Zainab Musa: SCT211-0510/2021

Chapter 4: Number Theory and Cryptography

4.1 Divisibility and Modular Arithmetic

Divisibility:

- An integer *a* divides *b* (denoted $a \mid b$) if there exists an integer *c* such that $b = a \times c$.
- Example: $4 \mid 20$ because $20 = 4 \times 5$.
- Divisibility has the following properties:
  - Reflexive: Every number divides itself.
  - Transitive: If $a \mid b$ and $b \mid c$, then $a \mid c$.
  - Multiplicative: If $a \mid b$ and $a \mid c$, then $a \mid (b + c)$ and $a \mid (b - c)$.

The Division Algorithm:

- For any integers *a* and $d > 0$, there exist unique integers *q* and *r* such that:
  - $a = dq + r$, where $0 \le r < d$.
- This forms the basis for Euclidean division and leads to GCD algorithms.

Modular Arithmetic:

- Two integers *a* and *b* are congruent modulo *m* if $m \mid (a - b)$, written as $a \equiv b \pmod{m}$.
- Arithmetic operations in modular systems preserve many useful properties:
  - Addition: (a + b) mod m = [(a mod m) + (b mod m)] mod m
  - Multiplication: (a × b) mod m = [(a mod m) × (b mod m)] mod m
  - Exponentiation: a^k mod m, with optimizations via modular exponentiation.
- Modular arithmetic forms the basis for many algorithms in computer science and cryptography.

Applications:

- Used in digital clocks, hashing, data encryption, calendars, and cyclic structures.
- Example: Digital clocks use mod 12 or mod 24 arithmetic. For instance, 17:00 + 10 hours = (17 + 10) mod 24 = 3:00.

Example:

- Compute 7^4 mod 10:
    - 7^4 = 2401, and 2401 mod 10 = 1

Additional Example:

- Determine whether 81 ≡ 4 mod 7:
    - 81 - 4 = 77, and 77 is divisible by 7, so the congruence holds.

4.2 Integer Representations and Algorithms

Number Representation in Different Bases:

- **Binary (base 2)**: Primary system used in computing. Only uses digits 0 and 1.
- **Octal (base 8)** and **Hexadecimal (base 16)**: Used to compactly represent binary numbers.
- **Decimal (base 10)** is the standard system for human-readable numbers.

Base Conversion Algorithms:

- From decimal to base *b*: repeatedly divide the number by *b* and record remainders.
- From base *b* to decimal: multiply each digit by b raised to the power of its position.

Example: Convert 43 to binary:

- 43 ÷ 2 = 21 remainder 1
- 21 ÷ 2 = 10 remainder 1
- 10 ÷ 2 = 5 remainder 0
- 5 ÷ 2 = 2 remainder 1
- 2 ÷ 2 = 1 remainder 0
- 1 ÷ 2 = 0 remainder 1 → Binary = 101011

Efficient Arithmetic in Binary:

- Computers perform binary addition, subtraction, multiplication and division using simple logic circuits.

Modular Exponentiation:

- A crucial technique to compute a^b mod n efficiently.
- Used in RSA, Diffie-Hellman key exchange, digital signatures.

Square-and-Multiply Algorithm:

- Converts exponent to binary and uses squaring/multiplication only when needed.

Example: Compute 3^13 mod 17 using binary exponentiation.

4.3 Primes and Greatest Common Divisors

Prime Numbers:

- A number $p > 1$ is prime if its only positive divisors are 1 and p.
- Infinite in number, as proven by Euclid.
- Importance in encryption: RSA and other cryptographic systems rely on prime numbers.

Fundamental Theorem of Arithmetic:

- Every integer greater than 1 is either prime or can be expressed uniquely as a product of prime numbers (up to the order of the factors).

