S̃tudy Notes

Last updated on : April 29, 2014

Contents

1 Cryptanalysis and Attacks on Cryptosystems

This section is study notes on Cryptanalysis and Attacks on Cryptosystems
Reference :
`http://math.colorado.edu/~hiba/crypto/cryptanalysis.html`

Cryptanalysis is the art of deciphering encrypted communications without knowing the proper keys. There are many cryptanalytic techniques. Some of the more important ones for a system implementer are described below.

1.1 Ciphertext-only attack

This is the situation where the attacker does not know anything about the contents of the message, and must work from ciphertext only. In practice it is quite often possible to make guesses about the plaintext, as many types of messages have fixed format headers. Even ordinary letters and documents begin in a very predictable way. For example, many classical attacks use frequency analysis of the ciphertext, however, this does not work well against modern ciphers.

Modern cryptosystems are not weak against ciphertext-only attacks, although sometimes they are considered with the added assumption that the message contains some statistical bias.

1.2 Known-plaintext attack

The attacker knows or can guess the plaintext for some parts of the ciphertext. The task is to decrypt the rest of the ciphertext blocks using this information. This may be done by determining the key used to encrypt the data, or via some shortcut.

One of the best known modern known-plaintext attacks is linear cryptanalysis against block ciphers.

1.3 Chosen-plaintext attack

The attacker is able to have any text he likes encrypted with the unknown key. The task is to determine the key used for encryption.

A good example of this attack is the differential cryptanalysis which can be applied against block ciphers (and in some cases also against hash functions).

Some cryptosystems, particularly RSA, are vulnerable to chosen-plaintext attacks. When such algorithms are used, care must be taken to design the application (or protocol) so that an attacker can never have chosen plaintext encrypted.

1.4 Man-in-the-middle attack

This attack is relevant for cryptographic communication and key exchange protocols. The idea is that when two parties, A and B, are exchanging keys for secure communication (e.g., using Diffie-Hellman), an adversary positions himself between A and B on the communication line. The adversary then intercepts the signals that A and B send to each other, and performs a key exchange with A and B separately. A and B will end up using a different key, each of which is known to the adversary. The adversary can then decrypt any communication from A with the key he shares with A, and then resends the communication to B by encrypting it again with the key he shares with B. Both A and B will think that they are communicating securely, but in fact the adversary is hearing everything.

The usual way to prevent the man-in-the-middle attack is to use a public key cryptosystem capable of providing digital signatures. For set up, the parties must know each others public keys in advance. After the shared secret has been generated, the parties send digital signatures of it to each other. The man-in-the-middle can attempt to forge these signatures, but fails because he cannot fake the signatures.

This solution is sufficient in the presence of a way to securely distribute public keys. One such way is a certificate hierarchy such as X.509. It is used for example in IPSec.

Correlation between the secret key and the output of the cryptosystem is the main source of information to the cryptanalyst. In the easiest case, the information about the secret key is directly leaked by the cryptosystem. More complicated cases require studying the correlation (basically, any relation that would not be expected on the basis of chance alone) between the observed (or measured) information about the cryptosystem and the guessed key information.

For example, in linear (resp. differential) attacks against block ciphers the cryptanalyst studies the known (resp. chosen) plaintext and the observed ciphertext. Guessing some of the key bits of the cryptosystem the analyst determines by correlation between the plaintext and the ciphertext whether she guessed correctly. This can be repeated, and has many variations.

The differential cryptanalysis introduced by Eli Biham and Adi Shamir in late 1980's was the first attack that fully utilized this idea against block ciphers (especially against DES). Later Mitsuru Matsui came up with linear cryptanalysis which was even more effective against DES. More recently, new

attacks using similar ideas have been developed.

Perhaps the best introduction to this material is the proceedings of EUROCRYPT and CRYPTO throughout the 1990's. There can be found Mitsuru Matsui's discussion of linear cryptanalysis of DES, and the ideas of truncated differentials by Lars Knudsen (for example, IDEA cryptanalysis). The book by Eli Biham and Adi Shamir about the differential cryptanalysis of DES is the "classical" work on this subject.

The correlation idea is fundamental to cryptography and several researchers have tried to construct cryptosystems which are provably secure against such attacks. For example, Knudsen and Nyberg have studied provable security against differential cryptanalysis.

Attack against or using the underlying hardware: in the last few years as more and more small mobile crypto devices have come into widespread use, a new category of attacks has become relevant which aim directly at the hardware implementation of the cryptosystem.

The attacks use the data from very fine measurements of the crypto device doing, say, encryption and compute key information from these measurements. The basic ideas are then closely related to those in other correlation attacks. For instance, the attacker guesses some key bits and attempts to verify the correctness of the guess by studying correlation against her measurements.

Several attacks have been proposed such as using careful timings of the device, fine measurements of the power consumption, and radiation patterns. These measurements can be used to obtain the secret key or other kinds information stored on the device.

This attack is generally independent of the used cryptographical algorithms and can be applied to any device that is not explicitly protected against it.

Faults in cryptosystems can lead to cryptanalysis and even the discovery of the secret key. The interest in cryptographical devices lead to the discovery that some algorithms behaved very badly with the introduction of small faults in the internal computation.

For example, the usual implementation of RSA private key operations are very susceptible to fault attacks. It has been shown that by causing one bit of error at a suitable point can reveal the factorization of the modulus (i.e. it reveals the private key).

