**实验6：非对称可搜索加密**

**学号：2112066**

**姓名：于成俊**

**专业：密码科学与技术**

## 一、实验内容

在这次的实验中，我们实现了PEKSBoneh2004方案。通过对实验代码的逐步分析，我们能够深入理解PEKSBoneh2004方案的工作原理。实验代码的步骤如下：首先，我们输入安全参数；接着，生成一对公私钥；然后，利用公钥对关键词进行加密；接下来，根据私钥和关键词计算出陷门。最后，我们通过陷门和密文进行匹配测试，以此来验证密钥和密文是否匹配，进而确认协议的正确性。

## 实验环境

系统：Ubuntu 18.04.6 LTS

语言：python 3.6.9

## 三、实验原理

### **1. 基本定义**

### （1）算法描述

定义（非对称可搜索加密，public key encryption with keyword search, PEKS）非对称密码体制下可搜索加密算法可描述为：

* ：输入安全参数，输出公钥和私钥；
* ：输入公钥和关键词，输出关键词密文；
* ：输入私钥和关键词，输出陷门；
* ：输入公钥、陷门和关键词密文，根据与的匹配结果，输出判定值

### （2）算法一致性

加密算法的一致性是指解密与加密互为逆过程，即对任意明文，使用公钥加密后得到密文，如果再使用对应的私钥解密，必能得到的一致性应满足：1）对任意关键词，；2）对任意关键词且，。鉴于此，Abdalla等人对如上所述的完美一致性进行扩展，定义针对PEKS的计算一致性和统计一致性。

计算一致性和统计一致性的定义都基于实验。攻击者已知公钥，其目标是通过一定次数访问随机预言机后（以中使用的哈希函数相应的查询），输出关键词对，满足且。攻击者具有攻击优势：

* 如果为任意攻击者且，则该方案达到统计一致性；
* 如果为任意多项式时间攻击者且，则该方案达到计算一致性。

### （3）典型应用

基于上述定义，Boneh等人提出不可信赖邮件服务器路由问题的解决思路：用户Alice掌握着私钥，并将相对应的公钥公开，为了让电子邮件网关分拣接收到的邮件，Alice会事先将一些特定关键字的陷门发送给电子邮件网关，使得它能够通过判断邮件中是否包含关键字来选择接受设备。与此同时，电子邮件网关在判断的过程中无法获得关于关键字和邮件内容的有效信息。

比如，Bob使用Alice的公钥加密邮件和相关关键词，并将形如的密文发送至邮件服务器。这里，为公钥密码加密算法，为邮件内容，为与关联的关键词。Alice将或长驻服务器，新邮件到来时，服务器自动对其关联的关键词执行与或相关的算法，如果输出1，便将该邮件转发至Alice的手机或个人电脑。

### 2. 基本方案

2004年，Boneh提出了第一个非对称的可搜索加密方案，具体构造如下：

令为双线性对，函数和为哈希函数。

* Keygen：输入安全参数，该安全参数决定群和的阶p，随机挑选和的生成元g，输出和。
* Encrypt：输入公钥和关键词，随机选择，计算，输出。
* TrapDoor：输入私钥和关键词，输出。
* Test：输入陷门和密文，记密文为，若，输出1，否则输出0。

正确性：





### 3. 关键词猜测攻击

PEKS本身定义存在严重的安全隐患：关键词猜测攻击。关键词猜测攻击是由于关键词空间远小于密钥空间，而且用户通常使用常用关键词进行检索，这就给攻击者提供了只需采用字典攻击就能达到目的的“捷径”。

导致关键词猜测攻击的原因可归结为：①关键词空间较小，且用户集中于使用常用词汇，给攻击者提供了遍历关键词空间的可能；②PEKS算法一致性约束，使攻击者拥有对本次攻击是否成功的预先判定：执行*Test*算法，返回1说明本次攻击成功，否则可以再继续猜测。

