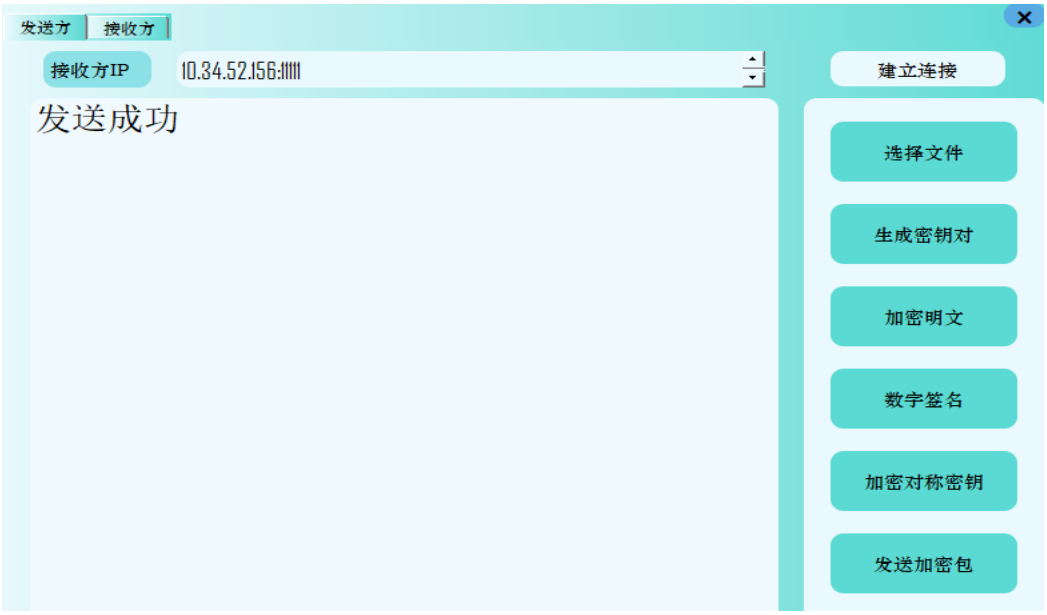

图 3-13 发送加密包给接收方

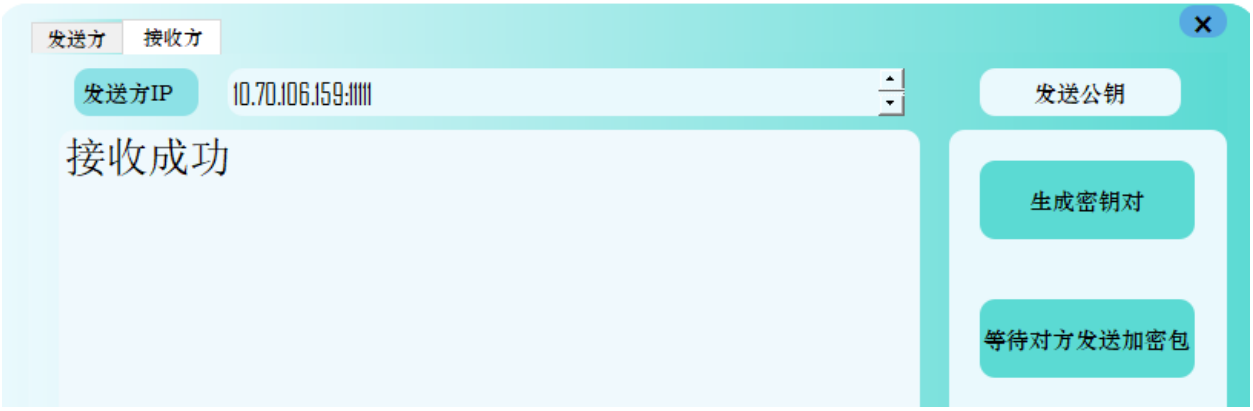

图 3-14 接收方接收成功

```
PS C:\Windows\System32\WindowsPowerShell\v1.0> & "D:/Program Files/Python310/python.exe" "d:/[数据]/空白/Desktop/安全文件传输 (3)/main.py"
文件发送完成
等待客户端连接...
已连接: ('10.70.106.159', 57542)
文件接收完成
解压缩完成到目标文件夹: d:\[数据]\空白\Desktop\安全文件传输 (3)
```

