

cryptophy-fina1-job

学号: 2111033

姓名: 艾明旭

年级: 2021 级

专业: 信息安全

- - 1. 大作业内容说明
 - 1.1. 实验目的
 - 1.2. 实验要求
 - 1.3. 假设条件
 - 1.4 实验环境
- - 2. 数据库系统的建立
- - 3. 保密通信协议
 - 3.1. 保密通信协议设计思路
 - 3.2. 保密通信协议设计的流程
 - 3.3. 相关重点概念
 - 3.4. 通讯的重要代码分析
- - 4. RSA 相关
 - 4.1. 相关概念介绍
 - 4.2. 重要函数分析
- - 5. AES 相关
 - 5.1. 流程分析
 - 5.2. 代码实现
 - 5.3. CBC 模式的补充
- - 6. MD5 相关
 - 6.1. 概念介绍
 - 6.2. 重要函数分析
 - 6.3. 代码实现
- - 7. 结果展示
 - 7.1. 开始的发送公钥和共享 AES 密钥

- 7.2. 收发消息

- 8. 总结与展望

- 8.1. 总结

- 8.2. 展望

1. 大作业内容说明

1.1. 实验目的

设计一个公钥证书管理机构的服务器程序。具体要求为：

- 1、为申请用户进行注册并进行身份的核对：主要检验内容为用户身份证和手机号码的真实性。
- 2、以 RSA 公钥算法为基础，为每位注册用户建立公钥证书；
- 3、在服务器中为每位用户存储用户的基本信息，包括姓名，身份证号，手机号码以及对应的公钥证书等；
- 4、支持证书的更新，撤销等基本功能；
- 5、采用服务器/客户端模式，服务器应该设计一些简单安全的协议，可以用来处理客户端的不同请求，比如用户注册、证书申请、证书更新、证书撤销，用户注销等等。

1.2. 大作业完成情况说明

本次实验要求在数据库系统下实现对公钥证书的插入删除更新（申请，更改，撤销）等操作，在数据库系统下我完成了证书以及加密解密程序的设计，包括数字签名的编写也用 MD5 哈希函数成功实现了。但是我们还需要实现许多功能，包括从数据库当中提取公钥和私钥进行加解密以及用户的数字签名进行验证等工作。这其中不得不涉及到大整数类型在数据库系统当中的存储问题，由于公钥密码系统当中我存储了 RSA 公钥，私钥，以及生成大素数的 p, q 的值，因为数据库系统对于大整数的存储仅限于 11 位，数据库系统对于大整数也只有 `bigint` 类型可以适当的扩充到 40 位。然而，不仅我的编译环境下无法实现 40 位长整数的存储，40 位长整数也无法满足 rsa 和 aes 当中动辄成百上千位，大于 128 位数的成功存储。在实验中，只能暂时利用字符型数据将其存储，并解码参与运算。

其中还包括许多 16 进制，bit 位数字的转换问题，虽然全部得到了解决，但是用户在感官上如果只能查看到这些超长字符串，解密与生成数字签名，验证数据来源可靠性等工作还需要用户手动解码进行的话，对于用户以及产品本身的可使用程度是一种很大的影响。

于是我学习了相关的计算机网络方面的知识，学会了基础的服务器编写，并查找了一部分资料成功编写了一个服务器，由于 C++ 作为我在学期中各次实验的工具，并且可以利用头文件和大量的库完成超长类型整数的存储功能，并且实现了数据库系统实现不了的功能，弥补了数据库系统在编写过程的问题。将二者共同作为我本次的大作业，还希望老师批评指正。

1.3. 假设条件

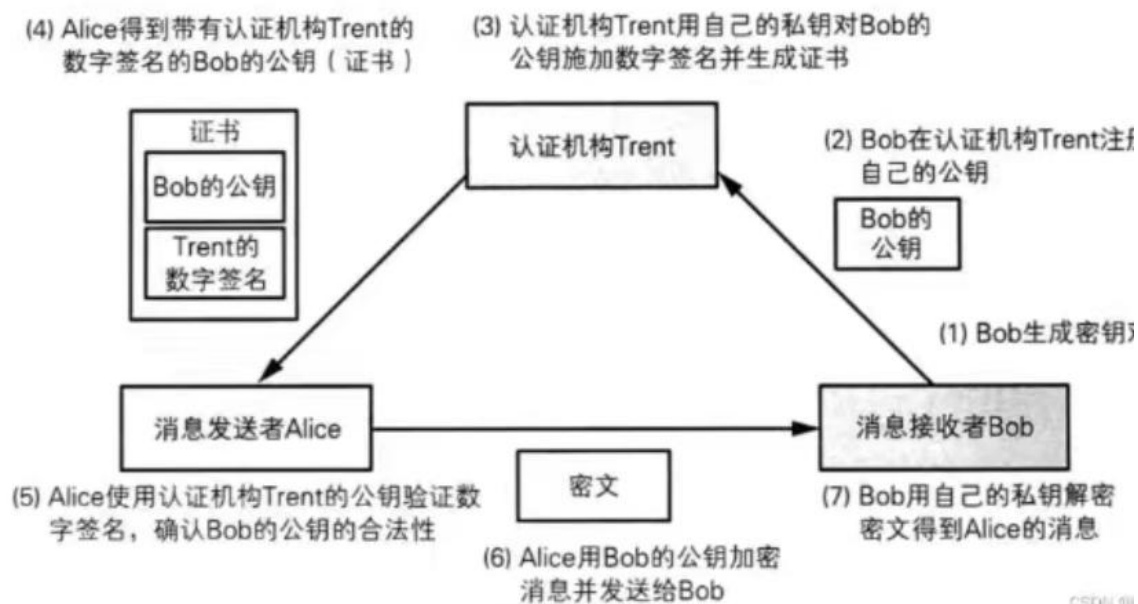
- 1、假设服务器程序有自己的公钥证书，其公钥证书为服务器用自己私钥 SK_{CA} 为自己签发的公钥证书， $C_{CA} = E_{SK_{CA}}[T, ID_{CA}, PK_{CA}]$ ，这个证书假设放在服务器某个地方，每个注册用户都可以获取，从而可以获取公钥证书管理机构的公钥。
- 2、假设这个公钥管理机构是可信的；

1.4 操作系统等环境配置

- 操作系统：win11
- 软件系统：pycharm/ Visual studio/vscode
- 编译工具：pycharm2023/vs2022/MySQL workbench
- 编程语言：python/C++/sql/html

3. 数据库系统的建立

公钥证书系统的建立



1. 首先选择了要求所需要的 RSA 公钥密码算法，并利用该算法为双方各分配一个公钥和一个私钥
2. 然后服务器与客户端成功进行连接之后分别将自己的公钥发送给对方

3. 在客户端使用公钥密码算法随机生成一个 AES 算法的会话密钥，并通过服务器的公钥进行加密，将加密后的密钥发送给管理员
4. 管理员收到加密后的会话密钥后，使用私钥进行解密，得到原始的 AES 会话密钥
5. 利用 AES 和 AES 加密算法，双方就可以对要传送的信息进行加密，并将加密后的信息发送给对方
6. 双方在收到加密后的信息后，利用 AES 和 AES 解密算法，对信息进行解密，就可以得到原始信息，实现双方的通讯

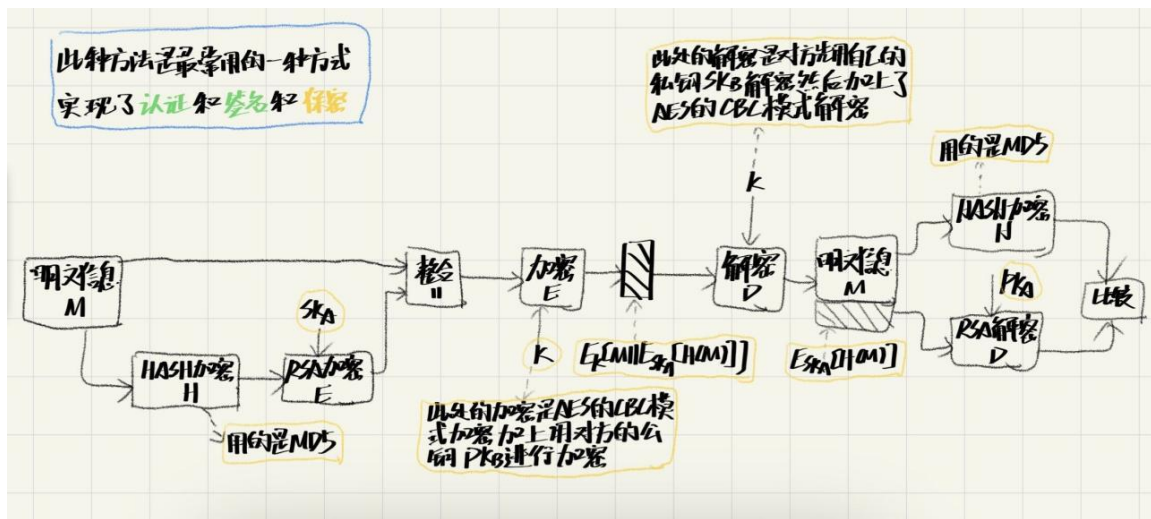
在以上步骤在设计保密通信协议完后，我发现对于本学期后半的课程所学的知识用到的并不多，所以添加了一些额外的步骤来保证信息的安全性，比如：

1. 双方可以在发送加密后的信息之前，通过 MD5 数字签名的方式对信息进行签名，以验证信息的完整性和可靠性，保证发出消息方无法反悔。
2. 双方可以利用时间戳的方式，对信息进行时间限制，以防止信息被重放

(2)后来补充的认证和签名

因为以上实现的程序总感觉还欠缺点什么，所以又对其加密传输过程进行了消息验证码（MAC）和数字签名的补充，更能符合这学期密码学课程设置的要求

补充的加密传输流程图如下：



1. 首先将明文信息用 MD5 哈希加密，并用自己的私钥再次进行加密，以达到签名的目的
2. 然后将明文消息与刚刚经过两次加密的密文消息整合
3. 接下来将整合的消息进行加密，此处是用 AES 和对方的公钥进行加密
4. 然后收方在收到消息之后对其进行解密，用自己的私钥和 AES 共享密钥解密

5. 将明文信息用 MD5 进行哈希加密，并和用对方的公钥解密的密文信息进行比较，达到认证的目的

数字签名

基本原理

先说数字签名的基本原理。数字签名过程跟加密通信有着一定的对称性，这种对称有着一种逻辑上的美。

加密通信是用公钥进行加密，而用私钥进行解密。而数字签名刚好相反，是采用私钥加密，公钥解密。对于加密通信，公钥加密过程就是通过加密算法把信息加密成密文，私钥解密过程就是通过解密算法解密密文。而对于数字签名，私钥加密过程是通过签名算法来生成数字签名的过程，而公钥解密过程是通过验证算法来确定数字签名是不是有私钥持有者签署的。可以看到，加密算法，解密算法，签名算法，验证算法，对称性是很明显的。

数字签名的主要的作用是认证签署人身份，说得具体点，就是让所有人能够确认这个数字签名是不是由私钥的持有人创建的。数字签名是由签名算法去生成的，签名算法的输入有两个，一个是私钥，另一个是被签署的信息，输出的一个字符串就是数字签名了。签名到底是不是由私钥持有人签署的，要通过验证算法判断。验证算法有三个输入，一个是信息本身，另外一个数字签名，第三个是公钥，输出的结果就是验证成功或者验证失败。数字签名过程中，私钥是签名 key，公钥是验证 key。

实际作用

然后是数字签名的作用。数字签名有三大作用，第一个是认证，第二个是防止抵赖，第三个是保证文件完整性，也就是没有被篡改过。

先看第一个认证，意思就是确认签署人身份，这个跟纸笔签名的作用是一致的。再看第二个防止抵赖，同样是纸笔签名也拥有的特性，一份合同签署了，就要承担责任，白纸黑字，不能耍赖。第三个作用是保证文件没有被篡改过，这个作用纸笔签名很难保证，比如签名只签署了最后一页，那么如果有人想悄悄换掉了前几页的内容，签名本身是阻止不了的。但是数字签名就可以，因为数字签名是由两个输入运算得出的，一个是私钥，另一个就是文件，所以如果在验证过程中，发现文件有改动，验证会失败。这个很类似于现实世界签合同的时候，有时候需要故意用签名或者图章覆盖有文字的区域，或者在写信的时候，给信封加上蜡封，这些措施也都是为了防篡改。

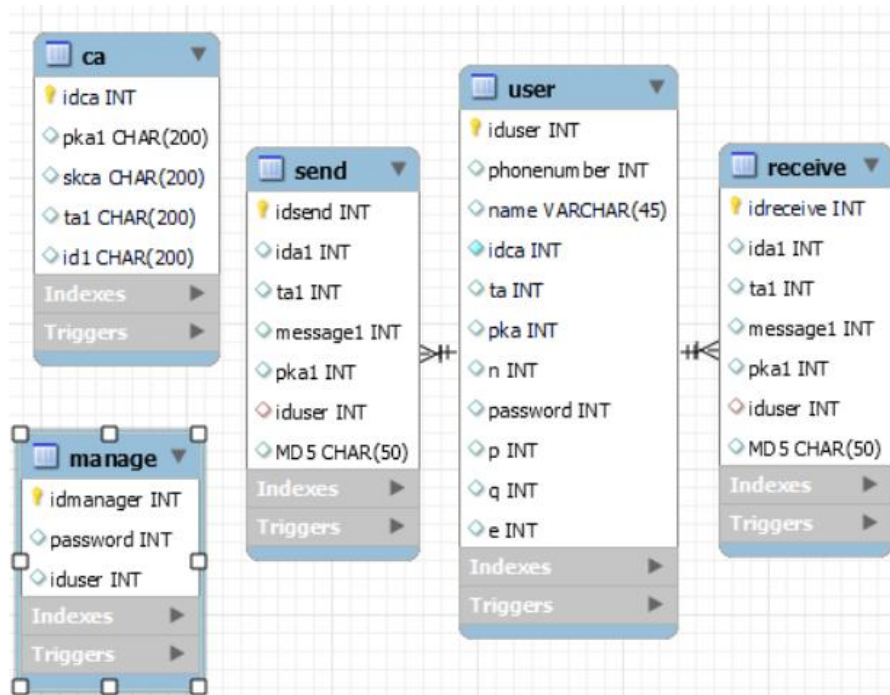
这就是数字签名的三大作用，主要用在各种防伪造场合，在世界上很多国家，数字签名都是有法律效力的。另外要注意，数字签名属于电子签名的一种，但并不是所有的电子签名都是数字签名，这两个概念要区分一下。

总结

数字签名是公钥加密技术的两大应用之一。主要采用了私钥加密，公钥解密的方式，文件签署者用私钥签署文件，就表示他认可了这个文件的内容。要验证数字签名，只需要签名人公布自己的公钥即可，其他人通过验证数字签名即可验证文件是私钥签署的。签署人不需要暴露自己的私钥，就可以间接证明自己拥有私钥。除了认证签署人身份，数字签名还有两个作用，分别是防止抵赖和保证文件完整性。

这是一种最常用的方式，实现了认证和签名和保密的功能

1. 数据库 E/R 图，其中 **senf** 和 **receive** 两个表之间要通过外键链接 **user** 表



部分触发器语句

```

CREATE TRIGGER insert_manage
BEFORE INSERT ON manage
FOR EACH ROW
BEGIN
    IF NOT EXISTS (SELECT * FROM user WHERE iduser = NEW.iduser) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid user id';
    END IF;
END;

CREATE TRIGGER delete_manage
BEFORE DELETE ON manage
FOR EACH ROW
BEGIN
    IF not EXISTS (SELECT * FROM manage WHERE iduser = OLD.iduser) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cannot delete due to foreign key constraint';
    END IF;
END;
    
```

```

END;

CREATE TRIGGER insert_ca
BEFORE INSERT ON ca
FOR EACH ROW
begin
END;

DELIMITER //
CREATE TRIGGER delete_ca
BEFORE DELETE ON ca
FOR EACH ROW
BEGIN
END;
    
```

2. pycharm 进行数据库的连接

连接数据库

```

connection = pymysql.connect(host='localhost', port=3306, user='root',
password='4239692', db='rsa', charset='utf8')
    
```

```
# 创建游标对象
cursor = connection.cursor()
app = Flask(__name__)
app.secret_key = "mysecretkey"
```

3. 数据库系统与前端页面的连接

```
@app.route('/')
def index():
    return render_template('index.html')
```

