Crypto_P4

MST 补充

部分问题补充

CRT

对RSA的攻击

直接分解

- 是一个困难问题
- 目前最高768bit
- http://www.factordb.com/

参数选取不当导致的分解

- p,q 过近 -- 开方寻找
  - 利用pq两个数字都在根号n附近
- 重用因子 -- gcd分解
  - n1 = p1*p2
  - n2 = p1*p3
  - gcd(n1, n2) = p1
  - p1|n1， p2 = n1//p1

明文枚举攻击(A2考)*

- 明文空间较小，枚举明文
  - e,d,n安全，但是m空间较小
  - 攻击者具有加密所需所有内容，除了明文
  - 攻击者枚举所有可能的明文，尝试加密，如果与已知密文相同，则可以确定明文

大数 mod-exp

```
def my_pow(a,b,c):
    t = 1
    for i in range(b):
        t *= a
        t %= c
    return t % c
# for bit in b:
#   if bit == 1:
#       t *= m
#    m *= m
#a^b
#b个a相乘
```

```
#....k2*2^2+k1*2^1+k0*2^0
#(a^(ki*2^i))*(a^(kj*2^j))...

def my_fast_pow(a,b,c):
    t = 1
    for bit in bin(b)[2:][::-1]:
        if bit == '1':
            t *= a
            t %= c
        a *= a
        a %= c
    return t % c
```

低加密指数攻击

- 明文过小直接开方密文
- m相同，e相同，N互素，CRT攻击。

```
老师 T
学生 A
学生 B
学生 C
------
第一次
------
T 想要发同一消息给 A 和 B,分别使用 A B的公钥加密。
正常情况:
    A的公钥:  e1, n1
    A的私钥:  d1
    B的公钥:  e2, n2
    B的私钥:  d2
    T 发送给 A: c1 = pow(m, e1, n1)
    T 发送给 B: c2 = pow(m, e2, n2)
问题场景:
    A、B使用了同一个n(共模)。
    T 发送给 A: c1 = pow(m, e1, n)
    T 发送给 A: c2 = pow(m, e2, n)

gcd(e1, e2) == 1
可以找到一组s和t 满足: s*e1 + t*e2 == gcd(e1, e2) == 1

攻击者C:
    已知: e1, e2, n, c1, c2
        找到满足要求的 s 和 t
        计算C1 = c1^s
        计算C2 = c2^t
        C1*C2 = c1^s*c2^t = m ^ (s*e1 + t*e2) = m^1 = m (mod n)
    攻击者不需要知道d1或是d2，也不需要分解n，即可知道m
------
第二次
------
```

```
T 想要发同一消息给 A 和 C,分别使用 A C的公钥加密。
B 可以共模攻击获取消息。
------
第三次
------
T 只发一个消息给 A, 使用 A 的公钥加密。
对于A: 持有 e1, d1, n, c1, 可以完成解密。
对于B: 持有 e2, d2, n, c1。
B 开展如下攻击:
    计算 kfn = e2*d2 - 1
    轻易获取 e1
    计算 d11 = invert(e1, kfn)
    计算 m = pow(c1, d11, n)
    因为d11 = invert(e1, kfn)
    所以 e1*d11 = k2*kfn +1 = (k2*k1)fn + 1
    pow(c1, d11, n) = pow(m, (k1*k2)*fn + 1, n) = m
```

共模攻击(A2考)*

m相同，N相同，使用互素的e发给不同用户，则不需要分解N即可还原。

已知ed解密他人消息(A2考)*

使用kfn和e计算d

```
d = invert(e, kfn)
e*d % kfn = 1
e*d % fn = 1
```

已知ed分解n(也可以作为A2的解法)

```
e*d = k*fn+1
K = e*d - 1 = k*fn
pow(g, K, n) == 1
pow(g, K, n) == 1
pow(g, K, p) == 1
pow(g, K, q) == 1
不断对K //= 2, 直到pow(g, K, n) 不为 1 也不为 n-1.
然后取GCD(pow(g,K,n)+1, n)完成分解。
分解后即可解密他人信息。

备注: 需要写参考。
```

p、q光滑时的攻击

如果p-1 q-1会被分解成很多小素数

由于

```
fn = (p-1)*(q-1)
;
只要将小素数相乘，则可以得到kfn
```

利用计算的性质，侧信道攻击

```
pow(m, d, n)
t = 1
for bit in d:
    if bit == 1:
        t *= m
    m *= m
```

DH 密钥交换

```
没有减法的世界：
1. 人人都会算加法，但是没人会减法
2. t + a + b = t + b + a

Alice Bob 通信
A 公开一个公共参数： 10
--- 攻击者（10）
A 选择自己的私钥：3 并计算公钥：10+3 = 13
B 选择自己的私钥：5 并计算公钥：10+5 = 15
A B 交换公钥
--- 攻击者（10, 13, 15）
A 计算共享密钥 15 + 3 = 18
B 计算共享密钥 13 + 5 = 18
共享密钥为 18
```

```
实际选取的运算：
A = g ^ a % p
正向计算：已知a，计算A是容易的；
反向计算：已知A，计算a是数学上的困难问题；
交换律：
(g ^ a % p) ^ b % p
= (g ^ a) ^ b % p
= (g ^ a*b) % p
= (g ^ b*a) % p
= (g ^ b % p) ^ a % p

公开信息：g, p
Alice
```

```
    私钥: a
    公钥: A = pow(g, a, p)
Bob
    私钥: b
    公钥: B = pow(g, b, p)

共享密钥: K = pow(B, a ,p) = pow(g, a*b, p) = pow(A, b, p)

g p A B

K
```

- MITM

```
  公开信息: g, p
  Alice
    私钥: a
    公钥: A = pow(g, a, p)
  Bob
    私钥: b
    公钥: B = pow(g, b, p)
  A --> B : A, g, p (a, A, g, p)
  B --> A : B (b, B, g, p, A) => [K = pow(A, b, p)]
  A : (a, A, g, p, B) => [K = pow(B, a, p)]

  ================================================
  user C
    私钥: c
  A --X B  --> C: A, g, p
    A: (a, A, g, p)
    B: (b)
    C: (A, g, p, c) => [C = pow(g, c, p), KA = pow(A, c, p)]
  C --> B: C, g, p
    A: (a, A, g, p)
    B: (b, g, p, C) => [B = pow(g, b, p), KB = pow(C, b, p)]
    C: (A, g, p, c, C)
  B --X A  --> C: B
    A: (a, A, g, p)
    B: (b, g, p, C, B)
    C: (A, g, p, c, C, B) => [KB = pow(B, c, p)]
  C --> A : C
    A: (a, A, g, p, C) => [KA = pow(C, a, p)]

  A-C : KA
  B-C : KB
  ====================
  m = (A,g,p)
  h = H(k||m)
  m1 = (C,g,p)
  h1 = H(k||m1) k是攻击者不知道的
  -------
  m1 || h
```

```
-------
m1
h1 = H(k||m1)
h1 != h
```

- 引入认证
    - H(k||m)
        - 长度扩展攻击 https://github.com/bwall/HashPump
    - HMAC : H(K xor OP || H(K xor IP) || M)
- 为什么这样做是有意义的？对比不使用DH密钥交换的方案：
    - 使用K0 加密所有Ki
        - K0 泄露、之前、之后所有信息不安全
    - 使用Ki-1加密Ki
        - Ki泄露，之后所有信息不安全
        - DH Ki泄露、之前之后信息都不会受影响。