Similar ideas have been applied to a wide range of algorithms and devices. It is thus necessary that cryptographical devices are designed to be

highly resistant against faults (and against malicious introduction of faults by cryptanalysts).

1.5 Quantum computing

Peter Shor's paper on polynomial time factoring and discrete logarithm algorithms with quantum computers has caused growing interest in quantum computing. Quantum computing is a recent field of research that uses quantum mechanics to build computers that are, in theory, more powerful than modern serial computers. The power is derived from the inherent parallelism of quantum mechanics. So instead of doing tasks one at a time, as serial machines do, quantum computers can perform them all at once. Thus it is hoped that with quantum computers we can solve problems infeasible with serial machines.

Shor's results imply that if quantum computers could be implemented effectively then most of public key cryptography will become history. However, they are much less effective against secret key cryptography.

Current state of the art of quantum computing does not appear alarming, as only very small machines have been implemented. The theory of quantum computation gives much promise for better performance than serial computers, however, whether it will be realized in practice is an open question.

Quantum mechanics is also a source for new ways of data hiding and secure communication with the potential of offering unbreakable security, this is the field of quantum cryptography. Unlike quantum computing, many successful experimental implementations of quantum cryptography have been already achieved. However, quantum cryptography is still some way off from being realized in commercial applications.

1.6 DNA cryptography

Leonard Adleman (one of the inventors of RSA) came up with the idea of using DNA as computers. DNA molecules could be viewed as a very large computer capable of parallel execution. This parallel nature could give DNA computers exponential speed-up against modern serial computers.

There are unfortunately problems with DNA computers, one being that the exponential speed-up requires also exponential growth in the volume of the material needed. Thus in practice DNA computers would have limits on their performance. Also, it is not very easy to build one.

2 Strength of Cryptographic Algorithms

This section is study notes on Strength of Cryptographic Algorithms
Reference : `http://math.colorado.edu/~hiba/crypto/strength.html`

Good cryptographic systems should always be designed so that they are as difficult to break as possible. It is possible to build systems that cannot be broken in practice (though this cannot usually be proved). This does not significantly increase system implementation effort; however, some care and expertise is required. There is no excuse for a system designer to leave the system breakable. Any mechanisms that can be used to circumvent security must be made explicit, documented, and brought into the attention of the end users.

In theory, any cryptographic method with a key can be broken by trying all possible keys in sequence. If using brute force to try all keys is the only option, the required computing power increases exponentially with the length of the key. A 32 bit key takes 232 (about 109) steps. This is something anyone can do on his/her home computer. A system with 40 bit keys takes 240 steps - this kind of computation requires something like a week (depending on the efficiency of the algorithm) on a modern home computer. A system with 56 bit keys (such as DES) takes a substantial effort (with a large number of home computers using distributed effort, it has been shown to take just a few months), but is easily breakable with special hardware. The cost of the special hardware is substantial but easily within reach of organized criminals, major companies, and governments. Keys with 64 bits are probably breakable now by major governments, and within reach of organized criminals, major companies, and lesser governments in few years. Keys with 80 bits appear good for a few years, and keys with 128 bits will probably remain unbreakable by brute force for the foreseeable future. Even larger keys are sometimes used.

However, key length is not the only relevant issue. Many ciphers can be broken without trying all possible keys. In general, it is very difficult to design ciphers that could not be broken more effectively using other methods. Designing your own ciphers may be fun, but it is not recommended for real applications unless you are a true expert and know exactly what you are doing.

One should generally be very wary of unpublished or secret algorithms. Quite often the designer is then not sure of the security of the algorithm, or its security depends on the secrecy of the algorithm. Generally, no algorithm that depends on the secrecy of the algorithm is secure. Particularly in software, anyone can hire someone to disassemble and reverse-engineer the algorithm. Experience has shown that the vast majority of secret algorithms that have become public knowledge later have been pitifully weak in reality.

The key lengths used in public-key cryptography are usually much longer

than those used in symmetric ciphers. This is caused by the extra structure that is available to the cryptanalyst. There the problem is not that of guessing the right key, but deriving the matching secret key from the public key. In the case of RSA, this could be done by factoring a large integer that has two large prime factors. In the case of some other cryptosystems it is equivalent to computing the discrete logarithm modulo a large integer (which is believed to be roughly comparable to factoring when the moduli is a large prime number). There are public key cryptosystems based on yet other problems.

To give some idea of the complexity for the RSA cryptosystem, a 256 bit modulus is easily factored at home, and 512 bit keys can be broken by university research groups within a few months. Keys with 768 bits are probably not secure in the long term. Keys with 1024 bits and more should be safe for now unless major cryptographical advances are made against RSA; keys of 2048 bits are considered by many to be secure for decades.

It should be emphasized that the strength of a cryptographic system is usually equal to its weakest link. No aspect of the system design should be overlooked, from the choice algorithms to the key distribution and usage policies.

3 Diffie Hellman Scheme

This section is study notes on Diffie Hellman Computational assumption, Decisive assumption and Key Exchange protocol.

The **Diffie-Hellman assumption's** are the assumption's that a certain computational problem within a cyclic group is hard.

