University of Technology
الجامعة التكنولوجية

Computer Sciences Department /
Network Management Branch

قسم علوم الحاسوب/ فرع اداره الشبكات

Network Security I

امنيه الشبكات I

Prof. Dr. Soukaena hassan hashem
ا.د. سكينه حسن هاشم

Network Security I

➢ Introduction to Network Security

- Definition of security
- Introductions to network
- Security Attacks.

➢Network Security Factors

- Message Confidentiality
- Message Integrity
- Message Authentication

➢ Message Confidentiality

- Confidentiality with Symmetric-Key Cryptography
- Confidentiality with Asymmetric-Key Cryptography

➢ Message Integrity

- Message and Message Digest
- Hash Function Criteria
- Hash Algorithms: SHA-1
- Biometric parameters

➢ Message Authentication

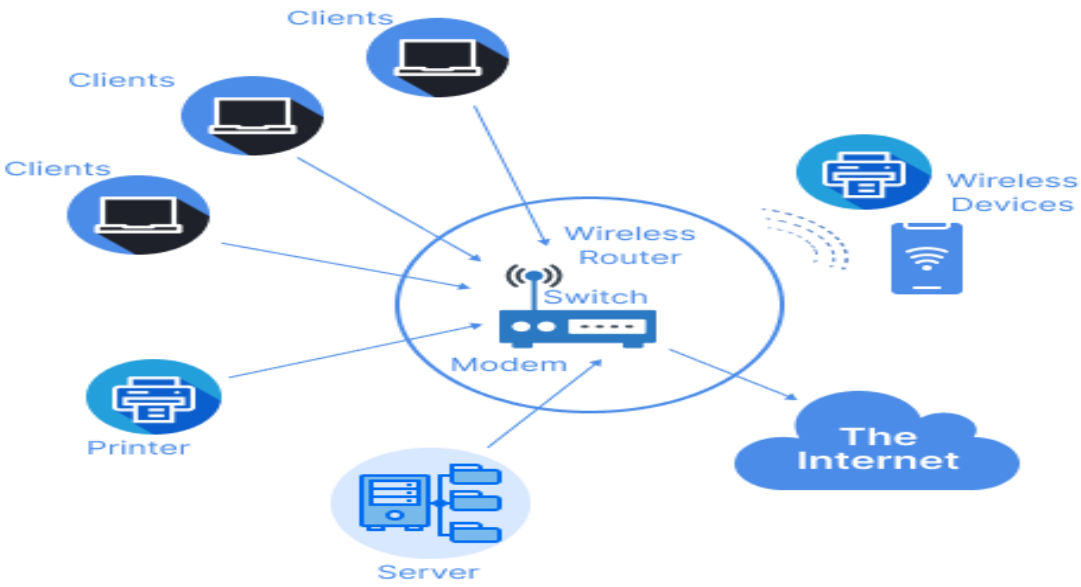
- Message Authentication Code (MAC)
- Digital Signature
- End-Point Authentication
- Passwords &Challenge-Response

References:

1. Computer Networking-A Top Down Approach-Kurose Ross – eighth Edition-2016

Introduction to Network Security

A **computer network** is a set of computers sharing resources located on or provided by network nodes. Computers use common communication protocols over digital interconnections to communicate with each other. These interconnections are made up of telecommunication network technologies based on physically wired, optical, and wireless radio-frequency methods that may be arranged in a variety of network topologies.



The nodes of a computer network can include personal computers, servers, networking hardware, or other specialized or general-purpose hosts. They are identified by network addresses and may have hostnames. Hostnames serve as memorable labels for the nodes and are rarely changed after initial assignment. Network addresses serve for locating and identifying the nodes by communication protocols such as the Internet Protocol. **Computer networks may be** classified by many criteria, including the transmission medium used to carry signals, bandwidth, communications protocols to organize network traffic, the network size, the topology, traffic control mechanisms, and organizational intent. **Computer networks support**

many applications and services, such as access to the World Wide Web, digital video and audio, shared use of application and storage servers, printers and fax machines, and use of email and instant messaging applications.

The enterprise network is large and complex, and probably relies on numerous connected endpoints. While this is good for your business operations, and makes your workflow easier to maintain, it also presents a challenge for security. The trouble is that the flexibility of movement within your network means that if a malicious actor gains access to your network, they are free to move around and cause damage, often without your knowledge. These network security threats leave your organization highly exposed to a data breach.

Data breaches are confirmed incidents that may lead to unauthorized access or disclosure of sensitive, confidential or other protected data.

Data breaches typically affect personally identifiable health information (PHI), personally identifiable information (PII), intellectual property, financial data like credit card or bank account numbers, personal data like social security numbers or user credentials, or commercially sensitive data like customer lists or manufacturing processes.

If any of these types of data, or similarly sensitive data, is exposed to unauthorized parties, this represents a data breach. Data breaches can damage an organization's reputation, may result in non-compliance with regulations or industry standards, and the organization can face fines or lawsuits in connection with the data it lost.

In 2018, Marriott International announced that hackers stole the data of approximately 500 million customers of its Starwood hotel brand. The breach happened in 2014, when Marriott acquired Starwood, but was only discovered

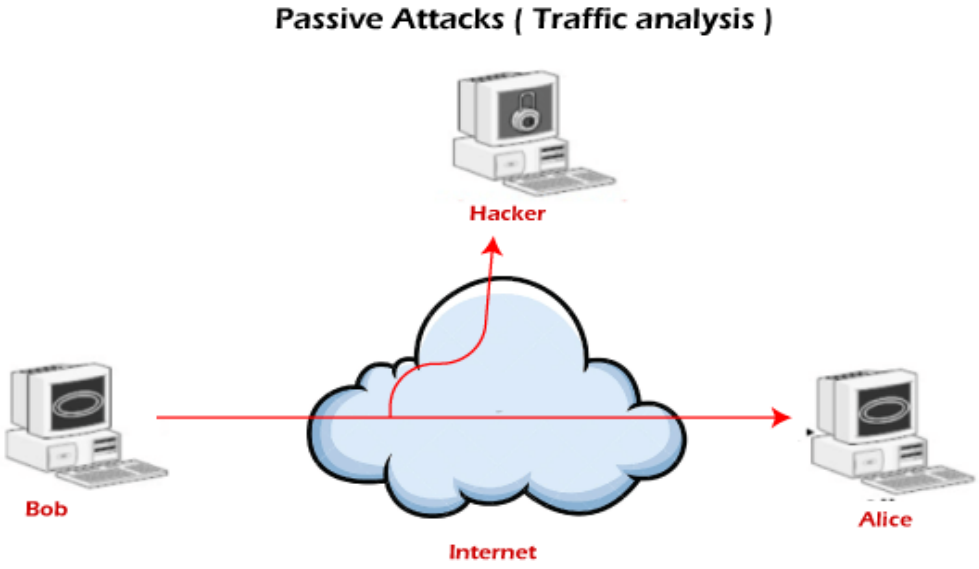
in 2018. Attackers stole data including names, contact details, travel information and passport numbers. The company said that credit card numbers or other financial information was exposed for 100 million customers, but that attackers may have been unable to make use of the data, which was strongly encrypted.

A data leak occurs when sensitive information is inadvertently exposed to unauthorized entities. Unlike data breaches, data leaks often result from human error or poor security practices rather than deliberate attacks. Examples include misconfigured databases, accidental emails sent to the wrong recipients, or unsecured cloud storage. While data leaks can be just as damaging as data breaches, they are generally unintentional and can be mitigated with better data handling

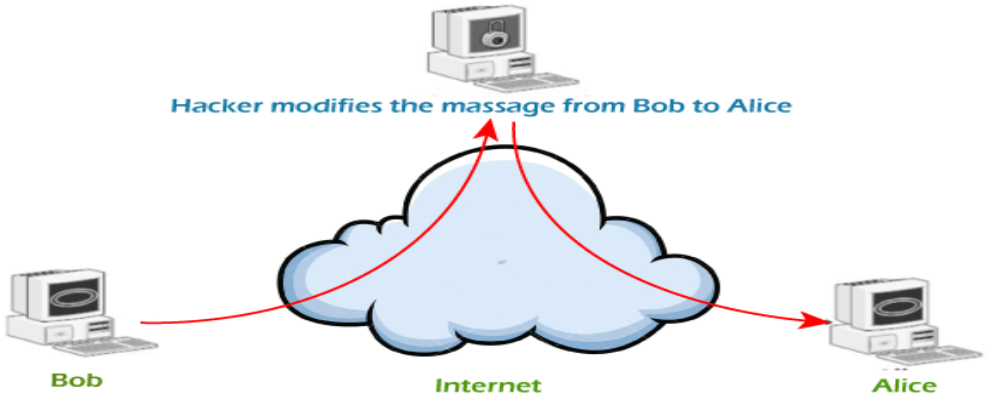
What Is a Network Attack?

A network attack is an attempt to gain unauthorized access to an organization's network, with the objective of stealing data or perform other malicious activity. There are two main types of network attacks:

Passive: Attackers gain access to a network and can monitor or steal sensitive information, but without making any change to the data, leaving it intact.



Active: Attackers not only gain unauthorized access but also modify data, deleting, encrypting or otherwise harming it.



Active Attacks (Modifications of messages)

We distinguish network attacks from several other types of attacks:

Endpoint attacks gaining unauthorized access to user devices, servers or other endpoints, typically compromising them by infecting them with malware.

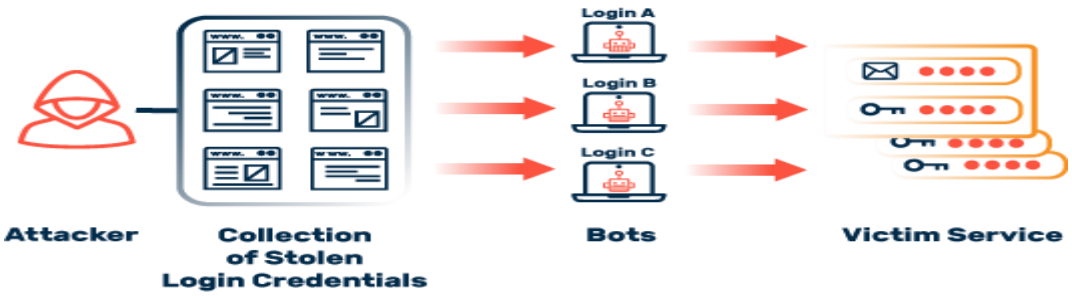
Malware attacks infecting IT resources with malware, allowing attackers to compromise systems, steal data and do damage. These also include ransomware attacks.

Vulnerabilities, exploits and attacks are exploiting vulnerabilities in software used in the organization to gain unauthorized access, compromise or sabotage systems.

Advanced persistent threats these are complex multilayered threats, which include network attacks but also other attack types.

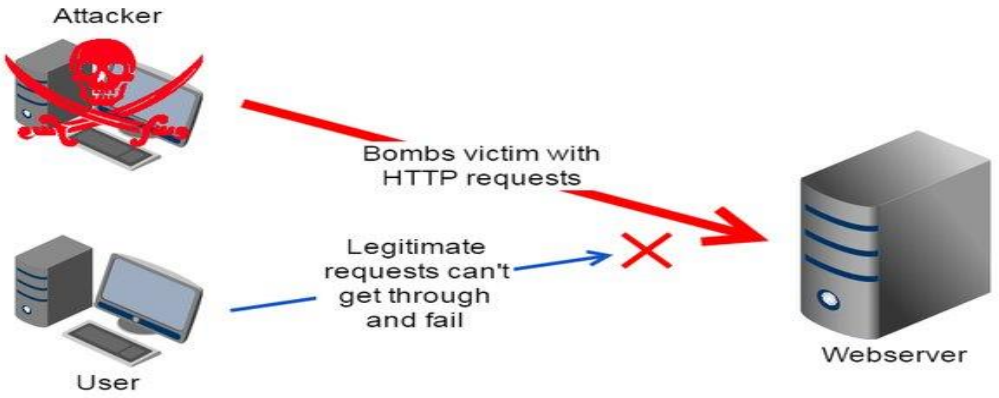
In a network attack, attackers are focused on penetrating the corporate network perimeter and gaining access to internal systems. Very often, once inside attackers will combine other types of attacks, for example compromising an endpoint, spreading malware or exploiting vulnerability in a system within the network. Following are common threat vectors attackers can use to penetrate your network.

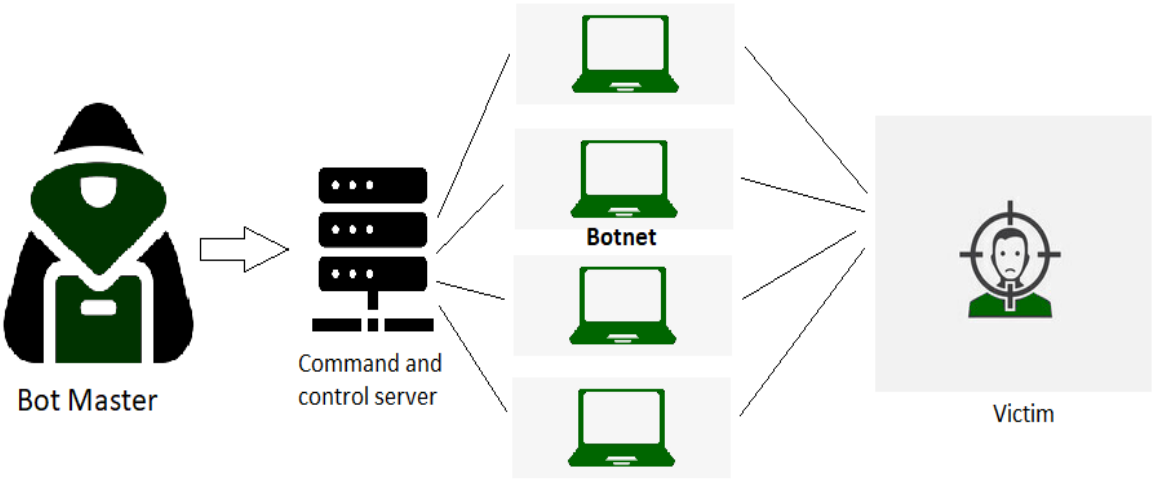
1. Unauthorized access Unauthorized access refers to attackers accessing a network without receiving permission. Among the causes of unauthorized access attacks are weak passwords, lacking protection against social engineering, previously compromised accounts, and insider threats.



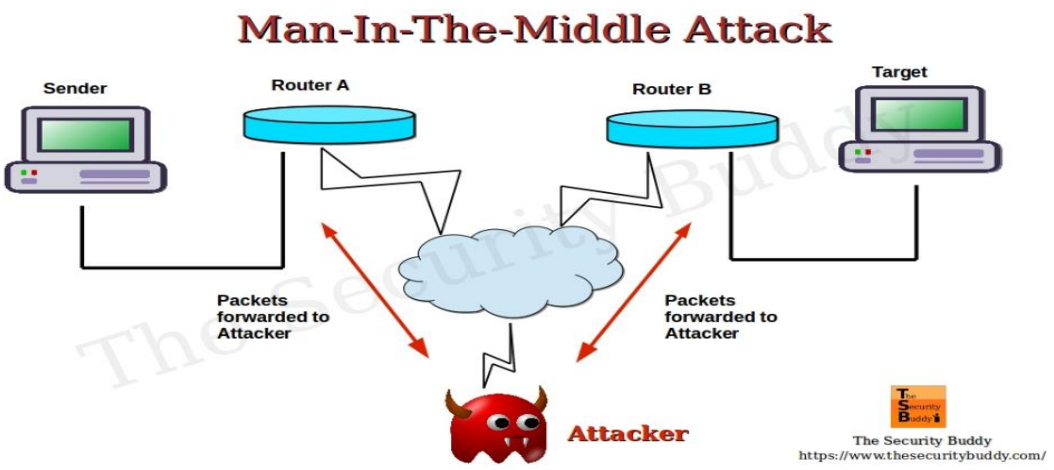
2. Distributed Denial of Service (DDoS) attacks Attackers build botnets, large fleets of compromised devices, and use them to direct false traffic at

your network or servers. DDoS can occur at the network level, for example by sending huge volumes of SYN/ACC packets which can overwhelm a server, or at the application level, for example by performing complex SQL queries that bring a database to its knees.

