Chapter 2: Cryptography

Review of Cryptography

Terminology

Encryption: process of encoding a message so that its meaning is not obvious
Decryption: reverse process, transforming an encrypted message back into its original form
Cryptosystem: systemm for encryption and decryption

`c = E(p) and p = D(c)`

- `c` represents the ciphertext
- `E` is encryption rule (algorithm)
- `p` is the plaintext
- `D` is the decryption rule (algorithm)

What we seek is a cryptosystem for which `P = D(E(P))`

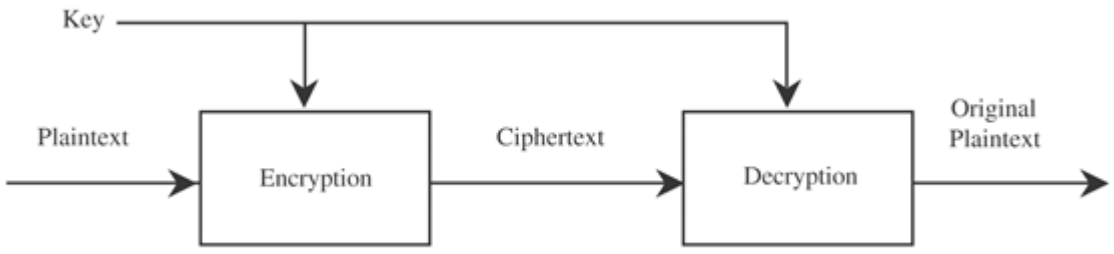
- encryption and decryption algorithms often use a set of keys, denoted by `K`, so the resulting cyphertext depends on the original plaintext message, the algorithm, and the key value:

  `c=E(K,P)`

Symmetric Encryption

The encryption and decryption keys are the same. We have:

`P = D(K,E(K,P))`



Asymmetric Encryption

The encryption and decryption keys come in pairs. A **decryption key**, K_D, inverts the encryption of key K_E

$P = D(K_D,E(K_E,P)$

Cryptanlayst

- person who studies encryptions and ciphertexts to figure out the corresponding plaintexts
- **break** a single message

- recognize **patterns** in ciphertexts, be able to break subsequent ones by applying decryption algo
- **infer** some meaning w/o breaking the encryption
- **deduce** the key to break subsequent ciphers
- find **weaknesses** in the **implementation** or environment of use of the encryption
- find general **weaknesses* in an encryption** algorithm**, w/o necessarily having intercepted any messages

Representing Characters

Convention: *plaintext* is written in UPPERCASE letters, ciphertext in *lowercase* letters

- encryption algorithms are usually based on mathematical transformations

Substitution Ciphers

- not secure to use, but good to know and use as some building blocks in other encryption methods
- **substitution**: one letter exchanged for another
- **transposition**: order of letters is rearranged

Mono-alphabetic

- fixed substitution over the entire message (simple substitution)
  - caesar cipher (shifting alphabet) , atbase cipher (reversing alphabet)

Poly-alphabetic

- uses a number of substitution at different positions in the message
  - vigenere cipher, enigma cipher

Caesar Cipher

- each letter is translated to the letter a fixed number of places after it in the alphabet
- uses a shift of 3, so plaintext letter p_i is enciphered as ciphertext letter c_i by the rule

  $c_i = E(p_i) = p_i+3$

Advantage and Disadvantage

- simplicity
- small piece of cipher text is enough to predict the entire pattern of the encryption

Cryptanalysis

- allow blank to be translated to itself
- sub known short words at appropriate places
- sub for matching characters at other places
- guess 3 letter words, guess unused letters

Complexity

- encryption and decryption with sub ciphers can be performed by direct lookup in a table illustrating the correspondence
- transforming a single character can be done in a constant ($O(1)$) time.
- complexity of the algo to encrypt a msg of n characters is proportional to n ($O(n)$)

Cryptanalysis

- guess, strategy
- mathematical skills
- language word patterns (short words, words w/ repeated patterns, common initial and final letters)
- letter frequency (Frequency Distrubuted Analysis)
- context of the text