4. 定义一个登录页面并判断登录是否成功，成功后可返回收件箱和发件箱链接

用户可以对收件箱当中的信息进行验证发送方是否合理，也可以验证发送方的证书是否是满足其公钥证书的


```

@app.route('/login', methods=['GET', 'POST'])
def login():
    # 获取表单数据
    if request.method == 'POST':
        iduser = request.form['iduser']
        password = request.form['password']
        sql = "SELECT * FROM user WHERE iduser='%s' AND password='%s'" % (iduser, password)
        cursor.execute(sql)
        result = cursor.fetchall()
        # 验证用户
        if len(result) > 0:
            session['iduser'] = iduser
            return redirect(url_for('me', user=iduser))
        else:
            message = "用户名或密码错误! "
            return render_template('login.html', message=message)
    # 显示登录页面
    return render_template('login.html')
# 重定向到“me”页面
@app.route('/me/<user>', methods=['GET'])
def me(user):
    with connection.cursor() as cursor:
        # 查询所有与当前用户相关的数据
        sql = "SELECT * FROM `receive`"
        cursor.execute(sql)
        receives = cursor.fetchall()
    return render_template('me.html', receives=receives)

```

登录

用户名: 密码:

登录


```
border-radius: 4px;
cursor: pointer;
}
</style>
</head>
<body>
  <form action="/login" method="post">
    <h1>登录</h1>
    <label for="iduser">用户名: </label>
    <input type="text" id="iduser" name="iduser" placeholder="请输入用户名" required>

    <label for="password">密码: </label>
    <input type="password" id="password" name="password" placeholder="请输入密码" required>

    <button type="submit">登录</button>
  </form>
</body>
</html>
```

- [View users](#)

idreceive	ida1	ta1	message1	pka1	iduser	MD5
3	1	1	123456	1	2	0xe10adc3949ba59abbe56e057f20f883e
13	1	13	1234	1	1	0x81dc9bdb52d04dc20036dbd8313ed055
27	2142	12423	243252	24234	8	0x53d7a0a333f28714648db73efbe1b57e

[send](#)

5. 注册以及发送证书，其中在管理员端还可以进行证书信息的更改

```

@app.route('/user/add', methods=['GET', 'POST'])
def add_user():
    if request.method == 'POST':
        iduser = request.form.get('iduser')
        phonenumber = request.form.get('onenumber')
        # 验证身份证号的正则表达式
        id_pattern = re.compile(r'^\d{4}(\d|X)$')
        # 验证手机号的正则表达式
        phone_pattern = re.compile(r'^1[3456789]\d{9}$')
        name = request.form.get('name')
        idca = request.form.get('idca')
        ta = request.form.get('ta')
        password = request.form.get('password')
        key_size = 8
        p = Generate_prime(key_size)
        q = Generate_prime(key_size)
        n, e1, d = KeyGen(p, q)
        pka = d
        e = e1
        key = os.urandom(16)
        aes_test = FileAES(key)
        p1 = str(pka)
        pka1 = aes_test.encrypt(p1)
        skca = key[0]
        t1 = str(ta)
        ta1 = aes_test.encrypt(t1)
        i1 = str(idca)

```

可以在前端页面

```

        with connection.cursor() as cursor:
            sql = "INSERT INTO user (iduser,phonenumner,name,idca,ta,pka,n,password,p,q,e)
            iduser, phonenumner, name, idca, ta, pka, n, password, p, q, e)
            cursor.execute("INSERT INTO ca (idca,pka1,skca,ta1,id1) VALUES (%s,%s,%s,%s,%s)
            (idca, pka1, skca, ta1, id1))
            print(sql)
            cursor.execute(sql)
            connection.commit()
        return redirect(url_for('users'))
    else:
        return render_template('add_user.html')

@app.route('/coach/edit/<int:iduser>', methods=['GET', 'POST'])
def edit_user(iduser):
    with connection.cursor() as cursor:
        cursor.execute("SELECT * FROM user WHERE iduser=%s", (iduser,))
        user = cursor.fetchone()
    if request.method == 'POST':
        x=iduser
        iduser = request.form('iduser')
        phonenumner = request.form('phonenumner')
        name = request.form('name')
        idca = request.form('idca')
        ta = request.form('ta')
        pka = request.form('pka')
        n = request.form('n')
        password = request.form('password')

```

管理端注册的用户列表示例（前几行为调试代码，不作为真正的 user 输入，后几行为按正确方法注册的用户）

user List

[Add user](#)

Delete	Delete	Delete	Delete	Delete	Delete	Delete	Delete	Delete	Delete				
id	user	phonenumner	name	idca	ta	pka	n	password	p	q	e	edit	delete
1		1		1	1	1	1	1	1	1	None	Edit	
2		2		2	2	2	29651	2	199	149	None	Edit	
3		2134	1232	1	26	2939	47671	1	193	247	None	Edit	
4		143224	1142	1	1	29069	30997	321443	223	139	None	Edit	
5		121413	dewd	1	1	449	18769	15421	137	137	None	Edit	
6		1241241	huiwheof	1	1	7241	29329	15324224	211	139	16301	Edit	
8		1241252	ge2efwqe	5	1645	15863	29999	5342523	229	131	18287	Edit	
12		1341124	jqwiopq	1	626	3559	20989	14232	151	139	19339	Edit	
1243		214242134	sdfjlf	4	1325	659	20567	221351	131	157	3539	Edit	

• [View all](#)

注册的部分证书（示例）前几行仍为调试数据，后面几行的数据均为经过 AES 加密之后的值

ca List

[Add ca](#)

[Delete](#) [Delete](#) [Delete](#) [Delete](#)

idca	pkal	skca	ta1	id1	edit delete
1 2939		1 1		1	Edit
2 1		1 1		1	Edit
4 qt1OjEkwx4xV7RrQyv1FPg==	50	P88ulbMnwmiH/ATrgNOMBQ==	wm25fwP3d9IQ16NA+v3mJw==		Edit
5 ptUOpG3SonLdHSWOYWny0g==	177	8mGmcM+QwsDHbDJ3XnafSA==	/vfWWJtWhO1nlXqR1fEndQ==		Edit

• [View all](#)

修改证书的参数

```
@app.route('/ca/edit/<int:idca>', methods=['GET', 'POST'])
def edit_ca(idca):
    with connection.cursor() as cursor:
        cursor.execute("SELECT * FROM ca WHERE idca=%s", idca)
        ca = cursor.fetchone()
    if request.method == 'POST':
        x=idca
        idca = request.form.get('idca')
        pkal = request.form.get('pkal')
        skca = request.form.get('skca')
        ta1 = request.form.get('ta1')
        id1 = request.form.get('id1')
        with connection.cursor() as cursor:
            cursor.execute("UPDATE `ca` SET idca=%s,pkal=%s, skca=%s, ta1=%s, id1=%s WHERE idca=%s (idca,pkal,skca,ta1,id1, x))")
            connection.commit()
    return redirect(url_for('ca'))
```

```
<h1>Edit user</h1>
<form method="post" action="{ url_for('edit_user', iduser=user[0]) }">
    <label>iduser:</label>
    <input type="number" name="iduser" value="{ user[0] }"><br>
    <label>phonenumner:</label>
    <input type="number" name="phonenumner" value="{ user[1] }"><br>
    <label>name</label>
    <input type="text" name="name" value="{ user[2] }"><br>
    <label>idca:</label>
    <input type="number" name="idca" value="{ user[3] }"><br>
    <label>ta:</label>
    <input type="number" name="ta" value="{ user[4] }"><br>
    <label>pka:</label>
    <input type="number" name="pka" value="{ user[5] }"><br>
    <label>n:</label>
    <input type="number" name="n" value="{ user[6] }"><br>
    <label>password:</label>
    <input type="number" name="password" value="{ user[7] }"><br>
    <label>p:</label>
    <input type="number" name="p" value="{ user[8] }"><br>
    <label>q:</label>
    <input type="number" name="q" value="{ user[9] }"><br>
    <label>e:</label>
    <input type="number" name="e" value="{ user[10] }"><br>
```

消息的发送以及接收:

```
<th>message1</th>
<th>pka1</th>
<th>iduser</th>
<th>MD5</th>
<th>edit</th>
<th>delete</th>
</tr>
{% for send in sends %}
<tr>
<td>{{send[0]}}</td>
<td>{{send[1]}}</td>
<td>{{send[2]}}</td>
<td>{{send[3]}}</td>
<td>{{send[4]}}</td>
<td>{{send[5]}}</td>
<td>{{send[6]}}</td>
<td><a href="{% url_for('edit_send', idsend=send[0]) %}">

</tr>
<form action='/send/delete/{{send[0]}}' method='POST'>
<input type='submit' value='Delete'>
</form>
```

send List

[Add send](#)

[Delete](#) [Delete](#) [Delete](#) [Delete](#)

idsend	ida1	ta1	message1	pka1	iduser	MD5	edit delete
3	1	1	123456	1	2	0xe10adc3949ba59abbe56e057f20f883e	Edit
5	253916	8812432	32143213	1471	5	ptUOpG3SonLdHSWOYWny0g==	Edit
13	1	13	1234	1	1	0x81dc9bdb52d04dc20036dbd8313ed055	Edit
27	2142	12423	243252	24234	8	0x53d7a0a333f28714648db73efbe1b57e	Edit

• [View all](#)

receive List

[Add receive](#)

idreceive	ida1	ta1	message1	pka1	iduser	MD5	edit	delete
3	1	1	123456	1	2	0xe10adc3949ba59abbe56e057f20f883e	Edit	
5	253916	8812432	32143213	1471	5	ptUOpG3SonLdHSWOYWny0g==	Edit	
13	1	13	1234	1	1	0x81dc9bdb52d04dc20036dbd8313ed055	Edit	
27	2142	12423	243252	24234	8	0x53d7a0a333f28714648db73efbe1b57e	Edit	

- [View all](#)

Add send

idsend:

ida1:

ta1:

message1:

pka1:

iduser:

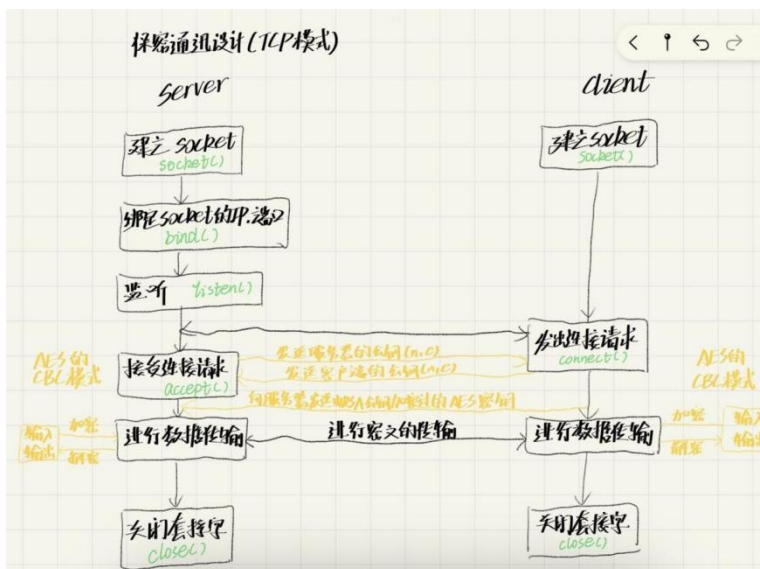
MD5:

[cancel](#)

消息接收者可以根据 MD5 消息认证码，以及发送者发送来的证书的部分信息，利用证书上可以查找到的公钥，验证是否是发送者发送的信息，同时也可以防止发送者抵赖

3. 保密通信协议

本次实验首先对保密协议进行设计，因为保密协议其实是整体上的大局观，类似一个指导编程的思维导图，如下是保密协议的相关流程：



3.1. 保密通信协议设计思路

在设计保密通信协议时，我采取了如下思路进行设计：

3.2. 保密通信协议设计的流程

(1) 开始的利用 AES 和 RSA 实现的通讯

通讯的总体流程图如下：

- 首先是服务端先建立 socket 端口，并绑定端口的 IP 地址以及端口号
- 然后服务器开始监听，此时客户端也需要绑定 socket 并发出连接请求

3. 当服务器收到了连接请求并成功与客户端建立连接之后，先有服务器将自己的公钥发给客户端，然后客户端再将自己的公钥发给服务器，实现了双发可以知道对方的公钥
4. 接下来有客户端随机生成一个 AES 密钥，并将此密钥用服务器的公钥进行加密发送给服务器
5. 服务器端将其用自己的私钥进行解密，然后就可以实现 AES 的密钥共享了
6. 接下来双方进行信息传输的时候，先使用 AES 的 CBC 模式进行对明文信息的加密，然后每一次再用对方的公钥再次加密并对其进行传输
7. 接收到信息后先用自己的私钥对其进行解密，然后用共享的 AES 密钥对其进行解密，然后将解密的信息在程序中输出，即实现了双方的通讯
8. 最后当任意一方输入 `exit` 后将结束本次对话，双方都需要关闭 `socket` 然后退出程序

3.3. 消息认证码 (MAC)

消息认证码 (Message Authentication Code) 是一种确认完整性并进行认证的计算，取三个单词的首字母，简称 MAC:

- 消息认证码的输入包括任意长度的消息和一个发送者与接收者之间的共享的密钥，它可以输出固定长度的数据，这个数据成为 MAC 值
- 根据任意长度的消息输出固定长度的数据，这一点和散列函数很类似。但是单向散列函数中计算散列值时不需要密钥，相对地，消息认证码中则需要使用发送者与接收者之间的共享密钥
- 要计算 MAC 必须持有共享密钥，没有共享密钥的人就无法计算 MAC 值，消息认证正是利用这一性质来完成认证的。此外，和单向散列函数的散列值一样，哪怕消息中发生 1 比特的变化，MAC 值一会产生变化，消息认证码正是利用这一性质来确认完整性的。

消息认证码是一种与密钥相关联的单向散列函数

应用实例

SWIFT

SWIFT 的全称是 Society for Worldwide Interbank Financial Telecommunication (环境银行金融电信协会)，是于 1973 年成立的一个组织，其目的是为国际银行之间的交易保驾护航。该组织成立时有 15 个成员国，2008 年时，已经发展到了 208 个成员国。

银行与银行之间是通过 SWIFT 来传递交易信息的。而为了确认消息的完整性以及对消息进行验证，SWIFT 中使用了消息认证码。

在使用公钥密码进行密钥交换之前，消息认证码所使用的共享密钥都是由人来进行配送的。

IPsec

IPsec 是对互联网基本通信协议——IP 协议（Internet Protocol）增加安全性的一种方式。IPsec 中，对通信内容的认证和完整性校验都是采用的消息认证码来完成的。

SSL/TLS

SSL/TLS 是我们在网上购物等场景中所使用的通信协议。SSL/TLS 中对通信内容的认证和完整性校验也使用了消息认证码。

实现方式

使用单向散列函数实现

使用 SHA-2 之类的单向散列函数可以实现消息认证码，其中一种实现方法称为 HMAC。

使用分组密码实现（也是本次实验采取的方式）

使用 AES 之类的分组密码可以实现消息认证码。

将分组密码的密钥作为消息认证码的共享密钥来使用，并用 CBC 模式将消息全部加密。此时，初始化向量是固定的。由于消息认证码不需要解密，因此将除最后一个分组以外的密文部分全部丢弃，而将最后一个分组用作 MAC 值。由于 CBC 模式的最后一个分组会受到整个消息以及密钥的双重影响，因此可以将它用作消息认证码。例如 AES-CMAC（RFC4493）就是一种基于 AES 来实现的消息认证码。

其他实现方法

使用流密码和公钥密码等也可以实现消息认证码。

3.4. 通讯的重要代码分析

这一次分为了好几个部分进行实现所以加密的部分代码以及消息认证等代码实现都会分节讲述，这一节讲述的主要代码是通讯的部分，由于通讯实现包装化，所以在主函数中服务器端和客户端分别只有一个 `runServer()` 和 `runClient()` 函数，这也是本次重点要讲的两部分代码：

(1) `runClient()`

- 其实客户端大致的流程已经在上述中讲到了，先加载套接字库、创建一个套接字供使用、然后向服务器发出连接请求
- 然后客户端先接收对面发来的公钥，接着把自己的公钥发过去，然后发送一个共享的 AES 密钥
- 最后的收发信息也是用补充完认证和签名机制后的流程来实现

