Chapter 5: Web Security

Authentication

- *Authentication refers to the process of verifying the identity of a user, system, or entity attempting to access a computer system, network, or digital resource.*
- It ensures that the entity claiming a particular identity is, indeed, who or what it purports to be.
- Authentication is a fundamental aspect of cybersecurity and is essential for protecting sensitive data, preventing unauthorized access, and ensuring the integrity and confidentiality of digital resources.
- Authentication mechanisms typically involve the presentation of credentials by the entity seeking access, followed by validation of these credentials against predefined criteria or stored information. Credentials may include passwords, PINs, biometric data (such as fingerprints or iris scans), security tokens, digital certificates, or other forms of unique identification.

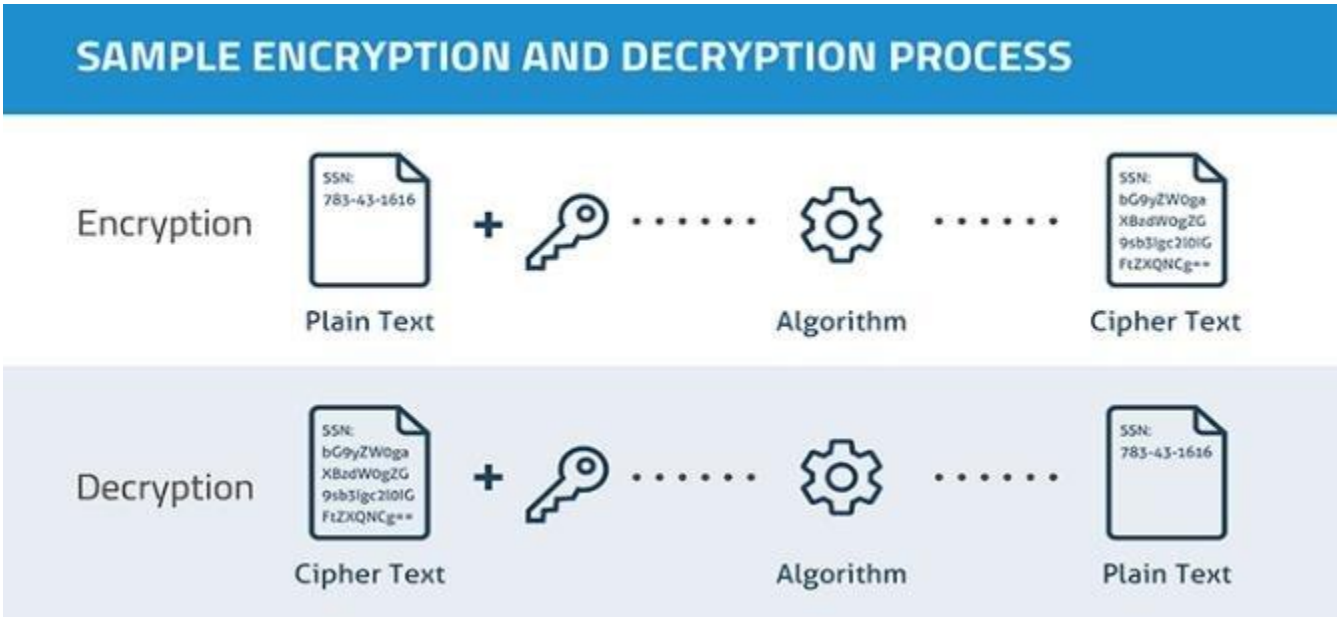
Encryption/Decryption

Encryption and decryption are processes used to secure sensitive information by converting it into an unreadable format (encryption) and then converting it back to its original format (decryption) using a secret key or algorithm.

Terminologies:

- **Encryption** *is the process of converting plaintext (readable data) into ciphertext (unreadable data) using an encryption algorithm and an encryption key.*
- **Decryption** *is the reverse process of encryption, involving the conversion of ciphertext back into plaintext using a decryption algorithm and a decryption key.*
- **Plain Text:** it is the text which is readable and can be understood by all users.
- **Cipher Text:** the message obtained after applying cryptography on plain text is Cipher text.

- **Encryption Algorithm:**An encryption algorithm is a mathematical procedure or set of rules used to transform plaintext data into ciphertext, or vice versa.
- **Encryption Key**:An encryption key is a piece of information, typically a string of characters or binary data, used by an encryption algorithm to transform plaintext into ciphertext, or vice versa
- **Decryption Algorithm:**A decryption algorithm is a set of mathematical procedures and rules used to reverse the encryption process and convert ciphertext back into plaintext.
- **Decryption Key:**A decryption key is a corresponding value or parameter used by a decryption algorithm to reverse the encryption process and convert ciphertext back into plaintext.



