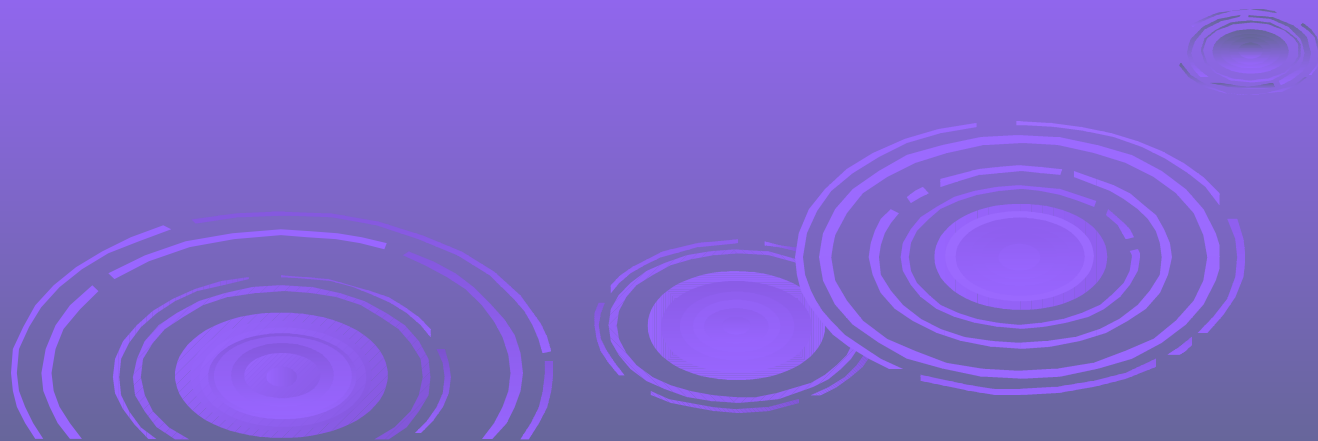


Cryptography and Network Security

Chapter 8



Prime Numbers

- prime numbers only have divisors of 1 and self
 - they cannot be written as a product of other numbers
 - note: 1 is prime, but is generally not of interest
- eg. 2,3,5,7 are prime, 4,6,8,9,10 are not
- prime numbers are central to number theory
- list of prime number less than 200 is:

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59
61 67 71 73 79 83 89 97 101 103 107 109 113 127
131 137 139 149 151 157 163 167 173 179 181 191
193 197 199

Prime Factorisation

- to **factor** a number n is to write it as a product of other numbers: $n = a \times b \times c$
- note that factoring a number is relatively hard compared to multiplying the factors together to generate the number
- the **prime factorisation** of a number n is when its written as a product of primes
 - eg. $91 = 7 \times 13$; $3600 = 2^4 \times 3^2 \times 5^2$

$$a = \prod_{p \in P} p^{a_p}$$

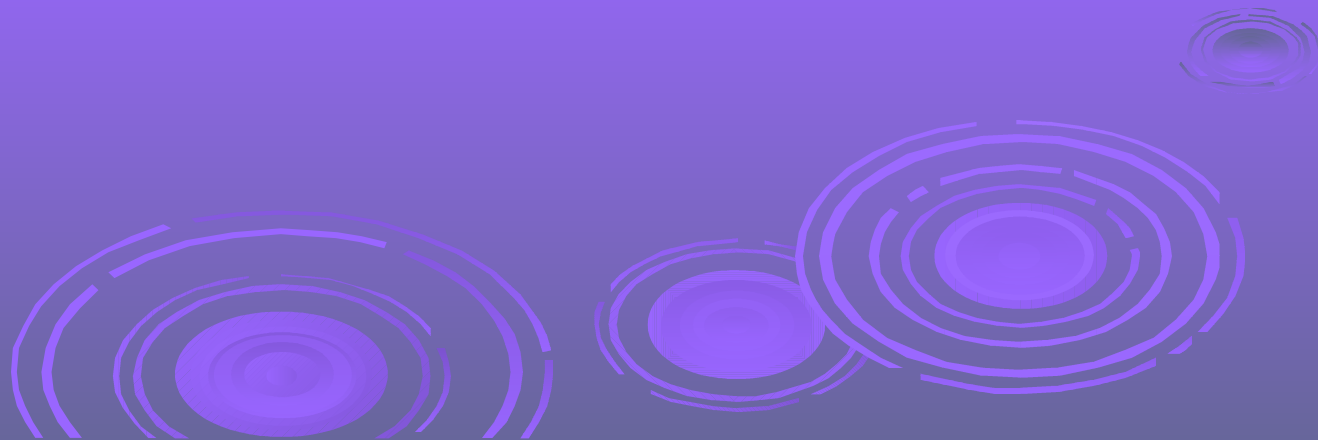
Relatively Prime Numbers & GCD

- two numbers a , b are **relatively prime** if have **no common divisors** apart from 1
 - eg. 8 & 15 are relatively prime since factors of 8 are 1,2,4,8 and of 15 are 1,3,5,15 and 1 is the only common factor
- conversely can determine the greatest common divisor by comparing their prime factorizations and using least powers
 - eg. $300=2^1 \times 3^1 \times 5^2$ $18=2^1 \times 3^2$ hence
 $\text{GCD}(18, 300) = 2^1 \times 3^1 \times 5^0 = 6$