代码如下：

```

int runClient()
{
    // 1. 加载套接字库
    WORD wVersionRequested;
    WSADATA wsaData;
    wVersionRequested = MAKEWORD(2, 2);
    int err = WSASStartup(wVersionRequested, &wsaData);
    if (err)
        printf("C: socket 加载失败\n");
    else
        printf("C: socket 加载成功\n");
    // 2. 创建一个套接字供使用
    SOCKET ServerSocket = socket(AF_INET, SOCK_STREAM, 0);
    // 3. 向服务器发出连接请求
    SOCKADDR_IN socksin; // 记录服务器端地址
    socksin.sin_family = AF_INET;
    socksin.sin_port = htons(6020);
    socksin.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
    int rf = connect(ServerSocket, (SOCKADDR*)&socksin, sizeof(socksin));
    if (rf == SOCKET_ERROR)
        printf("C: 与服务器连接失败\n");
    else
    {
        printf("C: 成功与服务器进行连接\n");

        // 接收公钥对 客户端应该也需要自己的公私钥
        // 采用服务端先发送 客户端后发送的策略 并准备一个flag 服务端为0 客户端为1
        // 一 客户端接收
        char recv_from_server_public_key[100] = { 0 };
        recv(ServerSocket, recv_from_server_public_key, 100, 0);
        printf("C: 从服务器接收服务器的公钥对:%s\n", recv_from_server_public_key);
        char recv_N[100], recv_E[100];
        int count_for_N = 0, count_for_E = 0;
        bool divide_for_key = false;
        for (int i = 0; i < strlen(recv_from_server_public_key); i++)
        {
            if (recv_from_server_public_key[i] == ',') { divide_for_key = true; continue; }
            if (!divide_for_key)
                recv_N[count_for_N++] = recv_from_server_public_key[i];
            else
                recv_E[count_for_E++] = recv_from_server_public_key[i];
        }

        PublicKey server_public_key;
        server_public_key.nE = atoi(recv_E);
        server_public_key.nN = atoll(recv_N);

        // 二 客户端发送
        RsaParam client_param = RsaGetParam();
        m_cParam.d = client_param.d;
        m_cParam.e = client_param.e;
        m_cParam.n = client_param.n;
        PublicKey client_public_key = GetPublicKey();
        char send_to_server_public_key[100], send_N[100], send_E[100];
        sprintf(send_N, "%lu", client_public_key.nN);
        itoa(client_public_key.nE, send_E, 10);
        strcpy(send_to_server_public_key, send_N);
        int len_of_N = strlen(send_N), len_of_E = strlen(send_E);
        send_to_server_public_key[len_of_N] = ',';
        strncpy(send_to_server_public_key + len_of_N + 1, send_E, len_of_E);
        send_to_server_public_key[len_of_N + len_of_E + 1] = '\0';
        printf("C: 向服务器发送客户端的公钥对:%s\n", send_to_server_public_key);
        send(ServerSocket, send_to_server_public_key, strlen(send_to_server_public_key), 0);

        // 准备发送AES 密钥
        char en_aes_key[300] = { '\0' };
        char aes_key[16] = { 0 };
        GenerateAesKey(aes_key);
        printf("C: 随机生成一个AES 密钥:");
    }
}

```

```

for (int i = 0; i < 16; i++) {
    printf("%c", aes_key[i]);
}
printf("\n");
int len_of_en_aes_key = 0;
for (int i = 0; i < 8; i++)
{
    int p1 = i * 2, p2 = i * 2 + 1;
    int num1 = int(aes_key[p1]);
    int num2 = int(aes_key[p2]);
    UINT64 sixteen_bits = (num1 << 8) + num2;
    UINT64 en_sixteen_bits = Encry(sixteen_bits, server_public_key);
    char en_sixteen_bits_to_char[20];
    sprintf(en_sixteen_bits_to_char, "%lu", en_sixteen_bits);
    strncpy(en_aes_key + len_of_en_aes_key, en_sixteen_bits_to_char, strlen(en_sixteen_bits_to_c
har));

    len_of_en_aes_key += strlen(en_sixteen_bits_to_char);
    char comma = ',';
    en_aes_key[len_of_en_aes_key] = comma;
    len_of_en_aes_key += 1;
}
en_aes_key[len_of_en_aes_key] = '\0';
// 发送加密的AES 密钥给服务器
printf("C: 向服务器发送加密的AES 密钥:%s\n", en_aes_key);
send(ServerSocket, en_aes_key, len_of_en_aes_key, 0);

// 开始准备收发信息
char plaintext[500], ciphtext[500] = { 0 };
while (true)
{
    string str_aes_key = charkey_to_strkey(aes_key);

    // 一 客户端发送加密信息
    // 补充: 先用哈希 (MD5) 进行加密 这个值再用私钥加密 后与整体进行一起加密
    printf("C: 请输入明文:");
    setbuf(stdin, NULL);
    scanf("%[^\n]s", plaintext); // 使得空行代表读取完毕而不是空格
    bool exit = false;
    if (strcmp(plaintext, "exit") == 0) { exit = true; }
    string mid_plain_text = plaintext;
    // hash 也是 32 个 16 进制数的字符串
    string hash = MD5(mid_plain_text).getstring();
    PublicKey client_private_key;
    client_private_key.nE = m_cParament.d;
    client_private_key.nN = m_cParament.n;
    // 用自己的私钥去加密
    char en_hash[100] = { '\0' };
    int len_of_en_text = 0;
    for (int i = 0; i < hash.size() / 4; i++)
    {
        int p1 = i * 4, p2 = i * 4 + 1;
        int p3 = i * 4 + 2, p4 = i * 4 + 3;
        int num1 = int(hash[p1]);
        int num2 = int(hash[p2]);
        int num3 = int(hash[p3]);
        int num4 = int(hash[p4]);
        UINT64 sixteen_bits = (num1 << 12) + (num2 << 8) + (num3 << 4) + num4;
        UINT64 en_sixteen_bits = Encry(sixteen_bits, client_private_key);
        char en_sixteen_bits_to_char[20];
        sprintf(en_sixteen_bits_to_char, "%lu", en_sixteen_bits);
        strncpy(en_hash + len_of_en_text, en_sixteen_bits_to_char, strlen(en_sixteen_bits_to_cha
r));

        len_of_en_text += strlen(en_sixteen_bits_to_char);
    }
    en_hash[len_of_en_text] = '\0';
    string wxn_en_hash = en_hash;
    // 1. 输入信息 并将其用 AES 加密为 128 位的数据 需要实现 CBC 分组加密 即需要加上 xor 并将分组加密完的数
据合成一个字符串
    // 为了方便区分最后的 hash
    mid_plain_text += ".";
    mid_plain_text += wxn_en_hash;
}

```

的字符串

```
string mid_en_text = en_cbc_aes(mid_plain_text, str_aes_key); // 返回的是一个CBC 分组加密完合成

// 2. 用对方的公钥(server_public_key)进行加密 由于合成的字符串每一个字符是4 位 所以需要4 个为一组
char en_text[500] = { '\0' };
int len_of_en_text = 0;
for (int i = 0; i < mid_en_text.size() / 4; i++)
{
    int p1 = i * 4, p2 = i * 4 + 1;
    int p3 = i * 4 + 2, p4 = i * 4 + 3;
    int num1 = int(mid_en_text[p1]);
    int num2 = int(mid_en_text[p2]);
    int num3 = int(mid_en_text[p3]);
    int num4 = int(mid_en_text[p4]);
    UINT64 sixteen_bits = (num1 << 12) + (num2 << 8) + (num3 << 4) + num4;
    UINT64 en_sixteen_bits = Encry(sixteen_bits, server_public_key);
    char en_sixteen_bits_to_char[20];
    sprintf(en_sixteen_bits_to_char, "%lu", en_sixteen_bits);
    strncpy(en_text + len_of_en_text, en_sixteen_bits_to_char, strlen(en_sixteen_bits_to_char));
    len_of_en_text += strlen(en_sixteen_bits_to_char);
    char comma = ',';
    en_text[len_of_en_text] = comma;
    len_of_en_text += 1;
}
en_text[len_of_en_text] = '\0';
// 3. 将加密完的字符串发送
send(ServerSocket, en_text, strlen(en_text), 0);
if (exit)
{
    break;
}

// 二 客户端接收加密信息
// 1. 接收对方发过来的字符串
recv(ServerSocket, ciphtext, 500, 0);
// 2. 用自己的私钥(client_private_key)解密发过来的字符串
int len_of_ciphtext = 0;
for (int i = 0; i < strlen(ciphtext) / 4; i++)
{
    if (en_aes_key[len_of_ciphtext++] == '.')
    {
        // 判断到加密后的hash 值了
        break;
    }
    char en_sixteen_bits_to_char[20] = { '\0' };
    int p = 0;
    while (en_aes_key[len_of_ciphtext++] != ',')
        en_sixteen_bits_to_char[p++] = en_aes_key[len_of_ciphtext - 1];
    UINT64 en_sixteen_bits = atoll(en_sixteen_bits_to_char);
    UINT64 sixteen_bits = Decry(en_sixteen_bits);
    ciphtext[i * 4] = (sixteen_bits >> 12) % 16;
    ciphtext[i * 4 + 1] = (sixteen_bits >> 8) % 16;
    ciphtext[i * 4 + 2] = (sixteen_bits >> 4) % 16;
    ciphtext[i * 4 + 3] = sixteen_bits % 16;
}
string mid_ciphtext = ciphtext;
// 补充: 对hash 的解密 需要用服务器的公钥解密
char recv_en_hash[100];
// 先保存之前的密钥 防止丢失
UINT64 mid_n = m_cParament.n;
UINT64 mid_d = m_cParament.d;
UINT64 mid_e = m_cParament.e;
// 赋值为服务器的公钥 (server_public_key)
m_cParament.n = server_public_key.nN;
m_cParament.d = server_public_key.nE;
for (int i = 0; i < strlen(ciphtext) / 4; i++)
{
    if (en_aes_key[len_of_ciphtext++] == '.')
    {
        // 判断到加密后的hash 值了
    }
}
```

```

        break;
    }
    char en_sixteen_bits_to_char[20] = { '\0' };
    int p = 0;
    while (en_aes_key[len_of_ciphertext++] != ',')
        en_sixteen_bits_to_char[p++] = en_aes_key[len_of_ciphertext - 1];
    UINT64 en_sixteen_bits = atoll(en_sixteen_bits_to_char);
    UINT64 sixteen_bits = Decry(en_sixteen_bits);
    recv_en_hash[i * 4] = (sixteen_bits >> 12) % 16;
    recv_en_hash[i * 4 + 1] = (sixteen_bits >> 8) % 16;
    recv_en_hash[i * 4 + 2] = (sixteen_bits >> 4) % 16;
    recv_en_hash[i * 4 + 3] = sixteen_bits % 16;
}
// 恢复之前的缓冲区
m_cParament.n = mid_n;
m_cParament.d = mid_d;
m_cParament.e = mid_e;
string recv_hash = recv_en_hash;
// 3. 将其用 AES 解密(在解密函数中将其分为 128 位一组的数据)各个组的 128 位的数据 需要实现 CBC 分组解密
即需要加上 xor
string de_mid_ciphertext = de_cbc_aes(mid_ciphertext, str_aes_key);
// 4. 输出解密后的明文(如果 hash 值验证相等的话)
if (MD5(de_mid_ciphertext).getString() == recv_hash)
{
    char* time;
    strcpy(time, getTime());
    printf("C: [%s]经过解密后的明文:", time);
    cout << de_mid_ciphertext;
    if (de_mid_ciphertext == "exit")
    {
        break;
    }
}
else
{
    cout << "本次接收有误, 需要重新接收" << endl;
    continue;
}
}
// 如果用户选择退出, 则向服务器发送退出请求
printf("\nC: 退出...");
}
closesocket(ServerSocket);
WSACleanup();
return 0;
}

```

(2)runServer()

- 其实服务器端大致的流程也在上述中讲到了, 先加载套接字库、创建一个套接字供使用、将套接字绑定到本地地址和端口上、然后将套接字设置为监听模式, 以接收客户端请求并等待并接收客户端请求, 返回新的连接套接字
- 然后服务器先发送自己的公钥, 接着等待接收对方的公钥, 然后等待对方的发来的共享 AES 密钥
- 最后的收发信息也是用补充完认证和签名机制后的流程来实现

代码如下:

```

int runServer()
{
    //1. 加载套接字库
    WORD wVersionRequested = MAKEWORD(2, 2);
    WSADATA wsaData;
    int err = WSASStartup(wVersionRequested, &wsaData);
    if (err)
        printf("S: socket 加载失败\n");
    else
        printf("S: socket 加载成功\n");
    //2. 创建一个套接字供使用
    SOCKET ServerSocket = socket(AF_INET, SOCK_STREAM, 0);
    //3. 将套接字绑定到本地地址和端口上
    SOCKADDR_IN addr;
    addr.sin_family = AF_INET;
}

```

```

addr.sin_port = htons(6020);
addr.sin_addr.S_un.S_addr = htonl(INADDR_ANY);
bind(ServerSocket, (SOCKADDR*)&addr, sizeof(SOCKADDR));
//4. 将套接字设置为监听模式, 以接收客户端请求
err = listen(ServerSocket, 5);
if (err)
    printf("S: 绑定端口失败 请重新启动程序\n");
else
    printf("S: 成功绑定端口 等待客户端连接\n");
//5. 等待并接收客户端请求, 返回新的连接套接字
SOCKADDR_IN addr_out;
int len = sizeof(SOCKADDR);
SOCKET ClientSocket = accept(ServerSocket, (SOCKADDR*)&addr_out, &len);
// 6. 计算密钥 主要是RSA 的公钥和私钥 并准备发给客户端 需要补充的是从客户端接收
// 采用服务端先发送 客户端后发送的策略 并准备一个flag 服务端为0 客户端为1
// 一 服务器发送
RsaParam server_param = RsaGetParam();
m_cParament.d = server_param.d;
m_cParament.e = server_param.e;
m_cParament.n = server_param.n;
PublicKey server_public_key = GetPublicKey();
char send_to_client_public_key[100], send_N[100], send_E[100];
sprintf(send_N, "%lu", server_public_key.nN);
itoa(server_public_key.nE, send_E, 10);
strcpy(send_to_client_public_key, send_N);
int len_of_N = strlen(send_N), len_of_E = strlen(send_E);
send_to_client_public_key[len_of_N] = ',';
strncpy(send_to_client_public_key + len_of_N + 1, send_E, len_of_E);
send_to_client_public_key[len_of_N + len_of_E + 1] = '\0';
printf("S: 向客户端发送服务器的公钥对:%s\n", send_to_client_public_key);
send(ClientSocket, send_to_client_public_key, strlen(send_to_client_public_key), 0);

// 二 服务器接收
char recv_from_client_public_key[100] = { 0 };
recv(ClientSocket, recv_from_client_public_key, 100, 0);
printf("S: 从客户端接收客户端的公钥对:%s\n", recv_from_client_public_key);
char recv_N[100], recv_E[100];
int count_for_N = 0, count_for_E = 0;
bool divide_for_key = false;
for (int i = 0; i < strlen(recv_from_client_public_key); i++)
{
    if (recv_from_client_public_key[i] == ',') { divide_for_key = true; continue; }
    if (!divide_for_key)
        recv_N[count_for_N++] = recv_from_client_public_key[i];
    else
        recv_E[count_for_E++] = recv_from_client_public_key[i];
}

PublicKey client_public_key;
client_public_key.nE = atoi(recv_E);
client_public_key.nN = atoll(recv_N);
//char encryKey[300];

// 接收加密的AES 密钥
char aes_key[20] = { '\0' };
char en_aes_key[300] = { '\0' };
recv(ClientSocket, en_aes_key, 300, 0); // 接收密钥
printf("S: 从客户端接收到加密的AES 密钥:%s\n", en_aes_key);
int len_of_en_aes_key = 0;
for (int i = 0; i < 8; i++)
{
    char en_sixteen_bits_to_char[20] = { '\0' };
    int p = 0;
    while (en_aes_key[len_of_en_aes_key++] != ',')
        en_sixteen_bits_to_char[p++] = en_aes_key[len_of_en_aes_key - 1];
    UINT64 en_sixteen_bits = atoll(en_sixteen_bits_to_char);
    UINT64 sixteen_bits = Decry(en_sixteen_bits);
    aes_key[i * 2] = sixteen_bits >> 8;
    aes_key[i * 2 + 1] = sixteen_bits % 256;
}
printf("S: 解密的AES 密钥为:%s\n", aes_key);