Cryptography

Definition

Cryptography is technique of securing information and communications through use of codes so that only those person for whom the information is intended can understand it and process it.Thus preventing unauthorized access to information.

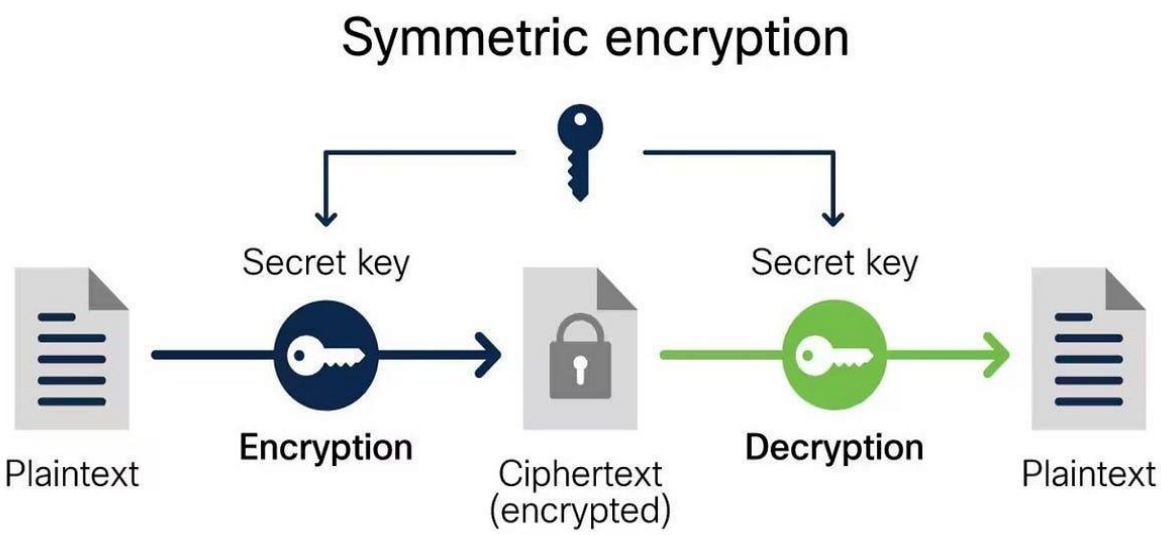
Features Of Cryptography are as follows:

1. **Confidentiality:** Information can only be accessed by the person for whom it is intended and no other person except him can access it.
2. **Integrity:** Information cannot be modified in storage or transition between sender and intended receiver without any addition to information being detected.
3. **Non-repudiation:** The creator/sender of information cannot deny his intention to send information at later stage.
4. **Authentication:** The identities of sender and receiver are confirmed. As well as destination/origin of information is confirmed.

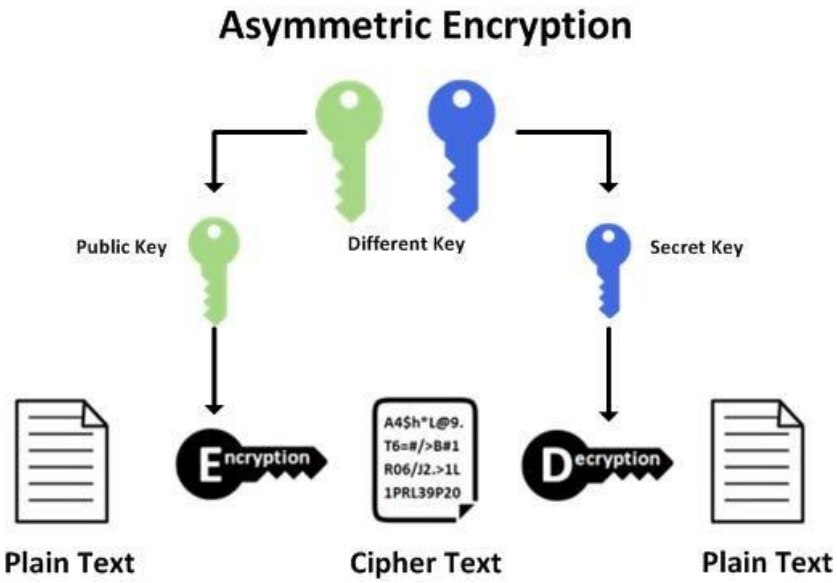
Types Of Cryptography:

1. **Symmetric Key Cryptography:** It is an encryption system where the sender and receiver of message use a single common key to encrypt and decrypt messages. Symmetric Key Systems are faster and simpler but the problem is that sender and receiver have to somehow exchange key in a secure manner. The most popular symmetric key cryptography system are Data Encryption

System(DES) and Advanced Encryption System(AES).