3.1 Computational Diffie-Hellman assumption

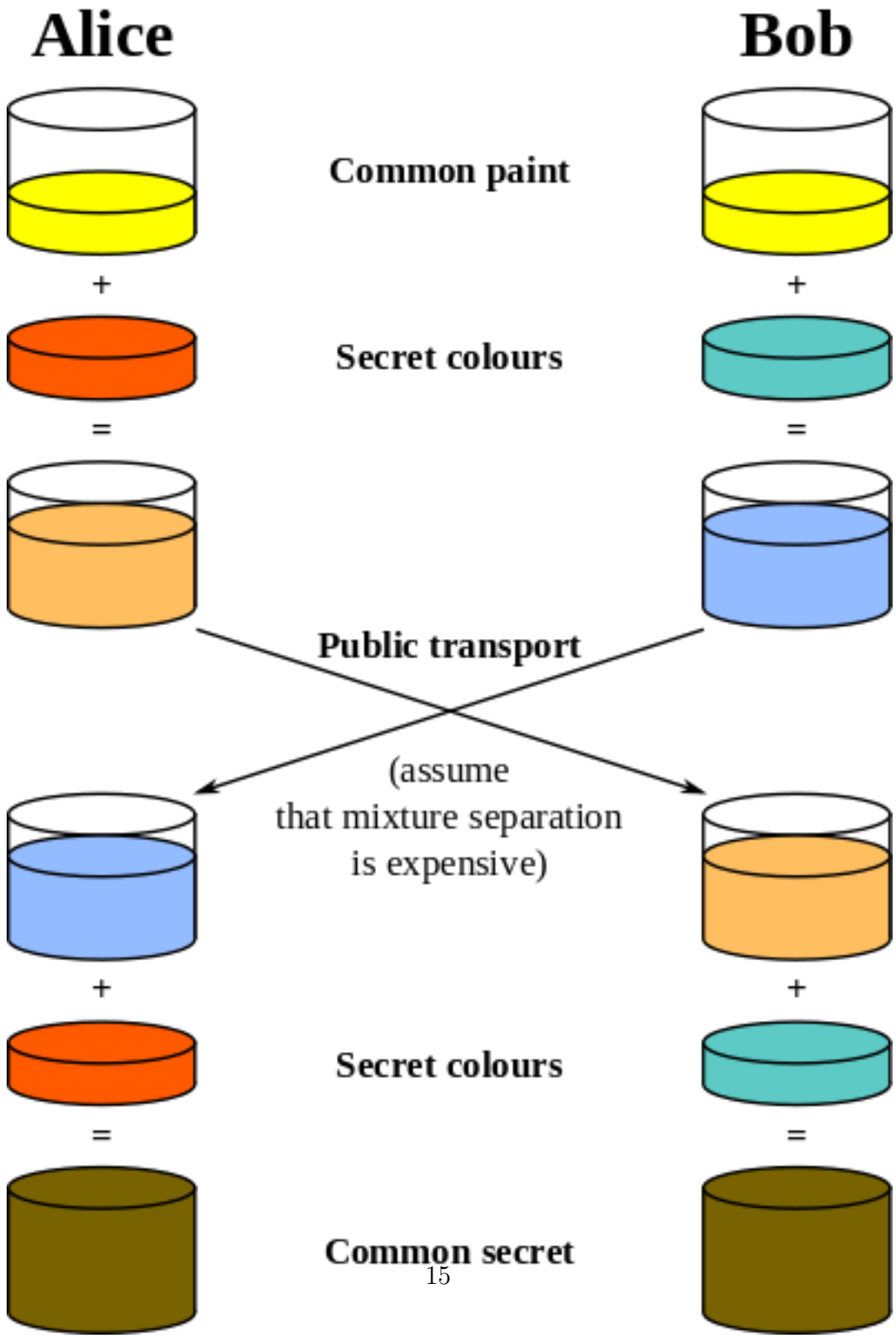
- The CDH assumption is related to the ***discrete logarithm assumption***.
- Given (g, g^a, g^b) its hard to compute g^{ab}

3.2 Decisional Diffie-Hellman assumption

- Chosen $a, b \in Z_p$, Given $(g, a, b, g^a, g^b,)$ and the value g^{ab} "looks like" a random element in G

3.3 Diffie Hellman Key Exchange

- Generate tow prime numbers (g, p)
- Communicate (g, p) and consider its available with both *Alice* and *Bob*
- Alice picks a random secret, a and computes
  - $A = g^a \bmod p$
- Bob picks a random secret, b and computes
  - $B = g^b \bmod p$
- Alice and Bob exchange A and B
- Alice Computes, A' as following
  - $A' = B^a \bmod p = (g^b \bmod p)^a \bmod p$
- Bob Computes, B' as following
  - $B' = A^b \bmod p = (g^a \bmod p)^b \bmod p$
- Alice and Bob exchanges A' and B'
- Now, both have $(g^{ab} \bmod p)$


Alice
Bob
Common paint
+
+
Secret colours
=
=
Public transport
(assume
that mixture separation
is expensive)
+
+
Secret colours
=
=
Common secret

4 Malleable Encryption

A cryptographic algorithm is considered to be *malleable*, if an encrypted cipher text of message m, can be converted into another cipher text, which can be decrypted with known function *f(m)*, without prior knowledge of m. Malleability does not refer to ability to read the encrypted message.

