RSA

ECE 4156/6156 Hardware-Oriented Security and Trust

Spring 2024

Assoc. Prof. Vincent John Mooney III

Georgia Institute of Technology

Reading Assistance

- RSA is covered in Chapter 8 of Introduction to Modern Cryptography by Katz and Lindell, but advanced number theory and other theoretical questions not covered in this lecture will not be required for this course
  - Obviously, theory that is covered – either in the slides or as explained – are required!
  - Another way to state this is that the expectation is that everything explained in lecture is understood; if not, please ask!!!

Introduction

- The idea of asymmetric crypto first published by Hellman and Diffie in 1976
  - Martin Hellman was a young faculty member in Electrical Engineering at Stanford
  - Whitfield Diffie joined Stanford's AI Lab in 1969 and worked for John McCarthy
    - In 1974, Whitfield Diffie started to work with Martin Hellman
- RSA invented by Ron Rivest, Adi Shamir and Leonard Adelman in 1977
  - Worked together for a year at MIT to develop a one-way function to implement Diffie-Hellman
  - Rivest and Shamir were trained in Computer Science; Adelman in Mathematics
    - Adelman would typically "break" the proposals of Rivest and/or Shamir
- Additional contemporary inventors of asymmetric cryptographic algorithms
  - Ralph Merkle developed an asymmetric crypto scheme as an undergraduate project at U.C. Berkeley and unsuccessfully tried to publish prior to 1977
  - Clifford Cocks, Government Communications Headquarters (GCHQ), U.K., also in the 1970s

RSA Overview

- Security based on difficulty of factoring large numbers
  - Public and private keys are functions of prime numbers
  - E.g., each key may have 200 digits
- Concept of a one-way function
  - Easy to compute in one direction, hard to compute in reverse
- Different keys accomplish encryption versus decryption
  - Allows one to publish encryption keys while keeping decryption keys secret
  - “Public-key” cryptography

Key Generation

- Choose two large prime numbers at random
  - p and q of approximately equal length
- Compute $n = p * q$
- Randomly choose encryption key e
  - e and $(p - 1)(q - 1)$ have no factor in common (other than 1)
    - Note: a number x is said to **factor** y iff x/y has no remainder (i.e., the remainder is zero!)
    - ➢ relatively prime or *coprime*
- Compute d such that the following holds
  - $ed = 1 \bmod (p - 1)(q - 1)$
  - In other words, $d = e^{-1} \bmod (p - 1)(q - 1)$
  - Note that in modular arithmetic e^{-1} is the multiplicative inverse of e

A Comment on Multiplicative Inverse in Modular Arithmetic

- In modular arithmetic, all numbers are whole
  - No fractions and no decimals
- For example, consider modulus with respect to *u*
- Suppose $u = 11$
  - 13 mod u = 13 mod 11 = 2
- Suppose $t = 7$
  - Let $v = t^{-1}$ = multiplicative inverse of t in modular arithmetic
    - ➢ tv mod $u = 1 = t*t^{-1}$
- Compute v such that $v = t^{-1}$
  - try $v = 1$: then tv mod u = 7 mod 11 = 7 ≠ 1
  - try $v = 2$: then tv mod u = 14 mod 11 = 3 ≠ 1
  - …
  - try $v = 8$: then tv mod u = 56 mod 11 = 1
    - ➢ $v = t^{-1} = 8$

Key Generation (cont’d)

- p and q are randomly chosen large prime numbers
- $n = p * q$
- Encryption key e is chosen such that
  - e and $(p - 1)(q - 1)$ have only 1 as a factor in common
    - ➢ relatively prime or *coprime*
- Compute d such that the following holds
  - $d = e^{-1} \bmod (p - 1)(q - 1)$ where e^{-1} is the multiplicative inverse of e
- d and n have no factor in common (other than 1)
    - ➢ d and n are coprime
- e and n are the public “key”
- d and n are the private “key”

Example of Key Generation

- Choose $p = 47$, $q = 71$
  - $n = p * q = 47 * 71 = 3337$
- e must have no factor in common with $(p - 1)(q - 1) = 46 * 70 = 3220$
  - Randomly choose $e = 79$
    - ➢ $d = e^{-1} \bmod (p - 1)(q - 1) = 79^{-1} \bmod 3220$
    - ➢ a program can be run to find d
      - variations of an original approach by Euclid
    - ➢ the result for this example is $d = 1019$
      - $ed = 79*1019 = 80501$
      - $3220*25 = 80500$
      - $ed \bmod 3220 = 80501 \bmod 3220 = 1$
- Public key (pair): $e = 79$ and $n = 3337$
- Private key: $d = 1019$
- Note: p and q may be thrown away & are no longer used or needed
  - Of course, p and q should not be revealed in any way!

RSA Encryption and Decryption

- Given message m, divide m into blocks m_i each of size 2^s
  - E.g., s is typically the largest number such that $2^s < n$
- The encrypted message c has blocks c_i each equal in size to m_i
- Encryption formula: $c_i = m_i^e \bmod n$
- Decryption formula: $m_i = c_i^d \bmod n$
  - Note the following where we skip the "mod n" everywhere:
  - $c_i^d = (m_i^e)^d = m_i^{ed} = m_i^{k(p-1)(q-1)+1}$ since $ed = 1 \bmod (p - 1)(q - 1)$
  - $m_i^{k(p-1)(q-1)+1} = m_i m_i^{k(p-1)(q-1)} = m_i(m_i^{(p-1)(q-1)})^k = m_i*1 = m_i$
  - Note that $(m_i^{(p-1)(q-1)})^k \bmod n = 1$ due to known results in group theory
- Note that may also encrypt with d and decrypt with e

Example of Encryption with Keys from Prior Ex.

- Public key (pair): $e = 79$ and $n = 3337$
- Private key (pair): $d = 1019$ and $n = 3337$
- $m = 688232687966683$,
  - $m_1 = 688$
  - $m_2 = 232$
  - $m_3 = 687$
  - $m_4 = 966$
  - $m_5 = 668$
  - $m_6 = 003$
- $c_1 = 688^{79} \bmod 3337 = 1570$, $c = 1570\ 2756\ 2091\ 2276\ 2423\ 158$
- $m_1 = 1570^{1019} \bmod 3337 = 688$, and so on

Why is RSA Difficult to Crack?

- Factoring is difficult, especially with large co-prime numbers *a* and *b*
  - In other words, given *N* where it is know that *N* = *a* * *b* (but the specific values of *a* and *b* are not known), there is no known PPT algorithm to find *a* and *b*
- Solving discrete exponentiation / logarithm is also difficult!
  - In other words, say you are given $f(x,p) = g^x \bmod p$
  - You are provided with p, x, the resulting answer from calculating $f(x,p)$, and the fact that all numbers are discrete (i.e., integers)
  - There is no known PPT algorithm to find g
  - In other words, given c_i, e and n, there is no known PPT algorithm to find m_i such that $c_i = m_i^e \bmod n$
  - Solving for m_i for such a discrete exponentiation / logarithm is difficult
    - Note that if $|m_i|$ = 2048 bits then a brute force attack (trying all m_i) is infeasible!