图 3-15 成功解压缩保存

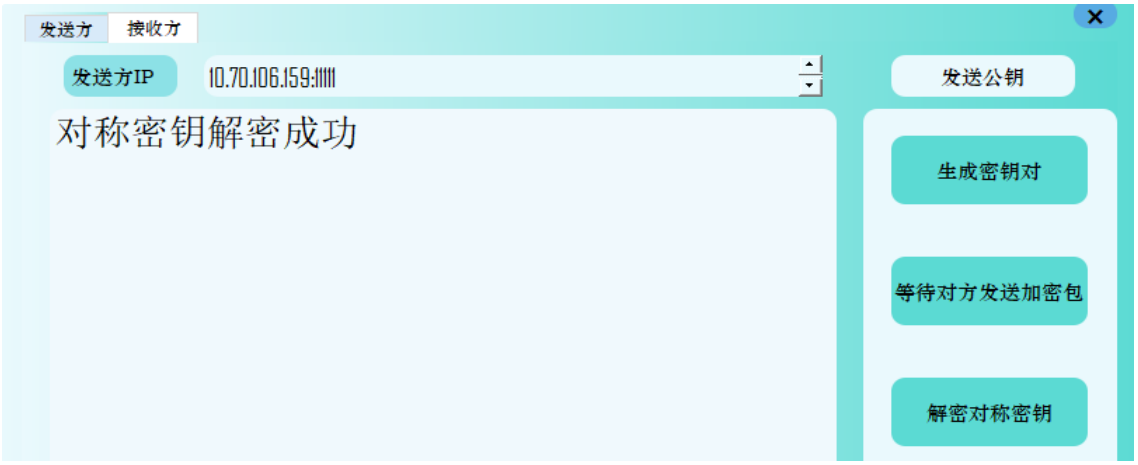

图 3-16 接收方解密对称密钥

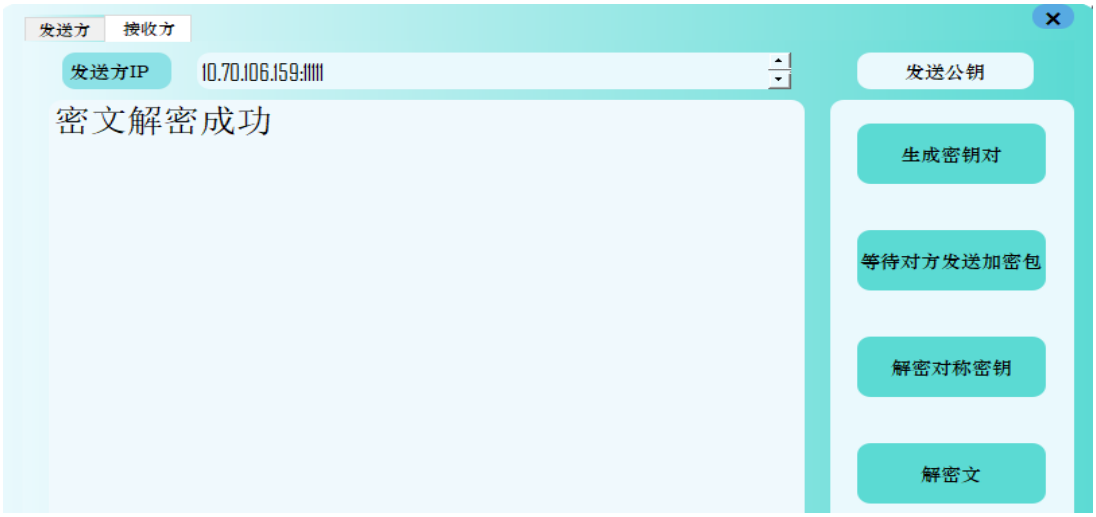

图 3-17 接收方加密密文

图 3-18 验证数字签名

容错测试：

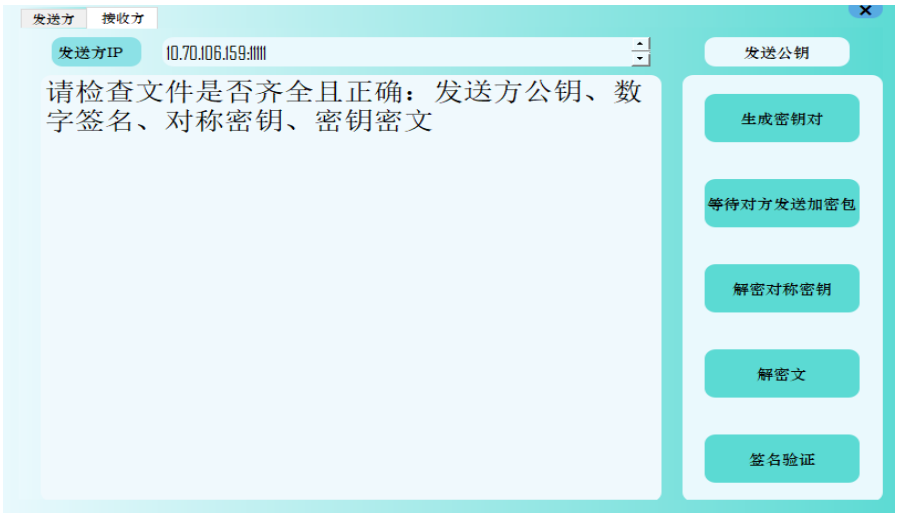

图 3-19 更改对方公钥后再验证签名

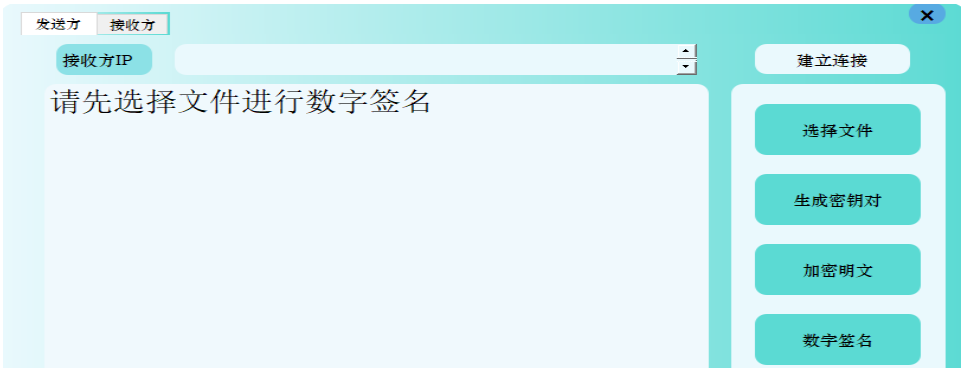

图 3-20 未选择文件进行签名

图 3-21 未连接发送加密包

4. 总结

4.1 团队总结

问题：

问题 1：加密和解密没有使用同一种算法导致无法解密。

问题 2：验证数字签名过程，误以为是用对方公钥解签名值得到摘要，实际是用 RAS 公钥来验证签名值和摘要。

问题 3：没有实现真正意义的文件传输方式，发送方和接收方只能通过 ip 连接，方式不够快捷和简便。

问题 4：代码和界面的交互性不够强，不能提供很好的显示效果。

问题 5：没能在用户界面实现类似于控制台的输出，且没有系统的日志记录。

进度：

11 月 28 日完成核心加解密代码

12 月 3 日完成 UI 界面设计
12 月 5 日完成 UI 界面和程序的对接
12 月 10 日完成报告

团队收获：

1. 安全传输理解： 在项目中，我们深入理解了数字信封的概念以及它在文件安全传输中的应用。学会了如何使用数字信封技术来加密文件，确保文件在传输过程中的安全性。

2. 加密算法应用： 通过项目，我们成功应用了 RSA 算法、RSA-PSS 数字签名算法、Fernet 算法和 SHA-256 算法。这使我们更熟练地掌握了这些加密算法的原理和实际应用。

3. 文件加密传输： 我们实现了文件加密传输的功能，这涉及到对文件进行加密、数字签名的生成和验证，以及密钥的安全传输。这为我们提供了一个实际应用场景，加深了对文件传输安全的认识。

4. 网络通信和套接字编程： 通过实现 TCP 客户端和服务器功能，我们学到了如何使用套接字进行网络通信。这对我们理解网络安全和数据传输过程中的通信机制提供了实际经验。