4.1 Malleability in Encryption

Generally, *Malleability* is an undesirable property for any crypto system. But for certain special cases, like *Homomorphic Encryption,* [: form of encryption which allows specific type of operations on cipher text, to generate a cipher text, which when decrypted matches to the operations performed on plain text.]

4.1.1 RSA Example

To demonstrate malleability, we can consider RSA encryption scheme.

RSA Encryption scheme for encrypting a message m, using the public key pair (e,n), can be expressed as,

$$E(m) = (m)^e \bmod n$$

Given the above parameters, except m, it is still possible to construct encryption of message *E(mt)* as below.

$$E(m).t^e \bmod n = ((m)^e \bmod n).(t^e \bmod n) = ((m.t)^e \bmod n) = E(mt)$$

5 ElGamal Encryption

This section is study notes on ElGamal Encryption Scheme, a *asymmetric key encryption* algorithm, based on *Diffie-Helman Key exchange*, proposed by *Taher Elgamal*, in 1985.

ElGamal encryption can be defined over any cyclic group G.
Its security depends upon the **difficulty of computing discrete logarithms.**

5.1 Algorithm

ElGamal encryption consists of three components:

- Key generator,
- Encryption algorithm,
- Decryption algorithm.

5.1.1 Key generation

- Define **cyclic group** G, of order q with generator g.
- Choose a random number x from {1,..q-1}
- Compute $p = g^x$

- **The tuple (G, q,g,p) is public key.**
- x **is the private key**

5.1.2 Encryption

- Randomly choose y from {1,..q-1}
- Calculate shared secret $s = p^y$
- Covert message m into m' of G
- Calculate $c_1 = g^y$
- Calculate $c_2 = m'.s$
- Calculate $(c_1, c_2) = (g^y, m' \cdot p^y) = (g^y, m' \cdot (g^x)^y)$

- If m is known, m' can be calculated from which, p^y can be calculated

5.1.3 Decryption

- calculates the shared secret $s = c_1^x$
- computes $m' = c_2 \cdot s^{-1}$
- $c_2 \cdot s^{-1} = m' \cdot p^y \cdot (g^{xy})^{-1} = m' \cdot g^{xy} \cdot g^{-xy} = m'$.

5.2 Security of ElGamal Encryption

- The security of the ElGamal scheme depends on the properties of the underlying group G as well as any padding scheme used on the messages.
- If the *computational DiffieHellman assumption* (CDH) holds in the underlying cyclic group G, then the encryption function is one-way.
- If the *decisional DiffieHellman assumption* (DDH) holds in G, then ElGamal achieves *semantic security*.
- Elgamal Encryption is ***Unconditionally Malleable***
- Hence it is ***Not secure under Chosen Ciphertext attack*** *(CCA)*
- i.e. If (c_1, c_2) are given for any message m, its easy to calculate, $(c_1, 2c_2)$ in order to decrypt message $2m$
- To achieve CCA security, appropriate padding scheme should be used.

6 Certificate Less Key Exchange, In Presence of a Strong Adversary

Exchanging two public key, over an insecure channel or without any presence of earlier shared secret, always needs a certificate to be issued by a *Trusted Third party*, aka, *Certifying Authority*. But this is not always possible, especially in *Ad hoc* networks. We propose a solution to such a scenario, to exchange the key, over an insecure channel, in presence of such the strong adversary, who has access to all of the messages transmitted over the channel and who can modify the message if he wishes.

7 Zero Knowledge Proof

This section is study notes on Zero Knowledge Proof Protocol.

7.1 Introduction

- A Zero-knowledge proof or Zero-knowledge protocol is a protocol used in cryptography.
- The are two parties communicating.
- The task of one party is to convince the other that it knows some secret, without revealing the secret.
- The prover of the statement is called as Peggy
- The verifier of the statement is Victor
- It requires some secret information on the part of the prover
- The definition implies that the verifier will not be able to prove the statement to anyone else
- An real-world example of such a protocol is the Feige-Fiat-Shamir Identification Scheme.

7.2 Story

7.2.1 Epilogue

- Peggy has uncovered the secret word used to open a magic door in a cave
- The cave is shaped like a circle, with entrance on one side & magic door blocking the opposite side.
- Victor says he'll pay her for the secret, but not until he's sure that she really knows it.
- Peggy says she'll tell him the secret, but not until she receives the money.
- They devise a scheme by which Peggy can prove that she knows the word without telling it to Victor.

7.2.2 The Story

- First, Victor waits outside the cave as Peggy goes in.
- They label the left and right paths from the entrance A and B.
- Peggy randomly takes either path A or B.
- Then, Victor enters the cave and shouts the name of the path he wants her to use to return
- Providing she really does know the magic word, this is easy:
- she opens the door, if necessary, and returns along the desired path.
- Note that Victor does not know which path she has gone down.
- The correctness can only be 50% in this scheme
- If they were to repeat this trick many times, say 20 times in a row,
- => her chance of successfully anticipating all of Victor's requests would become very small.
- For this reason, if Peggy reliably appears at the exit Victor names, he can conclude that ***she is very likely*** to know the secret word.

7.3 Properties of ZKP

A zero-knowledge proof must satisfy three properties:

1. Completeness
2. Soundness
3. Zero Knowledge

Completeness : if the statement is true, the honest verifier (that is, one following the protocol properly) will be convinced of this fact by an honest prover.

