Sarvasiddhant Education Society's
SWAMINARAYAN SIDDHANTA INSTITUTE OF TECHNOLOGY
Affiliated to Rashtrasant Tukdoji Maharaj Nagpur University
Nagpur Katol Highway Road, Khapri (Kothe),
Tal.Kalmeshwar, Nagpur, Maharashtra 441501
Department Of Computer Engineering
Session 2023-2024

CNS NOTES

CRYPTOGRAPHY & NETWORK SECURITY

SEMESTER: 7TH SEM (FINAL YEAR)

Subject Incharge: Prof. Ashvini Bais

Asst. Prof. (CE Dept.)

NOTES

CRYPTOGRAPHY & NETWORK SECURITY

Semester: 7th Sem (Final Year)

Rashtrasant Tukadoji Maharaj Nagpur University, Nagpur
FOUR YEAR B. TECH. COURSE
(Revised Curriculum as per AICTE Model Curriculum)
B.Tech VII Semester (Computer Engineering) Scheme & Syllabus

Seventh Semester:-

S. N.	Subject Code	Subject	Teaching Scheme			Evaluation Scheme			Credits	Minimum Passing Marks
			L	T	P	CA	UE	Total		
1	BTCME701T	Cryptography & Network Security	3	1	-	30	70	100	4	45
2	BTCME701P	Cryptography & Network Security-Lab	-	-	2	25	25	50	1	25
3	BTCME702T	Elective – IV	3	-	-	30	70	100	3	45
4	BTCME703T	Elective – V	3	-	-	30	70	100	3	45
5	BTCME704T	Open Elective-II	3	-	-	30	70	100	3	45
6	BTCME705P	Project Work Phase -I	-	-	6	50	50	100	3	50
7	BTCME706P	Report Writing Activity	-	-	2	-	-	-	Audit	Grade
		Total	**12**	**01**	**10**	**195**	**355**	**550**	**17**	

Elective IV: -
1. Deep Learning
2. Block chain Technology
3. Augmented & Virtual Reality
4. Salesforce Technology

Elective V: -
1. Compiler Design
2. Natural Language Processing
3. Introduction to Software Testing

Open Electives:
1. Joy of Computing using Python
2. Data Base Management System
3. Data Visualization

RASHTRASANT TUKADOJI MAHARAJ NAGPUR UNIVERSITY, NAGPUR
FOUR YEAR BACHELOR OF TECHNOLOGY (B.TECH.) DEGREE COURSE
SEMESTER: SEVENTH (CBCS)
BRANCH: Computer Engineering
Subject : Cryptography and Network Security

Subject : Cryptography and Network Security **Subject Code BTCME701T**

Load	Credit	Total Marks	Internal Marks	University Marks	Total
04Hrs (Theory)	03(L)+01(T)	100	30	70	100

Aim :To highlight the features of different technologies involved in Network Security.

Prerequisite(s): Mathematics, Algorithm, Networking

Course Objectives:

1	To develop the student's ability to understand the concept of security goals in various applications and learn classical encryption techniques
2	Apply fundamental knowledge on cryptographic mathematics used in various symmetric and asymmetric key cryptography
3	To develop the student's ability to analyze the cryptographic algorithms.
4	To develop the student's ability to analyze the cryptographic algorithms.

Course Outcomes:
At the end of this course student are able to:

CO1	To understand basics of Cryptography and Network Security and classify the symmetric encryption techniques.
CO2	Understand, analyze and implement the symmetric key algorithm for secure transmission of data.
CO3	Acquire fundamental knowledge about the background of mathematics of asymmetric key cryptography and understand and analyze asymmetric key encryption algorithms and digital signatures.
CO4	Analyze the concept of message integrity and the algorithms for checking the integrity of data.
CO5	To understand various protocols for network security to protect against the threats in the networks.

UNIT-I **[08 Hrs]**

Introduction, Model for network security. Mathematics of cryptography: modular arithmetic, Euclidean and extended Euclidean algorithm. Classical encryption techniques: substitution techniques-Caesar cipher, Vigenere's ciphers, Playfair ciphers and transposition techniques.

UNIT-II **[07 Hrs]**

Symmetric key cryptography: Block Cipher Principles, Data Encryption Standard (DES), Triple DES, Advanced Encryption Standard (AES), RC4, Key Distribution.

UNIT III **[07 Hrs]**

Asymmetric key cryptography: Euler's Totient Function, Fermat's and Euler's Theorem, Chinese Remainder Theorem, RSA, Diffie Hellman Key Exchange, ECC, Entity authentication: Digital signature.

UNIT IV **[07 Hrs]**

Message Integrity and authentication: Authentication Requirements and Functions, Hash Functions, MD5, Kerberos, Key Management, X.509 Digital Certificate format.

UNIT V **[07 Hrs]**

Network Security: PGP, SSL, Firewalls, IDS, Software Vulnerability: Phishing, Buffer Overflow, SQL Injection, Electronic Payment Types

Text Book:

1. William Stallings, "Cryptography and Network Security: Principles and Standards", Prentice Hall India, 7th Edition, 2017.
2. Bernard Menezes, "Network Security and Cryptography", Cengage Learning, 2010.

Reference Books:

1. Robert Brugge, Mark Rhodes, Heithstruggberg "Network Security, The Complete Reference", Tata McGraw Hill Publication, 2004.
2. Behrouz A. Forouzen, "Cryptography and Network Security", McGraw-Hill publication, 2nd Edition, 2010
3. Bruce Schneier, Applied Cryptography, John Wiley New York, 2nd Edition, 1996.

UNIT IV

Message Integrity and Authentication: Authentication requirements and Functions, Hash Functions, MD5, Kerberos, Key Management, X.509 Digital Certificate Format.

Message Integrity and Authentication

Authentication-

Authentication is the act of confirming the truth of an attribute of a single piece of data claimed true by an entity. In contrast with identification, which refers to the act of stating or otherwise indicating a claim purportedly attesting to a person or thing's identity, authentication is the process of actually confirming that identity. It might involve confirming the identity of a person by validating their identity documents, verifying the authenticity of a website with a digital certificate, determining the age of an artifact by carbon dating, or ensuring that a product is what its packaging and labeling claim to be. In other words, authentication often involves verifying the validity of at least one form of identification.

Message authentication is a procedure to verify that received messages come from the alleged source and have not been altered. Message authentication may also verify sequencing and timeliness. A digital signature is an authentication technique that also includes measures to counter repudiation by either source or destination. Any message authentication or digital signature mechanism can be viewed as having fundamentally two levels. At the lower level, there must be some sort of function that produces an authenticator: a value to be used to authenticate a message. This lower level function is then used as primitive in a higher-level authentication protocol that enables a receiver to verify the authenticity of a message.

The authentication service is concerned with assuring that a communication is authentic. In the case of a single message, such as a warning or alarm signal, the function of the authentication service is to assure the recipient that the message is from the source that it claims to be from.

In the case of an ongoing interaction, such as the connection of a terminal to a host, two aspects are involved.

- First, at the time of connection initiation, the service assures that the two entities are authentic, that is, that each is the entity that it claims to be.
- Second, the service must assure that the connection is not interfered with in such a way that a third party can masquerade as one of the two legitimate parties for the purposes of unauthorized transmission or reception.

Two specific authentication services are defined in X.800:

- **Peer entity authentication:** Provides for the corroboration of the identity of a peer entity in an association. Two entities are considered peers if they implement to same protocol in different systems; e.g., two TCP modules in two communicating systems. Peer entity authentication is provided for use at the establishment of, or at times during the data transfer phase of, a connection. It attempts to provide confidence that an entity is not performing either a masquerade or an unauthorized replay of a previous connection.
- **Data origin authentication:** Provides for the corroboration of the source of a data unit. It does not provide protection against the duplication or modification of data units. This type of service supports applications like electronic mail, where there are no prior interactions between the communicating entities. Access Control In the context of network security, access control is the ability to limit and control the access to host systems and applications via communications links. To achieve this, each entity trying to gain access must first be identified, or authenticated, so that access rights can be tailored to the individual.

Message Authentication Requirements-

- **Revelation:** It means releasing the content of the message to someone who does not have an appropriate cryptographic key.
- **Analysis of Traffic:** Determination of the pattern of traffic through the duration of connection and frequency of connections between different parties.
- **Deception:** Adding out of context messages from a fraudulent source into a communication network. This will lead to mistrust between the parties communicating and may also cause loss of critical data.
- **Modification in the Content:** Changing the content of a message. This includes inserting new information or deleting/changing the existing one.
- **Modification in the sequence:** Changing the order of messages between parties. This includes insertion, deletion, and reordering of messages.
- **Modification in the Timings:** This includes replay and delay of messages sent between different parties. This way session tracking is also disrupted.
- **Source Refusal:** When the source denies being the originator of a message.
- **Destination refusal:** When the receiver of the message denies the reception.

