CSCI361 Notes

Cryptography (University of Wollongong)

CSCI 361- Cryptography and secure applications

Lecture 0: Security concepts

- What, why and who
- Assets & threats
- Security goals
- Security principles

What, why and who

What do we mean by (computer) security?

- Computer security is about protecting computer-based assets against possible threats.

Why does computer security matter?

- Computer security matters because people attack computer systems for various reasons like:
  - For money
  - To obtain knowledge or intellectual property
  - Industrial sabotage
  - For Fun
  - Because they can

Who needs computer security?

- **Governments**
  - To safeguard military or diplomatic communications and **to protect national interests.**
- **Private sector**
  - To protect sensitive information such as health and legal records, financial transactions, credit ratings.
  - To protect information ownership
- **Individuals**
  - To protect sensitive information, and to protect an individual's privacy in the electronic world.
  - **Allow E-commerce, internet banking**, etc.

Assets & threats

- **Assets →** anything we possess of value or use to us. A computer system consists of **hardware, software and data**, each of which is an asset.
- **Threat →** something which potentially violates security. A threat is a **possible danger** that might **exploit a vulnerability**.

Security goals

- **Confidentiality →** Assets should be inaccessible to unauthorized parties
- **Integrity →** Assets should be unmodifiable or unforgeable, without detection, by unauthorised parties.
- **Availability (Authenticity) →** Assets should be available to authorised people.

Security principles

- **Principle of easiest penetration**
  - Intruders will use any available means of penetration
    - Makes security assessment of security a very difficult problem because all possible ways of breaching security must be examined.
- **Principle of adequate protection**
  - Also known as timeliness principle
    - Means items should only be protected while they are valuable and that the level of protection should be consistent with their value.
    - Means protect the asset until the value is lost
- **Principle of effectiveness**
  - Controls must be used properly to be effective
  - Controls should be efficient, easy to use and appropriate
- **Principle of the weakest link**
  - Security is only as strong as the weakest link in the system

Lecture 1: Classical cryptography

- Cryptography vs Cryptoanalysis
- Kirchoff's Law
- Possible attacks
- Caesar cipher
- Monoalphabetic ciphers
  - Additive, multiplicative, affine
  - Key phrase
- Statistical analysis
- Polyalphabetic ciphers
  - Vigenère cipher
- The Kasiski method
- The index of coincidence

Cryptography VS Cryptoanalysis

Cryptography → Designing algorithms to ensure security: i.e. Confidentiality, Integrity, Authenticity

Cryptanalysis → Analysing security algorithms with the aim of breaching security

Key dependence

Encoding → refers to if transformation does not depend on a key

Encryption → transformation is applied to plaintext to produce ciphertext. Transformation are not fixed, they are key dependent

Decoding → refers to inverse transformation

Kirchoff's Law

Main assumption of cryptology:

The cryptanalyst knows all the details of the **encryption and decryption** transformations, except for the **value of the secret key**/keys.

Possible attacks

Ciphertext only → Oscar knows Y. Alice and Bob don't want Oscar to figure out what either X or K is.

Known plaintext → Oscar knows some X-Y pairs. Alice and Bob don't want Oscar to figure out what **K** is, or the **correspondence between other X-Y pairs.**

Chosen plaintext → Oscar is allowed to choose some plaintext(X's), and receives the corresponding ciphertexts (Y's).

Chosen ciphertext → Oscar is allowed to choose some ciphertexts (Y's) and receives corresponding plaintexts (X's).

Caesar cipher

- Every letter is replaced by the letter “three to the right” in the alphabet, where this operation is cyclic. E.g. A→ D, B → E, X → A…
- For example: CABBAGE → FDEEDJH
- No key involved → not a true cipher

Monoalphabetic ciphers

- Also known as **simple substitution ciphers**
- Each letter of the plaintext alphabet is replaced with an element of the ciphertext alphabet.
- Substitution alphabet → key

a	b	c	d	e	…	x	y	z
F	G	N	T	A	…	K	P	L

a		b	a	d		d	a	y
F		G	F	T		T	F	P

- **Security**
  - Use exhaustive key search to **find the full key**
  - Means use each key to decipher the ciphertext and **accept the one that produce a meaningful plaintext**
  - For an alphabet of size N, the number of possible keys is N!

$$N! \approx \sqrt{2\pi N}\left(\frac{N}{e}\right)^{N}$$

  - E.g. English alphabet → 26! ≈ 4 x 10^{26} keys

- **Examples of Monoalphabetic Cipher**: **Additive, Multiplicative, Affine cipher**
- Additive & Multiplicative ciphers are examples of indexed constructions
- **Additive ciphers**
  - Also known as **translation ciphers**
  - The substitution alphabet is obtained by shifting the plaintext alphabet by a fixed value.
    - Amount of shift is the key

        - E.g. A + 3 shifts = D , B + 3 shifts = E
        - Similar to Caesar cipher
    - Plaintext character X, ciphertext character Y, shift (key) Z → an element of the set {0, 1, 2, 3, …., 25}
    - Y = X ⊕ Z, ⊕ denotes addition modulo 26
    - This is a key space, hence exhaustive key search is a feasible attack against additive ciphers.

- **Multiplicative ciphers**
  - Ciphertext alphabet can be determined using modular multiplication.
  - Multiply each plaintext alphabet by a constant value (key for this cipher)
    - Y = X ⊗ Z, ⊗ denotes multiplication modulo 26.
  - **Will not result in one-to-one mapping**
  - E.g. Z = 2,
    - 1 ⊗ 2 = 2 (b → c) [0 – a, 1 – b, 2 – c]
    - 14 ⊗ 2 = 2 (o → c) [14 x 2 = 28 mod 26 = 2]
  - For all even numbers, and 13, the mapping is not one-to-one. Hence the number of keys is 12.
  - Exhaustive key search can easily break the cipher (find the secret key).

- **Affine ciphers**
  - Increase the number of keys, we can **combine additive and multiplicative** ciphers → obtain affine ciphers
  - Y = $\alpha \otimes X \oplus Z$, where X, Y, Z $\in \{0,1,\ldots,25\}$
  - α $\in$ {1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25} → exclude all even numbers & 13
  - Number of keys = 12 * 26 = 312 → still too small!

- **Key phrase based ciphers**
  - **Using of a phrase** as the key (index). Also specify a starting letter for the translation alphabet
    - **Increases the size of the key space** but makes the key(index) easy to remember.
      - **More resistant to exhaustive key** search but is insecure against statistical cryptanalysis.

Statistical analysis

- Statistical properties of the plaintext language can be used to cancel many keys in one step and enable the cryptanalyst to find the key without trying all of them.

- Statistical analysis relies on there being a relationship between the statistical properties of the plaintext and the statistical properties of the ciphertext since we assume the attacker only has the ciphertext.

Polyalphabetic ciphers

- A ciphertext character represents more than one plaintext character
- Must be done in a way that allows the plaintext to be recovered.
    - E.g. if B represents n and t, we need to know when to decipher it as n and when as t.
- Uses a sequence of substitution alphabets.
    - If this sequence repeats after p characters, it has period p.

- **Vigenère cipher**
    - Uses 26 substitution alphabets and a key word / key phrase
        - Substitution alphabets are cyclically related
    - The frequency distribution of the ciphertext can be flattened by using multiple substitution alphabets.

    - We analyse these ciphers by:
        - 1) Finding the period
        - 2) Breaking the ciphertext into components each obtained from a single substitution alphabet.
        - 3) Solving each component using techniques discussed for monoalphabetic ciphers.

1. Finding the period:

a. The Kasiski method

i. Two identical plaintexts will be encrypted to the same ciphertext if their occurrence is m positions apart, where m = 0 mod p, i.e. when m is a multiple of the period p.

X	w	o	l	l	...	...	w	o	l	l
Z	c	a	f	e	...	...	c	a	f	e
y	Y	O	Q	P	...	...	Y	O	Q	P

ii. Search the ciphertext for repeated segments and measure the distance between such repeated segments.

- Likely that these distances will be a multiple of the period.

b. The index of coincidence

i. Consider a length of n, where each element is a letter from the English alphabet. X = x1, x2, …., Xn

$\lambda \in \{A,B,C,\ldots,Z\}$, and f_λ *frequency of* λ

$$IC(x) = \frac{\sum_{\lambda=A}^{Z} f_\lambda (f_\lambda - 1)}{n(n-1)}$$

ii. IC(x) is an estimate of the probability that **2 randomly chosen elements** of X are identical

iii. A measure of roughness of the histogram – indicates how uneven the histogram is.

iv. Can be used to estimate the period (Friedman or Kappa test)

$$\kappa = \frac{0.027n}{IC(n-1) - 0.038n + 0.065}$$

v. Random IC = 1/ 26 = 0.038 (approx.)

vi. English IC =

$$IC(x) \approx \sum_{\lambda=A}^{Z} p(\lambda)^2 \approx 0.065$$

vii. If Index of Coincidence is between 0.038 - 0.065, it is most like enciphered using polyalphabetic cipher

viii. If the ciphertext message has an index that is close to 0.065, it is most likely a monoalphabetic substitution

Lecture 2: Secret key cryptography

- One-time pad
  - Perfect secrecy
- Unicity distance
- Confusion and Diffusion
- DES design

- Iterated cipher
- Block and key size
- The Feistel structure
- Involution functions
- Encryption and Decryption
- S-box

* Triple DES

One-time-pad (Vernam cipher)

* Invented by Gilbert Vernam (1917/8) – then extended by Joseph Mauborgne
* A scheme Shannon 1949 proved to provide perfect secrecy
* Key is written on a **long tape** and is **used only once**.
* Encryption is by **addition**
* To decrypt, the same key sequence must be used.
  - Decryption is by **subtracting the key sequence from the ciphertext**.
  - Key sequence → a random sequence.

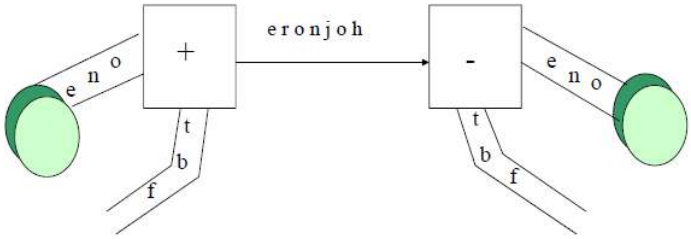
* Encryption of each element of plaintext is independent of the encryption on any other piece of plaintext.
* Encryption & Decryption:
  - Encryption:

    $Y_i = (X_i + K_i) \bmod 26$ Letter by letter.

    $Y_i = X_i \oplus K_i$ Bit by bit.
  - Decryption:

    $X_i = (Y_i + K_i) \bmod 26$ Letter by letter.

    $X_i = Y_i \oplus K_i$ Bit by Bit




```
Message    :   ONETIMEPAD
Key on pad:    TBFRGFARFM
Ciphertext:    HOJNOREGFP

  O + T  mod 26 = H
  N + B  mod 26 = O
  E + F  mod 26 = J ...
```

Question 4 (2 marks)

One possible implementation of "One-time pad" is to generate a random key sequence of the same length as message and encrypt the message using the cipher $C_i = M_i + K_i \ (mod\ 26)$, where K_i are random key characters and $M = K = C = \mathbb{Z}_{26}$. One of the recommendations for a proper use of "One-time pad" and to ensure perfect secrecy is to never reuse the same key for encryption of two different messages. Your task is to explain how only the knowledge of two different ciphertext sequences $C = C_1C_2 \dots C_n$ and $C' = C_1'C_2' \dots C_n'$, obtained by applying the same secret key, can compromise the security of One-time pad.

Solution:

Suppose the two messages are encrypted using the same key as follow:

$$C_i = M_i + K_i(mod\ 26) \quad \dots\dots eq1,$$
$$C_i' = M_i' + K_i(mod\ 26) \quad \dots\dots eq2$$

By solving the two equations, we have $M_i - M_i' = C_i - C_i'(mod\ 26)$ for $i = 1,2, \dots, n$. From here, we can see that if we know C_i and C_i' we can find $M_i\ and\ M_i'$. Alternatively, we can also apply known plaintext attack.

✶ Perfect secrecy (Shannon 1949)

- Knowledge of the cryptogram does not help the enemy
- P(X=x) = P(X=x|Y=y), ∀ x,y → Probability to find the plaintext, given the ciphertext.
- As likely to guess the plaintext associated with a ciphertext after they see the ciphertext as they are before they see it.
- In a system with perfect secrecy, the number of keys is at least equal to the number of messages.
  - Shannon proposed unicity distance as the measure of security.
- **Unicity distance**
  - N_0 is the least number of ciphertext characters needed to determine the key uniquely
  - If there are E keys, and they are chosen with uniform probability, unicity distance is given by:
    - $N_0 = \frac{\log_2 E}{d}$

        - **where d is the redundancy of the plaintext language**
        - **if N_0 is 25, we give the enemy 24 (number must always be less than N_0**
    - Redundancy of a language = rates of the language.
        - **d= R- r bits**
    - The absolute rate R of a language is the minimum number of bits to represent each character
    - The true rate r of a language is the average number of bits required to represent characters of that language.
        - **E.g. for English → R ≈ 4.7 bits, r ≈ 1 – 1.5 bits, d ≈ 3.2 bits**
    - True rate < absolute rate, the difference is the **redundancy**.
    - For **one-time-pad**:
        - Keys are **chosen randomly with uniform distribution:**

$E=26^k$, $\log_2 E \approx 4.7k$ so that

$N_0 \approx (4.7)/(3.7)k \approx 1.27k$

$\log_2 E$ = (ln E) / (ln2)

E = 26^k

$\log_2 26^k$ = (ln 26^k) / (ln2) = 4.7k (approx.)

N_0 = (4.7) / (3.7) k = 1.27k

    - Therefore, a unique solution cannot be obtained even if all the characters of the ciphertext are intercepted. → **Perfect Secrecy**

* **Measuring security**
    - Can use unicity distance as a measure to compare the security of various ciphers
    - Recall: security → protecting assets against possible threats
    - Recall: security principles → Principle of Adequate Protection or the concept of timeliness

* Unconditional VS Practical security & Practical or computational security
    - A system is unconditionally secure if it is secure against an enemy with unlimited resources (time, memory)
    - In an unconditionally secure system, the protection is provided in an information theoretic sense. → Reason that the enemy cannot break the cipher is that they do not have enough information.

        - The one-time-pad is the simplest only encryption system with unconditional security, but it requires many key bits and security is under ciphertext only attack
    - Computational or practical security → attempts to provide a more realistic model for a secure system than unconditional security.
    - In practical security, the main objective is to make sure the amount of work required to find that plaintext stays high even after the unicity distance
    - A cipher is computationally secure if the **difficulty of the best attack** exceeds the **computationally capability of the attacker**.

Question: What is the difference between an unconditionally secure cipher and a computationally secure cipher?

An encryption scheme is unconditionally secure if the ciphertext generated by the scheme DOES NOT contain enough information to determine uniquely the corresponding plaintext, no matter how much ciphertext is available.

An encryption scheme is said to be computationally secure if the following two conditions are noticeable in the scheme:

1. The cost of breaking the cipher exceeds the value of the encrypted information and
2. The time required to break the cipher exceeds the useful lifetime of the information.

Confusion and Diffusion

Diffusion → Diffuse the statistical structure of the plaintext into long range statistics. Statistics involved long combinations of letters.

E.g. classical transposition cipher

***Diffusion hides the relation between ciphertext and the plaintext.**

Confusion → The relationship between plaintext, ciphertext and key must be complex so that knowledge of plaintext/ciphertext pairs cannot easily be used to find the key.

In other words, Y = E(X, Z) → should be a nonlinear and complex function

E.g. Classical substitution cipher

***Confusion hides the relation between the ciphertext and key**

Question 1:

Does a one-time pad provide confusion and diffusion?

Answer:

From the definition, confusion is about having a complex relationship between **plaintext, ciphertext and key**. Diffusion is about **changing the statistical property** of the input (plaintext) such that a single bit change in the plaintext causes an unpredictable changes in the ciphertext. One-time pad provides a perfect confusion, but not diffusion.