One-Time Pads (Perfect Cipher)

- length of key should be the same as length of plaintex message
- Perfect secrecy: ciphertext gives no additional info about plaintext
- **issues**: synchronization b/w sender and receiver, unlimited number of keys

Transpositions (Permutations)

- reordering of characters of the plaintext
  - `p(0) = 2` means that A is encrypted to c
- one way is to use a *key* to control the permutation
- main goal is *diffusion*
  - widely spread the info from the msg or key across the ciphertext

Columnar transpositions

- rearrangement of characters of plaintext into columns
  - can set a key as a random order of columns then read the columns based on ascending order of key numbers

Complexity

- time needed to apply the algo is proportional to the length of the message (`O(n)`)
- sub cipher requires only a constant amount of space (26^2 locations)

Di-grams: pairs of adjacent letters **Tri-grams**: groups of three letters

Cryptanalysis

1. Computing letter frequencies

- if all letters appear w/ their normal frequencies, we can infer that a transposition has been performed

2. Break the ciphertext into columns

- two different strings of letters can represent di-grams from the plaintext
  - where do a pair of adjacent columns lie?
  - where are the ends of the columns?

3. Compare a block of characters (as a moving window) against characters successively farther away in the ciphertext
4. For each window positions:

- do common di-grams appear?
- do most of the di-grams look reasonable?

5. When di-grams indicate a possible match for a fragment of ciphertext, the next step is to try to extend the match

Good Ciphers

Shannon's Characteristics of "Good" Ciphers:

1. Amount of secrecy needed should determine the amount of labour appropriate for the encryption and decryption
2. The set of keys and the enciphering alogrithm should be free from complexity

- restrict neither the choice of keys nor the types of plaintext on which the algo works
- key must be transmitted, stored, and remembered so it must be short

3. Implementation process should be as simple as possible
4. Errors in ciphering should not propogate and cause corruption of further information in the message
5. Size of the ciphertext should be no larger than the text of the original message, otherwise:

- it gives cryptanalysts more data from which to infer a pattern
- longer ciphertext implies more space for storage and more time to communicate

Trustworthy Encryption Systems

Encryption is ***commercial grade*** or ***trustworthy*** by meet some constraints:

- it is based on sound mathematics
- it has been analyzed by competent experts and found to be sound
- has stood the "test of time". Flaws in many algos are discovered relatively soon after their release

Stream and Block Ciphers

Stream Cipher

- converts one symbol of plaintext immediately into a symbol of cyphertext
- transformation depends only on the symbol, the key, and control information of the algorithm
- skipping a character in the key during encryption affects the encryption of all future characters
  - receiver can recalibrate the key

Block Cipher

- group of plaintext symbols are encrypted as one block

Advantages

Stream: speed of transformation, low error propogation
Block: high diffusion, immunity to insertion of symbols

Disadvantages

Stream: low diffusion, susceptiblity to malicious insertions
Block: slowness of encryption, error propogation

Confusion and Diffusion

Confusion: interceptor should not be able to predict what will happen to the ciphertext by changing one character in the plaintext
Diffusion: cipher should spread the information from the plaintext over the entire ciphertext so changes in the plaintext affect many parts of the ciphertext

Cryptanalysis

Four types of information:

1. Ciphertext
2. Full plaintext
3. Partial plaintext
4. Algorithm

Five approches:

1. Ciphertext only

- decryption based on probabilities, distributions, and characteristics of the available ciphertext, plus publicly available knowledge

2. Full or Partial plaintext

- analyst has a sample message and its decipherment
- attempt to find `E` or `D` by using **known plaintext** attack

3. Ciphertext of any plaintext

- analyst may have inflitrated the sender's transmission process to be able to cause messages to be encrypted and sent at will. This attack is called a **chosen plaintext** attack

4. Algorithm & Ciphertext

- analyst may have both encryption algo and ciphertext, **chosen ciphertext** attack
- can run the algo on large amounts of plaintext to find one plaintext msg that encrypts as the ciphertext

