2 Cryptography : Key Management, Distribution and User Authentication

Syllabus

At the end of this unit, you should be able to understand and comprehend the following syllabus topics :

- Block cipher modes of operation
- Data Encryption Standard
- Advanced Encryption Standard (AES)
- RC5 algorithm
- Public key cryptography
  - RSA algorithm
- Hashing Techniques
  - SHA256
  - SHA-512
  - HMAC and CMAC
- Digital Signature
  - Digital Certificate
  - PKI X.509
  - Digital Signature Schemes
    - RSA
    - DSS
- Remote user Authentication Protocols
  - Kerberos

2.1 Methods of Encryption

As you know, encryption is primarily driven by two components : Keys and Algorithms. Based on the number of keys used in the encryption and decryption process, the encryption methods can be classified as shown in Fig. 2.1.1.

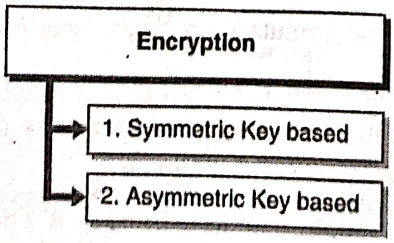


Fig. 2.1.1 : Methods of Encryption

2.1.1 Symmetric Key Encryption

Symmetric means same,

Definition : In Symmetric Key Encryption, the key used for encryption is same as the key used for decryption.

- The keys are identical. The sender as well as the receiver must have exactly the same key to encrypt or decrypt. As an example, symmetric key is like your regular lock key. The same key can be used for locking the door as well as unlocking the door.
- As you understand, a symmetric key is unique between a sender and a receiver. If there are more entities involved and each require to secretly communicate with the another, you end up having multiple keys.
- Let's take an example, suppose there are 4 friends : A, B, C and D and each one of them require communicating with one another secretly. It is obvious that you cannot share the keys between a pair of friends with another pair of friends. So, if A and B share a key, B and C cannot share the same key because C would also know the secret key between A and B and can then decrypt communication between A and B. So, you would require several keys to ensure that each pair of sender and receiver have a unique key. So, how many keys would you need?

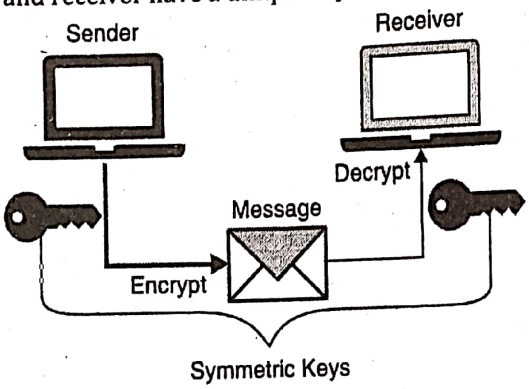


Fig. 2.1.2 : Symmetric Key Encryption

You would need below keys (one for each pair of sender and receiver)

1. A <> B
2. A <> C
3. A <> D
4. B <> C
5. B <> D
6. C <> D

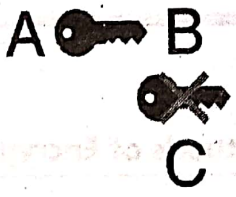


Fig. 2.1.3

- This can be effectively calculated using the formula $K = N \frac{(N-1)}{2}$ where 'N' is the number of entities that need to secretly communicate. So, in the above example, K = 4 * (4 – 1) /2 = 6. If we have 100 entities, it would require 100 * (100 – 1) /2 = 4,950 keys! What if the entire world wants to communicate with each other. Can you imagine? I will come back to it later on and answer that for you.
- Another problem here is how does 'A' send the key she is using to 'B' ? If the sender and receiver have to use the same key, there should be a way to securely transfer the key. Isn't it?

- **For example :** If you have to give your house keys to your friend, you probably exchange hands in person. You don't leave the key somewhere that could potentially be picked / looked by someone else other than your friend. I will answer this question as well later on.
- Some of the examples of symmetric key algorithm are DES, AES and Blowfish.

Advantages of Symmetric Keys

1. Computationally faster than the asymmetric keys
2. Hard to break if the key used is long

Disadvantages of Symmetric Keys

1. Requires a secure mechanism to exchange keys
2. Each pair of sender and receiver require a unique key
3. Provides only confidentiality but not authenticity and non-repudiation

2.1.2 Asymmetric Key Encryption

Unlike symmetric keys,

✍ ***Definition :*** *In Asymmetric Key Encryption, there are two keys that are mathematically related. If one is used for encryption, then only the corresponding other key can be used for decryption.*

Asymmetric means not equal. In the asymmetric system, two mathematically related keys work as key pair. If you use one key for encryption, then you need the other key in the pair for decryption. The encrypting key cannot be used for decrypting.

Asymmetric Keys based Cryptography is also called as **Public Key Cryptography.**

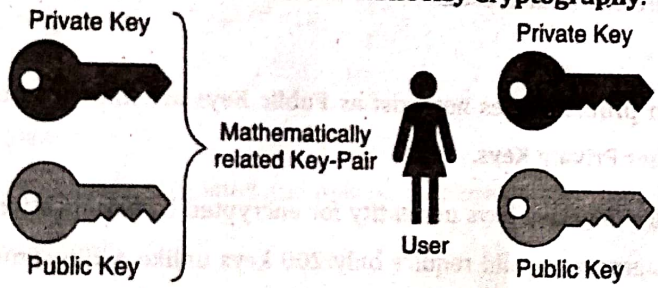


Fig. 2.1.4 : Asymmetric Key Encryption

Let's call one of the keys as the Public Key and its counterpart in the pair as the Private Key. A user owns both the keys. The public key is known to the world while the private key is known only to the user.

Table 2.1.1

Key Used	Corresponding Key Required	Security Service Provided
Encryption – Public Key	Decryption – Private Key	Confidentiality
Private Key	Public Key	Authentication and Non-repudiation

Let us understand these two use cases of the asymmetric keys.

Use Case 1 : User A wants to send a secret message to User B

- **User A knows :** User A's Public Key, User A's Private Key, User B's Public Key

- **User B knows :** User A's Public Key, User B's Public Key, User B's Private Key

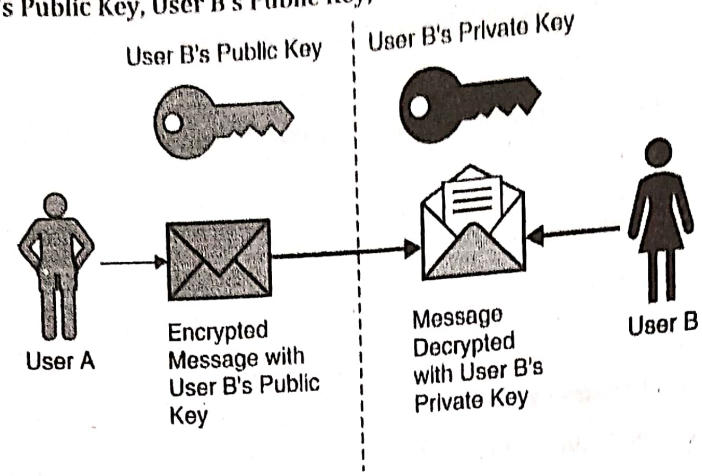


Fig. 2.1.5 : Encryption and Decryption using Public Key Cryptography

- User A wants to send an encrypted message such that only User B can read it. User A encrypts the message with User B's Public Key.
- Now, because User A used User B's Public Key, she is sure that only User B can decrypt the message as decryption would require the corresponding Private Key and only User B knows about her Private Key.
- Hence, in this scenario you find that asymmetric keys as well can be used to send encrypted messages as you saw in the case of symmetric keys.
- One core difference to note here is that User. A need not know User B's Private Key to send her an encrypted message.
- A separate key distribution problem does not exist as Public Keys are known to the world and only the user needs to know and protect her Private Keys.
- You also see that you require only two keys per entity for encrypted communication. So, for 100 people to send each other encrypted messages, we would require only 200 keys unlike 4,095 symmetric keys. So, asymmetric keys help you to overcome two of the limitations of symmetric keys :
  1. Key distribution and
  2. Number of keys to manage
- Hence, I answer the question for you that I asked in the previous section – how do we manage keys if the whole world wants to communicate with each other? The answer is using asymmetric keys!

Use Case 2 : Proving authenticity and non-repudiation

- The second use case of asymmetric keys is for proving authenticity of an entity. This is highly used today for server validation. Have you seen "https" in front of website address? That is one of the examples of this use case.
- Suppose, you want to do an online transaction. How do you ensure that the bank's website address you are interacting with is the right one ? That's precisely what asymmetric keys help you solve as well.

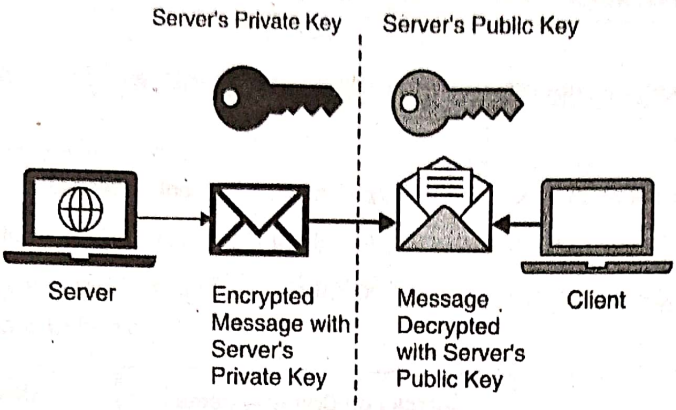


Fig. 2.1.6 : Non-repudiation using public key cryptography

- Client wants to authenticate the server before it begins the transaction. It sends the server a plaintext message and asks it to encrypt it with its Private Key. The server encrypts the message that the client sent with its Private Key and sends it back to the client.
- Client uses the world-known Public Key of the server and decrypts the message it received from the server. If the message gets successfully decrypted and it matches with what the client sent earlier to the server to encrypt, the client has now validated that it is indeed interacting with the authentic server because except the authentic server, no one else would have known server's Private Key. The client is satisfied, and it begins the secure transaction after having established the server's authenticity.
- Hence, you find that asymmetric keys could effectively be used for authentication and non-repudiation. Some examples of algorithms that use asymmetric keys are RSA, ECC, Diffie-Hellman, and El Gamal.

Advantages of Asymmetric Keys

1. Easy key distribution
2. Less number of keys to manage
3. Can also be used for providing authentication and non-repudiation

Disadvantages of Asymmetric Keys

1. Slower than symmetric keys
2. Requires significant CPU power due to complex mathematical relation between the keys

2.1.3 Comparison between Symmetric and Asymmetric Keys

Sr. No.	Comparison Attribute	Symmetric Keys	Asymmetric Keys
1.	Speed	High	Low
2.	Complexity	Low	High
3.	Number of keys	High	Low
4.	Key Distribution	Problematic	Easier
5.	Security Services	Confidentiality	Confidentiality, Authenticity, Non-repudiation

2.2 Cryptanalysis (Attacks on Cryptosystems)

Now that you have a general understanding of the cryptosystems, let's learn some of the possible attacks on them.

Note : The attacks described here are common for any cryptographic algorithm be it DES, AES, RSA or any other. While for some algorithms it is comparatively easier and for others it is theoretically possible. Any specific attack against an algorithm is described in its respective section. Otherwise, you could mention and elaborate on the following attacks.

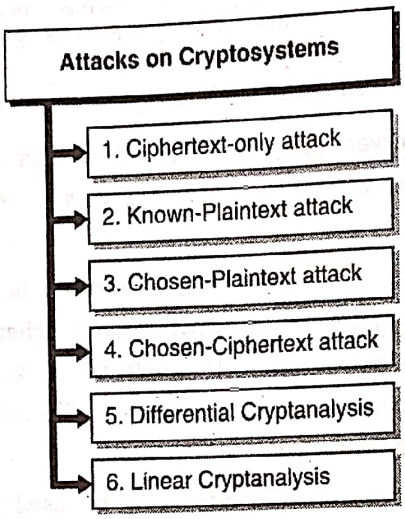


Fig. 2.2.1 : Attacks on Cryptosystems

1. Ciphertext-only attack

In this type of attack, the attacker has ciphertext of several messages. The algorithm is known, and the goal of the attack is to find out the key used in encryption. Once the key is found out, it is possible to decrypt messages that were encrypted using the key.

2. Known-Plaintext attack

The attacker knows the plaintext partially and the corresponding ciphertext. The goal is to find out the key. Once the key is known, the attacker can then use the key to decrypt ciphertext for which she does not know the plaintext.

For example : You might be using a fixed greeting in your messages (for example, "Dear Friend") or you might be sending same message (for example, "Good morning") everyday to someone. The attacker could know this and also the corresponding ciphertext and try to find out the key.

3. Chosen-Plaintext attack

- The attacker knows the exact plaintext and the corresponding ciphertext. The goal is to find out the key. Once the key is known, the attacker can then use the key to decrypt ciphertext for which she does not know the plaintext.

For example : The attacker can send you a message that she knows that you will definitely forward to your friends. While forwarding, you might encrypt the message using your key.

Now, the attacker can grab the ciphertext that you sent, and she already knows the plaintext message she sent you earlier.

4. **Chosen-Ciphertext attack**

In this attack, the attacker choses the ciphertext that she wants to be decrypted and know the corresponding plaintext. The goal again is to find out the key.

5. **Differential Cryptanalysis**

- In this attack, the attacker choses a pair of plaintexts and follows through each stage in their respective encryption process and compare the difference between the results at each stage.
- The key used in encrypting the pair is same.
- The goal again is to figure out the key by carefully studying the differences in results at various stages between the pair. Since, the attacker chooses the plaintexts, differential analysis is considered to be a type of chosen-plaintext attack.

6. **Linear Cryptanalysis**

The attacker carries out a known-plaintext attack and tries to figure out the key. She evaluates the input and output at various stages of the encryption process and tries to find out the probability of specific key values.

2.2.1 Comparison between Differential and Linear Cryptanalysis

Sr. No.	Comparison Attribute	Differential Cryptanalysis	Linear Cryptanalysis
1.	Plaintext selection	Carefully chosen	Any random plaintext
2.	Plaintext used	In pairs	One by one
3.	Complexity of attack	High	Low
4.	Mathematical relation between plaintexts used	Specific differences (such as XOR)	Linear approximation (such as a series of XOR operations)
5.	Goal of the attack	Identify some bits of the unknown key	Identify the linear relation between some bits of the plaintext, some bits of the cipher text and some bits of the unknown key

2.3 Concept Building - Types of Symmetric Algorithms (Ciphers)

Q. What are Block Cipher Modes. Describe any two in detail. **MU - Dec. 18, 10 Marks**

Symmetric key based algorithms (ciphers) can work either on blocks of bits (characters) or one bit at a time.

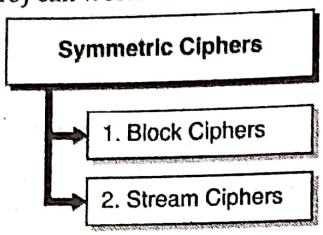


Fig. 2.3.1 : Symmetric Ciphers

Definition : The algorithms that work on blocks are called block ciphers.

Definition : The algorithms that work on one-bit at a time are called stream ciphers.

2.3.1 Block Ciphers

In block ciphers, the information that needs to be encrypted is broken into smaller and equal block sizes. Then, the encryption operation (substitution and transposition) is applied to each block. The resultant ciphertext from each block is then combined to produce the encrypted information.

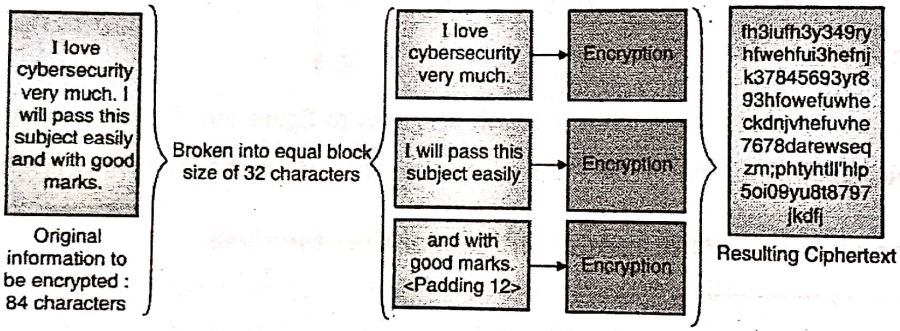


Fig. 2.3.2 : Block Ciphers

Fig. 2.3.2 illustrates simplified block diagram of how a block cipher works. The information is broken into equal size blocks and then the encryption operation is carried out on each block. If the block size has lesser number of characters than required to form a block, then padding is done to fill the block. Padding is just filling some temporary information to form a block. Finally, the resulting encrypted information from each block is combined to get the overall encrypted message.