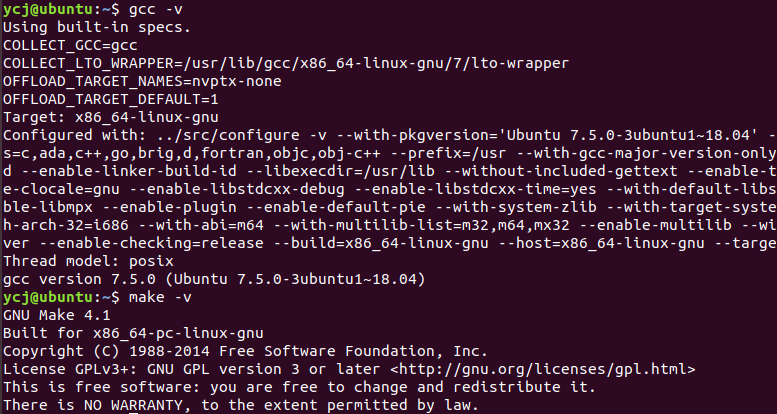
为抵御关键词猜测攻击，很多方案提出，比如可以在服务器端进行模糊陷门测试，过滤大部分不相关邮件，最后在本地精确匹配，得到检索结果。这种方法通过引入模糊陷门，一定程度降低了接收者外部PEKS算法一致性，使其能够抵御关键词猜测攻击，但增加了客户服务器通信量和用户端计算量。

## 实验步骤

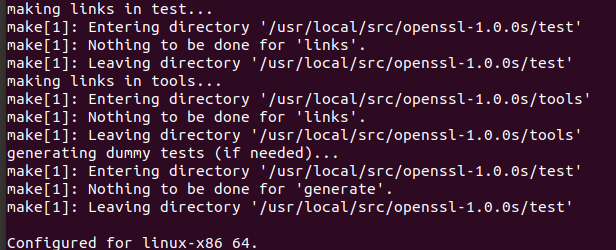
由于Ubuntu 18.04.6 LTS 自带python，所以不需要再安装python

### （1）部署环境

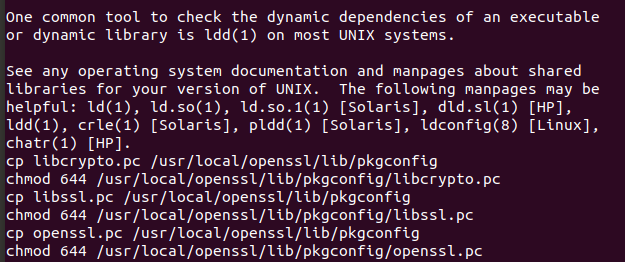
1. 使用命令`sudo apt-get install gcc make`安装gcc和make，安装完成后，可使用命令`gcc -v`和`make -v`来查看版本，以检验是否安装成功。



1. 使用命令`sudo apt-get install m4 flex bison`安装依赖库m4、flex、bison
2. 使用命令`sudo apt-get install python3-setuptools python3-dev libssl-dev`安装Python依赖包python3-setuptools、python3-dev 、libssl-dev
3. 使用命令`sudo apt-get install python3-pip`安装python3-pip，然后使用命令`pip3 install pyparsing==2.4.6`安装Python第三方包pyparsing
4. 编译安装OpenSSL 1.0.0s：在网上下载openssl-1.0.0s.tar.gz，并将其放在Downloads目录下，然后使用命令`sudo tar -zxvf openssl-1.0.0s.tar.gz -C /usr/local/src/`将压缩包openssl-1.0.0s.tar.gz解压至/usr/local/src/目录。使用命令`cd /usr/local/src/openssl-1.0.0s/`跳转目录，然后使用命令`sudo ./config shared --prefix=/usr/local/openssl --openssldir=/usr/lib/openssl`写入编译配置，配置写入成功的话会显示如下的信息：



接着使用命令`sudo make`进行编译，然后使用命令`sudo make install`进行安装。显示如下信息就表示成功安装：



附上openssl-1.0.0s.tar.gz下载链接：

https://link.zhihu.com/?target=https%3A//www.openssl.org/source/old/1.0.0/openssl-1.0.0s.tar.gz