5. Ciphertext & Plaintext

- analyst may have pairs of plaintext and matching ciphertext. We then must deduce the key

Type of Attack	**Known to Cryptanalyst**
Ciphertext Only * encryption algorithm * ciphertext”	”
Known Plaintext * encryption algorithm * ciphertext * one or more plaintext-ciphertext pairs formed with the secret key”	”
Chosen Plaintext * encryption algorithm * ciphertext * plaintext message chosen by cryptanalyst with its corresponding ciphertext generated with the secret key”	”
Chosen Ciphertext * encryption algorithm * ciphertext * ciphertext chosen by cryptanalyst with its corresponding decrypted plaintext generated with the secret key”	”
Chosen Text * encryption algorithm * ciphertext * plaintext message chosen by cryptanalyst with its corresponding ciphertext generated with the secret key * ciphertext chosen by cryptanalyst with its corresponding decrypted plaintext generated with the secret key”	”

Encryption Systems

Symmetric (Private-key) Cryptosystem

- two-way channel to their users: A & B share a secret key, can both encrypt info to send a decrypt info from the other
- **Difficulty**: *key distribution.* How do A & B obtain their shared secret key?

Asymmetric (Public-key) Cryptosystem

- Two separate keys: public key for encryption, private key for decryption
- key management is a major issue for both asymmetric and symmetric systems

Data Encryption Standard (DES)

- complex combination of two fundamental building blocks of encryption: substitution and transposition
- algorithm begins by encrypting the **plaintext as blocks of 64 bits**
- **key is 64 bits** long, but can be any 56-bit number (extra 8 bits often used as check digits)

DES accomplishes two things:

1. Ensuring that output bits have no obvious relationship to the input bits (Substitution provides confusion)
2. Spreading the effect of one plaintext bit to other bits in the ciphertext (Transposition provides the diffusion)

Steps

1. Input to the DES is divided into blocks of 64 bits
2. The 64 data bits are permuted by a so-called initial permutation
3. The data bits are transformed by a 64-bit key (of which only 56 bits are used)
4. Key is reduced from 64 bits to 56 bis by dropping bits 8, 16, 24,..., 64 (where the most significant bit is named bit 1). These bits are assumed to be the parity bits that carry no information in the key
5. Sequence of operations known as a cycle begins
6. The 64 permuted data bits are broken into a **left half** and a **right half** of **32 bits each**
7. Key is shifted left by a number of bits and permuted
8. Key is combined with the **right half**, which is then combined with the **left half**
9. Result of these combinations becomes the **new right half**. The **old right half** becomes the **new left half**
10. Cycles are repeated 16 times. After the last cycle is a final permutation, which is inverse of the initial permutation

For a 32-bit right half to be combined with a 64-bit key, two changes are needed:

- First, algorithm expands the 32-bit half to 48 bits by repeating certain bits, this is **expansion permutations**
- Second, the 56-bit key is reduced to 48 bits by choosing only certain bits, this is **permuted choices**

Details of each cycle of the algorithm:

1. Right half will be the left half of the next cycle
2. The right half is expanded from 32 bits to 48
3. Then, it is combined with a form of the key
4. Results of this operation is then substituted for another result and condensed to 32 bits at the same time
5. the 32 bits are permuted and combined with the left half to yield a new right half

S-Boxes

S-Box: a permuted choice function by which six bits of data are replaced by four bits

- Substitutions are performed by eight S-boxes
- 48-bit input is divided into eight 6-bit blocks $B_1B_2...B_8$
- block B_j is operated on by S-box S_j

Suppose block B_j is the six bits $b_1b_2b_3b_4b_5b_6$:

bits b_1 and b_6 are taken to form a two-bit binary number b_1b_6, with a decimal value from 0-3. Call this value `r` bits b_2, b_3, b_4, and b_5 are joined to form a 4-bit binary number $b_2b_3b_4b_5$, having a decimal value from 0-15. Call this value `c`

- substitution from the S-boxes transform each 6-bit block B_j into the 4-bit result shown in row `r`, column `c` of section S_i of S-boxes table