Data Encryption Standard (DES) and Advanced Encryption Standard(AES) are two of the examples of Symmetric Block Ciphers.

2.3.2 Stream Ciphers

Unlike block ciphers, stream ciphers work on one bit of plaintext at a time. Each bit of plaintext is combined with the bits of security key and then XORed to get ciphertext.

Note: If you recall from your logical design classes, the Table 2.3.1 is the truth table of XOR. For result to be 1, both the inputs should be different.

Table 2.3.1 : XOR Truth Table

Sr. No	Input X	Input Y	Output Z (XOR X, Y)
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

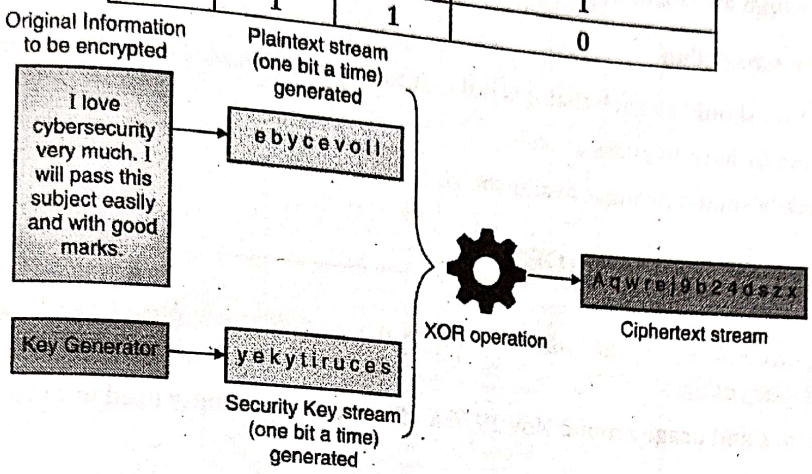


Fig. 2.3.3 : Stream Ciphers

RC4 is an example of stream cipher.

2.3.3 Comparison between Block and Stream Cipher

Sr. No.	Comparison Attribute	Block Cipher	Stream Cipher
1.	Security	High	Low
2.	Speed	Low	High
3.	Application	Non-real time such as documents	Real time data such as Voice
4.	Commonly used	Yes	No

2.4 Block Cipher Principles (for DES and other Ciphers)

There are three critical components in designing a block cipher as shown in Fig. 2.4.1.

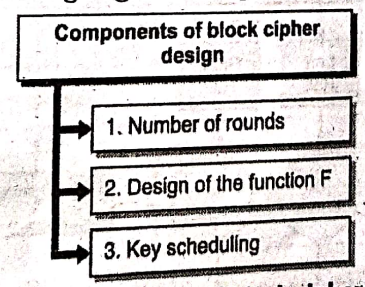


Fig. 2.4.1 : Components of block cipher design

Following are the principles around each of these.

1. **Design Principles for Number of Rounds in the Block Cipher Algorithm**

(i) The greater the number of rounds, the more difficult it is to perform cryptanalysis.

(ii) The number of rounds is chosen such that a known cryptanalysis takes a greater effort compared to brute-force attack.

2. **Design Principles for function F (Feistel network) in the Block Cipher Algorithm**

   (i) It must be difficult to re-assemble the substitution performed by the function F.

   (ii) F is non-linear which means it is difficult to establish any relation between input to F and output from

   (iii) F should have high avalanche effect.

3. **Principles for Key scheduling**

   (i) Subkey selection should be such that it is difficult to work backwards to derive the main key.

   (ii) Subkeys should be hard to guess as well.

   (iii) The key schedule should produce avalanche effect.

2.5 Data Encryption Standard (DES)

Definition : *Data Encryption Standard (DES) is a symmetric key based block cipher standard us encryption and decryption.*

It came into existence and usage around Nov 1976 and was predominantly used in the industry until 2002.

Major attributes of DES

- It is a symmetric key based algorithm
- It works as a block cipher
- It uses 64-bit blocks
- It uses a key size of 64-bits in which 56-bits are the actual keys and 8-keys are used for error detection
- It uses 16 rounds of operation (substitution and transposition) to convert a block of plaintext into ciphertex
- DES is now considered insecure and obsolete due to its short key-size (56-bits only)

2.5.1 Block Diagram and Internals of DES

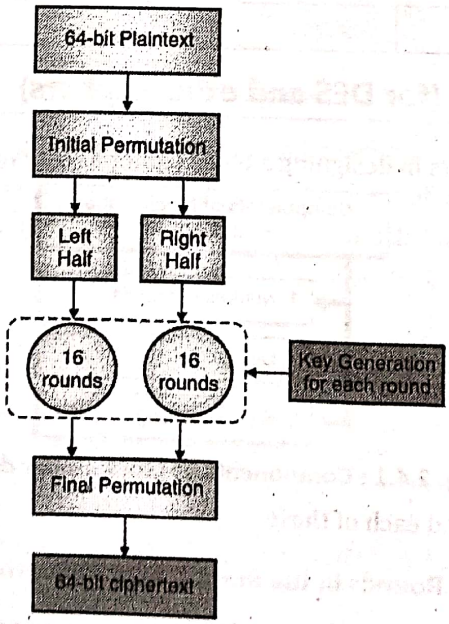


Fig. 2.5.1 : Block diagram of DES

Fig. 2.5.1 illustrates simplistic view of DES. Let us understand what happens at each stage.

Step 1 : Creation of 64-bit blocks

In this step, the plaintext information to be encrypted is broken into 64-bit blocks. DES is a block cipher and block creation is similar to as explained in the earlier section.

Table 2.5.1 : 64 bits of plaintext

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Step 2 : Initial Permutation

In this step, the 64 bits in the plaintext blocks are re-arranged (transposed). This is done as per the diffusion property of the cipher to ensure that any small variance in plaintext produces a large variance in the ciphertext.

Table 2.5.2 : Initial Permutation (re-arrange bits of plaintext)

58	50	42	34	26	18	10	2	← Column 2 becomes 1st row
60	52	44	36	28	20	12	4	← Column 4 becomes 2nd row
62	54	46	38	30	22	14	6	← Column 6 becomes 3rd row
64	56	48	40	32	24	16	8	← Column 8 becomes 4th row
57	49	41	33	25	17	9	1	← Column 1 becomes 5th row
59	51	43	35	27	19	11	3	← Column 3 becomes 6th row
61	52	45	37	29	21	13	5	← Column 5 becomes 7th row
63	55	47	39	31	23	15	7	← Column 7 becomes 8th row

Step 3 : Left Half and Right Half Split

In this step, the bits from the Initial Permutation stage are split into two parts – left half and right half each containing 32-bits. These individual 32-bit blocks are then continuously worked through the 16 rounds of operation.

Step 4 : Subkey Key Generation

For the 16 rounds of operation, a unique subkey is derived for each round from the 56-bit key. The key is derived using complex mathematical functions. Each generated subkey is 48-bit long.

Step 5 : Rounds

Left half and the right half both individually go through 16 rounds of encryption operation. In each of th rounds, the derived subkey is used to produce temporary ciphertext. This temporary ciphertext produce after each round is used in the next round until the final round is complete. Each round consists substitutions and successive permutations.

Step 6 : Final Permutation

In the last stage, we need to bring the bits back to their respective positions. The bit positions were change at the initial permutation stage.

Table 2.5.3 : Final permutation (re-arrange bits of ciphertext)

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Step 7 : Final Ciphertext

Once all the steps are done, you get the final ciphertext for the plaintext given the security key of your choi via DES.

Exam Tip : If you hear that an algorithm is broken or is insecure, it means that it is computationally feasible to find out the k used for encryption. Note that cryptography heavily depends upon our understanding of mathematics and t computation power available today. What is secure and infeasible today, could be insecure and feasible to cra in future.

2.5.2 Block Cipher Modes of Operation (for DES and other Block Ciphers in General)

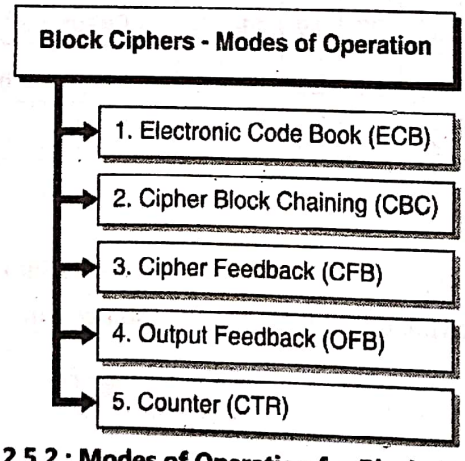


Fig. 2.5.2 : Modes of Operation for Block Ciphers

DES and other block ciphers can potentially work in several modes. Let's review them carefully.

1. Electronic Code Book (ECB) Mode

In this mode, the same key is used to encrypt all the blocks. Key derivatives or subkeys are not used. Additionally, each block is treated separately and the ciphertext of previous block does not influence successive blocks.

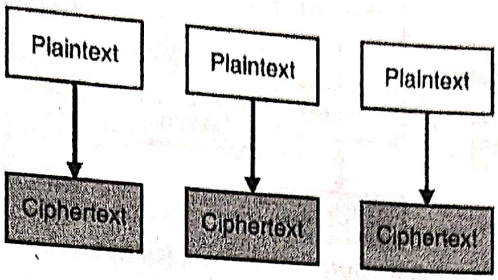


Fig. 2.5.3 : Electronic Code Book (ECB) Mode

2. Cipher Block Chaining (CBC) Mode

In this mode, the ciphertext of previous block is used with the next plaintext block. The two blocks (ciphertext of previous block and plaintext of next block) are XORed and then passed through the encryption operation. This generates a lot more randomness in the final ciphertext.

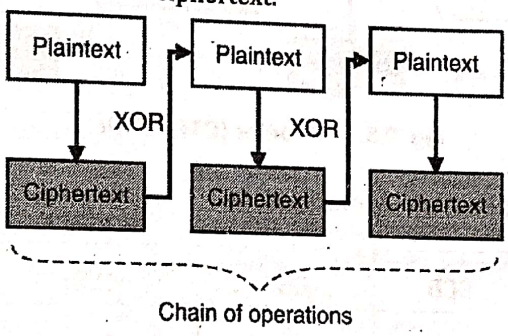


Fig. 2.5.4 : Cipher Block Chaining (CBC) Mode

3. Cipher Feedback (CFB) Mode

In this mode, the block cipher works like a stream cipher. The ciphertext from the previous block is XORed with the key (keystream) for the next block. This way the key increasingly becomes random and brings more randomness in the overall encryption process.

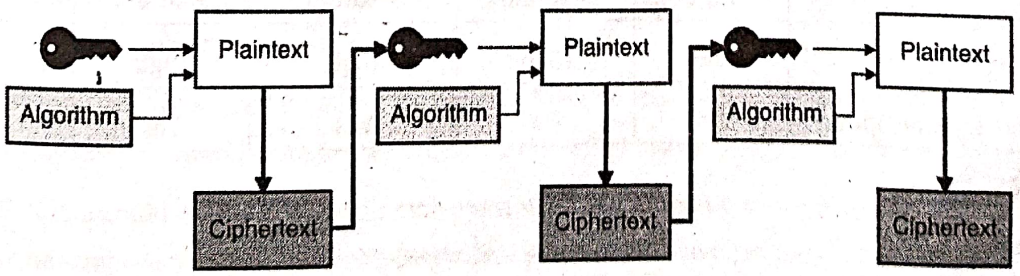


Fig. 2.5.5 : Cipher Feedback (CFB) Mode

4. Output Feedback (OFB) Mode

In this mode, the block cipher works like a stream cipher. The keystream used in the previous block is XOR with the keystream of the next block.

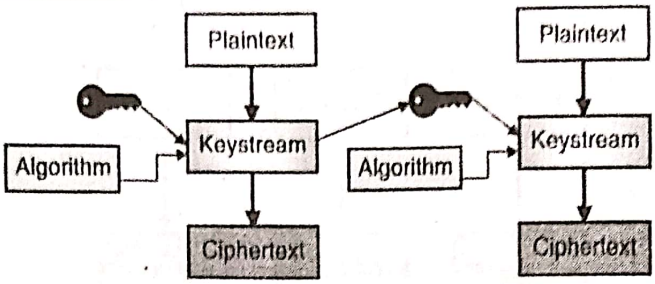


Fig. 2.5.6 : Output Feedback (OFB) Mode

5. Counter (CTR) Mode

In this mode as well, the block cipher works like a stream cipher. The key is converted into keystream (as used stream cipher) and the keystream is XORed with a counter that increases for every block.

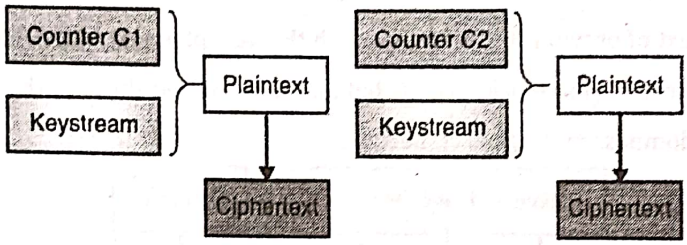


Fig. 2.5.7 : Counter (CTR) Mode

2.5.3 Comparison between Modes of Operation

Sr. No.	Mode	ECB	CBC	CFB	OFB	CTR
1.	In-Parallel block encryption	Yes	No	No	No	Yes
2.	Suited for	Small Information	Any size of information	Small Information	Small Information	Any size of information
3.	Security and randomness	Low	High	High	High	High
4.	Speed	High	Medium	Medium	Medium	High
5.	Complexity	Low	High	High	High	Low
6.	Works like stream cipher?	No	No	Yes	Yes	Yes

Weakness in DES

1. Small key size

56-bits of keys have a keyspace (possible values) of 2^{56}. While that might seem a lot, it is actually not given t compute power we have today. In 1990s, the compute power we had was significantly lower and hence w considered secure at that time.

Definition : The type of attack where each combination is tried in an attempt to find the right combination is also called as brute force attack.

2. **Prone to linear cryptanalysis**

   DES has been proven to be susceptible to linear cryptanalysis.

3. **Prone to differential cryptanalysis**

   DES has been proven to be susceptive to differential cryptanalysis.

2.5.4 Double DES

- In order to strengthen DES, it was considered to increase the key size to 112-bits effectively. The way it was chosen to do so was to use 2 keys of 56-bits each. Let's call them K1 and K2.
- Mathematically, it can be denoted as follows:

  Ciphertext C = Encryption(K2, Encryption(K1, Plaintext P))

  Plaintext P = Decryption(K1, Decryption(K2, Ciphertext C))

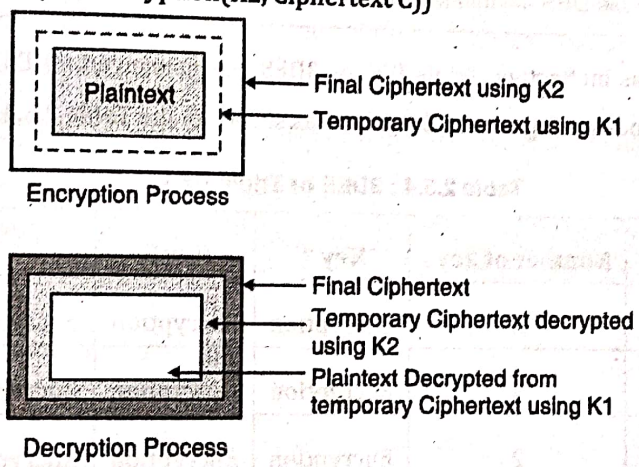


Fig. 2.5.8 : Double DES

  - During the encryption process, first the plaintext is encrypted with Key K1 and then the result is again encrypted with Key K2 to get the final ciphertext for plaintext.
  - During the decryption process, first the Key K2 is used to decrypt to get the ciphertext that Key K1 can decrypt to get the plaintext.

- However, Double DES was proven to be ineffective. Meet in the middle attack was shown to reduce the complexity to just 2^{57} (2^{56} attempts made twice, hence $2x2^{56} = 2^{57}$) instead of 2^{112} as originally thought.
- So, using K1 if you could derive temporary ciphertext using encryption process and using K2 if you could also derive the same temporary ciphertext using decryption process, you have found a match and the keys you chose (K1 and K2) are now known to you. Hence, you could effectively find both the keys and break Double DES without original thought of complexity of 112 bits.

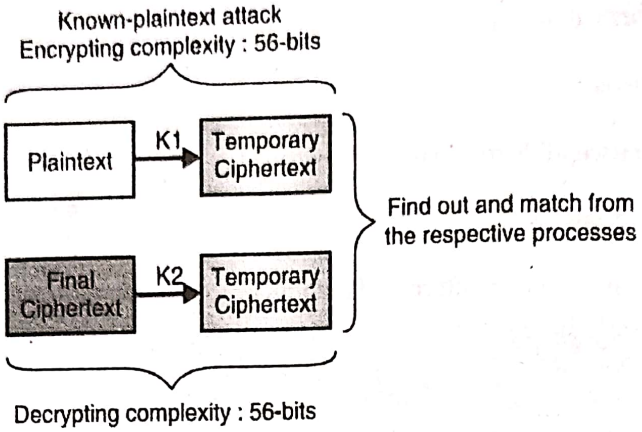


Fig. 2.5.9 : Complexity in Double DES

- Hence, Double DES was not adopted in the industry and is not used.