2. **Asymmetric Key Cryptography:** Under this system a pair of keys is used to encrypt and decrypt information. A receiver's public key is used for encryption and a receiver's private key is used for decryption. Public key and Private Key are different. Even if the public key is known by everyone, the intended receiver can only decode it because he alone knows his private key. The most popular asymmetric key cryptography algorithm is RSA algorithm.



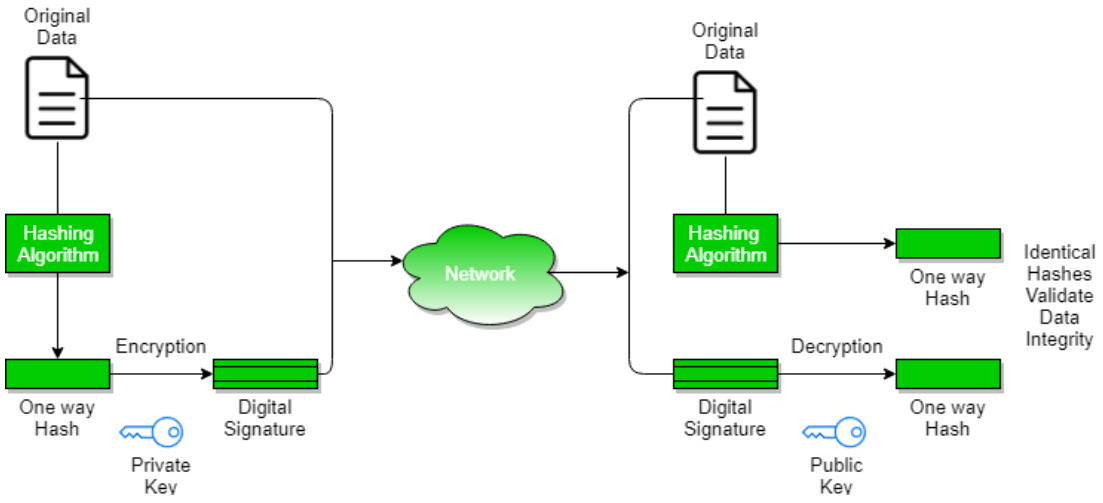
3. **Hash Functions:** There is no usage of any key in this algorithm. A hash value with fixed length is calculated as per the plain text which makes it impossible for contents of plain text to be recovered. Many operating systems use hash functions to encrypt passwords.

Digital Signature:

- A cryptographic technique used to verify the authenticity and integrity of digital messages or documents.
- Created by hashing the message/document and encrypting the hash with the sender's private key.
- Recipients can verify the signature using the sender's public key, ensuring that the message hasn't been altered and was indeed sent by the claimed sender.
- Used for document signing, email authentication, and ensuring data integrity in electronic transactions.

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.
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
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.