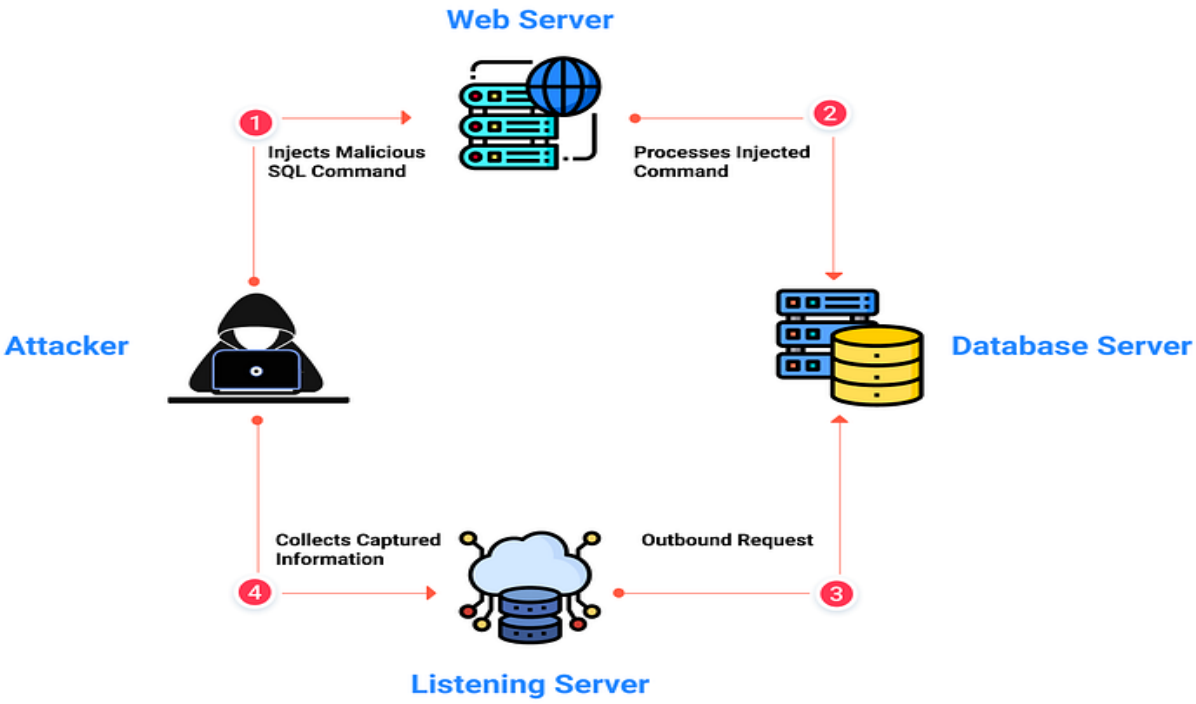




3. Man in the middle attacks A man in the middle attack involves attackers intercepting traffic, either between your network and external sites or within your network. If communication protocols are not secured or attackers find a way to circumvent that security, they can steal data that is being transmitted, obtain user credentials and hijack their sessions.

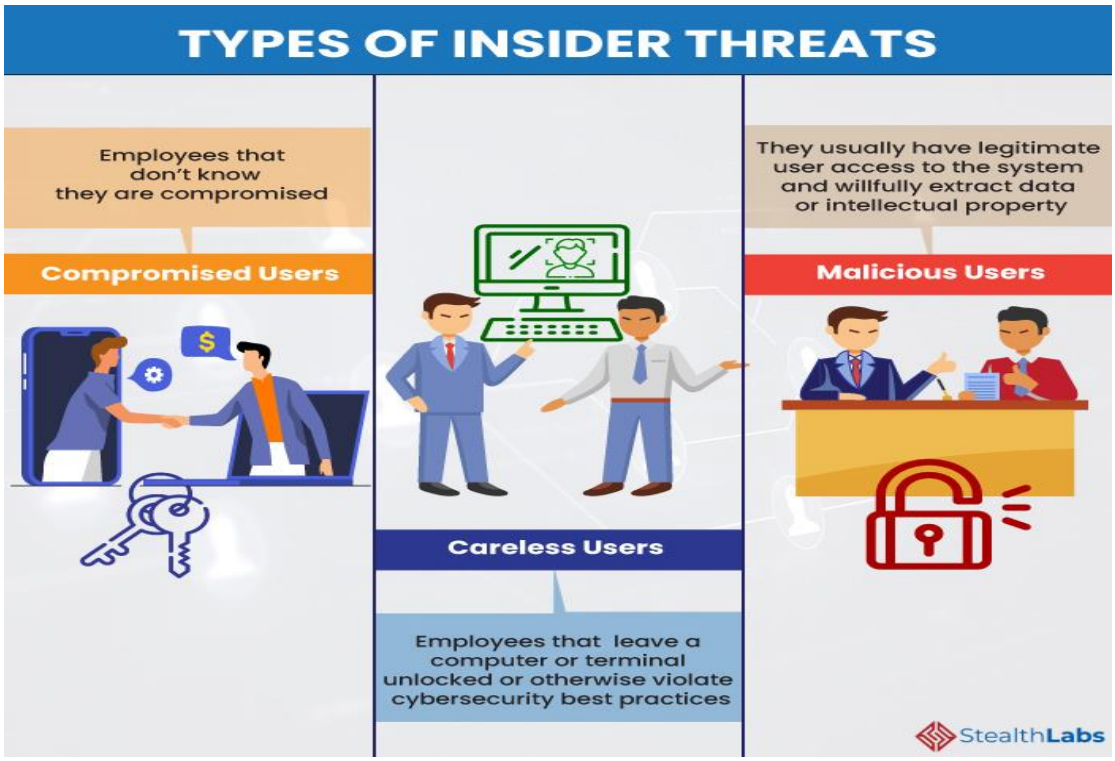


4. Code and SQL injection attacks Many websites accept user inputs and fail to validate and sanitize those inputs. Attackers can then fill out a form or make an API call, passing malicious code instead of the expected data values. The code is executed on the server and allows attackers to compromise it.



5. Privilege escalation Once attackers penetrate your network, they can use privilege escalation to expand their reach. Horizontal privilege escalation involves attackers gaining access to additional, adjacent systems, and vertical escalation means attackers gain a higher level of privileges for the same systems.

6. Insider threats A network is especially vulnerable to malicious insiders, who already have privileged access to organizational systems. Insider threats can be difficult to detect and protect against, because insiders do not need to penetrate the network in order to do harm. New technologies like User and Even Behavioral Analytics (UEBA) can help identify suspicious or anomalous behavior by internal users, which can help identify insider attacks.



Network Protection Best Practices

Segregate Your Network A basic part of avoiding network security threats is dividing a network into zones based on security requirements. This can be done using subnets within the same network, or by creating Virtual Local Area Networks (VLANs), each of which behaves like a complete separate network. Segmentation limits the potential impact of an attack to one zone, and requires attackers to take special measures to penetrate and gain access to other network zones.

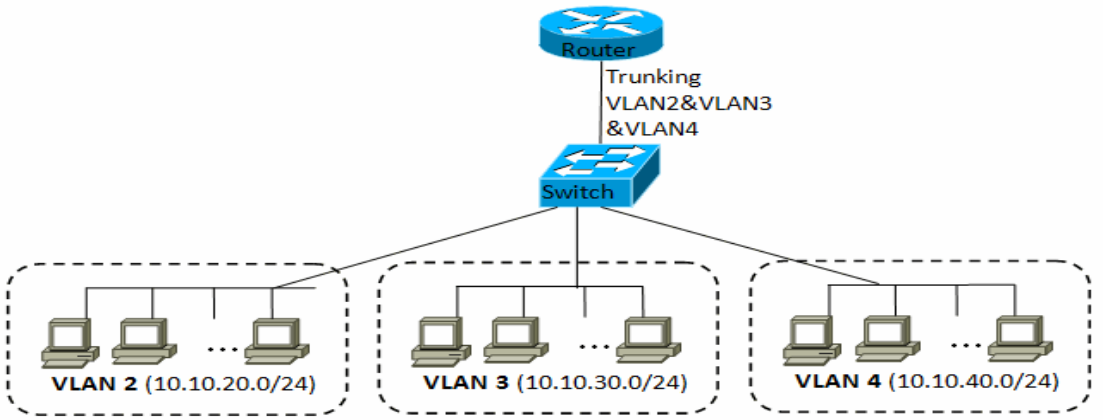
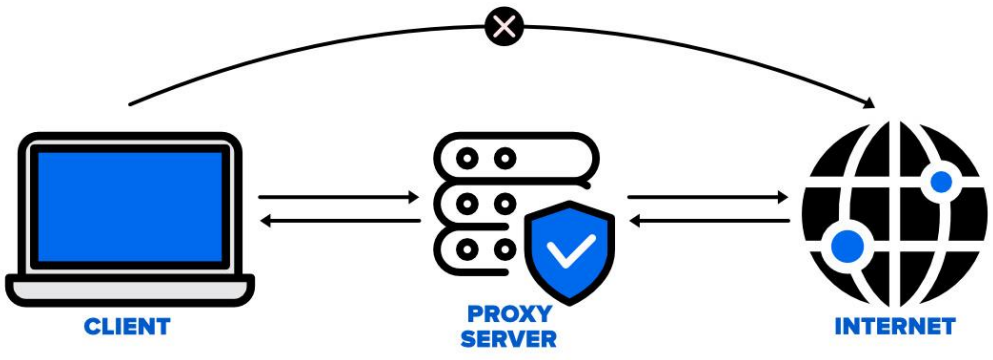
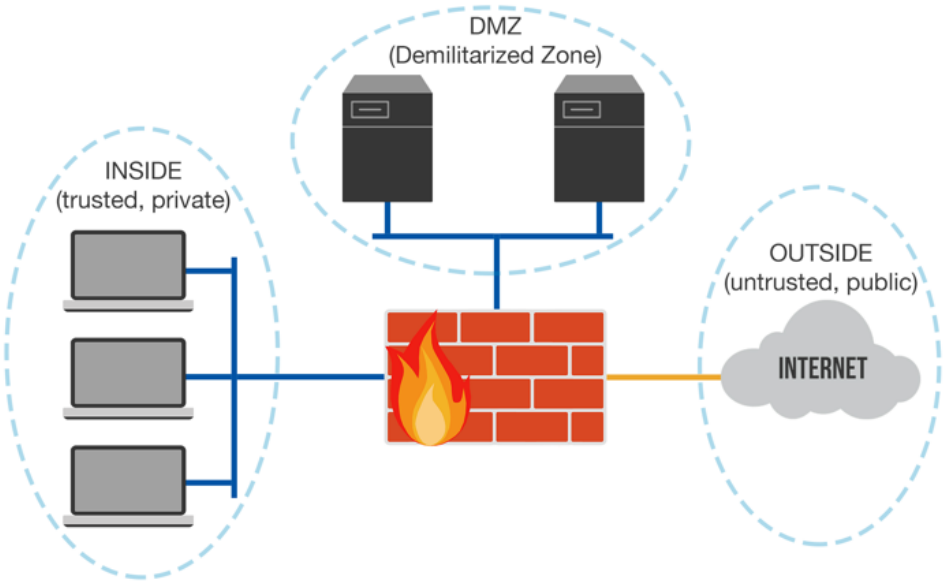


Figure 12.5. 802.1Q trunk between the router and the switch

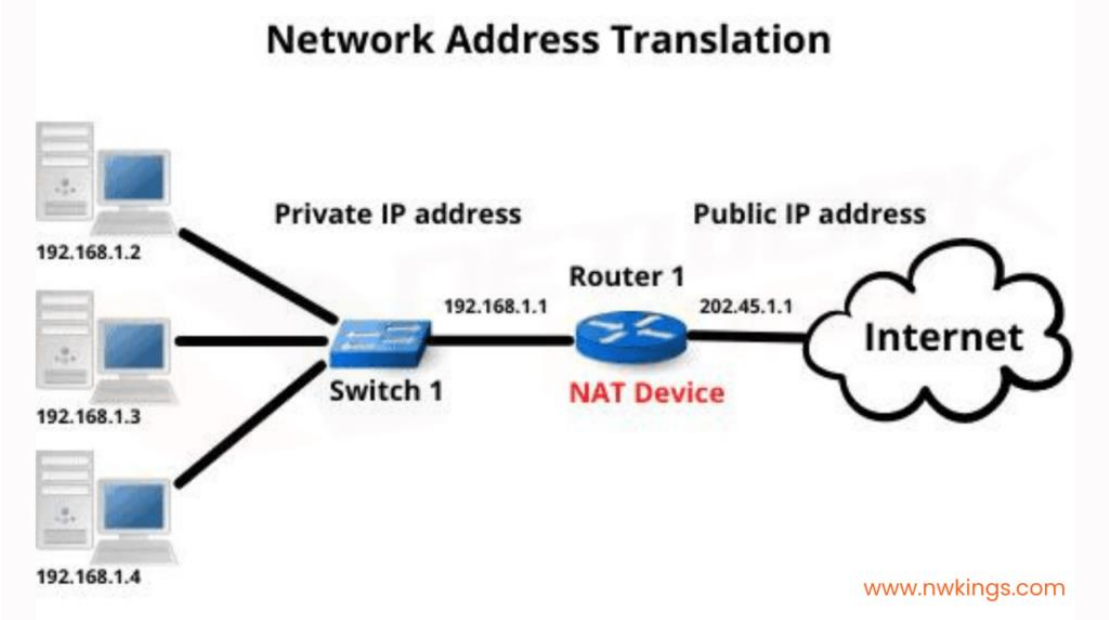
Regulate Access to the Internet via Proxy Server Do not allow network users to access the Internet unchecked. Pass all requests through a transparent proxy, and use it to control and monitor user behavior. Ensure that outbound connections are actually performed by a human and not a bot or other automated mechanism. Whitelist domains to ensure corporate users can only access websites you have explicitly approved.



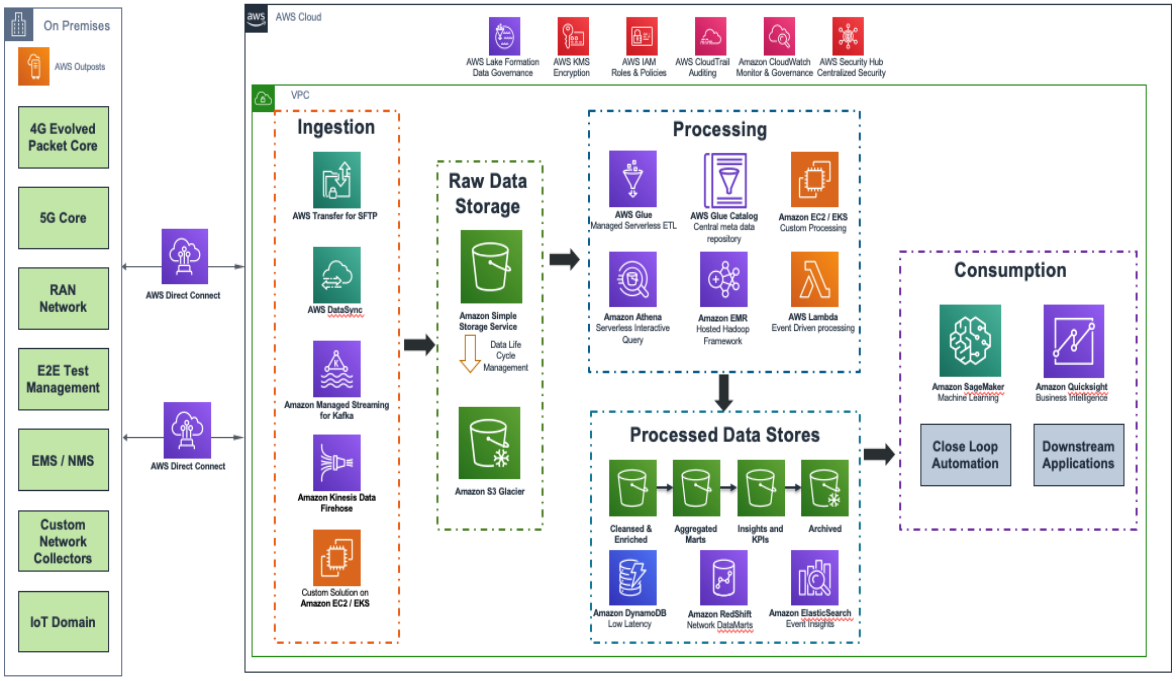
Place Security Devices Correctly Place a firewall at every junction of network zones, not just at the network edge. If you can't deploy full-fledged firewalls everywhere, use the built-in firewall functionality of your switches and routers. Deploy anti-DDoS devices or cloud services at the network edge. Carefully consider where to place strategic devices like load balancers – if they are outside the Demilitarized Zone (DMZ), they won't be protected by your network security apparatus.