1. 执行以下指令，通过软链接将OpenSSL 1.0.0s的命令和库文件链接到系统：

sudo mv /usr/bin/openssl /usr/bin/openssl.bak

sudo ln -s /usr/local/openssl/bin/openssl /usr/bin/openssl

sudo ln -s /usr/local/openssl/include/openssl /usr/include/openssl

1. 执行以下指令，配置动态库软链接：

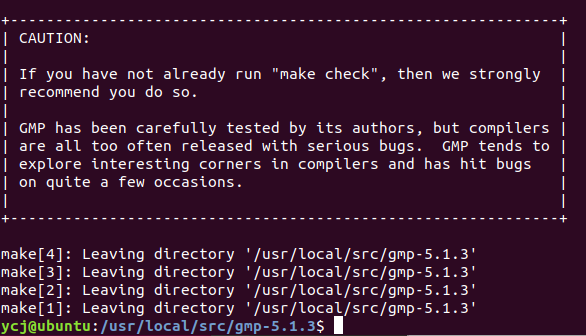
sudo ln -s /usr/local/openssl/lib/libssl.so.1.0.0 /usr/lib/libssl.so

sudo ln -s /usr/local/openssl/lib/libcrypto.so.1.0.0 /usr/lib/libcrypto.so

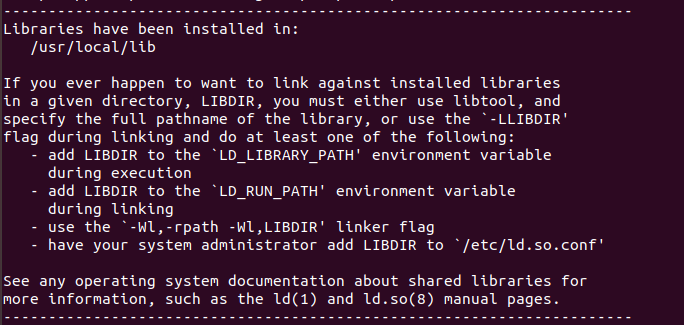
1. 执行指令`sudo gedit /etc/ld.so.conf `，然后在配置末尾增加一行，写入include /usr/local/openssl/lib然后保存退出。执行命令`openssl version`后会显示此时OpenSSL的版本号已经变为1.0.0s：

4

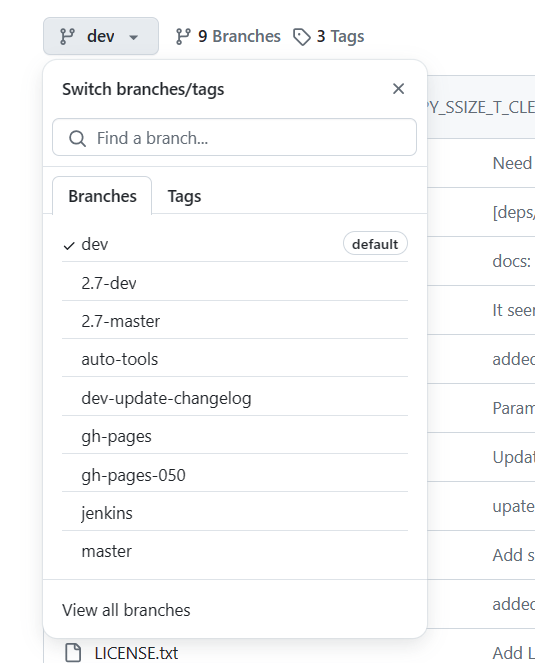
1. 在Downloads目录下使用命令`wget https://gmplib.org/download/gmp/gmp-5.1.3.tar.bz2`下载GMP压缩包，然后使用命令`sudo tar -xvjf gmp-5.1.3.tar.bz2 -C /usr/local/src`进行解压，输入命令`cd /usr/local/src/gmp-5.1.3`切换到gmp-5.1.3文件夹中，输入命令`sudo ./configure`来写入配置，接着输入命令`sudo make`进行编译，最后输入命令`sudo make install`进行安装，结果如下则安装成功：



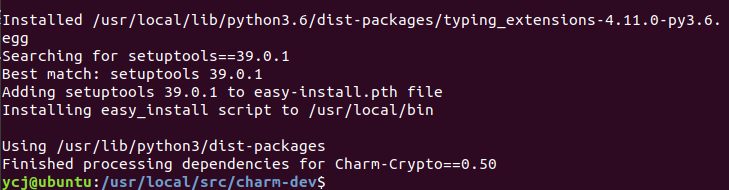
1. 在Downloads目录下使用命令`wget https://crypto.stanford.edu/pbc/files/pbc-0.5.14.tar.gz`下载PBC压缩包，然后使用命令`sudo tar -zxvf pbc-0.5.14.tar.gz -C /usr/local/src/`进行解压，输入命令`cd /usr/local/src/pbc-0.5.14/`进入pbc-0.5.14文件夹，输入命令`sudo ./configure`来写入配置，接着输入命令`sudo make`进行编译，最后输入命令`sudo make install`进行安装，若在终端的中间出现下面的信息则安装成功：