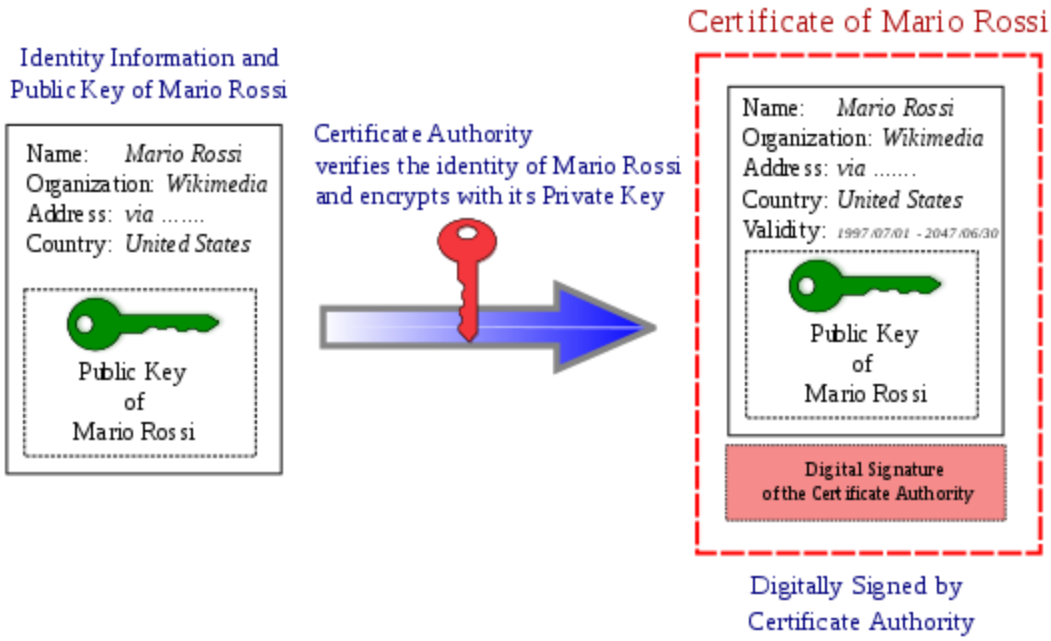
Digital Certificate

- Digital certificate is issued by a trusted third party which proves sender's identity to the receiver and receiver's identity to the sender.
    - Nepal Certifying Company (NCC) has been established in 2013 with the initiation of implementation and promotion of Digital Signature Certificates and provide services in the field of digital security.
- ***A digital certificate, also known as an SSL certificate or public key certificate, is a cryptographic document used to establish the identity of an entity (such as a website, organization, or individual) and to enable secure communication over the internet.***

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.



To view the digital certificate of a website like Yahoo.com, you can follow these steps using Google Chrome as an example:

- Open Google Chrome browser on your computer.
- Navigate to the Yahoo website by typing "yahoo.com" in the address bar and pressing Enter.
- Once the Yahoo website loads, click on the padlock icon located to the left of the website URL in the address bar. This indicates that the website is using HTTPS and has a digital certificate.
- In the dropdown menu that appears, select "Certificate (Valid)" or similar option. This will open a panel displaying detailed information about the digital certificate for Yahoo.com.
- You can now view information such as the certificate issuer (certificate authority), validity period, encryption strength, and other details about the certificate.

- You can also click on the "Details" tab to view additional information about the certificate, including the certificate chain, subject, issuer, and other fields.

SSL(Secure Socket Layer)

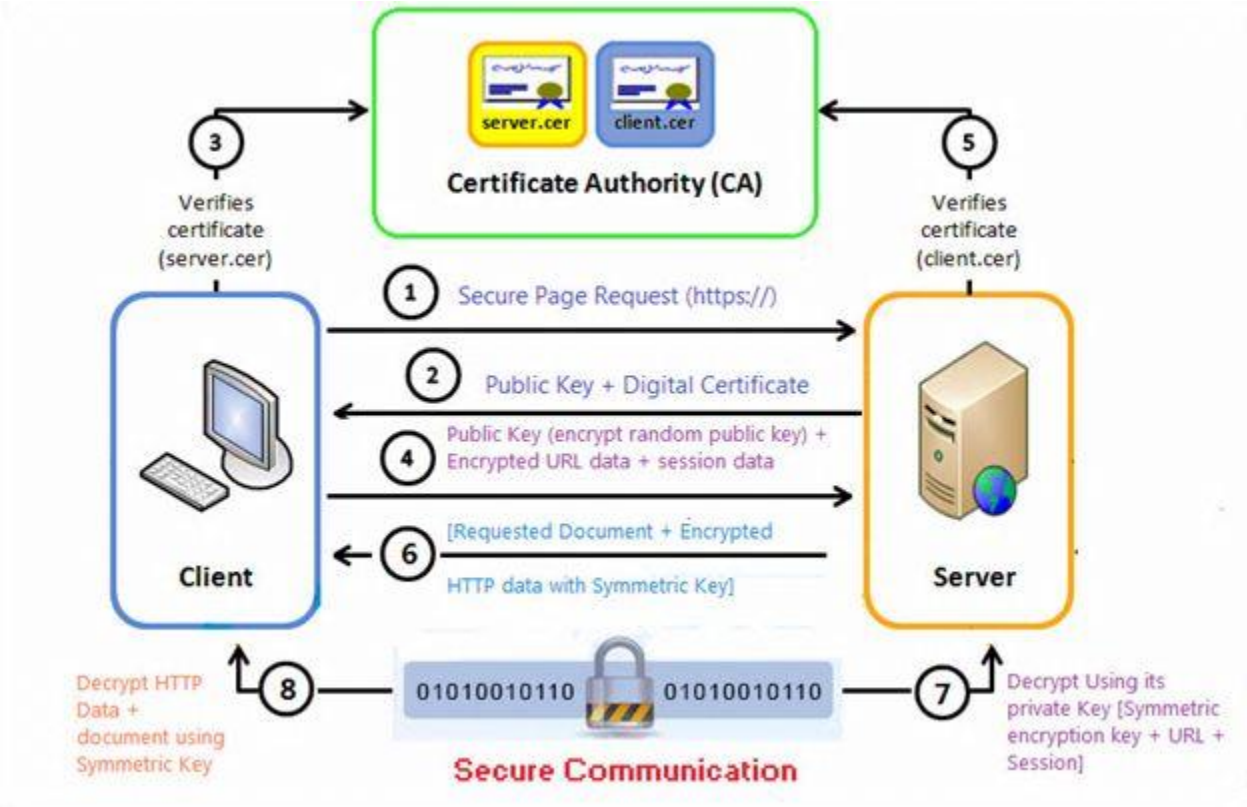
SSL stands for "Secure Sockets Layer." It is a cryptographic protocol designed to provide secure communication over a computer network, typically the internet. SSL ensures that data transmitted between a client and a server remains encrypted and confidential.