2.5.5 3DES or Triple DES

Q. What is the drawback of Double DES algorithm ? How is it overcome by Triple DES ? **MU - May 19, 5 Marks**

Finding that Double DES was ineffective, Triple DES or 3DES was conceived. 3DES uses 48 rounds of operation and can work in the following modes using two or three keys as shown in the Table 2.5.4.

Table 2.5.4 : 3DES or Triple DES

Sr. No.	Mode	Number of keys	Key 1	Key 2	Key 3
1.	DES-EEE3	3	Encryption	Encryption	Encryption
2.	DES-EDE3	3	Encryption	Decryption	Encryption
3.	DES-EEE2	2	Encryption	Encryption	Encryption Using Key 1
4.	DES-EDE2	2	Encryption	Decryption	Encryption Using Key 1

You might wonder how decryption helps? Note that if you encrypt a plaintext using a key (say K1) and run the decryption process using a different key (say K2), the text (from encryption process using K1) becomes more random.

The use of a different key in the decryption process from the encryption process brings added randomness and hence helps to make attacks such as linear or differential cryptanalysis extremely hard.

Fig. 2.5.10 : Decryption process

2.6 Advanced Encryption Standard (AES)

Definition : Advanced Encryption Standard (AES) is a symmetric key based block cipher standard used for encryption and decryption.

The standard became effective on May 26, 2002 and is predominantly used in the industry today due to its strong cipher properties. AES replaced DES as the new standard when DES was found to be insecure and vulnerable to various attacks.

Evaluation Criteria for AES

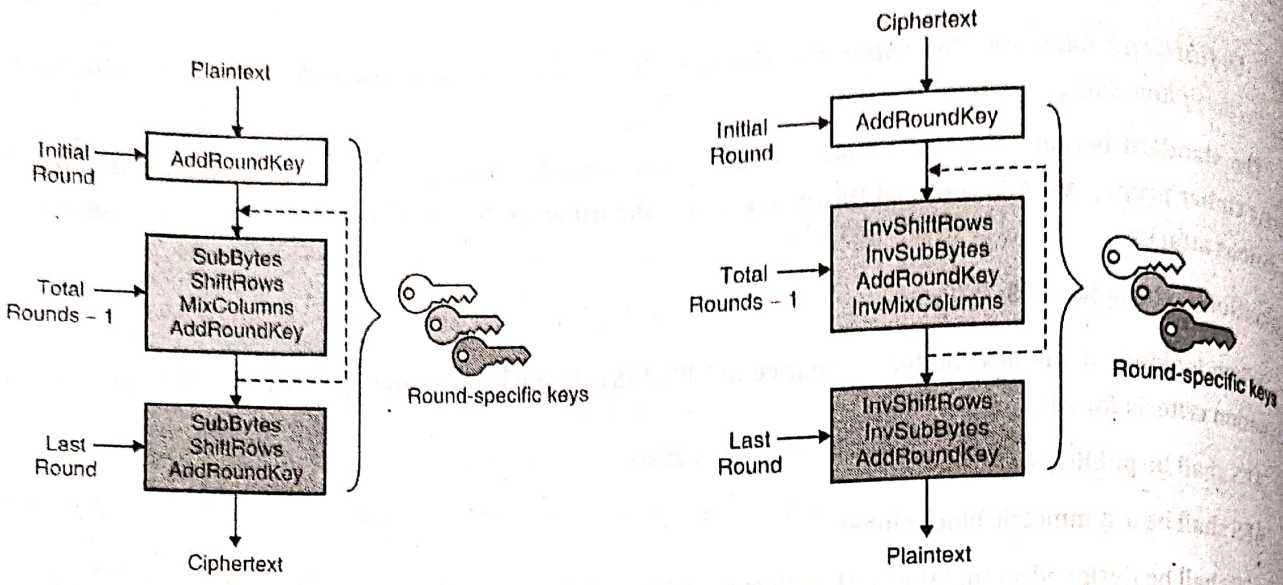
When looking for the next better alternative to DES, NIST defined the following acceptability requirements and evaluation criteria for AES.

1. AES shall be publicly defined.
2. AES shall be a symmetric block cipher.
3. AES shall be designed so that the key length may be increased as needed.
4. AES shall be implementable in both hardware and software.
5. AES shall either be a) freely available or b) available under terms consistent with the American National Standards Institute (ANSI) patent policy.
6. Algorithms which meet the above requirements will be judged based on the following factors
   a. security (i.e., the effort required to cryptanalyse)
   b. computational efficiency
   c. memory requirements
   d. hardware and software suitability
   e. simplicity
   f. flexibility
   g. licensing requirements

Major attributes of AES

- It is a symmetric key based algorithm
- It works as a block cipher
- It uses 128-bit blocks
- It can work with key sizes of 128, 192 and 256 bits
- Number of rounds of operation depends upon the key size
  - 128-bit key undergoes 10 rounds
  - 192-bit key undergoes 12 rounds
  - 256-bit key undergoes 14 rounds
- AES is considered highly secure due to its long key sizes and is used in the industry today

2.6.1 Block Diagram and Internals of AES



(a) AES Encryption Process (b) AES Decryption Process

Fig. 2.6.1 : Block diagram of AES

Let's understand the blocks.

1. **AddRoundKey**

   In this transformation step, a round key is generated and XORed with the intermediate (temporary) ciphertext. This block is used in both encryption as well as decryption process.

2. **SubBytes**

   In this transformation step, the intermediate ciphertext undergoes various substitution operations. It is used for encryption process.

3. **ShiftRows**

   In this transformation step, the intermediate ciphertext undergoes various row-wise transposition operations. It is used for encryption process.

4. **MixColumns**

   In this transformation step, the intermediate ciphertext undergoes various column-wise transposition operations.It is used for encryption process.

5. **InvSubBytes**

   This is inverse of SubBytes operation. It is used in the decryption process.

6. **InvShiftRows**

   This is inverse of ShiftRows operation. It is used in the decryption process.

7. **InvMixColumns**

   This is inverse of MixColumns operation. It is used in the decryption process.

2.6.2 Comparison between DES and AES

Q. Compare and contrast DES and AES. MU - May 19, 10 Marks

Sr. No.	Comparison Attribute	DES	AES
1.	Cryptographic Strength	Low	High
2.	Key Size	56-bit	128, 192 and 256 bits
3.	Block Size	64-bit	128-bit
4.	Rounds	16	10, 12, 14 - based on key size
5.	Usage	Obsolete - Not used	Currently used industry standard

2.7 RC5 Algorithm

Definition : RC5 is a symmetric key based block cipher standard used for encryption and decryption.

It was designed by Ronald Rivest in 1994. RC stands for "Rivest Cipher", or alternatively, "Ron's Code".

2.7.1 Major Attributes of RC5

- It is a symmetric key based algorithm
- It works as a block cipher
- It uses a variable block size – 32, 64 or 128
- It uses a variable length key size ranging from 0 to 2040 bits
- It uses variable number of rounds of operation – 0 to 255

2.7.2 Internals of RC5 Algorithm

RC5 is word-oriented. All of the basic computational operations have *w*-bit words as inputs and outputs. RC5 is a block-cipher with a two-word input (plaintext) block size and a two-word output (ciphertext) block size. The nominal choice for *w* is 32-bits for which RC5 has 64-bit plaintext and ciphertext block sizes.

The number *r* of rounds is the second parameter of RC5. Choosing a larger number of rounds provides an increased level of security. RC5 uses an "expanded key table" *S* that is derived from the user's supplied secret key. The size *t* of table *S* also depends on the number *r* of rounds. *S* has $t = 2(r+1)$ words.

2.7.3 Key Expansion

Key expansion consists of 3 steps:

1. Converting the Secret Key from Bytes to Words
2. Initializing the Array *S*
3. Mixing in the Secret Key

2.7.4 Encryption

Assume that the input block is given in two *w*-bit registers *A* and *B*. Assume that the key expansion step is already performed and thus the array S is already populated. The encryption algorithm can be outlined in the following pseudo code.

```
A = A + S[0];
B = B + S[1];
for i = 1 to r do
A = ((A xor B) <<< B) + S[2 * i];
B = ((B xor A) <<< A) + S[2 * i + 1];
```

The output is in the registers *A* and *B*.

2.8 Concept Building - Mathematics Behind Cryptography

Definition : *The modular arithmetic deals with operations on integers specifically around remainders from division.*

Let's take a few examples.

Dividend	Divisor	Quotient	Remainder (Modulus)
15	5	3	0
15	4	3	3
15	3	5	0
15	2	7	1
15	1	15	0

So, for example, number 15 when divided by 2, gives you Quotient of 7 and Remainder of 1. This can be mathematically written as 15 mod 2 = 1.

Note : Before you proceed, try finding out a few mods (remainders from division) for numbers of your choice.

1. Congruence Property

Two numbers are said to be in congruence modulo, if they give out the same mod.

For example :

15 mod 2=1

17 mod 2=1

Hence, 15 is congruent to 17 modulo 2 i.e. 15 and 17 when undergo mod operation with 2, they both give same result of 1. They can be mathematically denoted as

$15 \equiv 17 \pmod 2$

2. Modular Addition

$(A + B) \bmod C = (A \bmod C + B \bmod C) \bmod C$

Let A = 12, B = 15 and C = 5

Left Hand Side	Right Hand Side
= (12 + 15) mod 5 = 27 mod 5 = 2	= (12 mod 5 + 15 mod 5) mod 5 = (2 + 0) mod 5 = 2 mod 5 = 2

3. **Modular operation on Negative numbers**

If you come across modular operation on a negative number, make the number positive by repetitively adding mod until it becomes positive.

For example : If you have to find – 13 mod 5, keeping adding 5 to – 13 until you get a positive number. So,

$$-13 + 5 = -8 + 5 = -3 + 5 = 2$$

Now, do mod on the positive number you got. In this case, – 13 mod 5 becomes 2 mod 5. Hence, the answer is 2.

4. **Modular Subtraction**

(A – B) mod C = (A mod C – B mod C) mod C

Let A = 12, B = 15 and C = 5

Left Hand Side	Right Hand Side
= (12 – 15) mod 5 = –3 mod 5 = 2	= (12 mod 5 – 15 mod 5) mod 5 = (2 – 0) mod 5 = 2 mod 5 = 2

5. **Modular Multiplication**

(A * B) mod C = (A mod C * B mod C) mod C

Let A = 12, B = 15 and C = 5

Left Hand Side	Right Hand Side
= (12 * 15) mod 5 = 180 mod 5 = 0	= (12 mod 5 * 15 mod 5) mod 5 = (2 *0) mod 5 = 0 mod 5 = 0

6. **Modular Inverse**

Modular arithmetic does not have division operation. However, it has inverse. Inverse of a general number is 1 divided by that number.

For example: Inverse of 2 is $\frac{1}{2}$. In other words, a number when multiplied by its inverse would give 1. So, 2 multiplied by its inverse $\frac{1}{2}$ would give $2 * \frac{1}{2} = 1$.

So, modular inverse of A mod C is the value of B such that when A is multiplied by B and the mod C operation is carried out, it gives 1. Mathematically, it can be written as

$$A * B \equiv 1 \pmod{C}$$

Note : To understand the above, refer to the congruency equation. mod C operation on A*B should be same as mod C operation on 1.

Let's take an example.

Suppose you have to find modular inverse of 12 mod 5. Here, A = 12 and C = 5. Let's assume value of B from 0 onwards until we find (A*B) mod C = 1.

Table 2.8.1 : Modular inverse

Value of B	Operation	Result
0	(12 * 0) mod 5	0
1	(12 * 1) mod 5	2
2	(12 * 2) mod 5	4
3	(12 * 3) mod 5	1

Hence, modular inverse of 12 mod 5 is 3.

7. **Prime Numbers**

- These are whole numbers which are greater than 1 and are only divisible by 1 and itself.

  For example : 2, 3, 5, 7, 11 and various others.

8. **Coprime Numbers**

- Two integers a and b are said to be relatively prime, mutually prime, or coprime (also written co-prime) if the only positive integer (factor) that divides both of them is 1.

  For example : 5 and 7 are coprime because their common factor is only 1 whereas 14 and 18 are not coprime because their common factor can also be 2.

2.9 Greatest Common Divisor (GCD)

- Greatest Common Divisor (GCD) of two positive integers is the largest integer that can fully divide both the integers.

  For example : GCD (5, 10) is 5. GCD of (11, 13) is 1.

- GCD is usually found out by finding the factors of the respective integers and then choosing the common highest factor.

  For example : To find GCD (24, 70)

  Factors of 24 : 2 * 2 * 2 * 3 → (Factors could be 2, 3, 4, 6, 8, 12, 24)

  Factors of 70 : 2 * 5 * 7 → (Factors are only 2, 5, 7)

  Hence, the largest common factor is 2. Hence, GCD (24, 70) = 2.

2.9.1 Euclid's or Euclidean Algorithm

Finding GCD for smaller numbers is quite straight forward. But, when it comes to finding GCD of large numbers, it might be a complex task. This is precisely where Euclid's (or Euclidean) algorithm helps.

Euclid's algorithm states that

gcd(a, b) = gcd(a mod b, b) if a > b

gcd(a, b) = gcd(a, b mod a) if b > a

2.9.2 Solved Examples

Ex. 2.9.1 : Find gcd(50, 65) using Euclidean algorithm.

Soln. :

gcd(50, 65) = gcd(50, 65 mod 50) [Because 65 is greater than 50] = gcd(50, 15)

= gcd(50 mod 15, 15) [Because 50 is greater than 15] = gcd(5, 15)

= gcd(5, 15 mod 5) [Because 15 is greater than 5]

= gcd(5, 0) [Stop here once the mod of a term becomes 0]

gcd(50, 65) = 5 [Is the gcd(50, 65)]

Ex. 2.9.2 : Find gcd(464, 238) using Euclidean algorithm.

Soln. :

gcd(464, 238) = gcd(464 mod 238, 238)

= gcd(226, 238)

= gcd(226238 mod 226)

= gcd(226, 12)

= gcd(226 mod 12, 12)

= gcd(10, 12)

= gcd(10, 12 mod 10)

= gcd(10, 2)

= gcd(10 mod 2, 2)

= gcd(0, 2)

gcd(464, 238) = 2

Ex. 2.9.3 : Find gcd(105, 80).

Soln. :

gcd(105, 80) = gcd(105 mod 80, 80)

= gcd(25, 80 mod 25)

= gcd(25 mod 5, 5)

= gcd(0, 5)

gcd(105, 80) = 5

2.9.3 Extended Euclidean Algorithm

The Extended Euclidean Algorithm can be used to find the gcd of two numbers, and also to simultaneously express the gcd as a linear combination of these numbers. It helps to find values of coefficients x and y such that to satisfy the following equation : $ax + by = gcd(a, b)$

In the extended algorithm, the computation involves several entities. First, let's define them:

- **i(index)** : This would just be used to iterate (repeat) the process.

Note : If b is greater than a, then assume a = b and b = a. Swap the values to avoid negatives

- **r (remainder)** : temporary placeholder variable for a and b

  r would be calculated as $r_{i+1} = r_{i-1} - q_i r_i$
- **q** would be calculated as $q_{i+1} = r_{i-1} / r_i$
- **s** : temporary placeholder for values while deriving coefficient x

  s would be calculated as $s_{i+1} = s_{i-1} - q_i s_i$
- **t** : temporary placeholder for values while deriving coefficient y

  t would be calculated as $t_{i+1} = t_{i-1} - q_i t_i$
- **x** would be

$$s_{i+1} = \frac{b}{\gcd(a,b)}$$

- **y** would be

$$t_{i+1} = \frac{a}{\gcd(a,b)}$$

Ex. 2.9.4 : For a = 161 and b = 42, calculate gcd(a, b) and also the values of x and y to satisfy the extended Euclidean algorithm.

Soln. :

$r_0 = 161$ and $r_1 = 42$

i starts with 0

- First draw the initial table.