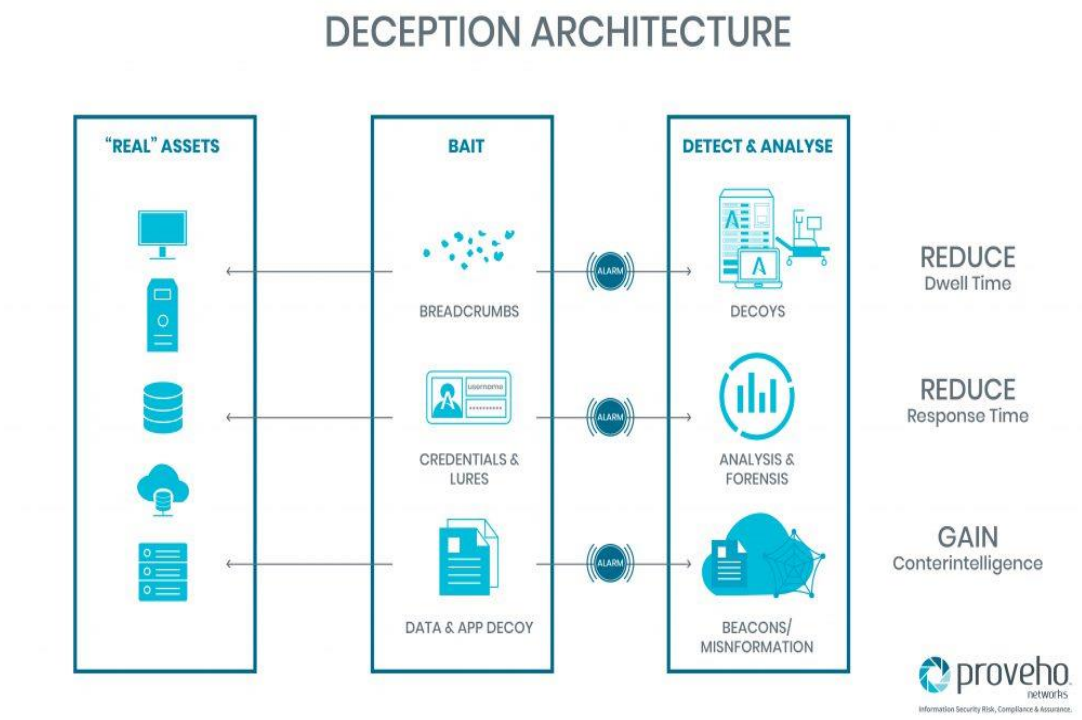
Use Network Address Translation Network Address Translation (NAT) lets you translate internal IP addresses into addresses accessible on public networks. You can use it to connect multiple computers to the Internet using a single IP address. This provides an extra layer of security, because any inbound or outgoing traffic has to go through a NAT device, and there are fewer IP addresses which makes it difficult for attackers to understand which host they are connecting to.



Monitor Network Traffic Ensure you have complete visibility of incoming, outgoing and internal network traffic, with the ability to automatically detect threats, and understand their context and impact. Combine data from different security tools to get a clear picture of what is happening on the network, recognizing that many attacks span multiple IT systems, user accounts and threat vectors.



Use Deception Technology No network protection measures are 100% successful, and attackers will eventually succeed in penetrating your network. Recognize this and place deception technology in place, which creates decoys across your network, tempting attackers to "attack" them, and letting you observe their plans and techniques. You can use decoys to detect threats in all stages of the attack lifecycle: data files, credentials and network connections.



Network Security Factors

Confidentiality, Integrity and Authentication

Message Confidentiality

At its core, data and message confidentiality is about keeping sensitive information secret. In our digital age, this means protecting personal details, financial transactions, and private communications from unauthorized access. **But how is this achieved? The answer lies in cryptography. Encryption** is a method of converting readable data into a coded form that can only be decoded and read by someone with the proper encryption key.

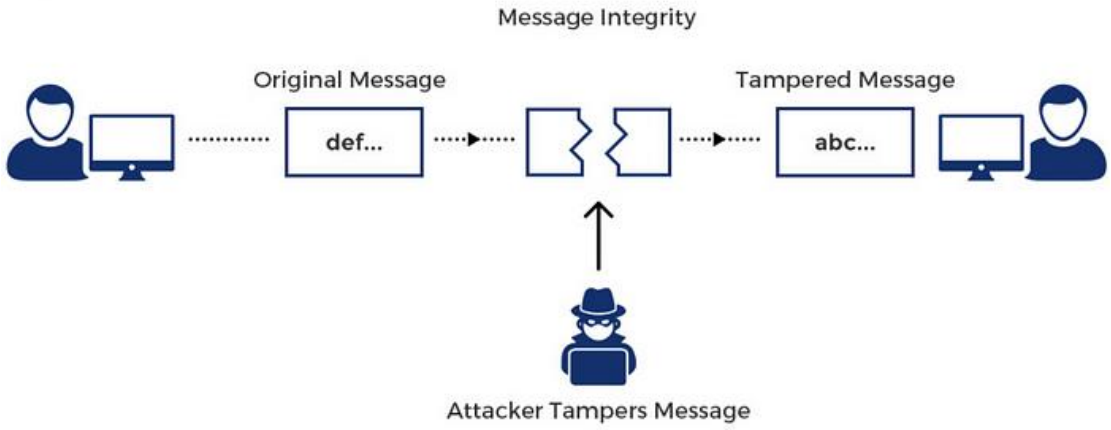
The magic of encryption, Encryption is not a modern invention; its history traces back to ancient times. However, in the digital realm, encryption algorithms have become incredibly sophisticated. These algorithms use complex mathematics to create a cipher – the rules of turning plain text into encrypted text, also known as ciphertext. When data is encrypted, it looks like gibberish to anyone who doesn't have the decryption key, which is the secret to turning the ciphertext back into readable form.

Despite the robustness of encryption, there are challenges in maintaining data and message confidentiality. These include:

- **Key Management:** Safely storing and managing encryption keys are a complex task. If keys are lost or stolen, the data they protect may be compromised.
- **Emerging Threats:** As technology advances, so do the methods of cyber attackers. Encryption must continually evolve to stay ahead of new threats.
- **Legal and Ethical Considerations:** Encryption can also pose challenges for law enforcement agencies, leading to ongoing debates about the balance between privacy and security.

Message Integrity

Integrity means ensuring that our data in transit or after receiving is intact or not. Or we can say that, it is the ability to make sure that data or information has not been changed or modified with. So for this purpose we use Cryptographic techniques like hash functions. Hash Functions are basically used to check the integrity of the data by finding out the changes to the data.



Message Authentication

Message authentication is used widely in information security to ensure that data integrity and authenticity are preserved while in transit and allow the receiver to verify the source of messages. Common message authentication mechanisms include MACs, Authenticated Encryption (AE), and digital signatures.

Some cryptographers distinguish between "message authentication without secrecy" systems – which allow the intended receiver to verify the source of the message, but they don't bother hiding the plaintext contents of the message – from authenticated encryption systems. Some cryptographers have researched subliminal channel systems that send messages that appear to use a "message authentication without secrecy" system, but in fact also transmit a secret message.

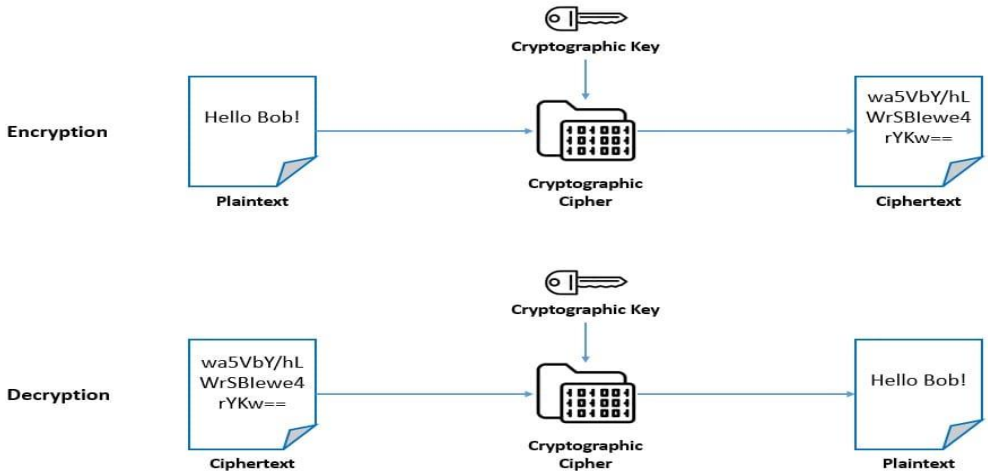
Message Confidentiality

Confidentiality with Symmetric-Key Cryptography

Confidentiality with Asymmetric-Key Cryptography

Message Confidentiality

Although cryptography is a wider field of study, we often view it as the science of encrypting and decrypting information. **Encryption and decryption are widely used to protect data** in transit and data at rest. Although it depends on the algorithm in use, there is a common scheme to it:



The unencrypted data is called plaintext, and the encrypted data is called ciphertext. A cipher is an **algorithm for performing encryption and decryption**. The cipher usually depends on a piece of information called the key for performing encryption and decryption.

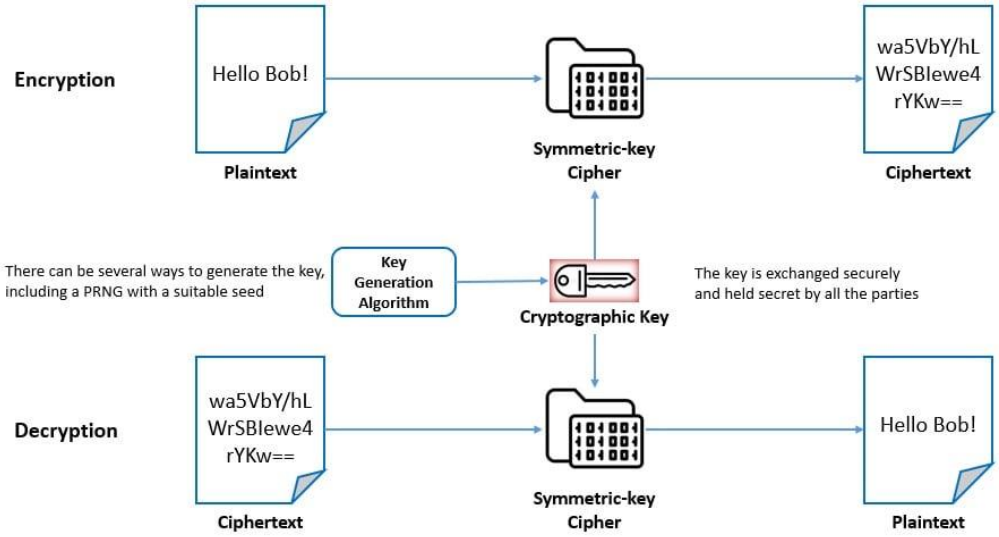
Cryptographic ciphers are often categorized based on their working principles. There are two broad categorizations of modern ciphers. There can be other categorizations based on the internal properties of the algorithm, but we'll discuss the ones most widely used.

Categorization Based on Keys

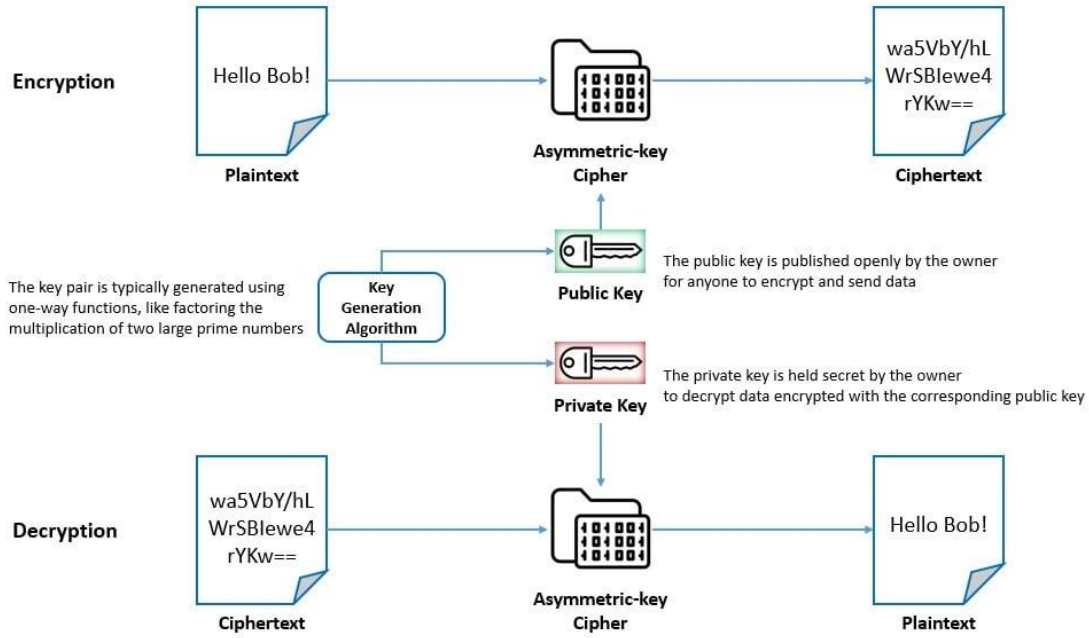
As we've seen earlier, a cipher uses a cryptographic key to perform encryption and decryption. The first categorization of ciphers is based on

whether it **uses the same key for both encryption and decryption or uses different keys** for them:

Symmetric-key cipher is also known as private-key encryption and **uses the same cryptographic keys for both** the encryption of plaintext and the decryption of ciphertext. The key here represents a shared secret between two or more parties. These ciphers have a smaller key size and hence **require less storage space and provide faster transmission**. But they require all parties to have access to the secret key in advance, which is a major drawback of these algorithms.



Asymmetric-key cipher is also known as public-key encryption and **uses pairs of related keys for encryption and decryption**. Each key pair consists of a public key used for encryption and a private key used for decryption:

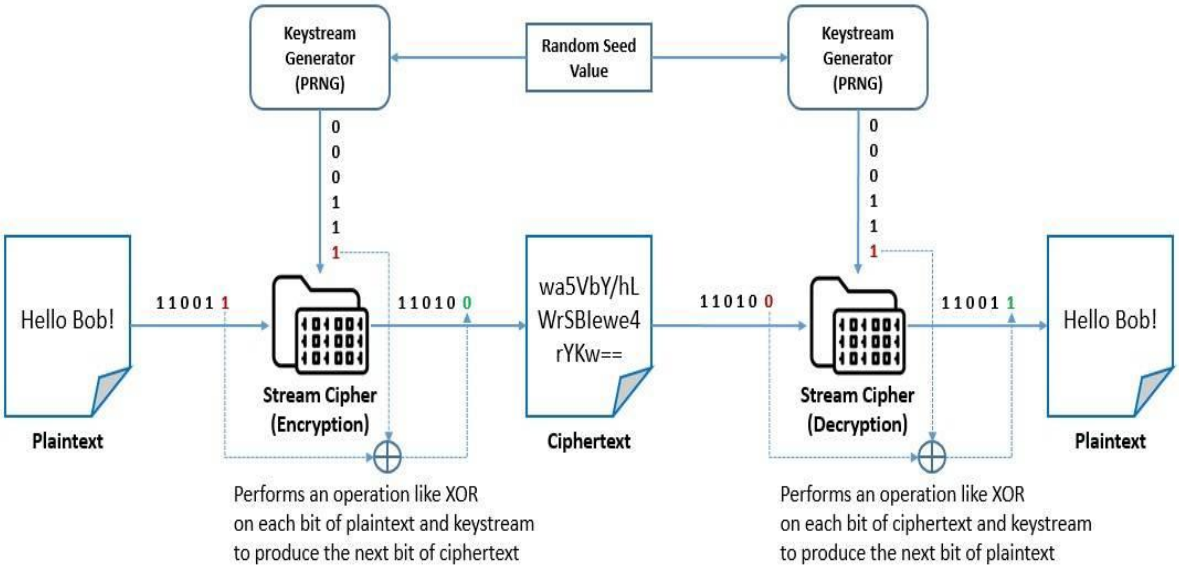


The key pairs are generated with cryptographic algorithms based on one-way functions. Here, the public key is openly distributed but the private key is held secret by the owner of the key pair. However, **these algorithms are too slow for many practical purposes** like bulk encryption and decryption.

