GGH非对称密码体制破解方法

原创

M3ng@L 于 2022-02-27 16:53:26 发布 137 收藏

分类专栏： 密码学知识总结 文章标签： Crypto GGH

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。

本文链接：https://blog.csdn.net/qq_51999772/article/details/123166472

版权

密码学知识总结 专栏收录该内容

18 篇文章 2 订阅

订阅专栏

GGH非对称密码体制破解方法

Encryption

Decryption

Attack

Example

Perference

GGH密码体制，基于格的CVP（最近接向量问题）设计的非对称密码算法

Encryption

- 生成密钥：

选择幺模矩阵U作为私钥

公钥：$B = U \cdot B$；其中B是格L中的基

加密公式：$c =$ _

其中

- m：由明文构成的一个`1×n`向量

> 类似于这样的向量：m = [1 2 3 4 5 6 7 8 9 10]

B：`n×n`矩阵，相当于公钥

e：`1×n`向量，属于干扰向量；有的GGH加密e每项都是3或者-3，有的GGH加密的e所包含的数值不只是3或者-3

c：密文向量，也就是`1×n`

Decryption

$c \cdot B$ = (m ... B ...

利用`Babai rounding technique`去除e. B . CVP - CTF Wiki (ctf-wiki.org)

（由于官方解密不是重点，就尽快带过）

最后$m=$

Attack

- *Nguyen s Attack*

`Nguyen's Attack`进行的前提条件是e必须是由3和-3构成

令$s=$ (3 3 ... 3)$\cdot$ ...

在加密等式两边同时加上s，得到

$c+$.

等式两边同取模6，这样$(e+s)$会等于0，得到

$c \equiv$ ...

得到明文取模6后的结果，也就是说，得到的是部分明文内容

- *embedded technique*

`embedded technique`简单来说，就是把c作为一组向量嵌入B中去，进而转换为`SVP`问题，直接使用封装函数求解SVP问题，得到的结果映射到原来的维度中就等于干扰向量e，在c中直接减去得到的干扰向量结果再乘以B的逆即可

原来的公钥向量B是`n×n`向量，这时把c作为行向量嵌套进去并且加入一个$(0,0,...,0,1)$的列向量，得到一个`(n+1)×(n+1)`的向量

```
temp = B.stack(c).augment(vector([0]*B.ncols()+[1]))
```

得到的$temp$向量是

$n+1$维度的，该向量作为$n+1$维的格的基，求解这个晶格的最小向量（SVP问题），而得到的

```
from sage.modules.free_module_integer import IntegerLattice
e = IntegerLattice(temp).shortest_vector()[:-1]
```

得到结果

```
m = B.solve_left(c - e)
```

完整代码（来自于hxp | VolgaCTF 2016 Quals: crypto300 “XXY” writeup）

```
from sage.modules.free_module_integer import IntegerLattice

B =
c =

B = matrix(B)
c = matrix(c)
temp = B.stack(c).augment(vector([0]*B.ncols()+[1]))
e = IntegerLattice(B).shortest_vector()[:-1]
m = B.solve_left(c - matrix(e))
```

Example

- 陇原战"疫"2021网络安全大赛_easytask

```
def get_random_U(n):
    def get_U1():
        A = Matrix(ZZ, n, n)
        for i in range(0,n):
            for j in range(0,n):
                if i<j:
                    A[i,j] = random.randint(1,1000)
                if i==j:
                    A[i,j] = 1
        return A
    def get_U2():
        A = Matrix(ZZ, n, n)
        for i in range(0,n):
            for j in range(0,n):
                if i>j:
                    A[i,j] = random.randint(1,1000)
                if i==j:
                    A[i,j] = 1
        return A
    return get_U1()*get_U2()
def get_public_key():
    U = get_random_U(9)
    V = matrix(V)
    W = V*U
    return W
def get_random_r():
    n = 9
    delta = 4
    r = random_vector(ZZ, n, x=-delta+1, y=delta)
    r = matrix(r)
    return r
def encrypt():
    M = [getPrime(10)for i in range(9)]
    m = matrix(M)
    W = get_public_key()
    r = get_random_r()
    e = m*W+r
    print("e =",e)
    print("W =",W)
    return M
def new_encrypt():
    M = encrypt()
    key = hashlib.sha256(str(M).encode()).digest()
    cipher = AES.new(key, AES.MODE_ECB)
    c = cipher.encrypt(flag).hex()
    print("c =",c)
new_encrypt()
```

该题目给出的e和W相当于GGH加密体制中的c和B，下面就按照GGH加密体制的习惯来进行解密

已知c,B，在GGH中求得的m作为AES的key来解密

按照`embedded technique`的方式来求解GGH中的m

```
# sagemath
from sage.modules.free_module_integer import IntegerLattice
c = e = [151991736758354,115130361237591,58905390613532,130965235357066,74614897867998,48099459442369,458944857829
43,7933340009592,25794185638]
B = W = [[-10150241248,-11679953514,-8802490385,-12260198788,-10290571893,-334269043,-11669932300,-2158827458,-7021995],
[52255960212,48054224859,28230779201,43264260760,20836572799,8191198018,14000400181,4370731005,14251110],
[2274129180,-1678741826,-1009050115,1858488045,978763435,4717368685,-561197285,-1999440633,-6540190],
[45454841384,34351838833,19058600591,39744104894,21481706222,14785555279,13193105539,2306952916,7501297],
[-16804706629,-13041485360,-8292982763,-16801260566,-9211427035,-4808377155,-6530124040,-2572433293,-8393737],
[28223439540,19293284310,5217202426,27179839904,23182044384,10788207024,18495479452,4007452688,13046387],
[968256091,-1507028552,1677187853,8685590653,9696793863,2942265602,10534454095,2668834317,8694828],
[33556338459,26577210571,16558795385,28327066095,10684900266,9113388576,2446282316,-173705548,-577070],
[35404775180,32321129676,15071970630,24947264815,14402999486,5857384379,10620159241,2408185012,7841686]]

c = matrix(c)
B = matrix(B)
temp = B.stack(c).augment(vector([0]*B.ncols()+[1]))
e = IntegerLattice(temp).shortest_vector()[:-1]
m = B.solve_left(c - matrix(e))
print(m)
```

得到的$_m$代入AES作为key解密即可

```
from Crypto.Util.number import *
from Crypto.Cipher import AES
import hashlib
c = 0x1070260d8986d5e3c4b7e672a6f1ef2c185c7fff682f99cc4a8e49cfce168aa0
m = [877,619,919,977,541,941,947,1031,821]
key = hashlib.sha256(str(m).encode()).digest()
aes = AES.new(key,AES.MODE_ECB)
flag = aes.decrypt(long_to_bytes(c))
print(flag)
```

Perference

hxp | VolgaCTF 2016 Quals: crypto300 “XXY” writeup

Intended Solution to GGH in GYCTF 2020 | Soreat_u’s Blog (soreatu.com)

GGH | 4XWi11的博客