## 9.篱笆墙的影子（栅栏编码）

打开题目给出一段编码：

felhaagv{ewtehtehfilnakgw}

其形式极像flag，根据题目名称可知使用了栅栏编码，将编码放到随波逐流CTF工具中，在分为13栏情况下发现flag：

flag{wethinkwehavetheflag}

## 10.Rabbit

打开题目，给出一段编码：

U2FsdGVkX1/+ydnDPowGbjjJXhZxm2MP2AgI

根据题目提示使用了rabbit编码，将编码放到在线rabbit解码工具上，得到的解码加上flag形式即可：

flag{Cute\_Rabbit}

## 11.RSA（已知pqe求d）

打开题目得到几段信息，我的电脑上又显示不全，查阅资料可知正确的信息为：

在一次RSA密钥生成中，假设p=473398607161，q=4511491，e=17

求解出d作为flag提交

注意：题目要求得到的 flag 请 将 noxCTF 替换为 flag ，格式为 flag{} 提交。

在python编译器中输入以下代码：

```shell

p=473398607161

q=4511491

e=17

d=((p-1)\*(q-1)+1)//e

print(d)

```

执行后得到结果“125631357777427553”，加上flag形式即可得到flag：

flag{125631357777427553}

## 12.丢失的MD5（代码补全：编码转换）

打开题目，给出了一个py文件，其代码加上注释为：

```shell

import hashlib  # 导入hashlib库用于计算MD5哈希值

# 遍历所有可能的ASCII可打印字符（十进制32-126对应空格到~）

for i in range(32, 127):

    for j in range(32, 127):

        for k in range(32, 127):

            # 创建MD5哈希对象

            m = hashlib.md5()

            # 构建待哈希的字符串：TASC + 第一个猜测字符 + O3RJMV + 第二个猜测字符 + WDJKX + 第三个猜测字符 + ZM

            # 注意：Python 3中update()需要bytes类型，原代码在Python 3会报错，这里假设运行在Python 2

            m.update('TASC' + chr(i) + 'O3RJMV' + chr(j) + 'WDJKX' + chr(k) + 'ZM')

            # 计算MD5哈希并转换为十六进制字符串

            des = m.hexdigest()

            # 检查哈希结果是否同时包含特定子串

            if 'e9032' in des and 'da' in des and '911513' in des:

                print des  # 输出符合条件的哈希值

```

可见这是一道代码补全题，代码的功能为找到三个 ASCII 字符（范围 32-126），使得它们插入到固定字符串模板 TASC{}O3RJMV{}WDJKX{}ZM 中后，计算出的 MD5 哈希值 同时包含子串 e9032、da 和 911513。

仔细阅读代码后发现代码的功能已经完善，关键在于缺少编码转换，可使用encode('utf-8') 将字符串转为 bytes，确保 Python 3 下 MD5 计算正确，补全后的代码为：

```shell

import hashlib

for i in range(32,127):

    for j in range(32,127):

        for k in range(32,127):

            m=hashlib.md5()

            m.update('TASC'.encode('utf-8')+chr(i).encode('utf-8')+'O3RJMV'.encode('utf-8')+chr(j).encode('utf-8')+'WDJKX'.encode('utf-8')+chr(k).encode('utf-8')+'ZM'.encode('utf-8'))

            des=m.hexdigest()

            if 'e9032' in des and 'da' in des and '911513' in des:

                print (des)

```

在python编译器上运行补全后的代码，得到的结果加上flag形式即可得到flag：

flag{e9032994dabac08080091151380478a2}