Categorization Based on Processing

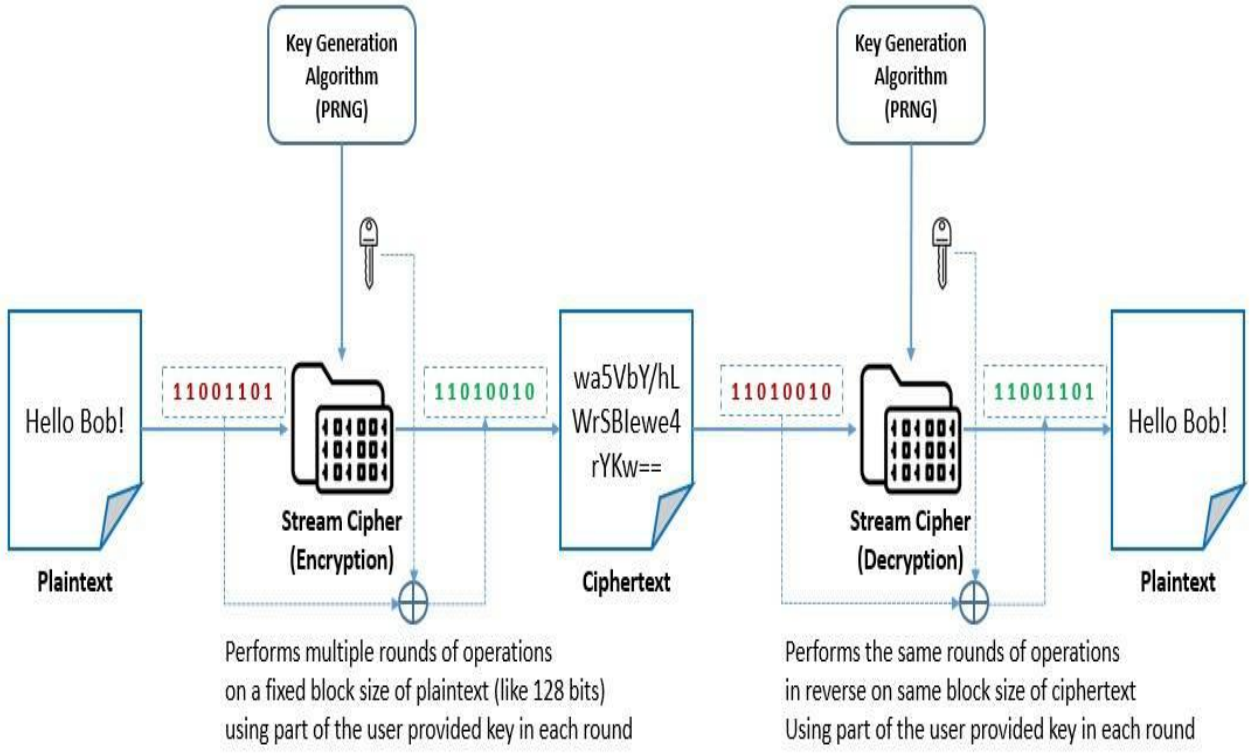
The second way in which we typically categorize ciphers is based on how it works on the plaintext. The board categories are whether they work on streams or blocks of data:

A stream cipher is a symmetric-key cipher that **combines the plaintext stream with a pseudorandom cipher digit stream or keystream** to generate the ciphertext stream. It encrypts plaintext digits one at a time with the corresponding keystream digit to produce the ciphertext digit:



The **keystream is typically generated serially from a random seed** value using digital shift registers. In a synchronous stream cipher, the key stream is generated independently of the plaintext and ciphertext. In a self-synchronizing stream cipher, the previous ciphertext digits are used to compute the keystream. Stream ciphers typically **execute at higher speeds and have lower hardware complexity**. However, through their operations, they only employ confusion. Hence, they are generally susceptible to stream cipher attacks. These threats can be largely mitigated by a better selection of keystreams.

Block cipher is a deterministic algorithm that **operates on a fixed-length group of bits called blocks**. It works with a pair of algorithms, one for encryption and the other for decryption. Both algorithms take an input block of size *n*-bits and a key of size *k*-bits to produce an *n*-bits output block:



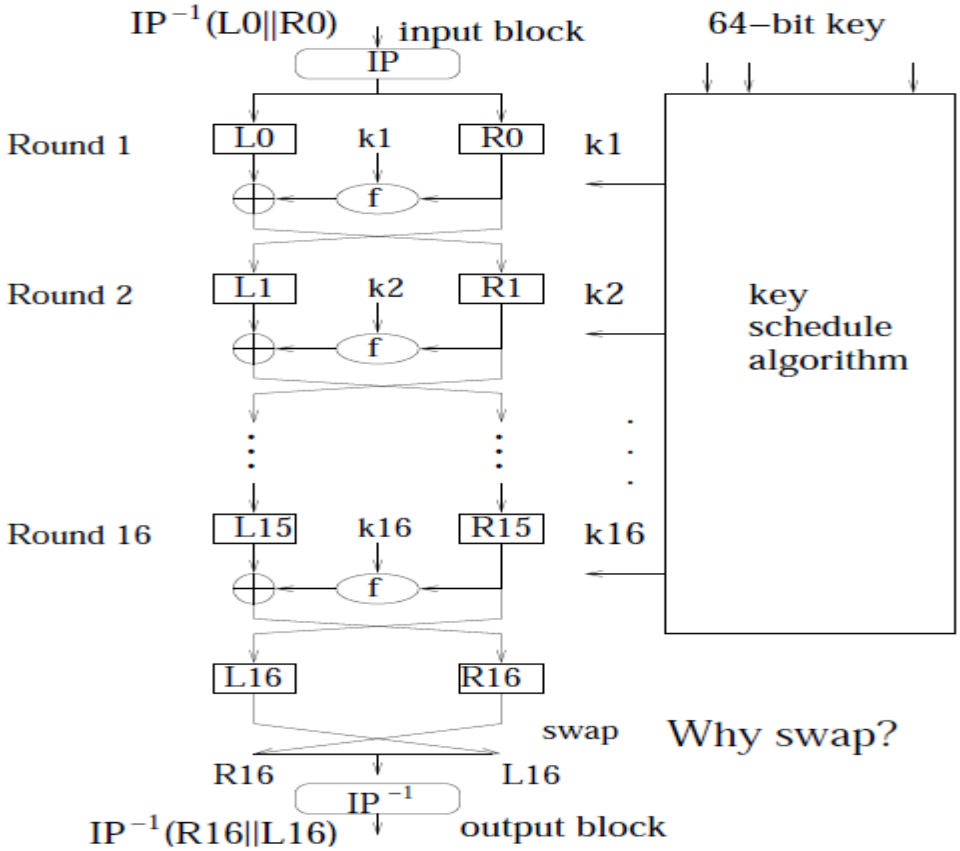
These are basically iterated product ciphers, which **carry out encryption in multiple rounds**, each of which uses a different subkey derived from the original key. For better security, some of the block ciphers also employ operations like substitutions for confusion and permutations for diffusion.

Block ciphers are only suitable for the secure transformation of data in blocks. For applying block ciphers on variable-length messages, **several modes of operation have been developed**. These typically provide security like confidentiality, authenticity, or even both in some modes.

Confidentiality with Symmetric-Key Cryptography

Data Encryption Standard (DES)

DES is a block cipher; it encrypts data in 64-bit blocks. A 64-bit block of plaintext goes in one end of the algorithm and a 64-bit block of ciphertext comes out the other end. DES is a symmetric algorithm: The same algorithm and key are used for both encryption and decryption (except for minor differences in the key schedule). The key length is 56 bits. (The key is usually expressed as a 64-bit number, but every eighth bit is used for parity checking and is ignored. These parity bits are the least- significant bits of the key bytes.) The key can be any 56-bit number and can be changed at any time. All security rests within the key.
At its simplest level, the algorithm is nothing more than a combination of the two basic techniques of encryption: confusion and diffusion. The fundamental building block of DES is a single combination of these techniques (a substitution followed by a permutation) on the text, based on the key. This is known as a round. DES has 16 rounds; it applies the same combination of techniques on the plaintext block 16 times.



Outline of the Algorithm

The basic process in enciphering a 64-bit data block using the DES consists of:

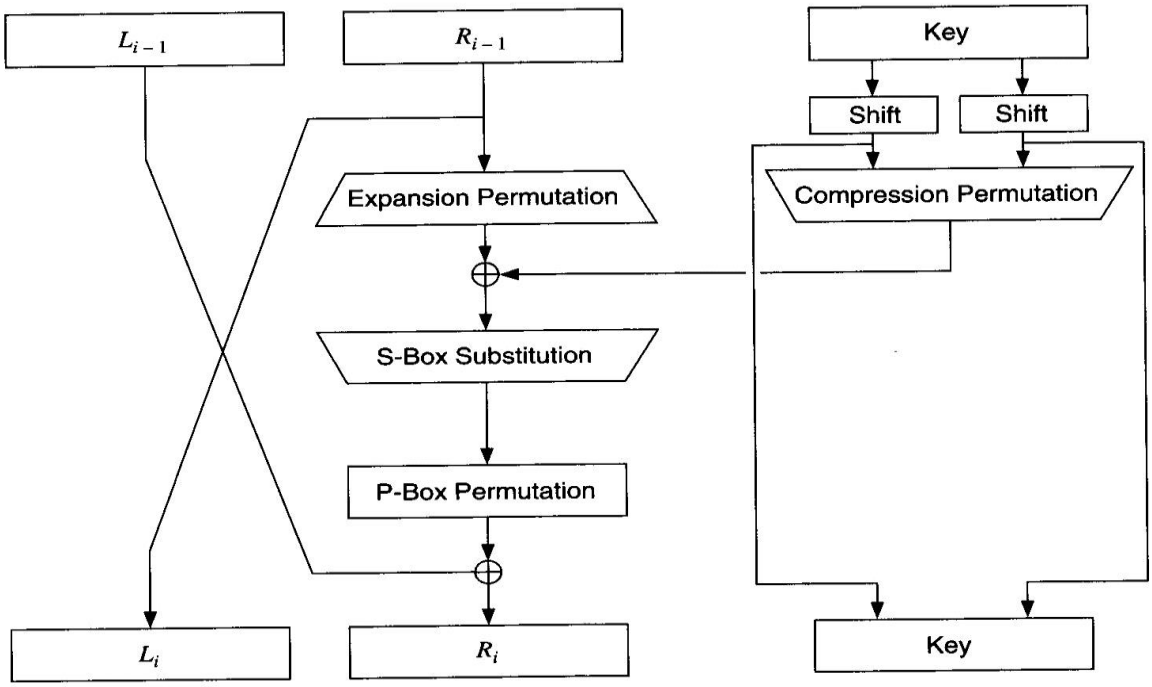
- an initial permutation (IP)
- 16 rounds of a complex key dependent calculation f
- final permutation, being the inverse of IP

In each round (see Figure 2,3,4,5), the key bits are shifted, and then 48 bits are selected from the 56 bits of the key. The right half of the data is expanded to 48 bits via an expansion permutation, combined with 48 bits of a shifted and permuted key via an XOR, sent through 8 S-boxes producing 32 new bits, and permuted again. These four operations make up Function f. The output of Function f is then combined with the left half via another XOR. The result of these operations becomes the new right half; the old right half becomes the new left half.

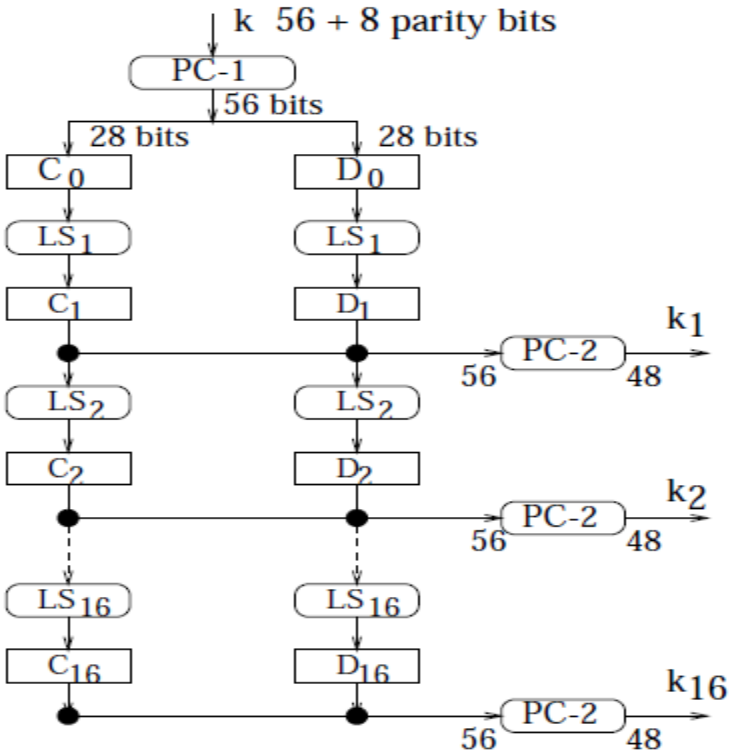
If Bi is the result of the ith iteration, Li and Ri are the left and right halves of Bi, Ki is the 48-bit key for round i, and f is the function that does all the substituting and permuting and XORing with the key, then a round looks like:

$$Li = R_{j-1}$$

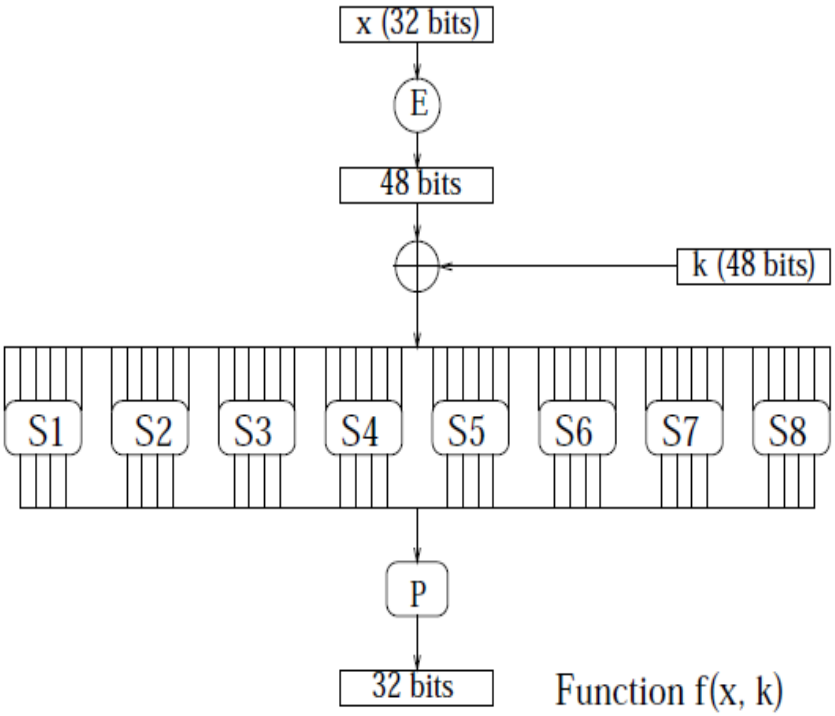
$$Ri = L_{i-1} \text{ Xor } f(R_{i-1}, K_i)$$



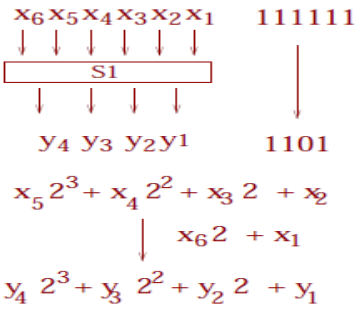
One round of DES



16th key generation



f-function



column	0	1	2	3
0	14	0	4	15
1	4	15	1	12
2	13	7	14	8
3	1	4	8	2
4	2	14	13	4
5	15	2	6	9
6	11	13	2	1
7	8	1	11	7
8	3	10	15	5
9	10	6	12	11
10	6	12	9	3
11	12	11	7	14
12	5	9	3	10
13	9	5	10	0
14	0	3	5	6
15	7	8	0	13

S-Boxes in F-function

The Initial Permutation

The initial permutation occurs before round 1; it transposes the input block as described in Table 1. This table, like all the other tables in this lecture, should be read left to right, top to bottom. For example, the initial permutation moves bit 58 of the plaintext to bit position 1, bit 50 to bit position 2, bit 42 to bit position 3, and so forth. The initial permutation and the corresponding final permutation do not improve DES's security, just make DES more complex.

Table 1 Initial Permutation

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

The Key Transformation

Initially, the 64-bit DES key is reduced to a 56-bit key by ignoring every eighth bit. Let us call this operation PC1. This is described in Table 2. PC2 is the operation which reduces the 56-bits key to a 48-bits subkey for each of the 16 rounds of DES. These subkeys, Ki, are determined in the following manner. PC1 splits the key bits into 2 halves (C and D), each 28-bits. The halves C and D are circularly shifted left by either one or two bits, depending on the round. This shift is given in Table

3.◌ after being shifted, 48 out of the 56 bits are selected. This is done by an operation called compression permutation, it permutes the order of the bits as well as selects a subsets of bits. Table 4 defines the compression permutation.

Table 2 Key Permutation

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

Table 3 Number of Key Bits Shifted per Round

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Table 4 Compression Permutation

14,	17,	11,	24,	1,	5,	3,	28,	15,	6,	21,	10,
23,	19,	12,	4,	26,	8,	16,	7,	27,	20,	13,	2,
41,	52,	31,	37,	47,	55,	30,	40,	51,	45,	33,	48,
44,	49,	39,	56,	34,	53,	46,	42,	50,	36,	29,	32.

The Expansion Permutation

This operation expands the right half of the data, R_i, from 32 bits to 48 bits. Because this operation changes the order of the bits as well as repeating certain bits, it is known as an expansion permutation. This operation has two purposes: It makes the right half the same size as the key for the XOR operation and it provides a longer result that can be compressed during the substitution operation.