Soundness : if the statement is false, no cheating prover can convince the honest verifier that it is true, except with some small probability.

Zero Knowledge : The verifier should know only the fact that prover can prove.

- The first two of these are properties of more general interactive proof systems.
- The third is what makes the proof zero-knowledge.

7.4 The soundness error

The small probability that prover can use to cheat verifier is called as **The soundness error**. ZKP are probabilistic "proofs" rather than deterministic proofs.

8 English Vocabulary

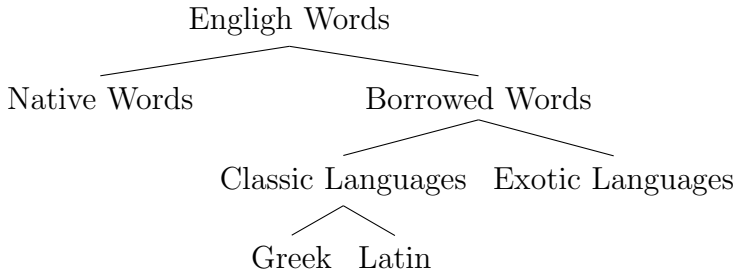
This is study notes on The Structure of English Words (LING150) course from http://darkwing.uoregon.edu/ l150web/index.html

8.1 Course Organization

- **Unit 1:** Basic Word Analysis
  - Historical sources of English words
  - Basic principles of word analysis (morphology)
- **Unit 2:** Intermediate Word Analysis and Basic Phonetics
  - Alternations in morpheme forms (allomorphy)
  - Basic principles of English sounds (phonetics)
- **Unit 3:** Advanced Word Analysis and Semantic Change
  - More alternations in morpheme forms (allomorphy rules)
  - Historical changes of meanings (semantic change)
- **Unit 4:** - The Origins and History of English
  - Pre-history of English and Indo-European languages
  - English history

8.2 Introduction to English vocabulary

8.2.1 Sources of English words



The native words are words which are used in evryday conversations. The native words falls into follwoing categories.

Type	Example
Body Parts	Foot, mouth, hand, leg
Family Relationships / Kinship terms	Father, mother, brother, sister
Everyday, Natural objects	rock, house, hill
Physical Acts	think, drive, ride
Physical charactrestics	red, cold, young

8.2.2 Learning Model

Learning new words and improving new vocabulary can be carried out by the following ways.

- Absorption
- Memorization
- Analysis

8.3 Analyzing words

8.3.1 Morphemes

Word analysis involves breaking a word into its morphemes. Literally morphemes means ***an element in a system of forms.***
In linugustics, it is defined as

"The smallest ***form*** which is paired with a particular ***meaning***"

From Dictionary: ***morpheme***, noun, very rare

Minimal meaningful language unit;
It cannot be divided into smaller meaningful units;

8.3.2 Characteristics of morphemes

Morphemes have four defining characteristics:

- They cannot be subdivided.
- They add meaning to a word.
- They can appear in many different words.

- They can have any number of syllables.

8.3.3 Problems with morphemes

There is no one-to-one mapping between *FORM* $\leftrightarrow$ *MEANING*

- One form, two (or more) meanings.

  in- '*not*' in words like **in**capable and **in**sufficient, &
  in- '*into, within*', as in **in**vade and **in**clude.

- Two (or more) forms, one meaning.

  - Two (or more) forms, one meaning = two morphemes

    andr- '*man,male*' in words like **and**roid &
    vir- '*man,male*', as in **vir**ile

  - Two (or more) forms, one meaning = one morpheme

    pan- and **pant-** , which are different forms of a Greek morpheme meaning 'all,overall'.

The alternate forms of a single morpheme are called **allomorphs**,literally 'other forms.'

8.3.4 How to analyze words

- Only Words borrowed from Classic languages like Greek and Latin can be analyzed
- Native words or words borrowed from exotice languages can not be analyzed

There are four steps as following in analyzing a word

- Parse : Divide it into its morpheme
- Gloss : Give the meaning for a morpheme
- Give a literal meaning : Use the meanings from the glosses to construct a literal meaning

- Give a dictionary definition : Extended literal meaning metaphorically to arrive at the actual meaning.

Example :

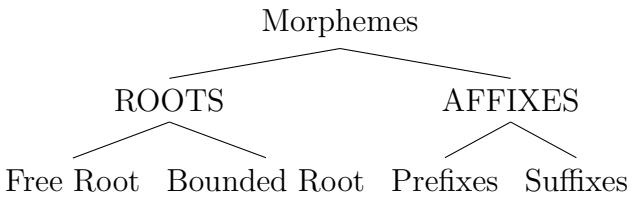
- **repellent**
- re- / pel / (l) / -ent
- 'again, back' / 'push' / (l) / A, N
- 'something which pushes back' / (l) /A,N

'A, N.' : The letters stand for 'ADJECTIVE, NOUN'

8.4 Words - their construction and use

1. How words are constructed
   (a) Types of Roots
   (b) Types of Affixes
2. How words are used - NOUNS, VERBS and ADJECTIVES

8.4.1 How words are constructed



Roots

- Roots usually have a rather specific meaning
- This meaning tends to be relatively constant across all the words that use the root.
- Roots also contribute the greatest conceptual content to the overall meaning of the word.
- Every word has at the least one root.