Look at the definitions of confusion and diffusion.

Diffusion is to do with each input bit having an effect on each output bit (within a block say). Confusion is about having a complex relationship, between plaintext, ciphertext and key. A one-time pad doesn't diffuse at all, but it provides perfect confusion.

Question 2:

What is the difference between diffusion and confusion? How to achieve them?

Explain and justify your answer with examples.

Answer:

In diffusion, the statistical structure of the plaintext is dissipated into long-range statistics of the ciphertext. This is achieved by **having each plaintext digit** affect the **value of many ciphertext digits**, which is equivalent to saying that many plaintext digits affect each ciphertext digit.

Confusion seeks to make the relationship between the **statistics of the ciphertext** and the **value of the encryption key** as complex as possible, again to thwart attempts to discover the key. Thus, even if an attacker can get hold of some of the statistics of the ciphertext, the way in which the key was used to produce that ciphertext is so complex as to make it difficult to deduce the key. This is achieved by using a complex substitution algorithm.

Avalanche Effect

Question:

In cryptography context, explain the avalanche effect.

Answer:

The avalanche effect is a property of any encryption algorithm such that a **small change in either the plaintext or the key** produces a **significant change in the ciphertext**.

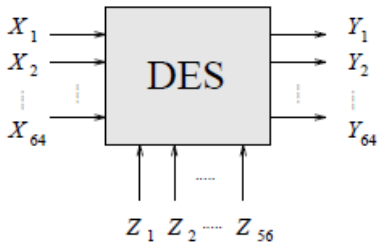
Secure block cipher:

- Needs a **block length large enough to deter statistical attacks**.
  - Length 64 is usually considered **large enough to make frequency analysis infeasible**
  - Larger block length = longer processing time. The longer you have to wait before you begin to process.
- Needs a **key space large enough** to make exhaustive key searches infeasible.
  - Keys should be short enough so that key generation, distribution and storage can be efficiently managed.

DES design

- **Iterated cipher**
  - Block cipher is called an iterated cipher as it is based on a simple function f, repeated (or iterated) several times.
  - Each iteration is called a round.
  - Output of each round is a function of the output of the previous round and a sub-key derived from the full secret key by a key scheduling algorithm.
  - Decryption is performed by using the inverse of the round functions and the sub-keys in reverse order.
  - This class of functions are the involution functions. These functions are self-invertible. Employing such functions allow encryption and decryption modules to be the same but keyed in the reverse order.

- **Block and key size**
  - DES → block cipher
  - Takes an input block of 64 bits and maps it into an output block of 64 bits using a key of 56 bits.
  - |X| = |Y| = 2^{64} , |Z| = 2^{56}



  - 56 bits? DES expects 64 bits of key, but only uses 56. Remainder are generally used for parity checking.

- **The Feistel structure**
  - Multiple rounds with ordered sub-keys
  - The input is split into 2 parts
  - The right-hand side is transformed by a round function f in each round, before being combined with the left-hand side using XOR operation.
  - The left- and right-hand sides are switched after each round to obtain the input to the next round.

- Feistel (1973) proposed this structure as a means of combining substitution and permutation elements to provide confusion and diffusion.

✶ Involution functions

- Each round of DES is a product cipher whose 1st component cipher is a substitution cipher F(.) & whose 2nd component cipher is the transposition T(.).
  - The substitution cipher is an involution

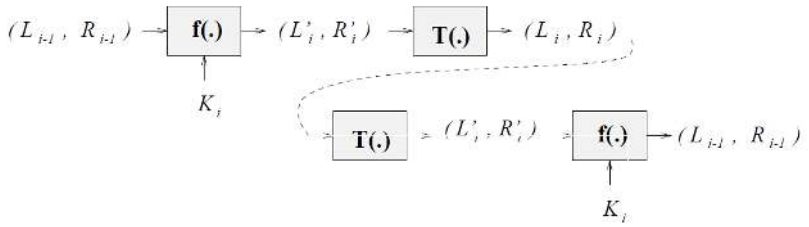
$$F_i(L_{i-1},R_{i-1}) = (L_{i-1} \oplus f(k_i,R_{i-1}),R_{i-1})$$
$$F_iF_i(L_{i-1},R_{i-1}) = F_i(L_{i-1} \oplus f(k_i,R_{i-1}),R_{i-1})$$
$$= (L_{i-1} \oplus f(k_i,R_{i-1}) \oplus f(k_i,R_{i-1}),R_{i-1})$$
$$= (L_{i-1},R_{i-1})$$

  - The transposition cipher T(.) is an involution

$$T(L_i^{'},R_i^{'}) = (R_i^{'},L_i^{'})$$
$$T(T(L_i^{'},R_i^{'})) = T(R_i^{'},L_i^{'}) = (L_i^{'},R_i^{'})$$

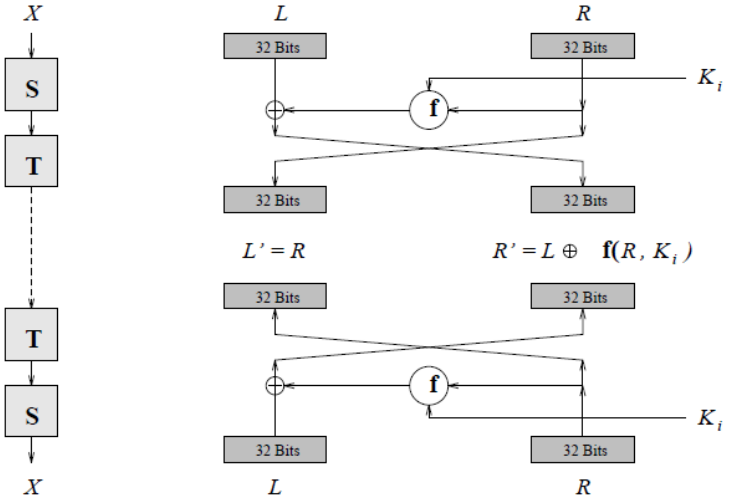
✶ Encryption and Decryption

To decrypt one round.



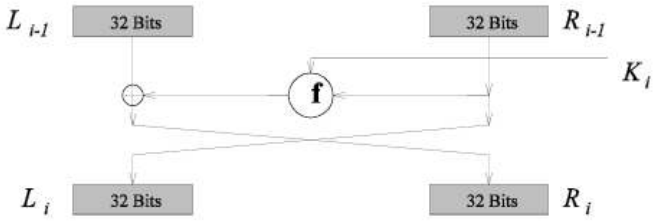
  - The full 16 round DES algorithm is a product cipher composed of 16 rounds, each round a substitution and a transposition (except the last).
  - DES = $(IP)^{-1}F_{16}TF_{15}T\ldots F_2TF_1(IP)$ → Encryption
  - $DES^{-1}=(IP)^{-1}F_1TF_2T\ldots F_{15}TF_{16}(IP)$ → Decryption

Decryption algorithm

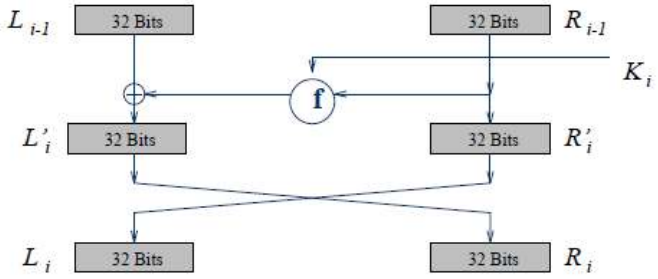


The decryptor for DES is identical to the encryptor, except that the 16 keys are used in reverse order.

The round structure of DES

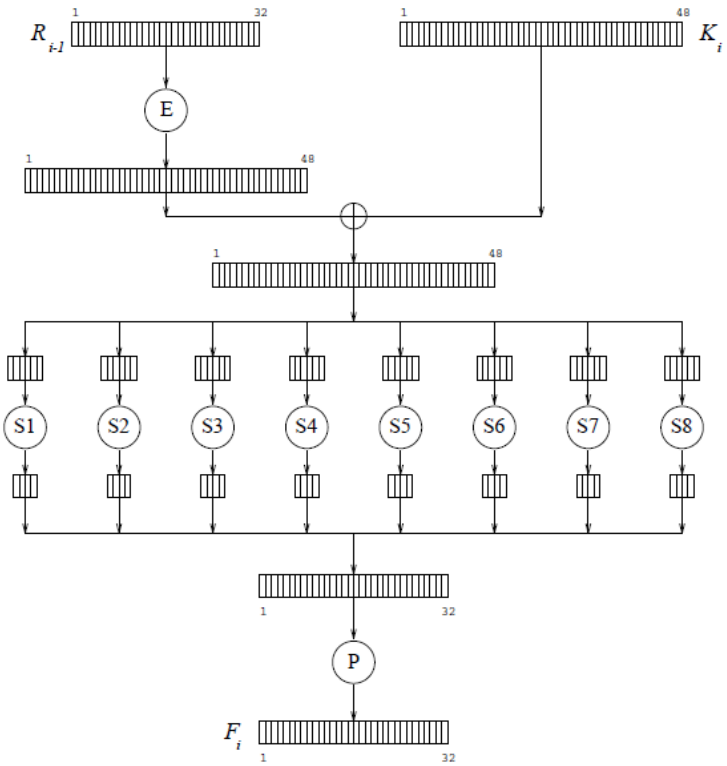


Why? This can be redrawn as …



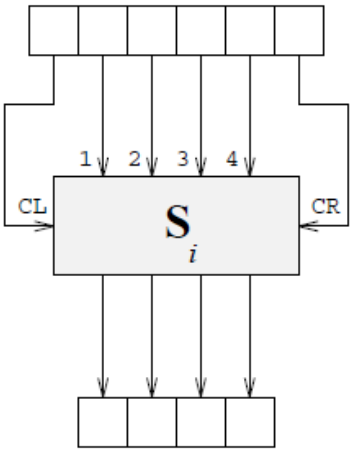
… the Feistel structure!

$f(k_i, R_{i-1})$



- **P is the permutation, E is the expansion**

- **S-boxes**
  - **Structure of S-Boxes**



  - **CL = left control bit, CR = right control bit**
  - The CL and CR select one row of the S-Box S_i to use
  - For each of the 4 choices of (CL, CR), S_i performs a different substitution on the 16 possible values of the 4 inner input bits.
  - E.g. S_i (1,0,1,1,1,0) = [1,0,1,1]

  - **Properties of S-Boxes**
    - 3 designs properties of S-boxes (by Designer Coppersmith – 1994)
    - These properties ensure good confusion and diffusion for the algorithm

    1) **For every choice of control bits, every output bit is a nonlinear function of input bits.**
    a. Each row of an S-box consists of 4 non-liner Boolean functions. This non-linearity is the basis of the required confusion for DES.

    2) **Changing a single input bit to an S-box changes at least 2 output bits of that S-box**
    a. This means that statistical properties of the input bits are spread over the output bits→ diffusion

    3) **When a single input bit is held constant, there is a good balance of 0's and 1's in the 2^5 = 32 output half bytes as the remaining 5 input bits are varied**
    a. Ensure a uniform distribution of the algorithm output and stops cryptanalysts from making statistical inferences.

    - **1, 2 and 3 imply good confusion and diffusion**

✶ DES weakness

1. Complement property:

Let ū denote complement of u. That is,
ū=u+1 (mod 2).

$$DES_Z(X) = DES_{\bar{Z}}(\bar{X})$$

So, if we know the ciphertext Y for a plaintext X produced with a key Z, then we also know the cryptogram that key complement Z produces for complement X.

- This effectively halves the number of keys to be tested in an exhaustive key search.

2. Not every key is a good key:

- Let Z denote the original 64 bit key.
- Z is a weak or self-dual key if the key scheduling algorithm applied to the key Z produces identical sub-keys
- $Z_1 = Z_2 = Z_3 = .. Z_{16}$
- Hence, encryption and decryption are the same operation
  - $DES_Z = DES_Z^{-1}$
- 4 weak keys (with each 8-bit word having odd-parity
  - [00000001] , [11111110], [00011111], [11100000 x4 , 11110001 x4]
  - These keys make C_0 and D_0 , all zero or all one

    (0101010101010101), (FEFEFEFEFEFEFEFE), (IFIFIFIF0E0E0E0E), and (E0E0E0E0FIFIFIFI)

    16bits for each 4 output – 16 x4 = 64bits
- Z is a semi-weak key, if there is another key Z' such that:

  $DES_Z^{-1} = DES_{Z'} \vee DES_{Z'}^{-1} = DES_Z$
  - There are 12 semi-weak keys (with odd parity for each 8-bit word) – 6 pairs

E001E001F101F101	01E001E001F101F1
FE1FFE1FFEOEFEOE	1FFE1FFEOEFEOEFE
E01FE01FF10EF10E	1FE01FEOOEF10EF1
01FE01FE01FE01FE	FE01FE01FE01FE01
011F011F010E010E	1F011F010E010E01
E0FEE0FEF1FEF1FE	FEE0FEE0FEF1FEF1

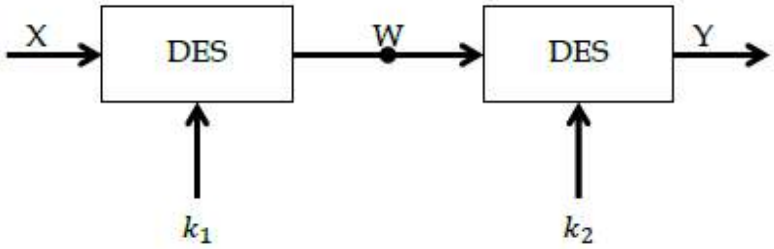
- Weak and semi-weak keys can have disastrous effects if multiple encryption strategies are used.

3. **Exhaustive key search:** (brute force attack)

- DES has $2^{56} \approx 10^{17}\ keys$
- Given a plaintext block and the corresponding ciphertext block, we can try each key to decrypt the ciphertext and compare the result with the known plaintext.
  - ➔ If we try one key every 10^{-6} seconds, → takes $10^{11}\ seconds \vee 3170\ years$
- But we can test key in parallel
  - ➔ Building a device with 10^{7} DES chips would allow exhaustive cryptanalysis with only 10^{10} decryptions per chip.

- **Multiple encryption**
  - **What is meet-in-the-middle attack?**
    - Meet-in-the-middle attack is an attack used by an attacker to attack a multiple encryption algorithm. This attack requires a known plaintext-ciphertext pair. In essence, the plaintext is encrypted to produce an intermediate value in the multiple encryption, and the ciphertext is decrypted to produce an intermediation value in the multiple encryption. Table lookup (dictionary attack) techniques can be used in such a way to dramatically improve on a brute-force try of all pairs of keys.

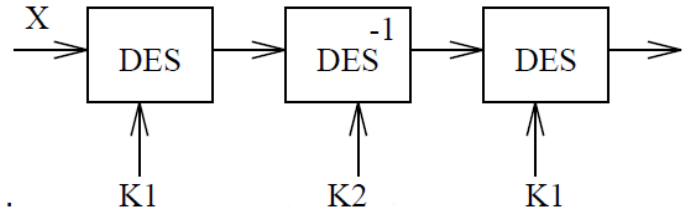
  - **One issue with DES is the key size of 56-bit too short. To increase the key size, one approach is to double encrypt, and hence effectively increase the key size to 112 bits. However, this approach is not really the same as if there were a single DES of 112-bit. Explain why is it much less secure to implement a double DES.**
    - With double DES, the plaintext Y is encrypted 2 times, with different keys; Y → $E_{k2}(E_{k1}(X))$



Since the plaintext X is encrypted two times with k_1 and k_2, the key-space size of double DES is basically $2^{56} \times 2^{56} = 2^{112}$. However, double DES can be attacked by "meet-in-the-middle", which takes not much more time than 2^{57}. With the "meet-in-the-middle" attack, the attacker knows a pair of plaintext and ciphertext (X, Y). The attacker tries to **decrypt** all 2^{56} values of permutations (identified as z'') and produces a list (w_i'', z_i'') where $w_i'' = DES_{z_i'}^{-1}(Y)$, and z_i'' is all 2^{56} values of permutation. The attacker then tries all 2^{56} values of permutation (identified as z') to **encrypt** X and produces a list (w_i', z_i') where $w_i' = DES_{z_i'}(X)$. The attacker then sorts the lists (w_i', z_i') in w_i' assending order, and (w_i'', z_i'') *in* w_i'' ascending order. The attacker is able to compare these two lists and determine the keys. If a match $(w_i' = w_i'')$ is found, then z_i'' is $k_2$and z_i' is k_1. With this attack, the number of operations is equal to the sum of the number of decryption $\left(w_i'' = DES_{z_i'}^{-1}(Y)\right)$ and the number of encryption $\left(w_i = DES_{z_i'}^{-1}(Y)\right)$. Hence the total number of operation required equals $2^{56} + 2^{56} = 2^{57}$. In order words, the effective key space is only 2^{57} bits and not 2^{112} bits.