SSL (Secure Sockets Layer) works by establishing a secure encrypted connection between a client (such as a web browser) and a server (such as a website). This secure connection ensures that data transmitted between the client and server remains confidential, secure, and tamper-proof. Here's a simplified overview of how SSL works:

- **Client Hello:** The SSL handshake begins when a client (e.g., web browser) sends a "Client Hello" message to the server. This message contains information about the client's SSL/TLS capabilities and preferences, such as supported encryption algorithms and protocols.
- **Server Hello:** Upon receiving the "Client Hello" message, the server responds with a "Server Hello" message. This message contains information about the server's SSL/TLS capabilities and preferences, including the selected encryption algorithm and protocol.
- **Certificate Exchange:** After the initial handshake, the server sends its digital certificate to the client. The certificate contains the server's public key, identity information (e.g., domain name), and other details signed by a trusted Certificate Authority (CA). The client verifies the certificate's authenticity using its built-in list of trusted CAs.
- **Key Exchange:** Once the client has verified the server's certificate, it generates a random symmetric encryption key (session key) and encrypts it using the server's public key obtained from the certificate. This encrypted key is then sent to the server.
- **Symmetric Encryption:** Both the client and server now have the same symmetric encryption key, which they use to encrypt and decrypt data transmitted during the

SSL session. This symmetric encryption ensures secure and efficient communication between the client and server.

- **Secure Data Transfer:** With the symmetric encryption key established, the client and server can securely exchange data over the encrypted SSL/TLS connection. All data transmitted between the client and server, including HTTP requests and responses, is encrypted and protected from eavesdropping and tampering by malicious attackers.
- **Session Resumption:** SSL/TLS supports session resumption mechanisms, allowing clients and servers to reuse previously established SSL sessions to minimize handshake overhead and improve performance.
- **Session Termination**: Once the SSL session is complete or terminated, both the client and server securely dispose of the session keys and cryptographic parameters used during the session. This ensures that the security of future sessions is not compromised.