However, neither of those is its main cryptographic purpose. For each 4-bit input block, the first and fourth bits each represent two bits of the output block, while the second and third bits each represent one bit of the output block. Table 5 shows which output positions correspond to which input positions. For example, the bit in position 3 of the input block moves to position 4 of the lutput

block, and the bit in position 21 of the input block moves to positions 30 and 32 of the output block.

Table 5 Expansion Permutation

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

The S-Box Substitution

After the compressed key is XORed with the expanded block, the 48-bit result moves to a substitution operation. The substitutions are performed by eight substitution boxes, or S-boxes. Each S-box has a 6-bit input and a 4-bit output, and there are eight different S-boxes. The 48 bits are divided into eight 6-bit sub-blocks. Each separate block is operated on by a separate S-box: The first block is operated on by S-box 1, the second block is operated on by S-box 2, and so on. Each S-box is a table of 4 rows and 16 columns. Each entry in the box is a 4-bit number. The 6 input bits of the S-box specify under which row and column number to look for the output. Table 6 shows all eight S-boxes. The input bits specify an entry in the S-box in a very particular manner. Consider an S-box input of 6 bits, labeled bi, b2, b3, b, b, and b6. Bits b, and b6are combined to form a 2-bit number, from 0 to 3, which corresponds to a row in the table. The middle 4 bits, b2 through b5, are combined to form a 4-bit number, from 0 to 15, which corresponds to a column in the table. For example, assume that the input to the sixth S-box (i.e., bits 31 through 36 of the XOR function) is 110011. The first and last bits combine to form 11, which corrspends to row 3 of the sixth S-box. The middle 4 bits combine to form 1001, which corresponds to the column 9 of the same S-box. The entry under row 3, column 9 of S-box 6 is 14. (Remember to count rows and columns from 0 and not from 1.) The value 1110 is substituted for 110011. The S-box substitution is the critical step in DES. The algorithm's other operanons are linear and easy to analyze. The S-boxes are nonlinear and, more than any.hing else, give DES its security. The result of this substitution phase is eight 4-bit blocks which are recombined into a single 32-bit block. This block moves to the next step: the P-box permutation.

Table 6 –Boxes

S-box 1:

14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,
0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,
4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,
15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13,

S-box 2:

15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,
3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,
0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,
13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9,

S-box 3:

10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,
13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,
13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,
1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12,

S-box 4:

7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,
13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,
10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,
3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14,

S-box 5:

2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,
14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,
41, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,
11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3,

S-box 6:

12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,
10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,
9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,
4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13,

S-box 7:

4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,
13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,
1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,
6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12,

S-box 8:

13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,
1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,
7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,
-2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11

The P-Box Permutation

The 32-bit output of the S-box substitution is permuted according to a P-box. This permutation maps each input bit to an output position; no bits are used twice and no bits are ignored. Table 7 shows the position to which each bit moves. For example, bit 21 moves to bit 4. While bit 4 moves to bit 3 1.

Table 7 P-Box Permutation

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

Finally, the result of the P-box permutation is XORed with the left half of the initial 64-bit block. Then the left and right halves are switched and another round begins.

The Final Permutation

The final permutation is the inverse of the initial permutation and is described in Table 8. Note that the left and right halves are not exchanged after the last round of DES; instead the concatenated block $R_{16}L_{16}$ is used as the input to the final permutation. There's nothing going on here; exchanging the halves and shifting around the permutation would yield exactly the same result. This is so that the algorithm can be used to both encrypt and decrypt.

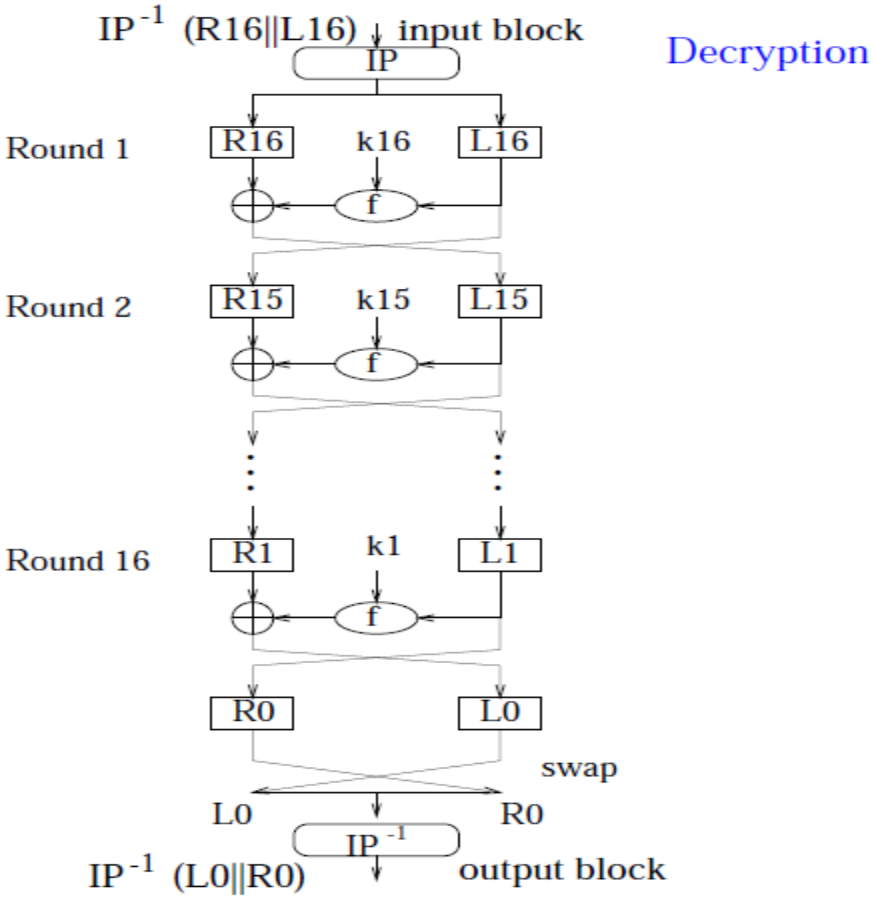
Table 8 Final Permutation

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.

Decrypting DES

After all the substitutions, permutations, XORs, and shifting around, you might think that the decryption algorithm is completely different and just as confusing as the encryption algorithm. On the contrary, the various operations were chosen to produce a very useful property: The same algorithm works for both encryption and

decryption. With DES it is possible to use the same function to encrypt or decrypt a block. The only difference is that the keys must be used in the reverse order. That is, if the encryption keys for each round are K1, K2, K3, . . . , K16, then the decryption keys are K16, K15, K14, . . . , K1,. The algorithm that generates the key used for each round is circular as well. The key shift is a right shift and the number of positions shifted is 0, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1.



DES Decryption

Confidentiality with Asymmetric-Key Cryptography

Exponential cipher

Both schemes encipher a message block $M \in [0, n-1]$ by computing the exponential: $C = M^e \bmod n$, Where e and n are the key to the enciphering transformation. M is restored by the same operation, but using a different exponent d for the key: $M = C^d \bmod n$.

Enciphering and deciphering can be implemented using the fast exponentiation algorithm:

$C = fast\_exp(M, e, n)$

$M = fast\_exp(C, d, n)$

Thm: Given e, d, M such that $ed \bmod \phi(n) = 1$, $M \in [0, n-1]$,$gcd(M, n) = 1$, Then $(M^e \bmod n)^d \bmod n = M$.

Note that by symmetry, enciphering and deciphering are commutative and mutual inverses; thus, $(M^d \bmod n)^e \bmod n = M^{de} \bmod n = M$

Given $\phi(n)$, it is easy to generate a pair (e, d) such that $ed \bmod \phi(n) = 1$. This is done by first choosing d relatively prime to $\phi(n)$, and then computing e as .... $e = inv(d, \phi(n))$

Because e and d both are symmetric, we could also pick e and compute $d = inv(e, \phi(n))$.

Given e, it is easy to compute d (or vice versa) if $\phi(n)$ is known. But if e and n can be released without giving away $\phi(n)$ or d, then the deciphering transformation can be kept secret, while the enciphering transformation is made public. It is the ability to hide $\phi(n)$ that distinguishes the two schemes.

RSA description and algorithm

RSA stands for Rivest, Shamir, and Adleman, they are the inventors of the RSA cryptosystem. RSA is one of the algorithms used in PKI (Public Key Infrastructure), asymmetric key encryption scheme. RSA is a block chiper, it

encrypt message in blocks (block by block). The common size for the key length now is 1024 bits for P and Q, therefore N is 2048 bits, if the implementation (the library) of RSA is fast enough, we can double the key size.

Key Generation Algorithm

1. Generate two large random primes, *p* and *q*, of approximately equal size such that their product n = pq is of the required bit length, e.g. 1024 bits.
2. Compute n = pq and (φ) phi = (p-1)(q-1).
3. Choose an integer *e*, 1 < e < phi, such that gcd(e, phi) = 1.
4. Compute the secret exponent *d*, 1 < d < phi, such that ed ≡ 1 (mod phi).
5. The public key is (n, e) and the private key is (n, d). Keep all the values d, p, q and phi secret.

- n is known as the *modulus*.
- e is known as the *public exponent* or *encryption exponent* or just the *exponent*.
- d is known as the *secret exponent* or *decryption exponent*.

Encryption Sender A does the following:-

1. Obtains the recipient B's public key (n, e).
2. Represents the plaintext message as a positive integer *m*.
3. Computes the ciphertext $c = m^e \bmod n$.
4. Sends the ciphertext *c* to B.

Decryption Recipient B does the following:-

1. Uses his private key (n, d) to compute $m = c^d \bmod n$.
2. Extracts the plaintext from the message representative *m*.

Example 1

This is an extremely simple example using numbers you can work out on a pocket calculator (those of you over the age of 35 45 can probably even do it by hand).

1. Select primes p=11, q=3.
2. n = pq = 11.3 = 33
   phi = (p-1)(q-1) = 10.2 = 20
3. Choose e=3
   Check gcd(e, p-1) = gcd(3, 10) = 1 (i.e. 3 and 10 have no common factors except 1),
   and check gcd(e, q-1) = gcd(3, 2) = 1
   therefore gcd(e, phi) = gcd(e, (p-1)(q-1)) = gcd(3, 20) = 1
4. Compute d such that ed ≡ 1 (mod phi)
   i.e. compute d = e^{-1} mod phi = 3^{-1} mod 20
   i.e. find a value for d such that phi divides (ed-1)
   i.e. find d such that 20 divides 3d-1.
   Simple testing (d = 1, 2, ...) gives d = 7
   Check: ed-1 = 3.7 - 1 = 20, which is divisible by phi.
5. Public key = (n, e) = (33, 3)
   Private key = (n, d) = (33, 7).

This is actually the smallest possible value for the modulus n for which the RSA algorithm works. Now say we want to encrypt the message m = 7,
c = m^e mod n = 7^3 mod 33 = 343 mod 33 = 13.
Hence the ciphertext c = 13.

To check decryption we compute
m' = c^d mod n = 13^7 mod 33 = 7.
Note that we don't have to calculate the full value of 13 to the power 7 here.

One way of calculating m' is as follows:-

$m' = 13^7 \bmod 33 = 13^{(3+3+1)} \bmod 33 = 13^3.13^3.13 \bmod 33$

$= (13^3 \bmod 33).(13^3 \bmod 33).(13 \bmod 33) \bmod 33$

$= (2197 \bmod 33).(2197 \bmod 33).(13 \bmod 33) \bmod 33$

$= 19.19.13 \bmod 33 = 4693 \bmod 33$

$= 7.$

Example 2:

1) Generate two large prime numbers, p and q To make the example easy to follow I am going to use small numbers, but this is not secure. To find random primes, we start at a random number and go up ascending odd numbers until we find a prime. Lets have:

p = 7

q = 19

2) Let n = pq ---------- n = 7 * 19 = 133

3) Let PHi = (p - 1)(q - 1)

PHi = (7 - 1)(19 - 1) = 6 * 18 = 108

4) Choose a small number, e coprime to PHi

e coprime to PHi, means that the largest number that can exactly divide both e and m (their greatest common divisor, or GCD) is 1. Euclid's algorithm is used to find the GCD of two numbers, but the details are omitted here.

e = 2 => GCD(e, 108) = 2 (no)

e = 3 => GCD(e, 108) = 3 (no)

e = 4 => GCD(e, 108) = 4 (no)

e = 5 => GCD(e, 108) = 1 (yes!)

5) Find d, such that de % m = 1

This is equivalent to finding d which satisfies de = 1 + nPHi where n is any integer. We can rewrite this as d = (1 + nPHi) / e. Now we work through values of n until an integer solution for e is found:

n = 0 => d = 1 / 5 (no)

n = 1 => d = 109 / 5 (no)

n = 2 => d = 217 / 5 (no)

n = 3 => d = 325 / 5

= 65 (yes!)

To do this with big numbers, a more sophisticated algorithm called extended Euclid must be used.

Public Key

n = 133 && e = 5

Secret Key

n = 133 && d =65

Encryption The message must be a number less than the smaller of p and q. However, at this point we don't know p or q, so in practice a lower bound on p and q must be published. This can be somewhat below their true value and so isn't a major security concern. For this example, let's use the message "6".

C = Pe % n

= 65 % 133

= 7776 % 133

= 62

Decryption This works very much like encryption, but involves a larger exponentiation, which is broken down into several steps.

P = Cd % n

= 6265 % 133

= 62 * 6264 % 133

= 62 * (622)32 % 133

= 62 * 384432 % 133

= 62 * (3844 % 133)32 % 133

= 62 * 12032 % 133

We now repeat the sequence of operations that reduced 6265 to 12032 to reduce the exponent down to 1.

= 62 * 3616 % 133

= 62 * 998 % 133

= 62 * 924 % 133

= 62 * 852 % 133

= 62 * 43 % 133

= 2666 % 133

= 6

Diffie-Hellman algorithm

The Diffie-Hellman algorithm is being used to establish a shared secret that can be used for secret communications while exchanging data over a public network using the elliptic curve to generate points and get the secret key using the parameters.

- For the sake of simplicity and practical implementation of the algorithm, we will consider only 4 variables, one prime P and G (a primitive root of P) and two private values a and b.
- P and G are both publicly available numbers. Users (say Alice and Bob) pick private values a and b and they generate a key and exchange it publicly. The opposite person receives the key and that generates a secret key, after which they have the same secret key to encrypt.

Step-by-Step explanation is as follows:

Alice	**Bob**
Public Keys available = P, G	Public Keys available = P, G

Alice	**Bob**
Private Key Selected = a	Private Key Selected = b
Key generated = $x=G^{a} \bmod P$	Key generated = $y=G^{b} \bmod P$
Exchange of generated keys takes place	
Key received = y	key received = x
Generated Secret Key = $ka=y^{a} \bmod P$	Generated Secret Key = $kb=x^{b} \bmod P$
Algebraically, it can be shown that $ka=kb$	
Users now have a symmetric secret key to encrypt	

Example:

Step 1: Alice and Bob get public numbers P = 23, G = 9

Step 2: Alice selected a private key a = 4 and

Bob selected a private key b = 3

Step 3: Alice and Bob compute public values

Alice: x =(9^4 mod 23) = (6561 mod 23) = 6

Bob: y = (9^3 mod 23) = (729 mod 23) = 16

Step 4: Alice and Bob exchange public numbers

Step 5: Alice receives public key y =16 and

Bob receives public key x = 6

Step 6: Alice and Bob compute symmetric keys

Alice: ka = y^a mod p = 65536 mod 23 = 9

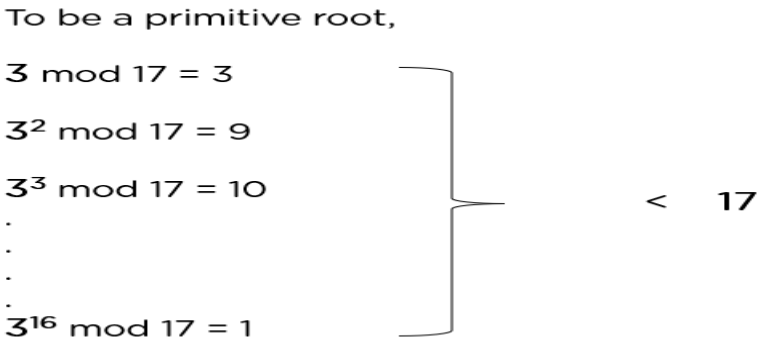
Bob: kb = x^b mod p = 216 mod 23 = 9

Step 7: 9 is the shared secret.

Practical Implementation

The steps needed to test the Diffie-Hellman key exchange are as follows:

Step 1: You choose the prime number q to be 17. For its primitive root, you select the value of α to be 3, since it satisfies the following criteria:



Step 2: You assume the sender's private key Xa to be 15. The public key can be calculated as Ya = 315 mod 17 = 6. The key pair for our sender becomes {15, 6}.

For the receiver's end, you assume private key Xb to be 13. The public key is calculated as Yb = 313 mod 17 = 12. The key pair for the receiver is now {13, 12}.

