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。

二、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$PEKS = (KeyGen, Encrypt, Trapdoor, Test)$：

- $(pk, sk) = KeyGen(\lambda)$：输入安全参数$\lambda$，输出公钥$pk$和私钥$sk$；
- $C_W = Encrypt(pk, W)$：输入公钥pk和关键词W，输出关键词密文C_W；
- $T_W = Trapdoor(sk, W)$：输入私钥sk和关键词W，输出陷门T_W；

- $b = Test(pk, C_W, T_{W'})$：输入公钥$pk$、陷门$T_{W'}$和关键词密文$C_W$，根据$W$与$W'$的匹配结果，输出判定值$b \in \{0,1\}$。

（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M，使用公钥pk加密后得到密文C，如果再使用pk对应的私钥sk解密，必能得到$M.PEKS$的一致性应满足：1）对任意关键词W，$\Pr\left[Test\left(pk, PEKS(pk, W), Trapdoor(sk, W)\right) = 1\right] = 1$；2）对任意关键词$W_1, W_2$且$W_1 \neq W_2$，$\Pr\left[Test\left(pk, PEKS(pk, W_1), Trapdoor(sk, W_2)\right) = 1\right] = 0$。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$Exp^{consist}$。攻击者A已知公钥pk，其目标是通过一定次数访问随机预言机$O_H(\cdot)$后（$O_H(\cdot)$以$PEKS$中使用的哈希函数$H(\cdot)$相应A的查询），输出关键词对(W_1, W_2)，满足$W_1 \neq W_2$且$Test\left(pk, PEKS(pk, W_1), Trapdoor(sk, W_2)\right) = 1$。攻击者$A$具有攻击优势$Adv^{consist}_{PEKS,O_H}(A) = \Pr\left[Exp^{consist}_{PEKS,O_H}(A) \Rightarrow true\right]$：

- 如果A为任意攻击者且$Adv^{consist}_{PEKS,O_H} < \varepsilon$，则该$PEKS$方案达到统计一致性；
- 如果A为任意多项式时间攻击者且$Adv^{consist}_{PEKS,O_H} < \varepsilon$，则该$PEKS$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T_W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W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pk加密邮件和相关关键词，并将形如$(PKE.Encrypt(pk, MSG),\ PEKS.Encrypt(pk, W_1),\ \cdots,\ PEKS.Encrypt(pk, W_n))$的密文发送至邮件服务器。这里，$PKE.Encrypt$为公钥密码加密算法，$MSG$为邮件内容，$W_1, \cdots, W_n$为与$MSG$关联的关键词。Alice 将$T_{"urgent"}$或$T_{"lunch"}$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$T_{"urgent"}$或$T_{"lunch"}$相关的$Test$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$e: G_1 \times G_1 \to G_2$为双线性对，函数$H_1: \{0,1\}^* \to G_1$和$H_2: G_2 \to \{0,1\}^{\log p}$为哈希函数。

• Keygen：输入安全参数，该安全参数决定群G_1和G_2的阶 p，随机挑选$\alpha \leftarrow Z_p^*$和G_1的生成元 g，输出$pk \coloneqq (g, h = g^{\alpha})$和$sk \coloneqq \alpha$。

• Encrypt：输入公钥和关键词，随机选择$r \leftarrow Z_p^*$，计算$t \coloneqq e(H_1(w), h^r)$，输出$c \coloneqq (g^r, H_2(t))$。

• TrapDoor：输入私钥和关键词，输出$td \coloneqq H_1(w)^{\alpha}$。

• Test：输入陷门和密文，记密文为$c = (c_1, c_2)$，若$H_2(e(td, c_1)) = c_2$，输出 1，否则输出 0。

正确性：

$$\mathrm{e}(td, \mathrm{c}_1) = \mathrm{e}(\mathsf{H}_1(\mathrm{w})^{\alpha}, \mathrm{g}^{\mathrm{r}}) = \mathrm{e}(\mathsf{H}_1(\mathrm{w}), \mathrm{g}^{\alpha \mathrm{r}}) = \mathrm{e}(\mathsf{H}_1(\mathrm{w}), \mathrm{h}^{\mathrm{r}}),$$