P-Boxes

Bit	**Goes to Position**
1-8	9 17 23 31 13 28 2 18
9-16	24 16 30 6 26 20 10 1
17-24	8 14 25 3 4 29 11 19
25-32	32 12 22 7 5 27 15 21

Decryption of DES

- the same DES algorithm is used both for encryption and decryption
- result is true because `cycle j` dervices from `cycle j-1`
- keys must be taken in reverse order (k_{16}, k_{15}, …, k_1)
- key shifts are applied in reverse

First cycle of encryption: L_0 `XOR f(`R_0`,`K_{16}`)`

Double DES using two keys: `c = E(`k_1`, E(`k_1`, p))`

- meet-in-the-middle attack shows double DES gives a strength equivalent to a 57-bit key

Triple DES using three keys: `c = E(`k_1`, D(`k_1`, p)))`

- gives a strength equivalent to a 112-bit key (double DES attack defeats the strength of 1/3 keys

Weaknesses of DES

- complements
  - `if c = E(k, p) then \~ = E(\~k, \~p)`
- semi-weak keys
  - two different keys k_1 and k_2 for which `c = E(`k_1`, p)` and `c = E(`k_2`, p`

Advanced Encryption Standard (AES)

- Primarily uses: substitution, transposition, shift, XOR, addition operations
- uses repeated cycles: 10, 12, or 14 cycles for keys of 128, 192, and 256 bits
- **block ciphier** of block size 128 bits
- 128 bit block is represented as a 4x4 matrix, "*state*", shown as `s[0,0], ..., s[3,3]`
- state is filled from the input in columns

Each cycle consists of four steps:

1. S-Boxes substitute each byte of a 128-bit block according to a substitution table (diffusion)
2. Shift row: transposition

- for 128 and 192 bit block sizes, row `n` is shifted left circular (`n-1`) bytes
- for 256-bit blocks, row 2 is shifted 1 byte and rows 3 and 4 are shifted 3 and 4 bytes, respectively (confusion)

3. Mix column: shifting left and XORing bits w/ themselves (confusion and diffusion)
4. Add subkey: portion of the key unique to this cycle is XORed with the cycle result (confusion, key incorporation)