Step 3: The secret key generated on the sender's side is K = 1215 mod 17 = 10.

The secret key generated from the receiver's side is K = 613 mod 17 = 10.

Since both the keys generated are equal, the Diffie-Hellman exchange is valid, and the secret key '10' can encrypt messages between the users.

Now that you know how the key exchange works, let us learn about its applications in the real world.

Message Integrity

Message and Message Digest

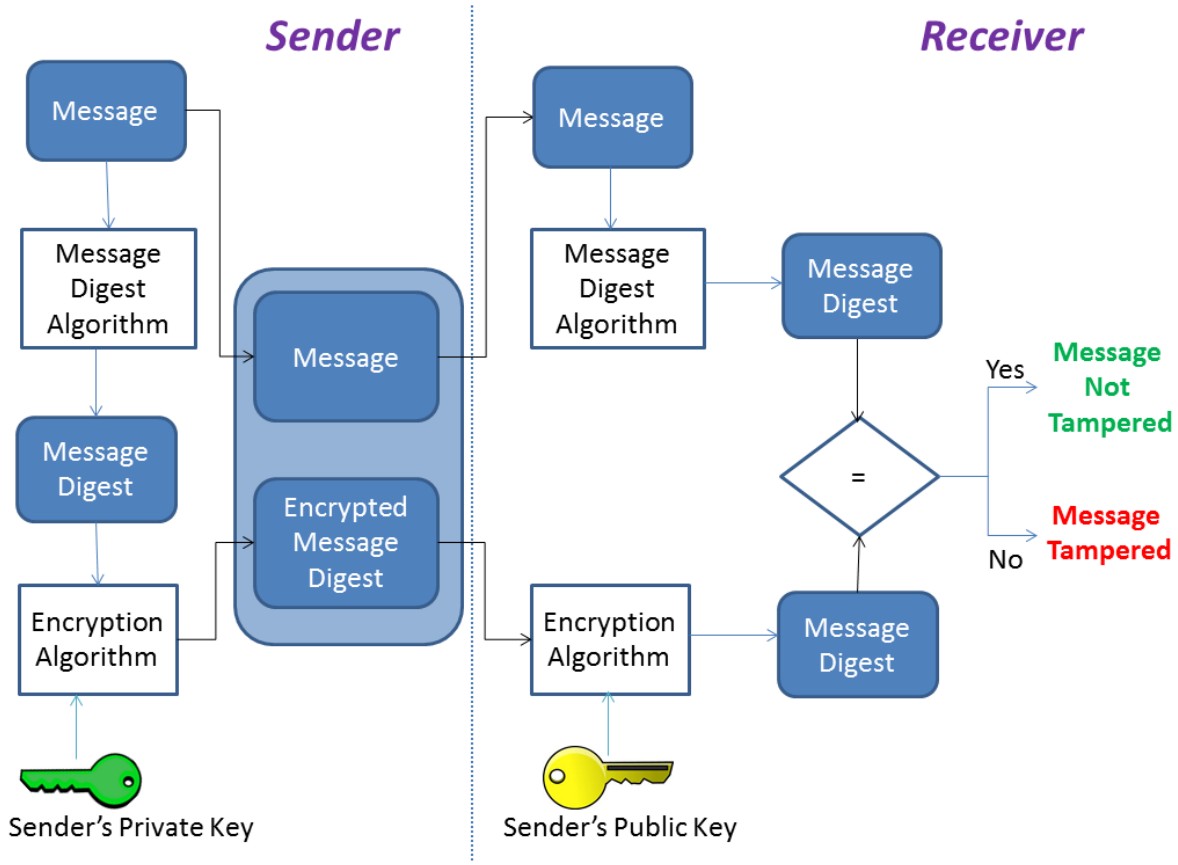
Hash Function Criteria

Hash Algorithms: SHA-1

Biometric parameters

Message and Message Digest

A Message is any information that flows via the network, such as files, emails, and financial transactions, from one device to another or from a set of devices. When a message is sent over a network, it must be secure to ensure that it is safe from anybody sitting in the middle listening to the conversation and having the ability to access, alter, or modify the message.



What is a Message Digest or Hash Value?

A message digests or hash value is a numeric string generated using the cryptographic hash function. The message string is passed to the hash function. Hash function computes a unique hash value for the provided message and this hash value acts as digital fingerprint of the message.

The cryptographic function creates the message digest, which may then be encrypted to give a second layer of security. This function is one-way; it can only be used to generate the message digest from the message, not the other way around. If the message digest is created using a symmetric key then it is known as MAC or Message Authentication Code. Encrypted message digests work as digital fingerprints and the receiver needs to decrypt the digests first to compare them.

Properties of a Message Digest

- The message digest is always a unique numeric hash value. It cannot be the same for two or more messages. Once the message digest value is generated for a message, it's only associated with that message.
- The size of the hash value is fixed. For any length of message a fixed length hash value is generated and that depends on the implementation of the hash function.
- If you use the hash function multiple times for the same function again and again, then you only get the same digest value each time. For example, if you send "hii" to the hash function the generated message digest value will be the same each time when you pass "hii" to the hash function. It's not going to change.
- We cannot generate the message by passing the message digest value to the hash function. A hash function is a one-way function.

How Does a Message Digest Work?

Message digests works as a digital fingerprint for the message. In the communication channel, the sender and receiver communicate with each

other so both must receive the right message. to ensure the integrity of the message digest is sent by the sender to the receiver along with the message. The receiver receives the message and the message digest value and the receiver uses the same hash function to generate a new message for the message he/she received. once generated receiver compares both the message digest values to verify that the message is received without any modifications and corrections. If both digest values are the same, it proves that the message has not been modified in the network by any person. Different digest values indicate that the received message is not the actual message send by the sender. This way we could check and verify the integrity of the message using message digest.

Hash Function Criteria

Hash functions are a fundamental concept in computer science and play a crucial role in various applications such as data storage, retrieval, and cryptography. In data structures and algorithms (DSA), hash functions are primarily used in hash tables, which are essential for efficient data management. This article delves into the intricacies of hash functions, their properties, and the different types of hash functions used in DSA.

What is a Hash Function?

A **hash function** is a function that takes an input (or 'message') and returns a fixed-size string of bytes. The output, typically a number, is called the **hash code** or **hash value**. The main purpose of a hash function is to efficiently map data of arbitrary size to fixed-size values, which are often used as indexes in hash tables.

Key Properties of Hash Functions

- **Deterministic**: A hash function must consistently produce the same output for the same input.

- **Fixed Output Size**: The output of a hash function should have a fixed size, regardless of the size of the input.
- **Efficiency**: The hash function should be able to process input quickly.
- **Uniformity**: The hash function should distribute the hash values uniformly across the output space to avoid clustering.
- **Pre-image Resistance**: It should be computationally infeasible to reverse the hash function, i.e., to find the original input given a hash value.
- **Collision Resistance**: It should be difficult to find two different inputs that produce the same hash value.
- **Avalanche Effect**: A small change in the input should produce a significantly different hash value.

Applications of Hash Functions

- **Hash Tables**: The most common use of hash functions in DSA is in hash tables, which provide an efficient way to store and retrieve data.
- **Data Integrity**: Hash functions are used to ensure the integrity of data by generating checksums.
- **Cryptography**: In cryptographic applications, hash functions are used to create secure hash algorithms like SHA-256.
- **Data Structures**: Hash functions are utilized in various data structures such as Bloom filters and hash sets.

Hash Algorithms: SHA-1

SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function designed by the National Security Agency (NSA) and published by the National Institute of Standards and Technology (NIST) as a U.S. Federal Information Processing Standard. Despite its historical importance, SHA-1 is now considered insecure and deprecated for most cryptographic uses due to vulnerabilities that make it susceptible to collision attacks.

Features of SHA-1:

- **Hash Length:** SHA-1 produces a 160-bit (20-byte) hash value, typically rendered as a 40-digit hexadecimal number.
- **Input Size:** Can hash any input size up to 2642^{64}264 bits.
- **Output:** The hash output is a fixed size of 160 bits, regardless of the size of the input data.

SHA-1 Hashing Process:

The SHA-1 algorithm processes the input in blocks of 512 bits. Here is a high-level overview of the steps involved:

1. **Padding the Message:**
    - The input message is padded so that its length is congruent to 448 modulo 512. Padding is done by adding a single '1' bit followed by '0' bits, and then the length of the original message as a 64-bit integer at the end.
2. **Initialize Hash Values:**
    - SHA-1 uses five constant 32-bit words, which form the initial hash value. These are:

plaintext

h0 = 0x67452301

h1 = 0xEFCDAB89

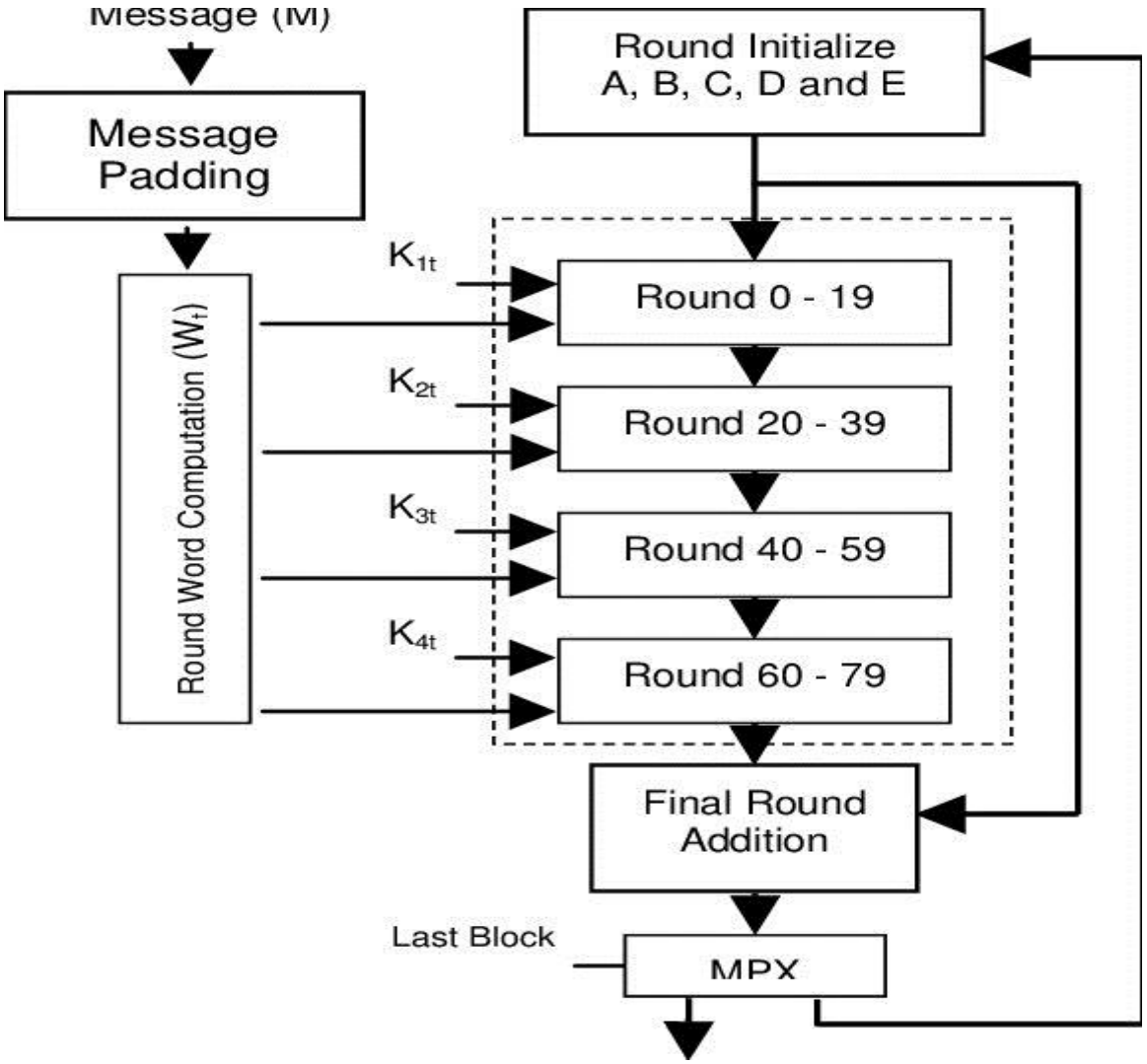
h2 = 0x98BADCFE

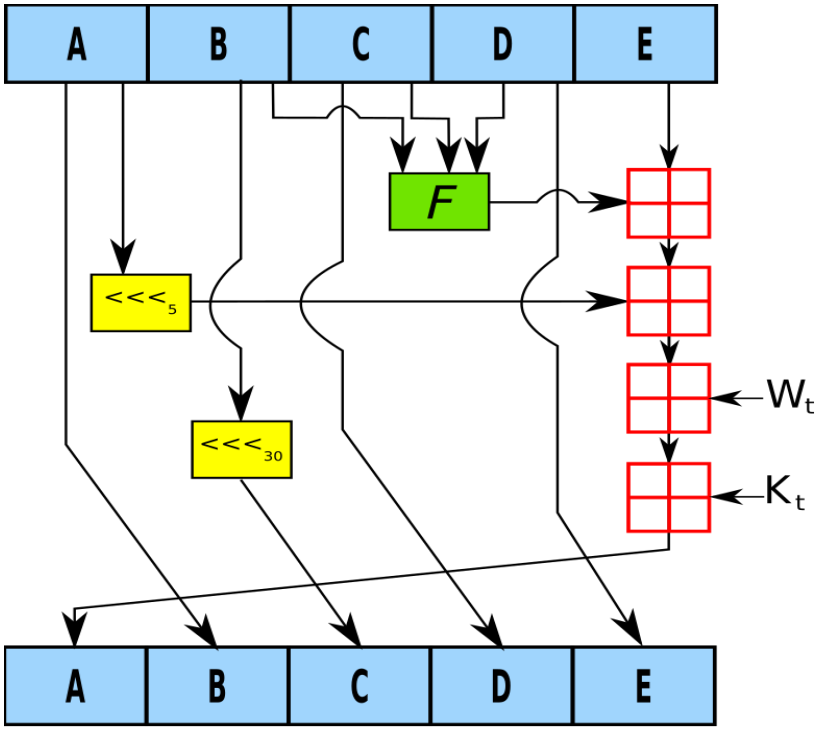
h3 = 0x10325476

h4 = 0xC3D2E1F0

3. **Process Message in 512-bit Blocks:**
    - The message is divided into 512-bit blocks. Each block is processed to update the hash value.
4. **Message Schedule:**
    - Each 512-bit block is expanded into 80 32-bit words using bitwise operations.

5. **Compression Function:**
   - The expanded words are processed in a loop using a series of bitwise operations and modular additions. This step involves logical functions and bitwise operations applied over the 80 rounds.
   - The results are added to the current hash values.
6. **Final Hash Value:**
   - After processing all blocks, the resulting 160-bit hash is a concatenation of the final values of h0,h1,h2,h3 andh4.





We can see that I have mentioned F(t), W(t) & K(t). We already know about W(t), these are the set of 32 bits, part of the 512 bits message but we are still unaware of F(t) and K(t). These are unique set of functions and values that are used to compute the hash and they remain constant for every 20 rounds i.e. they change 4 times while 80 rounds of computations take place. For every 20 rounds, F(t) and K(t) are constant they have a set of predefined values and function description which remains common depicted below.

Rounds 1-20

f(1) = (B **and** C) **or** ((**not** B) **and** D)

k(1) = 0x5A827999

Rounds 21-40

f(2) = B **xor** C **xor** D

k(2) = 0x6ED9EBA1

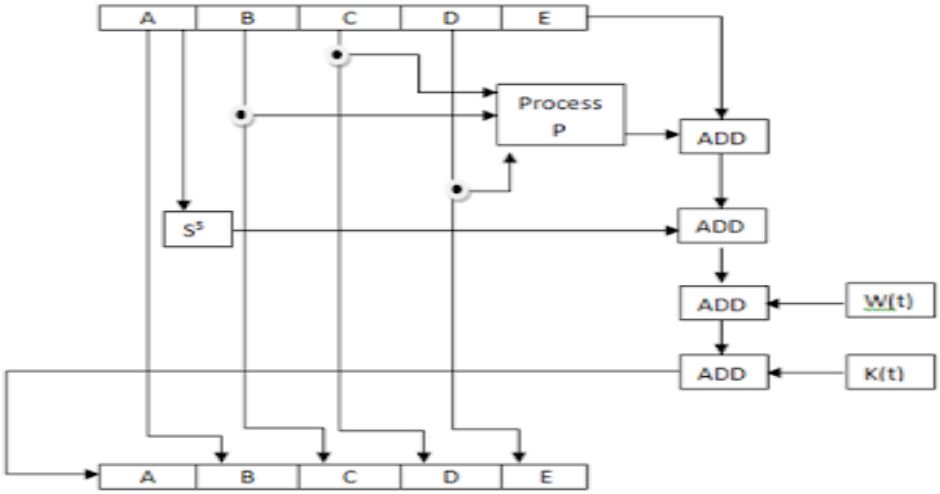
Rounds 41-60

f(3) = (B **and** C) **or** (B **and** D) **or** (C **and**)

k(3) = 0x8F1BBCDCRounds 61-80

f(4) = B **xor** C **xor** D

k(4) = 0xCA62C1D6



W(t) is the expanded message word of round t;
K(t) is the round constant of round t;

Vulnerabilities and Deprecation:

- **Collision Attacks:** SHA-1 is vulnerable to collision attacks, where two different inputs produce the same hash output. This weakness significantly reduces its effectiveness for ensuring data integrity and security.
- **Pre-image Attacks:** Though more challenging, advances in computational power and cryptanalysis have made pre-image attacks more feasible over time.
- **Replacement by SHA-2 and SHA-3:** Due to these vulnerabilities, SHA-1 has been largely replaced by the more secure SHA-2 and SHA-3 families of hash functions. These provide better security and are recommended for use in modern applications.