Message Authentication Functions-

All message authentication and digital signature mechanisms are based on two functionality levels:

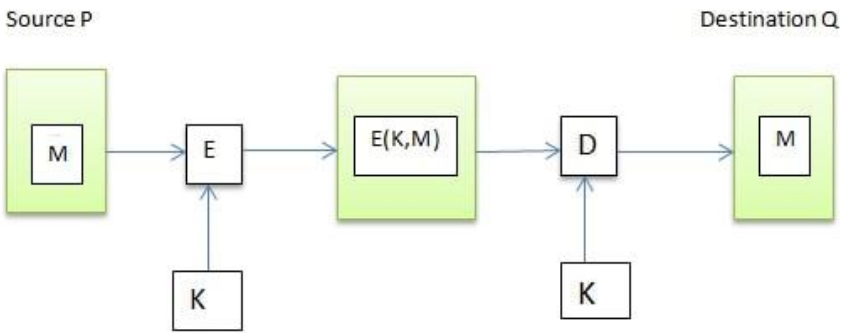
- **Lower level:** At this level, there is a need for a function that produces an authenticator, which is the value that will further help in the authentication of a message.

- **Higher-level:** The lower level function is used here in order to help receivers verify the authenticity of messages.

These message authentication functions are divided into three classes:

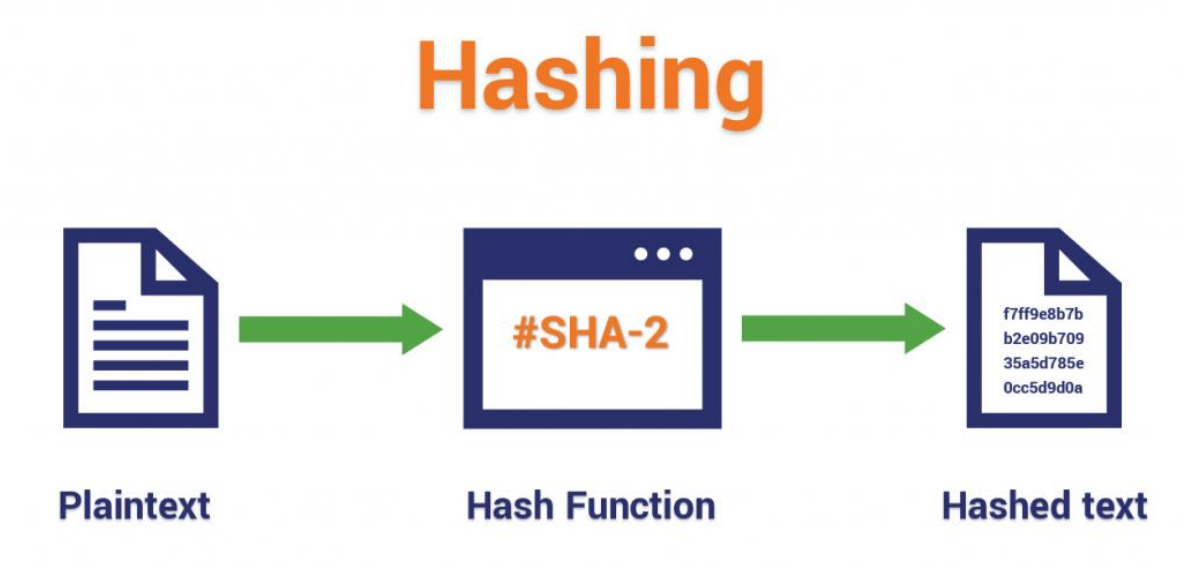
- **Message encryption:** While sending data over the internet, there is always a risk of a Man in the middle (MITM) attack. A possible solution for this is to use message encryption. In message encryption, the data is first converted to a ciphertext and then sent any further. Message encryption can be done in two ways:

- **Symmetric Encryption:** Say we have to send the message M from a source P to destination Q. This message M can be encrypted using a secret key K that both P and Q share. Without this key K, no other person can get the plain text from the ciphertext. This maintains confidentiality. Further, Q can be sure that P has sent the message. This is because other than Q, P is the only party who possesses the key K and thus the ciphertext can be decrypted only by Q and no one else. This maintains authenticity. At a very basic level, symmetric encryption looks like this:



Hash Functions-

Hash functions are mathematical functions that transform or "map" a given data set into a bit string of fixed size, also known as the "hash value." Hash functions are used in cryptography and have variable levels of complexity and difficulty. Hash functions are extremely useful and appear in almost all information security applications. A hash function is a mathematical function that converts a numerical input value into another compressed numerical value. The input to the hash function is of arbitrary length but output is always of fixed length.



Features of Hash Functions-

- **Fixed Length Output (Hash Value)**
  - Hash function coverts data of arbitrary length to a fixed length. This process is often referred to as hashing the data.
  - In general, the hash is much smaller than the input data, hence hash functions are sometimes called compression functions.
  - Since a hash is a smaller representation of a larger data, it is also referred to as a digest.
  - Hash function with n bit output is referred to as an n-bit hash function. Popular hash functions generate values between 160 and 512 bits.

- **Efficiency of Operation**
  - Generally for any hash function h with input x, computation of h(x) is a fast operation.
  - Computationally hash functions are much faster than a symmetric encryption.

Properties of Hash Functions-

- **Pre-Image Resistance**
  - This property means that it should be computationally hard to reverse a hash function.
  - In other words, if a hash function h produced a hash value z, then it should be a difficult process to find any input value x that hashes to z.
  - This property protects against an attacker who only has a hash value and is trying to find the input.

- **Second Pre-Image Resistance**

- This property means given an input and its hash, it should be hard to find a different input with the same hash.
- In other words, if a hash function h for an input x produces hash value h(x), then it should be difficult to find any other input value y such that h(y) = h(x).
- This property of hash function protects against an attacker who has an input value and its hash, and wants to substitute different value as legitimate value in place of original input value.

- **Collision Resistance**
  - This property means it should be hard to find two different inputs of any length that result in the same hash. This property is also referred to as collision free hash function.
  - In other words, for a hash function h, it is hard to find any two different inputs x and y such that h(x) = h(y).
  - Since, hash function is compressing function with fixed hash length, it is impossible for a hash function not to have collisions. This property of collision free only confirms that these collisions should be hard to find.
  - This property makes it very difficult for an attacker to find two input values with the same hash.
  - Also, if a hash function is collision-resistant then it is second pre-image resistant.

- **Determinism** — A hash algorithm should be deterministic, meaning that it always gives you an output of identical size regardless of the size of the input you started with. This means that if you're hashing a single sentence, the resulting output should be the same size as one you'd get when hashing an entire book.

- **Avalanche Effect** — What this means is that any change made to an input, no matter how small, will result in a massive change in the output. Essentially, a small change (such as adding a comma) snowballs into something much larger, hence the term "avalanche effect."

-
- **Hash Speed** — Hash algorithms should operate at a reasonable speed. In many situations, hashing algorithms should compute hash values quickly; this is considered an ideal property of a cryptographic hash function. However, this property is a little more subjective. You see, faster isn't always better because the speed should depend on how the hashing algorithm is going to be used. Sometimes, you want a faster hashing algorithm, and other times it's better to use a slower one that takes more time to run through. The former is better for website connections and the latter is better for password hashing.

What Does a Hash Function Do?

One purpose of a hash function in cryptography is to take a plaintext input and generate a hashed value output of a specific size in a way that can't be reversed. But they do more than that from a

10,000-foot perspective. You see, hash functions tend to wear a few hats in the world of cryptography. In a nutshell, strong hash functions:

- **Ensure data integrity**

  Hash functions are a way to ensure data integrity in public key cryptography. What I mean by that is that hash functions serve as a check-sum, or a way for someone to identify whether data has been tampered with after it's been signed. It also serves as a means of identity verification.

- **Secure against unauthorized modifications**
  - One of the best aspects of a cryptographic hash function is that it helps you to ensure data integrity. But if you apply a hash to data, does it mean that the message can't be altered? No. But what it does is inform the message recipient that the message has been changed. That's because even the smallest of changes to a message will result in the creation of an entirely new hash value.
  - Think of hashing kind of like you would a smoke alarm. While a smoke alarm doesn't stop a fire from starting, it does let you know that there's danger before it's too late

- **Protect stored passwords**

  But password hashes on their own isn't enough to protect you against certain types of attacks, including brute force attacks. This is why you first need to add a salt. A salt is a unique, random number that's applied to plaintext passwords before they're hashed. This provides an additional layer of security and can protect passwords from password cracking methods like rainbow table attacks. (Keep an eye out for our future article on rainbow tables in the next few weeks.)