1. 在charm github官网下载charm代码，将其放入Downloads目录下。（注意，要下载dev版本的）如下：

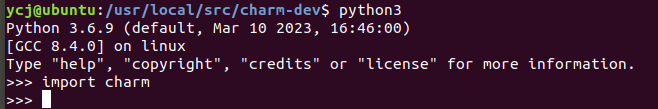


1. 使用命令`sudo mv ./charm-dev.zip /usr/local/src/`将压缩包移动/usr/local/src/下，然后使用命令`cd /usr/local/src/`切换目录，接着使用命令`sudo unzip ./charm-dev.zip`进行解压，使用命令`cd /usr/local/src/charm-dev/`进入charm-dev文件夹，输入命令`sudo ./configure.sh`来写入配置，接着输入命令`sudo make`进行编译，最后输入命令`sudo make install`进行安装，若终端出现下面的信息则安装成功：



charm github官网链接：https://github.com/JHUISI/charm

1. 在终端启动Python3，然后尝试import charm，若没有报错，则部署成功！



### （2）编写代码

1. 导入相关包

# 导入相关包

**from** charm.toolbox.pairinggroup **import** PairingGroup, ZR, G1, G2, GT, pair

**import** hashlib

1. 编写进行哈希的函数

# 创建名为Hash1pre的变量，并将其设置为hashlib模块中的md5函数

Hash1pre = hashlib.md5

# 进行哈希

**def** **Hash1**(w):

# 将参数w转换为字符串，然后使用utf8编码，接着使用之前定义的MD5哈希函数进行哈希，最后将哈希值转换为16进制字符串

hv = Hash1pre(str(w).encode('utf8')).hexdigest()

# 打印出哈希值。

print("hv: ", hv)

# 使用group的hash方法对哈希值进行再次哈希，哈希的类型为G1

hv = group.hash(hv, type=G1)

# 返回最终的哈希值。

**return** hv

1. 编写生成公钥和私钥的函数

#创建名为Hash2的变量，并将其设置为hashlib模块中的sha256函数

Hash2 = hashlib.sha256

# 生成公钥和私钥

**def** **Setup**(param\_id='SS512'):

# 使用PairingGroup类创建了一个名为group的配对群，配对群的参数由param\_id指定。

group = PairingGroup(param\_id)

# 在G1类型的群中随机生成了一个元素g。

g = group.random(G1)

# 在ZR类型的群中随机生成了一个元素alpha。

alpha = group.random(ZR)

# 将alpha序列化后赋值给sk

sk = group.serialize(alpha)

# 将g和g的alpha次方序列化后赋值给pk

pk = [group.serialize(g), group.serialize(g \*\* alpha)]

# 返回私钥和公钥。

**return** [sk, pk]

1. 编写加密函数

# 加密

**def** **Enc**(pk, w, param\_id='SS512'):

# 使用PairingGroup类创建了一个名为group的配对群，配对群的参数由param\_id指定。

group = PairingGroup(param\_id)

# 从公钥pk中反序列化出两个元素g和h。

g, h = group.deserialize(pk[0]), group.deserialize(pk[1])

# 在ZR类型的群中随机生成了一个元素r。

r = group.random(ZR)

# 计算配对函数pair的值，输入是消息w的哈希值和h的r次方。

t = pair(Hash1(w), h \*\* r)

# 计算g的r次方。

c1 = g \*\* r

# 将t赋值给c2。

c2 = t

# 打印出c2的序列化值。

print("group.serialize(c2): ", group.serialize(c2))

# 返回c1的序列化值和c2的序列化值的SHA256哈希值。

**return** [group.serialize(c1), Hash2(group.serialize(c2)).hexdigest()]

1. 编写计算陷门的函数

# 计算陷门

**def** **TdGen**(sk, w, param\_id='SS512'):

# 使用PairingGroup类创建了一个名为group的配对群，配对群的参数由param\_id指定。

group = PairingGroup(param\_id)