Triple DES

- Proposed by Tuchman (1978)
- Various “modes”



- In this case, meet-in-the-middle doesn’t work
- Triple DES provides a relatively simple method of increasing the key size of DES to protect against such attacks, without the need to design a completely new block cipher algorithm.
- Triple DES uses “key bundle” that comprises 3 DES keys, K1, K2, K3, each of 56 bits (excluding parity bits)
- Encryption algorithm: DES encrypt with K1, DES decrypt with K2, then DES encrypt with K3

  $$\text{ciphertext} = E_{K3}(D_{K2}(E_{K1}(\text{plaintext})))$$

- Decryption algorithm: decrypt with K3, encrypt with K2 then decrypt with K1

  $$\text{plaintext} = D_{K1}(E_{K2}(D_{K3}(\text{ciphertext})))$$

  Each triple encryption encrypts one block of 64 bits of data
  In each case the middle operation is the reverse of the first and last. This improves the strength of the algorithm when using keying option 2, and provide backward compatibility with DES with keying option 3.

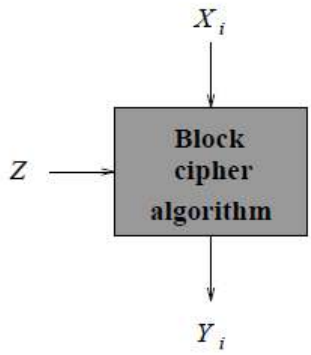
Lecture 3: Mode of operation (Modes for block ciphers)

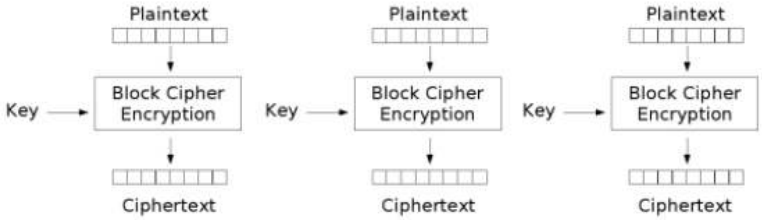
- Electronic codebook
- Cipher block chaining
- Cipher feedback
- Output feedback
- Counter
- Advantage / Disadvantage

A block cipher can be used in different modes. There are 4 standard modes which were recommended for DES.

1. Electronic codebook mode (ECB)
2. Cipher block chaining mode (CBC)
3. Cipher feedback mode (CFB)
4. Output feedback mode (OFB)

- **Electronic codebook mode (ECB)**
  - Basic mode of operation
  - A plaintext message is broken into blocks and each block is encrypted separately.





Electronic Codebook (ECB) mode encryption

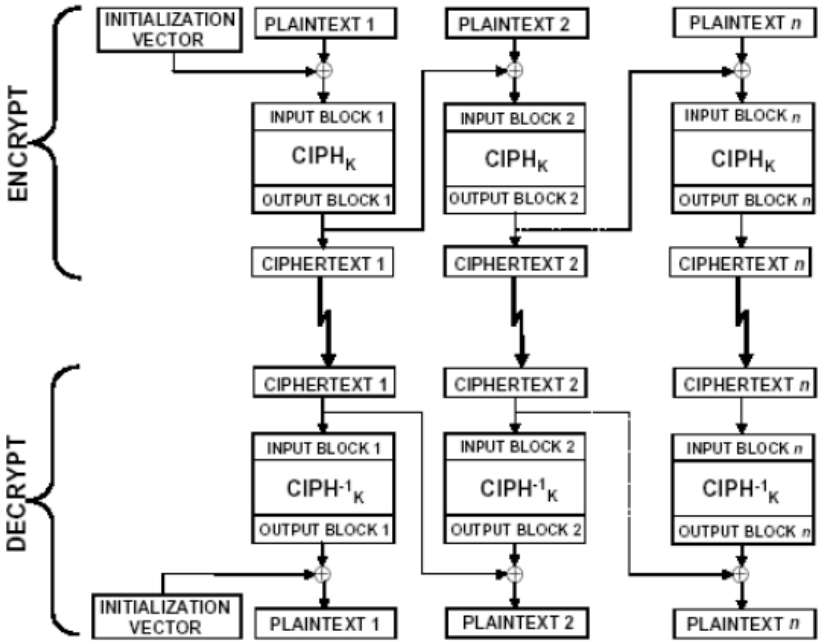
  - **Disadvantage**
    - If we encrypt 2 identical plaintext blocks, the resulting ciphertext blocks will be the same
    - This leaks some info to an eavesdropper
    - Formatting information, such as columns / paragraphs and other plaintext details → leak through the cipher system

- **Cipher block chaining mode (CBC)**

- A plaintext block is XORed with the ciphertext block of the previous round and then entered to the encryption algorithm
- Strong dependency between consecutive blocks created.

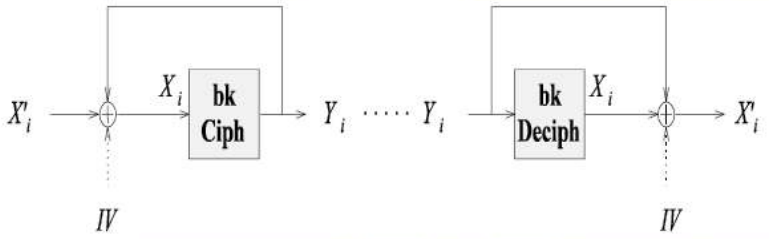
$$X_1 = X_1^{'} \oplus IV$$
$$X_2 = X_2^{'} \oplus Y_1$$
$$\dots$$
$$X_i = X_i^{'} \oplus Y_{i-1}$$

- Operations:
  - For the first block, since ciphertext from the previous block does not exist, a random initial vector, called the IV is used.
  - IV is not secret but it needs to be unpredictable.
  - CBC mode produces different ciphertext blocks even if plaintext blocks are the same.
  - Strong dependency between blocks suggest that CBC can also be used to provide integrity.



Input	000	001	010	011	100	101	110	111
Output	111	110	011	100	001	000	101	010

$XOR_0 = Plaintext_0 \oplus IV$ $XOR_i = Plaintext_i \oplus Ciphertext_{i-1}$



CBC IV=111

Plaintext	101	101	110	010
XOR	010	110	011	110
Ciphertext	011	101	100	101

Question:

We consider a Cipher-Block Chaining Mode (CBC mode) for a block cipher which implements the encryption as □□=□(□,⊕ −1) for i > 0 where □1,□2,□3 . . . are the messages and □0 is a randomly chosen initial vector.

(i) Explain how decryption is done and give the mathematical expression for the decryption.

(ii) How does a bit error in the ciphertext influence decryption? (Assume that *Ci* is obtained corrupted because of a bit error. How does it affect the next decryption steps?)

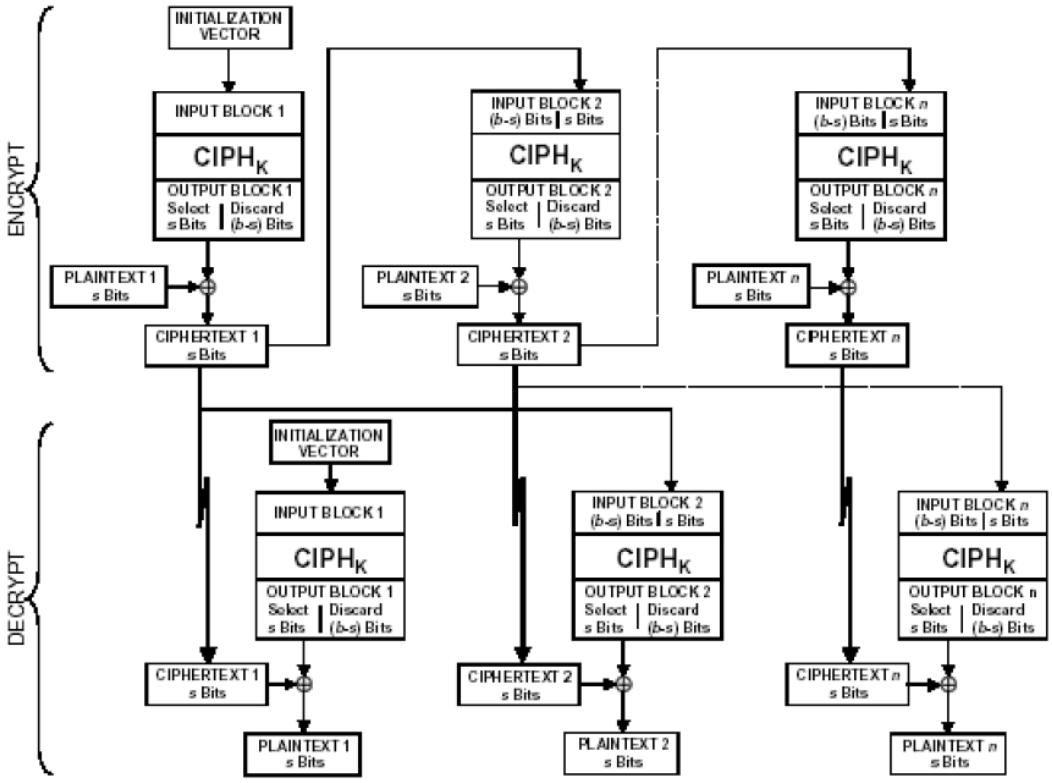
Answer:

i. To decrypt, each cipher block is passed through the decryption algorithm. The result of the decryption algorithm is then XORed with the preceding ciphertext block to produce the plaintext block. The plaintext can be recovered from just 2 adjacent blocks of the ciphertext, thus, decryption can be parallelized.
Decryption expression of the CBC mode described above is as follow:

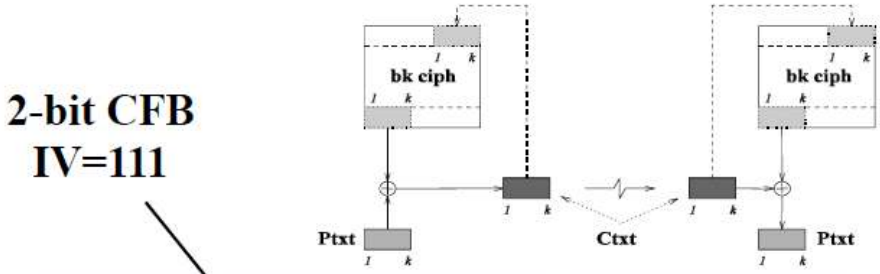
□□=□ (□,□ □) ⊕ □□-1 *h* □0=□□ (□□□□□□□□□□□□□□□□).

ii. Since the plaintext can be recovered from just 2 adjacent blocks of the ciphertext, if a bit error occurs in the ciphertext, this error will affect only 2 blocks of the plaintext, the plaintext of the current block as well as the next block. This is because the ciphertext of the current block is used to XORed with the preceding ciphertext block to produce the plaintext block. In other world, the error is propagated only to the next block, not further.

- **(k-bit) Cipher feedback mode (CFB)**
  - The block cipher algorithm is effectively turned into a pseudorandom generator that produces a k-bit pseudorandom number in every execution of the algorithm
  - For decryption, a similar generator is used to remove the masking pseudo-noise data.
  - The input to the cipher "at the top" is obtained by concatenating the input to the previous round, left shifted by k bits, with the k-bit feedback.
  - Using smaller k in general produces a more secure pseudorandom stream, with the cost of lowering speed.



Input	000	001	010	011	100	101	110	111
Output	111	110	011	100	001	000	101	010



Plaintext	10	11	01	11	00	10
Input	111	111	110	011	101	100
E(Input)	010	010	101	100	000	001
Ciphertext	11	10	11	01	00	10

Question:

(i) Encryption of large blocks using TEA (or any fixed size block cipher), as you have done for one of the tasks in your assignment, can be achieved through the means of modes. For the s-bit CFB (cipher feedback) mode, the encryption is depicted in the following diagram. **Draw the decryption block diagram** for the s-bit CFB mode shown below, and **give the mathematical expression** for the decryption.

(ii) What are the advantages and disadvantages of the CFB mode of operation?

Answer:

(i) To decrypt, the same scheme is used, except that the received ciphertext unit is XORed with the output of the encryption function to produce the plaintext unit. The mathematical formula to decrypt is:

$C_i = P_i \oplus E[K, (K, C_{i-1})]$, where $C_0 = IV$ (Initial Vector).

(ii) Advantages:

a. Same encryption function is used to encrypt and decrypt
b. The message (plaintext) does not need to be padded to a multiple of the cipher block size.
c. Decryption can be done with just 2 adjacent cipher blocks and hence decryption can be parallelized.

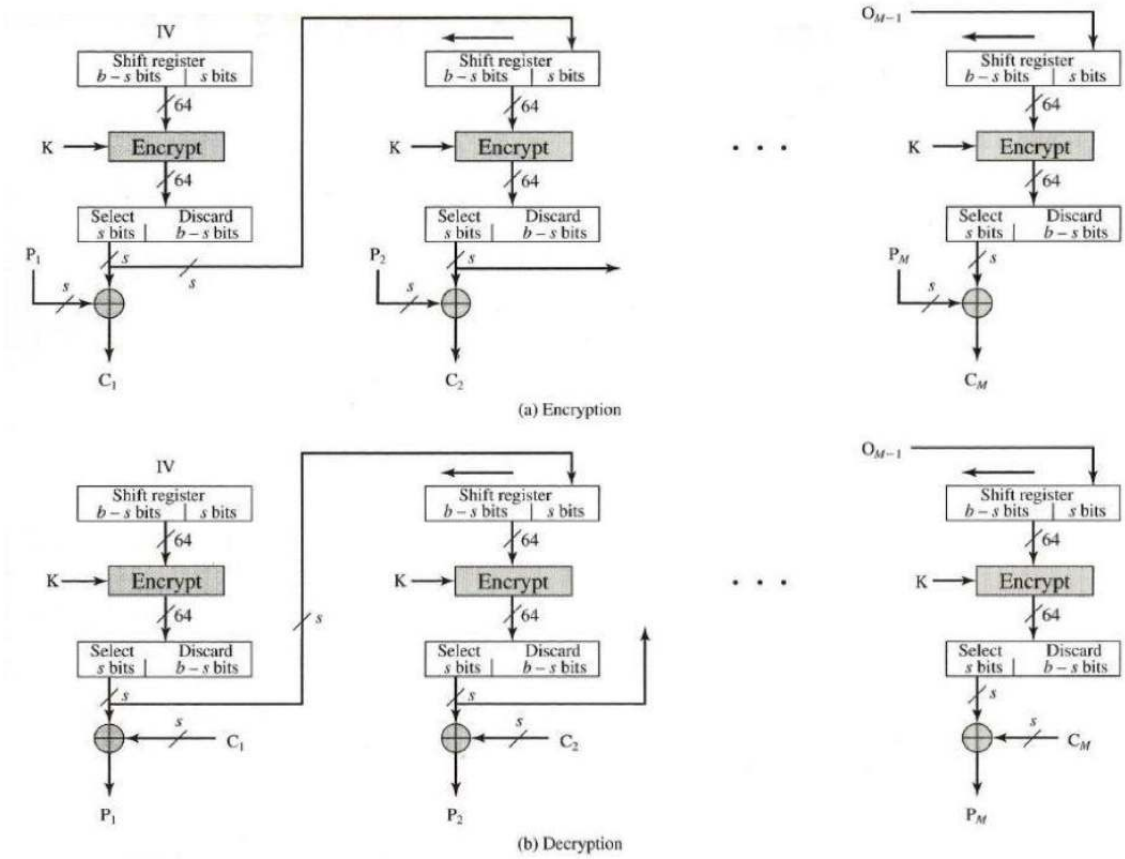
Disadvantages:

a. Due to the chaining of ciphertext, the encryption cannot be parallelized, however, the decryption is possible.