- **Operate at different speeds to suit different purposes**

  It's also important to note that hash functions aren't one-size-fits-all tools. As we mentioned earlier, different hash functions serve different purposes depending on their design and hash speeds. They work at different operational speeds — some are faster while others are much slower. These speeds can aid or impede the security of a hashing algorithm depending on how you're using it. So, some fall under the umbrella of secure hashing algorithms while others do not.

Applications of Hash Functions-

There are two direct applications of hash function based on its cryptographic properties.

1. **Password Storage**

- Instead of storing password in clear, mostly all logon processes store the hash values of passwords in the file.
- The Password file consists of a table of pairs which are in the form (user id, h(P)).

2. **Cryptography**: These are used in cryptography to ensure the confidentiality and integrity of data. They generate digital signatures, which verify the authenticity of a message or document. Hash functions also create message digests, ensuring data integrity during transmission.

3. **Data fingerprinting**: These uniquely identify data, such as file-sharing networks. Generating a hash value for a piece of data makes it possible to identify and ensure it is uniquely safe.

4. **Blockchain**: These are extensively popular in blockchain technology. Each block in a blockchain contains a hash value of the previous block, ensuring the entire blockchain's integrity. Additionally, they also mine new partnerships in a blockchain.

5. **Digital forensics**: These are popular in digital forensics to ensure the authenticity of evidence. Generating hash values for evidence makes it possible to ensure the evidence is safe.

6. **Data indexing**: These create indexes for large data sets. This allows for quick retrieval of data, even from extensive databases.

7. **Data Integrity Check:** It is the most common application of the hash functions. It is used to generate the checksums on data files. This application provides assurance to the user about the correctness of the data.

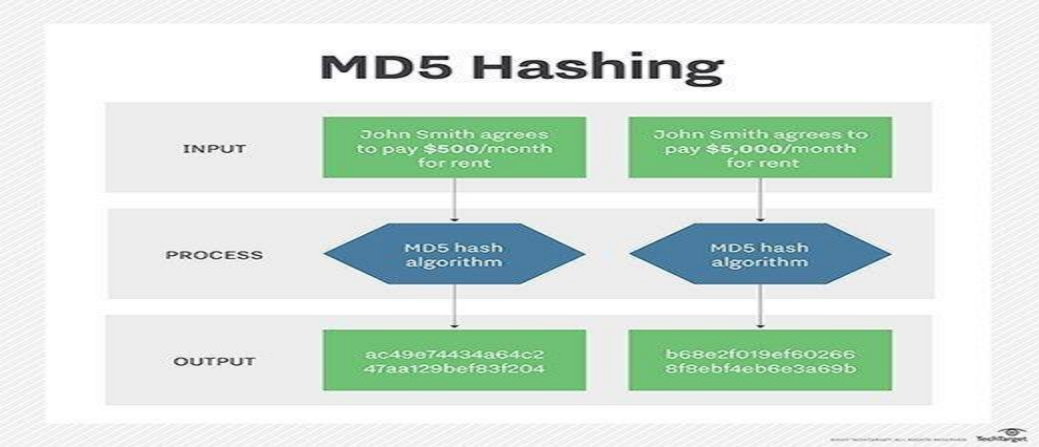
Message Digest 5 (MD5)-

The MD5 hashing algorithm is a one-way cryptographic function that accepts a message of any length as input and returns as output a fixed-length digest value to be used for authenticating the original message. The MD5 hash function was originally designed for use as a secure cryptographic hash algorithm for authenticating digital signatures. MD5 has been deprecated for uses other than as a non-cryptographic checksum to verify data integrity and detect unintentional data corruption. Although originally designed as a cryptographic message authentication code algorithm for use on the internet, MD5 hashing is no longer considered reliable for use as a cryptographic check sum because researchers have demonstrated techniques capable of easily generating MD5 collisions on commercial off-the-shelf computers.

Ronald Rivest, founder of RSA Data Security and institute professor at MIT, designed MD5 as an improvement to a prior message digest algorithm, MD4. Describing it in Internet Engineering Task Force RFC 1321, "The MD5 Message-Digest Algorithm," he wrote:

The algorithm takes as input a message of arbitrary length and produces as output a 128-bit 'fingerprint' or 'message digest' of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given pre-specified target message digest. The MD5 algorithm is intended for digital signature applications, where a large file must be 'compressed' in a secure manner before being encrypted with a private (secret) key under a public-key cryptosystem such as RSA.

The IETF suggests MD5 hashing can still be used for integrity protection, noting "Where the MD5 checksum is used in line with the protocol solely to protect against errors, an MD5 checksum is still an acceptable use." However, it added that "any application and protocol that employs MD5 for any purpose needs to clearly state the expected security services from their use of MD5."



Message Digest algorithm characteristics:

Message digests, also known as hash functions, are one-way functions; they accept a message of any size as input, and produce as output a fixed-length message digest.

MD5 is the third message digest algorithm created by Rivest. All three (the others are MD2and MD4) have similar structures, but MD2 was optimized for 8-bit machines, in comparison with the two later formulas, which are optimized for 32-bit machines. The MD5 algorithm is an extension of MD4, which the critical review found to be fast, but possibly not absolutely secure. In comparison, MD5 is not quite as fast as the MD4 algorithm, but offered much more assurance of data security.

How MD5 works:

MD5 algorithm follows the following steps

1. Append Padding Bits: In the first step, we add padding bits in the original message in such a way that the total length of the message is 64 bits less than the exact multiple of 512.

Suppose we are given a message of 1000 bits. Now we have to add padding bits to the original message. Here we will add 472 padding bits to the original message. After adding the padding bits the size of the original message/output of the first step will be 1472 i.e. 64 bits less than an exact multiple of 512 (i.e. 512*3 = 1536).

Length(original message + padding bits) = 512 * i – 64 where i = 1,2,3 . . .

2. Append Length Bits: In this step, we add the length bit in the output of the first step in such a way that the total number of the bits is the perfect multiple of 512. Simply, here we add the 64-bit as a length bit in the output of the first step.
i.e. output of first step = 512 * n – 64
length bits = 64. After adding both we will get **512 * n** i.e. the exact multiple of 512.

3. Initialize MD buffer: Here, we use the 4 buffers i.e. J, K, L, and M. The size of each buffer is 32 bits.

- J = 0x67425301
- K = 0xEDFCBA45
- L = 0x98CBADFE
- M = 0x13DCE476

4. Process Each 512-bit Block: This is the most important step of the MD5 algorithm. Here, a total of 64 operations are performed in 4 rounds. In the 1st round, 16 operations will be performed, 2nd round 16 operations will be performed, 3rd round 16 operations will be performed, and in the 4th round, 16 operations will be performed. We apply a different function on each round i.e. for the 1st round we apply the F function, for the 2nd G function, 3rd for the H function, and 4th for the I function.
We perform OR, AND, XOR, and NOT (basically these are logic gates) for calculating functions. We use 3 buffers for each function i.e. K, L, M.

- F(K,L,M) = (K AND L) OR (NOT K AND M)
- G(K,L,M) = (K AND L) OR (L AND NOT M)
- H(K,L,M) = K XOR L XOR M
- I(K,L,M) = L XOR (K OR NOT M)

After applying the function now we perform an operation on each block. For performing operations we need

- add modulo 2^{32}

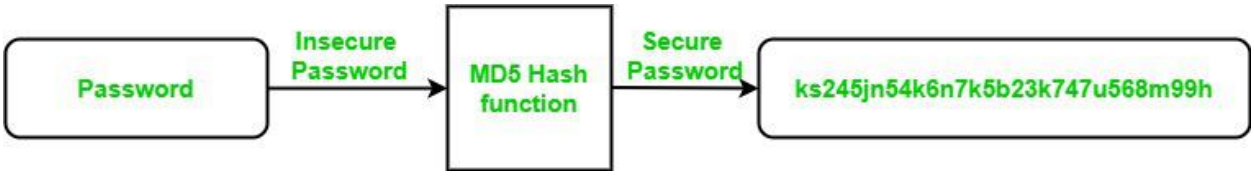
- M[i] – 32 bit message.
- K[i] – 32-bit constant.
- <<<n – Left shift by n bits.

Now take input as initialize MD buffer i.e. J, K, L, M. Output of K will be fed in L, L will be fed into M, and M will be fed into J. After doing this now we perform some operations to find the output for J.

- In the first step, Outputs of K, L, and M are taken and then the function F is applied to them. We will add modulo 2^{32} bits for the output of this with J.
- In the second step, we add the M[i] bit message with the output of the first step.
- Then add 32 bits constant i.e. K[i] to the output of the second step.
- At last, we do left shift operation by n (can be any value of n) and addition modulo by 2^{32}.