# 从私钥sk中反序列化出一个元素。

sk = group.deserialize(sk)

# 计算消息w的哈希值的sk次方。

td = Hash1(w) \*\* sk

# 返回td的序列化值。

**return** group.serialize(td)

1. 编写测试函数

# 测试

**def** **Test**(td, c, param\_id='SS512'):

# 使用PairingGroup类创建了一个名为group的配对群，配对群的参数由param\_id指定。

group = PairingGroup(param\_id)

# 从密文c中反序列化出一个元素c1。

c1 = group.deserialize(c[0])

# 从密文c中取出哈希值c2。

c2 = c[1]

# 打印出哈希值c2。

print("c2: ", c2)

# 从td中反序列化出一个元素。

td = group.deserialize(td)

# 计算陷门td和元素c1的配对值的序列化值的SHA256哈希值，然后与哈希值c2进行比较，如果相等则返回True，否则返回False。

**return** Hash2(group.serialize(pair(td, c1))).hexdigest() == c2

1. 编写主函数

# 主函数

**if** \_\_name\_\_ == '\_\_main\_\_':

# 设置配对群的参数为'SS512'，然后调用Setup函数生成私钥和公钥。

param\_id = 'SS512'

[sk, pk] = Setup(param\_id)

# 创建一个名为group的配对群，配对群的参数由param\_id指定。

group = PairingGroup(param\_id)

# 对关键字"yes"进行加密来生成密文。

c = Enc(pk, "yes")

# 生成陷门

td = TdGen(sk, "yes")

# 测试陷门和密文是否匹配，如果匹配则返回True，否则返回False。这里应该返回True。

**assert** (Test(td, c))

# 生成一个新的陷门，对应的关键字是"no"。

td = TdGen(sk, "no")

# 测试新的陷门和原来的密文是否匹配，这里应该返回False。

**assert** (not Test(td, c))

# 对新的关键字"Su\*re"进行加密，然后测试新的陷门和新的密文是否匹配，这里应该返回False。

c = Enc(pk, "Su\*re")

**assert** (not Test(td, c))

# 对关键字"no"进行加密，然后测试新的陷门和新的密文是否匹配，这里应该返回True。

c = Enc(pk, "no")

**assert** (Test(td, c))

# 对关键字9 \*\* 100进行加密，然后生成对应的陷门。

c = Enc(pk, 9 \*\* 100)

td = TdGen(sk, 9 \*\* 100)

# 测试陷门和密文是否匹配，这里应该返回True。

**assert** (Test(td, c))

# 生成一个新的陷门，对应的关键字是9 \*\* 100 + 1。

td = TdGen(sk, 9 \*\* 100 + 1)