```

```

// 准备收发消息
char plaintext[500], ciphtext[500] = { 0 };
while (true)
{
    string str_aes_key = charkey_to_strkey(aes_key);

    // 一 服务器接收加密信息
    // 1.接收对方发过来的字符串
    recv(ClientSocket, ciphtext, 500, 0);
    // 2.用自己的私钥(server_private_key)解密发过来的字符串
    int len_of_ciphtext = 0;
    for (int i = 0; i < strlen(ciphtext) / 4; i++)
    {
        if (en_aes_key[len_of_ciphtext++] == '.')
        {
            // 判断到加密后的hash 值了
            break;
        }
        char en_sixteen_bits_to_char[20] = { '\0' };
        int p = 0;
        while (en_aes_key[len_of_ciphtext++] != ',')
            en_sixteen_bits_to_char[p++] = en_aes_key[len_of_ciphtext - 1];
        UINT64 en_sixteen_bits = atoll(en_sixteen_bits_to_char);
        UINT64 sixteen_bits = Decry(en_sixteen_bits);
        ciphtext[i * 4] = (sixteen_bits >> 12) % 16;
        ciphtext[i * 4 + 1] = (sixteen_bits >> 8) % 16;
        ciphtext[i * 4 + 2] = (sixteen_bits >> 4) % 16;
        ciphtext[i * 4 + 3] = sixteen_bits % 16;
    }
    string mid_ciphtext = ciphtext;
    // 补充: 对hash 的解密 需要用服务器的公钥解密
    char recv_en_hash[100];
    // 先保存之前的秘钥 防止丢失
    UINT64 mid_n = m_cParament.n;
    UINT64 mid_d = m_cParament.d;
    UINT64 mid_e = m_cParament.e;
    // 赋值为客户端的公钥 (client_public_key)
    m_cParament.n = client_public_key.nN;
    m_cParament.d = client_public_key.nE;
    for (int i = 0; i < strlen(ciphtext) / 4; i++)
    {
        if (en_aes_key[len_of_ciphtext++] == '.')
        {
            // 判断到加密后的hash 值了
            break;
        }
        char en_sixteen_bits_to_char[20] = { '\0' };
        int p = 0;
        while (en_aes_key[len_of_ciphtext++] != ',')
            en_sixteen_bits_to_char[p++] = en_aes_key[len_of_ciphtext - 1];
        UINT64 en_sixteen_bits = atoll(en_sixteen_bits_to_char);
        UINT64 sixteen_bits = Decry(en_sixteen_bits);
        recv_en_hash[i * 4] = (sixteen_bits >> 12) % 16;
        recv_en_hash[i * 4 + 1] = (sixteen_bits >> 8) % 16;
        recv_en_hash[i * 4 + 2] = (sixteen_bits >> 4) % 16;
        recv_en_hash[i * 4 + 3] = sixteen_bits % 16;
    }
    // 恢复之前的缓冲区
    m_cParament.n = mid_n;
    m_cParament.d = mid_d;
    m_cParament.e = mid_e;
    string recv_hash = recv_en_hash;
    // 3. 将其用AES 解密(在解密函数中将其分为128 位一组的数据)各个组的128 位的数据 需要实现CBC 分组解密 即需
    // 要加上 xor
    string de_mid_ciphtext = de_cbc_aes(mid_ciphtext, str_aes_key);
    // 4. 输出解密后的明文(如果 hash 值验证相等的话)
    if (MD5(de_mid_ciphtext).getstring() == recv_hash)
    {
        char* time;
        strcpy(time, getTime());
    }
}

```



```

        printf("S: [%s]经过解密后的明文:", time);
        cout << de_mid_ciphtext;
        if (de_mid_ciphtext == "exit")
        {
            break;
        }
    }
else
{
    cout << "本次接收有误, 需要重新接收" << endl;
    continue;
}

// 二 服务器发送加密信息
// 补充: 先用哈希 (MD5) 进行加密 这个值再用私钥加密 后与整体进行一起加密
printf("S: 请输入明文:");
setbuf(stdin, NULL);
scanf("%[^\n]s", plaintext); // 使得空行代表读取完毕而不是空格
bool exit = false;
if (strcmp(plaintext, "exit") == 0) { exit = true; }
string mid_plain_text = plaintext;
// hash 也是 32 个 16 进制数的字符串
string hash = MD5(mid_plain_text).getstring();
PublicKey server_private_key;
server_private_key.nE = m_cParament.d;
server_private_key.nN = m_cParament.n;
// 用自己的私钥去加密
char en_hash[100] = { '\0' };
int len_of_en_text = 0;
for (int i = 0; i < hash.size() / 4; i++)
{
    int p1 = i * 4, p2 = i * 4 + 1;
    int p3 = i * 4 + 2, p4 = i * 4 + 3;
    int num1 = int(hash[p1]);
    int num2 = int(hash[p2]);
    int num3 = int(hash[p3]);
    int num4 = int(hash[p4]);
    UINT64 sixteen_bits = (num1 << 12) + (num2 << 8) + (num3 << 4) + num4;
    UINT64 en_sixteen_bits = Encry(sixteen_bits, server_private_key);
    char en_sixteen_bits_to_char[20];
    sprintf(en_sixteen_bits_to_char, "%lu", en_sixteen_bits);
    strncpy(en_hash + len_of_en_text, en_sixteen_bits_to_char, strlen(en_sixteen_bits_to_char));
    len_of_en_text += strlen(en_sixteen_bits_to_char);
}
en_hash[len_of_en_text] = '\0';
string wxn_en_hash = en_hash;
// 1. 输入信息 并将其用 AES 加密为 128 位的数据 需要实现 CBC 分组加密 即需要加上 xor 并将分组加密完的数据合成一个字符串
// 为了方便区分最后的 hash
mid_plain_text += ".";
mid_plain_text += wxn_en_hash;
string mid_en_text = en_cbc_aes(mid_plain_text, str_aes_key); // 返回的是一个 CBC 分组加密完成的字符串

// 2. 用对方的公钥(client_public_key) 进行加密 由于合成的字符串每一个字符是 4 位 所以需要 4 个为一组
char en_text[500] = { '\0' };
int len_of_en_text = 0;
for (int i = 0; i < mid_en_text.size() / 4; i++)
{
    int p1 = i * 4, p2 = i * 4 + 1;
    int p3 = i * 4 + 2, p4 = i * 4 + 3;
    int num1 = int(mid_en_text[p1]);
    int num2 = int(mid_en_text[p2]);
    int num3 = int(mid_en_text[p3]);
    int num4 = int(mid_en_text[p4]);
    UINT64 sixteen_bits = (num1 << 12) + (num2 << 8) + (num3 << 4) + num4;
    UINT64 en_sixteen_bits = Encry(sixteen_bits, client_public_key);
    char en_sixteen_bits_to_char[20];
    sprintf(en_sixteen_bits_to_char, "%lu", en_sixteen_bits);
    strncpy(en_text + len_of_en_text, en_sixteen_bits_to_char, strlen(en_sixteen_bits_to_char));
    len_of_en_text += strlen(en_sixteen_bits_to_char);
    char comma = ',';
    en_text[len_of_en_text] = comma;
}

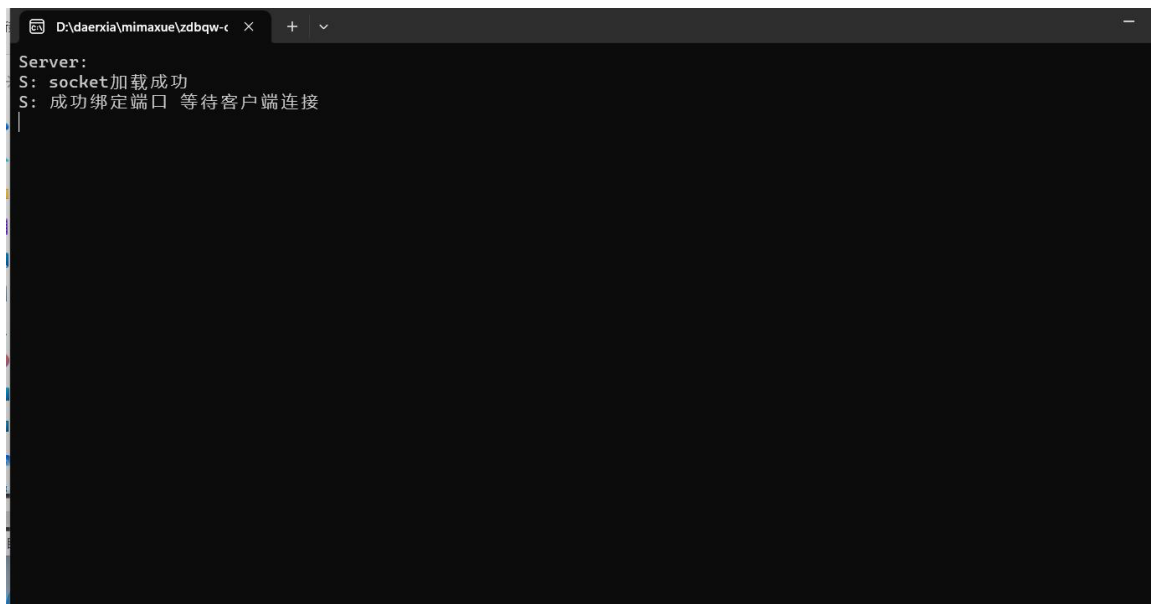
```

```

        len_of_en_text += 1;
    }
    en_text[len_of_en_text] = '\0';
    // 3. 将加密完的字符串发送
    send(ClientSocket, en_text, strlen(en_text), 0);
    if (exit)
    {
        break;
    }
}
printf("\nS: 退出...");

//关闭套接字
closesocket(ServerSocket);
WSACleanup();
return 0;
}

```



```

Server:
S: socket加载成功
S: 成功绑定端口 等待客户端连接

```

4. RSA 相关

4.1. 相关概念介绍

公钥密码体系的基本概念

传统对称密码体制要求通信双方使用相同的密钥，因此应用系统的安全性完全依赖于密钥的保密。针对对称密码体系的缺陷，Differ 和 Hellman 提出了新的密码体系—公钥密码体系，也称为非对称密码体系。在公钥加密系统中，加密和解密使用两把不同的密钥。加密的密钥（公钥）可以向公众公开，但是解密的密钥(私钥)必须是保密的，只有解密方知道。公钥密码体系要求算法要能够保证：任何企图获取私钥的人都无法从公钥中推算出来。

公钥密码体制中最著名算法是 RSA，以及背包密码、McEliece 密码、Diffe_Hellman、Rabin、零知识证明、椭圆曲线、ElGamal 算法等。

公钥密码体系的特点

公钥密码体制如下部分组成：

1. 明文：作为算法的输入的消息或者数据
2. 加密算法：加密算法对明文进行各种代换和变换
3. 密文：作为算法的输出，看起来完全随机而杂乱的数据，依赖明文和密钥。对于给定的消息，不同的密钥将产生不同的密文，密文是随机的数据流，并且其意义是无法理解的
4. 公钥和私钥：公钥和私钥成对出现，一个用来加密，另一个用来解密
5. 解密算法：该算法用来接收密文，解密还原出明文。

RSA 加密算法的基本工作原理

RSA 加密算法是一种典型的公钥加密算法。RSA 算法的可靠性建立在分解大整数的困难性上。假如找到一种快速分解大整数算法的话，那么用 RSA 算法的安全性会极度下降。但是存在此类算法的可能性很小。目前只有使用短密钥进行加密的 RSA 加密结果才可能被穷举解破。只要其密钥的长度足够长，用 RSA 加密的信息的安全性就可以保证。

python 代码部分：

```
# 求最大公约数

def gcd(a, b):

    if a < b:

        return gcd(b, a)

    elif a % b == 0:

        return b

    else:

        return gcd(b, a % b)

# 快速幂+取模

def power(a, b, c):

    ans = 1

    while b != 0:

        if b & 1:

            ans = (ans * a) % c

        b >>= 1

        a = (a * a) % c

    return ans

# 快速幂

def quick_power(a: int, b: int) -> int:

    ans = 1
```

```

while b != 0:

    if b & 1:

        ans = ans * a

    b >>= 1

    a = a * a

return ans

# 大素数检测

def Miller_Rabin(n):

    a = random.randint(2, n - 2) # 随机第选取一个  $a \in [2, n-2]$ 

    # print("随机选取的 a=%lld\n"%a)

    s = 0 # s 为 d 中的因子 2 的幂次数。

    d = n - 1

    while (d & 1) == 0: # 将 d 中因子 2 全部提取出来。

        s += 1

        d >>= 1

    x = power(a, d, n)

    for i in range(s): # 进行 s 次二次探测

        newX = power(x, 2, n)

        if newX == 1 and x != 1 and x != n - 1:

            return False # 用二次定理的逆否命题，此时 n 确定为合数。

        x = newX

    if x != 1: # 用费马小定理的逆否命题判断，此时  $x = a^{(n-1)} \pmod n$ ，那么 n 确定为合数。

        return False

    return True # 用费马小定理的逆命题判断。能经受住考验至此的数，大概率为素数。

# 卢卡斯-莱墨素性检验

def Lucas_Lehmer(num: int) -> bool: # 快速检验  $2^m - 1$  是不是素数

    if num == 2:

        return True

    if num % 2 == 0:

        return False

    s = 4

    Mersenne = pow(2, num) - 1 #  $2^m - 1$  是梅森数

    for x in range(1, (num - 2) + 1): # num-2 是循环次数，+1 表示右区间开

        s = ((s * s) - 2) % Mersenne

    if s == 0:

        return True

```

```

else:

    return False

# 扩展的欧几里得算法,  $ab=1 \pmod m$ , 得到  $a$  在模  $m$  下的乘法逆元  $b$ 

def Extended_Eulid(a: int, m: int) -> int:

    def extended_eulid(a: int, m: int):

        if a == 0: # 边界条件

            return 1, 0, m

        else:

            x, y, gcd = extended_eulid(m % a, a) # 递归

            x, y = y, (x - (m // a) * y) # 递推关系, 左端为上层

            return x, y, gcd # 返回第一层的计算结果。

        # 最终返回的  $y$  值即为  $b$  在模  $a$  下的乘法逆元

        # 若  $y$  为复数, 则  $y+a$  为相应的正数逆元

    n = extended_eulid(a, m)

    if n[1] < 0:

        return n[1] + m

    else:

        return n[1]

# 按照需要的 bit 来生成大素数

def Generate_prime(key_size: int) -> int:

    while True:

        num = random.randrange(quick_power(2, key_size - 1), quick_power(2, key_size))

        if Miller_Rabin(num):

            return num

# 生成公钥和私钥

def KeyGen(p: int, q: int):

    e = random.randint(1, (p - 1) * (q - 1))

    while gcd(e, (p - 1) * (q - 1)) != 1:

        e = random.randint(1, (p - 1) * (q - 1))

    n = p * q

    d = Extended_Eulid(e, (p - 1) * (q - 1))

    return n, e, d

def Encrypt(message: int, e: int, n: int) -> int:

    ciphertext = power(message, e, n)

    return ciphertext

def Decrypt(ciphertext: int, d: int, n: int) -> int:

```

```
plaintext = power(ciphertext, d, n)
return plaintext
```

4.2. 重要函数分析

大素数生成（512 位）原理：

1. 线性同余算法生成随机数

- 该算法产生的是伪随机数，只具有统计意义上的随机性，**易被攻破**，不宜在现实情况中使用
- 参数：
 - 模数： $m(m > 0)$ ，为使随机数的周期尽可能大， m 应尽量大，本次实验取 $m = 2^{31} - 1$
 - 乘数： $a(0 \leq a < m)$ ，是 m 的原根，*eg.* $a = 7^5 = 16807$
 - 增量： c （本次实验中取 0）
 - 初值种子： X_0 ，随机选取一 32 位整数，
- 随机数序列： $X_{n+1} = (aX_n + c) \bmod n$

2. Rabin-Miller 素数概率检测算法

- 定理 1：如果 p 为大于 2 的素数，则方程 $x^2 \equiv 1 \pmod{p}$ 的解只有 $x \equiv 1$ 和 $x \equiv -1$
- 定理 2：若 n 为素数，则 $a^{n-1} \equiv 1 \pmod{n}$
- 每轮检测的伪代码：

```
witness(a,n){
    d=1; //d 初值为1
    for i=k downto 0 do{
        x=d;
        d=(d^2) % n;
        if(d==1&&x!=1&&x!=n-1) //若为素数，x 不可能不是1或n-1
            return FALSE;
        if (n-1 的 2^i 位为1) //
            d=(d*a) % n;
    }
    if(d!=1) return FALSE; //定理2
    return TRUE;
}
```