After all steps, the result of J will be fed into K. Now same steps will be used for all functions G, H, and I. After performing all 64 operations we will get our message digest.

Output:
After all, rounds have been performed, the buffer J, K, L, and M contains the MD5 output starting with the lower bit J and ending with Higher bits M.



MD5 security:

The goal of any message digest function is to produce digests that appear to be random. To be considered cryptographically secure, the hash function should meet two requirements: first, that it is impossible for an attacker to generate a message matching a specific hash value; and second, that it is impossible for an attacker to create two messages that produce the same hash value.

MD5 hashes are no longer considered cryptographically secure, and they should not be used for cryptographic authentication.

In 2011, the IETF published RFC 6151, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms," which cited a number of recent attacks against MD5 hashes, especially one that generated hash collisions in a minute or less on a standard notebook and another that could generate a collision in as little as 10 seconds on a 2.66 GHz Pentium 4 system. As a result, the IETF suggested that new protocol designs should not use MD5 at all, and that the recent research attacks against the algorithm "have provided sufficient reason to eliminate MD5 usage in applications where collision resistance is required such as digital signatures."

What is MD5 used for?

Although originally designed as a cryptographic message authentication code algorithm for use on the internet, MD5 hashing is no longer considered reliable for use as a cryptographic checksum because security experts have demonstrated techniques capable of easily producing MD5 collisions on commercial off-the-shelf computers. An encryption collision means two files have the same hash. Hash functions are used for message security, password security, computer forensics and cryptocurrency. Ronald Rivest, founder of RSA Data Security LLC and professor at Massachusetts Institute of Technology, designed MD5 in 1991 as an improvement to a prior message-digest algorithm, MD4. Describing it in Internet Engineering Task Force (IETF) Request for Comments (RFC) 1321, "The MD5 Message-Digest Algorithm," he wrote:

The algorithm takes as input a message of arbitrary length and produces as output a 128-bit 'fingerprint' or 'message digest' of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given prespecified target message digest. The MD5 algorithm is intended for digital signature applications, where a large file must be 'compressed' in a secure manner before being encrypted with a private (secret) key under a public-key cryptosystem such as RSA.

Is MD5 secure?

The goal of any message-digest function is to produce digests that appear to be random. MD5 hashes are no longer considered cryptographically secure methods and should not be used for cryptographic authentication, according to IETF. To be considered cryptographically secure, the hash function should meet two requirements:

1. It is impossible for an attacker to generate a message matching a specific hash value.
2. It is impossible for an attacker to create two messages that produce the same hash value.

Application Of MD5 Algorithm:

- We use message digest to verify the integrity of files/ authenticates files.
- MD5 was used for data security and encryption.
- It is used to Digest the message of any size and also used for Password verification.

- For Game Boards and Graphics.

Advantages of MD5 Algorithm:

- MD5 is faster and simple to understand.
- MD5 algorithm generates a strong password in 16 bytes format. All developers like web developers etc use the MD5 algorithm to secure the password of users.
- To integrate the MD5 algorithm, relatively low memory is necessary.
- It is very easy and faster to generate a digest message of the original message.

Disadvantages of MD5 Algorithm:

- MD5 generates the same hash function for different inputs.
- MD5 provides poor security over SHA1.
- MD5 has been considered an insecure algorithm. So now we are using SHA256 instead of MD5
- MD5 is neither a symmetric nor asymmetric algorithm.

Kerberos-

Kerberos is a protocol for authenticating service requests between trusted hosts across an un trusted network, such as the internet. Kerberos is built in to all major operating systems, including Microsoft Windows, Apple OS X, FreeBSD and Linux.

Since Windows 2000, Microsoft has incorporated the Kerberos protocol as the default authentication method in Windows, and it is an integral component of the Windows Active Directory service. Broadband service providers also use Kerberos to authenticate DOCSIS cable modems and set-top boxes accessing their networks.

Kerberos was originally developed for Project Athena at the Massachusetts Institute of Technology (MIT). The name Kerberos was taken from Greek mythology; Kerberos (Cerberus) was a three-headed dog who guarded the gates of Hades. The three heads of the Kerberos protocol represent a client, a server and a Key Distribution Center (KDC), which acts as Kerberos' trusted third-party authentication service.

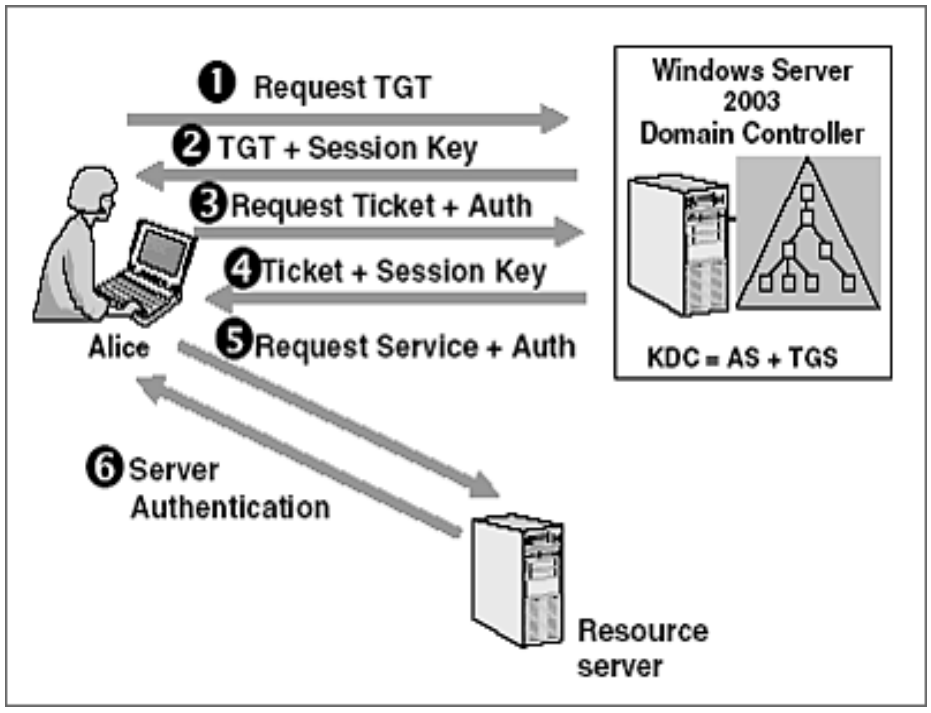
Users, machines and services using Kerberos need only trust the KDC, which runs as a single process and provides two services: an authentication service and a ticket granting service. KDC "tickets" provide mutual authentication, allowing nodes to prove their identity to one another in a secure manner. Kerberos authentication uses conventional shared secret cryptography to prevent packets traveling across the network from being read or changed and to protect messages from eavesdropping and replay attacks.

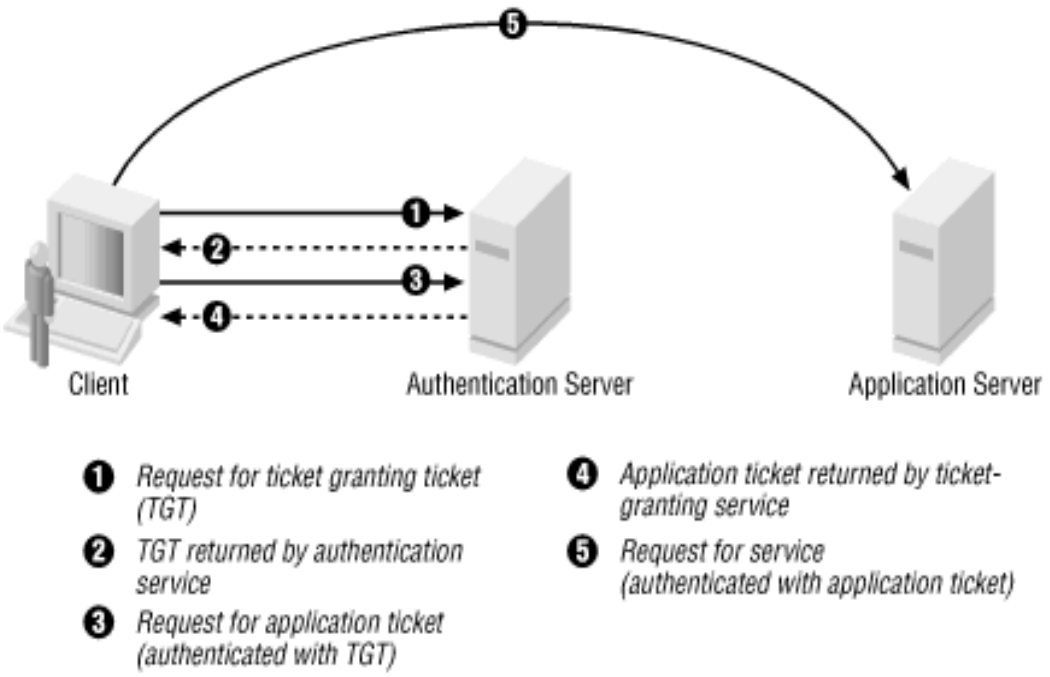
Kerberos4 is an authentication service developed as part of Project Athena at MIT. The problem that Kerberos addresses is this: Assume an open distributed environment in which users at workstations wish to access services on servers distributed throughout the network. We would like for servers to be able to restrict access to authorized users and to be able to authenticate requests for service. In this environment, a workstation cannot be trusted to identify its users correctly to network services.