Index i	quotient q for i	Remainder r for i	S for i	t for i
0		161	1	0
1		42	0	1

- Starting steps :

Index i	quotient q for i	Remainder r for i	S for i	t for i
0		161	1	0
1		42	0	1
2	161 / 42= 3	161 – 3 * 42= 35	1 – 3 * 0 = 1	0 – 3 * 1= –3

- For the highlighted row,

i = 1 (we are just doing the calculation for 2nd row, hence the value of i is still 1)

q_1 can be written as q_i where i = 1

So, $q_1 = r_{i-1} / r_i = r_0 / r_1 = 161 / 42 = 3$

r_2 can be written as r_{i+1} where i = 1

So, $r_2 = r_{i-1} - q_i r_i$ which means $r_2 = r_0 - q_1 * r_1$

$r_2 = 161 - 3 * 42$

$r_{i-1} = r_0$ which is 161

$r_i = r_1$ which is 42

- Similarly, s_2 can be written as s_{i+1} where i = 1

So, $s_2 = s_{i-1} - q_i s_i = s_0 - q_1 s_1 = 1 - 3 * 0 = 1$

- Similarly, t_2 can be written as t_{i+1} where i = 1

So, $t_2 = t_{i-1} - q_i t_i = t_0 - q_1 t_1 = 0 - 3 * 1 = -3$

Similarly proceed to the next step.

Index i	quotient q for i	Remainder r for i	S for i	t for i
0		161	1	0
1		42	0	1
2	161 / 42 = 3	161 – 3 * 42 = 35	1 – 3 * 0 = 1	0 – 3 * 1 = –3
3	42 / 35 = 1	42 – 1 * 35 = 7	0 – 1 * 1 = –1	1 – 1 * –3 = 4

- Here again :

$q_i = q_2$, where i = 2

So, $q_2 = r_{i-1} / r_i = r_1 / r_2 = 42 / 35 = 1$

$r_3 = r_1 - q_2 * r_2 = 42 - 1 * 35 = 7$

$s_3 = s_1 - q_2 * s_2 = 0 - 1 * 1 = -1$

$t_3 = t_1 - q_2 * t_2 = 1 - 1 * -3 = 4$

Index i	quotient q for i	Remainder r for i	S for i	t for i
0		161	1	0
1		42	0	1
2	161 / 42 = 3	161 – 3 * 42 = 35	1 – 3 * 0 = 1	0 – 3 * 1 = –3
3	42 / 35 = 1	42 – 1 * 35 = 7	0 – 1 * 1 = –1	1 * 1 * –3 = 4
4	35 / 7 = 5	35 – 5 * 7 = 0	Do not calculate	Do not calculate

$q_i = q_3$, where i = 3

So, $q_3 = r_2 / r_3 = 35 / 7 = 5$

$r_4 = r_2 - q_3 * r_3 = 35 - 5 * 7 = 0$

- Do not calculate s and t once you get r = 0
- Last calculated s and t become x and y respectively

- Last calculated r becomes gcd
- So, according to extended Euclidean algorithm, for numbers 161 and 42

  gcd(161, 42) = 7
- In the equation ax + by = gcd(161,42)

  x = – 1

  y = 4
- You can verify the answer by putting the values in the equation.
- Left-hand side:

  ax + by = 161 * – 1 – 42 * 4

  = – 161 + 168

  ax + by = 7 [which is equal to gcd(161, 42)]

Ex. 2.9.5 : For a = 256 and b = 5004, calculate gcd(a, b) and also the values of x and y to satisfy the extended Euclidean algorithm.

Soln. :

First note that b > a. Hence, lets swap the values to make the calculations simple. We would re-swap them in the final answer.

So, assume a = 5004 and b = 256

Index i	quotient q for i	Remainder r for i	S for i	t for i
0		5004	1	0
1		256	0	1

Index i	quotient q for i	Remainder r for i	s for i	t for i
0		5004	1	0
1		256	0	1
2	5004 / 256 = 19	5004 – 19 * 256 = 140	1 – 19 * 0= 1	0 – 19*1 = – 19
3	256 / 140 = 1	256 – 1 * 140 = 116	0 – 1 * 1= – 1	1 – 1 * – 19 = 20
4	140 / 116 = 1	140 – 1 * 116 = 24	1 – –1 * 1= 2	– 19 – 1 * 20 = – 39
5	116 / 24 = 4	116 – 24 * 4 = 20	– 1 – 4 * 2= –9	20 – 39 * 4 = 176
6	24 / 20 = 1	24 – 20 * 1 = 4	2 – 9 * 1= 11	– 39 – 1 * 176 = – 215
7	20 / 4 = 5	20 – 4* 5 = 0	Do not calculate	Do not calculate

gcd(256, 5004) = 4

- We originally swapped the value. So, re-swap it.
- Hence, x = – 215 and y = 11
- Putting it in the equation, you get,

Left-Hand Side

$ax + by = 256 * -215 + 5004 * 11$

$= -55,040 + 55,044$

$ax + by = 4$ [which is equal to right hand side = gcd(256, 5004)]

Ex. 2.9.6 : For a = 86 and b = 14, calculate gcd(a, b) and also the values of x and y to satisfy the extended Euclidean algorithm.

Soln. :

Index i	quotient q for i	Remainder r for i	s for i	t for i
0		86	1	0
1		14	0	1
2	86 / 14 = 6	86 – 6 * 14 = 2	1 – 6 * 0 = 1	0 – 6 * 1 = – 6
3	14 / 2 = 7	14 – 2 * 7 = 0	Do not calculate	Do not calculate

gcd(86,14) = 2

x = 1, y = – 6

To verify, let's put the above values in the equation:

$ax + by = 86 * 1 - 14 * 6$

$= 86 - 84$

$= 2$ [this is gcd value that we got for 86,14]

Ex. 2.9.7 : For a = 999 and b = 9, calculate gcd(a, b) and also the values of x and y to satisfy the extended Euclidean algorithm.

Soln. :

Index i	quotient q for i	Remainder r for i	s for i	t for i
0		999	1	0
1		9	0	1
2	999 / 9= 111	999 – 111*9= 0	Do not calculate	Do not calculate

gcd(999,9) = 9

x = 0, y = 1

Let's put those values in the equation :

$ax + by = 999 * 0 + 1 * 9$

$= 9$ **[which matches the gcd value we got for 999,9]**

Tip : It would perhaps be easy for you to calculate the first two columns q and r until you get 0 in r. That way you are focusing on one set of calculation at a time. Once you have q and r computed in the first two columns, calculate s and t by just substitution the values of q and r in the respective equations.

2.9.4 Multiplicative Inverse using Extended Euclidean Algorithm

If you recall our discussion from the previous section on modular inverse, you understand what inverse operation is. One application of the extended Euclidean Algorithm is to find out multiplicative inverse.

Definition : *A modular multiplicative inverse of an integer a is an integer x such that the product ax is congruent to 1 with respect to the modulus m*

In modular arithmetic, it can be written as, $ax \equiv 1 \pmod{m}$

According to the extended Euclidean Algorithm,

$$ax + my = \gcd(a, m) = 1$$

$$ax + my = 1$$

$$ax - 1 = (-y)m$$

Dividing both sides by (mod m)

$$ax \pmod{m} - 1 \pmod{m} = (-y)m \pmod{m}$$

$$ax \pmod{m} - 1 \pmod{m} = 0$$

$$ax \equiv 1 \pmod{m}$$

Note : The multiplicative inverse of a modulo m exists if and only if a and m are coprime (i.e. if gcd(a, m) = 1)

So, if you come across a question where it is asked to calculate multiplicative inverse such that gcd(a, m) is not 1, do not attempt to solve the problem. Just calculate the gcd and show that the numbers are not coprime and hence the multiplicative inverse does not exist.

Ex. 2.9.8 : Find multiplicate inverse of 24140 mod 40902.

Soln. :

$$\begin{aligned}
\gcd(24140, 40902) &= \gcd(24140, 40902 \bmod 24140) \\
&= \gcd(24140, 16762) \\
&= \gcd(24140 \bmod 16762, 16762) \\
&= \gcd(7378, 16762) \\
&= \gcd(7378, 16762 \bmod 7378) \\
&= \gcd(7378, 2006) \\
&= \gcd(7378 \bmod 2006, 2006) \\
&= \gcd(1360, 2006) \\
&= \gcd(1360, 2006 \bmod 1360) \\
&= \gcd(1360 \bmod 646, 646) \\
&= \gcd(68, 646 \bmod 68) \\
&= \gcd(68 \bmod 34, 34) \\
&= \gcd(0, 34) \\
\gcd(24140, 40902) &= 34
\end{aligned}$$

Here you find that gcd(24140, 40902) = 34. Hence, multiplicative inverse of 24140 mod 40902 does NOT exist.

Note : You can also calculate gcd using tabular method as you learnt in the extended Euclidean algorithm section to avoid repeating the gcd steps to calculate x and y if gcd = 1 does exist.

Ex. 2.9.9 : Find multiplicate inverse of 8 mod 11.

Soln. :

- Since 8 < 11, let's swap the values for simplicity.

Index i	quotient q for i	Remainder r for i	s for i	t for i
0		11	1	0
1		8	0	1

- Calculate next steps :

Index i	quotient q for i	Remainder r for i	s for i	t for i
0		11	1	0
1		8	0	1
2	11 / 8= 1	11 – 8 * 1= 3	1 – 1 * 0= 1	0 – 1 * 1= – 1
3	8 / 3= 2	8 – 3 * 2= 2	0 – 2 * 1= – 2	1 – 2 * – 1= 3
4	3 / 2 = 1	3 – 2 * 1 = 1	1 – 1 * – 2 = 3	– 1 – 1 * 3 = – 4
5	2 / 1 = 2	2 – 1 * 2 = 0	Do not calculate	Do not calculate

Let's re-swap the values.

Hence, x = – 4 and y = 3 Putting the values in the extended Euclidean algorithm, you get

$$ax + by = 1$$

$$8(-4) + 11(3) = 1$$

Since, you have to find multiplicative inverse in mod 11, divide both sides by mod 11.

$$8(-4) \bmod 11 + 11(3) \bmod 11 = 1 \bmod 11$$

$$8(-4) \bmod 11 + 0 = 1$$

Recall our discussion on calculating mod for negative numbers. You need to keep adding mod until the number turns positive and then calculate mod on the positive number you got.

$$-4 + 11 = 7$$

$$7 \bmod 11 = 7$$

Hence, $8(7) \bmod 11 = 1$

So, multiplicative inverse of 8 mod 11 is 7.

Note : You can also test your solution. If there are mistakes, re-visit the steps you took. In this example,8*7 = 56 and 56 mod 11 = 1. Hence, you find that 7 is indeed multiplicative inverse of 7 in mod 11.

Ex. 2.9.10 : Find multiplicate inverse of 1234 mod 4321.

Soln. :

Since 1234 < 4321, let's swap the values for simplicity.

Index i	quotient q for i	Remainder r for i	s for i	t for i
0		4321	1	0
1		1234	0	1
2	4321 / 1234 = 3	4321 – 3 * 1234 = 619	1 – 3 * 0 = 1	0 – 3 * 1 = – 3

Index i	quotient q for i	Remainder r for i	s for i	t for i
3	1234 / 619 = 1	1234 – 1 * 619 = 615	0 – 1 * 1 = – 1	1 – 1 * – 3 = 4
4	619 / 615 = 1	619 – 1 * 615 = 4	1 – 1 * – 1 = 2	– 3 – 1 * 4 = – 7
5	615 / 4 = 153	615 – 4 * 153 = 3	– 1 – 153 * 2 = – 307	4 – 153 * – 7 = 1075
6	4 / 3 = 1	4 – 1 * 3 = 1	2 – 1 * – 307 = 309	–7 – 1 * 1075 = – 1082
7	3 / 1 = 3	3 – 3 * 1 = 0	Do not calculate	Do not calculate

Let's re-swap the values.

Hence, x = – 1082 and y = 309

Putting it in the equation,

$$ax + by = 1$$

$$1234(-1082) + 4321(309) = 1$$

Dividing both sides by mod 4321, you get

$$1234 (-1082) \bmod 4321 + 4321 (309) \bmod 4321 = 1 \bmod 4321$$

$$1234 (-1082) \bmod 4321 + 0 = 1$$

Convert –1082 to positive

$$-1082 + 4321 = 3239$$

$$3239 \bmod 4321 = 3239$$

Hence,

$$1234 (3239) \bmod 4321 = 1$$

Or 3239 is multiplicative modular inverse of 1234 in mod 4321.

2.10 Public Key Cryptography

You learnt about the use of asymmetric keys earlier. Recall and revise that section before you proceed.

Public key cryptography relies on the use of such asymmetric keys for providing various cryptographic services such as encryption and digital signature.

✍ ***Definition :*** *Public key cryptography is a cryptographic scheme that uses two mathematically related keys, a public key and a private key, for providing various cryptographic services.*

2.10.1 Principles of Public Key Cryptosystems

Following are some basic principles of public key-based cryptosystems.

1. Public key cryptosystems require the use of two keys – a public key and a private key.
2. Public keys are widely known.
3. Private key is kept secret with its owner.
4. The two keys are mathematically related and form a key pair.
5. One key in the key pair cannot be used to derive the other key in the key pair.
6. Any key in the key pair can be used for encryption. The other key then must be used for decryption.
7. Sender and the receiver both must have their own key pairs.

In this section, you are going to learn about various asymmetric key based algorithms.

2.10.2 RSA Algorithm

RSA, named after its inventors Ron Rivest, Adi Shamir and Leonard Adleman, is an asymmetric key based algorithm. As you understand, RSA, or any other asymmetric key based algorithms can be used for confidentiality [encryption, decryption], authentication and non-repudiation. RSA is based on finding prime factors for very large numbers. The length of numbers that we are referring to here is around 500 digits!

Sr. No.	RSA Key Length	Number of digits
1.	1024-bit	309
2.	2048-bit	617
3.	4096-bit	1233

Let's understand how RSA derives public and private keys and how does encryption and decryption process work based on the derived keys.

1. Choose two random large prime numbers, 'p' and 'q'
2. Multiply the numbers, $n = p*q$
3. Choose a random integer to be encryption key 'e' such that 'e' and $(p-1)(q-1)$ are relatively prime.
4. Decryption key is computed as $d = e^{-1} \bmod ([p-1]*[q-1])$
5. The public key = (n, e)
6. The private key = (n, d)
7. For encrypting message 'M' with public key (n, e), you get ciphertext $C = M^e \bmod n$.
8. For decrypting ciphertext with private key (n, d), you get plaintext $M = C^d \bmod n$.

Ex. 2.10.1 : Perform encryption and decryption using RSA algorithm with p = 7, q = 11, e = 17 and M = 8.

Soln. :

$$n = p * q$$
$$n = 7 * 11 = 77$$
$$r = (p-1) * (q-1)$$
$$r = 6*10 = 60$$
$$d = e^{-1} \bmod r$$
$$ed \equiv 1 \bmod 60$$
$$17d \equiv 1 \bmod 60$$

Let's calculate modulo inverse using extended Euclidean algorithm (swapping 17 and 60)

Index i	quotient q for i	Remainder r for i	s for i	t for i
0		60	1	0
1		17	0	1
2	60 / 17 = 3	60 – 3 * 17 = 9	1 – 3 * 0 = 1	0 – 3 * 1 = – 3
3	17 / 9 = 1	17 – 1 * 9 = 8	0 – 1 * 1 = – 1	1 – 1 * – 3 = 4
4	9 / 8 = 1	9 – 1 * 8 = 1	1 – 1 * – 1 = 2	– 3 – 1 * 4 = – 7
5	8 / 1 = 8	8 – 1 * 8 = 0	Do not calculate	Do not calculate

- Let's re-swap the values.

$$x = -7$$

$$y = 2$$

- Putting the values in the extended Euclidean algorithm, you get

$$ax + by = 1$$

$$17(-7) + 60(2) = 1$$

Since, you have to find multiplicative inverse in mod 60, divide both sides by mod 60.

$17(-7) \bmod 60 + 60(2) \bmod 60 = 1 \bmod 60$

$17(-7) \bmod 60 + 0 = 1$

Recall our discussion on calculating mod for negative numbers. You need to keep adding mod until the number turns positive and then calculate mod on the positive number you got.

$$-7 + 60 = 53$$

$$53 \bmod 60 = 53$$

Hence, $17(53) \bmod 60 = 1$

Hence, value of decrypting key, d = 53

Now, you have all the values needed for encryption and decryption.