该算法为概率性检测，若进行 s 轮检测，则是素数的概率至少为 $1 - 2^{-s}$

3. Eratosthenes 素数筛选法

程序框图：

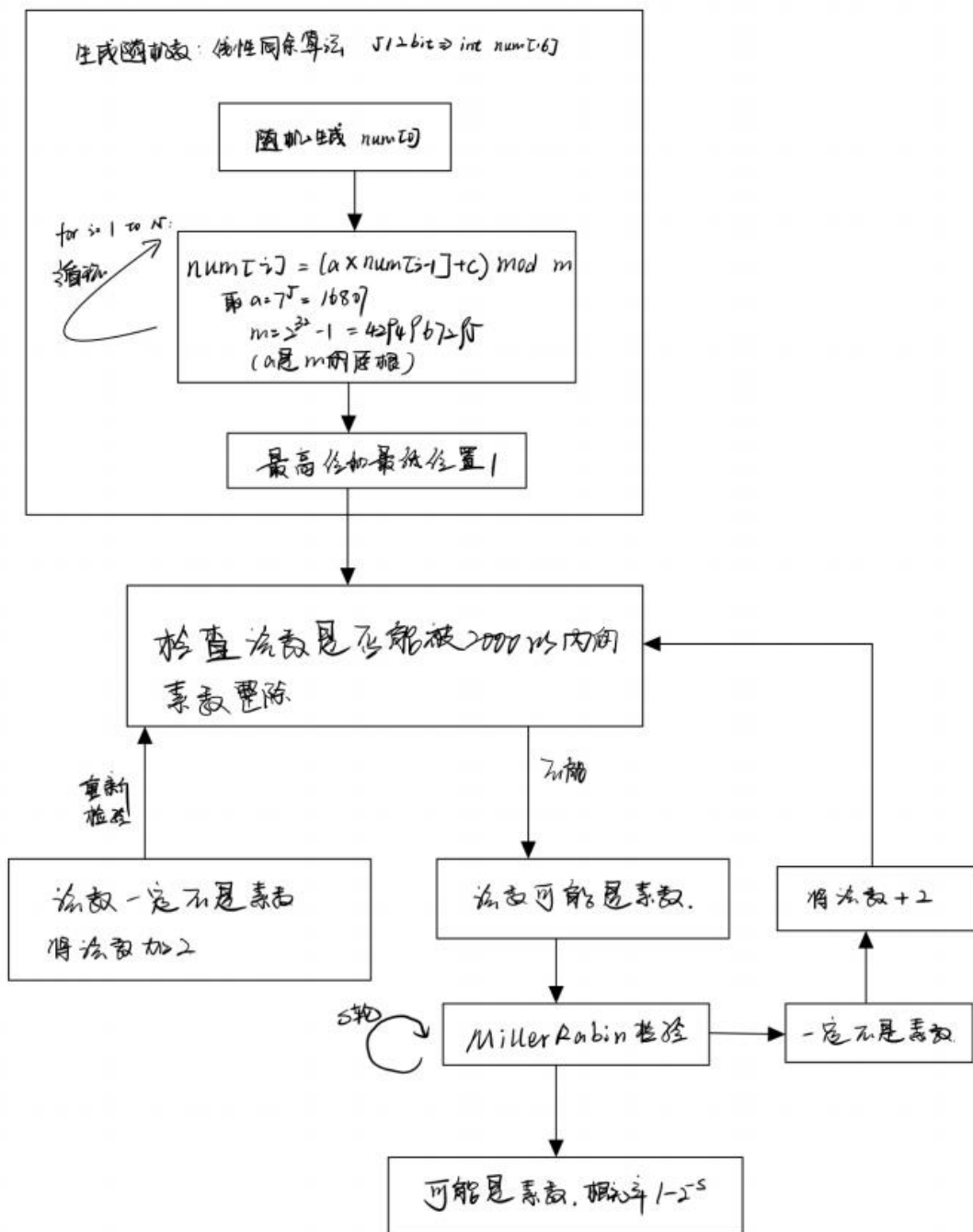


image-20210508195843864

构建 n 的长度为 1024 比特的 RSA 算法，并利用该算法实现对明文的加密和解密。

程序框图

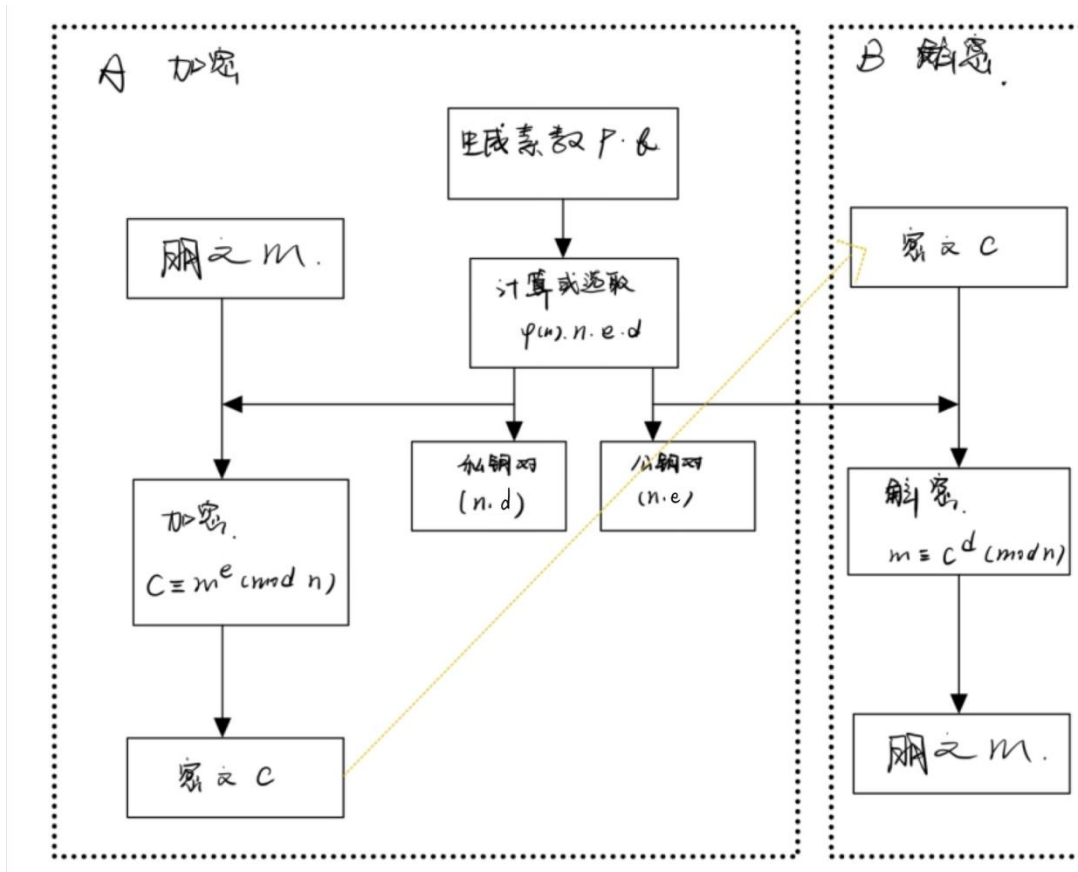


image-20210508200351702

1. 生成密钥对

```

RSA_::RSA_(big p, big q, big e)
{
    big v1;
    v1.set(1);

    this->p = p;
    this->q = q;
    this->e = e;

    n = mul(p, q); // n=pq
    big p1, q1;
    p1 = sub(p, v1); // p-1
    q1 = sub(q, v1); // q-1

    phi = mul(p1, q1); // phi=(p-1)(q-1)
    d = getinv(phi, e); // 逆元
}

```

2. 加密

```

RSAen_::RSAen_(RSA_ a, big m)
{
    this->n = a.n;
    this->e = a.e;
    this->m = m;

    c = pow(m, e, n); // m^e mod n
}

```

3. 解密

```
RSAdede::RSAdede(RSA_ a, big c)
{
    this->n = a.n;
    this->d = a.d;
    this->c = c;

    m = pow(c, d, n); // c^d mod n
}
```

5. AES 相关，作为发送给服务器端的加密信息

5.1. 流程分析

密钥扩展

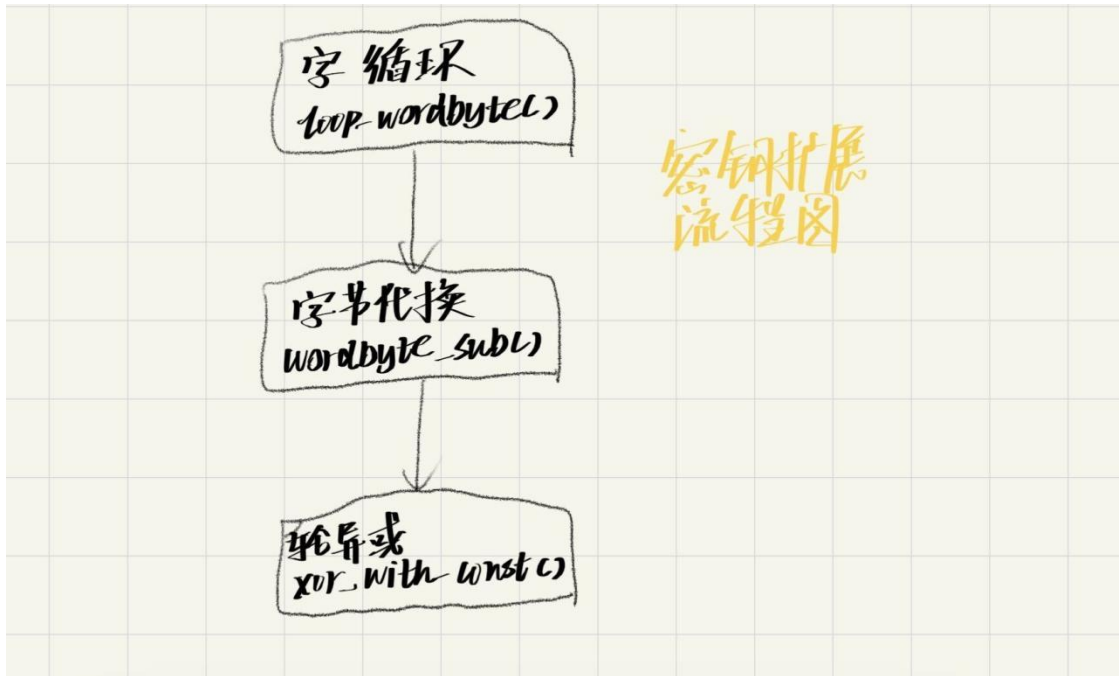
AES 算法通过密钥扩展程序（Key Expansion）将用户输入的密钥 K 扩展生成 $Nb(Nr+1)$ 个字，存放在一个线性数组 $w[Nb*(Nr+1)]$ 中。具体如下：

1. 位置变换函数 `loop_wordbyte()`，接受一个字 $[a0, a1, a2, a3]$ 作为输入，循环左移一个字节后输出 $[a1, a2, a3, a0]$ 。
2. S 盒变换函数 `wordbyte_sub()`，接受一个字 $[a0, a1, a2, a3]$ 作为输入。S 盒是一个 16×16 的表，其中每一个元素是一个字节。对于输入的每一个字节，前四位组成十六进制数 x 作为行号，后四位组成的十六进制数 y 作为列号，查找表中对应的值。最后函数输出 4 个新字节组成的 32-bit 字。（S 盒和逆 S 盒在程序中已经提前声明了一个结构体）
3. 轮常数 `Rcon[]`，如何计算的就不说了，直接把它当做常量数组。
4. 扩展密钥数组 $w[]$ 的前 Nk 个元素就是外部密钥 K ，以后的元素 $w[i]$ 等于它前一个元素 $w[i-1]$ 与前第 Nk 个元素 $w[i-Nk]$ 的异或，即 $w[i] = w[i-1] \text{ XOR } w[i-Nk]$ ；但若 i 为 Nk 的倍数，则 $w[i] = w[i-Nk] \text{ XOR } \text{wordbyte_sub}(\text{loop_wordbyte}(w[i-1])) \text{ XOR } \text{Rcon}[i/Nk-1]$ 。

在本次的实验中主要采用的是字符型的变量来进行参数的传递和转移，但是在计算的过程中会将字符型的数组转变为数字，但由于 128 位的数字太大，所以将其分为 4 组，每一个都由四个 8 进制数来构成，如下所示：

```
vector<string> group_key(string& key)
{
    // 四组
    vector<string> groups(4);
    // 初始下标
    int index = 0;
    // 分组
    for (string& g : groups)
    {
        g = key.substr(index, 8);
        index += 8;
    }
    return groups;
}
```

在密钥扩展中，当下标为 4 的倍数的时候，进行的过程较为复杂，此部分的流程图如下所示：



AES 加密

AES 加密总共大体上可分为 4 个部分来构造，分别为 S 盒变换、行变换、列变换以及与扩展密钥的异或，各个部分的大致内容如下：

S 盒变换-wordbyte_sub()

在密钥扩展部分已经讲过了，S 盒是一个 16 行 16 列的表，表中每个元素都是一个字节。S 盒变换很简单：函数 `wordbyte_sub()` 接受一个 4x4 的字节矩阵作为输入，对其中的每个字节，前四位组成十六进制数 x 作为行号，后四位组成的十六进制数 y 作为列号，查找表中对应的值替换原来位置上的字节。

行变换-move_row()

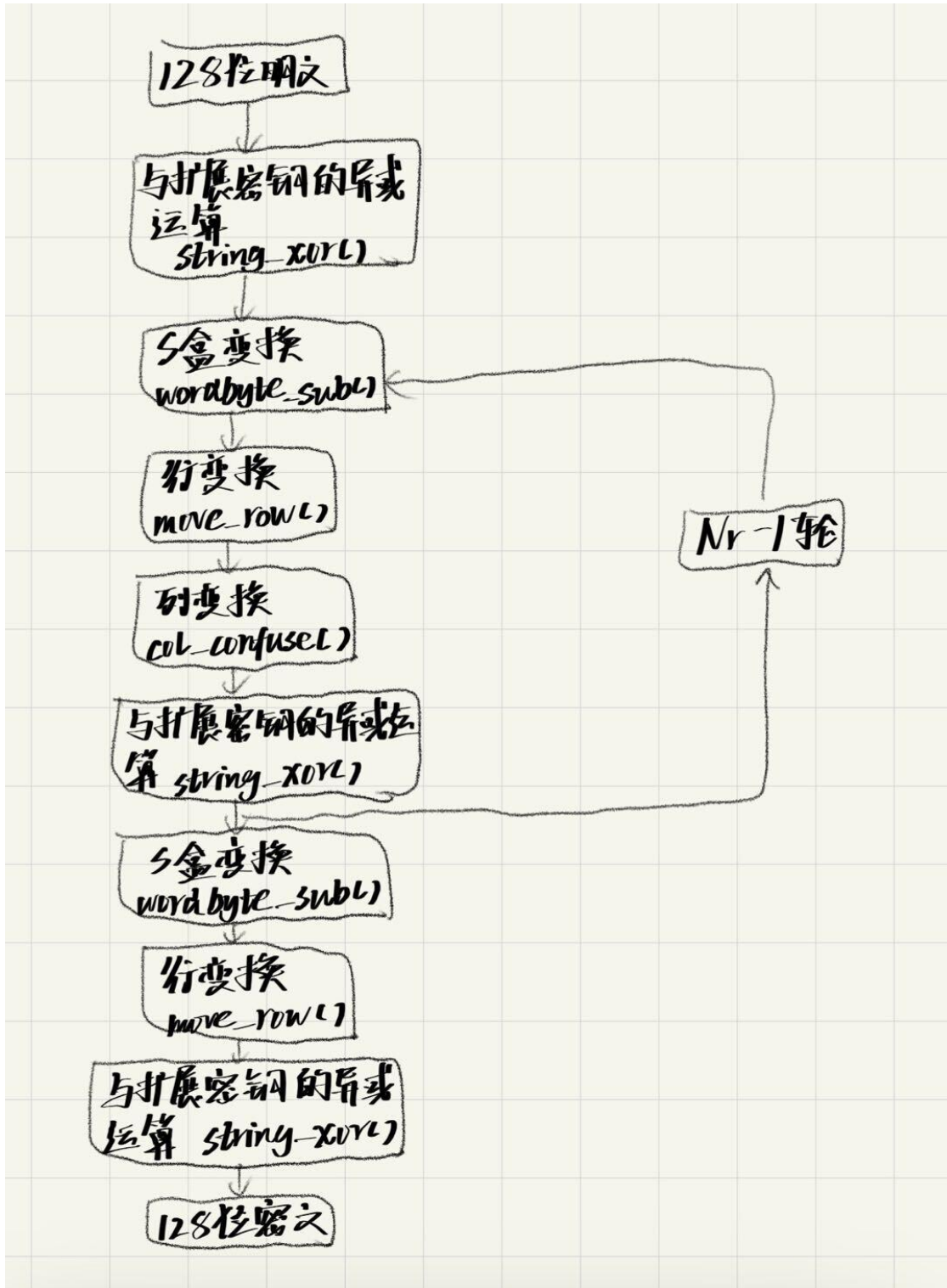
行变换也很简单，它仅仅是将矩阵的每一行以字节为单位循环移位：第一行不变，第二行左移一位，第三行左移两位，第四行左移三位。