In particular, the following three threats exist:

- A user may gain access to a particular workstation and pretend to be another user operating from that workstation.
- A user may alter the network address of a workstation so that the requests sent from the altered workstation appear to come from the impersonated workstation.
- A user may eavesdrop on exchanges and use a replay attack to gain entrance to a server or to disrupt operations. In any of these cases, an unauthorized user may be able to gain access to services and data that he or she is not authorized to access.

Rather than building in elaborate authentication protocols at each server, Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users. Kerberos relies exclusively on symmetric encryption, making no use of public-key encryption. Two versions of Kerberos are in common use. Version 4 [MILL88, STEI88] implementations still exist. Version 5 [KOHL94] corrects some of the security deficiencies of version 4 and has been issued as a proposed Internet Standard (RFC 4120).5 We begin this section with a brief discussion of the motivation for the Kerberos approach. Then, because of the complexity of Kerberos, it is best to start with a description of the authentication protocol used in version 4. This enables us to see the essence of the Kerberos strategy without considering some of the details required to handle subtle security threats. Finally, we examine version 5.





Kerberos Protocol Overview:

Kerberos is a key distribution and user authentication service developed at MIT. Kerberos is a computer network authentication protocol which works on the basis of 'tickets' to allow nodes communicating over a non-secure network and nodes prove their identity to one another in a secure manner. It is aimed primarily at a client–server model and it also provides mutual authentication. It gives protection against eavesdropping and replay attacks. The need for Kerberos protocol is, when using the services of an open distributed network, the service providing server must identify the authorized workstation otherwise there will be a possibility of three types of threats, they are given as,

- A user may gain access to a particular workstation and pretend to be another user operating from that workstation.
- A user may alter the network address of a workstation so that the requests sent from the altered workstation appear to come from the impersonated workstation.
- A user may eavesdrop on exchanges and use a replay attack to gain entrance to a server or to disrupt operations. In any of these cases, an unauthorized user may be able to gain access to services and data that he or she is not authorized to access. Rather than building elaborate authentication protocols at each server, Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users.

Kerberos relies exclusively on symmetric encryption. There are five versions of Kerberos present, the first three are internal to MIT, the version 4 and versions 5 are available commercially. A Kerberos server is known as a Key Distribution Centre (KDC). Each KDC has an authentication service (AS) and a Ticket Granting service (TGS) Figure gives the overview of the Kerberos protocol. Kerberos deals with three kinds of security object:

Ticket: a token issued to a client by the Kerberos ticket-granting service (TGS) for presentation to a particular server, verifying that the sender has recently been authenticated by Kerberos. Tickets include an expiry time and a newly generated session key for the use by the client and the server. Authenticator: a token constructed by a client and sent to a server to prove the identity of the user and the currency of any communication with a server. It contains client's name and a timestamp and is encrypted in the appropriate session key.

Session key: a secret key generated by Kerberos and issued to a client for use when communicating with a particular server Working module of Kerberos version 5: The client authenticates to AS using a secret key (User login password) and receives a ticket from the AS. Later the client can use this ticket to get additional tickets from TGS for server. A Kerberos ticket has a fixed period of validity starting at time t1 and ending at time t2. A ticket for a client C to access a server S takes the form:

{C, S, t1, t2, Kcs} Ks, which we denote as {ticket(C,S)} Ks. To obtain a ticket for any server S, C constructs an authenticator encrypted in KcT of the form:

{C, t} KcT, which we denote as {auth(C)} KcT

In a first step, client obtain a session ticket and TGT ticket by passing his secret key to AS and TGS and these tickets are once per login session.

C->A: C,T, n.

Client C requests the Kerberos authentication server AS to supply a ticket for communication with the TGS T

A->C: {KcT, n, {ticket(C, T)} KT} Kc.

A->C: {KcT, n, {ticket(C, T)} KT} Kc.

Returns a message containing a ticket encrypted in its secret key and a session key for C to use with T.

In a second step, Client obtain ticket for a server S, once per client-server session

C->T: {auth(C)}KcT, {ticket(C, T)} KT, S, n

C requests the ticket-granting server T to supply a ticket for communication with another server S

T->C: {Kcs, n, {ticket(C,S)} Ks} KcT

T checks the ticket. If it is valid T generates a new session key Kcs and returns it with a ticket for encrypted in the server's secret key Ks).

In a third step, client allowed to access a service on server requested with a ticket

C->S: {auth(C)} Kcs, {ticket(C, S)} Ks, request, n

C sends the ticket to S with a generated authenticator for C and a request.

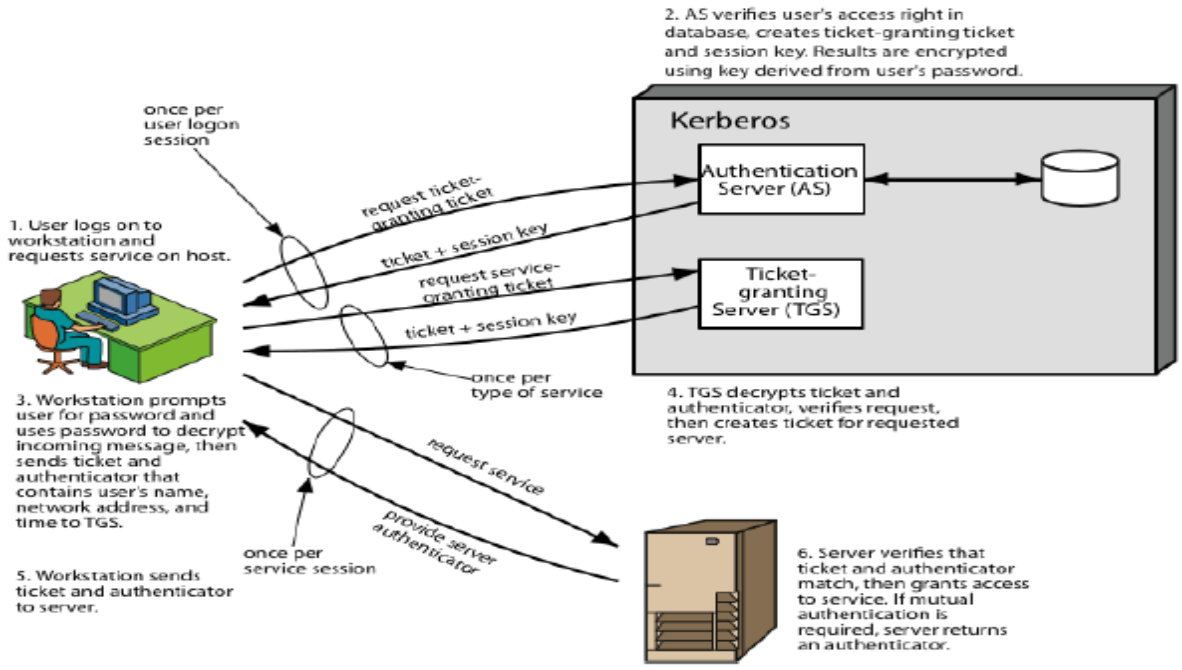
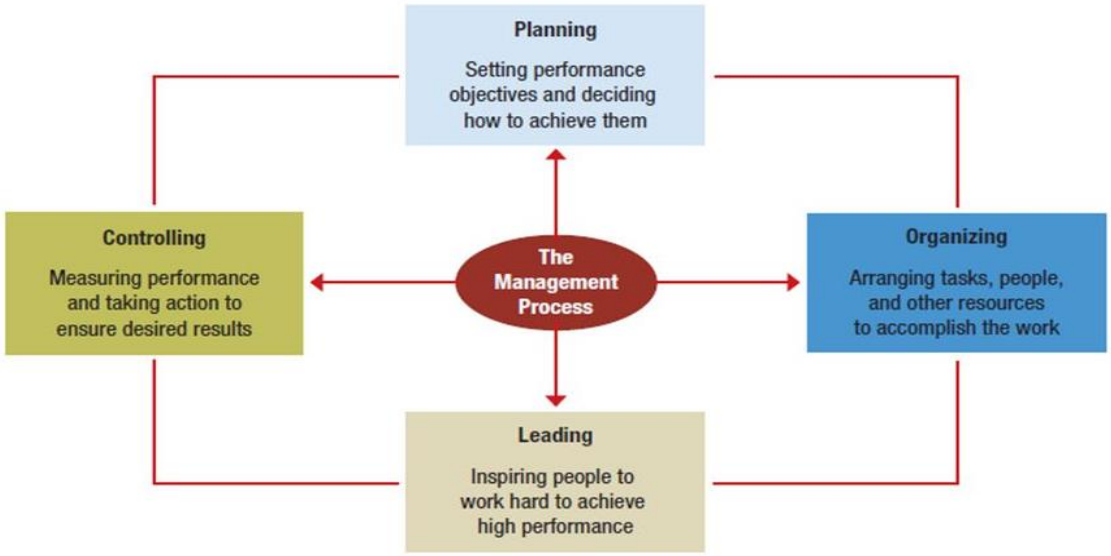


Figure 2.2: Overview of Kerberos.