As per RSA,

$$C = M^e \bmod n$$

$$C = 8^{17} \bmod 77$$

$$\mathbf{C = 57}$$

$$M = C^d \bmod n$$

$$M = 57^{53} \bmod 77$$

$$\mathbf{M = 8}$$

Ex. 2.10.2 : Given modulus n=91 and public key, e=5, find the values of p, q, phi(n), and d using RSA. Encrypt M=25. Also perform decryption. **MU - Dec. 18, 10 Marks**

Soln. :

Since, n is 91, assuming p and q were 7 and 13 respectively. (by factorizing 91).

$$r = (p-1) * (q-1)$$

$$r = 6*12 = 72$$

phi(n) can be calculated using the formula

$$\text{phi}(n) = n \times \left(1 - \frac{1}{7}\right) \times \left(1 - \frac{1}{13}\right)$$ [where 7 and 13 are prime factors of 91)

Hence, $\text{phi}(n) = 72$

$$d = e^{-1} \bmod r$$

$$ed \equiv 1 \bmod 72$$

$$5d \equiv 1 \bmod 72$$

Let's calculate modulo inverse using extended Euclidean algorithm (swapping 5 and 72)

Index i	quotient q for i	Remainder r for i	s for i	t for i
0		72	1	0
1		5	0	1
2	72 / 5 = 14	72 – 5 * 14 = 2	1 – 0 * 14 = 1	0 – 1 * 14 = – 14
3	5 / 2 = 2	5 – 2 * 2 = 1	0 – 1 * 2 = – 2	1 – -14 * 2 = 29
4	2 / 1 = 2	2 – 1 * 2 = 0	Do not calculate	Do not calculate

Let's re-swap the values.

$$x = 29$$

$$y = -2$$

Putting the values in the extended Euclidean algorithm, you get

$$ax + by = 1$$

$$5 \times 29 + 72 \times (-2) = 1$$

Since, you have to find multiplicative inverse in mod 72, divide both sides by mod 72.

$$5 \times 29 \bmod 72 + 72 \times (-2) \bmod 72 = 1 \bmod 72$$

$$5 \times 29 \bmod 72 + 0 = 1 \bmod 72$$

$$5 \times 29 \bmod 72 = 1 \bmod 72$$

Hence, multiplicative is 29.

Therefore, the value of decrypting key, $d = 29$.

Now, you have all the values needed for encryption and decryption.

As per RSA,

$$C = M^e \bmod n$$

$$C = 25^5 \bmod 91$$

$$\mathbf{C = 51} \text{ [encrypted message]}$$

$$M = C^d \bmod n$$

$$M = 51^{29} \bmod 91$$

$$\mathbf{M = 25} \text{ [decrypted message]}$$

Ex. 2.10.3 : Given modulus n=221 and public key, e=7, find the values of p, q, phi(n), and d using RSA. Encrypt M = 5.

MU - May 19, 10 Marks

Soln. :

Since, n is 221, assuming p and q were 13 and 17 respectively. (by factorizing 221).

$$r = (p-1) \times (q-1)$$

$$r = 12 \times 16 = 192$$

phi(n) can be calculated using the formula

$$phi\,(n) = n \times \left(1 - \frac{1}{13}\right) \times \left(1 - \frac{1}{17}\right)$$

Hence, $phi\,(n) = 192$

$$d = e^{-1} \bmod r$$

$$ed \equiv 1 \bmod 192$$

$$7d \equiv 1 \bmod 192$$

Let's calculate modulo inverse using extended Euclidean algorithm (swapping 7 and 192)

Index i	quotient q for i	Remainder r for i	s for i	t for i
0		192	1	0
1		7	0	1
2	192 / 7 = 27	192 – 7 * 27 = 3	1 – 0 * 27 = 1	0 – 1 * 27 = – 27
3	7 / 3 = 2	7 – 3 * 2 = 1	0 – 1 * 2 = – 2	1 – (– 27) * 2 = 55
4	3 / 1 = 3	3 – 1 *3 = 0	Do not calculate	Do not calculate

Let's re-swap the values.

$$x = 55$$

$$y = -2$$

Putting the values in the extended Euclidean algorithm, you get

$$ax + by = 1$$

$$7 \times 55 + 192 \times (-2) = 1$$

Since, you have to find multiplicative inverse in mod 192, divide both sides by mod 192.

$$7 \times 55 \bmod 192 + 192 \times (-2) \bmod 192 = 1 \bmod 192$$

$$7 \times 55 \bmod 192 - 0 = 1 \bmod 192$$

Hence, multiplicative is 55.

Therefore, the value of decrypting key, d = 55.

Now, you have all the values needed for encryption and decryption.

As per RSA,

$$C = M^e \bmod n$$

$$C = 5^7 \bmod 192$$

$$\mathbf{C = 173}\ \text{[encrypted message]}$$

$$M = C^d \bmod n$$

$$M = 173^{55} \bmod 192$$

$$\mathbf{M = 5}\ \text{[decrypted message]}$$

2.10.2(A) Attacks on RSA

1. **Brute-force Attack**

   Here the attacker tries to find factors of 'n' by trying out various possibilities.

2. **Common Modulus**

- To avoid generating a different modulus n = p * q for each user one may wish to fix 'n' for all the users. It might seem like deriving the decrypting key 'd' is not possible for every encrypting key e by any other user since encrypting key 'e' is a randomly chosen value. But, the problem with this approach is that a particular user who knows her pair of 'e' and 'd' can successfully use her own pair to find the factors for common modulus. Once the factors are known, since the encrypting key 'e' is known to everyone (because it is public key), the decrypting key 'd' could be found out. Hence, you should not be using common modulus to generate keys for multiple users.

3. **Choosing Smaller Numbers**

   The security of the algorithm comes from the fact that factoring large numbers is computationally intensive. Sometimes, to improve system performance, smaller numbers can be chosen which can significantly enhance the performance but at the cost of making the algorithm weaker. Hence, you should always choose large numbers to maintain the strength of the algorithm.

4. **Man in the Middle Attack**

   The attacker could collect all the ciphertext coming out from the user's system (that is encrypted with her private key) and try to find the private key. The information known to the attacker is the ciphertext and public key.

2.11 Message Authentication Requirements

- So far, in the earlier chapters, you learnt about information secrecy – which was all about keeping the message secret. The focus of this chapter is information accuracy which is about ensuring the message integrity or message authentication.

✍ ***Definition :*** *Message authentication is a process to ensure that the received message is exactly the same as it was sent.*

What does this mean? This means that :

- The message has not been altered in anyway
  - No addition
  - No modification
  - No deletion
- The message is actually sent by the sender (proof of sender's identity)
- The order or the sequence of the messages is not changed
- The messages are sent and received within an expected time frame

- Let's consider a day to day scenario. You go to a shop to buy bread and in-exchange you give the shopkeeper a Rupees 100 currency note.
- The shopkeeper happily takes the note from you, examines it briefly, keeps it in the drawer and return you some change. You take the change, examine the returned currency briefly, slide it down in your wallet and move.
- What just happened? Why did the shopkeeper take the note you gave without any hesitation? Why did you take the change without any hesitation?
- What did the shopkeeper examine when you gave the note and why? What did you examine when you received the change and why? Are you telling me, *come on*, this is child simple? Are you trying to explain me the following?
- There were two parties to the transaction – you and shopkeeper
- Shopkeeper wanted to accurately determine that the note you gave was genuine and not fake
- Shopkeeper looked at some of the known properties of the note (that were attached to the note itself) and verified it accurately
- You wanted to accurately determine that the change currency that you got was genuine and not fake
- You looked at some of the known properties of the note that you got and verified the note accurately
- So, you understand that in this world, where the faith is based on the principle of "trust but verify", you need a mechanism to accurately determine the accuracy of the information that you get.
- Have you ever trusted a fake news or video clip and later laughed at yourself or felt bad about trusting the information without checking it genuinely?
- Don't you feel that there should be *someone* who could just tell you if the information you got was accurate or not? Did you really receive the email that you were sent, or someone modified it on its way ? Did someone modify your file? If all these problems worry you, I request you not to worry, you have a way out. Please heartily welcome Cryptographic Hash functions!

2.12 Message Authentication Functions

At a high level, the message authentication can be performed using three mechanisms :

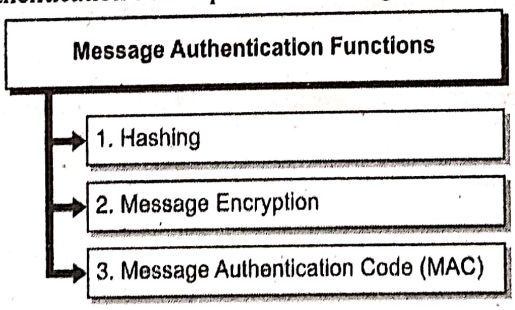


Fig. 2.12.1 : Message authentication function

1. Hashing

It deals with producing a unique hash value of the message that can be computed at the sender's and the receiver's end. If the hashes match at both the ends, the message is verified.

2. **Message encryption**

In this, the ciphertext of the entire message can be used to serve as its authenticator. The message is encrypted at the sender's and the receiver's end. If they both get the same ciphertext using the same key, it verifies the message. Note here that the focus is not to encrypt the message but to get the resulting ciphertext to serve as a way to verify the authenticity of the message.

3. **Message Authentication Code (MAC)**

MAC is very similar to hash. It uses a key to calculate the hash value of the message. The MAC is calculated at both the ends (sender's and receiver's) using the same key. If the MAC value matches at both the ends, the message is verified. You will learn about it in detail in the subsequent sections.

2.13 Cryptographic Hash Functions

As encryption provides message confidentiality, hashing provides message integrity and authentication.

2.13.1 Introduction

✍ ***Definition :*** *Hashing is the process of taking any length of input information and finding a unique fixed length representation of that input information.*

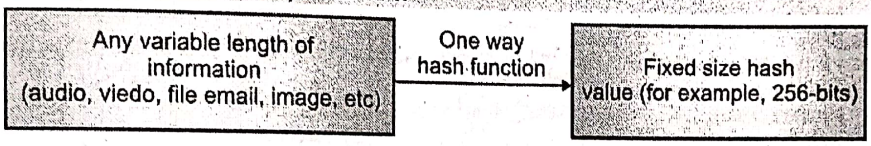


Fig. 2.13.1 : Example of hash function

Hashing is the process of finding a unique message digest (or hash value) that corresponds to the input information.

The length of input could be just one character or a huge video file. Hashing always produces a fixed size representation of the information. Note here that hashing does not make the information unreadable. It is not same as encryption.

Hashing is just a way to attach some additional information to the original information that could later prove that the information is not modified.

Recall the discussion from concept building Section 3.1. The shopkeeper can see Rupees 100 written clearly but verifies the currency note using other properties "attached" to it that proves its originality and authenticity.

2.13.2 How does this Work?

At the source of the information (it could be sender, website, company or anything else where the information is created) the hash value is calculated. This hash value along with the original information is sent to the receiver as shown in Fig. 2.13.2.

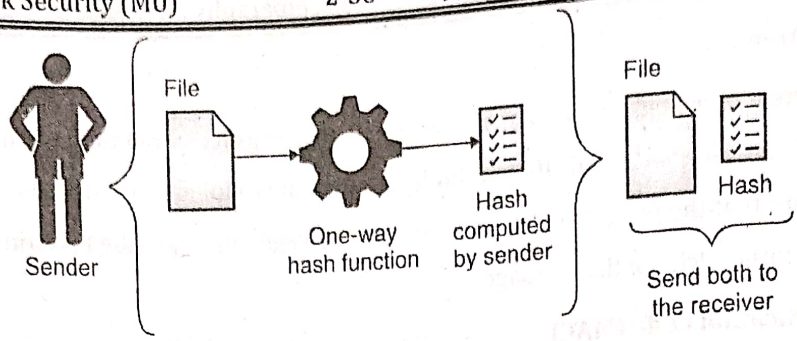


Fig. 2.13.2 : Working of hash algorithms

At the destination of the information (it could be a receiver, website, email program, or anything else where th information is received to process further), the hash value is calculated again and matched with the hash value tha came with the information from source. If the two hash values (at source and at destination) match, the information i determined to be unmodified and is consumed. But, if the two hash values do not match, it proves that th information is altered and is often rejected as shown in Fig. 2.13.3.

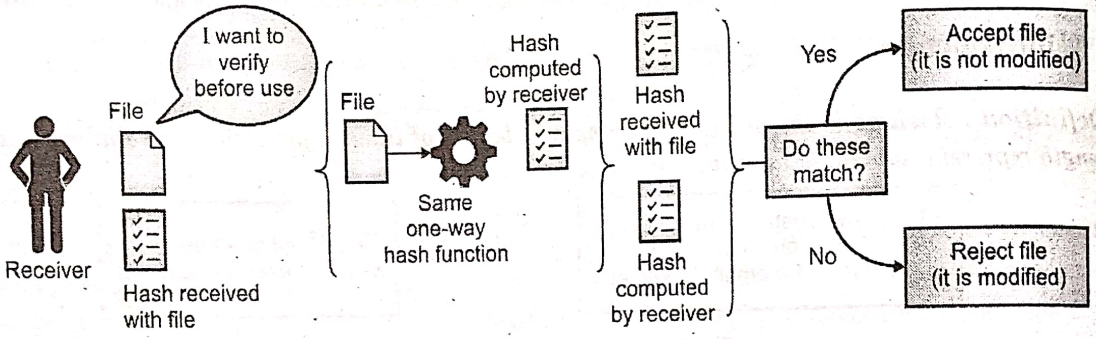


Fig. 2.13.3 : Hashing used for proving data integrity

I want to again stress on the point that hashing is only for integrity checking of the information. The origin information may or may not be encrypted. If the information is to be encrypted, following could be one of th sequences for integrity checking.

1. Create information
2. Calculate its hash value
3. Encrypt information
4. Send encrypted information and hash value
5. Receive encrypted information and hash value
6. Decrypt information
7. Calculate its hash value at the receiving end
8. Match source and destination hash
9. If matched, process information
10. If not matched, reject information

Now, you might be thinking, yeah, this sounds good but what if someone captures the message and the ha alters the message and creates and attaches the new hash to match the altered message and sends it to the receiv How would receiver ever know? Great question, I must say. I would answer that later on. Let's park it at the momen

2.13.3 Characteristics of Hash Functions

Unlike encryption, hash functions have some unique properties are shown in Fig. 2.13.4. Let's learn about them.

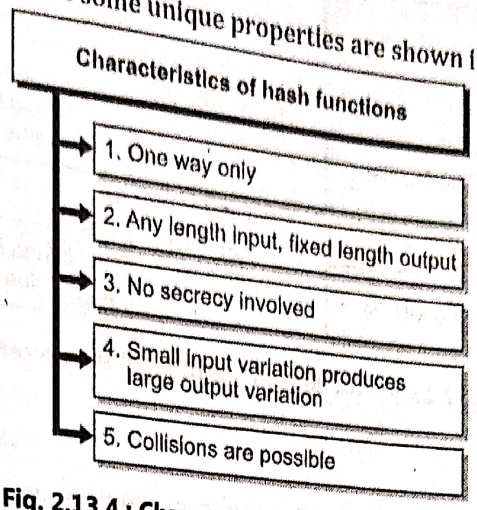


Fig. 2.13.4 : Characteristics of hash function

1. **One-way only**

This means that it is easy to calculate hash value from information and nearly impossible to convert the hash value back to the original information.

Understand it like this. It is easy to convert milk into cheese, but can you convert cheese back to the original milk? Sounds weird and impossible right? That's how hash functions are One-way only.

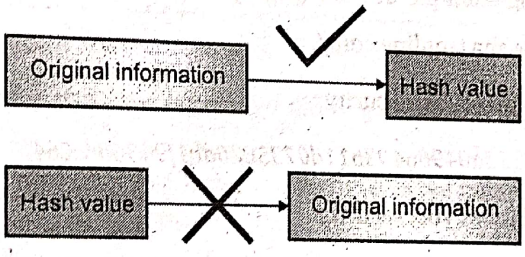


Fig. 2.13.5 : One-way only

2. **Any length input, fixed length output**

Hash functions can work on any length of input. It could work on a single character or a huge video file. It would always produce the same length output.

Unlike encryption where the output size is more or less same as the input size, output from hashing process is only a unique representation of the original information and does not contain the information itself. Hence, the output length from hashing does not need to change with the length of input.

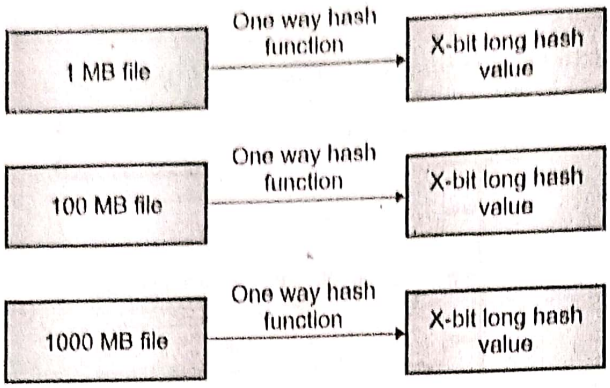


Fig. 2.13.6 : Any length input, fixed length output

3. **No secrecy involved**

Hashing process does not require a key and neither it requires any secret. The hashing algorithms are implemented such that they are only a one-way process producing a unique representation of the input information. Algorithms are publicly known as well.

4. **Small variation in input produces large variation in output**

It is nearly impossible to establish any relation between input and output in the hashing process. A small variation in the input totally changes the hash output. This is also called avalanche effect. Let's see an example. You can also try the following example at your end by calculating SHA-1 hash output using an online SHA-1 calculator such as http://www.sha1-online.com/.