$$\mathsf{H}_2(\mathrm{e}(td, c_1)) = c_2$$

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。

四、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。

```
ycj@ubuntu:~$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/7/lto-wrapper
OFFLOAD_TARGET_NAMES=nvptx-none
OFFLOAD_TARGET_DEFAULT=1
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 7.5.0-3ubuntu1~18.04' -
s=c,ada,c++,go,brig,d,fortran,objc,obj-c++ --prefix=/usr --with-gcc-major-version-only
d --enable-linker-build-id --libexecdir=/usr/lib --without-included-gettext --enable-t
e-clocale=gnu --enable-libstdcxx-debug --enable-libstdcxx-time=yes --with-default-libs
ble-libmpx --enable-plugin --enable-default-pie --with-system-zlib --with-target-syste
h-arch-32=i686 --with-abi=m64 --with-multilib-list=m32,m64,mx32 --enable-multilib --wi
ver --enable-checking=release --build=x86_64-linux-gnu --host=x86_64-linux-gnu --targe
Thread model: posix
gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)
ycj@ubuntu:~$ make -v
GNU Make 4.1
Built for x86_64-pc-linux-gnu
Copyright (C) 1988-2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

2. 使用命令`sudo apt-get install m4 flex bison`安装依赖库 m4、flex、bison

3. 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

4. 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

5. 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：

```
making links in test...
make[1]: Entering directory '/usr/local/src/openssl-1.0.0s/test'
make[1]: Nothing to be done for 'links'.
make[1]: Leaving directory '/usr/local/src/openssl-1.0.0s/test'
making links in tools...
make[1]: Entering directory '/usr/local/src/openssl-1.0.0s/tools'
make[1]: Nothing to be done for 'links'.
make[1]: Leaving directory '/usr/local/src/openssl-1.0.0s/tools'
generating dummy tests (if needed)...
make[1]: Entering directory '/usr/local/src/openssl-1.0.0s/test'
make[1]: Nothing to be done for 'generate'.
make[1]: Leaving directory '/usr/local/src/openssl-1.0.0s/test'

Configured for linux-x86_64.
```

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：

```
One common tool to check the dynamic dependencies of an executable
or dynamic library is ldd(1) on most UNIX systems.

See any operating system documentation and manpages about shared
libraries for your version of UNIX.  The following manpages may be
helpful: ld(1), ld.so(1), ld.so.1(1) [Solaris], dld.sl(1) [HP],
ldd(1), crle(1) [Solaris], pldd(1) [Solaris], ldconfig(8) [Linux],
chatr(1) [HP].
cp libcrypto.pc /usr/local/openssl/lib/pkgconfig
chmod 644 /usr/local/openssl/lib/pkgconfig/libcrypto.pc
cp libssl.pc /usr/local/openssl/lib/pkgconfig
chmod 644 /usr/local/openssl/lib/pkgconfig/libssl.pc
cp openssl.pc /usr/local/openssl/lib/pkgconfig
chmod 644 /usr/local/openssl/lib/pkgconfig/openssl.pc
```

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

6. 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
```

7. 执行以下指令，配置动态库软链接：

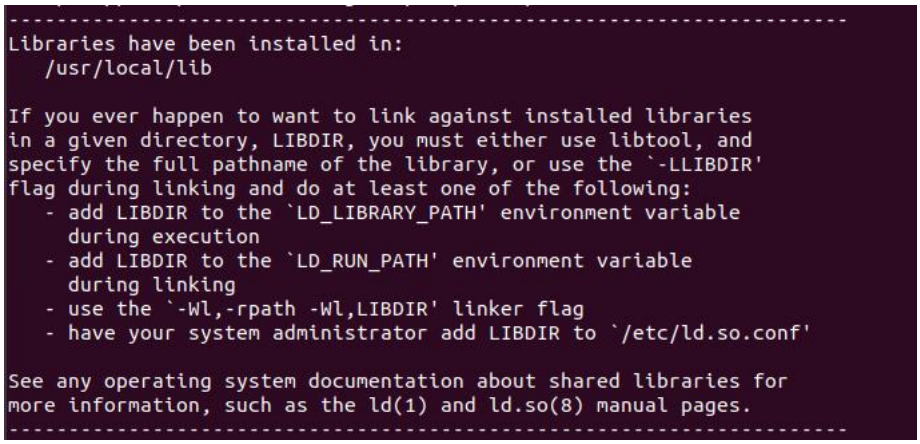
```
sudo ln -s /usr/local/openssl/lib/libssl.so.1.0.0 /usr/lib/libssl.so
sudo ln -s /usr/local/openssl/lib/libcrypto.so.1.0.0 /usr/lib/libcrypto.so
```