Mathematics

- Greatest Common Divisor (GCD)
  - GCD of two integers is the largest positive integer that divides the numbers w/o a remainder
- Euclidean Algorithm
  - ```
    int gcd(int a, int b){
            if(a == 0)
                return b;
            return gcd(b%a, a);
        }
    ```
- Modular Arithmetic
- Computing Inverses
  - For any prime `p` and any element `a < p`, the inverse of `a` is an element `x` such that:

  `ax mod p = 1`
- Fermat's Theorem
  - For any prime `p` and any element `a < p`:

  $a^{p-1} \bmod p = 1$

Public Key Encryption Systems

- each user has two keys: a public key and a private key
- the user may publish the public key freely

  Let k_{priv} denote a user's private key Let k_{pub} denote the corresponding public key $p = D(k_{priv}, E(k_{pub}, p))$
- with a private key, a user can decode a cipher encrypted with the corresponding public key

Second public key encryption algorithm:

$p = D(k_{pub}, E(k_{priv}, p))$

- user can encrypt a message with a private key and the msg can be decrypted with the corresponding public key

Merkle-Hellman Knapsacks

- encode a bin message as a solution to a knapsack problem, reducing the ciphertext to the target sum obtained by adding terms corresponding to 1s in the plaintext
- knapsack problem is NP-complete, implying that to solve it probably requires time exponential in the size of the problem - in this case, the number of integers
- algorithm begins with a knapsack set, where each element is larger than the sum of all previous elements

General Knapsacks

- examines a **sequence of $a_1, a_2, \ldots, a_n$ of integers** and a **target sum, T**. The problem is to **find a vector of 0s and 1s** such that the **sum of integers associated with 1s equals T**. That is given $S = [a_1, a_2, \ldots, a_n]$, and `T`, find a vector `V` of 0s and 1s such that:

$$\sum_{i=1}^{n} a_i \times v_i = T$$
    ```

Superincreasing Knapsacks (additional restriction)

- integers of `S` must for a *superincreasing* sequence, ie one where each integer is greater than the sum of all preceding integers. Then, every integer a_k would be of the form

$$a_k > sum_j = 1^k - 1a_j$$

- the solution of a superincreasing knapsack (also called a **simple knapsack**) is easy to find

Knapsack Problem as a Public Key Crypto Algorithm

- pub key is a set of integers of a knapsack problem (hard knapsack)
- private key is a corresponding superincreasing knapsack (simple knapsack)
- Merkle and Hellman: designed a technique for converting a superincreasing knapsack into a regular one
  - trick is to change the number in a nonobvious but reversible way
- we *need* a superincreasing knapsack that we can transform into a hard knapsack
- after selecting a simple knapsack `S = [s_1, s_2, ..., s_m]`, we choose a multiplier `w` and a modulus `n`
  - `n` should be greater than the sum of all s_i
  - the multiplier, `w`, should have no common factors with the modulus
  - guarantee this property by choosing a modulus that is a prime number
- replace every integer s_i in the simple knapsack with the term

$$h_i = w * s_i mod n$$

- Then, `H = [h_1, h_2, h_m]` is a hard knapsack

Weakness

- interceptor does not have to solve the basic knapsack problem to break the encryption
  - if the value of modulus `n` is known, it may be possible to determine the simple knapsack (private key)
  - may try to deduce `w` and `n` from the h_i (public key) alone

Rivest-Shamir-Adelman (RSA) Encryption

- operates with arithmetic mod `n`
- two keys, `d` and `e`, used for decryption and encryption, which are interchangable
- plaintext block p is **encrypted** as:

$$c = p^e \ mod \ n$$

- the exponentiation is performed mod `n`
  - thus factoring p^w to uncover the encrypted plaintext is difficult
- **decrypting** is done as

$$p = c^d \ mod \ n$$

- legit receiver who knows `d` simply computes `p` and recovers it w/o having to factor p^e

$$p = c^d \ mod \ n = (p^e)^d \ mod \ n = (p^d)^e \ mod \ n$$

- can apply the encrypting transformation and then the decrypting one, or vice versa
- encryption (public) key consists of the pair of integers (`e, n`)
- decryption (private) key is (`d, n`)

Steps

1. Select a value for `n`, should be quite large, a product of two primes `p` and `q`

- typically, `p` and `q` are nearly 100 digits each, so `n` is approx. 200 decimal digits (about 512 bits long)
- 768, 1024, or more bits may be more appropriate

2. A relatively large integer `e` is chosen so that `e` is relatively prime to (`p` - 1) * (`q` - 1)

- easy way to guarantee `e` is relatively prime to (`p` - 1) * (`q` - 1) is to choose `e` as a prime that is larger than both (`p` - 1) and (`q` - 1)

3. Finally, select `d` such that `e` * `d` = 1 mod (`p` - 1) * (`q` - 1)

Mathematical Foundations of the RSA Algorithm

- Euler's totient function

  $$\phi(n)$$