Original information : I love cybersecurity

SHA-1 Hash value : *653da04906493b11d0735020db1f94360cac64f0*

Original information : U love cybersecurity

SHA-1 Hash value : *8b9a7e5b4e5c8306c6ba678454e485356cb0aa24*

Note that it does not matter which online calculator you use. You would get the same SHA-1 output for the given input.

5. **Collisions are possible**

While it is impossible to find original information from hash output, a collision condition is possible. Collision is a situation where two different inputs produce the same hash output. While this condition is extremely rare, some historical hashing algorithms such as MD2, MD4, MD5, SHA-1 have been shown to produce collision. These algorithms are no more used in the industry today. A hashing algorithm is considered strong if it provides high collision resistance. Newer hashing algorithms such as SHA-256 and SHA3-256 provide strong collision resistance.

2.14 Hash Functions (Algorithms)

National Institute of Standards and Technology (NIST) has specified the family of hashing algorithms that may or may not be fit for current use in the industry.

Table 2.14.1 : Family of hashing algorithms

Hash Family Name	List of hashing algorithms	Approved for use?
SHA-1	SHA-1	No
SHA-2	SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 and SHA-512/256	Yes
SHA-3	SHA3-224, SHA3-256, SHA3-384, SHA3-512	Yes

Table 2.14.2 : Hashing Algorithms

Hashing Algorithm	Output size in bits	Block size in bits	Rounds	First published
SHA-1	160	512	80	1995
SHA-224	224	512	64	2004
SHA-256	256	512	64	2001
SHA-384	384	1024	80	2001
SHA-512	512	1024	80	2001
SHA-512/224	224	1024	80	2012
SHA-512/256	256	1024	80	2012
SHA3-224	224	1152	24	2015
SHA3-256	256	1088	24	2015
SHA3-384	384	832	24	2015
SHA3-512	512	576	24	2015

2.14.1 SHA-1

✍ ***Definition :*** *Secure Hash Algorithm 1 is a cryptographic hash function that produces 160-bit long hash value from a given input.*

Collisions have been found in the algorithm and hence it is avoided in the industry wherever possible.

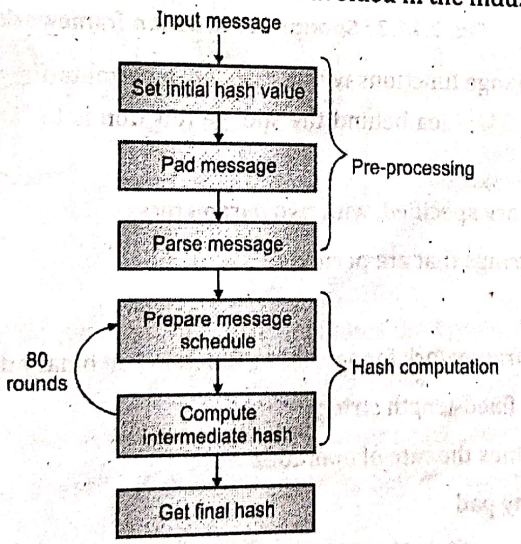


Fig. 2.14.1 : SHA-1

The algorithm has two stages are shown in Fig. 2.14.1.

1. **Pre-processing**

   Pre-processing involves padding a message, parsing the padded message into equal sized blocks and then setting initial values to be used for hash computation.

2. **Hash computation**

   The hash computation generates a message schedule (sequence of processing) from the padded message and uses that schedule along with various functions, constants and mathematical operations to generate a series of intermediate hash values per round. The final hash is value is computed after the last round and is 160-bit long.

2.14.2 SHA-3

✍ ***Definition :*** *Secure Hash Algorithm 3 is the newest addition to the hash function family. It is comprised of various hash functions that compute secure hash values.*

It is considered to be most secure against collision and is highly recommended to be used in the industry. SHA-3 is structurally quite different from earlier versions : SHA-1 and SHA-2. SHA-3 is a subset of the broader cryptographic primitive family KECCAK.

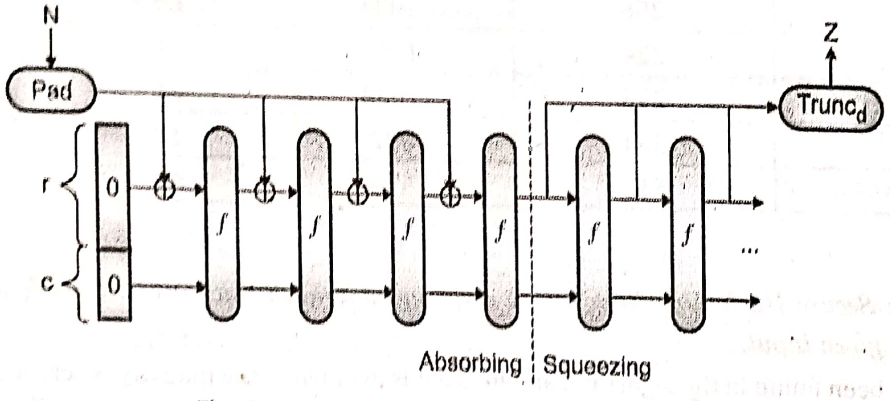


Fig. 2.14.2 : Sponge construction framework

KECCAK is the family of all sponge functions with a KECCAK-*p* permutation as the underlying function and multi-rate padding as the padding rule. The idea behind the sponge function is to "absorb" the information and "squeeze out" the hash value.

- The KECCAK-*p* permutations are specified, with two parameters :
  1. The fixed length of the strings that are permuted
  2. The number of rounds
- The sponge construction is a framework for specifying functions on binary data. It has three components :
  1. Function f that works on fixed-length strings
  2. Parameter r that determines the rate of operation
  3. A padding rule denoted by pad
- The sponge function can be denoted as SPONGE[f, pad, r].

- A sponge function takes two inputs:
  1. A bit string that is denoted by N
  2. The bit length that is denoted by d
- So, the overall function is denoted as SPONGE[f, pad, r](N, d).
- In the sponge function, Absorb Phase involves :
  1. XORing of message blocks into a subset of the state
  2. Using the permutation function for transferring the whole state
- In the sponge function, Squeeze Phase involves :
  1. Reading output blocks from the same subset of state
  2. Replacing with state transformation function

2.15 MAC (Message Authentication Code)

Hash values provide integrity. But what if someone alters the message and adds a new hash? How do you know that you received the original message that the sender had intended to send? Hash values are computed on the original message and usually sent along with the message. Creating hash values does not require any keys. Hence, hash values can only provide integrity but cannot provide any additional assurance that the message is authentic.

✍ ***Definition :*** *A Message Authentication Code (MAC) is a piece of information that can be used to authenticate a message.*

MAC confirms that the message came from the stated sender and has not been changed. The MAC value protects both a message's data integrity as well as its authenticity. MAC is sometimes also called as ***keyed* hash.**

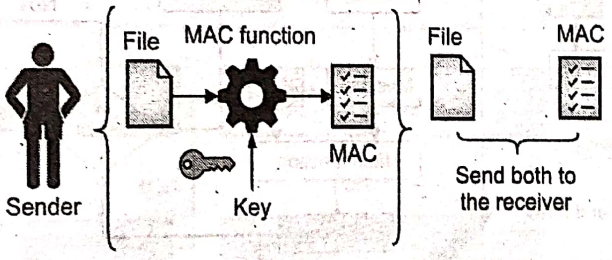


Fig. 2.15.1

The sender intends to send a file (or a message). He computes the MAC value for the file using the pre-shared key. He then forwards the message as well as the corresponding MAC value to the receiver.

At the receiver's end, the MAC value is again computed using the same pre-shared key. This computed MAC value is then compared with the received MAC. If these two match, the file (or the message) is not altered and is authenticated.

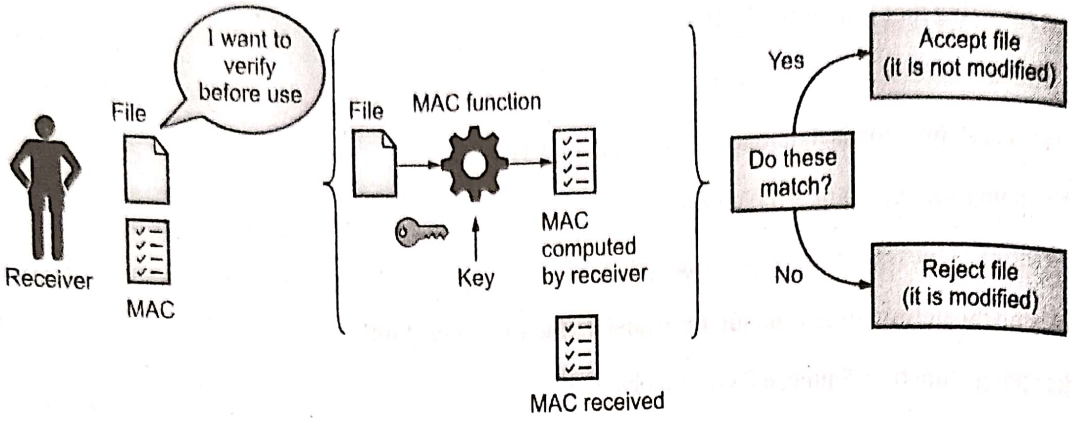


Fig. 2.15.2

There are three types of MAC :

1. HMAC (Hash MAC)
2. CBC-MAC
3. CMAC (Cipher-based MAC)

Let's learn about each of them.

2.15.1 HMAC

Definition : *In HMAC, a hashing function is used as a MAC function to calculate the MAC value.*

The hashing function could be general hash functions such as MD5, SHA-1, or SHA-2.

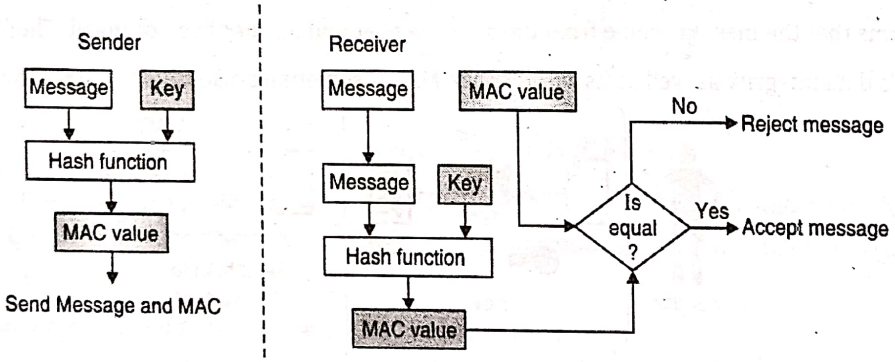


Fig. 2.15.3

On the sender's side, the message and a pre-shared key is passed together into a hash function to compute a MAC. The computed MAC along with the original message is sent to the receiver. The key is not sent.

Note here that the key is not used to encrypt the message. The key just provides a random value that when passed with the message to a hashing function generates a MAC value. The key has no other role except to provide a random value. On the receiver's side, the message is again passed through the same hashing function using the same pre-shared key. A MAC value is computed at the receiver's side and is matched with the MAC value that came from the sender. If the two MAC values match, the message is accepted else the message is rejected.

2.15.2 CBC-MAC

Definition : *In CBC-MAC, a symmetric block cipher encryption function in the CBC mode is used as the MAC function to calculate the MAC value.*

The message is encrypted with a symmetric block cipher in the CBC mode, and the output of the final block of ciphertext is used as the MAC. The sender does not send the encrypted version of the message, but instead sends the plaintext version and the computed MAC.

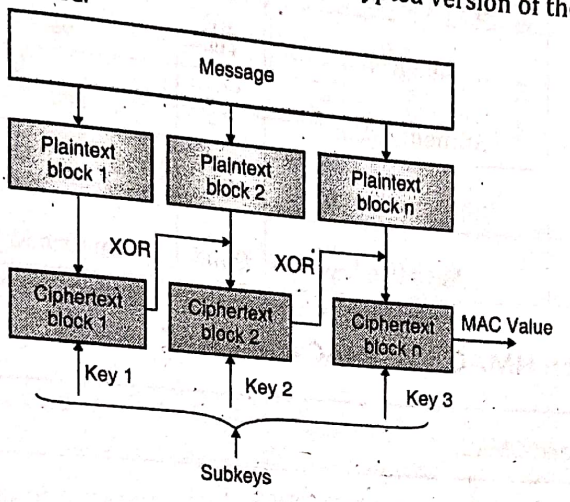


Fig. 2.15.4

The receiver receives the plaintext message and encrypts it with the same symmetric block cipher in CBC mode and calculates a MAC value at her end. If the two MAC values (one received and one computed) match, the message is accepted else it is rejected. This method does not use a hashing algorithm as in HMAC.

2.15.3 CMAC

CMAC (Cipher-based MAC) is very similar to CBC-MAC.

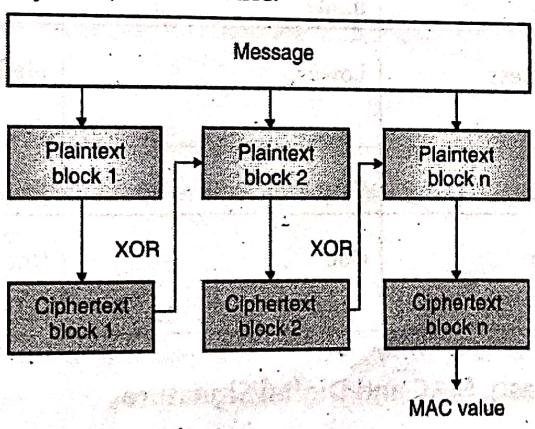


Fig. 2.15.5

Definition : *In CMAC, a symmetric block cipher encryption function is used as the MAC function to calculate the MAC value.*

Note here that like CBC-MAC, a symmetric block cipher encryption function is used here as well but not in CBC mode. That's the major difference between CBC-MAC and CMAC. CMAC is typically calculated using AES-128 algorithm and provides strongest form of message authentication and integrity. It is also called as One-Key MAC or OMAC.

2.15.4 Comparison between Hash and MAC

The following Table 2.15.1 summarizes the difference between Hash and MAC.

Table 2.15.1

Sr. No	Comparison Attribute	Hash	MAC
1.	Integrity	Yes	Yes
2.	Authentication	No	Yes
3.	Non-repudiation	No	No
4.	Keys Used	None	Symmetric Keys

2.15.5 Comparison between HMAC, CBC-MAC and CMAC

Q. Compare and contrast HMAC and CMAC. **MU - Dec. 18, 5 Marks**

The following Table 2.15.2 summarises the key differences between HMAC, CBC-MAC and CMAC.

Table 2.15.2

Sr. No.	Comparison Attribute	HMAC	CBC-MAC	CMAC
1.	MAC generation function	Hash Function	Symmetric cipher in CBC mode	Symmetric cipher
2.	Speed of MAC generation	Highest	Lowest	Medium
3.	Strength of MAC	High	Very High	Very High
4.	Number of Keys used	One	One	One key divided into multiple sub-keys

2.15.6 Comparison between Hash, MAC and Digital Signature

The following Table 2.15.3 summarizes the difference between Hash, MAC and Digital Signatures. Your understanding of these differences is crucial.

Table 2.15.3

Comparison Attribute	Hash	MAC	Digital Signature
Integrity	Yes	Yes	Yes
Authentication	No	Yes	Yes
Non-repudiation	No	No	Yes
Keys Used	None	Symmetric Keys	Asymmetric Keys

2.16 Security of Hash Functions and MAC

Hash functions and MAC need to be secure by themselves to be useful. You cannot rely on these message authentication mechanisms if it is easy to "break" them. Let's understand some of the desired security properties of these mechanisms and possible attacks on them.

2.16.1 Security of Hash Functions

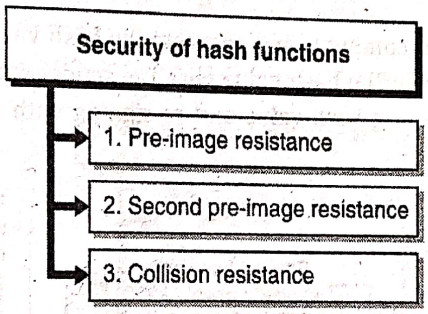


Fig. 2.16.1 : Security of hash functions

The three desired security properties of hash functions are as following.

1. **Pre-image resistance**

This property ensures the one-way only security criteria of hash functions. It states that for a given hash value h, it should be impossible to find any message m such that h = hash (m). So, in a nutshell, you cannot find the original message from its hash value.

2. **Second pre-image resistance**

This property states that given a message m_1, it is hard to find another message m_2, such that hash(m_1) = hash(m_2). So, looking at a message it should be hard to find another message which could produce the same hash. This property is also called **weak collision resistance.**

3. **Collision resistance**

This property states that it should be hard to find any two message pairs m_1 and m_2 such that hash(m_1) = hash(m_2). This property is also called **strong collision resistance.**

2.16.2 Attacks on Hash Functions and MAC

There are two possible ways to attack hash functions and MAC.

