CTF-web Xman-2018 第三天 crypto-RSA

原创

iamsongyu 于 2018-12-06 11:56:50 发布 457 收藏 1
分类专栏： 理论知识 CTF 文章标签： CTF 密码学

本文链接：https://blog.csdn.net/iamsongyu/article/details/84850183


理论知识 同时被 2 个专栏收录
109 篇文章 7 订阅
订阅专栏


CTF
35 篇文章 12 订阅
订阅专栏

RSA

这个算法先不多说，在密码学的学习中，肯定是少不了数学，要想弄明白一个密码，基本的数学原理还是必须要懂得，其实每个步骤的大致原理必须懂，有关数学原理公式推导可以暂时不要深究，但是涉及算法步骤的必须懂。RSA作为一个非常火热的加密算法，其大致的原理还是较为容易理解，而攻击的方法也只需要学习一下大手子们写的具体介绍和实例也是可以接受的。

在该算法中个，背后的依靠是大素数分解的难题，公钥和私钥的使用使得发送信息方送到另一方的信息可以被还原，注意着其中私钥和公钥的产生，在RSA中这是关键，两者是一对，可以对同一信息进行加密解密，那么二者的产生一定遵循某种规律或者说是公式，只要理解了两个的产生和加密解密，RSA算法也就明了了。

RSA加密基本原理

加密过程

选择两个大素数p和q，计算出模数N = p * q

计算φ = (p−1) * (q−1) 即N的欧拉函数，然后选择一个e (1<e<φ)，且e和φ互质

取e的模反数为d，计算方法: e * d ≡ 1 (mod φ)

对明文A进行加密：`B≡A^e (mod n)` 或 `B = pow(A,e,n)`，得到的B即为密文

对密文B进行解密，`A≡B^d( mod n)` 或 `A = pow(B,d,n)`，得到的A即为明文

> p 和 q：大整数N的两个因子（factor）
>
> N：大整数N，我们称之为模数（modulus）
>
> e 和 d：互为模反数的两个指数（exponent）
>
> c 和 m：分别是密文和明文，这里一般指的是一个十进制的数

一般有如下称呼：

> （N，e）：公钥
>
> （N，d）：私钥

基本原理，公钥与私钥的产生

1. 随机选择两个不同大质数 p 和 q，计算 N=p×q
2. 根据欧拉函数，求得 r=φ(N)=φ(p) φ(q)=(p−1)(q−1)
3. 选择一个小于 r的整数 e，使 e 和 r 互质。并求得 e 关于 r 的模反元素，命名为 d，有 ed≡1(modr)
4. 将 p 和 q 的记录销毁

此时，(N,e) 是公钥，(N,d) 是私钥。

消息加密

首先需要将消息 m 以一个双方约定好的格式转化为一个小于 N，且与 N 互质的整数n。如果消息太长，可以将消息分为几段，这也就是我们所说的块加密，后对于每一部分利用如下公式加密得到密文c：

n**e ≡ c mod N

消息解密，利用密钥 d 进行解密得到明文n：

c**d ≡ n (mod N)

模不互素

攻击原理

当存在两个公钥的 N 不互素时，我们显然可以直接对这两个数求最大公因数，然后直接获得 p，q，进而获得相应的私钥。

SCTF RSA2

这里我们以 SCTF rsa2 为例进行介绍。直接打开 pcap 包，发现有一堆的消息，包含 N 和 e，然后试了试不同的 N 是否互素

```
import gmpy2
n1 = 20823369114556260762913588844471869725762985812215987993867783630051420241057912385055482788016327978
n2 = 19083821613736429958432024980074405375408953269276839696319265596855426189256865650651460460079819368
print gmpy2.gcd(n1, n2)
```

结果发现竟然不互素。

```
➜  scaf-rsa2 git:(master) ✗ python exp.py
```

```
122281872221091773923842091258531471948886120336284482555605167683829690073110898673260712865021244633908 98
```

那么我们就可以直接来解密了，这里我们利用第一对公钥密码，获得私钥用来解密密文。代码如下

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_v1_5, PKCS1_OAEP
import gmpy2
from base64 import b64decode