(k-bit) Output feedback mode (OFB)

- Very similar to CFB, but the feedback is independent of the transmitted data
- This means the pseudorandom stream can be pre-generated prior to any transmission.

Question:



/

ii) The encryption and decryption formula are as follow:

$C_i = P_i \oplus$ □□[□(□, O_i −1)]

$P_i = C_i \oplus$ □□[□(□, O_i −1)]

C_0 =□□

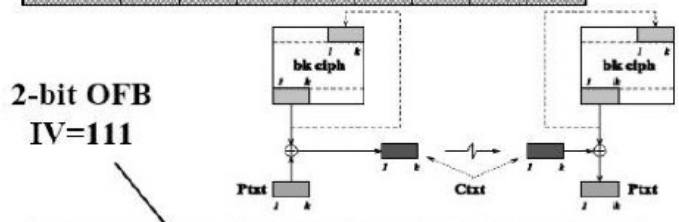
(ii) The cipher of a 3-bit block cipher is given as follow:

Input	000	001	010	011	100	101	110	111
Output	111	110	011	100	001	000	101	010

Given the Initial Vector (IV) = 111, and a 2-bit shift OFB operation mode, encrypt the following plaintext.

Plaintext	10	11	01	11	00	10
Input						
E (Input)						
Ciphertext						

Input	000	001	010	011	100	101	110	111
Output	111	110	011	100	001	000	101	010



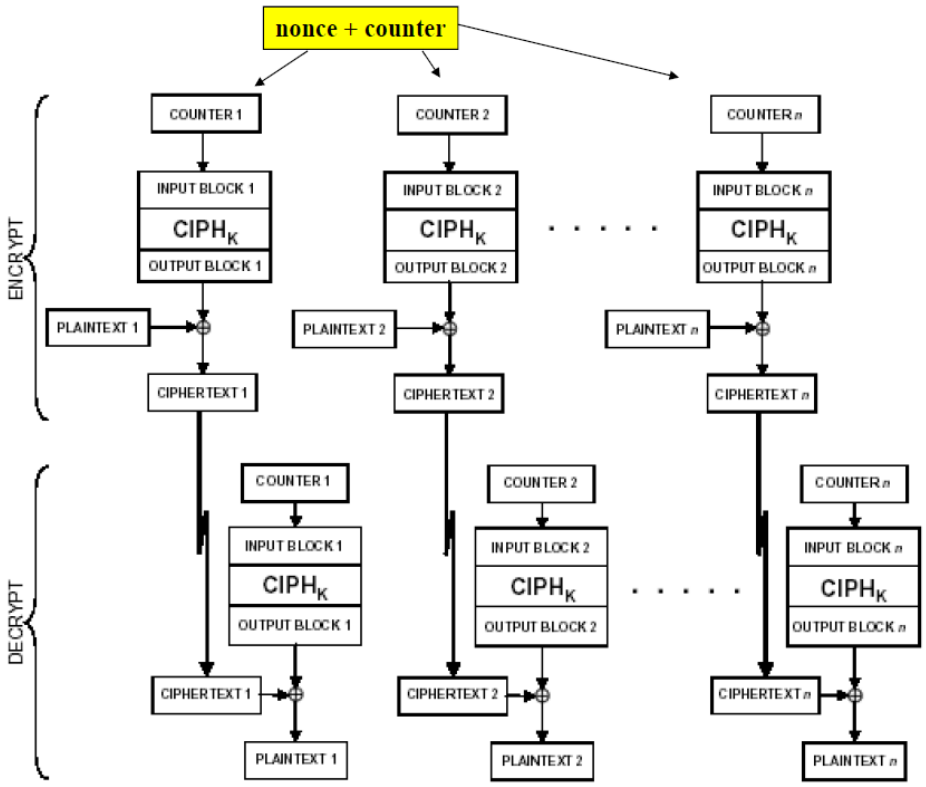
Plaintext	10	11	01	11	00	10
Input	111	**101**	**100**	**000**	**011**	**110**
E(Input)	**010**	**000**	**001**	**111**	**100**	**101**
Ciphertext	11	11	01	00	10	00

Advantages of CBC, CFB, and OFB:

1. All hide patterns in plaintext since $X_{i+n}=X_i$ does not imply that $Y_{i+n}=Y_i$
   a. 2 identical plaintext blocks at 2 different locations in a ciphertext will produce 2 distinct ciphertexts.
      i. This will stop the leakage of information which occurs in ECB mode.
2. Prevent data tampering (CFB)
   a. The dependency between ciphertext blocks can be used to detect tampering of the ciphertext. That is, a single bit change in a ciphertext block will affect decryption of all the following ciphertext blocks.
3. The k-bit ciphers (CFB and OFB) can be tuned, with respect to k, to allow for the user's data format, required level of security and resources.
4. The OFB cipher can be performed in parallel, with pre-computation of the pseudorandom stream.

Counter Mode (CTR)

- Cipher is applied to a set of input blocks called counters
- The sequence of output blocks are XORed with the plaintext to produce the ciphertext
- The sequence of counters must satisfy this property:
  - Each block in the sequence is different from every other block, across all of the messages that are encrypted under the given key.



  - In both CTR encryption and decryption the ciphers can be performed in parallel.
    - The plaintext block that corresponds to any particular ciphertext block can be recovered independently from the other plaintext blocks.
    - The ciphers can be applied to the counters prior to the availability of the plaintext or ciphertext data.

Input	000	001	010	011	100	101	110	111
Output	111	110	011	100	001	000	101	010

CTR
IV=000

$$\text{Counter}_i = \text{Counter}_{i-1} + 1$$

Plaintext	101	101	110	010
IV	000	001	010	011
E(IV)	111	110	011	100
Ciphertext	010	011	101	110

TEA – Tiny Encryption Algorithm

- Designed by Needham and Wheeler (1994)
- 64-bit block cipher
- 128-bit key
- 64-round Feistel structure, with the rounds being paired into 32 “cycles”
- Uses XOR, shift operations and modular addition
- Simple (effectively trivial) key scheduling
- TEA has some problems with key equivalencies:
    - Each key is equivalent to three others. So the effective key space is 126 bits.
    - TEA was used in the Xbox, not for encryption purposes, but for hashing. Basically TEA was used to check if memory had been tampered with.
- TEA is also vulnerable to related-key attacks. With 2^{23} chosen plaintexts, encrypted under two related keys, 2^{32} computations are enough to break TEA.
- Due to weaknesses of TEA, mainly the related-key attack, several variants have been proposed.
- XTEA (1997), Block TEA (1997), XXTEA (1998)
- Best attack against XTEA is 27 round related-key differential attack requiring $2^{20.5}$ chosen plaintexts under a related key-pair and required $2^{115.15}$ 27-round XTEA encryptions.

Lecture 4: AES and Message Authentication

- Block and key size
- No. of round
- Round transformation
  - ByteSub
  - ShiftRow
  - MixColumn
  - AddRoundKey
- Stream Cipher
  - RC4

AES

- Implement symmetric key cryptography as a block cipher
- Block size of 128 bits
- Key sizes of 128, 192, and 256 bits
- Number of rounds, N_r depends on the key size

Key Size	Rounds
128	10
192	12
256	14

- Round transformation

```
Round (State, RoundKey)
{
        ByteSub(State);
        ShiftRow(State);
        MixColumn(State);
        AddRoundKey(State, RoundKey);
}
```

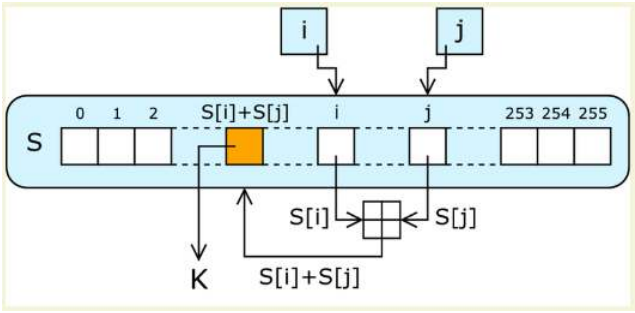
- Final round → no MixColumn

```
FinalRound(State, RoundKey)
{
        ByteSub(State);
        ShiftRow(State);
        AddRoundKey(State, RoundKey);
}
```

**** THE REST OF AES, REFER TO NOTES ***

RC4

- **Stream ciphers**
  - In block ciphers, plaintext characters are grouped in blocks and then each block is encrypted.
  - In stream ciphers, characters are encrypted one at a time.
    - For example, Vigenere Cipher
- Developed by Rivest (1987)
- Encryption and Decryption operations are the same
- Variable sized secret key up to 256 bytes.
- RC4 operates on bytes
- Used in the WEP protocol, SSL/TLS
  - Wired Equivalent Privacy (Wireless)
  - Secure Sockets Layer / Transport Layer Security (Web browsers ←→ servers)
- RC4 generates a pseudo-random stream of bits (a key-stream). As with any stream cipher, these can be used for encryption by combining it with the plaintext using XOR.
- Decryption is performed the same way (since exclusive-or is a symmetric operation)
- To generate the key stream, the cipher makes use of a secret internal state which consists of 2 parts:
  1. A permutation of all 256 possible bytes (denoted “S” below)
  2. Two 8-bit index pointers (denoted “i” and “j”)
  3. The permutation is initialized with a variable length key, typically 40 and 256 bits, using the key scheduling algorithm (KSA). Then the stream of bits is generated by a pseudo-random generation algorithm.



The lookup stage of RC4. The output byte is selected by looking up the values S(i) and S(j), adding them together modulo 256, and then looking up the sum in S; S(S(i) + S(j)) is used as a byte of the key-stream, K.

- Encryption / decryption (Byte by byte processing)

```
i = 0
j = 0
loop until all message encrypted
        Get the next input byte
        i = (i + 1) mod 256
        j = (j + S[i]) mod 256
        swap (S[i],S[j])
        k = S[(S[i], S[j]) mod 256]
        output: XOR k with input byte
```

- **Attacks against RC4**
  - RC4 is easily broken if a **key is used twice**.
    - Weakness in WEP due to key re-use
  - There is a need to discard the first output (1024 bytes) of the keystream.
    - Statistics of first bytes are **non-random**

Message Authentication Code (MAC)

- Error detection vs MAC
- Unconditionally secure MAC
- Block cipher-based MAC

Message Integrity

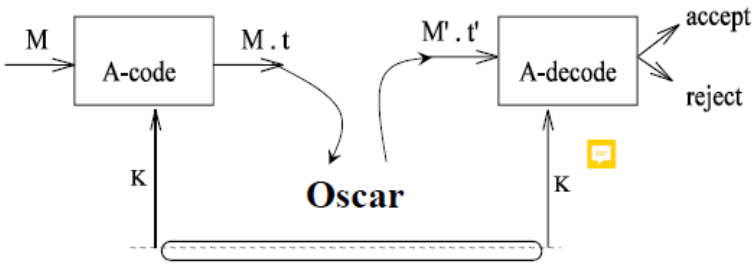
- One common mechanism of implementing protection against either threat is to append to the message a tag or checksum.
  - Tag is a string of bits, in a binary representation.

Error detection

- Algorithms for error detection **do not need a secret key**
- An algorithm for generating parity bits:
  - Attach **3 parity bits** to each block by finding remainder of the integer **divided by 7**.
  - Consider the message, 01111101 = 125
  - The parity bits are calculated as 125 mod 7 = 6 = 110 (binary)
  - So, we send the block 01111101 110
  - If one-bit error occurs, e.g. the receiver gets 01011101 110, they detect the error as follows:
    - 01011101 = 93 mod 7 = 2 = 010 not equal to 110
- Error-detecting codes will not work if there is an active attacker
  - Do not provide protection against spoofing, i.e. against deliberate modification or impersonation attacks.
- Attacker can modify the text and send it to the receiver. receiver will not detect that there has been modification made to the message

Message Authentication Codes

- **Transmitter and receiver share a secret key K. To transmit M, the transmitter calculates a MAC and appends it to M, thus (M.t = $MAC_k(M)$¿**



- The receiver receives a message (M.X). It uses the key K and M to calculate $MAC_k(M)$¿ and compare it with X. If the 2 match, the received message is accepted as authentic.
- The MAC is also called a cryptographic checksum
- A MAC is secure if forging ($M, MAC_k(M)$¿ , i.e. modifying it without being detected is hard.
- Unconditional security MAC or Practical Security MAC
  - A MAC with unconditional security requires many key bits.
  - Unconditional security in this case means that the chance of success of the enemy is always less than 1.

An unconditionally secure MAC

- Suppose our message consists of a number between *0* and *p-1*, where *p* is a prime.
- Any such message can be broken into blocks of size $log_2 p$.
- The transmitter and the receiver choose/establish/exchange a key **K** which is a pair of numbers; **K**=(a,b) where $0 \le a, b \le p-1$.
- Then

$$MAC_K(M) = a\,M + b \bmod p$$

- The enemy observes **(M, MAC_K(M))**, but without knowing **K** cannot change **M** and recalculate its cryptographic checksum.
- This is an example of an Authentication-code (A-code).

- Example: **p=11, K=(a,b)=(2,3).**
- To find the **MAC** for a message 5, we calculate
  $MAC_K(5)= 2*5 + 3 \bmod 11 = 2$
- The enemy doesn't know **K,** but does know a*5 + b mod 11 = 2.
- For an arbitrary choice of 'a' (11 values) the enemy can calculate a corresponding value for 'b'. So the possible keys are (0,2), (1,8), (2, 3), …
- So the chance of correctly guessing the key is 1/11.
- The tag is always one of the 11 elements.
- If the enemy receives another message **M'** whose MAC is calculated using the same key; say M'=7 with $MAC_K(7)=5$, then the enemy can solve the following equations:
  a*5 + b mod 11 = 2 a * 7 + b mod 11 = 6
  to find the key **K=(2,3)**.
- Thus the key must be changed with every message.

- In general it can be proven that:

  Unconditionally secure authentication systems require a large amount of key amount and so are impractical.

- **Example:** Design a MAC system with unconditional security such that the chance of success for an intruder is $P \leq 0.01$.
- Let **p=101, and K=(a,b)= (5,11)**.
- A message is an integer in the range 0-100; that is it requires 7 bits to represent it. The key is 14 bits long. The transmitted message is 14 bits long.

  $MAC_K(90)= 5 * 90 +11 \bmod 101$ 0111001.
  So the message sent is 1011010 **0111001**.

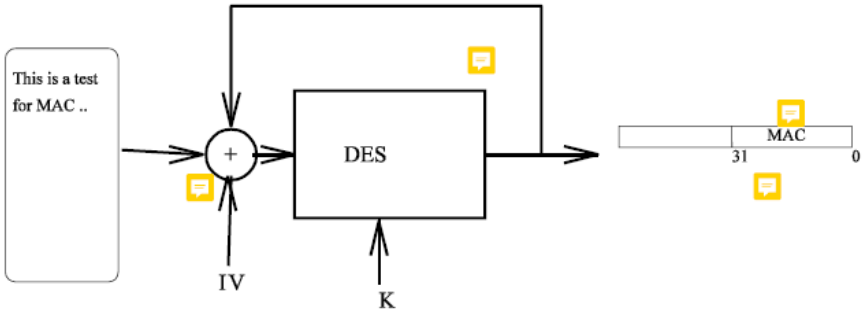
The chance of the enemy's success is P=1/101 <1/100.