1. **Brute-force attack**

   In this, the attacker repeatedly tries to find various combinations of messages so as to produce a collision. The brute-force attack is more difficult to carry out on MAC than hash functions.

2. **Cryptanalysis**

   Similar to finding weakness in the encryption algorithms, the attackers tries to find a weakness in the hash functions and exploit that weakness to carry out further attacks.

2.17 Digital Signature

✍ ***Definition :*** *A digital signature is a hash value that has been encrypted with the sender's private key. The act of signing means encrypting the message's hash value with a private key (since no one else knows the sender's private key).*

2.17.1 How does this Work ?

So, as shown in Fig. 2.17.1 the sender computes and encrypts the hash value with her private key. As shown in Fig. 2.17.2, at the receiving end, you decrypt the hash value with the sender's public key. Now, because no one else knows the private key of the sender, altering hash value and re-signing with the private key of the sender is not possible.

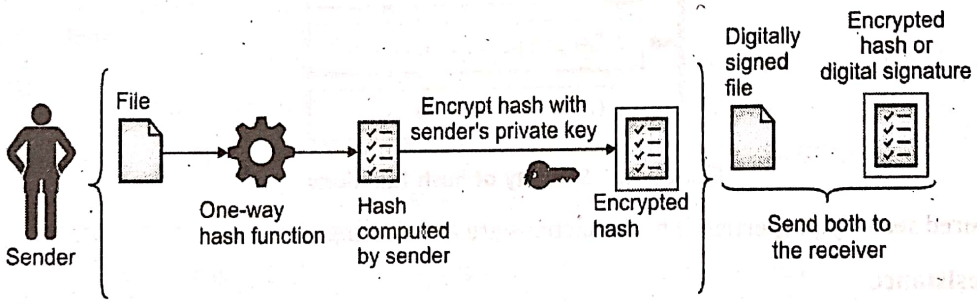


Fig. 2.17.1 : Computation of Digital Signature at the Sender's side

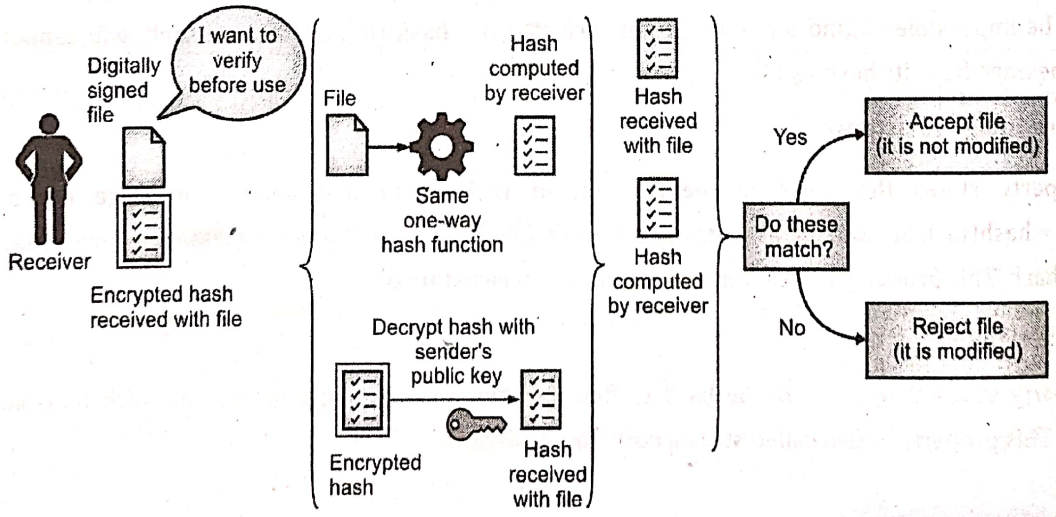


Fig. 2.17.2 : Verification of Digital Signature at the Receiver's side

Table 2.17.1

Sr. No.	Processing applied on a message	Security Property achieved
1.	Encryption	Confidentiality
2.	Hashing	Integrity
3.	Digitally Signing	Integrity, authentication, non-repudiation
4.	Encryption and digitally signing	Confidentiality, Integrity, authentication, non-repudiation

2.17.2 Application and Use of Digital Signature

1. Sending and receiving secure emails
2. Signing documents. For example, you can sign income tax returns using digital signature
3. Sending and receiving important files. For example, insurance policy documents, Aadhar card e-letter, etc.

2.17.3 Properties of Digital Signature

Q. What is the significance of a digital signature on a certificate? Justify. **MU - May 19, 5 Marks**

Digital signature provides three security properties as shown in Fig. 2.17.3.

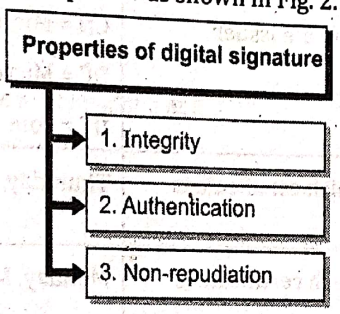


Fig. 2.17.3 : Properties of digital signature

1. **Integrity :** Via hash value calculation
2. **Authentication :** Via the ability to prove sender's identity by decrypting hash with the sender's known public key
3. **Non-repudiation :** Sender cannot deny sending the message because she used her private key to encrypt the hash

2.18 PKI X.509 Certificate

Q. Design Sample Digital Certificate and explain each field of it. **MU - Dec. 18, 5 Marks**

X.509 is a standard that defines the requirements for public key certificates.

Definition : *A certificate is a signed data structure that binds a public key to a person, computer, or organization.*

The certificate uniquely identifies the owner of a public key. Certificates are issued by certification authorities (CAs) such as Verisign, Global Sign etc. In a secure communication, certificates are used to identify the systems and establish trust amongst the communicating parties. Since its inception in 1998, three versions of the X.509 public key certificate standard have evolved. The most commonly used version of X.509 is version 3. X.509 certificates are used for secure communication such as establishing a https connection.

Table 2.18.1 : Contents of a X.509 version 3 certificate

Sr. No.	Fields	Purpose	Example
1.	Version	Identifies the version of the certificate	V3
2.	Serial Number	Unique number for the certificate	79ad16a14aa0a5ad4c7358f407132e65
3.	Signature algorithm	Algorithm used to create digital signature for certificate	sha1RSA
4.	Signature hash algorithm	Algorithm used to calculate hash of certificate	sha1
5.	Issuer	Name of certificate issuer	CN = Microsoft Root Certificate Authority DC = Microsoft DC = com
6.	Valid from	Date from which certificate is valid	Thursday, May 10, 2001 4:49:22 AM
7.	Valid to	Date until which certificate is valid	Monday, May 10, 2021 4:58:13 AM
8.	Subject	Name of certificate owner	CN = Microsoft Root Certificate Authority DC = Microsoft DC = com
9.	Public key	Public key	a5 94 ef 15 14 89 fd 4b 73 ... (4096 bits)
10.	Issuer unique ID	ID of issuing Certificate Authority (CA)	03475573948593hgfyuerwe327e7e52513fc2ae3
11.	Subject unique ID	ID of subject	0eac826040562797e52513fc2ae10a539559e4a4
12.	Extensions	Optional Information	Thumbprint, Friendly Name, Key Usage, etc.

Note: If you use Microsoft® Windows® OS, you can go to (Windows Key + R) Run and type certmgr.msc. It would open up certificate manager where you can browse various certificates stored on your system. Double-click on any certificate and examine the various fields it has.

2.19 Digital Signature Schemes

There are various schemes to create and verify digital signatures. These schemes are developed and implemented using various encryption and hashing algorithms. Let's learn about a few of them.

2.19.1 RSA Digital Signature Scheme

Definition : RSA signatures are based on public key cryptography.

RSA uses public key cryptography for creating and verifying digital signatures.

2.19.1(A) Key Generation

RSA digital signatures work on public and private key pairs. They can be generated by the regular key pair generating method by a Certificate Authority (CA) or on the user's system by herself. Recall from your reading on RSA algorithm that the keys are as following:

- The public key = *(n, e)*
- The private key = *(n, d)*

2.19.1(B) Message Signing

To sign a message, M

- Calculate the hash value of the message M at sender's end
- *h = hash(M)*
- Encrypt *h* using RSA private key
- Signature $S = (h)^d \bmod n$

2.19.1(C) Signature Verification

- Decrypt Signature *S* using public key
- $h' = (S)^e \bmod n$
- Calculate the hash value of the message M at receiver's end
- h = hash(M)
- If h = h', the signature is valid else the signature is invalid

Example of RSA Digital Signature

Ex. 2.19.1 : Alice chooses public key as (7, 33) and B chooses public key as (13, 221). Calculate their private keys. A wishes to send message m=5 to B. Show the message signing and verification using RSA digital signature. **(10 Marks)**

Soln. :

For Alice

- Public key is (7, 33). Hence, n = 33 and e = 7, where e is the encrypting key. Let's calculate Alice's private key which would give us d, decrypting key.
- Assume that 3 and 11 were the chosen prime factors for n = 33.

$n = p * q = 3 * 11 = 33$

$r = (p - 1) * (q - 1) = (3 - 1) * (11 - 1)$

$r = 2 * 10 = 20$

$d = e^{-1} \bmod r$

$ed \equiv 1 \bmod 20$

$7d \equiv 1 \bmod 20$

Let's calculate modulo inverse using extended Euclidean algorithm (swapping 7 and 20)

Index i	quotient q for i	Remainder r for i	s for i	t for i
0		20	1	0
1		7	0	1
2	20 / 7 = 2	20 – 7 *2 = 6	1 – 0*2 = 1	0 – 1*2 = -2
3	7 / 6 = 1	7 – 6 * 1 = 1	0 – 1 * 1 = -1	1 – (-2)*1 = 3
4	6 / 1 = 6	6 – 1 * 6 = 0	Do not calculate	Do not calculate

- Let's re-swap the values.

  $x = 3$

  $y = -1$

- Putting the values in the extended Euclidean algorithm, you get

  $ax + by = 1$

  $3 * 7 + 20 * (-1) = 1$

- Since, you have to find multiplicative inverse in mod 20, divide both sides by mod 20.

  $3 * (7) \bmod 20 + 20 * (-1) \bmod 20 = 1 \bmod 20$

  $3 * (7) \bmod 20 + 0 = 1 \bmod 20$

  $3 * (7) \bmod 20 = 1 \bmod 20$

- Hence, multiplicative inverse is 3.
- Therefore, the value of decrypting key, $d = 3$.
- Hence, the Private Key of user Alice is (3, 33).

- **For User B :**

  Public key is (13, 221). Hence, n = 221 and e = 13, where e is the encrypting key. Let's calculate B's private key which would give us *d*, decrypting key.

  Assume that 13 and 17 were the chosen prime factors for n = 221.

  $n = p * q = 13 * 17 = 221$

  $r = (p - 1) * (q - 1) = (13 - 1) * (17 - 1)$

  $r = 12 * 16 = 192$

  $d = e^{-1} \bmod r$

  $d \equiv 1 \bmod 192$

  $13d \equiv 1 \bmod 192$

Let's calculate modulo inverse using extended Euclidean algorithm (swapping 13 and 192)

Index i	quotient q for i	Remainder r for i	s for i	t for i
0		192	1	0
1		13	0	1
2	192 / 13 = 14	192 – 13 * 14 = 10	1 – 0*14 = 1	0 – 1*14 = -14
3	13 / 10 = 1	13 – 10 * 1 = 3	0 – 1 *1 = -1	1 – (-14)*1 = 15
4	10 / 3 = 3	10 – 3 * 3 = 1	1 - (-1) * 3 = 4	-14 - 15 * 3 = -59
5	3 / 1 = 3	3 – 1*3 = 0	Do not calculate	Do not calculate

- Let's re-swap the values.

  x = – 59

  y = 4

- Putting the values in the extended Euclidean algorithm, you get

  ax + by = 1

  13 * (– 59) + 192 * (4) = 1

- Since, you have to find multiplicative inverse in mod 192, divide both sides by mod 192.

  13 * (– 59) mod 192 + 192 *(4) mod 192 = 1 mod 192

  13 * (– 59) mod 192 + 0 = 1 mod 192

  13 * (– 59) mod 192 = 1 mod 192

- Recall our discussion on calculating mod for negative numbers. You need to keep adding mod until the number turns positive and then calculate mod on the positive number you got.

  – 59 + 192 = 136

  136 mod 192 = 136

- Hence, multiplicative inverse is 136.
- Therefore, the value of decrypting key, $d = 136$.
- Hence, the Private Key of user B is (136, 221).
- Now, the question isn't asking you to encrypt and decrypt the plaintext message 5. It is asking you to show the message signing and verification using RSA digital signature. Hence, do not encrypt the plaintext message 5. It is not required.
  - As per RSA digital signature,
  - Message Signing
  - To sign a message, M
- Calculate the hash value of the message M at sender's end

  h = hash(M)

- Encrypt *h* using RSA private key
- Signature S = $(h)^d$ mod n
- Signature Verification
  - Decrypt Signature S using public key
  - h' = (S)e mod n
  - Calculate the hash value of the message M at receiver's end
  - h = hash(M)
- If *h = h'*, the signature is valid else the signature is invalid
- Since, hash algorithm or the hash value of the message to be digitally signed is not given, let's assume that the hash value *h* of the plaintext message 5 is 12. Let's create and verify the digital signature using the RSA digital signature scheme.
- To sign a message, M
  - Encrypt h using RSA private key
  - Signature $S = (h)^d \ mod \ n$
- User A wishes to send the message to B. Hence, you would use A's private key (3, 33) for encrypting the hash value of the message. The encryption key is 3 from the private key. We have assumed that 12 is the hash value of the plaintext message 5.
  - $S = 12^3$ mod 33
  - S = 12
- User A would send the message and its signature S = 12 to user B.
- Now, user B can verify the signature as
  - Decrypt Signature *S* using public key
  - $h' = (S)^e \ mod \ n$
- The public key of the user A is (7, 33).
  - $h' = 12^7$ mod 33
  - h' = 12
- You find that h = h' = 12. Hence, the digital signature is verified.

2.19.2 Schnorr Digital Signature Scheme

✍ ***Definition :*** *Schnorr signatures are based on discrete logarithm problems.*

Schnorr signatures were developed by Claus P Schnorr and subsequently protected by U.S. Patent until late 2008. As a result of the patent, Schnorr signatures have not been standardized or widely used in crypto libraries today. It is believed to be a more elegant signature solution with a simple mathematical proof.

2.19.2(A) Key Generation

- Choose a private signing key, x
- The public verification key is $y = g^x$ where g is a generator point.

2.19.2(B) Message Signing

To sign a message, M

- Choose a random value k
- Let $r = g^k$
- Let e = Hash (r|| M) e
- Let S = k – xe

The signature is the pair(s, e).

2.19.2(C) Signature Verification

- Let $r_v = g^s y^e$
- Let e_v = Hash(r_v|| M)

If $e_v = e$, then the signature is verified.

2.19.3 ElGamal Digital Signature Scheme

✍ ***Definition** : The ElGamal digital signature scheme is based on the difficulty of computing discrete logarithms.*

It was conceived by Taher ElGamal in 1984. It is not used widely in the industry today.

Suppose a message m needs to be signed. Following are the set of steps in the scheme.

2.19.3(A) Key Generation

- Choose a large prime p and a primitive root α.
- Choose a secret integer z and calculate $\beta \equiv \alpha z \pmod p$.
- The values of p, α and β are made public and z is kept private.

2.19.3(B) Message Signing

- Select a secret random integer k such that $GCD(k, p - 1) = 1$.
- Compute $r \equiv \alpha^k \pmod p$.
- Compute $s \equiv k^{-1}(m - zr) \pmod{p - 1}$.
- The signed message is the triplet (m, r, s).

2.19.3(C) Signature Verification

- Compute $v_1 \equiv \beta^{r^s} \pmod p$ and $v_2 \equiv \alpha^m \pmod p$.
- The signature is declared valid if $v_1 \equiv v_2 \pmod p$.

2.19.4 Digital Signature Standard (DSS)

✍ ***Definition*** *: National Institute of Standards and Technology (NIST) defined the Digital Signature Standard to provide approved techniques for generating and validating digital signatures for authenticating messages (or any binary data in general).*

It approved three techniques as part of the standard.

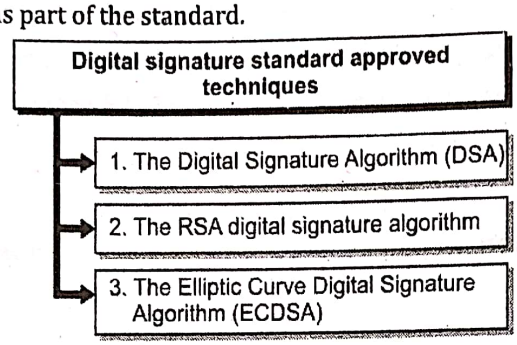


Fig. 2.19.1 : Techniques of digital signature

Out of the three currently listed techniques, DSA was the primary and the original proposal in the standard. Let's learn about it in brief.

