背包加密

背包问题

首先，我们先来介绍一下背包问题，假定一个背包可以称重 W，现在有 n 个物品，其重量分别为 $a_1, a_2, \ldots, a_n$ 我们想问一下装哪些物品可以恰好使得背包装满，并且每个物品只能被装一次。这其实就是在解这样的一个问题

$$x_1a_1 + x_2a_2 +, \ldots, +x_na_n = W$$

其中所有的 x_i 只能为 0 和 1。显然我们必须枚举所有的 n 个物品的组合才能解决这个问题，而复杂度也就是 2^n，这也就是背包加密的妙处所在。

在加密时，如果我们想要加密的明文为 x，那么我们可以将其表示为 n 位二进制数，然后分别乘上 a_i 即可得到加密结果。

但是解密的时候，该怎么办呢？我们确实让其他人难以解密密文，但是我们自己也确实没有办法解密密文。

但是当 a_i 是超递增的话，我们就有办法解了，所谓超递增是指序列满足如下条件

$$a_i > \sum_{k=1}^{i-1} a_k$$

即第 i 个数大于前面所有数的和。

为什么满足这样的条件就可以解密了呢？这是因为如果加密后的结果大于 a_n 的话，其前面的系数为必须 1 的。反之，无论如何也无法使得等式成立。因此，我们可以立马得到对应的明文。

但是，这样又出现了一个问题，由于 a_i 是公开的，如果攻击者截获了密文，那么它也就很容易去破解这样的密码。为了弥补这样的问题，就出现了 Merkle–Hellman 这样的加密算法，我们可以使用初始的背包集作为私钥，变换后的背包集作为公钥，再稍微改动加密过程，即可。

这里虽然说了超递增序列，但是却没有说是如何生成的。

Merkle–Hellman

公私钥生成

生成私钥

私钥就是我们的初始的背包集，这里我们使用超递增序列，怎么生成呢？我们可以假设 $a_1 = 1$，那么 a_2 大于 1 即可，类似的可以依次生成后面的值。

生成公钥

在生成公钥的过程中主要使用了模乘的运算。

首先，我们生成模乘的模数 m，这里要确保

$$m > \sum_{i=1}^{i=n} a_i$$

其次，我们选择模乘的乘数 w，作为私钥并且确保

$$gcd(w, m) = 1$$

之后，我们便可以通过如下公式生成公钥

$$b_i \equiv wa_i \bmod m$$

并将这个新的背包集 b_i 和 m 作为公钥。

加解密

加密

假设我们要加密的明文为 v，其每一个比特位为 v_i，那么我们加密的结果为

$$\sum_{i=1}^{i=n} b_i v_i \bmod m$$

解密

对于解密方，首先可以求的 w 关于 m 的逆元 w^{-1}。

然后我们可以将得到的密文乘以 w^{-1} 即可得到明文，这是因为

$$\sum_{i=1}^{i=n} w^{-1} b_i v_i \bmod m = \sum_{i=1}^{i=n} a_i v_i \bmod m$$

这里有

$$b_i \equiv w a_i \bmod m$$

对于每一块的加密的消息都是小于 m 的，所以求得结果自然也就是明文了。

破解

该加密体制在提出后两年后该体制即被破译，破译的基本思想是我们不一定要找出正确的乘数 w（即陷门信息），只需找出任意模数 `m'` 和乘数 `w'`，只要使用 `w'` 去乘公开的背包向量 B 时，能够产生超递增的背包向量即可。

例子

这里我们以 2014 年 ASIS Cyber Security Contest Quals 中的 Archaic 为例，题目链接 [https://github.com/ctfs/write-ups-2014/tree/b02bcbb2737907dd0aa39c5d4df1d1e270958f54/asis-ctf-quals-2014/archaic]。

首先查看源程序

```
secret = 'CENSORED'
msg_bit = bin(int(secret.encode('hex'), 16))[2:]
```

首先得到了 secret 的所有二进制位。

其次，利用如下函数得到 keypair，包含公钥与私钥。

```
keyPair = makeKey(len(msg_bit))
```

仔细分析 makekey 函数，如下

```
def makeKey(n):
    privKey = [random.randint(1, 4**n)]
    s = privKey[0]
    for i in range(1, n):
        privKey.append(random.randint(s + 1, 4**(n + i)))
        s += privKey[i]
    q = random.randint(privKey[n-1] + 1, 2*privKey[n-1])
    r = random.randint(1, q)
    while gmpy2.gcd(r, q) != 1:
        r = random.randint(1, q)
    pubKey = [ r*w % q for w in privKey ]
    return privKey, q, r, pubKey
```

可以看出 prikey 是一个超递增序列，并且得到的 q 比 prikey 中所有数的和还要大，此外我们得到的 r，恰好与 q 互素，这一切都表明了该加密是一个背包加密。

果然加密函数就是对于消息的每一位乘以对应的公钥并求和。

```
def encrypt(msg, pubKey):
    msg_bit = msg
    n = len(pubKey)
    cipher = 0
    i = 0
```

```
    for bit in msg_bit:
        cipher += int(bit)*pubKey[i]
        i += 1
    return bin(cipher)[2:]
```

对于破解的脚本我们直接使用 GitHub [https://github.com/ctfs/write-ups-2014/tree/b02bcbb2737907dd0aa39c5d4df1d1e270958f54/asis-ctf-quals-2014/archaic] 上的脚本。进行一些简单的修改。

```
import binascii
# open the public key and strip the spaces so we have a decent array
fileKey = open("pub.Key", 'rb')
pubKey = fileKey.read().replace(' ', '').replace('L', '').strip('[]').split(',')
nbit = len(pubKey)
# open the encoded message
fileEnc = open("enc.txt", 'rb')
encoded = fileEnc.read().replace('L', '')
print "start"
# create a large matrix of 0's (dimensions are public key length +1)
A = Matrix(ZZ, nbit + 1, nbit + 1)
# fill in the identity matrix
for i in xrange(nbit):
    A[i, i] = 1
# replace the bottom row with your public key
for i in xrange(nbit):
    A[i, nbit] = pubKey[i]
# last element is the encoded message
A[nbit, nbit] = -int(encoded)

res = A.LLL()
for i in range(0, nbit + 1):
    # print solution
    M = res.row(i).list()
    flag = True
    for m in M:
        if m != 0 and m != 1:
            flag = False
            break
    if flag:
        print i, M
        M = ''.join(str(j) for j in M)
        # remove the last bit
        M = M[:-1]
        M = hex(int(M, 2))[2:-1]
        print M
```

输出之后再解码下

```
295 [1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1,
1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1,
0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0,
0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1,
0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,
0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0,
1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0]
415349535f396264336435666432343232363832633139353638383036613037303631636 5
>>> import binascii
>>> binascii.unhexlify('415349535f396264336435666432343232363832633139353638383036613037303631636 5')
'ASIS_9bd3d5fd2422682c19568806a07061ce'
```

需要注意的是，我们得到的 LLL 攻击得到的矩阵 res 的只包含 01 值的行才是我们想要的结果，因为我们对于明文加密时，会将其分解为二进制比特串。此外，我们还需要去掉对应哪一行的最后一个数字。

flag 是 `ASIS_9bd3d5fd2422682c19568806a07061ce`。

题目

- 2017 国赛 classic