  is the number of positive integers less than `n` that are relatively prime to `n`. If `p` is prime, then

  $$\phi(p) = p - 1$$

- if `n` = `p * q`, where `p` and `q` are both prime, then

  $$\phi(n) = \phi(p) * \phi(q) = (p-1) * (q-1)$$

- Euler and Fermat proved that

  $$x^{\phi(n)} \equiv 1 \; mod \; n$$

  for any integer `x` if `n` and `x` are relatively prime

Cryptanalysis

- RSA-149 (463 bits) was factored in 1999 in 1 month using approx. 200 machines
- RSA-155 (512 bits) was factored in 1999 in approx. 3.7 months using 300 machines
- RSA-160 was factored in 2003 in only 20 days
- RSA-200 (663 bits) was factored in 2005 after about 18 months, using unspecified # of machines
- 1024 or longer bit keys are still secure enough to use

The El Gamal Cryptosystem

- relies on the difficulty of computing discrete lograithms over finite fields
- since it is based on arithmetic in finite fields, as is RSA, it has some similartiy to RSA
- for any given plaintext `P`, the generated ciphertext has two separate parts, c_1 and c_2

Digital Signature

- means of associating a mark unique to an individual with a body of text
- mark should be unforgeable, only originator should be able to compute the signature value
- mark should be verifiable, others should be able to check the signature comes from claimed originator
- general way of computing digital signatures is with public key encryption
  - signer computes a signature vvalue by using a private key
  - others use pub key to verify that the signature came from the corresponding private key

El Gamal Algorithm

Key Generation

- First, choose a prime `p` and two integers, `a` and `x`, such that `a` < `p` and `x` < `p` and calculate:

$$y = a^x \ mod \ p$$

- `a` should be a generator of `GF(p)`
- prime `p` should be chosen so that (`p` - 1) has a large prime factor, `q`
- **private key**: `(x, p, a)`
- **public key**: `(y, p, a)`

To sign a message `m`

- choose a random int `k | 0 < k < p - 1`, which has not been used before and which is relatively prime to (p - 1)
- compute `r = a`^{k} `mod p`
- compute `s =` k^{-1} `(m - xr) mod (p - 1)`
- message signature is then (r, s)

To verify signature

- use pub key `y` to compute `y`^{r}`r`^{s} `mod p` and determine that it is equivalent to `a`^{m} `mod p`

US Digital Signature Algorithm (DSA)

- DSA is El Gamal algo with a few restrictions:
  1. Size of `p` is fixed at 2^{511} `< p <` 2^{512} (roughly 170 dec digits)
  2. `q`, the large prime factor of (p - 1), is chosen such that 2^{159} `< q <` 2^{160}
  3. Algo explicitly uses `H(m)`, a hash value, instead of a the full msg text `m`
  4. Computations of `r` and `s` are taken `mod q`

Secret Sharing

- sharing a secret among a group of participants
- secret can be reconstructed only when a sufficient number (threshold `t`) of shares are participated in the decryption process
- shown by `(t, n)` in which `n` is the total number of partipants, `t` is the min number of required participants for secret reconstruction
- limitations to all unconditionally secure secret sharing schemes:
  1. Each share must be at elast as large as the secret itself
  – based on info theory, given `t-1` shares, no info can be revealed from the secret. Thus, all shares must ocntain as much info as the secret itself
  2. All secret sharing schemes use random bits
  – to distribute a 1-bit secret among `t` participants, `t-1` random bits are necessary

Trivial Secret Sharing

- encode secret as arbitrary length bin number `s`
- give ot each player `i` (except the last one) a random number `p`$_{i}$ with the same length as `s`
- give to last player the result of `s XOR p`$_{1}$ `XOR p`$_{2}$`XOR ... XOR p`$_{n-1}$

Shamir's Secret Sharing Scheme

- we can fit a unique polynomial of degree `(t-1)` to any set of `t` points that lie on the polynomial
- method:
  - create a polynomial of degree `t-1`
  - set the secret as the first coefficient
  - remaining coefficients are selected at random
  - select `n` points on the curve as the shares
- it's sufficient that at least `t` out of `n` participants use their shares to fit a `(t-1)` degree polynomial to them
- the first coefficient would be the secret

Secret Reconstruction

- select any 3 shares
- use polynomial interpolation method, such as *Lagrange*, to find coefficients of the polynomial

Proactive Secret Sharing

- method to periodically update the shares, attacker has less time to compromise the shares
- each participant `i` selects `i` random numbers from finite field and creates a polynomial of degree `t`

$$\delta_i(z) = \delta_{i_1}(z^1) + \delta_{i_2}(z^2) + ... + \delta_{i_t}(z^t)$$

- participant i secretly sends

$$\delta_i$$

  `(j)` to the participant `j`
- participant `i` computes the new share as:

$$s_i = s_i + (\delta_i(1) + \delta_i(2) + ... + \delta_i(n))$$

- old shares will be discarded, new shares used for secret reconstruction