列变换-col_confuse()

函数 `col_confuse()` 同样接受一个 4x4 的字节矩阵作为输入，并对矩阵进行逐列变换，注意公式中用到的乘法是伽罗华域（GF，有限域）上的乘法。

与扩展密钥的异或-string_xor()

扩展密钥只参与了这一步。根据当前加密的轮数，用 `w[]` 中的 4 个扩展密钥与矩阵的 4 个列进行按位异或。到这里 AES 加密的各个部分差不多了。



流程图

最后 AES 加密的流程图如下图所示：

AES 解密

AES 解密与 AES 加密类似，基本上都是其加密的逆过程，总共大体上也可分为 4 个部分来构造，分别为逆行变换、逆 S 盒变换、逆列变换以及与扩展密钥的异或，各个部分的大致内容如下（由于与扩展密钥的异或和 AES 加密的部分一样，在这里就不再过多赘述）：

逆行变换-

`in_move_row()`

上面讲到 `move_row()` 是对矩阵的每一行进行循环左移，所以 `in_move_row()` 是对矩阵每一行进行循环右移。

逆 S 盒变换-

`in_wordbyte_sub()`

与 S 盒变换一样，也是查表，查表的方式也一样，只不过查的是另外一个置换表（S-Box 的逆表）。

逆列变换-

`in_col_confuse()`

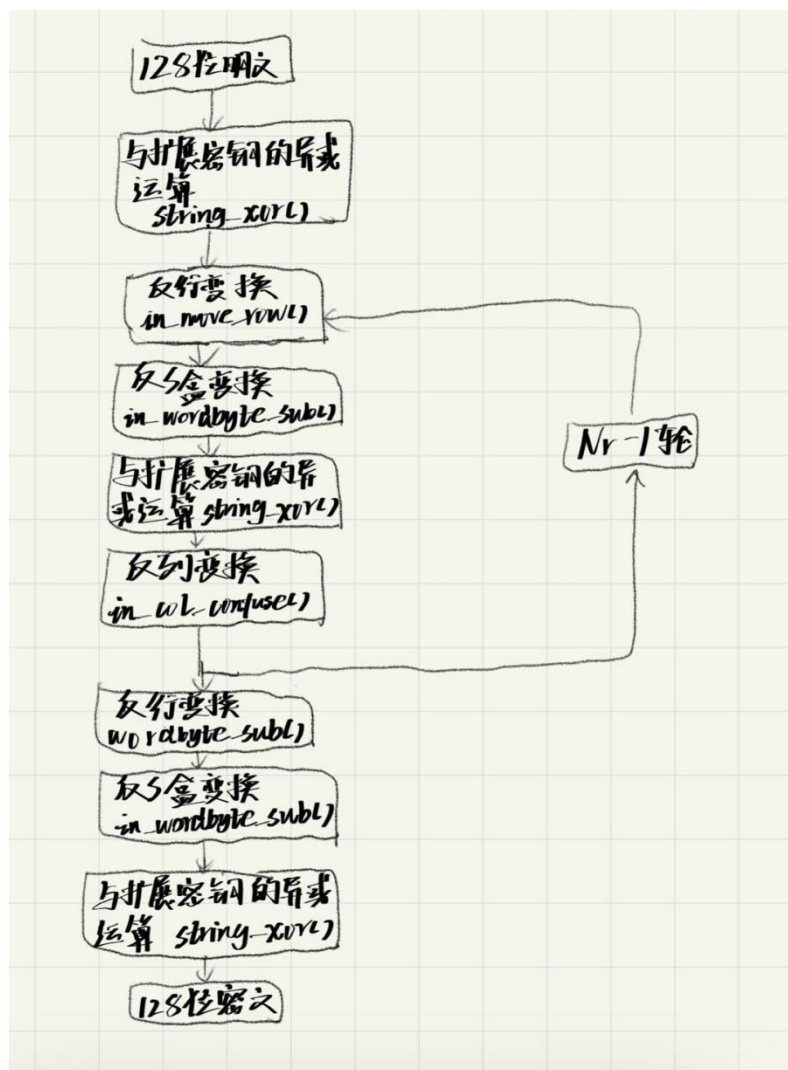
与列变换的方式一样，只不过计算公式的系数矩阵发生了变化，只要写出三个逆变换的函数，然后根据伪代码就很容易实现 AES 解密算法了。

流程图

最后 AES 解密的流程图如下图所示：

5.2. 代码实现

python 当中由于已经存在相应的 AES 代码包，所以可以调用相关库就能直接使用



```

import base64
1 个用法
class FileAES:
    def __init__(self, key):
        self.key = key #将密钥转换为字符型数据
        self.mode = AES.MODE_CBC #操作模式选择CBC

3 用法
    def encrypt(self, text):
        """加密函数"""
        file_aes = AES.new(self.key, self.mode) #创建AES加密对象
        text = text.encode('utf-8') #明文必须编码成字节流数据, 即数据类型为bytes
        while len(text) % 16 != 0: # 对字节型数据进行长度判断
            text += b'\x00' # 如果字节型数据长度不是16倍整数就进行补充
        en_text = file_aes.encrypt(text) #明文进行加密, 返回加密后的字节流数据
        return str(base64.b64encode(en_text).encoding='utf-8') #将加密后得到的字节流数据进行base

    def decrypt(self, text):
        """解密函数"""
        file_aes = AES.new(self.key, self.mode)
        text = bytes(text, encoding='utf-8') #将密文转换为bytes, 此时的密文还是由base64编码过的
        text = base64.b64decode(text) #对密文再进行base64解码
        de_text = file_aes.decrypt(text) #密文进行解密, 返回明文的bytes
        return str(de_text, encoding='utf-8').strip() #将解密后得到的bytes型数据转换为str型, 并去除

```

各类的结构体（即书本上出现的置换表以及各种扩展和置换运算）已经预先定义

各类预处理函数

- 首先是对字节循环的实现

```

string loop_wordbyte(string& wi_1)
{
    string ans = wi_1.substr(2) + wi_1.substr(0, 2);
    return ans;
}

```

- 接下来是对字节代换的实现

```

string wordbyte_sub(string& wi_1)
{
    int len = wi_1.length();
    string ans = "";
    for (int i = 0; i < len; i += 2)
    {
        // 先获取当前的下标
        int x = ch_to_int(wi_1[i]), y = ch_to_int(wi_1[i + 1]);
        // 然后获取当前的数字
        int num = S[x][y];
        // 先将数值转化为字符串
        string s = int_to_chs(num);
        // 然后不足的话补0
        while (s.length() < 2)
        {
            s = "0" + s;
        }

        // 加起来
        ans += s;
    }
    return ans;
}

```

- 然后是对密钥的轮常量异或的实现

```
string xor_with_const(string& wi_1, int rounds)
{
    // 先将字符串变为数字
    long long num = 0;
    for (int i = 0; i < 8; ++i)
    {
        char ch = wi_1[i];
        num = num * 16 + ch_to_int(ch);
    }
    // 计算异或结果
    num ^= Rcon[rounds];

    // 将num转化为字符串
    string res = int_to_chs(num);
    while (res.length() < 8)
    {
        res = "0" + res;
    }
    return res;
}
```

- 密钥拓展的时候，下标为4的倍数时，需要使用一个特殊的变换函数

```
string T(string& wi_1, int round)
{
    // T变换由3部分构成，用的即为上述描述的三个函数
    // 先进行字循环
    string ans = loop_wordbyte(wi_1);
    // 然后字节代换
    ans = wordbyte_sub(ans);
    // 最后是轮异或
    ans = xor_with_const(ans, round);

    return ans;
}
```

- 密钥编排的实现

```
vector<string> extend_key(string& key)
{
    // 先分组
    vector<string> w_key = group_key(key);
    for (int i = 0; i < 40; ++i)
    {
        string w = "";
        int index = 4 + i;
        string temp = w_key[index - 1];
        // 4的倍数的时候，需要调用T函数
        if (index % 4 == 0)
        {
            temp = T(temp, index / 4 - 1);
        }
        w = string_xor(temp, w_key[index - 4]);

        // 压入数组中
        w_key.push_back(w);
    }

    return w_key;
}
```

- 行移位函数的实现

```
vector<string> move_row(vector<string>& s)
{
    vector<string> ans = s;
    // 字符串数组每个对应一列，所以是对应到列进行移位
    // 一行对应有两个16进制数，所以需要两个一起移动，相当于两列一起移动
    for (int i = 0; i < 4; ++i)
    {
        int k = i * 2;
        // 就原本矩阵对应的行移位，对于字符串数组就是列移位
    }
}
```



```

        for (int j = 0; j < 4; ++j)
        {
            ans[j][k] = s[(j + i) % 4][k];
            ans[j][k + 1] = s[(j + i) % 4][k + 1];
        }
    }
    return ans;
}

```

- 列混淆函数的实现

```

vector<string> col_confuse(vector<string>& s)
{
    vector<string> ans = s;
    // 算法中对应的是列，这边就直接变成了行，即字符
    for (int i = 0; i < 4; ++i)
    {
        // 需要先将字符串拆分成两两一组，共4组
        auto temp = split_s(s[i]);
        // 先转成数字
        int s0 = str_long(temp[0]), s1 = str_long(temp[1]), s2 = str_long(temp[2]),
            s3 = str_long(temp[3]);
        // 计算混淆后的值
        int t0 = power(s0) ^ power(s1) ^ s1 ^ s2 ^ s3;
        int t1 = s0 ^ power(s1) ^ power(s2) ^ s2 ^ s3;
        int t2 = s0 ^ s1 ^ power(s2) ^ s3 ^ power(s3);
        int t3 = s0 ^ power(s0) ^ s1 ^ s2 ^ power(s3);
        // 转换成字符串再相加
        ans[i] = int_ch2(t0) + int_ch2(t1) + int_ch2(t2) + int_ch2(t3);
    }
    return ans;
}

```

- 行移位的逆操作函数的实现

```

vector<string> in_move_row(vector<string>& s)
{
    vector<string> ans = s;
    // 现在变成了逆操作
    for (int i = 0; i < 4; ++i)
    {
        int k = i * 2;
        // 就原本矩阵对应的行移位，对于字符串数组就是列移位
        for (int j = 0; j < 4; ++j)
        {
            ans[j][k] = s[(j - i + 4) % 4][k];
            ans[j][k + 1] = s[(j - i + 4) % 4][k + 1];
        }
    }
    return ans;
}

```

- 逆字节代换的实现

```

string in_wordbyte_sub(string& wi_1)
{
    int len = wi_1.length();
    string ans = "";
    for (int i = 0; i < len; i += 2)
    {
        // 先获取当前的下标
        int x = ch_to_int(wi_1[i]), y = ch_to_int(wi_1[i + 1]);
        // 然后获取当前的数字
        int num = S1[x][y];
        // 先将数值转化为字符串
        string s = int_to_chs(num);
        // 然后不足的话补0
        while (s.length() < 2)
        {
            s = "0" + s;
        }

        // 加起来
        ans += s;
    }
}

```

```

    }
    return ans;
}

```

- 列混淆函数的逆变换实现

```

vector<string> in_col_confuse(vector<string>& s)
{
    // 逆变换其实原来的变换矩阵的逆矩阵，对应0xe, 0xb, 0xd, 0x9
    // 4 列

    vector<string> ans = s;
    for (int i = 0; i < 4; ++i)
    {
        // 先分割成4个两位数字
        auto temp = split_s(s[i]);
        // 转换成数字
        vector<int> nums(4);
        for (int j = 0; j < 4; ++j)
        {
            nums[j] = str_long(temp[j]);
        }
        vector<int> t4(4, 0);
        for (int j = 0; j < 4; ++j)
        {
            for (int t = 0; t < 4; ++t)
            {
                int k = (t - j + 4) % 4;
                t4[j] ^= power(power(power(nums[t]))); // 表示8
                switch (k)
                {
                    case 0: // 0xe = 8 + 4 + 2
                    {
                        t4[j] ^= power(power(nums[t])) ^ power(nums[t]);
                        break;
                    }
                    case 1: // 0xb = 8 + 2 + 1
                    {
                        t4[j] ^= power(nums[t]) ^ nums[t];
                        break;
                    }
                    case 2: // 0xd = 8 + 4 + 1
                    {
                        t4[j] ^= power(power(nums[t])) ^ nums[t];
                        break;
                    }
                    default: // 0x9 = 8 + 1
                        t4[j] ^= nums[t];
                        break;
                }
            }
        }
        // 将数字转换成字符串存储
        ans[i] = int_ch2(t4[0]) + int_ch2(t4[1]) + int_ch2(t4[2]) + int_ch2(t4[3]);
    }

    return ans;
}

```

加密过程

- 按流程图进行实现

```

vector<string> aes(string& plain_text, string& key)
{
    // 先拓展密钥
    vector<string> keys = extend_key(key);

    int index = 0;
    // 然后就是10轮迭代
    // 需要知道明文其实是32位，所以需要搞4下
    // 可以先把明文也分组

```

```

// 一开始的先进行一次轮密钥加
vector<string> texts = group_key(plain_text);
for (int i = 0; i < 4; ++i)
{
    texts[i] = string_xor(texts[i], keys[i]);
}
index += 4;

// 然后十次迭代
for (int k = 0; k < 10; ++k)
{
    for (int j = 0; j < 4; ++j)
    {
        // 先是字节代换
        texts[j] = wordbyte_sub(texts[j]);
    }
    // 然后是行移位
    texts = move_row(texts);

    if (k < 9)
    {
        // 再来列混淆
        texts = col_confuse(texts);
    }

    // 轮密钥加
    for (int i = 0; i < 4; ++i)
    {
        texts[i] = string_xor(texts[i], keys[i + index]);
    }

    index += 4;
}

// 最后进行输出
// show(texts);
return texts;
}

```

解密过程

- 按流程图进行实现

```

vector<string> in_aes(string& text, string& key)
{
    // 先拓展密钥
    auto keys = extend_key(key);

    // 初始下标
    int index = 40;

    // 对密文分组
    vector<string> texts = group_key(text);

    // 一开始先进行依次轮密钥加
    for (int i = 0; i < 4; ++i)
    {
        texts[i] = string_xor(texts[i], keys[index + i]);
    }
    index -= 4;

    // 然后十次迭代
    for (int i = 0; i < 10; ++i)
    {
        // 先逆行移位
        texts = in_move_row(texts);

        // 然后是字节代换逆操作
        for (int j = 0; j < 4; ++j)
        {
            texts[j] = in_wordbyte_sub(texts[j]);
        }
    }
}

```

```

// 轮密钥加
for (int j = 0; j < 4; ++j)
{
    texts[j] = string_xor(texts[j], keys[index + j]);
}

// 除了最后一轮，都要列混淆逆变换
if (i < 9)
{
    texts = in_col_confuse(texts);
}
index -= 4;
}

// show(texts);

return texts;
}

