Functions/Formulas

**Euler’s Totient Function**

phi(*n*) = *m*, where m is the number of prime numbers between 0 and *n*.

* if *n* is prime: phi(*n*) = *n*-1
* if *n* is the product of two primes: phi(*n*) = phi(*a* \* *b*) = phi(*a*) \* phi(*b*) = *a-*1 \* *b*-1
* if *n* is a power of *m*: phi(*n*) = *nm* – *nm*-1

**Congruence**

*a* == *b* mod *n* if and only if *a*-*b* = *k*\**n*, where *k* is some integer.

In other words, *n* mod (*a*-*b*) = 0

**Chinese Remainder Theorem**

*x* = ( t[sum]i=1 *n/di \* yi \* xi* ) mod *n*, where *yi* is some solution of *n*/*di* \* *yi* mod *di* = 1

* eg. 3*x* mod 10 = 1 (find multiplicative inverse of 3 mod 10)
  + split *m*into *i* primes: 10 = 2 \* 5
    - 3*x1* mod 2 = 1 *d1* = 2,  *x1* = 1
    - 3*x2* mod 5 = 1 *d2* = 5, *x2* = 2
  + then find all *yi* IN *n / di* \* *yi* mod *di* = 1:
    - 10/2 *y1* mod 2 = 1 *y1* = 1
    - 10/5 *y2* mod 5 = 1 *y2* = 3
  + therefore:
    - *x* = (5\*1\*1 + 2\*3\*2) mod 10 = 17 mod 10 = 7

**Fermat’s Little Theorem**

Where *p* is prime, for every *i* such that gcd(*i, p*) = 1, *ip-1* mod *p* = 1

**Euler’s generalisation**

For any *a* and *n* where gcd(*a,n)* = 1, *aphi(n)* mod *n* = 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *i* | *a* | *z* | *x* | *n* |
| 0 | 9 | 11 | 1 | 13 |
| 1 | 9 | 10 | 9 |  |
| 2 | 3 | 5 | 9 |  |
| 3 | 3 | 4 | 1 |  |
| 4 | 9 | 2 | 1 |  |
| 5 | 9 | 1 | 9 |  |
| 6 |  | 0 | 3 |  |

* find multiplicative inverse with:
  + *ax* mod *n* = 1
  + *x =* *aphi(n)-1* mod *n*
  + *ax = a\*aphi(n)-1*mod *n ax = aphi(n)-1+1*mod *n*
  + *aphi(n)* mod *n* = 1

**Fast exponentiation**

exp(**a, z, n**) – *az* mod *n*, where *z* = *phi*(*n*) – 1

eg. 9*x* mod 13 = 1

while *z* != 0

while *z* mod 2 = 0

*ai+1 = a­i\*ai* mod *n*

*zi+1 = z* / 2

*xi+1* = *x*

*xi+1* = *x\*a* mod *n*

*zi+1* = *z* – 1

*x* = fastexp

**Extended Euclid’s Algorithm**

**Entropy**

A measure of how much information can be gleaned from a message

* *H(X)* = -(*n[SUM]i=1 p(Xi) log2 p(Xi)*)
* eg. *p(g)* = 0.9, *p(m)* = 0.1, where *p(g)* is the probability of a goal and *p(m)* is the probability of a miss
  + –(0.9\*log2(0.9) + 0.1\*log2(0.1))

**Unicity Distance**

A measure of the secrecy of a cipher

* *Hc(K)* = *[SUM]p(C) [SUM]pc(K) log2*(1/*pc(K)*), where *Hc(K) =* the key equivocation of ciphertext *C* and *pc(K)* is the probability of *K* given *C.*

Attacks

**Birthday attack**

**Insertion attack**

* Attacker intercepts the ciphertext
* Solution: Never use the same key twice

Week 1

Week 2

Week 3 – Number Theory, Galois Fields, Perfect Secrecy

Week 4

Week 5

Week 6 – DES, Block and Stream Ciphers

**Stream ciphers**

* Streams can be synchronised or self-synchronising
* Synchronised: Key is generated independently of ciphertext
  + not error-propagating
  + sender and receiver must re-synchronise if initial key is garbled – a completely different key must be generated.
* Self-synchronising: Key is generated as a function of ciphertext.
  + vulnerable to play-back attack.
  + error-propagating (one garbled ciphertext will result in continued errors until the key re-synchronises.

**Block ciphers**

* Takes an *n*-bit block of plaintext and produces an *n*-bit block of ciphertext.
* For encryption to be possible, each plaintext block must encrypt into a unique ciphertext block – this is **reversible** (one-to-one) mapping as opposed to **irreversible** (one-way) mapping
* Mapping should be arbitrary.
* For block size:
  + **small** – converts to monoalphabetic cipher (not secure)
  + **large** – arbitrary mapping and secure system, but cannot be implemented easily
* **Feistel:**
  + uses a sequence of substitutions and permutations
  + alternates **confusion** (obscures local PT structure) and **diffusion** (obscures global structure)

**Confusion and Diffusion**

* Stream ciphers rely on confusion, block ciphers use confusion and diffusion

Week 7

Week 8

Week 9

Week 10 – Hash Functions & Digital Signatures

**Hash functions**

* Requirements:
  + Easy to compute
  + Applied to block of any size/outputs string of fixed length
  + Preimage resistant: infeasible to find *x* such that *H(x)* = *h* for any given *h*.
  + Weak collision: infeasible to find *y* such that *H(x)* = *H(y)* for any given *x.*
  + Strong collision: infeasible to find (*x,y*) such that *H(x)* = *H(y)*.

**Birthday Attack**

* **Steps:**
  + Sender **A** prepares to sign a message (create an *m*-bit hash code and encrypt it with A’s private key.
  + Attacker **C** generates 2m/2 variations of A’s private key and the same number of false messages.
  + The two sets of code are compared to find a pair that generates the same hash code – more messages if no match is found.
  + C offers the valid variation to A for signature but sends the false variation to receiver **B.**
* Based on the *birthday paradox* – model that reduces the complexity of finding collisions in a hash function.

**Security of Hash Functions**

* Brute force:
  + hash function:
    - one way – 2m-1
    - weak collision resistance – 2m-1
    - strong collision resistance – 2m/2
  + MAC (Message Authentication Code)
    - min – (2k, 2m)

**Hash & MAC Algorithms**

* Hash functions
  + condense arbitrary-sized message to fixed size by processing message in blocks through a compression function
  + custom or block-cipher based
* MAC (Message Authentication Code)
  + fixed-size authentication to provide authentication for message using block-cipher or hash function

Week 11

Week 12