2.19.4(A) Digital Signature Algorithm (DSA)

✍ ***Definition*** *: The Digital Signature Algorithm (DSA) is based on the difficulty of computing discrete logarithms.*

It is based on ElGamal and Schnorr digital signature schemes.

2.19.4(B) Key Generation

- p is a prime number
- q is a prime divisor of $(p - 1)$
- $g = h^{(p-1)/q} \bmod p$ where h is any integer with $1 < h < (p - 1)$
- x is user's private key
- y is user's public key

2.19.4(C) Message Signing

- M is message to be signed
- H(M) = SHA1(M)
- $r = (g^k \bmod p) \bmod q$
- $s = [k^{-1} (H(M) + xr)] \bmod q$
- *Signature* = (r, s)

2.19.4(D) Signature Verification

- *Assume* M', r', s' are as received at the receiver's end
- $w = (s')^{-1} \bmod q$
- $u_1 = [H(M')w] \bmod q$
- $u_2 = (r')w \bmod q$
- $v = [(g^{u1} y^{u2}) \bmod p] \bmod q$
- *v should match r'*

2.20 Authentication Applications (Remote user Authentication Protocols)

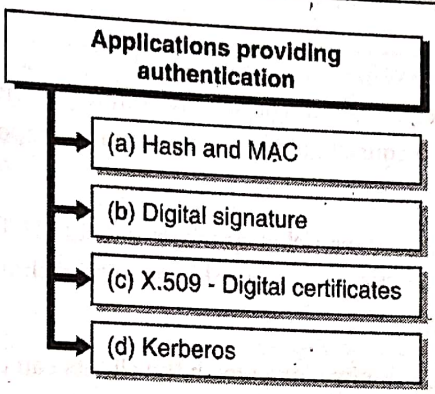


Fig. 2.20.1 : Application providing authentication

Authentication is a way to build trust in communicating parties or systems. Authentication serves as the crucial protection mechanism towards securing a communication – ensuring that the right set of parties are involved in the communication and no one else.

Broadly speaking, there are four ways that provide major authentication services today. You have already learnt about the other 3 earlier. Let's now learn about Kerberos.

2.20.1 Kerberos

✍ ***Definition :*** *Kerberos is a network-based authentication protocol.*

It was developed around mid-1980s at MIT. It works on the client/server model and uses symmetric key cryptography. Today, Kerberos is extensively used for authentication in Microsoft Windows, Unix, Linux and Apple OS.

2.20.2 Problems Addressed by Kerberos

1. **Sharing passwords over the network**

   With Kerberos, you need not share passwords over the network for authentication.

2. **Establishing trust between two parties**

   Kerberos acts as a third party and helps to establish trust between two non-trusting parties.

3. **Difficult to spoof authentication**

Kerberos employs several mechanisms to secure the authentication process and makes it difficult to spoof authentication.

4. **Scalable**

Kerberos supports a large number of servers and clients and hence is suitable for distributed network architectures.

2.20.3 Components of Kerberos

The Kerberos environment has several components that are required for functioning.

1. **Key Distribution Center (KDC)**

KDC is the most important component in the Kerberos environment. It holds all the information required for the functionality of Kerberos. Basically, it consists of the following sub-components :

(a) Authentication Service (AS)

The Authentication Service (AS) issues ticket-granting tickets (TGTs) that is used to connect to the Ticket Granting Service (TGS). It also verifies principals that require authentication.

(b) Ticket Granting Service (TGS)

The TGS issues the tickets to the clients using which the clients can connect to the desired server.

(c) Principals

In the Kerberos terminology, Principals can be users, clients, servers, applications or network services that require authentication. The KDC has the information about each principal account and its secret key. For example, a user could be a principal requiring access to a print server that could be another principal.

(d) KDC Database

All the principal related information is stored in the KDC database.

(e) Realm

In Kerberos terminology, a realm is the set of principals who can authenticate to each other. It is a logical grouping of principals. One KDC can have one realm or several realms.

2. **Client**

Typically, a client is the principal that requires to authenticate to another principal (server).

3. **Server**

Typically, a server is the principal that holds resources that the client is interested in and provides the resources that can be consumed after successful authentication.

2.20.4 How does it Work ?

Q. Two users wish to establish a secure communication channel and exchange a session key after mutual authentication. Show how this can be done with the help of a KDC. **MU - May 19, 10 Marks**

Q. Show how Kerberos protocol can be used to achieve single sign-on in distributed systems. **MU - May 19, 10 Marks**

Kerberos heavily uses the concept of tickets that works very much like your train ticket. A ticket is just a temporary proof. Let's understand the working in detail.

(a) Scenario 1 : User authenticating to a computer

(i) The user enters the username and password on the computer.

(ii) The Kerberos software running on the computer sends the username to the Authentication Service (AS).

(iii) The Authentication Service checks if the username is present. If yes, it sends back a ticket-granting ticket (TGT) which is encrypted with the user's pre-shared secret key (password).

(iv) If the user entered the correct password, she can decrypt the TGT and then is granted access to the computer.

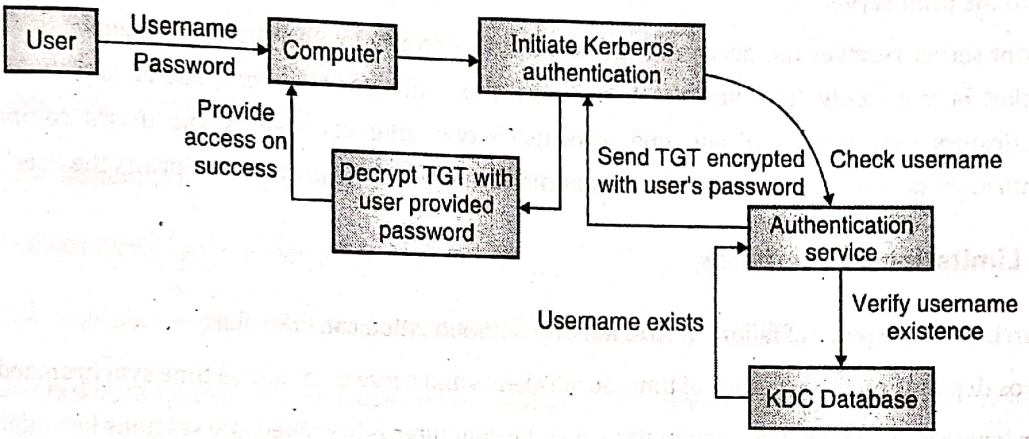


Fig. 2.20.2 : User authentication to computer

(b) Scenario 2 : Authenticated User now wants to print a document

From Scenario 1, the user has successfully authenticated to her computer. Now, suppose that she wants to print a document and hence needs to authenticate to the print server.

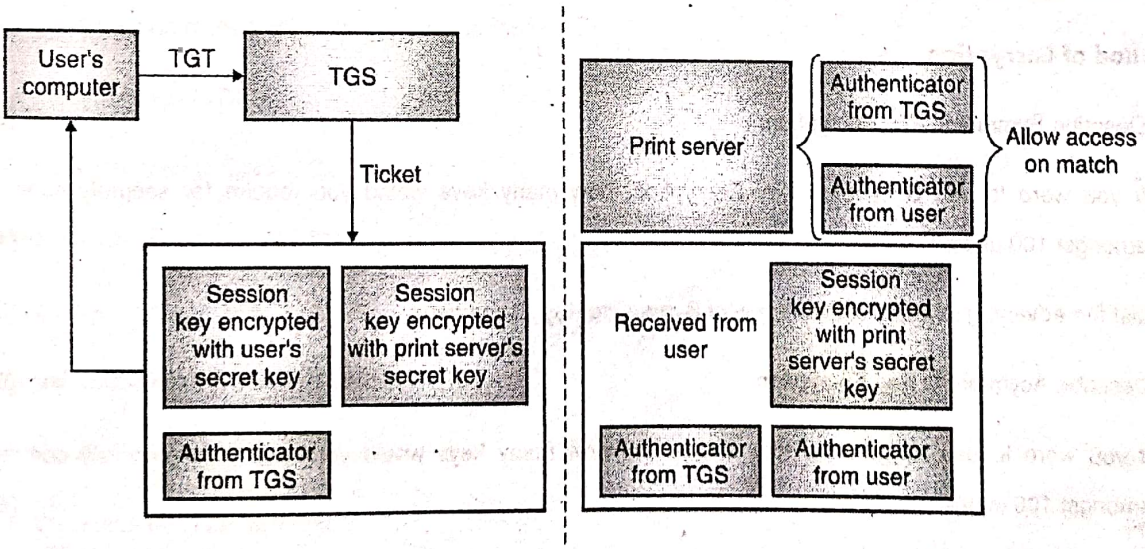


Fig. 2.20.3 : Authenticated uses accessing an authorized resource

The following steps are taken :

(i) The computer sends the TGT it received earlier (when the user wanted to authenticate) to the TGS. The TGT is a proof for the TGS that the user has already been successfully authenticated (as in scenario 1).

(ii) TGS creates a new ticket and puts two copies of the same session key (temporary secret key) in it. It encrypts the first copy of the session key with the user's secret key and the second copy with print server's secret key. This ticket also contains an authenticator information that holds the value of the user's computer's IP Address, sequence number and timestamp from where the TGT came. It then sends the ticket to the user's computer.

(iii) The user decrypts the ticket created by the TGS with her secret key and obtains the session key. It adds another set of authenticator information (computer's IP Address, sequence number and timestamp) to it and sends the ticket to the print server.

(iv) The print server receives the ticket and extracts the session key by decrypting it. It knows that the KDC created the ticket because only KDC had the knowledge about the print server's secret key. It also gets the two authenticators (one from TGS and one from user) that uniquely identify the user's computer. If the two authenticators match, the user is successfully authenticated, and the print server prints the user's document.

2.20.5 Limitations of Kerberos

1. KDC can be a single point of failure. If KDC fails, no authentication can take place.
2. Kerberos depends on the accuracy of time. So, all clients and servers should be time synchronized.
3. Session keys are stored on user's computer. So, if the computer is breached, the sessions key might be stolen.

Review Questions

Here are a few review questions to help you gauge your understanding of this chapter. Try to attempt these questions and ensure that you can recall the points mentioned in the chapter.

[A] Method of Encryption

Q. 1 Describe Symmetric Key Encryption. **[6 Marks]**

Q. 2 If you were to use Symmetric Key Encryption, how many keys would you require for securely communicating amongst 100 users? **[4 Marks]**

Q. 3 List the advantages and disadvantages of Symmetric Key Encryption. **[5 Marks]**

Q. 4 Describe Asymmetric Key Encryption. **[6 Marks]**

Q. 5 If you were to use Asymmetric Key Encryption, how many keys would you require for securely communicating amongst 100 users? **[4 Marks]**

Q. 6 List the advantages and disadvantages of Asymmetric Key Encryption. **[5 Marks]**

Q. 7 Compare between Symmetric and Asymmetric Keys encryption. **[4 Marks]**

Q. 8 Describe the various attacks possible on Cryptosystems. [6 Marks]

Q. 9 Compare differential and linear cryptanalysis. [6 Marks]

[B] Types of Symmetric Algorithms

Q. 10 With a suitable diagram and example, explain what is a block cipher? [4 Marks]

Q. 11 With a suitable diagram and example, explain what is a stream cipher? [4 Marks]

Q. 12 Compare block and stream cipher. [4 Marks]

[C] Data Encryption Standard (DES)

Q. 13 List the Major attributes of DES. [4 Marks]

Q. 14 List the Block Cipher Design Principles. [6 Marks]

Q. 15 Draw a block diagram of DES and explain. [6 Marks]

Q. 16 Describe various Block cipher modes of operation. [8 Marks]

Q. 17 With a suitable diagram, explain Electronic Code Book (ECB) Mode of block cipher operation. [4 Marks]

Q. 18 With a suitable diagram, explain Cipher Block Chaining (CBC) Mode of block cipher operation. [4 Marks]

Q. 19 With a suitable diagram, explain Cipher Feedback (CFB) Mode of block cipher operation. [4 Marks]

Q. 20 With a suitable diagram, explain Output Feedback (OFB) Mode of block cipher operation. [4 Marks]

Q. 21 With a suitable diagram, explain Counter (CTR) Mode of block cipher operation. [4 Marks]

Q. 22 Compare various modes of block cipher operations. [8 Marks]

Q. 23 List the weakness in DES. [4 Marks]

Q. 24 Write a short note on Double DES. [4 Marks]

Q. 25 Write a short note on Triple DES. [4 Marks]

Q. 26 Write a short note on 3DES. [4 Marks]

[D] Advanced Encryption Standard (AES)

Q. 27 What is AES? List its major attributes. [4 Marks]

Q. 28 List the evaluation criteria for AES. [4 Marks]

Q. 29 Draw the block diagram of AES and explain its working. [6 Marks]

Q. 30 Compare AES and DES. Which one would you use and why? [6 Marks]

[E] RC5 Algorithm

Q. 31 List the major attributes of RC5 algorithm. [4 Marks]

Q. 32 Explain RC5 algorithm details. [6 Marks]

[F] Public Key Cryptography

Q. 33 List the principles of public key cryptosystems. [6 Marks]

Q. 34 Take an example of your choice and work it through explaining RSA. [8 Marks]

Q. 35 Explain the various attacks on RSA. [6 Marks]

Q. 36 Perform encryption and decryption using RSA algorithm with p = 7, q = 11, e = 17 and M = 8. [6 Marks]

Q. 37 Given modulus n=91 and public key, e=5, find the values of p, q, phi(n), and d using RSA. Encrypt M=25. Also perform decryption. [6 Marks]

Q. 38 Given modulus n=221 and public key, e=7, find the values of p, q, phi(n), and d using RSA. Encrypt M = 5. [6 Marks]

[G] Message Authentication Requirements

Q. 39 Why do you need message authentication? [4 Marks]

[H] Message Authentication Functions

Q. 40 Give an overview of Message Authentication Functions. [6 Marks]

[I] Cryptographic Hash Functions

Q. 41 Write a short note on hashing. [4 Marks]

Q. 42 With suitable diagrams, explain how a hash function works. [8 Marks]

Q. 43 Describe the characteristics of hash functions. [8 Marks]

Q. 44 Explain SHA1. [8 Marks]

Q. 45 Explain SHA3. [8 Marks]

Q. 46 Write a short note on MD4. [4 Marks]

Q. 47 Explain MD5. [8 Marks]

Q. 48 Compare SHA and MD5. Which one would you suggest using and why? [6 Marks]

Q. 49 Explain Security of hash functions and possible attacks on them. [6 Marks]

Q. 50 Write a short note on MAC (Message Authentication Code). [4 Marks]

Q. 51 With suitable diagrams, explain how MAC (Message Authentication Code) works. [8 Marks]

Q. 52 Describe the various types of MAC (Message Authentication Code). [8 Marks]

Q. 53 With a suitable diagram, explain HMAC. [8 Marks]

Q. 54 With a suitable diagram, explain CBC-MAC. [6 Marks]

Q. 55 With a suitable diagram, explain CMAC. [6 Marks]

Q. 56 Compare HMAC, CBC-MAC and CMAC. [6 Marks]

Q. 57 Compare Hash, MAC and Digital Signature. [6 Marks]

Q. 58 Explain Security of MAC and possible attacks on them. [6 Marks]

[J] Digital Signature

Q. 59 Write a short note on X.509. [4 Marks]

Q. 60 Describe the format of a X.509 Certificate. [6 Marks]

Q. 61 Write a short note on digital signature and also list its usage. [4 Marks]

Q. 62 With suitable diagrams, explain how digital signature works. [8 Marks]

Q. 63 List the properties of digital signature. [4 Marks]

Q. 64 List the steps for key generation, message signing and signature verification in RSA Digital Signature Scheme. [8 Marks]

Q. 65 List the steps for key generation, message signing and signature verification in Schnorr Digital Signature Scheme. [8 Marks]

Q. 66 List the steps for key generation, message signing and signature verification in ElGamal Digital Signature Scheme. [8 Marks]

Q. 67 List the steps for key generation, message signing and signature verification in Digital Signature Algorithm (DSA). [8 Marks]

Q. 68 Alice chooses public key as (7, 33) and B chooses public key as (13, 221). Calculate their private keys. A wishes to send message m=5 to B. Show the message signing and verification using RSA digital signature. [10 Marks]

[K] Kerberos

Q. 68 Write a short note on Kerberos. **[4 Marks]**

Q. 69 List the problems addressed by Kerberos. **[4 Marks]**

Q. 70 Describe the core components of Kerberos. **[6 Marks]**

Q. 71 With suitable diagrams, explain how Kerberos works. **[8 Marks]**

Q. 72 Explain the limitations of Kerberos. **[4 Marks]**

□□□