Pseudocode SHA1

Suppose the message ‘abc’ were to be encoded using SHA-1, with the message ‘abc’ in binary being

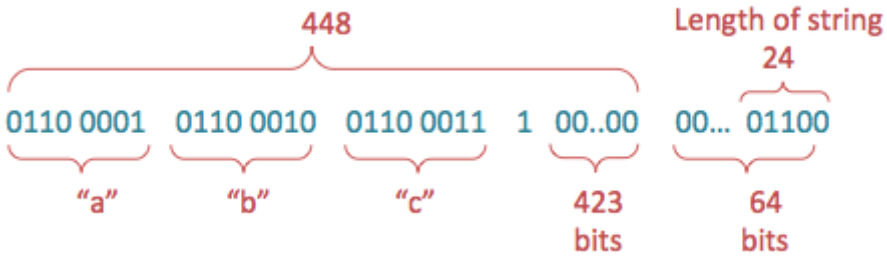
01100001 01100010 0110001101100001 01100010 01100011

and that in hex being 616263.

1) The first step is to initialize five random strings of hex characters that will serve as part of the hash function (shown in hex):

$$H_0 = 67DE2A01$$
$$H_1 = BB03E28C$$
$$H_2 = 011EF1DC$$
$$H_3 = 9293E9E2$$
$$H_4 = CDEF23A9.$$

2) The message is then padded by appending a 1, followed by enough 0s until the message is 448 bits. The length of the message represented by 64 bits is then added to the end, producing a message that is 512 bits long:



Padding of string "abc" in bits, finalized by the length of the string, which is 24 bits.

3) The padded input obtained above, *M*, is then divided into 512-bit chunks, and each chunk is further divided into sixteen 32-bit words, 0… 15*W*0…*W*15 . In the case of 'abc', there's only one chunk, as the message is less than 512-bits total.

4) For each chunk, begin the 80 iterations, *i*, necessary for hashing (80 is the determined number for SHA-1), and execute the following steps on each chunk, :*Mn*:

- For iterations 16 through 79, where 16≤ *I* ≤79, perform the following operation:

$$W(i) = S^1\big(W(i-3) \oplus W(i-8) \oplus W(i-14) \oplus W(i-16)\big),$$

where XOR, or $\oplus$, is represented by the following comparison of inputs *x* and *y*:

x	*y*	Output
0	0	0
1	0	1
0	1	1
1	1	0

- For example, when *i* is 16, the words chosen are *W*(13),*W*(8),*W*(2),*W*(0), and the output is a new word, *W*(16), so performing the XOR, or $\oplus$, operation on those words will give this:

W(0)	01100001 01100010 01100011 1000000001100001 01100010 01100011 100 00000
W(2)	00000000 00000000 00000000 0000000000000000 00000000 00000000 000 00000
W(8)	00000000 00000000 00000000 0000000000000000 00000000 00000000 000 00000
W(13)	00000000 00000000 00000000 0000000000000000 00000000 00000000 000 00000
	$\oplus\oplus$
W(16)	01100001 01100010 01100011 1000000001100001 01100010 01100011 100 00000

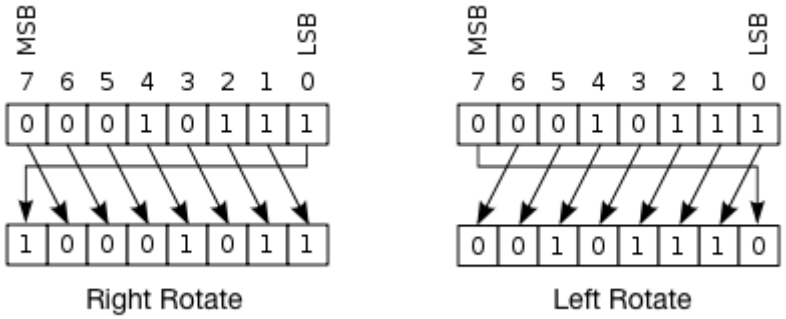
Circular Shift Operation

Now, the circular shift operation $Sn(X)$ on the word X by n bits, n being an integer between 00 and 3232, is defined by

$$S^n(X) = (X << n) \quad \mathbf{OR} \quad (X >> 32 - n),$$

where $X<<n$ is the **left-shift** operation, obtained by discarding the leftmost n bits of X and padding the result with n zeroes on the right.

$X>>32-n$ is the **right-shift** operation obtained by discarding the rightmost n bits of X and padding the result with n zeroes on the left. Thus $Sn(X)$ is equivalent to a circular shift of X by n positions, and in this case the circular left-shift is used.



So, a left shift $\mathsf{Sn}(W(i))$, where $W(i)$ is 10010,10010, would produce 0100101001, as the rightmost bit 00 is shifted to the left side of the string. Therefore, (16)W(16) would end up being

11000010 11000100 11000111 000000000.11000010 11000100 11000111 00 0000000.

5) Now, store the hash values defined in step 1 in the following variables:

$A=H0$

$B=H1$

$C=H2$

$D=H3$

$E=H4$.

6) For 8080 iterations, where $0\leq i\leq 79$, compute .

$$TEMP = S5*(A) + f(i;B,C,D) + E + W(i) + K(i).$$

See below for details on the logical function, f, and on the values of K(i). Reassign the following variables:

$$E = D$$

$$D = C$$

$$C = S30(B)$$

$$A = TEMP.$$

7) Store the result of the chunk's hash to the overall hash value of all chunks, as shown below, and proceed to execute the next chunk:

$$H0 = H0 + A$$

$$H1 = H1 + B$$

$$H1 = H2 + C$$

$$H1 = H3 + D$$

$$H1 = H4 + E.$$

8) As a final step, when all the chunks have been processed, the message digest is represented as the 160-bit string comprised of the OR logical operator, ∨, of the 5 hashed values:

$$HH = S^{128}(H_0) \vee S^{96}(H_1) \vee S^{64}(H_2) \vee S^{32}(H_3) \vee H_4.$$

So, the string 'abc' becomes represented by a hash value akin to a9993e364706816aba3e25717850c26c9cd0d89d.

If the string changed to 'abcd', for instance, the hashed value would be drastically different so attackers cannot tell that it is similar to the original message.

The hash value for 'abcd' is 81fe8bfe87576c3ecb22426f8e57847382917acf.

Functions used in the algorithm

A sequence of logical functions are used in SHA-1, depending on the value of i, where $0 \leq i \leq 79$, and on three 32-bit words B, C, and D, in order to produce a 32-bit output. The following equations describe the logical functions, where $\neg$ is the logical *NOT*, $\vee$ is the logical *OR*, $\wedge$ is the logical *AND*, and $\oplus$ is the logical XOR:

$$
\begin{aligned}
f(i;B,C,D) &= (B \wedge C) \vee ((\neg B) \wedge D) && \text{for } 0 \geq i \geq 19 \\
f(i;B,C,D) &= B \oplus C \oplus D && \text{for } 20 \geq i \geq 39 \\
f(i;B,C,D) &= (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) && \text{for } 40 \geq i \geq 59 \\
f(i;B,C,D) &= B \oplus C \oplus D && \text{for } 60 \geq i \geq 79.
\end{aligned}
$$

Additionally, a sequence of constant words, shown in hex below, is used in the formulas:

$$
\begin{aligned}
K(i) &= 5A827999, && \text{where } 0 \leq i \leq 19 \\
K(i) &= 6ED9EBA1, && \text{where } 20 \leq i \leq 39 \\
K(i) &= 8F1BBCDC, && \text{where } 40 \leq i \leq 59 \\
K(i) &= CA62C1D6, && \text{where } 60 \leq i \leq 79.
\end{aligned}
$$

Biometric parameters

Ensuring the integrity of biometric parameters is crucial for maintaining the security and reliability of biometric systems. Biometric systems use unique physical or behavioral characteristics (such as fingerprints, facial recognition,

iris scans, or voice recognition) for identification or authentication purposes. If the integrity of these biometric parameters is compromised, the system can be vulnerable to various types of attacks, including spoofing, identity theft, and unauthorized access.

Key Aspects of Ensuring Biometric Parameters Integrity

1. **Data Protection:**
    - **Encryption:** Biometric data should be encrypted both in transit and at rest to prevent unauthorized access. Advanced encryption standards like AES are commonly used.
    - **Hashing:** Storing only hashed versions of biometric templates can add an extra layer of security. Techniques such as SHA-256 or SHA-3 can be used for hashing.
2. **Anti-Spoofing Measures:**
    - **Liveness Detection:** Implementing measures to detect signs of life (e.g., blinking, pulse detection) ensures that the biometric data is being collected from a live person rather than a replica or photo.
    - **Multimodal Biometrics:** Using multiple biometric modalities (e.g., combining fingerprint and facial recognition) makes it more difficult for attackers to spoof the system.
3. **Data Integrity Checks:**
    - **Digital Signatures:** Applying digital signatures to biometric data ensures that any alterations can be detected. Digital signatures provide a way to verify the authenticity and integrity of the data.
    - **Checksum and Hash Verification:** Regularly computing and verifying checksums or hash values of biometric data can help detect any unauthorized modifications.
4. **Secure Data Storage:**

- **Isolated Storage:** Biometric data should be stored in a secure, isolated environment that is protected from unauthorized access. Hardware security modules (HSMs) or secure enclaves can be used for this purpose.
- **Access Control:** Implement strict access control policies to ensure that only authorized personnel and systems can access biometric data.

5. **Regular Audits and Monitoring:**
   - **Audit Trails:** Maintain detailed logs of all access and modifications to biometric data. Regularly review these logs for any suspicious activities.
   - **Real-time Monitoring:** Implement real-time monitoring to detect and respond to potential security incidents involving biometric data.
6. **User Consent and Awareness:**
   - **Informed Consent:** Ensure that users are aware of how their biometric data will be used, stored, and protected. Obtain explicit consent before collecting biometric data.
   - **Transparency:** Be transparent about the measures in place to protect biometric data and the policies governing its use.

Message Authentication

Message Authentication Code (MAC)

Digital Signature

End-Point Authentication

Passwords &Challenge-Response

Message Authentication Code (MAC)

In cryptography, a **message authentication code** (**MAC**), sometimes known as an **authentication tag**, is a short piece of information used for authenticating and integrity-checking a message. In other words, to confirm that the message came from the stated sender (its authenticity) and has not been changed (its integrity). The MAC value allows verifiers (who also possess a secret key) to detect any changes to the message content.

Digital Signature

Informally, a message authentication code system consists of three algorithms:

- A key generation algorithm selects a key from the key space uniformly at random.
- A signing algorithm efficiently returns a tag given the key and the message.
- A verifying algorithm efficiently verifies the authenticity of the message given the same key and the tag. That is, return *accepted* when the message and tag are not tampered with or forged, and otherwise return *rejected*.

A secure message authentication code must resist attempts by an adversary to forge tags, for arbitrary, select, or all messages, including under conditions of known- or chosen-message. It should be computationally infeasible to compute a valid tag of the given message without knowledge of the key, even

if for the worst case, we assume the adversary knows the tag of any message but the one in question.

Formally, a **message authentication code** (**MAC**) system is a triple of efficient[4] algorithms (G, S, V) satisfying:

- G (key-generator) gives the key k on input 1^n, where n is the security parameter.
- S (signing) outputs a tag t on the key k and the input string x.
- V (verifying) outputs *accepted* or *rejected* on inputs: the key k, the string x and the tag t.

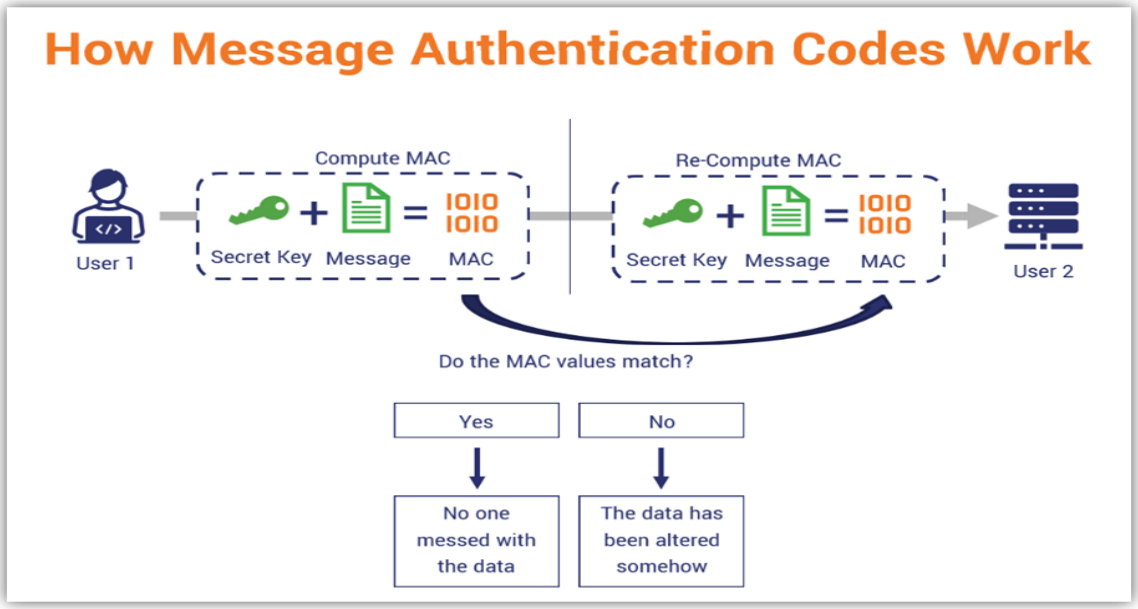
S and V must satisfy the following:

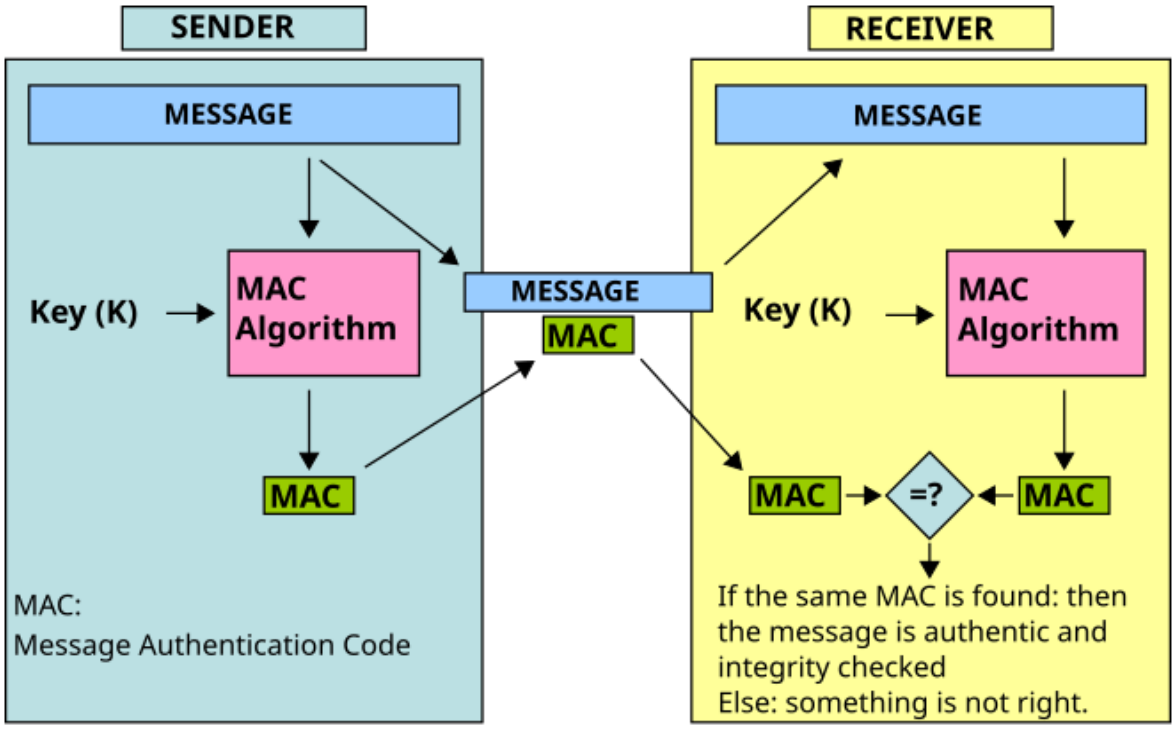
$\Pr [k \leftarrow G(1^n), V(k, x, S(k, x)) = accepted] = 1.$

A MAC is **unforgeable** if for every efficient adversary A

$\Pr [k \leftarrow G(1^n), (x, t) \leftarrow A^{S(k, \cdot)}(1^n), x \notin \text{Query}(A^{S(k, \cdot)}, 1^n), V(k, x, t) = accepted] < \text{negl}(n)$,

where $A^{S(k, \cdot)}$ denotes that A has access to the oracle $S(k, \cdot)$, and $\text{Query}(A^{S(k, \cdot)}, 1^n)$ denotes the set of the queries on S made by A, which knows n. Clearly we require that any adversary cannot directly query the string x on S, since otherwise a valid tag can be easily obtained by that adversary.