8. 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：

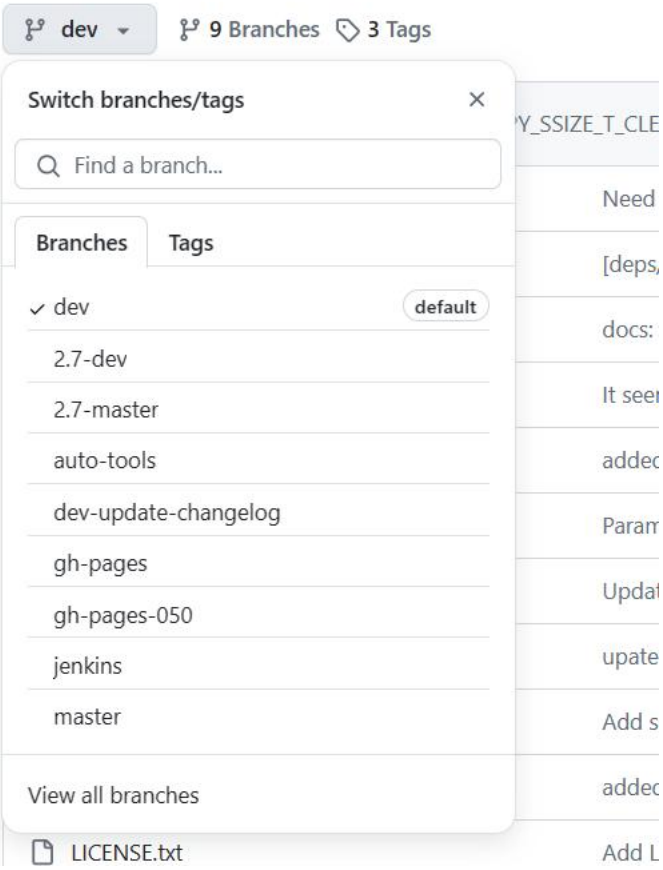
```
ycj@ubuntu:/usr/local/src/openssl-1.0.0s$ openssl version
OpenSSL 1.0.0s 11 Jun 2015
```

9. 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：

10. 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：



11. 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：

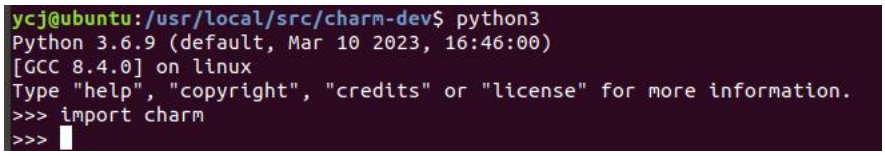


12. 使用命令`sudo mv ./charm-dev.zip /usr/local/src/`将压缩包移动/usr/local/src/下，然后使用命令`cd /usr/local/src/`切换目录，接着使用命令`sudo unzip ./charm-dev.zip`进行解压，使用命令`cd /usr/local/src/charm-dev/`进入 charm-dev 文件夹，输入命令`sudo ./configure.sh`来写入配置，接着输入命令`sudo make`进行编译，最后输入命令`sudo make install`进行安装，若终端出现下面的信息则安装成功：

```
Installed /usr/local/lib/python3.6/dist-packages/typing_extensions-4.11.0-py3.6.
egg
Searching for setuptools==39.0.1
Best match: setuptools 39.0.1
Adding setuptools 39.0.1 to easy-install.pth file
Installing easy_install script to /usr/local/bin

Using /usr/lib/python3/dist-packages
Finished processing dependencies for Charm-Crypto==0.50
ycj@ubuntu:/usr/local/src/charm-dev$
```

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

13. 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
```

2. 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
```

3. 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]
```

4. 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()]
```

5. 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)
```

6. 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
```