Fermat's Theorem

- $a^{p-1} = 1 \pmod{p}$
 - where p is prime and $\gcd(a, p) = 1$
- also known as Fermat's Little Theorem
- also $a^p = a \pmod{p}$
- useful in public key and primality testing



Euler Totient Function $\phi(n)$

- when doing arithmetic modulo n
- **complete set of residues** is: $0 \dots n-1$
- **reduced set of residues** is those numbers (residues) which are relatively prime to n
 - eg for $n=10$,
 - complete set of residues is $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
 - reduced set of residues is $\{1, 3, 7, 9\}$
- number of elements in reduced set of residues is called the **Euler Totient Function $\phi(n)$**

Euler Totient Function $\phi(n)$

- to compute $\phi(n)$ need to count number of residues to be excluded
- in general need prime factorization, but
 - for p (p prime) $\phi(p) = p-1$
 - for $p.q$ (p, q prime) $\phi(pq) = (p-1) \times (q-1)$
- eg.

$$\phi(37) = 36$$

$$\phi(21) = (3-1) \times (7-1) = 2 \times 6 = 12$$

Euler's Theorem

➤ a generalisation of Fermat's Theorem

➤ $a^{\phi(n)} = 1 \pmod{n}$

- for any a, n where $\gcd(a, n) = 1$

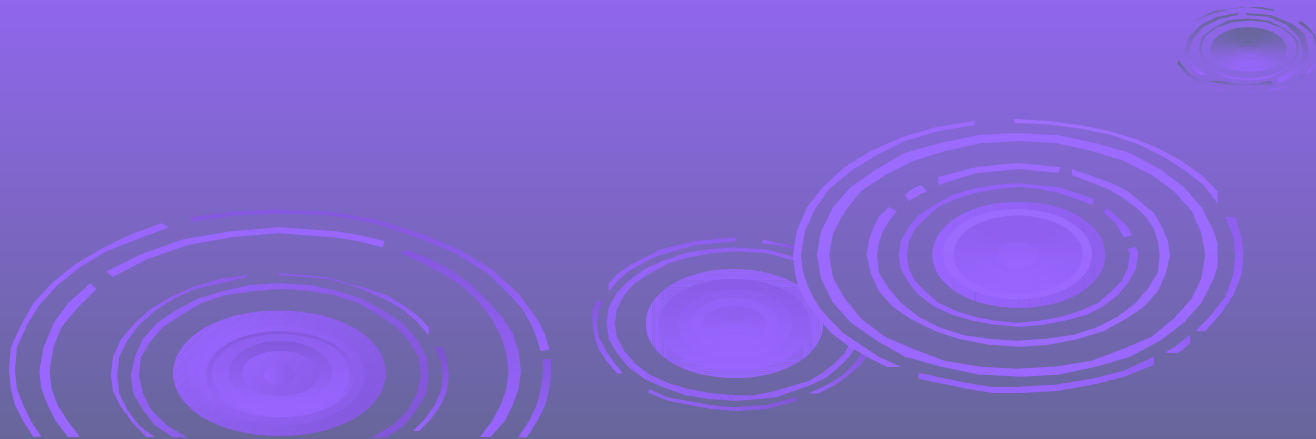
➤ eg.

$a=3; n=10; \phi(10)=4;$

hence $3^4 = 81 = 1 \pmod{10}$

$a=2; n=11; \phi(11)=10;$

hence $2^{10} = 1024 = 1 \pmod{11}$



Primality Testing

- often need to find large prime numbers
- traditionally **sieve** using **trial division**
 - ie. divide by all numbers (primes) in turn less than the square root of the number
 - only works for small numbers
- alternatively can use statistical primality tests based on properties of primes
 - for which all primes numbers satisfy property
 - but some composite numbers, called pseudo-primes, also satisfy the property
- can use a slower deterministic primality test

Miller Rabin Algorithm

➤ a test based on Fermat's Theorem

➤ algorithm is:

TEST (n) is:

1. Find integers $k, q, k > 0, q$ odd, so that $(n-1) = 2^k q$

2. Select a random integer $a, 1 < a < n-1$

3. **if** $a^q \bmod n = 1$ **then** return ("maybe prime");