- Roots have freer distribution; that is, they can occur almost anywhere in a word
- Two ore more roots combine to form Compound words.
- Compound words may or may not have affixes.

Free Roots

- Free Roots are roots that can occur alone as whole words
- They can stand alone as single words.
- Free roots can also be combined with other roots or affixes to form more complex words

Bounded Roots

- Bound Roots can never occur alone as whole words.
- They cannot stand alone;
- They must occur in combination with other morphemes
- Almost all the Latin and Greek roots we are studying are bound roots

Affixes

- Affixes are morphemes which attach to roots or a combination of roots and other affixes.
- Their main use is to modify the meaning conveyed by the root or roots.
- Affixes by definition are always bound or affixed to a root.
- They are divided into two different types depending on where they attach to the root.
- Prefixes occur before a root (although several prefixes can be strung together before a single root).
- Suffixes occur after a root (although multiple suffixes can occur at the ends of words).

8.5 Suffixes and Prefixes

1. Suffixes
   (a) Inflectional suffixes
      - Add only grammatical information
      - Never change the part of speech

Rule	Explanation
-s 'plural'	The two girls had eaten dinner
-'s 'possessive'	The cat's tail was twitching
-ed 'past tense'	The blackest dog never barked
-s '3rd person present tense'	The smaller dog barks a lot
-ing 'present participle'	The cat's tail was twitching
-en 'past participle'	The two girls had eaten dinner
-er 'comparative'	The smaller dog barks a lot
-est 'superlative'	The blackest dog never barked

   (b) Derivational suffixes
      - Make a new word with a new meaning
      - Usually change the part of speech
   (c) Meanings of Derivational Suffixes
2. Prefixes
   (a) Spatial Prefixes
      - The largest group of prefixes denote relationships that occur in space.
   (b) Non-Spatial Prefixes
      - Comparative relations
      - Quantity and Size
      - Negative Prefixes
      - Intensive prefixes

9 Number-Theoretic Reference Problems

9.1 Introduction

- Security of many public-key cryptosystems dependent on intractability of the computational problems.
- A problem is tractable if its possible to solve in polynomial time.
- Problem A can be said as reducable to Problem B, if A can be solved by employing B and other finite operations ξ.
- If A $\leq_p$ B, (i.e, A is reducable to B), then A is *Computationaly Equivalent* to B, written as A $\equiv_p$ B
- If A $\equiv_p$ B, then both are tractabe or intractable.

9.2 Integer Factorization Problem

- Given a positive integer n, find its prime factors;
- Write n $\mid n = p_1^{e_1} p_2^{e_2} ... p_k^{e_k}$, where p_i are pairwise primes and $e_i \geq 1$.
- A set of integers is said to be pairwise coprime if a and b are coprime for every pair (a, b) of different integers in it.

10 Cryptographic Primitives

10.1 Introduction

A symmetric or private-key cipher is one in which knowledge of the encryption key is explicitly or implicitly equivalent to knowing the decryption key.

An asymmetric or public-key cipher is one in which the encryption key is effectively public knowledge, without giving any useful information about the decryption key.

trapdoor is a computation which runs much more easily in one direction than back.

Example : Multiplication of large integers is easy, while factoring of large integers is difficult.
Merely testing large numbers for primality is easy.

10.2 RSA Cipher

1. Generate two primes, p and q, such that, atleast $> 2^{512}$ or preferably $> 2^{1024}$

2. Compute **RSA Modulus, n =p.q**

3. Choose encryption exponent, **e** > 2, **randomly.**
   Often, its **Fermats number** [A positive integer, of form, $F_n = 2^{2^n} + 1$]

$$\text{e} = 2^{16} + 1 = 65537$$

4. Make $\boldsymbol{n}$ and $\boldsymbol{e}$ public

5. Compute decryption exponent, such that,

$$e.d = 1 \ mod \ (p-1)(q-1)$$

6. Encryption : $E_{n,e}(x) = x^e \ \% \ n \Rightarrow y$

7. Decryption : $D_{n,d}(y) = y^d \ \% \ n \Rightarrow x$

11 Logarithms

11.1 Logarithm definition

If, a number b is raised to a power x, such that,

$$b^x = y$$

Then the base b logarithm of x is equal to y:

$$\log_b(x) = y$$

Then, we can conjure logarithms as the inverse of exponentiation, like

$$b^y = x$$

Example :

$$\log_2(32) = 5 \Rightarrow 2^5 = 32$$
$$\log_2(16) = 4 \Rightarrow 2^4 = 16$$
$$\log_2(8) = 3 \Rightarrow 2^3 = 8$$

11.2 Logarithms Rules

Rule	**Explanation**
Logarithm product rule	$log_b(x.y) = log_b(x) + log_b(y)$
Logarithm quotient rule	$log_b(x/y) = log_b(x) - log_b(y)$
Logarithm power rule	$log_b(x^y) = y.log_b(x)$
Logarithm base switch rule	$log_b(c) = 1/log_c(b)$
Logarithm base change rule	$log_b(x) = log_c(x)/log_c(b)$
Derivative of logarithm	$f(x) = log_b(x) \Rightarrow f^{'}(x) = 1/(xln(b))$
Integral of logarithm	$\int log_b(x)dx = x.(log_b(x) - 1/ln(b)) + C$
Logarithm of negative number	$log_b(x)$ is undefined when $x \leq 0$
Logarithm of 0	$log_b(0)$ is undefined $\lim_{x \to 0^+} \log_b(x) = -\infty$
Logarithm of 1	$log_b(1) = 0$
Logarithm of the base	$log_b(b) = 1$

12 English Syllables

Divide words based on the number of vowels: The number of syllables in a word coincide with the number of vowel sounds you hear when speaking the word.