Digital Signature

Digital Signature

A digital signature is a mathematical technique used to validate the authenticity and integrity of a message, software, or digital document.

1. **Key Generation Algorithms**: Digital signature is electronic signatures, which assure that the message was sent by a particular sender. While performing digital transactions authenticity and integrity should be assured, otherwise, the data can be altered or someone can also act as if he was the sender and expect a reply.
2. **Signing Algorithms**: To create a digital signature, signing algorithms like email programs create a one-way hash of the electronic data which is to be signed. The signing algorithm then encrypts the hash value using the private key (signature key). This encrypted hash along with other information like the hashing algorithm is the digital signature. This digital signature is appended with the data and sent to the verifier. The reason for encrypting the hash instead of the entire message or document is that a hash function converts any arbitrary input into a

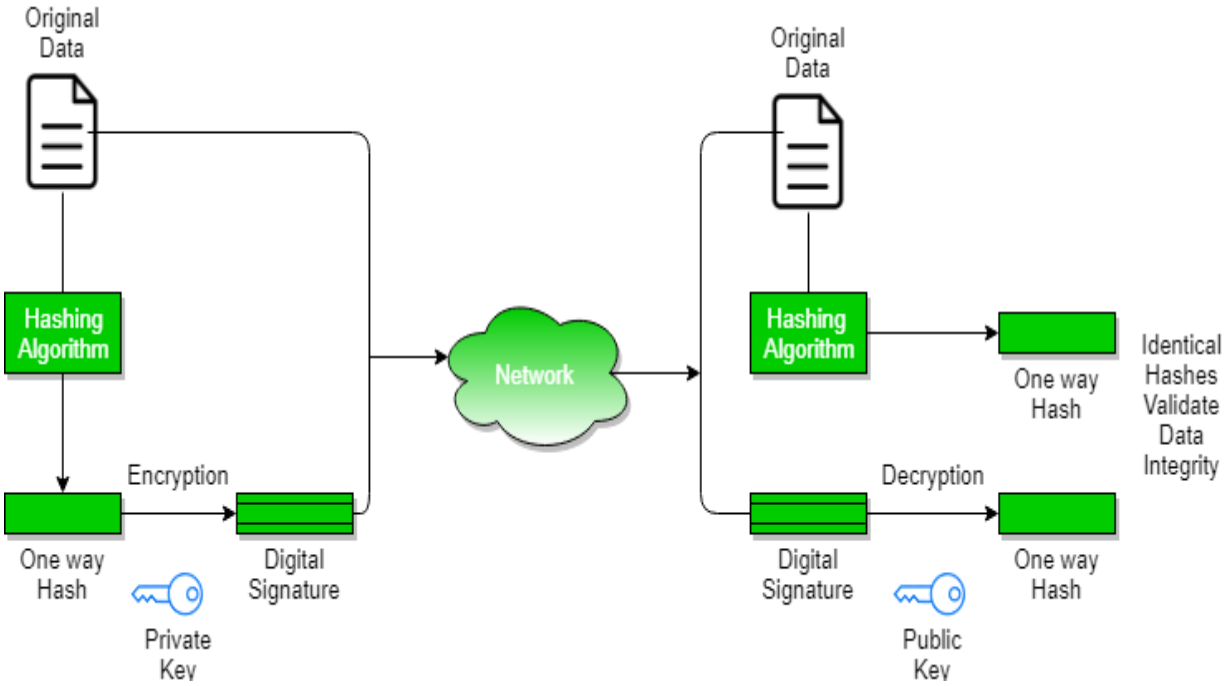
much shorter fixed-length value. This saves time as now instead of signing a long message a shorter hash value has to be signed and moreover hashing is much faster than signing.

3. **Signature Verification Algorithms** : Verifier receives Digital Signature along with the data. It then uses Verification algorithm to process on the digital signature and the public key (verification key) and generates some value. It also applies the same hash function on the received data and generates a hash value. If they both are equal, then the digital signature is valid else it is invalid.

The steps followed in creating digital signature are :

1. Message digest is computed by applying hash function on the message and then message digest is encrypted using private key of sender to form the digital signature. (digital signature = encryption (private key of sender, message digest) and message digest = message digest algorithm(message)).
2. Digital signature is then transmitted with the message.(message + digital signature is transmitted)
3. Receiver decrypts the digital signature using the public key of sender.(This assures authenticity, as only sender has his private key so only sender can encrypt using his private key which can thus be decrypted by sender's public key).
4. The receiver now has the message digest.
5. The receiver can compute the message digest from the message (actual message is sent with the digital signature).
6. The message digest computed by receiver and the message digest (got by decryption on digital signature) need to be same for ensuring integrity.

Message digest is computed using one-way hash function, i.e. a hash function in which computation of hash value of a message is easy but computation of the message from hash value of the message is very difficult.



Assurances about digital signatures

The definitions and words that follow illustrate the kind of assurances that digital signatures offer.

1. **Authenticity**: The identity of the signer is verified.
2. **Integration:** Since the content was digitally signed, it hasn't been altered or interfered with.
3. **Non-repudiation:** demonstrates the source of the signed content to all parties. The act of a signer denying any affiliation with the signed material is known as repudiation.
4. **Notarization:** Under some conditions, a signature in a Microsoft Word, Microsoft Excel, or Microsoft PowerPoint document that has been time-stamped by a secure time-stamp server is equivalent to a notarization.

Benefits of Digital Signatures

- **Legal documents and contracts:** Digital signatures are legally binding. This makes them ideal for any legal document that requires a signature authenticated by one or more parties and guarantees that the record has not been altered.
- **Sales contracts:** Digital signing of contracts and sales contracts authenticates the identity of the seller and the buyer, and both parties can be sure that the signatures are legally binding and that the terms of the agreement have not been changed.
- **Financial Documents:** Finance departments digitally sign invoices so customers can trust that the payment request is from the right seller, not from a bad actor trying to trick the buyer into sending payments to a fraudulent account.
- **Health Data:** In the healthcare industry, privacy is paramount for both patient records and research data. Digital signatures ensure that this confidential information was not modified when it was transmitted between the consenting parties.

Drawbacks of Digital Signature

- **Dependency on technology:** Because digital signatures rely on technology, they are susceptible to crimes, including hacking. As a result, businesses that use digital signatures must make sure their systems are safe and have the most recent security patches and upgrades installed.
- **Complexity:** Setting up and using digital signatures can be challenging, especially for those who are unfamiliar with the technology. This may result in blunders and errors that reduce the system's efficacy. The process of issuing digital signatures to senior citizens can occasionally be challenging.

- **Limited acceptance:** Digital signatures take time to replace manual ones since technology is not widely available in India, a developing nation.

Digital Certificate

Digital certificate is issued by a trusted third party which proves sender's identity to the receiver and receiver's identity to the sender.
A digital certificate is a certificate issued by a Certificate Authority (CA) to verify the identity of the certificate holder. Digital certificate is used to attach public key with a particular individual or an entity.

Digital certificate contains

- Name of certificate holder.
- Serial number which is used to uniquely identify a certificate, the individual or the entity identified by the certificate
- Expiration dates.
- Copy of certificate holder's public key.(used for decrypting messages and digital signatures)
- Digital Signature of the certificate issuing authority.

Digital certificate is also sent with the digital signature and the message.

Advantages of Digital Certificate

- **NETWORK SECURITY :** A complete, layered strategy is required by modern cybersecurity methods, wherein many solutions cooperate to offer the highest level of protection against malevolent actors. An essential component of this puzzle is digital certificates, which offer strong defence against manipulation and man-in-the-middle assaults.
- **VERIFICATION :** Digital certificates facilitate cybersecurity by restricting access to sensitive data, which makes authentication a crucial component of cybersecurity. Thus, there is a decreased chance

that hostile actors will cause chaos. At many different endpoints, certificate-based authentication provides a dependable method of identity verification. Compared to other popular authentication methods like biometrics or one-time passwords, certificates are more flexible.

- **BUYER SUCCESS :** Astute consumers demand complete assurance that the websites they visit are reliable. Because digital certificates are supported by certificate authority that users' browsers trust, they offer a readily identifiable indicator of reliability.

Disadvantages of Digital Certificate

- **Phishing attacks:** To make their websites look authentic, attackers can fabricate bogus websites and obtain certificates. Users may be fooled into providing sensitive information, such as their login credentials, which the attacker may then take advantage of.
- **Weak encryption:** Older digital certificate systems may employ less secure encryption methods that are open to intrusions.
- **Misconfiguration:** In order for digital certificates to work, they need to be set up correctly. Websites and online interactions can be attacked due to incorrectly configured certificates.

Digital certificate vs digital signature

Digital signature is used to verify authenticity, integrity, non-repudiation ,i.e. it is assuring that the message is sent by the known user and not modified, while digital certificate is used to verify the identity of the user, maybe sender or receiver. Thus, digital signature and certificate are different kind of things but both are used for security. Most websites use digital certificate to enhance trust of their users

End-Point Authentication

End-point authentication is a crucial aspect of network security that involves verifying the identity of devices or endpoints (such as computers, mobile

devices, IoT devices, etc.) before allowing them to access network resources. It helps ensure that only authorized devices can connect to the network, protecting it from unauthorized access, malware, and other security threats.

Key Concepts and Techniques for End-Point Authentication

1. **Device Identification:**
    - **MAC Address Filtering:** Restrict network access based on the device's MAC address. However, this method can be vulnerable to MAC address spoofing.
    - **Certificates:** Use digital certificates issued by a trusted Certificate Authority (CA) to authenticate devices. Each device presents its certificate to prove its identity.
2. **Authentication Protocols:**
    - **IEEE 802.1X:** A network access control protocol that provides an authentication mechanism for devices trying to connect to a LAN or WLAN. It uses the Extensible Authentication Protocol (EAP) for communication between the device, authenticator, and authentication server.
    - **RADIUS (Remote Authentication Dial-In User Service):** A networking protocol that provides centralized Authentication, Authorization, and Accounting (AAA) management for devices connecting to the network.
    - **TACACS+ (Terminal Access Controller Access-Control System Plus):** Similar to RADIUS but separates authentication, authorization, and accounting functions, providing more flexibility and security.
3. **Multi-Factor Authentication (MFA):**
    - Combining two or more authentication factors (e.g., something the user knows, something the user has, and something the user is) to verify the device's identity. For example, using a password

(something the user knows) and a hardware token (something the user has).

4. **Public Key Infrastructure (PKI):**
    - **Certificates:** Use X.509 certificates for device authentication. Devices must present a valid certificate issued by a trusted CA.
    - **Key Pairs:** Use public and private key pairs for secure communication and authentication. The private key remains on the device, while the public key is shared with the server.
5. **Endpoint Security Solutions:**
    - **Endpoint Detection and Response (EDR):** Monitors and detects suspicious activities on endpoints. It can be used to enforce authentication policies and ensure that only compliant devices access the network.
    - **Mobile Device Management (MDM):** Enforces security policies on mobile devices, including authentication requirements.

Steps for Implementing End-Point Authentication

1. **Define Authentication Policies:**
    - Determine the authentication requirements for different types of devices and users. Define policies that specify which devices can connect to the network and under what conditions.
2. **Deploy Authentication Infrastructure:**
    - Set up authentication servers, such as RADIUS or TACACS+, to handle authentication requests. Configure network devices (e.g., switches, routers, wireless access points) to use these servers for authentication.
3. **Issue and Manage Certificates:**

- Use a PKI to issue digital certificates to devices. Ensure that certificates are securely distributed and managed, with appropriate expiration and revocation policies.

4. **Configure Network Devices:**
   - Enable IEEE 802.1X authentication on network devices to enforce authentication policies. Ensure that devices are configured to communicate with the authentication server.
5. **Implement Multi-Factor Authentication:**
   - Deploy MFA solutions to enhance security. Ensure that devices and users are required to provide multiple authentication factors before gaining access to the network.
6. **Monitor and Audit:**
   - Continuously monitor authentication activities and audit logs to detect and respond to suspicious activities. Use EDR and other endpoint security solutions to enhance visibility and control over endpoints.

Example of End-Point Authentication using IEEE 802.1X and RADIUS

1. **Set up RADIUS Server:**
   - Configure a RADIUS server to handle authentication requests. For example, you can use FreeRADIUS on a Linux server.
2. **Configure Network Switch:**
   - Enable IEEE 802.1X authentication on the switch ports.
3. **Deploy Certificates:**
   - Issue and install certificates on the devices that need to authenticate to the network.
4. **Enable 802.1X on Devices:**
   - Configure the devices (e.g., laptops) to use 802.1X for network authentication. This can typically be done through the network

settings on the device, specifying the use of a certificate for authentication.

By following these steps, you can implement robust end-point authentication to ensure that only authorized devices can access your network, enhancing the overall security and integrity of your network infrastructure.

Passwords &Challenge-Response

Passwords and challenge-response mechanisms are fundamental techniques used in authentication systems to verify the identity of users and devices. They offer different levels of security and are often used together to enhance protection.

Passwords

Passwords are secret strings of characters that users provide to authenticate themselves to a system. They are the most common form of authentication. Here are some key points and best practices related to password-based authentication:

Key Points:

1. **Simplicity and Familiarity:** Passwords are easy to implement and familiar to most users.
2. **Weaknesses:** Passwords can be guessed, stolen, or cracked, especially if they are weak or reused across multiple accounts.
3. **Password Storage:** Passwords should be stored securely using hashing algorithms such as bcrypt, Argon2, or PBKDF2, with proper salting to protect against rainbow table attacks.

Best Practices:

1. **Complexity:** Require passwords to be complex (e.g., a mix of letters, numbers, and special characters).

2. **Length:** Enforce minimum password length (e.g., at least 12 characters).
3. **Expiration:** Implement password expiration policies to compel users to change passwords periodically.
4. **Multi-Factor Authentication (MFA):** Combine passwords with another form of authentication to increase security.
5. **Avoid Common Passwords:** Use mechanisms to prevent users from choosing easily guessable passwords.

Challenge-Response Mechanisms

Challenge-response authentication enhances security by using dynamic data rather than static passwords. It involves the server sending a challenge to the user or device, which must then respond correctly to be authenticated.

Key Points:

1. **Dynamic Authentication:** Uses one-time or dynamic data, making it resistant to replay attacks.
2. **Cryptographic Methods:** Often relies on cryptographic techniques to generate challenges and validate responses.

Types of Challenge-Response Mechanisms:

1. **One-Time Passwords (OTPs):** A server sends a challenge (e.g., a random number), and the client generates a response based on a shared secret and the challenge.
2. **Public Key Cryptography:** The server sends a challenge, and the client signs it with their private key. The server verifies the signature using the client's public key.
3. **Zero-Knowledge Proofs:** The user proves knowledge of a secret without revealing the secret itself.

Example: Challenge-Response with OTP

1. **Setup:**
    - Both server and client share a secret key.

- The server generates a random challenge (e.g., a nonce).

2. **Challenge Generation:**
   - The server sends the challenge to the client.
3. **Response Generation:**
   - The client uses the shared secret key and the challenge to generate a response using a secure algorithm (e.g., HMAC with SHA-256).
4. **Verification:**
   - The server verifies the response by generating its own expected response using the shared secret and the challenge. If the client's response matches the expected response, the client is authenticated.

Combining Passwords and Challenge-Response

To enhance security, passwords and challenge-response mechanisms are often used together in multi-factor authentication (MFA). This approach mitigates the weaknesses of passwords alone by adding an additional layer of security:

1. **User enters a password:** The system verifies the password.
2. **Challenge-response process:** If the password is correct, the system initiates a challenge-response protocol for additional verification.

By combining these techniques, organizations can significantly enhance the security of their authentication systems, making it much harder for attackers to gain unauthorized access.