4. **for** $j = 0$ **to** $k - 1$ **do**

5. **if** $(a^{2^j q} \bmod n = n-1)$
then return(" maybe prime ")

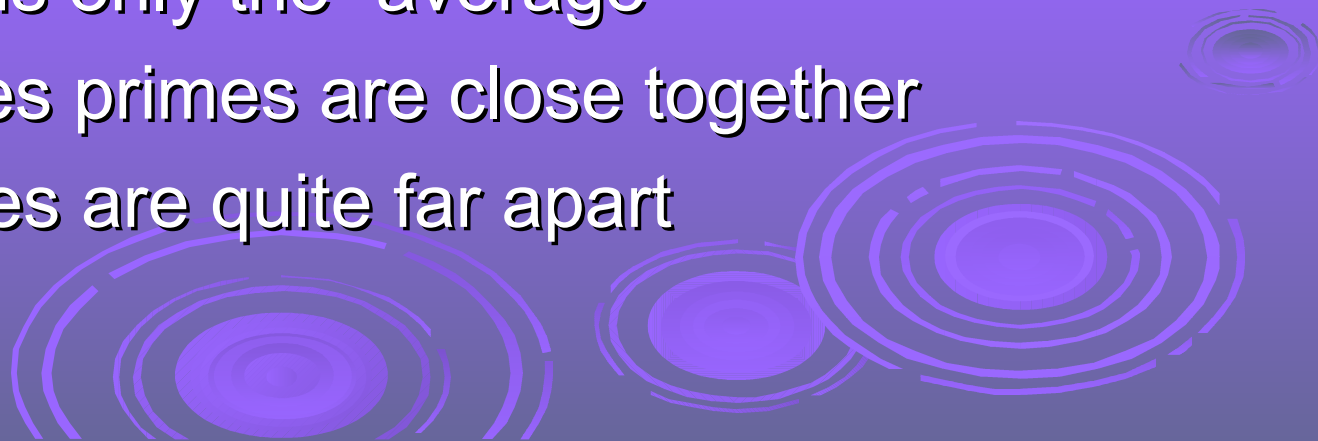
6. return ("composite")

Probabilistic Considerations

- if Miller-Rabin returns “composite” the number is definitely not prime
- otherwise is a prime or a pseudo-prime
- chance it detects a pseudo-prime is $< 1/4$
- hence if repeat test with different random a then chance n is prime after t tests is:
 - $\text{Pr}(n \text{ prime after } t \text{ tests}) = 1 - 4^{-t}$
 - eg. for $t=10$ this probability is > 0.99999

Prime Distribution

- prime number theorem states that primes occur roughly every $(\ln n)$ integers
- but can immediately ignore evens
- so in practice need only test $0.5 \ln(n)$ numbers of size n to locate a prime
 - note this is only the “average”
 - sometimes primes are close together
 - other times are quite far apart



Chinese Remainder Theorem

- used to speed up modulo computations
- if working modulo a product of numbers
 - eg. $\text{mod } M = m_1 m_2 \dots m_k$
- Chinese Remainder theorem lets us work in each moduli m_i separately
- since computational cost is proportional to size, this is faster than working in the full modulus M

Chinese Remainder Theorem

- can implement CRT in several ways
- to compute $A \pmod{M}$
 - first compute all $a_i = A \pmod{m_i}$ separately
 - determine constants c_i below, where $M_i = M/m_i$
 - then combine results to get answer using:

$$A \equiv \left(\sum_{i=1}^k a_i c_i \right) \pmod{M}$$

$$c_i = M_i \times (M_i^{-1} \pmod{m_i}) \quad \text{for } 1 \leq i \leq k$$

Primitive Roots

- from Euler's theorem have $a^{\phi(n)} \bmod n = 1$
- consider $a^m = 1 \pmod n$, $\text{GCD}(a, n) = 1$
 - must exist for $m = \phi(n)$ but may be smaller
 - once powers reach m , cycle will repeat
- if smallest is $m = \phi(n)$ then a is called a **primitive root**
- if p is prime, then successive powers of a "generate" the group $\bmod p$
- these are useful but relatively hard to find

Discrete Logarithms

- the inverse problem to exponentiation is to find the **discrete logarithm** of a number modulo p
- that is to find x such that $y = g^x \pmod{p}$
- this is written as $x = \log_g y \pmod{p}$
- if g is a primitive root then it always exists, otherwise it may not, eg.
 - $x = \log_3 4 \pmod{13}$ has no answer
 - $x = \log_2 3 \pmod{13} = 4$ by trying successive powers
- whilst exponentiation is relatively easy, finding discrete logarithms is generally a **hard** problem

Summary

- have considered:
 - prime numbers
 - Fermat's and Euler's Theorems & $\phi(n)$
 - Primality Testing
 - Chinese Remainder Theorem
 - Discrete Logarithms