MAC is very inefficient:

- In terms of the number of bits appended to the message
- In terms of the amount of key information, especially when we consider it requires a different key for every message
- Similar to the one-time-pad in encryption
  - Unconditionally secure encryption
  - Key information required = the information in the message to be transmitted.
- In a MAC with practical security, it is always possible to forge a message, but the amount of computation required to do so is large, to the extent it is impractical for the attacker to do so.

Block cipher based MAC

- A block encryption algorithm in CBC (chain-block cipher) mode can be used to generate a checksum.
- In CBC mode, the ciphertext gets “feed” back through the encryption algorithm.
- Example: Consider a 32-bit MAC generated using CBC mode of DES. For an arbitrary length message M, $MAC_K(M)$ is the right 32 bits of the output of the algorithm



In order for enemy to forge the message, he needs to break the DES.
Complexity to break this - 2 ^ 56

To improve the security:

- replace this DES with triple DES
- complexity of breaking triple DES: 2 ^ (2 x 56)
- but to break the MAC, it will still be 2 ^ 32
- what to do?
  - extend the size of MAC to 64 bits
  - hence complexity will change from (2 ^ 32) to (2 ^ 64)

Attacking block-based MAC

Question 1: What is the cost/chance of the enemy successfully forging a message?

Attack 1:

- Enemy uses an Exhaustive key search to find the key
- Then modifies the message as desired
- Then calculates the MAC for the new message using the key.

This cost something like $2^{64} operations$

Attack 2:

- The enemy chooses any message
- The enemy randomly selects a 32-bit string and attaches it to the message.
- The change of the enemy succeeding is $\geq \frac{1}{2^{32}}$

Question 2: Is the enemies cost/chance of success changes by using triple DES?

Answer: It is more difficult to find the key but the answer is no, because the best attack is NOT finding the key of the block cipher algorithm

Lecture 5: Public key cryptography

- Advantages
- One-way trapdoor function
- Knapsack cryptosystem
  - Super-increasing knapsack
  - Trapdoor knapsack
- RSA
  - Key generation
    - Primality test
  - Encryption
  - Decryption
  - Correctness
  - Security assumption
  - Fast Exponentiation
  - Common modulus attack

PKC

Public key cryptography (PKC) is an encryption technique that **uses a paired public and private key** (or asymmetric key) algorithm for secure data communication. A message sender uses a **recipient's public key** to encrypt a message. To decrypt the sender's message, only the **recipient's private key** may be used.

Example: RSA (Rivest, Shamir and Adelman), DSA (Digital Signature Algorithm)

Drawbacks of symmetric key crypto

- Key management – bottleneck in symmetric key cryptography
  - Generating good keys, distributing and storing them in a secure way.
- Encryption is not identified with an individual → doesn't provide authenticity.
  - Non-repudiation cannot be achieved.
    - provides proof of the integrity and origin of data
    - authentication that can be said to be genuine with high confidence
  - A non-repudiation service provides Alice protection against Bob later denying that some communication exchange took place.
    - Allows the elements of the communication exchange to be linked with the transmitter.
  - Alice will have irrefutable evidence to support her claim.

Advantages of Public Key Cryptography

- Example: Diffie-Hellman (1975)
- Encryption and decryption keys are different
- Given a public key, it is computationally infeasible to discover the corresponding decryption key. Knowledge of Encryption algorithm does not imply knowledge of Decryption algorithm.
- A PK cryptosystem can provide confidentiality, because only the **receiver can decrypt and find the plaintext**, and authenticity, because only **the sender can create such a cryptosystem**.

- A PKC should be assessed under chosen-plaintext attack.
- We can construct a PKC using trapdoor one-way function which is resistant to such an attack.

Trapdoor one-way function

- A function f is called one-way if
  - For all X finding f(X) is easy
  - Knowing f(X) it is hard to find X
- A trapdoor is a piece of knowledge which makes it easier to find X from f(X).
- A function which looks like a one-way function but is equipped with a secret trapdoor. If this secret door is known, the inverse can be easily calculated.

- Trapdoor one-way function can be used to realize a PKC:

  - E_z is easy to compute (from the public component of the key Z) but it is hard to invert.
  - Knowledge of z which is private component of the key (i.e. is the trapdoor) allows easy computation of D_z (inverting E_z)
- **Knapsack for encryption**
  - Knapsacks are implemented as block ciphers.
  - For a block size of n bits, we require a key or cargo vector, of positive integers a_i, $1 \leq i \leq$ n, referred to as weights
    - a= ($a_1, a_2, a_3, \ldots, a_n$¿
  - Then, to encrypt a message block of n bits:
    - X = ($x_1, x_2, x_3, \ldots, x_n$¿

      we find the number

      $$T = \sum_{i=1}^{n} x_i a_i$$

      and write it in a binary representation.

  - To decrypt: The receiver knows T and all a_i , so goes through all possible plaintext blocks to find a block that satisfies the condition.
  - In this system:
    - Encryption is easy and requires n additions
    - Decryption is difficult, even if the key is known.
    - Resulting cryptosystem is not useful
  - To construct an acceptable cryptosystem, Merkle & Hellman (1977) used a special case of the knapsack problem, involving super-increasing knapsacks, where decryption is easy with the key.

- **Super-increasing knapsacks**
  - A knapsack is super-increasing if each element of the cargo vector is greater than the sum of the preceding elements.

Example: a=(1,2,4,8)

Given T=14 say, it is easy to find $X=(x_1,x_2,x_3,x_4)$ such that $x_1+2x_2+4x_3+8x_4=14$.

i	a_i	Total	In?
4	8	14	1
3	4	6	1
2	2	2	1
1	1	0	0

X=0111

- Super-increasing knapsacks → could be used as a symmetric key cryptosystem
- But super-increasing knapsacks cannot be used as directory for public key cryptography, because making the encryption key public allows everyone to decipher X.
- Idea of Merkle and Hellman (1977) → start with an easy knapsack and then disguise it to look difficult for people without knowledge of the trapdoor.
- Trapdoor will only be known by the person who wants to decrypt the cryptogram. → it will be their private key

Trapdoor knapsacks

- Alice chooses a super-increasing knapsack, a= ($a_1, a_2, \ldots, a_n$¿ , as her private key.
- Then she chooses an integer m > $\sum_{i=1}^{n} a_i$, called the modular and
- A random integer w, called multiplier → relatively prime to m
  - Relatively prime → largest common factor of w and m is 1 → gcd (m, w) = 1
  - Condition implies there exists an inverse of w modulo m
- Alice's public key is b where
  - b = ($b_1, b_2, \ldots, b_n$¿ and $b_i = w * a_i \bmod m$ → multiplier * super-increasing mod (modular)
- Alice publishes a permuted version of b as her public key. Her secret key is (a, m, w)

- **To encrypt**: Bob sends a message X to Alice by computing:
  - $T = \sum_{i=1}^{n} x_i b_i$

- **To decrypt**: Alice receives T and …
  - Uses the inverse of w, w^{-1} , to calculate:
    - R = $w^{-1}T \bmod m$
  - Uses the easy knapsack a to find X, since R = a.X

$$
\begin{aligned}
R &= w^{-1}T(\bmod m) \\
&= w^{-1}\sum_{i=1}^{n} b_i x_i (\bmod m) \\
&= w^{-1}\sum_{i=1}^{n} (wa_i)x_i (\bmod m) \\
&= \sum_{i=1}^{n} (w^{-1}wa_i)x_i (\bmod m) \\
&= \sum_{i=1}^{n} a_i x_i (\bmod m) \\
&= a.X
\end{aligned}
$$

Brute force attack

- For those who do not know the secret trapdoor, decryption requires an exhaustive search through all 2^n possible X.
- Example:
  - We had: a= (171, 197, 459, 1191, 2410)
  - Let w= 2550 and m = 8443
  - b= (5457, 4213, 5316, 6013, 7439) is the public cargo vector
- Multiple layers and the fall of knapsacks
  - The disguising process can be repeated several times on the cargo vector to create more and more difficult knapsack problems (w_1, m_1¿,(w_2, m_2) , … The result is → not equivalent to a single (w, m) transformation.

Example: PKC using a trapdoor knapsack

- Secret key **a**=(2, 5, 10, 21).
- Trapdoor: **m**=39>38 (sum of weights).
  - Random w, **w**=15.
  - gcd algorithm, gcd(39,15)=3≠1
  - Another w, **w**=11.
  - gcd(39,11)=1
  - Inverse of 11 mod 39 = 32
    (32 = -7+39)

- Using **w**=11 and **m**=39 disguise **a**.
  b_1=2*11=22 (mod 39)
  b_2=5*11=16 (mod 39)
  b_3=10*11=32 (mod 39)
  b_4=21*11=36 (mod 39)
- Public key: **b**=(22, 16, 32, 36).

- Secret key: **a**=(2, 5, 10, 21).
  (w,m)=(11, 39)
  (remember also that $\mathbf{w}^{-1}$=32)

- To decrypt a message T=48…
  – Find R = 48*32 mod 39 = 15
  – Use the cargo vector **a** to decrypt R and find X=(0, 1, 1, 0).

  – We see this agrees with the public key version too (16+32=48).
  – In useful sized examples it would be not trivial to find the solution using just the public key, although it is easy to check the answer using it.

PKC can be used for:

- Confidentiality (secrecy)
- Authentication (digital signatures)

2 algorithms that can easily be used for both secrecy and authentication:

- **RSA and ElGamal**

Public Key Cryptography II

RSA

- RSA Public Key Cryptosystem (Rivest, Shamir and Adleman (1978)) most popular and versatile PKS
- Supports secrecy and authentication and can be used to produce digital signatures
- Uses the knowledge that it is easy to find primes & multiply them together
  - E.g. p1 * p2 = composite number, where p is prime
  - But it is difficult to factor a composite number (like the one way function)
    - Easy to compute f(x) given x, but difficult to find x given f(x).

The Euler phi Function

For $n \geq 1$, $\phi(n)$ denotes the number of integers in the interval [1, n] which are relatively prime to n. The function ϕ is called the Euler phi function (or the Euler totient function).

Fact 1. The Euler phi function is multiplicative, i.e. if gcd(m, n) = 1, then $\phi(mn) = \phi(m) \times \phi(n)$.

Fact 2. For a prime p and an integer $e \geq 1$, $\phi(p^e) = p^{e-1}(p-1)$.

- From these two facts, we can find ϕ for any composite n if the prime factorization of n is known.

RSA Algorithm

The algorithm

1. Choose two primes p and q. Compute n = pq and $m=\phi(n)= (p-1)(q-1)$.
   - $\phi(n)$ is Euler's totient function: It is the number of positive integers less than n that are relatively prime to n.
2. Choose e, $1 \leq e \leq m - 1$, such that gcd(e,m)=1.
3. Finds d such that ed=1 mod m.
   - This is possible because of the choice of e.
   - d is the multiplicative inverse of e modulo m and can be found using the extended Euclidean (gcd) algorithm.
4. The **Public key** is **(e, n)**.
   The **Private key** is **(d, p, q)**.

Encryption and decryption

- Let **X** denote a plaintext block, and **Y** denote the corresponding ciphertext block.
- Let **(z_A, Z_A)** denote the private and public components of Alice's key.

- If Bob want to encrypt a message **X** for Alice. He uses Alice's public key and forms the cryptogram:

$$Y = E_{Z_A}(X) = X^e \bmod n$$

- When Alice wants to decrypt **Y**, she uses the private component of her key z_A**=(d,n)** and calculates

$$X=D_{z_A}(Y) = Y^d \bmod n$$

- **X** and **Y** are both integers in {0, 1, 2, …, n-1}.

- **Example**: Choose p=11 and q=13.
  n=11*13=143
  m=(p-1)(q-1)=10*12=120
  e=37 ➔ gcd(37,120)=1

Using the gcd algorithm we find d such that ed=1 mod 120; d=13 → de=481.

X,Y∈{0,1,…,142}

- **To encrypt**: Break the input binary string into blocks of *u* bits, where $2^u \leq 142$, so we choose u=7.
  In general there is some agreed **padding scheme** from the message space into the space on which a "single run" of the cipher can act. Optimal Asymmetric Encryption Padding (OAEP) is often used.
- For each 7-bit block **X**, a number between 0 and 127 inclusive, we calculate the ciphertext as $\mathbf{Y=X^e}$.
- For X=2=(0000010) we have
  $E_Z(X)=X^{37}=12 \pmod{143}$ ➔ Y=0001100
- **To decrypt**: $X=D_Z(Y)=12^{13}=2 \pmod{143}$

  - An important property of the RSA algorithm is that encryption and decryption are the same function: both exponentiation mod n

$$E_{Z_A}(D_{Z_A}(X)) = X$$

  - Example: first decrypt → $2^{13}=41 \, mod \, 143$, then encrypt:
    $41^{37}=2 \, mod \, 143$
  - Basis of using RSA for authentication

Using RSA:

1. **Confidentiality**: To hide the content of a message X, A sends $E_{Z_B}(X)$ to B.
2. **Authentication**: To ensure integrity of a message X,
   a. Alice signs the message by using her decryption key to form $D_{Z_A}(X)$ and sends (X, $D_{Z_A}(X)$) = (X, S) to Bob.
   b. When Bob wants to verify the authenticity of the message:
      i. He computes X' = $E_{Z_A}(S)$
      ii. If X'=X the message is accepted as authentic and from Alice.
   c. Both message integrity and sender authenticity are verified.
      i. This is true because even one bit change to the message can be detected, and because z_A is only known to Alice (Alice's private key)
   d. Method is inefficient.
3. **Secrecy and authentication:**

a. A sends Y = $D_{Z_A}\left(E_{Z_B}(X)\right)$ to B

b. B recovers X= $D_{Z_B}\left(E_{Z_A}(Y)\right)$ → Bob decrypt using his private key

This scheme provides non-repudiation if Bob holds on to $D_{Z_A}(X)$.

That is, Bob is protected against Alice, trying to deny sending the message.

Finding primes:

- Given a number n, there exist efficient algorithms to check whether it is prime or not. Such algorithms are called primality testing algorithms
- Prime generation for RSA – guessing and testing.
- Primality Testing: Deterministic algorithms for proving primality are non-trivial and only advisable on high performance computers.
  - Probabilistic tests allow an educated guess as to whether a candidate number is prime or not.
  - This means that the probability of the guess being wrong can be arbitrarily small.

Lehman's test

- **Is a primality test; it determines probabilistically whether a given integer is composite or a prime.**
- Let n be an odd number. For any number a define:
  - e(a, n) = $a^{\frac{n-1}{2}} \bmod n$
  - G = {e (a, n)} : G, a $\in$ Z_n^* , where Z_n^* = {1, 2, …., n-1}
- Example: n =7
  - (n – 1) / 2 = 6 / 2 = 3
  - $2^3 \bmod 7 = 8 \bmod 7 = 1$
  - $3^3 \bmod 7 = 27 \bmod 7 = 6$
  - $4^3 \bmod 7 = 64 \bmod 7 = 1$
  - $5^3 \bmod 7 = 125 \bmod 7 = 6$
  - $6^3 \bmod 7 = 216 \bmod 7 = 6$
  - G = {1, 6}
- Lehman's theorem:
  - If n is odd, G = {1, n – 1} if and only if n is prime
  - For example, n = 15 isn't prime: (n – 1) / 2 = 7
  - $2^7 \bmod 15 = 8 \bmod 15$
  - $3^7 \bmod 15 = 12 \bmod 15$
- test from 1 up to (n-1) to be 100% sure that the number is prime number
- prime number * prime number give non-prime number

```
if (gcd(a,n) >1) return('composite')
else
   if (a^(n-1)/2=1) or (a^(n-1)/2=-1)
      return('prime_witness')
   else
      return('composite')
```

RSA Security Assumptions

The RSA Assumption is that the RSA Problem is hard to solve when the **modulus n** is sufficiently large and randomly generated, and the plaintext M (and hence the ciphertext C) is a random integer between 0 and n − 1. The assumption is the same as saying that the **RSA function is a trapdoor one-way function** (the private key is the trapdoor). The randomness of the plaintext M over the range [0, n−1] is important in the assumption. If M is known to be from a small space, for instance, then **an adversary can solve for M by trying all possible values for M**. The RSA Problem is the basis for the security of RSA public-key encryption as well as RSA digital signature schemes.