Key Management-

Key distribution is the function that delivers a key to two parties who wish to exchange secure encrypted data. Some sort of mechanism or protocol is needed to provide for the secure distribution of keys.

- Key distribution often involves the use of master keys, which are infrequently used and are long lasting, and session keys, which are generated and distributed for temporary use between two parties.
- Public-key encryption schemes are secure only if the authenticity of the public key is assured. A public-key certificate scheme provides the necessary security.
- X.509 defines the format for public-key certificates. This format is widely used in a variety of applications.
- A public-key infrastructure (PKI) is defined as the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography.
- Typically, PKI implementations make use of X.509 certificates.



Key Management:

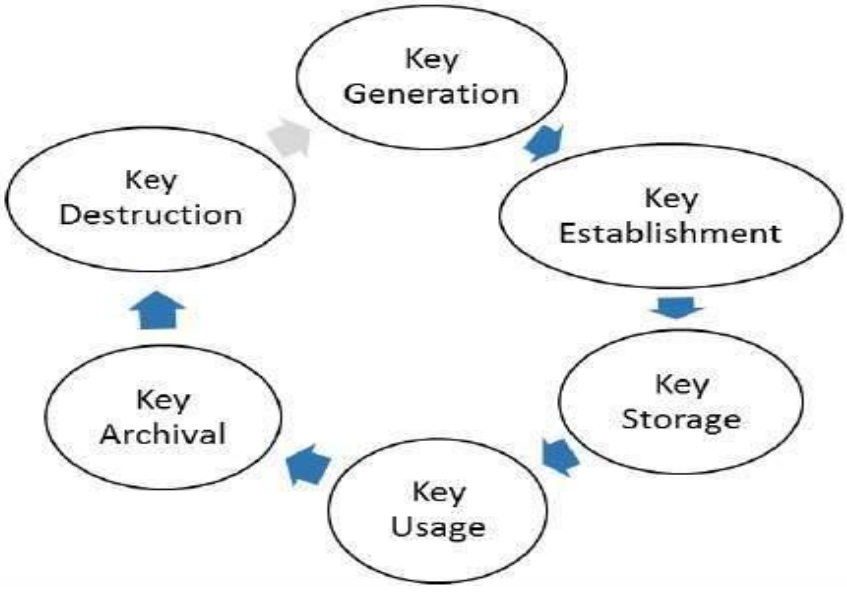
It goes without saying that the security of any cryptosystem depends upon how securely its keys are managed. Without secure procedures for the handling of cryptographic keys, the benefits of the use of strong cryptographic schemes are potentially lost.

It is observed that cryptographic schemes are rarely compromised through weaknesses in their design. However, they are often compromised through poor key management.

There are some important aspects of key management which are as follows −

- Cryptographic keys are nothing but special pieces of data. Key management refers to the secure administration of cryptographic keys.

Key management deals with entire key lifecycle as depicted in the following illustration-



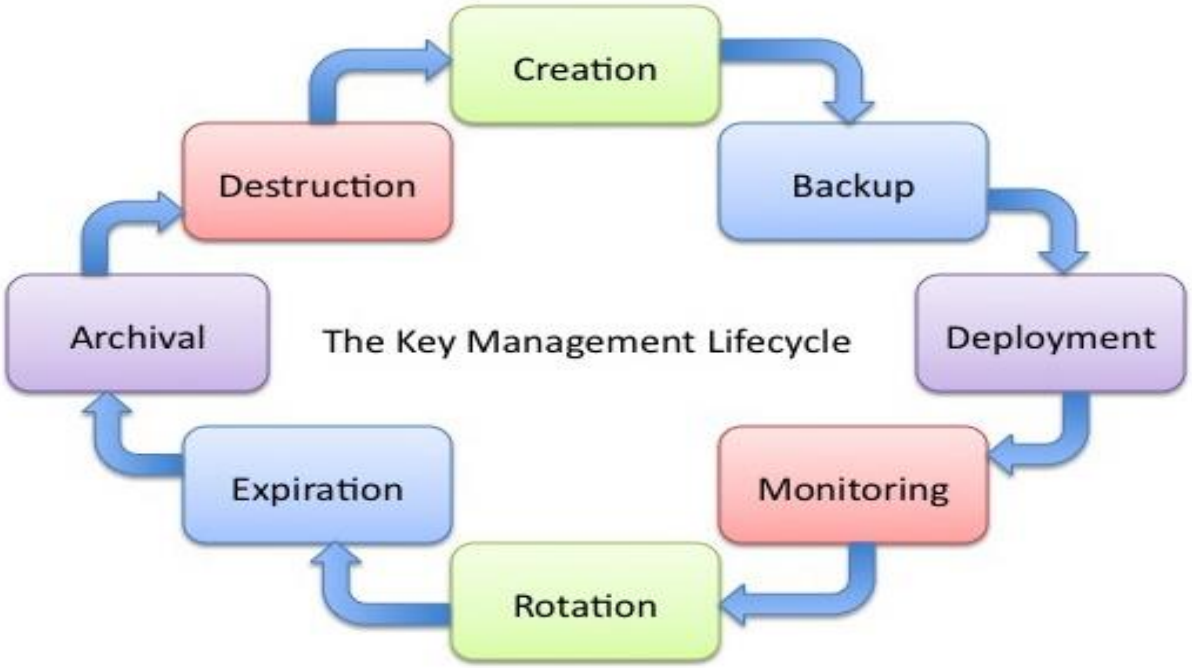
There are two specific requirements of key management for public key cryptography.

- **Secrecy of private keys.** Throughout the key lifecycle, secret keys must remain secret from all parties except those who are owner and are authorized to use them.
- **Assurance of public keys.** In public key cryptography, the public keys are in open domain and seen as public pieces of data. By default there are no assurances of whether a public key is correct, with whom it can be associated, or what it can be used for. Thus key management of public keys needs to focus much more explicitly on assurance of purpose of public keys.

The most crucial requirement of 'assurance of public key' can be achieved through the public-key infrastructure (PKI), a key management systems for supporting public-key cryptography.

Key management refers to management of cryptographic keys in a cryptosystem. This includes dealing with the generation, exchange, storage, use, crypto-shredding (destruction) and replacement of keys. It includes cryptographic protocol design, key servers, user procedures, and other relevant protocols.

Key management concerns keys at the user level, either between users or systems. This is in contrast to key scheduling, which typically refers to the internal handling of keys within the operation of a cipher. Successful key management is critical to the security of a cryptosystem. It is the more challenging side of cryptography in a sense that it involves aspects of social engineering such as system policy, user training, organizational and departmental interactions, and coordination between all of these elements, in contrast to pure mathematical practices that can be automated.



Key Exchange:

Prior to any secured communication, users must set up the details of the cryptography. In some instances this may require exchanging identical keys (in the case of a symmetric key system). In others it may require possessing the other party's public key. While public keys can be openly exchanged (their corresponding private key is kept secret), symmetric keys must be exchanged over a secure communication channel. Formerly, exchange of such a key was extremely troublesome, and was greatly eased by access to secure channels such as a diplomatic bag. Clear text exchange of symmetric keys would enable any interceptor to immediately learn the key, and any encrypted data.

The advance of public key cryptography in the 1970s has made the exchange of keys less troublesome. Since the Diffie-Hellman key exchange protocol was published in 1975, it has become possible to exchange a key over an insecure communications channel, which has substantially reduced the risk of key disclosure during distribution. It is possible, using something akin to a book code, to include key indicators as clear text attached to an encrypted message. The encryption technique used by Richard Sorge's code clerk was of this type, referring to a page in a statistical manual, though it was in fact a code. The German Army Enigma symmetric encryption key was a mixed type early in its use; the key was a combination of secretly distributed key schedules and a user chosen session key component for each message.