5. 团队协作和总结： 在项目中，我们通过团队协作共同完成了任务。通过团队总结，我们更好地理解了项目的整体流程、设计和实现。个人总结也帮助我们发现自身在安全传输方面的成长和不足之处。

6. 法律和合规性认知： 在项目总结中，我们提到了对社会法律影响方面的理解认知和应用实践。这表明我们对数字信封系统在法规和合规性方面的考虑有了更深入的了解。

7. 技术创新意识： 我们对数字信封技术的不断创新有了更深的认识。了解到技术的不断创新推动着数字信封的发展，这促使我们对未来数字安全领域的关注和思考。

交流截图：

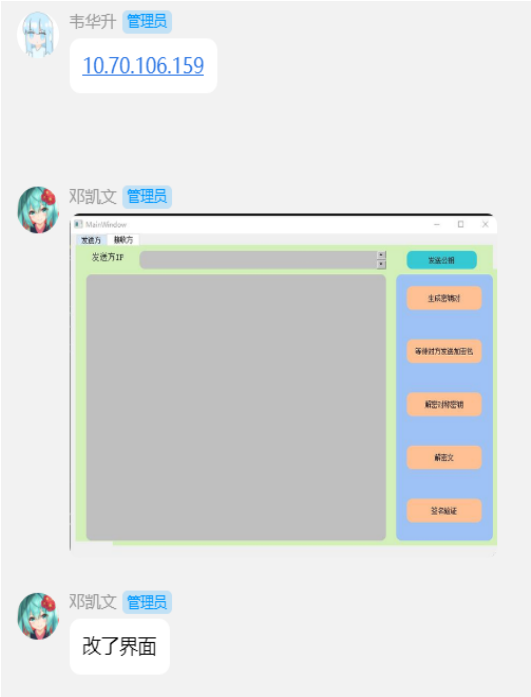

图 4-1 聊天记录 1

图 4-2 聊天记录 2

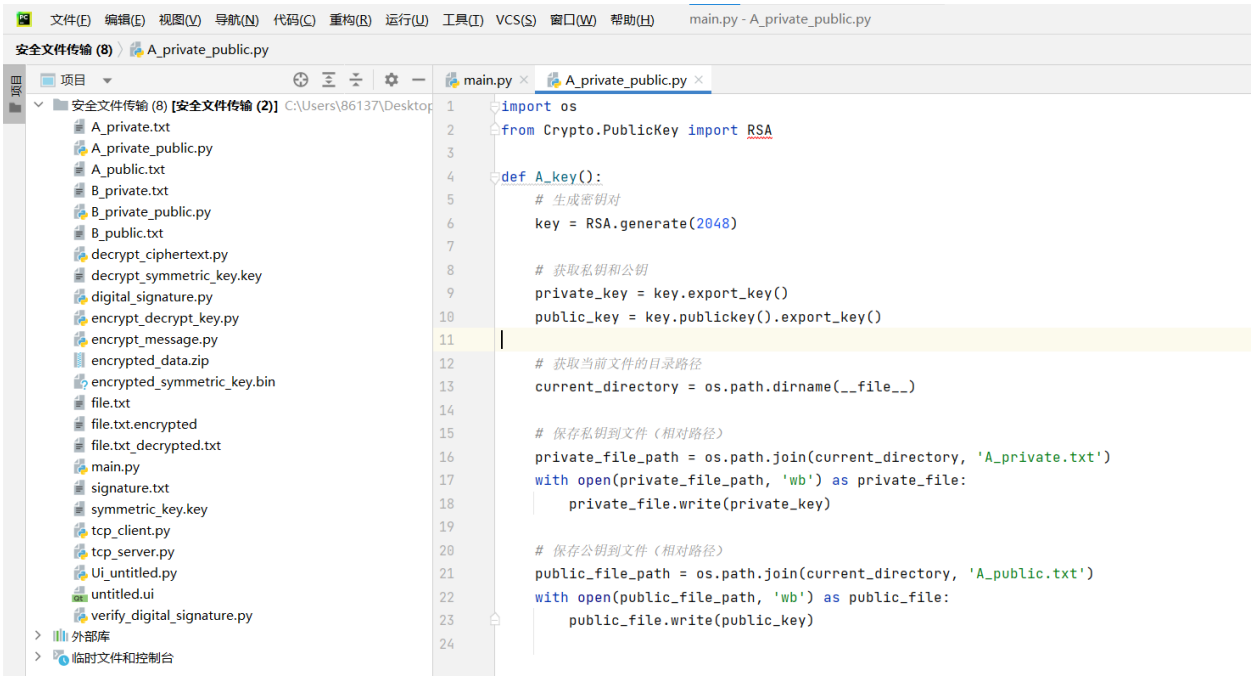

图 4-3 开发平台

4.2 个人总结

2100301010 邓凯文：在本次项目实践中，我主要负责设计两个使用者的文件传输和结合代码界面，通过建立网络连接，熟悉了 TCP 连接的基本原理和流程。在结合代码和界面时，遇到了许多不懂的问题，通过不断地学习和查阅资料带大多得到了解决。在团队协作过程中，我认识到沟通和合作的重要性，以及如何更好地利用集体智慧和力量解决问题。在未来的学习和工作中，我将继续努力扩展自己的知识和技能，并用所学为社会做出更多的贡献。

2100300930 宋佳辉：在本次项目中，我负责设计程序 UI 界面，参与报告写作，设计程序流程图等内容。通过本次小组合作，我锻炼了设计程序流程图的能力，加深了对 RSA 等加密算法的理解和使用。在解决遇到的问题中，我认识到了团队协作的总要性，通过互相交流沟通能够有效提高我们各自工作的效率。

2100300931 韦华升：在本次小组项目中，我主要负责了设计程序 UI 界面以及参与报告写作等内容。这次小组合作中，我学习了 Qt designer 制作 UI 的知识，并结合本次任务：文件安全传输中接收方与发送方的具体操作步骤，使用 Qt designer 参与完成了项目 UI 的构建与制作。此外，我对文件传输的加解密过程也有了一定理解认识，这次合作使我收获颇丰，不仅使我对专业知识有了一定认识，还锻炼了我的团队协作能力与沟通分析能力，我认为这是小组成员之间一次成功的合作。