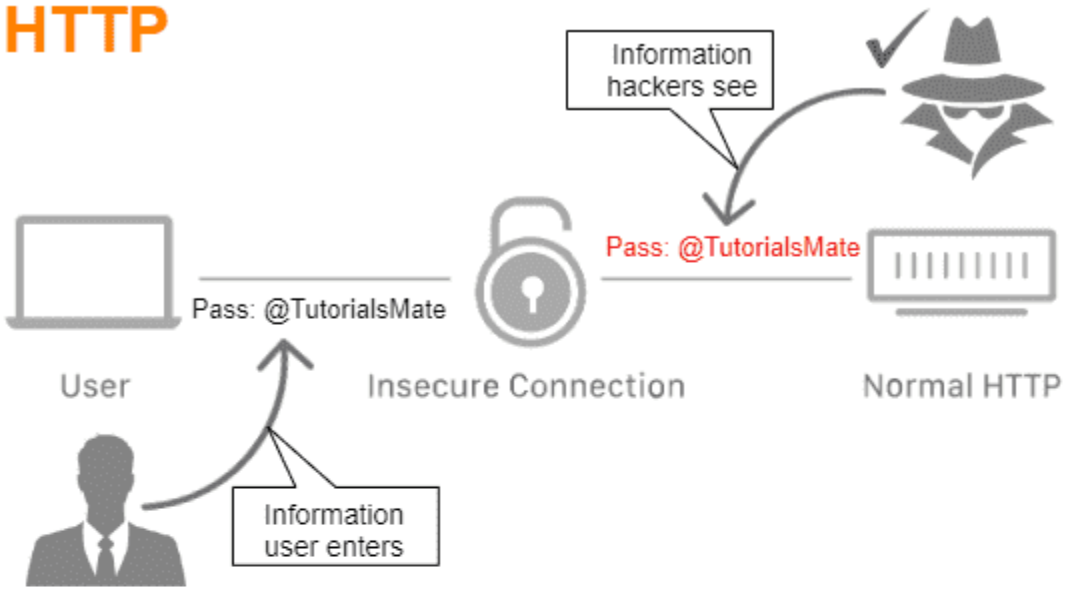


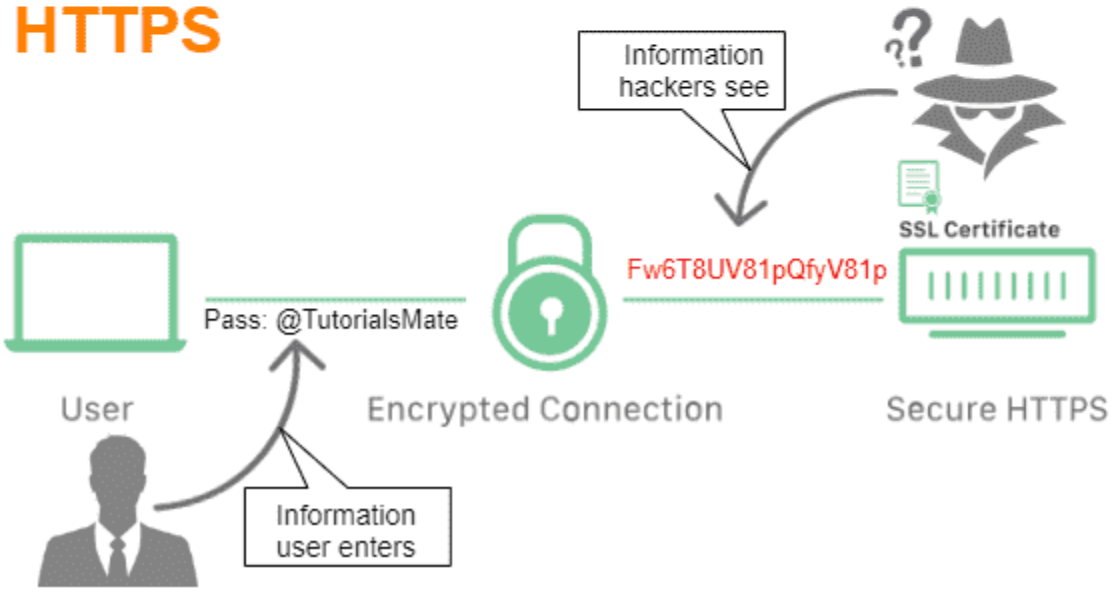
HTTPS

- HTTPS (Hypertext Transfer Protocol Secure) is built upon the foundation of SSL (Secure Sockets Layer) protocol. Initially, SSL was the primary technology used to secure internet communication.
- HTTPS is essentially HTTP over SSL/TLS (Transport Layer Security). It adds an additional layer of security to the standard HTTP protocol by encrypting the data transmitted between a web server and a web browser.
- When a website uses HTTPS, it means that it is utilizing SSL/TLS protocols to encrypt and secure communication. This ensures that sensitive information, such as login credentials, personal details, and financial transactions, remains confidential and secure during transmission.