Example : Ba | *na* | na

Vowels: 'a,' 'e,' 'i,' 'o,' 'u' and 'y' are considered as vowels, when found in the middle or at the end of a word.

Subtract 1 for each silent vowel in the word: Example : 'came' and 'gone,' 'E' is the most common silent vowel.

Consider words ending in 'le' as a vowel: Divide the word before the consonant right before the 'le'.
Example : Little would be divided lit/tle; fumble as fum/ble; able as a/ble.

Count diphthongs as 1 syllable instead of 2 : diphthongs are two vowels, combined giving one syllable. Example : Words like 'haul,' 'moon,' and 'coil' are diphthongs.

Split words between consonants, surrounded by vowels. Example : hap/py, din/ner, bas/ket, un/der.

Find syllables by splitting a word before a single consonant. Examples are: au/tumn, o/pen, de/tail:

Mark prefixes and suffixes as syllabels:

13 Group

A group G is a finite or infinite set of elements together with a binary operation (called the group operation) that together satisfy the four fundamental properties of #1 closure, #2 associativity, #3 the identity property, and #4 the inverse property.

1. Closure: If A and B are two elements in G, then the product AB is also in G.

2. Associativity: The defined multiplication is associative, i.e., for all A,B,C in G, (AB)C=A(BC).

3. Identity: There is an identity element I (a.k.a. 1, E, or e) such that IA=AI=A for every element A in G.

4. Inverse: There must be an inverse (a.k.a. reciprocal) of each element. Therefore, for each element A of G, the set contains an element B=A$\hat{}$(-1) such that AA$\hat{}$(-1)=A$\hat{}$(-1)A=I.

Group Theory: Study of groups.

Finite Group: The group has a finite number of elements.

Subgroup: A subset of group, which is *closed* under *group operation* is called as subgroup.

Cyclic Group: Cyclic group is a group, that is generated by a single element. That is,

- It consists of a set of elements, with single invertible associative operation and
- a element g, such that, every other element in the group can be obtained by applying the group operation or its inverse to g.

Cyclic group is closed under addition, associative and has unique inverse.

Homomorphism: A map between two groups, which preserve identity and group operation is called Homomorphism.

Isomorphism: If a homomorphism has an inverse which is also an homomorphism, this is called as *isomorphism* and the two group is said to be isomorphic.

14 Android Services

We will see,

1. What is a Service?
2. Service Life-cycle
3. Permissions
4. Process Life-cycle
5. Local Service Sample
6. Remote Messenger Service Sample

14.1 Introduction

A Service is an application component, representing either

- an application's desire to perform a longer-running operation while not interacting with the user. *This corresponds to calls to* ***Context.startService()***
- to supply functionality for other applications to use. *This corresponds to calls to* ***Context.bindService()***

Service is

- Not a seperate process
- Not a seperate thread

Each service class must have a corresponding <service> declaration in its package's ***AndroidManifest.xml.***
Services can be started with ***Context.startService()*** and ***Context.bindService().***

14.2 Service Lifecycle

If someone calls *Context.startService()* $\longrightarrow$ *onCreate()* $\longrightarrow$ *onStartCommand(Intent, int, int)* $\longrightarrow$ run until *Context.stopService()* or *stopSelf()* is called

START_STICKY is used for services that are explicitly started and stopped as needed.
START_NOT_STICKY or *START_REDELIVER_INTENT* are used for services that should only remain running while processing any commands sent to them.
onDestroy() method is called and the service is effectively terminated.

14.3 Permissions

Global access to a service can be enforced when it is declared in its manifest's <service> tag. By doing so, other applications will need to declare a corresponding <uses-permission> element in their own manifest to be able to start, stop, or bind to the service.
This can be bypassed by using *Intent.FLAG_GRANT_READ_URI_PERMISSION* and/or *Intent.FLAG_GRANT_WRITE_URI_PERMISSION* on the Intent, when using *Context.startService(Intent)*.

15 Two step verification

15.1 Factors of Authentication

Factor	Explanation	Example
Ownership factor	Something that user has	ID Card, A hardware, Token
Knowledge factor	Something that user knows	Password, PIN
Inherence factor	Something that is inhere to user	Biometrics like fingerprint, retina etc

15.2 Two step verification

For a secured and positive authentication, atleast there should be two or all three factors involve in authentication.

Single Sign On A method to authenticate in a system which has multiple secured, independent systems. By this Single Sig On, a user logins in and autheticates one system and gains acess to all the systems.

16 Newline representation

Carriage Return [CR] : Reset typewriters horizontal position to the far left.

Line Feed [LF] : Advance the paper by one line.

ASCII Based systems

A type writer typically needs both. Windows uses CR+LF. Linux uses only LF. MAC OS, prior to OS-X used CR alone.