```

5.3.ECB 模式的补充

(1)ECB 概念

ECB 模式全称是 Electronic CodeBook 模式，在 ECB 模式中，将明文分组加密之后的结果将直接成为密文分组。

ECB 模式的特点

ECB 模式中，明文分组与密文分组是一一对应的关系，因此，如果明文中存在多个相同的明文分组，则这些明文分组最终都将被转换为相同的密文分组。这样一来，只要观察一下密文，就可以知道明文存在怎样的重复组合，并可以以此为线索来破译密码，因此 ECB 模式是存在一定风险的。

python:

```

import base64

class FileAES:

    def __init__(self,key):

        self.key = key #将密钥转换为字符型数据

        self.mode = AES.MODE_ECB #操作模式选择 ECB

    def encrypt(self,text):

        """加密函数"""

        file_aes = AES.new(self.key,self.mode) #创建 AES 加密对象

        text = text.encode('utf-8') #明文必须编码成字节流数据，即数据类型为 bytes

        while len(text) % 16 != 0: # 对字节型数据进行长度判断

            text += b'\x00' # 如果字节型数据长度不是 16 倍整数就进行补充

        en_text = file_aes.encrypt(text) #明文进行加密，返回加密后的字节流数据

        return str(base64.b64encode(en_text),encoding='utf-8') #将加密后得到的字节流数据进行 base64 编码并再转换为 unicode 类型

    def decrypt(self,text):

        """解密函数"""

```

```

file_aes = AES.new(self.key,self.mode)

text = bytes(text,encoding='utf-8') #将密文转换为 bytes，此时的密文还是由 basen64 编码过的

text = base64.b64decode(text) #对密文再进行 base64 解码

de_text = file_aes.decrypt(text) #密文进行解密，返回明文的 bytes

return str(de_text,encoding='utf-8').strip() #将解密后得到的 bytes 型数据转换为 str 型，并去除末尾的填充

```

(2)C++

在 C++ 程序的编写过程当中，我采用了 CBC 模式，ECB 模式只进行了加密，而 CBC 模式则在加密之前进行了一次 XOR。

加密过程：

- 在 CBC 模式中，无法单独对一个中间的明文分组进行加密。例如，如果要生成密文分组 3，则至少需要凑齐明文分组 1、2、3 才行。

解密过程：

- 假设 CBC 模式加密的密文分组中有一个分组损坏了。在这种情况下，只要密文分组的长度没有发生变化，则解密时最多只有 2 个分组受到数据损坏的影响。
- 假设 CBC 模式的密文分组中有一些比特缺失了，那么此时即便只缺失 1 比特，也会导致密文分组的长度发生变化，此后的分组发生错位，这样一来，缺失比特的位置之后的密文分组也就全部无法解密。

对 CBC 模式的攻击

攻击者无需破译密码就能操纵明文。

下面举一个例子说明：

假设分组长度为 128bit（16 个字节），某银行的转账请求数据由以下 3 个分组构成。

分组 1=付款人的银行账号

分组 2=收款人的银行账号

分组 3=转账金额。

场景是：从 A-5374 账号向 B-6671 账号转账 1 亿元

16 进制数据表示如下：

```

明文分组 1 = 41 2D 35 33 37 34 20 20 20 20 20 20 20 20 20 20 (付款人: A-5374)
明文分组 2 = 42 2D 36 36 37 31 20 20 20 20 20 20 20 20 20 20 (收款人: B-6671)
明文分组 3 = 31 30 30 30 30 30 30 30 30 30 20 20 20 20 20 20 (转账金额: 100000000)

```

将上面数据用 ECB 加密，加密后，看不出明文分组内容。

密文分组 1 = 59 7D DE CC EF EC BA 9B BF 83 99 CF 60 D2 59 B9 (付款人: ????)
密文分组 2 = DF 49 2A 1C 14 8E 18 B6 53 1F 38 BD 5A A9 D7 D7 (收款人: ????)
密文分组 3 = CD AF D5 9E 39 FE FD 6D 64 8B CC CB 52 56 8D 79 (转账金额: ????)

攻击者将密文分组 1 和 2 对调。

密文分组 1 = DF 49 2A 1C 14 8E 18 B6 53 1F 38 BD 5A A9 D7 D7 (付款人: ????)
密文分组 2 = 59 7D DE CC EF EC BA 9B BF 83 99 CF 60 D2 59 B9 (收款人: ????)
密文分组 3 = CD AF D5 9E 39 FE FD 6D 64 8B CC CB 52 56 8D 79 (转账金额: ????)

攻击者没有试图破译密码，但场景却发生了变化：

明文分组 1 = 42 2D 36 36 37 31 20 20 20 20 20 20 20 20 20 (付款人: B-6671)
明文分组 2 = 41 2D 35 33 37 34 20 20 20 20 20 20 20 20 20 (收款人: A-5374)
明文分组 3 = 31 30 30 30 30 30 30 30 30 20 20 20 20 20 20 (转账金额: 100000000)

现在场景变成了：B-6671 账号向 A-5374 账号转账 1 亿元。

完全相反，这就是 ECB 的弱点，不破译密文的情况下操纵明文。

假设主动攻击者的目的是通过修改密文来操纵解密后的明文。如果攻击者能够对初始化向量中的任意比特进行反转（将 1 变成 0，将 0 变成 1），则明文分组中相应的比特也会被反转。这是因为在 CBC 模式的解密过程中，第一个明文分组会和初始化向量进行 XOR 运算。见下图。

但是想对密文分组也进行同样的攻击就非常困难了。例如，如果攻击者将密文分组 1 中的某个比特进行反转，则明文分组 2 中相应比特也会被反转，然而这一比特的变化却对解密后的明文分组 1 中的多个比特造成了影响，也就是说，只让明文分 1 中所期望的特定比特发生变化是很困难的。

五 填充提示攻击

填充提示攻击是一种利用分组密码中填充部分来进行攻击的方法。在分组密码中，当明文长度不为分组长度的整数倍时，需要在最后一个分组中填充一些数据使其凑满一个分组长度。在填充提示攻击中，攻击者会反复发送一段密文，每次发送时都对填充数据进行少许改变。由于接收者（服务器）在无法正确解密时会返回一个错误消息，攻击者通过这一错误消息就可以获得一部分与明文相关的信息。这一攻击并不仅限于 CBC 模式，而是适用所有需要进行分组填充的模式。

2014 年对 SSL3.0 造成了重大影响 POODLE 攻击实际上就是一种填充提示攻击。

六 对初始化向量（IV）进行攻击

初始化向量（IV）必须使用不可预测的随机数。然而在 SSL/TLS 的 TLS1.0 版本协议中，IV 并没有使用不可预测的随机数，而是使用上一次 CBC 模式加密时的最后一个分组。为了防御攻击者对此进行攻击，TLS1.1 以上的版本中改为了必须显示传送 IV。

七 CBC 模式应用

确保互联网安全的通信协议之一 SSL/TLS，就是使用 CBC 模式来确保通信机密性的，如使用 CBC 模式三重 DES 的 3DES_EDE_CBC 以及 CBC 模式 256 比特 AES 的 AES_256_CBC 等。

加密

由于已经实现了 AES 的单个加密，所以只需要将其整合起来即可：

- AES 的 CBC 加密模式，默认 iv 是全零（这个称为初始化向量），由于是分组加密，所以下一组的 iv，就用前一组的加密的密文来充当，本次由于字符串用 16 进制表示，所以就是 32 个 4 位，iv 初始化为 32 个零
- 加密每一次循环首先是明文与 iv 异或，然后是进行加密得到密文，同时密文是下次加密的 iv，最后将本次的密文设置为下次加密的 iv

代码如下：

```
// 定义一个实现 CBC 分组加密的 AES 数据
string en_cbc_aes(string& plain_text, string& key)
{
    string en_plain_text;
    int text_len = plain_text.size() / 32 + 1;
    // 先实现分组
    vector<string> groups(text_len);
    // 初始下标
    int index = 0;
    // 分组
    for (string& g : groups)
    {
        g = plain_text.substr(index, 32);
        index += 32;
    }
    // CBC 模式进行计算
    // AES 的 CBC 加密模式，默认 iv 是全零（这个称为初始化向量），由于是分组加密，所以下一组的 iv，就用前一组的加密
    的密文来充当
    // 本次由于字符串用 16 进制表示，所以就是 32 个 4 位，iv 初始化为 32 个零
    string iv = "00000000000000000000000000000000";
    // 中间变量组
    vector<string> groups_wxn(text_len);
    // 最终变量组
    vector<string> groups_zyl(text_len);
    vector<string> texts;
    for (int w = 0; w < text_len; w++)
    {
        // 明文与 iv 异或
        groups_wxn[w] = string_xor(iv, groups[w]);
        // 进行加密得到密文，同时密文是下次加密的 iv
        texts = aes(groups_wxn[w], key);
        groups_zyl[w] = texts[0] + texts[1] + texts[2] + texts[3];
        // 本次的密文是下次加密的 iv
        iv = groups_zyl[w];
    }
    // 最后整合
    for (string& z : groups_zyl)
    {
        en_plain_text += z;
    }
    return en_plain_text;
}
```

解密

由于已经实现了 AES 的单个解密，所以只需要将其整合起来即可：

- AES 的 CBC 解密模式，默认 iv 是全零（这个称为初始化向量），由于是分组解密，所以下一组的 iv，就用前一组的解密的密文来充当，本次由于字符串用 16 进制表示，所以就是 32 个 4 位，iv 初始化为 32 个零
- 加密每一次循环首先是密文块解密，然后是 iv 异或得到明文，最后设置下次解密用到的 iv

代码如下：

```
// 定义一个实现 CBC 分组解密的 AES 数据
string de_cbc_aes(string& en_text, string& key)
{
    string de_text;
    int text_len = en_text.size() / 32 + 1;
    // 先实现分组
    vector<string> groups(text_len);
    // 初始下标
    int index = 0;
    // 分组
    for (string& x : groups)
    {
        x = en_text.substr(index, 32);
        index += 32;
    }
    // CBC 模式进行计算
    // AES 的 CBC 解密模式，默认 iv 是全零（这个称为初始化向量），由于是分组解密，所以下一组的 iv，用前一组密文来充
    当
    // 本次由于字符串用 16 进制表示，所以就是 32 个 4 位，iv 初始化为 32 个零，与加密的时候相同
    string iv = "00000000000000000000000000000000";
    // 中间变量组
    vector<string> groups_wxn(text_len);
    // 最终变量组
    vector<string> groups_zyl(text_len);
    vector<string> texts;
    for (int w = 0; w < text_len; w++)
    {
        // 密文块解密
        texts = in_aes(groups[w], key);
        groups_wxn[w] = texts[0] + texts[1] + texts[2] + texts[3];
        // 与 iv 异或得到明文
        groups_zyl[w] = string_xor(iv, groups_wxn[w]);
        // 设置下次解密用到的 iv
        iv = groups_wxn[w];
    }
    // 最后整合
    for (string& z : groups_zyl)
    {
        de_text += z;
    }
    return de_text;
}
```

6. MD5 相关

6.1. 概念介绍

MD5 消息摘要算法，属 Hash 算法一类。MD5 算法对输入任意长度的消息进行运行，产生一个 128 位的消息摘要(32 位的数字字母混合码)。

MD5 主要特点

不可逆，相同数据的 MD5 值肯定一样，不同数据的 MD5 值不一样

(一个 MD5 理论上的确是可能对应无数多个原文的，因为 MD5 是有限多个的而原文可以是无数多个。比如主流使用的 MD5 将任意长度的“字节串映射为一个 128bit 的大整数。也就是一共有 2^{128} 种可能，大概是 3.4×10^{38} ，这个数字是有限多个的，而但是世界上可以被用来加密的原文则会有无数的可能性)

MD5 的性质

1. 压缩性：任意长度的数据，算出的 MD5 值长度都是固定的(相当于超损压缩)
2. 容易计算：从原数据计算出 MD5 值很容易
3. 抗修改性：对原数据进行任何改动，哪怕只修改 1 个字节，所得到的 MD5 值都有很大区别
4. 弱抗碰撞：已知原数据和其 MD5 值，想找到一个具有相同 MD5 值的数据（即伪造数据）是非常困难的
5. 强抗碰撞：想找到两个不同的数据，使它们具有相同的 MD5 值，是非常困难的

MD5 用途

1. 防止被篡改：
 1. 比如发送一个电子文档，发送前，我先得到 MD5 的输出结果 a。然后在对方收到电子文档后，对方也得到一个 MD5 的输出结果 b。如果 a 与 b 一样就代表中途未被篡改
 2. 比如我提供文件下载，为了防止不法分子在安装程序中添加木马，我可以在网站上公布由安装文件得到的 MD5 输出结果
 3. SVN 在检测文件是否在 CheckOut 后被修改过，也是用到了 MD5.
2. 防止直接看到明文：现在很多网站在数据库存储用户的密码的时候都是存储用户密码的 MD5 值。这样就算不法分子得到数据库的用户密码的 MD5 值，也无法知道用户的密码。（比如在 UNIX 系统中用户的密码就是以 MD5（或其它类似的算法）经加密后存储在文件系统中。当用户登录的时候，系统把用户输入的密码计算成 MD5 值，然后再去和保存在文件系统中的 MD5 值进行比较，进而确定输入的密码是否正确。通过这样的步骤，系统在并不知道用户密码的明文的情况下就可以确定用户登录系统的合法性。这不但可以避免用户的密码被具有系统管理员权限的用户知道，而且还在一定程度上增加了密码被破解的难度。）
3. 防止抵赖（数字签名）：这需要一个第三方认证机构。例如 A 写了一个文件，认证机构对此文件用 MD5 算法产生摘要信息并做好记录。若以后 A 说这文件不是他写的，权威机构只需对此文件重新产生摘要信息，然后跟记录在册的摘要信息进行比对，相同的话，就证明是 A 写的了。这就是所谓的“数字签名”。

6.2. 重要函数分析

python 当中由于有 *hashlib* 库的存在，对于 MD5 的编写可以利用 *hashlib* 库快速实现

```
import hashlib

#加密
s='1'
m = hashlib.md5(s.encode())
print(m) #<md5 HASH object @ 0x029D7BD0>
# m = hashlib.sha224( s.encode() )
result = m.hexdigest() #获取加密后的结果
print(result) #c4ca4238a0b923820dcc509a6f75849b

#撞库 #加盐
salt='24dfw32R@#@#@#$'
password = input('password:')
password += salt
m = hashlib.md5(_password.encode())
result = m.hexdigest() #获取加密后的结果
print(result)

def md5(s,salt=''):
    new_s = str(s) + salt
    m = hashlib.md5(new_s.encode())
    return m.hexdigest()
```

而 C++ 语言进行编写的过程当中需要调用下列函数，在前面的实验当中我们也有所涉及，此处我们再详细说明一遍

(1)zip()函数

- 首先 zip 函数，如四个运算和循环左移等 F(),G(),H(),I(),leftshift(), 四轮函数计算等
- 然后把这些函数整合起来放入
- 在能够压缩处理之前，还需要对数据进行分组，采用 for 循环，将数据分为 32 位一组
- 然后再执行一些列计算操作

(2)encode()函数

- 加密函数将 int 数组转换为 char，即将十进制转换为十六进制

- 在转换时，使用小端序存放结果

(3)init()函数

1. 首先设定了一个 `count[2]` 数组，负责记录当前字符串位数
2. 获取当前已有的字节数 (`count[0]>>3` 再模 64)
3. 接下来用 `count[0]+len<<3`，即已有位数加上新增加的位数
4. 然后判断是否有溢出
5. 若有，则将高位 `count[1]+1`
6. 让 `count[1]` 获取高位的位数，即先让 `len` 右移 32 得到高位，再左移 3 位得到位数，即 `count[1]+=len>>29`
7. 将新加入字符长度与待填充长度进行比较，在第一次调用此函数时，等同于将输入的字符串字节数与 64 字节数即 512 比特进行比较
8. 若大于等于，则可以先将数据按 64 字节分组，先对这些 64 字节的组执行压缩函数，然后再对最后一组不足 64 字节的数据执行后续填充操作等
9. 最后函数执行 `memcpy(&buffer[nowlength], &input[i], len - i)`，进行拷贝。