Weak implementations

1. **Common modulus attack******:
   a. Consider a group of users who's public keys consists of the same modulus and different exponents. If an intruder intercept 2 cryptograms where
      i. They are encryptions of the same message with different keys.
      ii. The two encryption exponents do not have any common factor. Then the attacker can find the plaintext.

The enemy knows $e_1, e_2, N, Y_1 \wedge Y_2$ and furthermore that $Y_1 = X^{e_1} \bmod N$ and $Y_2 = X^{e_2} \bmod N$

Since e_1 and e_2 are relatively prime, the Extended Euclidean algorithm can be used to find a and b such that $ae_1 + be_2 = 1$

$$X = \left(Y_1^{\,a} \times Y_2^{\,b}\right) \bmod N$$

$$X = \left(\left(X^{e_1}\right)^a \times \left(X^{e_1}\right)^b\right) \bmod N$$

$$X = X^{a \cdot e_1 + b \cdot e_2} \bmod N$$

Since $ae_1 + be_2 = 1$,

$$X = X(1) \bmod N$$

$$X = X$$

Question:

RSA is insecure against chosen ciphertext attack. Explain or show by example that RSA is insecure against chosen ciphertext attack.

Answer:

An adversary wants to decrypt y, a ciphertext, in the form of y = $m^e \bmod n$ to obtain the plaintext m. For example, Alice sends y to Bob and Charlie (the adversary) wants to know what message Alice sends.

Assumption made:

Charlie(the adversary) has access to a decryption oracle that can decrypt any ciphertext message except of the ciphertext y (i.e. **y =** $m^e \bmod n$) that Alice sends.

Charlie creates a ciphertext y' that is not equal to y, i.e. y' ≠ y, such that y' = y^n x y mod n

Charlie computes y'' by randomly choosing a plaintext m_1 such that gcd (m_1, n¿ = 1, and computes $y^n = m_1^e$ **mod n**

Thus y' = m_1^e x y mod n

= m_1^e mod n (let $M_1^e = m_1^e$ x y mod n)

Since $M_1^e = y' \bmod n$

= y^n x y mod n

M_1^e = m_1^e x $m^e \bmod n$

M_1 = m_1 x m mod n

m = M_1 ($\frac{m_1}{¿¿^{-1}}$ mod n

Since e and n (Bob's public key) are public information and gcd(m_1 ,n) = 1, m can be easily computed.

Question 2:

If an attacker has a polynomial algorithm to factor n, which is a large arbitrary integer. Why this makes RSA based public key cryptography insecure?

Answer:

The reason is that the attacker can compute the victim's private key from the victim's public key.

It is noted that *ed=1 mod ϕn*, and *ϕn=(p-1)(q-1)*. If the attacker has a polynomial algorithm, then the attacker can compute *n=p ×q*. Knowing the value of *n*, the attacker can then compute *d* from the victim's public key *e* using *d=e-1 mod ϕ(n)*. Thus, the message can is revealed.

Question 3:

Common modulus attack:

Adam and Barbie share the same modulus *n* for RSA to generate their encryption key e_A and e_B. Charlie sends them (Adam and Barbie) the same message *m* encrypted with e_A and e_B respectively. The resulting ciphertexts are c_A and c_B. Eve intercepts both c_A and c_B. Show how Eve can use the *common modulus attack* to compute the plaintext or the message *m* sent by Charlie if gcd(e_A, e_B) = 1?

Answer:

Eve knows the ciphertext $c_A \equiv m^{e_A} \bmod n$ and $c_B \equiv m^{e_B} \bmod n$. Eve also knows that the $\gcd(e_A, e_B) = 1$. Thus Eve can compute the inverse multiplicative of e_A and e_B using extended Euclidean algorithm to get $(e_A)(a) + (e_B)(b) = 1$.

Eve then computes $(c_A)^a \times (c_B)^b \bmod n$ which she can obtain the message m as follow:

$$= (m^{e_A})^a \times (m^{e_B})^b \bmod n$$
$$= (m^{e_A \times a}) \times (m^{e_B \times b}) \bmod n$$
$$= m^{(e_A)(a)+(e_B)(b)} \bmod n$$
$$= m \bmod n$$
$$= m$$

Lecture 6: Public Key Cryptography IV (Rabin and Elgamal)

Rabin & ElGamal

- Key generation
- Encryption
- Decryption
- Security assumption

The Rabin cryptosystem

- Security of this system is equivalent to the difficulty of factoring

Key generation:

- Bob randomly chooses 2 large primes, p & q and calculates N = pq
- The public key is N and the secret key is (p, q).

To Encrypt:

- The ciphertext Y for a message (plaintext) X: Y = $X^2 \bmod N$

To Decrypt:

- Bob must find the square root of Y mod N
- Knowing (p, q) it is easy to find the square root, otherwise it is provably as difficult as factoring.

Special case:

- p and q are 3 mod 4
- we construct 4 intermediate factors.
  - $x_1 = Y^{(p+1)/4} \bmod p$
  - $x_2 = p - x_1$
  - $x_3 = Y^{(q+1)/4} \bmod q$
  - $x_4 = q - x_3$

We define:

a= q ($q^{-1} \bmod p$¿ b = p ($p^{-1} \bmod q$¿

This means:

For example, that you calculate $q^{-1} \bmod p$ and multiply the result by q

4 possible plaintexts → 4 ways to decrypt rabin

1. $x_1 = (a x_1 + b x_3) \bmod N$
2. $x_2 = (a x_1 + b x_4) \bmod N$
3. $x_3 = (a x_2 + b x_3) \bmod N$
4. $x_4 = (a x_2 + b x_4) \bmod N$

An example:

1) We choose p =7 and q =11, so N = 77
   - note that p and q are 3 mod 4 (criteria)
2) Bob's public key is 77 and his private key is (7, 11)
3) To encrypt X = 3, Alice calculates:
   a. Y = $3^2 \, mod \, N = 9 \, mod \, 77$
4) To decrypt Bob calculates:
   a. $x_1 = 9^2 \, mod \, 7 = 4$
   b. $x_2 = 7 - 4 = 3$
   c. $x_3 = 9^3 \, mod \, 11 = 3$
   d. $x_4 = 11 - 3 = 8$
5) Bob then finds a and b:
   a. 7(7^{-1} mod 11) = 7 x 8 = 56 (By using Extended Euclidean, we found that inverse 7 mod 11 is -3, 11 – 3 = 8 hence (7 x 8)
   b. 11 ($11^{-1} \, mod \, 7$) = 11 x 2 = 22
6) … and then the 4 four possible plaintexts:
   a. $x_1 = ¿$ 4 x 22 + 3 x 56 = 11 + 14 = 25 mod 77
   b. $x_2 = ¿$ 4 x 22 + 8 x 56 = 11 + (-14) = -3 = 74 mod 77
   c. $x_3 = ¿$ 3 x 22 + 3 x 56 = -11 + 14 = 3 mod 77
   d. $x_4 = ¿$ 3 x 22 + 8 x 56 = -11 – 14 = -25 = 52 mod 77

Advantages & disadvantages:

- Provable security
- Unless RSA exponent e is small, Rabin's encryption is considerably faster than RSA, requiring one modular exponentiation
  - Decryption requires roughly the same time as RSA
- Decryption of a message generates 4 possible plaintexts. The receiver needs to be able to decide which one is the right message.
  - One can append messages with known patterns, eg. 20 zeros, to allow easy recognition of the correct plaintext.
- Rabin's system is mainly used for authentication (signatures).

Generator of $Z_p * ¿$

- An element α is a generator of $Z_p * ¿$ if α^i , 0 < i $\leq$ p -1 generates all number 1, …, p -1
- Finding a generator in general → hard problem → no efficient algorithm known.
- If factorization of p – 1 is known, it is not hard.
- In particular, if p = $2_{p_1} + 1$ where p_1 is also a prime.
- α in $Z_p * ¿$ and a $\neq \pm$ 1 mod p
- Then α is a primitive element if and only if
  - $\alpha^{\frac{p-1}{2}} \neq$ 1 mod p → if it's not equal to 1 mod p, then alpha is a generator else it's not. → this algorithm is always correct.
- Suppose α in $Z_p * ¿$ and α is not primitive element.

- Then - α is a primitive element
- For example: $Z^{¿}_{11}$
  - 3^5 = 1 mod 11
  - 3 is not primitive, -3 = 8 is primitive. → 11- 3 = 8 → 8 will be a generator since 3 is not a generator.

An example $:\mathbf{Z}_{11}^{*}$

	1	2	3	4	5	6	7	8	9	10
1	1	1								
2	**2**	**4**	**8**	**5**	**10**	**9**	**7**	**3**	**6**	**1**
3	3	9	5	4	1	3				
4	4	5	9	3	1	4				
5	5	3	4	9	1	5				
6	**6**	**3**	**7**	**9**	**10**	**5**	**8**	**4**	**2**	**1**
7	**7**	**5**	**2**	**3**	**10**	**4**	**6**	**9**	**8**	**1**
8	**8**	**9**	**6**	**4**	**10**	**3**	**2**	**5**	**7**	**1**
9	9	4	3	5	1	9				
10	10	1	10							

Discrete Logarithm Problem (DLP):

Input:

- $Z_p*¿$
- g in $Z_p*¿$, g a generator of $Z_p*¿$
- h in $Z_p*¿$

Find the unique number a < p such that h = g^a

DL Assumption: There is no efficient algorithm (polynomial time) to solve DL problem

- When p is small, discrete log can be found by exhaustive search
- The size of prime for which DL can be computed is approx. the same as the size of integers that can be factored.
- DL is an example of one-way function:
- A function f is one-way if
  - Given x, finding f(x) is easy
  - Given y, finding x such that f(x) = y is hard.

The ElGamal Cryptosystem

- Public-key cryptosystem
- Based on discrete logarithm problem
  - Hard to find k < p, such that y = $g^k \bmod p$
  - g is a generator of $Z_p*¿$
  - k is in $Z_p*¿$
- Proposed by Tather Elgamal in 1984
- A variant of Diffie-Hellman providing a one-pass protocol with unilateral key authentication.

The El Gamal Cryptosystem

- **Key generation**:
  - Alice chooses a prime **p** and two random numbers **g** and **u**, both less than **p**, where **g is** a generator of $\mathbf{Z_p^*}$.
  - Then she finds:

$$y = g^u \bmod p$$

Alice's public key is **(p, g, y)**, her secret key is **u**.

- To encrypt a message **X** for Alice, Bob chooses a random number **k** such that **gcd(k,p-1)=1**. Then he calculates:

$$a = g^k \bmod p$$

$$b = y^k \times X \bmod p$$

- The cryptogram is **(a,b)**
- The length is twice the length of the plaintext.

- To decrypt **(a,b)** Alice calculates

$$X = \frac{b}{a^u} \bmod p$$

ElGamal Cryptosystem

Public Key : $y = g^u$, *Secret Key* : u

Encryption :

$a = g^k \mod p$

$b = y^k \times X \mod p$, *ciphertext* : (a,b)

Decryption :

$X = \frac{b}{a^u}$, *division uses Euclid extended Algorithm*

To avoid division :

$X = b \times a^{-u} = b \times a^{p-1-u}$

Example

- We choose p=11, g=3, u=6.
- We calculate $y=3^6=3 \mod 11$
  $3^6 \mod 11=(3^2)^3 \mod 11$
  $=(-2)^3 \mod 11$
  $= - 8 \mod 11$
- The public key is **(11,3)**.

- To encrypt **X=6**, Bob chooses k=7 and calculates:
  $a=3^7=9 \mod 11$, $b=3^7 \times 6=10 \mod 11$
  The cryptogram is **(9,10)**.

- To decrypt Alice finds:
  $X=10/9^6=10/9=10 \times 5=\mathbf{6} \mod 11$

Security problems with ElGamal

- Similar to RSA:
- Given (a, b) = (g^k, y^k x m) mod p
- Eve:
  - chooses a random number r in Z_p^{ι}
  - computes rb
  - sends (a, rb) to Alice
- Alice returns the decryption: r x m mod p

- Eve will find $\frac{r x m}{r}$ = m mod p

Provable security

Adversary power

- Eve is polynomially bounded.
  - Time and memory is bounded by a polynomial in the size of input
  - **Cannot exhaustively search**
- Eve has access to 'oracles' (decryption box):
  - Can ask queries and receive responses
  - Attacks are modelled as a 'game' between an attacker Eve and an innocent user (Oracle)

Question:

In El Gamal encryption, the private key *x* in an element in □□∗. The public key is given by □= g^x , where g is a primitive element in $Z_p^{¿}$. To encrypt message *m,* one generates a random key < □−1 , sets c_1 = g^k , and c_2 =× y^k and output the ciphertext □= c_1, c_2 . Describe the decryption process and show that it always returns the encrypted message *m*.

Answer:

To decrypt c_1, c_2 with private key x, the recipient compute m = $\frac{c}{(¿¿1)^x \, mod \, p}$ $\frac{c_2}{¿}$

in 3 steps:

i. Compute c_t = c_1^x mod p using fast exponentiation
ii. Compute the inverse $inv\, c_t = c_t^{-1} mod\ p$ using extended Euclidean algorithm
iii. Compute m = c_2 x $inv\, c_t$ mod p

The computation described above will always return the encrypted message m because:

m = $\frac{c}{(¿¿1)^x \, mod \, p}$ $\frac{c_2}{¿}$

m = $\frac{m \times y^k}{(g^k)^x} mod\, p$

m = $\frac{m \times y^k}{(g^x)^k} mod\, p$ since y = g^k

m = $\frac{m \times y^k}{y^k}$ mod p

m = m mod p

Hence, m = m

Digital Signatures

Digital signature using PKC

- RSA signature

ElGamal signature scheme

Digital Signature Standard (DSS)

Specially designed signatures:

- Blind Signatures
- Designated Verifier Signature, Undeniable Signature
- Group Signature, Ring Signature

Digital Signature

- Introduced Diffie-Hellman (1976)
- Electronic analog of handwritten signature
- Ensures integrity of the message and authenticity of the sender.
  - If Bob gets a message which supposedly comes from Alice, he is able to check that …
    - The message has not been modified
    - The **sender is truly Alice**.
- Similar to Message Authentication Codes, only difference: **they didn't allow for the authenticity to check on the sender**. ***

Digital Signature using PKC

- Alice signs X by creating Y = $E_{z_A}(X)$ → using Alice's private key. The signed message is (X, Y)
- Bob validates Alice's signature by computing X' = $E_{Z_A}(Y)$ → using Alice public key and comparing it with X.
- Everyone has access to E_{z_A} and hence anyone can verify the signature by computing $E_{Z_A}(Y)$.
- Since E_{z_A} is a trapdoor one-way function, it is not possible for an intruder to find z_A – Alice's private key.
- Hence Alice's signature cannot be forged.

Digital Signature can also provide non-repudiation. This prevents the sender from denying he/she has sent the message.

If Alice signs a message, since she is the only one who knows z_A hence she will be held accountable for it. → the system therefore provides non-repudiation.

If Alice claimed that E_{z_A} was compromised, we may use public notary.

- A public Notary N signs messages on top of Alice's signature. Hence Alice cannot claim the message is forged. N sends a copy of the signed message back to Alice.

Digital Signatures Schemes

The Elgamal Signature scheme

- Based on the difficulty of computing discrete logarithms over Z_p , where p is a prime.

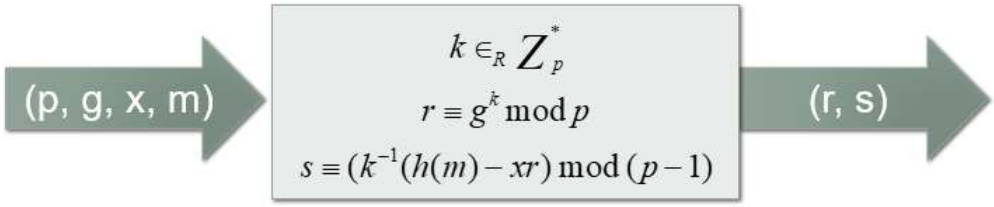
Setup and Key generation:

- The key generation algorithm ElGamal digital signature system is the same as the one employed by the ElGamal Asymmetric Encryption system.
- For every user, it generates a public ElGamal verification key (p, g, y) and a corresponding private ElGamal signing key x.
- All users may be using the same p and g.