Composite Numbers:

- Have more than two positive divisors.
- Example: $12 = 2 \times 2 \times 3$

GCD and LCM:

- The greatest common divisor (GCD) of a and b is the largest number that divides both.
- The least common multiple (LCM) of a and b is the smallest number divisible by both.
- Relationship: $\gcd(a, b) \times \text{lcm}(a, b) = a \times b$

Euclidean Algorithm:

- A classic and efficient algorithm to compute GCD:
  - $a = bq + r$
  - Repeat with (b, r) until $r = 0$

Extended Euclidean Algorithm:

- Produces integers x and y such that ax + by = gcd(a, b)
- Critical in finding modular inverses used in RSA.

Example: Find GCD of 99 and 78:

- 99 = 78×1 + 21
- 78 = 21×3 + 15
- 21 = 15×1 + 6
- 15 = 6×2 + 3
- 6 = 3×2 + 0 ➔ GCD = 3

4.4 Solving Congruences

Linear Congruences:

- General form: ax ≡ b (mod m)
- Has solution if and only if gcd(a, m) divides b.

Solution Steps:

1. Simplify if possible.
2. Use the extended Euclidean algorithm to find modular inverse of a.
3. Multiply both sides by the inverse modulo m.
4. General solution: $x \equiv x_0 + k(m/d)$ for integer k, where d = gcd(a, m)

Chinese Remainder Theorem (CRT):

- If $m_1, m_2, \ldots, m_n$ are pairwise coprime, and we have:
  - $x \equiv a_1 \pmod{m_1}$
  - $x \equiv a_2 \pmod{m_2}$
  - ...
  - $x \equiv a_n \pmod{m_n}$
- Then there exists a unique solution modulo $M = m_1 \times m_2 \times \ldots \times m_n$
- Applied in solving large systems efficiently.

Fermat's Little Theorem:

- If *p* is prime and *a* is not divisible by *p*, then:
  - a^(p−1) ≡ 1 mod p
- Used in primality testing, finding inverses in RSA.

Example: Find 3^6 mod 7

- Since 7 is prime, 3^6 ≡ 1 mod 7 by Fermat's theorem.

4.5 Applications of Congruences

Hashing:

- Used in data structures (hash tables), authentication systems, and load balancing.
- Example: key mod table size to find storage index.

Check Digits and Error Detection:

- Used in ISBN, credit cards, UPC codes.
- Example: In ISBN-10:
    - Compute: $(1\times d_1 + 2\times d_2 + \dots + 10\times d_{10}) \bmod 11 = 0$

Pseudorandom Number Generation:

- Uses linear congruential generators (LCGs):
    - x_{n+1} = (a x_n + c) mod m
- If parameters are chosen properly, can produce long sequences with good randomness properties.

Calendar Algorithms:

- Use modular arithmetic to compute day of week, leap years.
- Example: Zeller's Congruence computes the day of the week for any date.

4.6 Cryptography

Classical Ciphers:

- Caesar Cipher: shift characters using modular arithmetic.
    - E_k(p) = (p + k) mod 26
    - D_k(c) = (c - k) mod 26

Symmetric Key Cryptography:

- Same secret key used by both sender and receiver.
- Fast but key distribution is a challenge.
- Examples: DES, AES.

Public Key Cryptography (RSA):

- Uses two keys: public key for encryption and private key for decryption.
- Based on the difficulty of factoring large semiprimes.

RSA Algorithm:

1. Choose large primes p and q
2. Compute $n = pq$
3. Compute $\phi(n) = (p-1)(q-1)$
4. Choose e such that $\gcd(e, \phi(n)) = 1$
5. Compute d such that $ed \equiv 1 \bmod \phi(n)$
6. Public key: (e, n), Private key: (d, n)
7. Encrypt: c = m^e mod n
8. Decrypt: m = c^d mod n

Digital Signatures:

- Verify sender identity and message integrity
- Sign: hash the message and encrypt with private key
- Verify: decrypt signature with public key and compare hashes

Homomorphic Encryption:

- Allows operations on encrypted data
- Example: Add two encrypted salaries without decrypting either