IBM Pc's based on EBCDIC uses a special character called as *Newline **[NL]***

17 Turing Machine

A Turing machine can be thought of as a primitive, abstract computer. Alan Turing, who was a British mathematician and cryptographer, invented the Turing machine as a tool for studying the computability of mathematical functions. Turing's hypothesis (also known has Church's thesis) is a widely held belief that a function is computable if and only if it can be computed by a Turing machine. This implies that Turing machines can solve any problem that a modern computer program can solve. There are problems that can not be solved by a Turing machine (e.g., the halting problem); thus, these problems can not be solved by a modern computer program.

17.1 Components of a Turing Machine

A Turing machine has

- an infinite tape that consists of adjacent cells (or squares).
- On each cell is written a symbol. The symbols that are allowed on the tape are finite in number and include the blank symbol.
- Each Turing machine has it's own alphabet (i.e., finite set of symbols), which determines the symbols that are allowed on the tape.
- a tape head that is positioned over a cell on the tape. This tape head is able to read a symbol from a cell and write a new symbol onto a cell.
- The tape head can also move to an adjacent cell (either to the left or to the right).
- A Turing machine has a finite number of states, and, at any point in time, a Turing machine is in one of these states.
- A Turing machine begins its operation in the start state.
- A Turing machine halts when it moves into one of the halt states.

17.2 Opeartion of a Turing Machine

1. The Turing machine reads the tape symbol that is under the Turing machine's tape head. This symbol is referred to as the current symbol.

2. The Turing machine uses its transition function to map the following:

   {the current state and current symbol} = {the next state, the next symbol and the movement for the tape head.}

3. The Turing machine changes its state to the next state, which was returned by the transition function.

4. The Turing machine overwrites the current symbol on the tape with the next symbol, which was returned by the transition function.

5. The Turing machine moves its tape head one symbol to the left or to the right, or does not move the tape head, depending on the value of the 'movement' that is returned by the transition function.

6. If the Turing machine's state is a halt state, then the Turing machine halts. Otherwise, repeat sub-step #1.

18 NP Complete

Decision problem : Any problem with an answer, YES or NO.

P : A decision problem that can be solved in polynomial time. That is, given an instance of the problem, the answer yes or no can be decided in polynomial time.

Example : Given a graph connected G, can its vertices be colored using two colors so that no edge is monochromatic. Algorithm: start with an arbitrary vertex, color it red and all of its neighbors blue and continue. Stop when you run our of vertices or you are forced to make an edge have both of its endpoints be the same color.

NP : A decision problem where instances of the problem for which the answer is yes have proofs that can be verified in polynomial time. This means that if someone gives us an instance of the problem and a certificate (sometimes called a witness) to the answer being yes, we can check that it is correct in polynomial time.

Example: Integer factorization is NP. This is the problem that given integers n and m, is there an integer f with 1 ¡ f ¡ m such that f divides n (f is a small factor of n)? This is a decision problem because the answers are yes or no. If someone hands us an instance of the problem (so they hand us integers n and m) and an integer f with 1 ¡ f ¡ m and claim that f is a factor of n(the certificate) we can check the answer in polynomial time by performing the division n / f.

NP-complete : An NP problem X for which it is possible to reduce any other NP problem Y to X in polynomial time. Intuitively this means that we can solve Y quickly if we know how to solve X quickly. Precisely, Y is reducible to X if there is a polynomial time algorithm f to transform instances x of X to instances y = f(x) of Y in polynomial time with the property that the answer to x is yes if and only if the answer to f(x) is yes.

Example: 3-SAT. This is the problem wherein we are given a conjunction of 3-clause disjunctions (i.e., statements of the form

(x_v11 or x_v21 or x_v31) and (x_v12 or x_v22 or x_v32) and ... and (x_v1n or x_v2n or x_v3n) where each x_vij is a boolean variable or the negation of a variable from a finite predefined list (x_1, x_2, ... x_n). It can be shown that every NP problem can be reduced to 3-SAT. The proof of this is technical and requires use of the technical definition of NP (based on non-deterministic Turing machines and the like). This is known as Cook's theorem.

What makes NP-complete problems important is that if a deterministic polynomial time algorithm can be found to solve one of them, every NP problem is solvable in polynomial time (one problem to rule them all).

NP-hard : Intuitively these are the problems that are even harder than the NP-complete problems. Note that NP-hard problems do not have to be in NP (they do not have to be decision problems). The precise definition here is that a problem X is NP-hard if there is an NP-complete problem Y

such that Y is reducible to X in polynomial time. But since any NP-complete problem can be reduced to any other NP-complete problem in polynomial time, all NP-complete problems can be reduced to any NP-hard problem in polynomial time. Then if there is a solution to one NP-hard problem in polynomial time, there is a solution to all NP problems in polynomial time.

The halting problem is the classic NP-hard problem. This is the problem that given a program P and input I, will it halt? This is a decision problem but it is not in NP. It is clear that any NP-complete problem can be reduced to this one.

P = NP : This the most famous problem in computer science, and one of the most important outstanding questions in the mathematical sciences. In fact, the Clay Institute is offering one million dollars for a solution to the problem (Stephen Cook's writeup on the Clay website is quite good). It's clear that P is a subset of NP. The open question is whether or not NP problems have deterministic polynomial time solutions. It is largely believed that they do not. Here is an outstanding recent article on the latest (and the importance) of the P = NP problem: The Status of the P versus NP problem.

The best book on the subject is Computers and Intractability by Garey and Johnson. My favourite NP-complete problem is the Minesweeper problem.