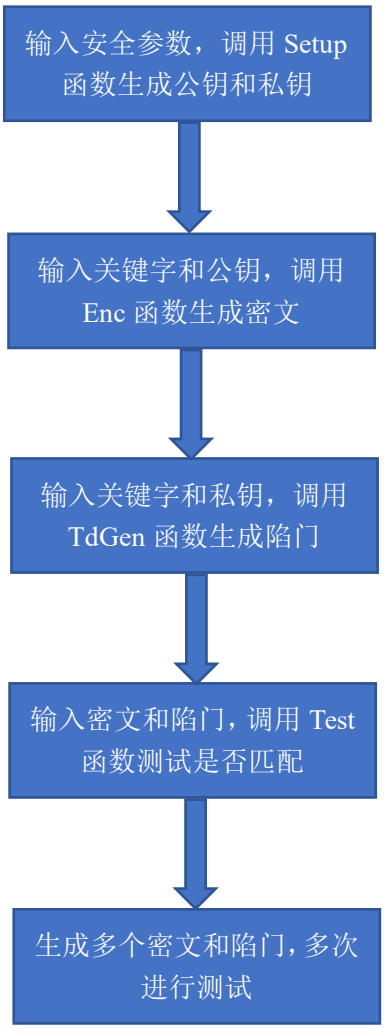
7. 编写主函数

```
# 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))
```

8. 代码流程图如下：



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：

```
ycj@ubuntu:~/Desktop/Experiment$ python3 main.py
hv:  a6105c0a611b41b08f1209506350279e
group.serialize(c2):  b'3:GNNux4jCkayiOZEaNmNuVCLvZ4PnG46MAiJo0bCxIfkki3/pMxtyh5RFFuKENwbmf6Bexb4lY0EcR+LSnkHmX6IfvRv+RlAvr10AFCkJKHad2SgDMxvU7eGxw/7Q9vPThl
x6xAK0Ba8ANyjMfGKXzno2bbv7wrkFCeapE3zhHhE='
hv:  a6105c0a611b41b08f1209506350279e
c2:  cf05a7aa47fda2a63308ae1356fef0158cf400bd68acedd30c21939bb47986a5
hv:  7fa3b767c460b54a2be4d49030b349c7
c2:  cf05a7aa47fda2a63308ae1356fef0158cf400bd68acedd30c21939bb47986a5
hv:  7cc314e49fab446daf87c56184a11159
group.serialize(c2):  b'3:mZikeMwAd2AwI3dJh0NOPL6neGsYRCeMUD81UALzi89bAde4c4LUw2U8aQNij9Wv25/NRPKmiISlUZmrsT1/5AoKb+sXwboNKsG7WexkIZYWNCYZBNJacUsMnDWBH3s+zy
TDjtcsE8vLMbAdiwNYsaBMWlFVzDUNHpZOjcghv5k='
c2:  21c139b2f8349ff2bbeb5a70d23135dcbc1888951347e6d727ab00c8f164436b
hv:  7fa3b767c460b54a2be4d49030b349c7
group.serialize(c2):  b'3:RG3j402hlsrgWRW3UCDCj5oAYIWvGxAEHbTtSX2Fdn80jYI5RME076U/DQVVCmM03AuhFrSQRU3lFicvO3ZtkgKsmhqH/UwOCeusItpIYPZcEnlbGG/chv2MJUxopaCEKq
99oKlTgXIkiMolf5mgoxJdnIlka5G1jxDclQ6kpw0='
c2:  37bbc80bdcc6560ade78615d6c5b8c61c31e075b36022c3e4061f933fcef4c6e
hv:  363fe582a68dcbde9181446ceac13c8a
group.serialize(c2):  b'3:Ti8QlU+1hb9xSfKa+JW+SO+u1mcFbzxwNP5BMHJEc+vqDXfcPRV16BmuTfUhQFbNZHMAb+DJW3p58J8ZzLI1Z0c7uaj8nuhiktNMsRMJOJVqRqmmBwuu+v8LE4tpgp52Qd
ZOF4Cv4FkYCARrNVY3U9whXuIkAGu7y9HSy94HbqA='
hv:  363fe582a68dcbde9181446ceac13c8a
c2:  efc358237f8dfd2362b0d6c382898f20ac42b0e254847146045cfb93b1ebaedd
hv:  b5f7ba7fc154d3fb6510f4f04cfbc6af
c2:  efc358237f8dfd2362b0d6c382898f20ac42b0e254847146045cfb93b1ebaedd
ycj@ubuntu:~/Desktop/Experiment$
```

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
```

七、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。