Message signing:

- Select a number *k* from Z_p^* such that *gcd(k, p-1)=1.*
- Compute the first element of the signature $r = g^k \bmod p$.
- Hash the message *m*, and the result *h(m)* is used to compute the second element of the signature $s = k^{-1}(h(m) - xr) \bmod (p-1)$

Elgamal as digital signature:



The algorithm takes as input a private ElGamal signing key x together with the modulus p, generator g, and a message m.

Generates as output the digital signature for m, that consists of two numbers, r and s, that are both element of Z_p^*

Signature verification:

- Verify that
  1. $1 \le r \le p-1$
  2. $g^{h(m)} \bmod p = y^r r^s \bmod p$
- The signature is valid only if both the verification checks are positive.

(Note: it is important to verify that $1 \le r \le p-1$ because ElGamal signing algorithm is probabilistic which means there may be many valid signatures for any given message.)

Example: Suppose that Alice wants to send to Bob a message m=1463.

Key generation:

1. Alice chooses a prime number p = 2357 and a generator g = 2 of Z^*_{2357}
2. Alice chooses a private key x = 1751 and compute

$$y = g^x \bmod 2357$$
$$= 2^{1751} \bmod 2357$$
$$= 1185$$

As an exercise, compute 2^{1751} mode 2357.

- Alice's public key is thus (2357, 2, 1185).

Message signing:

- For simplicity, I choose an identity hash function to hash the message, thus

  $h(m) = h(1463) = 1463$.

- To sign the hashed message $h(1463)$, Alice selects a random integer k = 1529 and compute $r = g^k \bmod p$.

$$r = g^k \bmod p$$
$$= 2^{1529} \bmod 2357$$
$$= 1490$$

- Next Alice compute $s = k^{-1}(h(m) - xr) \bmod (p-1)$.

$$k^{-1} \bmod (p-1)$$
$$1529^{-1} \bmod 2356 = 245$$

Thus, s = 245 (1463 – (1751 x 1490)) mod 2356 = 1777.

- Alice's signature for m is thus the pair (r = 1490, s = 1777).

Signature verification:

- Bob verifies that

1. $1 \le r \le p-1$;

   $1 \le 1490 \le 2356$ positive!

2. Bob verifies that

   $g^{h(m)} \bmod p = y^r r^s \bmod p$

$$2^{1463} \bmod 2357 = 1185^{1490} \times 1490^{1777} \bmod 2357$$
$$1072 = 1072 \quad \text{positive!}$$

Both verifications are positive, thus Bob is assured that the message and signature are authentic.

ElGamal DSS – Summary

Public paramters:

- A large prime number p, a generator g of Z_p^*.

Key generation:

- Generate a random $x \in Z_{p-1}$ and compute

$$y = g^x \bmod p.$$
$$Secret\ key = x.$$
$$Public\ key = y.$$

Signing the message:

- Hashed message $h(m) \in Z_{p-1}$.
- Pick a random k $\in Z^*_{p-1}$, compute r = $g^k \bmod p$ and $s = k^{-1}(h(m) - xr) \bmod (p-1)$.

The signature is (r, s).

Verifying the message:

- Check that
  1. $1 \le r \le p-1$
  2. $g^{h(m)} \bmod p \equiv y^r r^s \bmod p$

Digital Signature Algorithm (DSA)

- **Key generation and setup**:
  - **q** a 160-bit prime is chosen
  - **p** a prime. 512-1024 bits long with the length a multiple of 64. **q** is a factor of **p-1**.
  - **h** a number less than **p-1** such that
    g= $h^{p-1/q}$ mod p > 1
  - **u < q**
  - **y=g^u mod p**

- **Public parameters**: p, q, g. (Multiple people could use these).
- **Private key**: u.
- **Public key**: y.

Generating and verifying a signature:

- To sign a message …
  - **A**lice generates a random **k**, such that **k<q**.
  - **A**lice computes
    r = (g^k mod p) mod q
    s = (k^{-1}(H(X) + ur))) mod q
- To verify a message Bob calculates…
    w=s^{-1} mod q
    t_1=(H(X)*w) mod q
    t_2=rw mod q
    v=((g^{t_1}*y^{t_2}) mod p) mod q
  … and accepts the signature if **v=r**.

Using the received values

H(X) is the hash of X. We will get to these soon.

Example:

- **Setup**:
  - p=29, q=7 (is a factor of 28).
  - h=3, $g=h^{(29-1)/7}=3^4=23 \bmod 29$
  - u=2 < q.
  - $y=g^2 \bmod 29 = 23^2 \bmod 29$

    $= (-6)^2 \bmod 29$

    $= 36 \bmod 29 = 7$

- **Signing:**
  - To sign **H(X)=5**, **A**lice chooses **k=4** (<q) and calculates $k^{-1}=2$.

    $r = (g^4 \bmod 29) \bmod 7$

    $= ((-6)^4) \bmod 29) \bmod 7$

    $= (36*36 \bmod 29) \bmod 7$

    $= (7*7 \bmod 29) \bmod 7$

    $= (49 \bmod 29) \bmod 7 = 20 \bmod 7 =$ **6**

    $s = k^{-1}(H(X) + ur) \bmod q$

    $= 2(5+2*6) \bmod 7$

    $= 34 \bmod 7$

    = **6**

- **Verifying**:
  $w = s^{-1} \bmod q = 6^{-1} \bmod 7 = \mathbf{6}$
  $t_1 = (H(X)*w) \bmod q$
  $= 5*6 \bmod 7 = \mathbf{2}$
  $t_2 = rw \bmod q$
  $= 6*6 \bmod 7 = \mathbf{1}$
  $v = ((g^{t_1}*y^{t_2}) \bmod p) \bmod q$
  $= ((23^2*7^1) \bmod 29) \bmod 7$
  $= (((-6)^2*7) \bmod 29) \bmod 7$
  $= ((7*7) \bmod 29) \bmod 7$
  $= \mathbf{6}$
- Since **v=r** the message and signature are accepted as authentic.

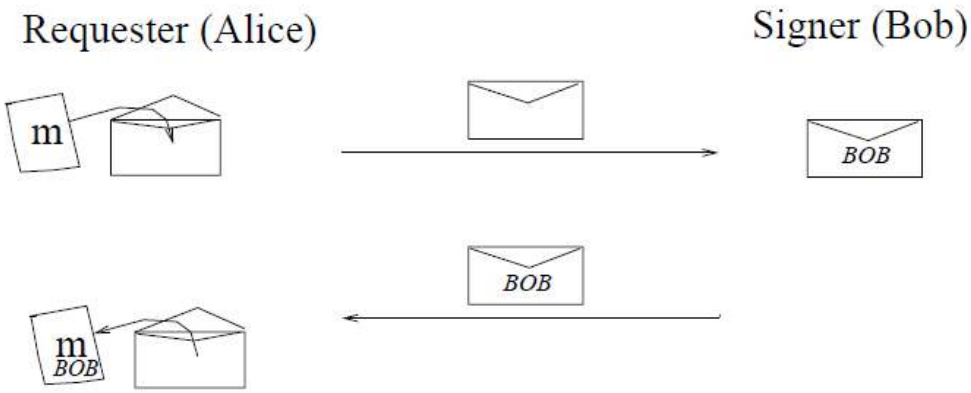
***Recommended hash function for DSA is SHA-1**

Blind Signature Scheme

- Necessary to get the signature of a party without allowing them to see the message.

In general:

- The requester wants to obtain the signer's signature of message m.
- The requester doesn't want to reveal m to anyone, including the signer.
- The signer signs m bindly, not knowing what they are signing.
- The requester can retrieve the signature.



Blind signatures using RSA

- **Setup**: **B**ob has (d,e), a (private,public) key pair.
  - $N=pq$, where p, q are large primes, associated with **B**ob.
- For a message **X**, that **A**lice wants **B**ob to sign, **A**lice constructs $\mu=mr^e \bmod N$, where $\mathbf{r} \in_R Z_N^*$, and sends μ to **B**ob.
- **B**ob signs μ and sends his signature $\sigma=\mu^d \bmod N$ back to **A**lice.
- **A**lice retrieves the signature s of m by computing:

$$s = \frac{\sigma}{r} = \frac{\mu^d}{r} = \frac{m^d r^{ed}}{r} = \frac{m^d r}{r} = m^d \bmod N$$

Undeniable signatures:

- Non-transferable signatures
- Signature can be verified only with the collaboration of the signer.
  - Could be useful for example, when a software distributor signs software. In this case, it is important that the verifiable software cannot be duplicated. That is, the signature can be different for different people and the distributor will be able to see if there is duplication.

Group signatures scheme:

- Properties: only members of the group can sign messages.
- The receiver can verify the signature but cannot discover which member of the group generated it.
- In the case of a dispute, the signature can be "opened" to reveal the identity of the signer.

Fail-stop signature schemes:

- Provides enhanced security against the possibility that a very powerful adversary might be able to forge a signature.
- In the event that Oscar is able to forge Bob's signature on a message, Bob will (with high probability) subsequently be able to prove that Oscar's signature is a forgery.

Threshold signature schemes:

- The parties need to work together to construct a proof of authority
- This problem can be solved by means of a secret sharing scheme.
- A (t, w) threshold scheme→ a method of sharing a key K among a set of w participants in such a way that any t participants can compute the value of K, but **no group of t-1 participants can do so**.

Question:

A signature scheme introduced by David Chaum, allows a person to get a message signed by another party without revealing any information about the message to the other party. This signing protocol is known as blind signature. In general, this is what happens:

- The requester wants to obtain the signer's signature of message *m*.
- The requester doesn't want to reveal *m* to anyone, including the signer.
- The signer signs *m* blindly, not knowing what they are signing.
- The requester can then retrieve the signature.

(i) Using RSA as the digital signature scheme, describe how the blind signature is realized.
(ii) Show or explain that the requester can indeed verify that the signature of the signatory is valid or correct.

(i) Setup:

- ➔ Bob has (d, e), a (private, public) key pair, and $N=pq$, where p, q are large primes, associated with Bob.
- ➔ For a message *m*, that Alice wants Bob to sign, Alice constructs □= mr^e □□□□ , where □∈$_n^i$ □□□ □ □□□□□′□ □□□□□□□ □□□ and sends μ to Bob. μ is known as the blinded message.
- ➔ Bob signs μ using his private key *d* and sends his signature □′= μ^d □□□ □ back to Alice. The signature □′ that Alice receives is Bob's signature on the blinded message.
- ➔ Alice then verify that Bob actually sign the blinded message by dividing Bob's blind signature □′ by *r*, such that □= $\frac{s'}{r}$ □□□ □= r^{-1} □′ □□□ □, to verify that it is correct.

(ii) Alice can verify that the message was signed by Bob by computing:

$$S = \frac{s'}{r} \bmod N$$
$$= \frac{\mu^d}{r} \bmod N$$
$$= \frac{(mr^e)^d}{r} \bmod N$$
$$= \frac{(m^d r^{ed})}{r} \bmod N \qquad \text{Since ed = 1 mod N,}$$
$$= \frac{m^d r}{r} \bmod N$$
$$S = m^d \bmod N$$

$$Hence,\ m = S^e \bmod N$$
$$= (m^d)^e \bmod N$$
$$m = m$$

Question 2:

In cryptography, in particular digital signature context, explain the term nonrepudiation.

Suggested answer:

Nonrepudiation provides protection against denial by one of the entities involved in communication of having participated in all or part of the communication. It is a service in digital signature that provides proof that the **message was sent** by the specified party as well as proof that the **message was received** by the specified party.

Question 3:

What is factorization problem? Show an example of a signature scheme that relies on the security of factorization problem

Answer:

Factorization refers to splitting of an integer number into a set of factors (a smaller set of prime numbers) which when multiplied together will get back the original integer. All integer numbers may be prime-factorized; i.e., expressed as a product of many prime numbers. When one has an integer number and wants to find the factors of these prime numbers, that can produce back the integer number, **is difficult**. This problem is known as factorization problem. Many public-key cryptosystems base on this factorization problem, including the **RSA cryptosystem** as well as **RSA digital signature system**.

RSA Digital Signature

Key generation:

- The key generation algorithm of RSA digital signature system is the same as the one employed by the RSA cryptosystem.
- Every user will generate his/her public key pair *(e, n)* and private key pair *(d, n)*. The user chooses two prime numbers *p* and *q* and compute the modulus *n* = *p* x *q*.
- The user next chooses two more numbers, *e* and *d*. The number *e* is coprime (relatively prime) to *(p-1)(q-1)*. The number *d* is chosen such that *((e* x *d) – 1)* is divisible by *(p-1)(q-1)*.
- The *(e, n)* pair is the public key, and the *(d, n)* pair is private key.

Message signing:

- To sign a message, the sender signs the message using his/her private key;
  i.e. S= m^d mod n

where

- S is the signature,
- m is the message,
- (d, n) the sender's private key.

- The sender sends the message *m* and the signature *S* to the recipient (receiver).

Signature verification:

- To verify that the message is indeed signed by the sender, the receiver verifies the message authentication using the sender's public key;
  i.e. m' = S^e mod n

where

- m' is the message recovered by decrypting the digital signature.
- S is the sender's digital signature
- (e, n) pair is the sender's public key.

If the message received m is the same as the recovered message m', the receiver can be assured that the messages sent are authentic.

Lecture 7: Hash Functions

- **Collision resistance**
- **Hash-then-sign**
- **Birthday attack**
- **Design**
  - **Block cipher (CBC)**
  - **Modular Arithmetic**
  - **SHA-1**

Collision resistance

- Collision → 2 distinct messages product the same message digest
- Example:
  - H(X) = X mod 10
  - H(56) = H(96) = H(156) ➔ 56 mod 10 = 6, 96 mod 10 = 6, 156 mod 10 = 6
- Hash functions with collision resistance → where it is **computationally difficult** to find 2 inputs x and y such that H(x) = H(y).
-

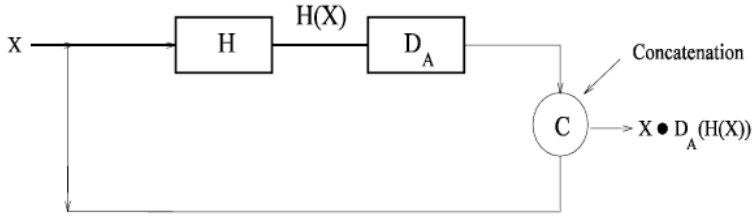
Cryptographic hash functions:

1. It can be applied to any size input
2. The output must be of fixed size
3. **One-way**: Easy to calculate but hard to invert
4. **Pre-image resistant**: For any given Y, it is difficult to find an X such that H(X) = Y
5. **Second Pre-image resistant**: Given X, it should be **difficult to find another** X_2 such that H(X_1) = H(X_2)
6. **Collision resistant**: it is computationally infeasible to find messages X and Y with X ≠ Y such that H(X) = H(Y).

Properties 1, 2 and 3 → required for **efficient signature generation**
Properties 4, 5, and 6 → required to **stop attackers forging signatures**

Signing with hash functions

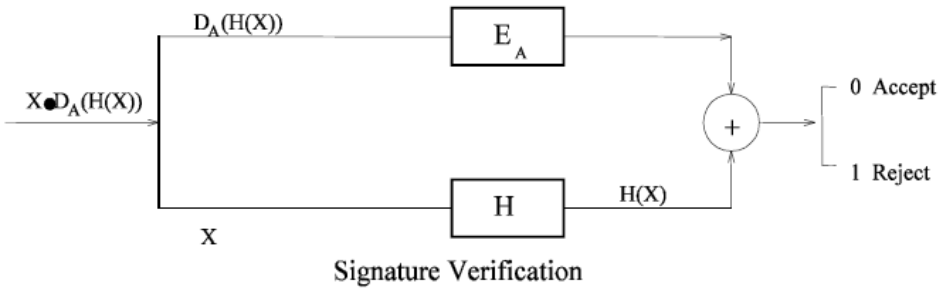
- Provide message integrity
- One of the main applications of hash function is for digital signatures
- To sign a message X, the hash value H(X) is calculated and signed.