In more modern systems, such as Open PGP compatible systems, a session key for a symmetric key algorithm is distributed encrypted by an asymmetric key algorithm. This approach avoids even the necessity for using a key exchange protocol like Diffie-Hellman key exchange.

Another method of key exchange involves encapsulating one key within another. Typically a master key is generated and exchanged using some secure method. This method is usually cumbersome or expensive (breaking a master key into multiple parts and sending each with a trusted courier for example) and not suitable for use on a larger scale. Once the master key has been securely exchanged, it can then be used to securely exchange subsequent keys with ease. This technique is usually termed key wrap. A common technique uses block ciphers and cryptographic hash functions.

A related method is to exchange a master key (sometimes termed a root key) and derive subsidiary keys as needed from that key and some other data (often referred to as diversification data). The most common use for this method is probably in smartcard-based cryptosystems, such as those found in banking cards. The bank or credit network embeds their secret key into the card's secure key storage during card production at a secured production facility. Then at the point of sale the card and card reader are both able to derive a common set of session keys based on the shared secret key and card-specific data (such as the card serial number). This method can also be used when keys must be related to each other (i.e., departmental keys are tied to divisional keys, and individual keys tied to departmental keys). However, tying keys to each other in this way increases the damage which may result from a security breach as attackers will learn something about more than one key. This reduces entropy, with regard to an attacker, for each key involved.

Key storage:

However distributed, keys must be stored securely to maintain communications security. Security is a big concern and hence there are various techniques in use to do so. Likely the most common is that an encryption application manages keys for the user and depends on an access password to control use of the key. Likewise, in the case of smart phone keyless access platforms, they keep all identifying door information off mobile phones and servers and encrypt all data, where just like low-tech keys, users give codes only to those they trust.

Key use:

The major issue is length of time a key is to be used, and therefore frequency of replacement. Because it increases any attacker's required effort, keys should be frequently changed. This also limits loss of information, as the number of stored encrypted messages which will become readable when a key is found will decrease as the frequency of key change increases. Historically, symmetric keys have been used for long periods in situations in which key exchange was very difficult or only possible intermittently. Ideally, the symmetric key should change with each message or interaction, so that only that message will become readable if the key is learned (*e.g.*, stolen, cryptanalyzed, or social engineered).

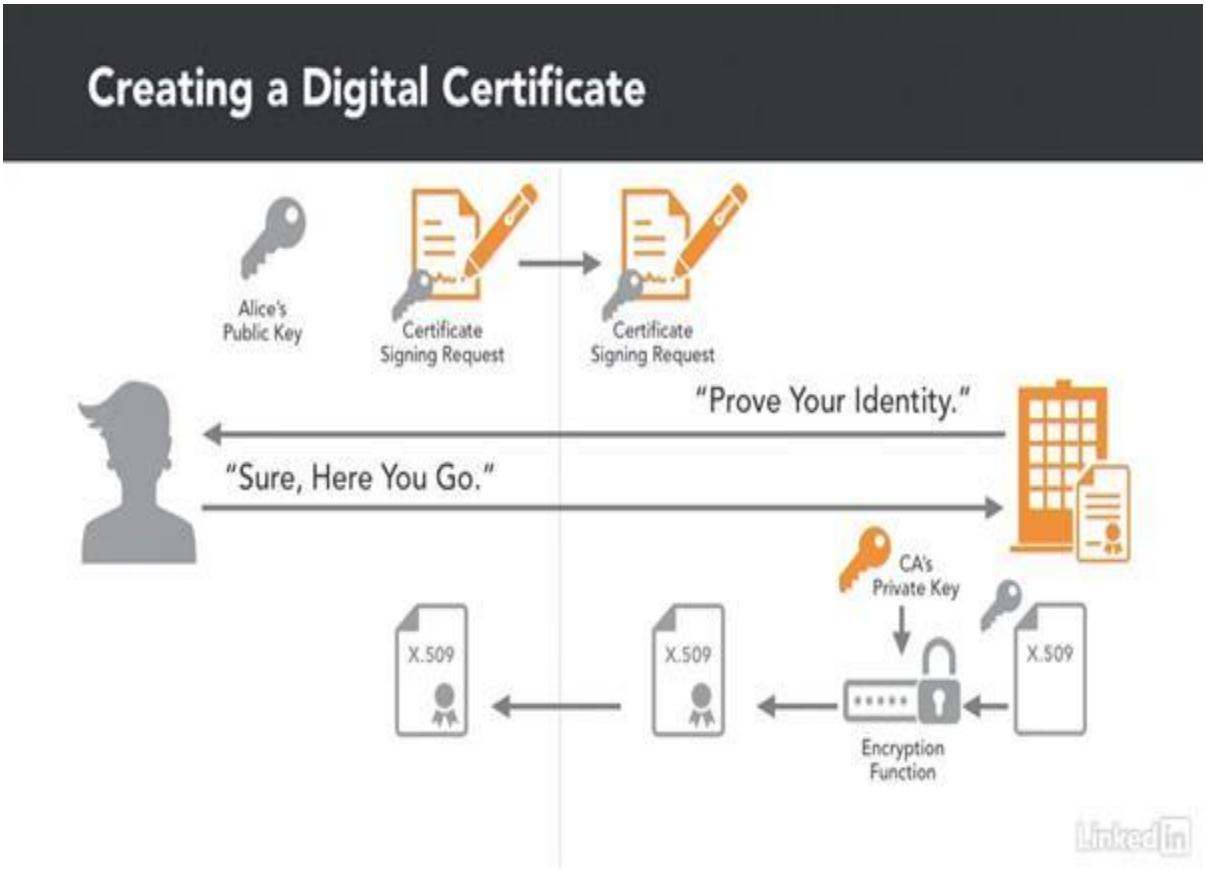
Key Management Generations:

Cryptographic keys shall be generated within cryptographic module with at least a FIPS 140-2 compliance. For explanatory purposes, consider the cryptographic module in which a key is generated to be the key-generating module. Any random value required by the key-generating module shall be generated within that module; that is, the Random Bit Generator that generates the random value shall be implemented within cryptographic module with at least a FIPS 140-2 compliance that generates the key. Hardware cryptographic modules are preferred over software cryptographic modules for protection.

X.509 Digital Certificate-

A digital certificate is an electronic "passport" that allows a person, computer or organization to exchange information securely over the Internet using the public key infrastructure (PKI). A digital certificate may also be referred to as a public key certificate. Just like a passport, a digital certificate provides identifying information, is forgery resistant and can be verified because it was issued by an official, trusted agency. The certificate contains the name of the certificate holder, a serial number, expiration dates, a copy of the certificate holder's public key (used for encrypting messages and digital signatures) and the digital signature of the certificate-issuing authority (CA) so that a recipient can verify that the certificate is real.

To provide evidence that a certificate is genuine and valid, it is digitally signed by a root certificate belonging to a trusted certificate authority. Operating systems and browsers maintain lists of trusted CA root certificates so they can easily verify certificates that the CAs have issued and signed. When PKI is deployed internally, digital certificates can be self-signed. Many digital certificates conform to the X.509 standard.



X.509 Certificates:

ITU-T recommendation X.509 is part of the X.500 series of recommendations that define a directory service. The directory is, in effect, a server or distributed set of servers that maintains a database of information about users. The information includes a mapping from user name to network address, as well as other attributes and information about the users. X.509 defines a framework for the provision of authentication services by the X.500 directory to its users. The directory may serve as a repository of public-key certificates. Each certificate contains the public key of a user and is signed with the private key of a trusted certification authority. In addition, X.509 defines alternative authentication protocols based on the use of public-key certificates. X.509 is an important standard because the certificate structure and authentication protocols defined in X.509 are used in a variety of contexts.

X.509 certificate format:

X.509 certificate format is used in S/MIME, IP Security (Chapter 19), and SSL/TLS (Chapter 16). X.509 was initially issued in 1988. The standard was subsequently revised to address some of the security concerns documented in [IANS90] and [MITC90]; a revised recommendation was issued in 1993. A third version was issued in 1995 and revised in 2000. X.509 is based on the use of public-key cryptography and digital signatures. The standard does not dictate the use of a specific algorithm but recommends RSA. The digital signature scheme is assumed to require the use of a hash function. Again, the standard does not dictate a specific hash algorithm. The 1988 recommendation included the description of a recommended hash algorithm; this algorithm has since been shown to be insecure and was dropped from the 1993 recommendation. Figure 14.13 illustrates the generation of a public-key certificate. Certificates The heart of the X.509 scheme is the public-key certificate associated with each user. These user certificates are assumed to be created by some trusted certification authority (CA) and placed in the directory by the CA or by the user. The directory server itself is not responsible for the creation of public keys or for the certification function; it merely provides an easily accessible location for users to obtain certificates. Figure shows the general format of a certificate, which includes the following elements.

