Implementation Tutorial on RSA

Maciek Adamczyk; m_adamczyk@umail.ucsb.edu
Marianne Magnussen; mariannemagnussen@umail.ucsb.edu

Overview

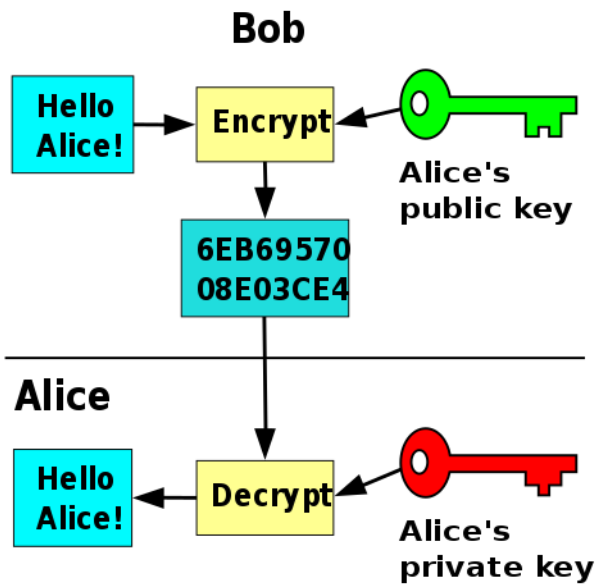
- Introduction to Cryptography
- Public-Key Cryptography
- RSA
- RSA - An Example
- RSA Implementation

Introduction

- Cryptography: A technique for securing communication between two or more parties
- Encrypting - Making the information unreadable for other people
- Decrypting - Decoding unreadable information to its original form
- Different types of cryptography:
  - Symmetric-key - same key for encryption and decryption
  - Public-key - two different keys for encryption and decryption

Public-key Cryptography

- Encryption scheme that uses two mathematically related, but not identical, keys
- The public key: for encryption and may be freely distributed
- The private key: for decryption and is private



RSA

- **Given a composite number, it is considered a hard problem to determine it's prime factors** [2]
- The keys in RSA are based on $n = pq$, where p and q are two large and random primes
- RSA is a set of two algorithms:

1. Key Generation
   - Generate the public and private key
2. RSA Function Evaluation
   - Process of transforming plaintext into ciphertext or vice versa

RSA: Security

- RSA is considered a secure algorithm since no known attempts to break it have yet been successful
- The algorithm is hard to break because of the difficulty of factoring large numbers $n = pq$

Security Level	RSA Modulus Size	Strength
80 bits	1024 bits	86.76611925028119
112 bits	2048 bits	116.8838132958159
128 bits	3072 bits	138.7362808527251
192 bits	7680 bits	203.01873594417665
256 bits	15360 bits	269.38477262128384

Security strength of RSA in relation with modulus

RSA: Strengths and Weaknesses

- The keys work both ways - the other key will always be able to decrypt. [3]
- RSA is hard to break but relatively easy to compute
  - Multiplying the two primes is an easier process than doing the factorization
- A weak choice of p and q makes RSA vulnerable to attacks
- Large primes does RSA more secure but the device performance and calculation time rises with the length of those numbers

RSA: Key Generation - Steps

- Generate a large integer n = pq, where p and q are two distinct and large primes
- Compute the Totient ϕ: $\phi(n) = (p-1)(q-1)$
  - $\phi(n)$ is used to calculate e
- The Public key is made by n and e
  - In the range $[3, \phi(n))$ choose a prime number e that has gcd $= 1$ with $\phi(n)$
  - Choosing e too small, may lead to security flaws
- The Private key is made by n and d
  - Compute d by taking the multiplicative inverse of the Public Key.
  - $e * d = 1 mod(\phi(n))$

RSA: Function Evaluation - Steps

- Encrypting the message m
  - $F(m, e) = m^e mod(n) = c$
- Decrypting the cipher text c
  - $F(c, d) = c^d mod(n) = m$

RSA: Simple Example [2] - Key Generation

- Choose two primes, p = 11 and q = 17: n = 11x17 = 187
- This gives us: $\phi(187) = (p-1)(q-1) = 160$
- The Public key, (n, e):
  - Select $e = 3$
    - gcd(3, $\phi(n)$) = 1
    - 3 is in the range [3, $\phi(n)$)
- The Private key, (n, d):
  - Finding d:
  - Find inverse of e with $\phi(n)$.
  - $d = 3^{-1} mod 160 = 107$

RSA: Simple Example - Function Evaluation

- We can now choose our message, lets say we want to encrypt "HI"
- Convert letters to numbers: H = 8 and I = 9
- m = 89
- Encryption
  - $m^e mod(n) = 89^3 mod 187 = 166 = c$
- Decryption
  - $c^d mod(n) = 166^{107} mod 187 = 89 = m$
- Different from a real world example:
  - Would need **much** bigger primes
  - Converting text to numeric format, easy with ascii

RSA: Simple Example Code

- Simple implementation of example above in Python
- **PS:** This is just an example, you should never implement this by yourself unless you really know these things.
- Use a library.

```
1  msg = 89 #Message "HI" converted to 89
2  p = 11
3  q = 17
4  e = 3
5  n = p*q
6  phi = (p-1)*(q-1)
7
8  def findInverse(a, p):
9    for i in range(0, p):
10     if(a * i % p ==  1):
11           return i
12   return False
13
14 def encrypt(m):
15   c = m ** e % n
16   print("Encrypted Message: ", c)
17     return c
18
19 d = findInverse(e, phi) #d = e^-1 mod(phi)
20
21 def decrypt(c):
22     dmsg = c ** d % n
23     print("Decrypted Message: ", dmsg)
24     return dmsg
25
26 print("Original Message:", msg)
27 c = encrypt(msg)
28 decrypt(c)
```

References

1. https://sahandsaba.com/cryptography-rsa-part-1.html
2. http://doctrina.org/How-RSA-Works-With-Examples.html
3. http://www.ijstr.org/final-print/july2017/Rsa-Public-Key-Cryptography-Algorithm-A-Review.pdf