# 测试新的陷门和原来的密文是否匹配，这里应该返回False。

**assert** (not Test(td, c))

1. 代码流程图如下：

输入安全参数，调用Setup函数生成公钥和私钥

输入关键字和公钥，调用Enc函数生成密文

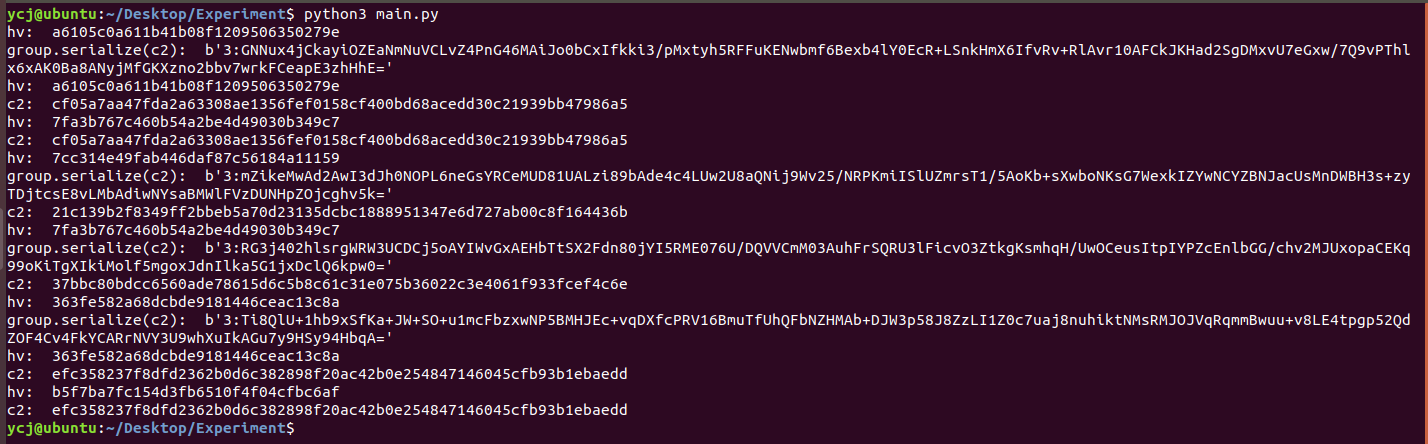
输入关键字和私钥，调用TdGen函数生成陷门

输入密文和陷门，调用Test函数测试是否匹配

生成多个密文和陷门，多次进行测试

## 五、实验结果

在main.py所在目录下运行命令行，使用命令`python main.py`运行程序。结果如下：



从输出上可以看出，每个assert都通过了，说明与设想的一样，成功的实现了方案。

## 六、附加题

基于Boneh的第一个PEKS方案，简述抵御关键词猜测攻击的办法。如果可以，请通过代码实现。

在基于Boneh的第一个PEKS方案中，抵御关键词猜测攻击的主要方法是使用安全哈希函数和双线性映射。这种方法的基本思想是将关键词转化为不可逆的哈希值，然后使用双线性映射进行加密。由于哈希函数的单向性和双线性映射的复杂性，攻击者即使知道哈希值也无法推断出原始关键词，从而实现对关键词猜测攻击的防御。但是由于关键字空间远小于密钥空间，敌手可以通过离线关键字猜测攻击轻松破解PEKS体制。代码如下：

# 导入所需的库

**from** charm.toolbox.pairinggroup **import** PairingGroup,ZR,G1,G2,GT,pair

**from** charm.toolbox.PKSig **import** PKSig

**from** charm.toolbox.hash\_module **import** Hash, hashPair **as** extractor

# 定义PEKS\_Boneh类**class** **PEKS\_Boneh**:

# 初始化函数，设置群和哈希函数

**def** **\_\_init\_\_**(self, group):

self.group = group

self.hash = Hash('sha256')

# 生成公私钥对的函数

**def** **keygen**(self):

g = self.group.random(G1) # 随机选择一个生成元g

t = self.group.random(ZR) # 随机选择一个私钥t

h = g \*\* t # 计算公钥h

pk = {'g': g, 'h': h} # 公钥包括g和h

sk = {'t': t} # 私钥是t

**return** (pk, sk) # 返回公私钥对

# 加密关键词的函数

**def** **encrypt**(self, pk, keyword):

r = self.group.random(ZR) # 随机选择一个r

h = self.hash.hashToZr(keyword) # 计算关键词的哈希值

c = pk['g'] \*\* r # 计算密文的一部分c

d = (pk['h'] \*\* r) \* (pk['g'] \*\* h) # 计算密文的另一部分d

**return** {'c': c, 'd': d} # 返回密文

# 生成陷门的函数

**def** **trapdoor**(self, sk, keyword):

h = self.hash.hashToZr(keyword) # 计算关键词的哈希值

**return** sk['t'] \* h # 返回陷门

# 测试密文和陷门是否匹配的函数

**def** **test**(self, pk, ct, td):

e1 = pair(ct['d'], pk['g']) # 计算配对e1

e2 = pair(ct['c'], pk['h'] \*\* td) # 计算配对e2

**return** e1 == e2 # 如果e1等于e2，返回True，否则返回False

## 实验感想

在进行PEKSBoneh2004方案的实验过程中，我深深地体会到了密码学的魅力和挑战。这个实验不仅让我对PEKSBoneh2004方案有了深入的理解，也让我更加明白了保护数据安全的重要性。

首先，通过编写和分析实验代码，我对PEKSBoneh2004方案的工作原理有了更深的理解。我了解到如何通过公私钥对、哈希函数和双线性映射来实现关键词的加密和匹配测试。这个过程虽然复杂，但却让我对密码学的深度和广度有了新的认识。

其次，我对如何抵御关键词猜测攻击有了更深的理解。我明白了，只有深入理解攻击的原理和可能的防御策略，才能更好地保护我们的数据安全。