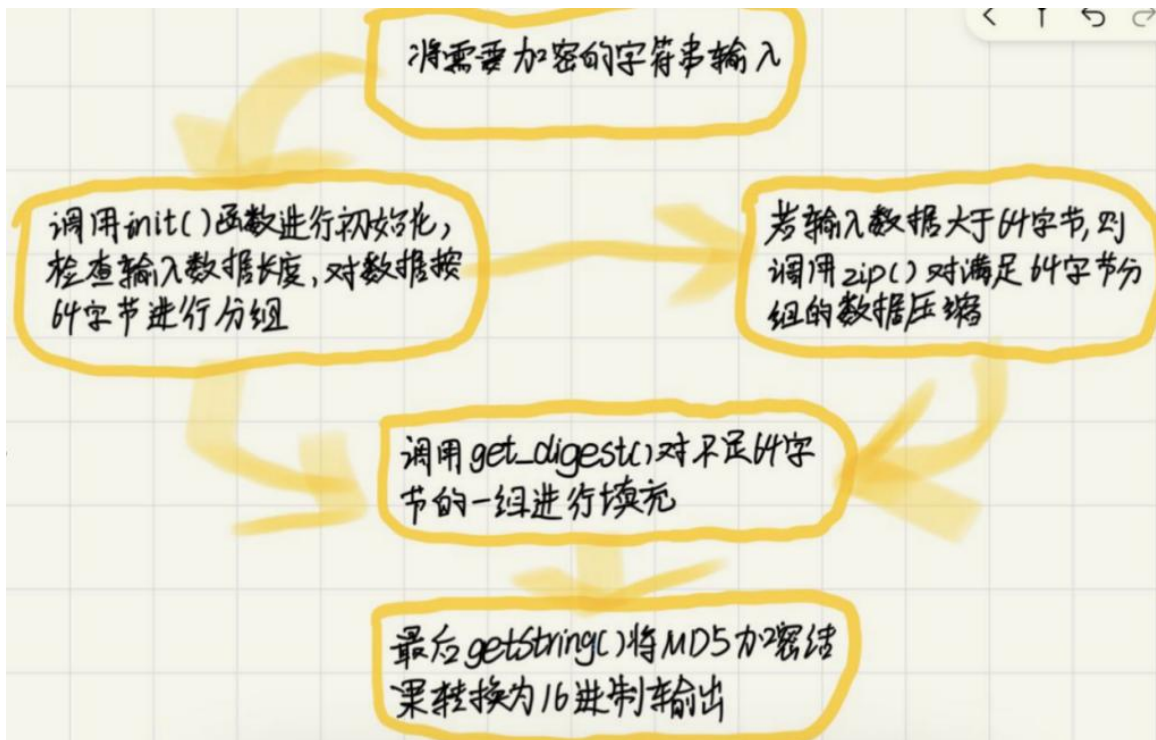
(4)get_digest()函数

该函数负责对最后一组进行填充、压缩等操作：

1. 首先将 `count` 中计算的数据长度用 `encode` 转换，转换完的数据可以直接填在最后 8 字节
2. 接着获取当前数据已有的字节，计算填充字节长度，如果长度小于 56 字节即 448 比特，则填充字节=56-数据字节，否则，填充字节=120-数据字
3. 填充字节是以 1 开头，后续全是 0，则构造一个数组 `tinachong[64]`，其开头为 0x80 即 10000000，用这个数组进行填充
4. 调用刚刚的 `init(tianchong, padLen)` 将填充数组的内容以一定长度填入我们的最后一个分组
5. 将最后 8 字节填入之前算好数据长度，再调用压缩函数进行计算即可

(5)getstring()函数

- 获得加密后的字符串函数使用 `for` 循环，将数据转化为 16 进制数
总体流程图如下所示：



较为重要的函数到这里结束，接下来给出相应的代码

6.3. 代码实现

- 首先还是为了方便后面的雪崩效应写的将 16 进制字符串转换为 2 进制 bit 流的函数

```

void getbit(string a, bitset<128>& temp)
{
    int num = 127;
    // 如果无前缀0x 则这里i 需要从2 开始
    for (int i = 0; i < a.length(); i++)
    {
        if (a[i] <= '9')
        {
            for (int j = 0; j < 4; j++)
            {
                temp[num--] = HexToBit[a[i] - 48][j];
            }
        }
        else
        {
            for (int j = 0; j < 4; j++)
            {
                temp[num--] = HexToBit[a[i] - 65 + 10][j];
            }
        }
    }
}

```

- 接下来是 4 个轮函数的定义，其中 FQ,GQ,HQ,IQ函数是书本上的基本逻辑函数

```

void round1(unsigned int& a, unsigned int& b, unsigned int& c, unsigned int& d, unsigned int& x, unsigned int s, unsigned int ac)
{
    a += F(b, c, d) + x + ac;
    a = leftshift(a, s);
    a += b;
}

void round2(unsigned int& a, unsigned int& b, unsigned int& c, unsigned int& d, unsigned int& x, unsigned int s, unsigned int ac)
{
    a += G(b, c, d) + x + ac;
    a = leftshift(a, s);
    a += b;
}

void round3(unsigned int& a, unsigned int& b, unsigned int& c, unsigned int& d, unsigned int& x, unsigned int s, unsigned int ac)
{
    a += H(b, c, d) + x + ac;
    a = leftshift(a, s);
    a += b;
}

void round4(unsigned int& a, unsigned int& b, unsigned int& c, unsigned int& d, unsigned int& x, unsigned int s, unsigned int ac)
{
    a += I(b, c, d) + x + ac;
    a = leftshift(a, s);
    a += b;
}

```

```

d int s, unsigned int ac)
{
    a += G(b, c, d) + x + ac;
    a = leftshift(a, s);
    a += b;
}
void round3(unsigned int& a, unsigned int& b, unsigned int& c, unsigned int& d, unsigned int& x, unsigned int s, unsigned int ac)
{
    a += H(b, c, d) + x + ac;
    a = leftshift(a, s);
    a += b;
}
void round4(unsigned int& a, unsigned int& b, unsigned int& c, unsigned int& d, unsigned int& x, unsigned int s, unsigned int ac)
{
    a += I(b, c, d) + x + ac;
    a = leftshift(a, s);
    a += b;
}

```

- 然后是填充和压缩函数

```

unsigned char* MD5::get_digest()
{
    if (!finished)
    {
        finished = true;

        unsigned char bits[8];
        unsigned int oldState[4];
        unsigned int oldCount[2];
        int nowlength, waitlength;

        memcpy(oldState, link, 16);
        memcpy(oldCount, count, 8);

        encode(count, bits, 8);

        nowlength = (unsigned int)((count[0] >> 3) & 0x3f);
        waitlength = (nowlength < 56) ? (56 - nowlength) : (120 - nowlength);
        init(tianchong, waitlength);

        nowlength = (unsigned int)((count[0] >> 3) & 0x3f);

        memcpy(&buffer[nowlength], bits, 8);
        zip(buffer);

        encode(link, digest, 16);
        memcpy(link, oldState, 16);
        memcpy(count, oldCount, 8);
    }
    return digest;
}

```

- 接下来是初始化函数

```

void MD5::init(unsigned char* input, int len)
{
    unsigned int i, nowlength, waitlength;

    finished = false;

    nowlength = (unsigned int)((count[0] >> 3) & 0x3f);
    count[0] += (unsigned int)len << 3;
    if ((count[0]) < ((unsigned int)len << 3)) {
        count[1] += 1;
    }
    count[1] += ((unsigned int)len >> 29);

    waitlength = 64 - nowlength;
    if (len >= waitlength) {

```

```

        memcpy(&buffer[nowlength], input, waitlength);
        zip(buffer);
        for (i = waitlength; i + 63 < len; i += 64) {
            zip(&input[i]);
        }
        nowlength = 0;
    }
    else {
        i = 0;
    }
    memcpy(&buffer[nowlength], &input[i], len - i);
}

```

- 接下来是压缩函数，整合全局步骤

```

void MD5::zip(unsigned char block[64])
{
    unsigned int a = link[0], b = link[1], c = link[2], d = link[3], x[16];
    for (int i = 0, j = 0; j < 64; ++i, j += 4)
    {
        x[i] = (((unsigned int)block[j]) | (((unsigned int)block[j + 1]) << 8) | (((unsigned int)block[j + 2]) << 16) | (((unsigned int)block[j + 3]) << 24));
    }
    round1(a, b, c, d, x[0], s[0][0], 0xd76aa478);
    round1(d, a, b, c, x[1], s[0][1], 0xe8c7b756);
    round1(c, d, a, b, x[2], s[0][2], 0x242070db);
    round1(b, c, d, a, x[3], s[0][3], 0xc1bdceee);
    round1(a, b, c, d, x[4], s[0][0], 0xf57c0faf);
    round1(d, a, b, c, x[5], s[0][1], 0x4787c62a);
    round1(c, d, a, b, x[6], s[0][2], 0xa8304613);
    round1(b, c, d, a, x[7], s[0][3], 0xfd469501);
    round1(a, b, c, d, x[8], s[0][0], 0x698098d8);
    round1(d, a, b, c, x[9], s[0][1], 0x8b44f7af);
    round1(c, d, a, b, x[10], s[0][2], 0xffff5bb1);
    round1(b, c, d, a, x[11], s[0][3], 0x895cd7be);
    round1(a, b, c, d, x[12], s[0][0], 0x6b901122);
    round1(d, a, b, c, x[13], s[0][1], 0xfd987193);
    round1(c, d, a, b, x[14], s[0][2], 0xa679438e);
    round1(b, c, d, a, x[15], s[0][3], 0x49b40821);

    round2(a, b, c, d, x[1], s[1][0], 0xf61e2562);
    round2(d, a, b, c, x[6], s[1][1], 0xc040b340);
    round2(c, d, a, b, x[11], s[1][2], 0x265e5a51);
    round2(b, c, d, a, x[0], s[1][3], 0xe9b6c7aa);
    round2(a, b, c, d, x[5], s[1][0], 0xd62f105d);
    round2(d, a, b, c, x[10], s[1][1], 0x2441453);
    round2(c, d, a, b, x[15], s[1][2], 0xd8a1e681);
    round2(b, c, d, a, x[4], s[1][3], 0x7d3fbc8);
    round2(a, b, c, d, x[9], s[1][0], 0x21e1cde6);
    round2(d, a, b, c, x[14], s[1][1], 0xc33707d6);
    round2(c, d, a, b, x[3], s[1][2], 0xf4d50d87);
    round2(b, c, d, a, x[8], s[1][3], 0x455a14ed);
    round2(a, b, c, d, x[13], s[1][0], 0xa9e3e905);
    round2(d, a, b, c, x[2], s[1][1], 0xfcefa3f8);
    round2(c, d, a, b, x[7], s[1][2], 0x676f02d9);
    round2(b, c, d, a, x[12], s[1][3], 0x8d2a4c8a);

    round3(a, b, c, d, x[5], s[2][0], 0xfffa3942);
    round3(d, a, b, c, x[8], s[2][1], 0x8771f681);
    round3(c, d, a, b, x[11], s[2][2], 0x6d9d6122);
    round3(b, c, d, a, x[14], s[2][3], 0xfde5380c);
    round3(a, b, c, d, x[1], s[2][0], 0xa4beea44);
    round3(d, a, b, c, x[4], s[2][1], 0x4bdecfa9);
    round3(c, d, a, b, x[7], s[2][2], 0xf6bb4b60);
    round3(b, c, d, a, x[10], s[2][3], 0xbedbf60);
    round3(a, b, c, d, x[13], s[2][0], 0x289b7ec6);
    round3(d, a, b, c, x[0], s[2][1], 0xeeaa127fa);
    round3(c, d, a, b, x[3], s[2][2], 0xd4ef3085);
    round3(b, c, d, a, x[6], s[2][3], 0x4881d05);
    round3(a, b, c, d, x[9], s[2][0], 0xd9d4d039);
    round3(d, a, b, c, x[12], s[2][1], 0xeddb99e5);
    round3(c, d, a, b, x[15], s[2][2], 0x1fa27cf8);
    round3(b, c, d, a, x[2], s[2][3], 0xc4ac5665);
}

```

```

round4(a, b, c, d, x[0], s[3][0], 0xf4292244);
round4(d, a, b, c, x[7], s[3][1], 0x432aff97);
round4(c, d, a, b, x[14], s[3][2], 0xab9423a7);
round4(b, c, d, a, x[5], s[3][3], 0xfc93a039);
round4(a, b, c, d, x[12], s[3][0], 0x655b59c3);
round4(d, a, b, c, x[3], s[3][1], 0x8f0ccc92);
round4(c, d, a, b, x[10], s[3][2], 0xffeff47d);
round4(b, c, d, a, x[1], s[3][3], 0x85845dd1);
round4(a, b, c, d, x[8], s[3][0], 0x6fa87e4f);
round4(d, a, b, c, x[15], s[3][1], 0xfe2ce6e0);
round4(c, d, a, b, x[6], s[3][2], 0xa3014314);
round4(b, c, d, a, x[13], s[3][3], 0x4e0811a1);
round4(a, b, c, d, x[4], s[3][0], 0xf7537e82);
round4(d, a, b, c, x[11], s[3][1], 0xbd3af235);
round4(c, d, a, b, x[2], s[3][2], 0x2ad7d2bb);
round4(b, c, d, a, x[9], s[3][3], 0xeb86d391);

link[0] += a;
link[1] += b;
link[2] += c;
link[3] += d;
}

```

7. 服务器结果展示

到这里本次大作业也将近尾声了，最后展示一下通讯程序运行的结果：

7.1. 开始的发送公钥和共享 AES 密钥

结果如图所示：

```

Client:
C: socket加载成功
C: 成功与服务器进行连接
C: 从服务器接收服务器的公钥对:2362329539,24085
C: 向服务器发送客户端的公钥对:566700097,4967
C: 随机生成一个AES密钥:!x[{4<2HwJL&xa!U
C: 向服务器发送加密的AES密钥:2288242818,1970124067,1974904059,1813943469,2293457526,1032438663,1801430689,451696394,
C: 请输入明文:123456
C: [2023-06-27 21:59:05]经过解密后的明文:
C: 请输入明文:|

```

可以看出与设计相同，实现正确

7.2. 收发消息

```

Server:
S: socket加载成功
S: 成功绑定端口 等待客户端连接
S: 向客户端发送服务器的公钥对:2362329539,24085
S: 从客户端接收客户端的公钥对:566700097,4967
S: 从客户端接收到加密的AES密钥:835569018,183268,514792851,507119755,
S: 解密的AES密钥为:.{fel'm.B%C |\\j+
S: [2022-12-28 14:34:52]经过解密后的明文:hello
S: 请输入明文:hi
S: [2022-12-28 14:35:23]经过解密后的明文:how was you lately?
S: 请输入明文:i am very fine.
S: [2022-12-28 14:35:55]经过解密后的明文:i don't know your name

```

结果如图所示：

可以看出与设计相同，实现正确

8. 总结与展望

8.1. 总结

本次作业一开始使用的数据库系统在进行数据的插入删除更新是问题不大，用户也可以通过多条 sql 语句尝试把一条注册信息加入到 user 用户界面和 ca 证书界面当中，但是却发现其中大整数的存储出现了明显的问题，不仅是 rsa 所需要的 1024bit 位数，包括用户与用户之间，用户与服务器之间所需要加密的 MD5 哈希函数和 AES 加解密函数当中都需要运用 128 位以上的大整数，然而由于数据库系统 int 变量只能存储 11 位整数，即使 bigint 运用也只能存储 40 位数，其中大量的密钥和身份信息等变量需要在存储过后进行进一步运算，显然我们的数据库系统就在运算当中出现了问题，我尝试进行了 RSA 降低位数和 AES 降低输入位数，发现由于一定要求的存在，加密所需必须有大整数的存在，说明此方案的不可行。但我还是利用他进行了基本数据的实现，将整型转换为字符型，能够起到存储，证书申请，证书更新，证书修改，用户注册，用户注销等功能，用户也可以用文件中包含的程序进行相关数据的验证，但是其方法仍然比较麻烦，需要较多的手动操作，在工业中虽然也可以应用，但却会给使用者带来一定的困扰。于是我尝试了使用服务器框架来实现一些未能完成的功能，整体上完成了本次大作业的全部要求。

之后我学习了网络通讯的相关知识，借助之前旧的服务器框架进行改进和功能添加，得到了最终的服务器，作为本次大作业的展示。

我们使用了加密技术来保护用户的敏感信息，避免了信息被不法分子盗用，也增加了传输的安全性和可靠性。同时我们还设计了数据库来存储用户的信息，保证了信息的安全存储。

通过这个项目，我们对密码学的理论知识和实际应用有了更深入的理解，对服务器和数据库的运用和管理也更加熟练。

8.2. 展望

未来，我们可以继续完善这个信息传输系统，不断增加安全性和可靠性，同时也可以将这个系统应用到更多的场合中，如银行、政府、企业等领域，保护数据的安全。也可以使用更加先进的加密算法和技术，加强系统的安全性。