**非对称可搜索加密**

**一、基础知识**

**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。

**二、实验步骤**

思考问题：如何实现Boneh的第一个PEKS方案？

密码工具库：PEKSBoneh2004方案用到双线性对，这里推荐一个实用的python密码工具库charm-crypto，该工具库由约翰斯·霍普金斯大学科研人员开发。

项目地址：http://charm-crypto.io/

PEKSBoneh2004方案代码实现如下：

|  |
| --- |
| #coding=utf-8  from charm.toolbox.pairinggroup import PairingGroup, ZR, G1, G2, GT, pair  import hashlib  Hash1pre = hashlib.md5  def Hash1(w):  hv = Hash1pre(str(w).encode('utf8')).hexdigest()  print(hv)  hv = group.hash(hv, type=G1)  return hv  Hash2 = hashlib.sha256  def Setup(param\_id='SS512'):  group = PairingGroup(param\_id)  g = group.random(G1)  alpha = group.random(ZR)  sk = group.serialize(alpha)  pk = [group.serialize(g), group.serialize(g \*\* alpha)]  return [sk, pk]  def Enc(pk, w, param\_id='SS512'):  group = PairingGroup(param\_id)  g, h = group.deserialize(pk[0]), group.deserialize(pk[1])  r = group.random(ZR)  t = pair(Hash1(w), h \*\* r)  c1 = g \*\* r  c2 = t  print(group.serialize(c2))  return [group.serialize(c1), Hash2(group.serialize(c2)).hexdigest()]  def TdGen(sk, w, param\_id='SS512'):  group = PairingGroup(param\_id)  sk = group.deserialize(sk)  td = Hash1(w) \*\* sk  return group.serialize(td)  def Test(td, c, param\_id='SS512'):  group = PairingGroup(param\_id)  c1 = group.deserialize(c[0])  c2 = c[1]  print(c2)  td = group.deserialize(td)  return Hash2(group.serialize(pair(td, c1))).hexdigest() == c2  if \_\_name\_\_ == '\_\_main\_\_':  param\_id = 'SS512'  [sk, pk] = Setup(param\_id)  group = PairingGroup(param\_id)  c = Enc(pk, "yes")  td = TdGen(sk, "yes")  assert(Test(td, c))  td = TdGen(sk, "no")  assert(not Test(td, c))  c = Enc(pk, "Su\*re")  assert(not Test(td, c))  c = Enc(pk, "no")  assert(Test(td, c))  c = Enc(pk, 9 \*\* 100)  td = TdGen(sk, 9 \*\* 100)  assert(Test(td, c))  td = TdGen(sk, 9 \*\* 100 + 1)  assert(not Test(td, c)) |

**三、实验要求**

**实验时间：**2周

**报告要求：**

1. 实验内容说明（简述实验做了些什么）

2. 实验代码分析（针对实验给出的代码，具体分析各关键函数的作用及如何实现的，给出代码流程图）

2. 实验步骤及截图

3. 实验结果分析

4. 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。

**提交方式：**通过雨课堂提交