n1 = 208233691145562607629135888444718697257629858122159879938677836300514202410579123850554827880163279784
n2 = 190838216137364299584320249800744053754089532692768396963192655968554261892568656506514604600798193689
p1 = gmpy2.gcd(n1, n2)  # 最大公因子 也就是p
q1 = n1 / p1   # 另一个分解的q
e = 65537
phin = (p1 - 1) * (q1 - 1)   # 得到  r
d = gmpy2.invert(e, phin)  # 根据r e 得到私钥d  选择一个小于r的数e 计算模反元素d 即为私钥的一部分
cipher = 0x68d5702b70d18238f9d4a3ac355b2a8934328250efd4efda39a4d750d80818e6fe228ba3af471b27cc529a4b0bef70a2
plain = gmpy2.powmod(cipher, d, n1)  # 密文 私钥, 素数n
plain = hex(plain)[2:]  # 去掉0x
if len(plain) % 2 != 0:   #补全16进制数据，因为字符转16进制必为偶数位
    plain = '0' + plain
print plain.decode('hex')
```

散列哈希函数

散列值的目的如下

- 确保消息的完整性，即确保收到的数据确实和发送时的一样（即没有修改、插入、删除或重放），防止中间人篡改。
- 冗余校验
- 单向口令文件，比如linux系统的密码
- 入侵检测和病毒检测中的特征码检测

目前的Hash函数主要有MD5，SHA1，SHA256，SHA512。目前的大多数hash函数都是迭代性的，即使用同一个hash函数，不同的参数进行多次迭代运算。

算法类型	输出 Hash 值长度
MD5	128 bit / 256 bit
SHA1	160 bit
SHA256	256 bit
SHA512	512 bit

MD5

MD5的输入输出如下

- 输入：任意长的消息，512比特长的分组。
- 输出：160比特的消息摘要。

关于详细的介绍，请自行搜索。

此外，有时候我们获得到的md5是16位的，其实那16位是32位md5的长度，是从32位md5值来的。是将32位md5去掉前八位，去掉后八位得到的。

一般来说，我们可以通过函数的初始化来判断是不是MD5函数。一般来说，如果一个函数有如下四个初始化的变量，可以猜测该函数为MD5函数，因为这是MD5函数的初始化Ⅳ。

```
0x67452301，0xEFCDAB89，0x98BADCFE，0x10325476
```

```
目前可以说md5已经基本被攻破了，一般的MD5的碰撞都可以在如下网上获取到
```

- http://www.cmd5.com/
- http://www.ttmd5.com/
- http://pmd5.com/

SHA1

基本描述

SHA1的输入输出如下

- 输入：任意长的消息，分为 **512 比特**长的分组。首先在消息右侧补比特 1，然后再补若干个比特 0，直到消息的比特长度满足对 512 取模后余数是 448，使其与 448 模 512 同余。
- 输出：160 比特的消息摘要。

关于详细的介绍，请自行搜索。

一般来说，我们可以通过函数的初始化来判断是不是 SHA1 函数。一般来说，如果一个函数有如下五个初始化的变量，可以猜测该函数为 SHA1 函数，因为这是 SHA1 函数的初始化Ⅳ。

```
0x67452301
0xEFCDAB89
0x98BADCFE
0x10325476
0xC3D2E1F0
```

```
前面四个与 MD5 类似，后面的是新加的。
```

攻击

```
哈希长度拓展攻击（hash length extension attacks）
```

哈希长度扩展攻击(Hash Length Extension Attacks)是指针对某些允许包含额外信息的加密散列函数的攻击手段。该攻击适用于在**消息与密钥的长度已知**的情形下，所有采取了 H(key // message) 此类构造的散列函数。MD5和SHA-1 等基于 Merkle–Damgård 构造的算法均对此类攻击显示出脆弱性。

这类哈希函数有以下特点

- 消息填充方式都比较类似，首先在消息后面添加一个1，然后填充若干个0，直至总长度与 448 同余，最后在其后附上64位的消息长度（填充前）。
- 每一块得到的链接变量都会被作为下一次执行hash函数的初始向量Ⅳ。在最后一块的时候，才会将其对应的链接变量转换为hash值。

一般攻击时应满足如下条件

- 我们已知 key 的长度，如果不知道的话，需要爆破出来
- 我们可以控制 message 的消息。
- 我们已经知道了包含 key 的一个消息的hash值。

这样我们就可以得到一对(messge,x)满足x=H(key // message)虽然我们并不清楚key的内容。

攻击原理

这里不妨假设我们我们知道了 hash(key+s) 的 hash 值，其中 s 是已知的，那么其本身在计算的时候，必然会进行填充。那么我们首先可以得到 key+s 扩展后的字符串 now，即

now=key|s|padding

那么如果我们在 now 的后面再次附加上一部分信息extra，即

key|s|padding|extra

这样再去计算hash值的时候，

1. 会对 extra 进行填充直到满足条件。
2. 先计算 now 对应的链接变量 IV1，而我们已经知道这部分的 hash 值，并且链接变量产生 hash 值的算法是可逆的，所以我们可以得到链接变量。
3. 下面会根据得到的链接变量 IV1，对 extra 部分进行哈希算法，并返回hash值。

那么既然我们已经知道了第一部分的 hash 值，并且，我们还知道 extra 的值，那么我们便可以得到最后的hash值。

而之前我们也说了我们可以控制 message 的值。那么其实 s，padding，extra 我们都是可以控制的。所以我们自然可以找到对应的(message,x)满足x=hash(key|message)。