Signature Generation

Verification

- Involves calculating the hash on the message again and comparing it with the transmitted hash.



Signature Verification

Assessing the security of hash functions (Birthday attack)

- Size of message space > digest space → we can always find collisions
- Example:
  - Let messages belong to $Z_p^{¿}$ = {1, 2, …, p -1} and digests belong to $Z_q^{¿}$ = {1, 2, …, q -1} where q is a prime and p > q
- Choose a number g $\in Z_q^{¿}$
- Define a hash function h(x) = g^x mod q

Birthday Attack

- Suppose there are m possible hash values (message digits). Assume the associated with the same number of messages. If we evaluate the hash value of k randomly selected messages, the probability of at least one collision is:

$$P(m, k) > 1 - e^{\frac{-k(k-1)}{2m}} = \varepsilon$$

- Chance of success in this attack depends only on:
  - The size of the message digest, and
  - The number of message digests are calculated for
- Does not depend on the specific hash function used.
  - Size of digest space puts a lower bound on the probability of success.

$$k = \frac{1}{1-\varepsilon}\,(¿¿)\,2m\ln ¿\sqrt{¿}$$

- Use lower bound to find the required size of digest space, that is the number of bits of the digest, if we assume certain values for the level of feasible computation.
  - $m = \frac{k^2}{2\ln\left(\frac{1}{1-\varepsilon}\right)}$

Techniques for hashing

Hashing can be divided into 3 main categories:

1. Techniques based on block ciphers (symmetric key).
2. Techniques based on modular arithmetic
3. Dedicated or designed hash functions (Others)

1. Hash functions from block ciphers

- Produce secure hash functions is not easy.
- Simplest way to use a block cipher in **cipher-block chaining mode**
- A Message Authentication Code (CBC-MAC), where the key must be known to both sides.
  - X = $X_1, X_2, \ldots, X_n$
  - $Y_i = E_k (X_i \oplus Y_{i-1}$¿
  - $H(X) = Y_n$
- Standardized for banking authentication.

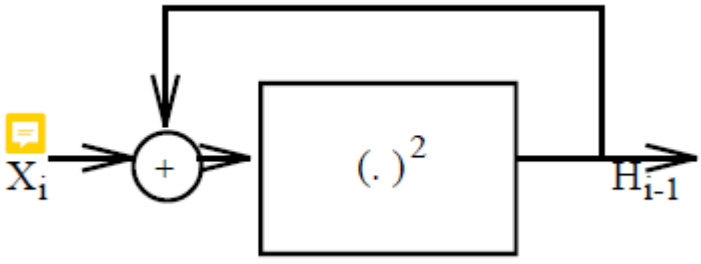
Disadvantages:

- The recipient must have the key
- If the key is known, it is easy to forge messages, that is, it doesn't actually provide authentication.
- For any n-bit block Y and any sequence of n-bit blocks $X_1, X_2, \ldots, X_w$ it is possible to choose a further n-bit block X_{w+1} such that:
  - $X_{w+1} = D_K(Y) \oplus Y_w$

  Where Y_w is the hash value of $X_1, X_2, \ldots, X_w$
- New messages with w + 1 blocks has exactly the same hash value since xor is cancel out.

2. Hash functions based on modular arithmetic

- This scheme is a keyed hash function
- Break the message into fixed size blocks of m-bits. H_0 is a randomly chosen secret initial value (the key).
  - M is prime satisfying M $\geq$ 2^{m-1}
  - + is integer addition
- Then

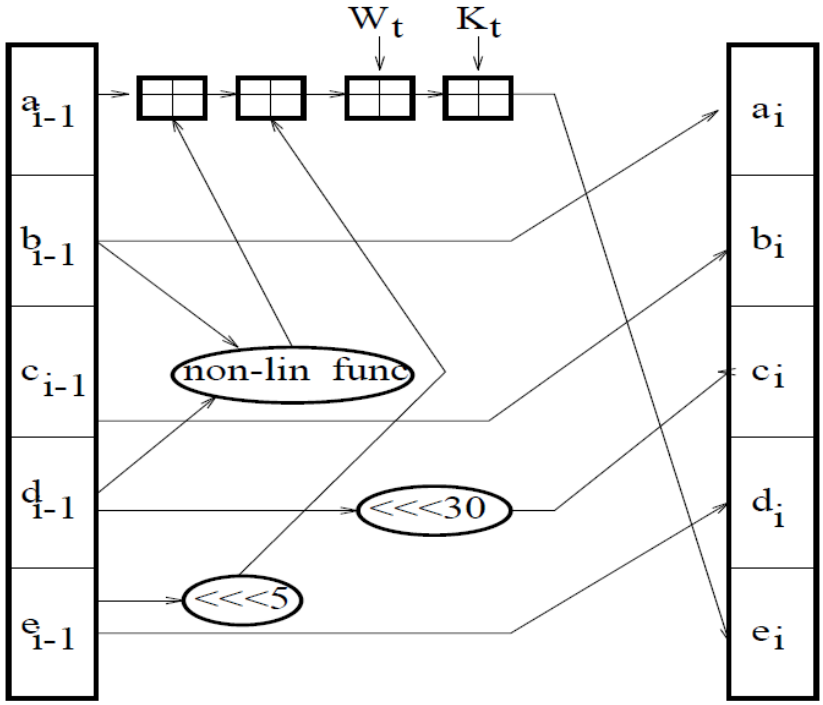


$$H$$
$$(¿¿i-1+X_i)^2 \bmod M$$
$$H_i=¿$$

- H_n is the hash value
- Scheme is broken (Coppersmith)

3. **Designed hash functions**
   - **MD5 (Rivest 1992)**
   - One of the best known algorithms
     - Processes the input as blocks of 512-bit and produces a 128-bit message digest.
     - Uses simple algorithm such as addition modulo 32, and can therefore be coded quite compactly and executed very quickly.

   - **Secure Hash Algorithm (SHA-1)**
   - Hash algorithm proposed and adopted by NIST, for use with the DSA standard.
   - Successor to MD5
   - Produces a 160 bit message digest and uses the design approach used in MD5
   - Plus boxes addition mod 2^{32}
   - Non-linear function and K_t vary for different operations.



- In SHA-1, the message is padded to make 512-bit blocks (also done in MD5)
- Algorithm processes 512 bit blocks in each round
  - Each round has 4 sub rounds and each 4 sub round consists of 20 operations. The nonlinear function used in each round is different.
- $W_t, i=0,1,\ldots.79$ are 32-bit blocks constructed from a 512-bit message blocks.

- K_t is a constant dependent upon the sub-round.

Lecture 8: Key Management

- Diffie-Hellman key exchange
- Man-in-the-middle attack
- Key distribution in PKC
- Digital certificates

Diffie-Hellman Key Exchange

Alice and Bob agree on a common prime p and a common primitive element g of Z_p

Step 1: Secret keys X_A and X_B

- A chooses a random number X_A , 1 < X_A ≤ p
- B chooses a random number X_B , 1 < X_B ≤ p

Step 2: Public keys Y_A and Y_B

- A calculates $Y_A = g^{X_A} mod\ p$
- B calculates $Y_B = g^{X_B} mod\ p$

A → B: Y_A

B → A: Y_B

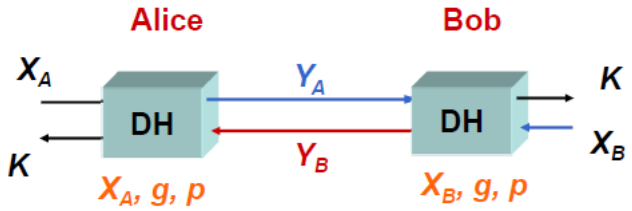
Step 3:

- A calculates $(Y_B)^{X_A}$ mod p
- B calculates $(Y_A)^{X_B}$ mod p
- These factors both equal $g^{X_A X_B} mod\ p$

The security of this system depends on the difficult of computing discrete logarithms.

- In this context, obtaining X_A from Y_A

The protocol



1: A → B: Y_A

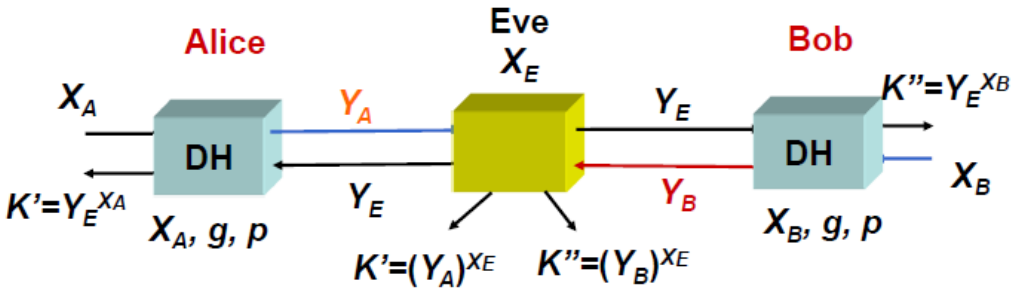
2: B → A: Y_B

- **Example**: **p = 13**, **g = 2**.
  - **A** chooses $X_A = 4$.
  - **B** chooses $X_B = 5$.
  - $Y_A = 2^4 \bmod 13 = 3$.
  - $Y_B = 2^5 \bmod 13 = 6$.
  - $(Y_B)^{X_A}$ **mod p** $= 6^4 \bmod 13 = 9$.
  - $(Y_A)^{X_B}$ **mod p** $= 3^5 \bmod 13 = 9$.

Alice and Bob have established a common key 9.

Diffie-Hellman Key Exchange

- **Man-in-the-Middle Attack**



1: A → E: Y_A

2: E → B: Y_E

3: B → E: Y_B

4: E → A: Y_E

There are various ways of fixing this problem (although we aren't going to look at them here).

Describe the Diffie-Hellman key agreement protocol.

The Diffie-Hellman key agreement protocol is explain using example as follow:

- Adam and Barbie choose a large random prime p and a generator $g \in Zp$.
- The prime p and g are publicly known.
- Privately, Adam chooses integer x, a secret known to Adam, and Barbie chooses y, a secret known to Barbie.
- Next Adam sends $g^x \bmod p$ to Barbie, and Barbie sends $g^y \bmod p$ to Adam.
- Adam and Barbie can now compute the joint key $(g^x)^y = (g^y)^x \bmod p$.

Question:

Explain Diffie-Hellman key exchange. On what hard problem does its security depend? Describe the Diffie-Hellman key agreement protocol. Why is Diffie-Hellman susceptible to man-in-the-middle attacks? Name one way to prevent such attacks.

Answer:

Two parties each create a public-key, private-key pair and communicate the public key to the other party. The keys are designed in such a way that both sides can calculate the same unique secret key based on each side's private key and the other side's public key.
Diffie-Hellman key exchange bases on "discrete logarithm problem" to achieve its security.

The key agreement protocol is explained using example as follow:
Adam and Barbie choose a large random prime *p* and a generator □ ∈ Z_p .
The prime *p* and *g* are publicly known.
Privately, Adam chooses integer x, a secret known to Adam, and Barbie chooses y, a secret known to Barbie.
Next Adam sends g^x □□□□ to Barbie, and Barbie sends g^y □□□ □ to Adam.
Adam and Barbie can now compute the joint key $(g^x)^y = (g^y)^x \bmod p$

Diffie-Hellman key exchange protocol susceptible to man-in-the-middle attacks because any party could generate a Diffie-Hellman key exchange message as there is no identifying secret in the protocol. One-way to prevent man-in-the-middle attack, both parties can add authentication by signing the join key.

Certificates

- Useful technique that provide a partial solution to the authenticity and integrity problems.
- This technique assumes:
  - A central (trusted) authority T
  - A secure communication channel between each user and T, e.g. each user knows Z_T, the public key of T
- Operations:
  - Alice securely sends Z_A to T
  - Alice receives a certificate, C_A, signed by T that binds Z_A to Alice.
  - This certificate is verifiable by everyone who has the public key T.
- A certificate has the following form:
  - M = [$Z_A, Alice's ID, validity\ period$ ¿
  - $C_A = D_{Z_T}(M)$

Encryption with a certificate

When Alice wants to send an encrypted message to Bob:

1. She obtains Bob's certificate
2. Verifies the signature of T on the certificate
3. Extracts Z_B and uses it to encrypt the message.

A certificate may be invalidated before its expiry date if the key is known to be compromised.

The central authority → **periodically issue a list of invalidated certificates**

Key distribution in a certificate-based PKC

C_A and C_B are certificates issued by T. Alice may obtain Bob's certificate by:

- Asking Bob directly
- Looking into a public directory

In above both cases → potential integrity problem arises

- The certificate obtained may have recently been invalidated.

One solution is to obtain Bob's certificate through T.

- Advantage: the **key** Z_B **is authentic**
- Disadvantage: T is a bottleneck, concentration of trust, etc.

Alternatively, one can implement a **security policy** which advises person to **check for up-to-date invalidations** like a list of the trusted authorities website before using the key.

PKC for symmetric key distribution

Major application of PKC → distribution of symmetric keys.

PKC can be used to establish the common information (secret keys, initializing vectors, etc) required by a symmetric cryptosystem.

PKC can be used to send messages securely and authentically. The information shared for the symmetric cryptosystem→ a particular message.

Lecture 9: Secret sharing and its applications

- Threshold secret sharing
- Sharmir's secret sharing scheme
- Homomorphic property

Threshold secret sharing

- Consider a six digit combination lock.
  - The combination can be shared among 4 people.
  - Any three can calculate the combination.
  - No two people can calculate the combination.

Person	c_1	c_2	c_3	c_4	c_5	c_6
One	1	1	1	0	0	0
Two	0	0	1	1	1	0
Three	1	0	0	0	1	1
Four	0	1	0	1	0	1

Each c_i appears twice. As long so no more than one person is missing, somebody present knows c_i.

This is a *threshold secret sharing scheme.*

Shamir's Secret Sharing (1979)

- A threshold scheme → using polynomial interpolation
- An honest dealer D distributes a secret s among n users, such that at least t users must collaborate to find the secret.
  - Less than t players cannot have any information about the secret.

Lagrange interpolation

- Suppose you have *n* pairs $(x_i, y_i = f(x_i))$ and want to find the polynomial *f*.
- The polynomial of degree n-1 through the data is given by Lagrange interpolation.

$$f(x) = \sum_{j=1}^{n} f_j(x) \qquad f_j(x) = y_j \prod_{k=1, k\neq j}^{n} \frac{(x - x_k)}{(x_j - x_k)}$$

*** REFER TO TUTORIAL SLIDES ***

Homomorphic property

- f(1), f(2), f(n) are shares of polynomial f(x)
- g(1), g(2).. g(n) are shares of polynomial g(x)

Then f(1) + g(1), f(2) + g(2), … f(n) + g(n) are shares of f(x) + g(x)

- That is the secret f(0) + g(0)

To multiply a secret by a constant, each share holder has to multiply by the same constant.

Example

- Sharing s=5 among 7 people such that any three can find the secret
- $f(x) = 5+2x+3x^2 \mod 11$

$f(1)=10, f(2)= 10, f(3)=5, f(4)=6, f(5)=2, f(6)=9, f(7)=1$

- Sharing s=7 among the same people
- $g(x)= 7+x+x^2 \mod 11$

$g(1)=9, g(2)= 2, g(3)=8, g(4)=5, g(5)=4, g(6)=5, g(7)=8$

- Shares of s=1 for the same people
- $1 (=5+7 \mod 11)$
- $u(x) = f(x)+g(x)= 1+3x+4x^2 \mod 11$

$u(1)=8, u(2)= 1$ ..

A *perfect* threshold scheme is a threshold scheme in which knowing only t − 1 or fewer shares provide no advantage (no information about S whatsoever, in the information-theoretic sense) to an opponent over knowing no pieces.