2100301010 邓晋直：在本次项目中，我负责编写主要的代码和参与报告的撰写。通过本次小组合作，我了解了数字信封对于文件传输过程起到的保护作用。通过使用对方公钥加密对称密钥，用对称密钥加密明文和使用 SHA-256 生成摘要签名。最后解对称密钥，解密文，验证签名。一步步的解析和理解，使我对文件加密有了一定的认识。对 RSA 算法生成公钥对、Fernet 算法生成对称密钥、SHA-256 算法生成摘要、RSA-PSS 数字签名算法进行了深入学习，初步掌握了这几种加密算法。

4.3 系统特点

(1)本系统基于 RSA 算法、RSA-PSS 数字签名算法、Fernet 算法和 SHA-256 算法，可以实现文件安全传输和对数字签名进行验证。
(2)系统 UI 友好，用户操作简便。
(3)系统采用了 socket 模块创建网络连接，使用了 PyQt5 库实现了 UI 界面。

4.4 本项目设计实现中对社会法律影响方面的理解认知和应用实践

1.隐私保护和合规性：理解和遵循相关的隐私法规，例如通用数据保护法规、个人信息保护法等，确保软件设计符合法律要求。
2.合法的数据传输和存储：数字信封中的数据传输和存储符合国家和地区的相关法规。
3.知识产权保护：程序的设计不侵犯他人的知识产权，包括专利、商标和著作权等。
4.安全性：程序只提供文件加密传输功能，不会主动联网，不会对数据造成破坏、泄露。

5. 参考文献

[1]侯艳玲, 谢兴昶, 洪之坤, 刘晓夫. 数字经济时代网络信息安全基本现状与发展趋势[J]. 山东工业技术, 2023, (03): 60-64.

[2]肖志恒. 文件传输安全技术研究[D].大连工业大学,2015.

[3]赵延博, 张学杰, 姜永玲. 基于数字信封的高强度文件加密的应用研究[J]. 计算机工程与设计, 2007, (18): 4357-4359.

[4]PyQt5 快速入门-https://www.bilibili.com/video/BV1LT4y1e72X,2023.11.29