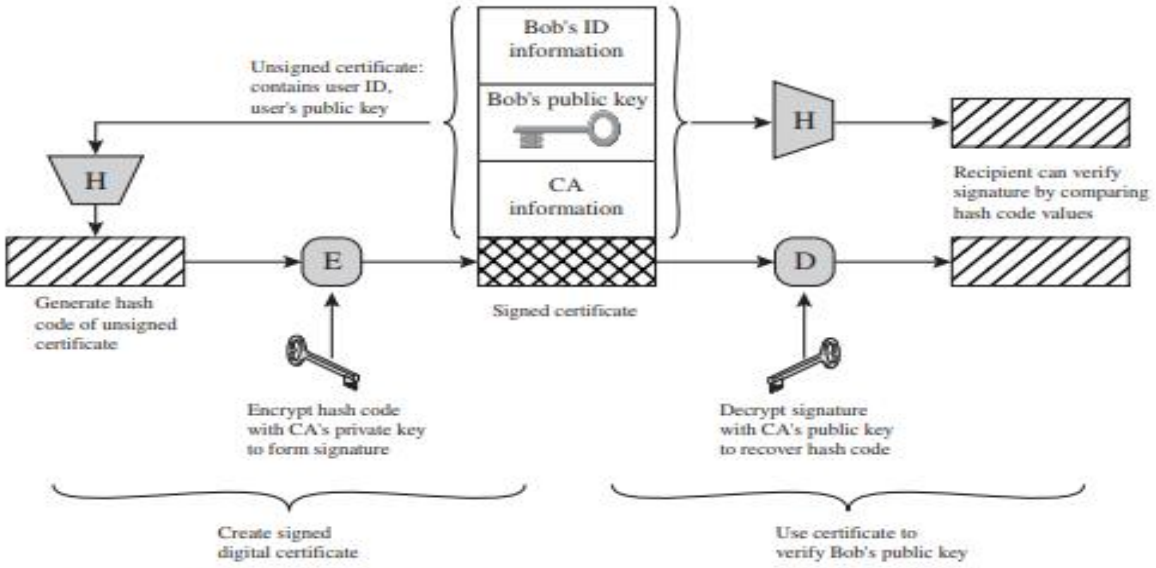


Figure 14.13 Public-Key Certificate Use

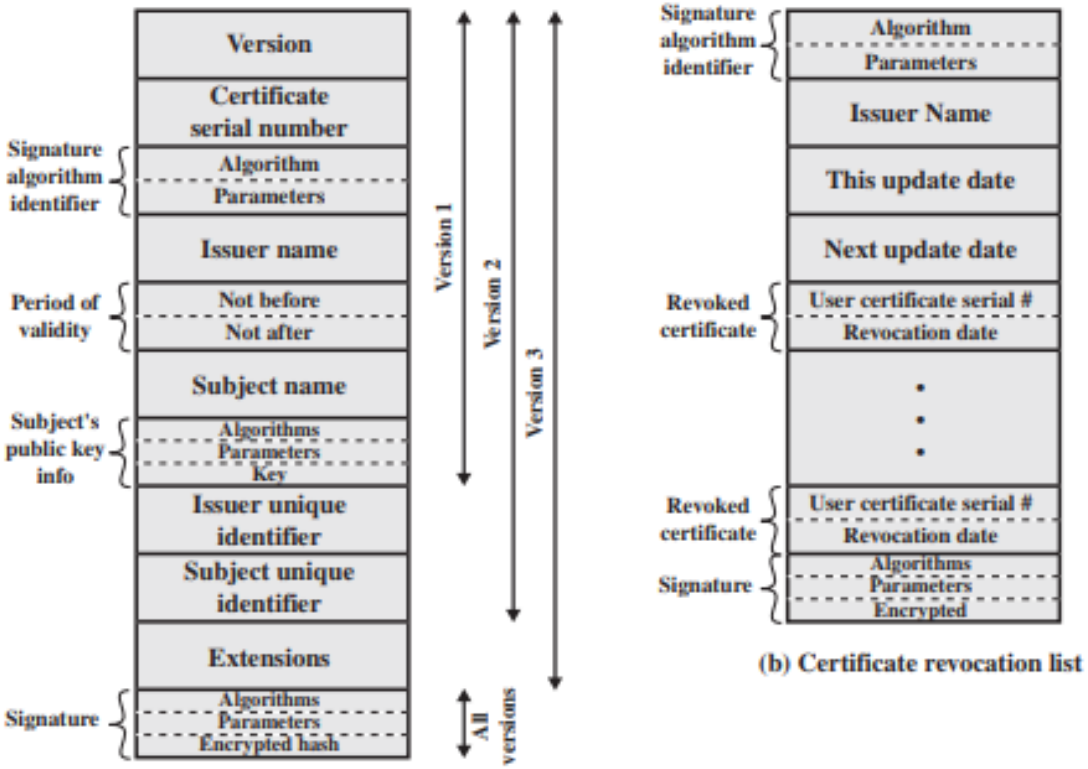


Figure 14.14 X.509 Formats

Version:

Differentiates among successive versions of the certificate format; the default is version

- If the issuer unique identifier or subject unique identifier are present, the value must be version
- If one or more extensions are present, the version must be version
- Serial number: An integer value unique within the issuing CA that is unambiguously associated with this certificate.

Signature algorithm identifier:

- The algorithm used to sign the certificate together with any associated parameters. Because this information is repeated in the signature field at the end of the certificate, this field has little, if any, utility.
- Issuer name: X.500 is the name of the CA that created and signed this certificate.
- Period of validity: Consists of two dates: the first and last on which the certificate is valid.
- Subject name: The name of the user to whom this certificate refers. That is, this certificate certifies the public key of the subject who holds the corresponding private key.

- Subject's public-key information: The public key of the subject, plus an identifier of the algorithm for which this key is to be used, together with any associated parameters.
- Issuer unique identifier: An optional-bit string field used to identify uniquely the issuing CA in the event the X.500 name has been reused for different entities. Certificate serial number Version Issuer name Signature algorithm identifier Subject name Extensions Issuer unique identifier Subject unique identifier Algorithm Parameters Not before Algorithms Parameters Key Algorithms Parameters Encrypted hash
  (a) X.509 certificate Not after Subject's public key info Signature Period of validity Version 1 Version 2 Version 3 All versions Issuer Name This update date Next update date Signature algorithm identifier Algorithm Parameters User certificate serial #
  (b) Certificate revocation list Revocation date Algorithms Parameters Encrypted Signature Revoked certificate User certificate serial # Revocation date Revoked certificate Figure X.509 Formats

An X.509 certificate is a digital certificate that uses the widely accepted international X.509 public key infrastructure (PKI) standard to verify that a public key belongs to the user, computer or service identity contained within the certificate.

An X.509 certificate contains information about the identity to which a certificate is issued and the identity that issued it. Standard information in an X.509 certificate includes:

- Version – which X.509 version applies to the certificate (which indicates what data the certificate must include)
- Serial number – the identity creating the certificate must assign it a serial number that distinguishes it from other certificates
- Algorithm information – the algorithm used by the issuer to sign the certificate
- Issuer distinguished name – the name of the entity issuing the certificate (usually a certificate authority)
- Validity period of the certificate – start/end date and time
- Subject distinguished name – the name of the identity the certificate is issued to
- Subject public key information – the public key associated with the identity
- Extensions (optional)

Many of the certificates that people refer to as Secure Sockets Layer (SSL) certificates are in fact X.509 certificates.

The first X.509 certificates were issued in 1988 as part of the International Telecommunications Union's Telecommunication Standardization Sector (ITU-T) and the X.500 Directory Services Standard. In 1993, version 2 added two fields to support directory access control. Version 3 was released in 1996 and defines the formatting used for certificate extensions.

X.509 Certificate · issued by a Certification Authority (CA) each certificate contains:

- version
- serial number (unique within CA)
- algorithm identifier (used to sign certificate)
- issuer (CA)
- period of validity (from - to dates)
- subject (name of owner)
- public-key (algorithm, parameters, key)
- signature (of hash of all fields in certificate) ·

any user with access to CA can get any certificate from it · only the CA can modify a certificate CA Hierarchy · CA form a hierarchy · each CA has certificates for clients and parent · each client trusts parents certificates · enable verification of any certificate from one CA by users of all other CAs in hierarchy · X<> means certificate for A signed by authority X.