Feature	**HTTP**	**HTTPS**
Protocol	Hypertext Transfer Protocol	Hypertext Transfer Protocol Secure
Default Port	80	443
Encryption	Not encrypted	Encrypted using SSL/TLS protocols
Data Security	Vulnerable to interception and tampering	Protected from interception and tampering
Security Indicator	Not indicated in URL or browser	Displayed as "https://" in URL
Authentication	No authentication mechanism	Authenticates the server's identity
Trust	No trust assurance	Trusted certificates issued by CAs
Usage	**Standard for general web traffic**	**Used for secure transactions and data**


http://www.
VS.
https://www.
SECURE


HTTP
Information hackers see
Pass: @TutorialsMate
Pass: @TutorialsMate
User
Insecure Connection
Normal HTTP
Information user enters


HTTPS
Information hackers see
SSL Certificate
Fw6T8UV81pQfyV81p
Pass: @TutorialsMate
User
Encrypted Connection
Secure HTTPS
Information user enters

Cross-site scripting (XSS)

Cross-site scripting (XSS) is an attack in which an attacker injects malicious executable scripts into the code of a trusted application or website.

How XSS is carried out?

- **Identifying Vulnerable Input Points:** Attackers first identify input fields within the web application where user-supplied data is not properly validated or sanitized. These input points can include URL query parameters, form fields, cookies, headers, and other user-controllable data.
- **Injecting Malicious Payloads:** Once a vulnerable input point is identified, the attacker injects malicious scripts or payloads into the application. These payloads are typically JavaScript code snippets embedded within HTML or other markup languages.
- **Submitting or Triggering the Payload:** The attacker submits the manipulated input containing the malicious payload to the web application. Alternatively, they may trick unsuspecting users into triggering the payload by visiting a specially crafted URL or clicking on a malicious link.
- **Execution in Victim's Browser:** When other users interact with the compromised web page or content containing the injected payload, the malicious script is executed in their browsers. Since the script runs in the context of the victim's session, it can access sensitive information, perform actions on behalf of the victim, or steal data from their browser.
- **Exploiting the Vulnerability:** Depending on the attacker's goals, they may use the XSS vulnerability to perform various malicious actions, including:
    - Stealing session cookies or authentication tokens.
    - Hijacking user sessions or accounts.
    - Defacing websites by modifying content.
    - Redirecting users to phishing or malware-infected websites.
    - Stealing sensitive information such as passwords, credit card numbers, or personal data entered by users on compromised pages.

There are three main types of XSS attacks:

Reflected XSS: In a reflected XSS attack, the malicious script is injected into a web application through a vulnerable parameter, such as a URL query parameter or form input. When the victim visits a specially crafted URL containing the injected script, the server reflects the payload back in the response, executing it in the victim's browser.
Stored XSS: In a stored XSS attack, the malicious script is permanently stored on the server, such as in a database or in user-generated content. When other users view the affected page containing the stored script, the script is executed in their browsers, potentially compromising their accounts or stealing their data.
DOM-based XSS: DOM-based XSS occurs when the client-side script modifies the Document Object Model (DOM) of the web page in an unsafe way. This can happen when user-controlled data is used to dynamically update the DOM without proper validation or sanitization.

Preventing XSS attacks typically involves:

- **Validating and sanitizing user input:** Filter and sanitize user-supplied data to prevent the injection of malicious scripts.
- **Encoding output:** Encode user-generated content before rendering it in HTML to prevent script execution.
- **Using Content Security Policy (CSP):** Implement CSP to mitigate the impact of XSS attacks by specifying which content sources are allowed to be loaded and executed on a web page.
- **Using secure coding practices:** Follow secure coding practices to minimize the risk of introducing XSS vulnerabilities during development.

Example Scenario

It is a simple example to illustrate how XSS attacks work in a controlled environment. Let's create a basic HTML page vulnerable to XSS:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>XSS Vulnerable Page</title>
</head>
```

```
<body>
    <h1>Welcome to My Website</h1>
    <p>This is a vulnerable web page.</p>
    <div id="container"></div>
    <script>
        // Assume user input is directly inserted into the DOM without proper validation or
encoding
        var userInput = "<script>alert('XSS Attack!');</script>";
        document.getElementById('container').innerHTML = userInput;
    </script>
</body>
</html>
```

In this example, the webpage takes user input and directly inserts it into the DOM without proper validation or encoding. The user input is a simple JavaScript alert message, but in a real-world scenario, it could be a malicious script crafted by an attacker.

If an attacker manages to inject the following payload into the vulnerable webpage:

```
"><script>alert('XSS Attack!');</script><"
```

The resulting HTML code would look like this:

```
<div id="container">
    "><script>alert('XSS Attack!');</script><"
</div>
```

When the victim visits this page, the injected script is executed in their browser, resulting in an alert dialog displaying "XSS Attack!" This demonstrates how an attacker could exploit an XSS vulnerability to execute arbitrary JavaScript code within the context of a vulnerable webpage.

To minimize the risk of XSS attacks in the first example, where user input is directly inserted into the DOM without proper validation or encoding, the solution is to implement output encoding. Output encoding ensures that any user-supplied data rendered in the HTML document is treated as plain text rather than executable code. Here's how you can modify the code to implement output encoding using JavaScript:

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>XSS Vulnerable Page with Encoding</title>
</head>
<body>
  <h1>Welcome to My Website</h1>
  <p>This is a vulnerable web page.</p>
  <div id="container"></div>
  <script>
      // Assume user input is directly inserted into the DOM without proper validation or encoding
      var userInput = "<script>alert('XSS Attack!');</script>";

      // Function to encode user input before rendering it in the DOM
      function encodeHTML(input) {
          return input.replace(/</g, '&lt;').replace(/>/g, '&gt;');
      }

      // Encode user input before inserting it into the DOM
      var encodedInput = encodeHTML(userInput);
      document.getElementById('container').innerHTML = encodedInput;
  </script>
</body>
</html>
```

In this modified version, the `encodeHTML` function is used to encode the user input before it is inserted into the DOM. The function replaces the characters `<` and `>` with their HTML entity equivalents `<` and `>`, respectively. This ensures that any HTML tags or JavaScript code within the user input are treated as plain text and cannot be executed as code.

SQL Injection

SQL injection is a cyberattack technique that allows attackers to insert malicious SQL queries into input fields of a web application, potentially leading to unauthorized access, data theft, or manipulation of a database.

Here's an example to demonstrate SQL injection:

Suppose you have a simple login form on a website that checks user credentials against a database:

```
<?php

// Retrieve user input from the login form

$username = $_POST['username'];

$password = $_POST['password'];



// Construct the SQL query

$query = "SELECT * FROM users WHERE username='$username' AND password='$password'";



// Execute the SQL query

$result = mysqli_query($connection, $query);



// Check if the query returned any rows

if(mysqli_num_rows($result) > 0) {

    // User is authenticated, proceed with login

    // ...

} else {

    // Authentication failed, display error message
```

```
    // ...

}

?>
```

In this code, the values of `$username` and `$password` obtained from the login form are directly concatenated into the SQL query string. If an attacker enters a malicious input like `' OR '1'='1' --`, the resulting SQL query would become:

```
SELECT * FROM users WHERE username='' OR '1'='1' --' AND password=''
```

This modified query will always return true (`'1'='1'` is always true), allowing the attacker to bypass the login authentication and gain unauthorized access to the system.

To prevent SQL injection, it's important to use parameterized queries or prepared statements with bound parameters. Here's how the above code can be modified to use prepared statements:

```
<?php

// Prepare the SQL query with placeholders

$query = "SELECT * FROM users WHERE username=? AND password=?";



// Prepare the statement

$stmt = mysqli_prepare($connection, $query);



// Bind parameters to the statement

mysqli_stmt_bind_param($stmt, 'ss', $username, $password);



// Execute the statement
```

```
mysqli_stmt_execute($stmt);


// Get the result set
$result = mysqli_stmt_get_result($stmt);


// Check if the query returned any rows
if(mysqli_num_rows($result) > 0) {
    // User is authenticated, proceed with login
    // ...
} else {
    // Authentication failed, display error message
    // ...
}
?>
```

Using prepared statements with bound parameters ensures that user input is properly sanitized and treated as data rather than executable code, effectively preventing SQL injection attacks.

Ways to prevent SQL injection

1. **Use Prepared Statements (Parameterized Queries)**: Utilize prepared statements with bound parameters in your SQL queries. This approach separates SQL logic

from data and automatically escapes user input, preventing attackers from injecting malicious SQL code. Example in PHP using MySQLi:

```
$stmt = $mysqli->prepare("SELECT * FROM users WHERE username=? AND password=?");

$stmt->bind_param("ss", $username, $password);

$stmt->execute();
```

2. **Input Validation**: Validate and sanitize user input to ensure it conforms to expected formats and constraints before using it in SQL queries. Use whitelisting to allow only expected characters and reject unexpected input. However, note that input validation alone is not sufficient to prevent SQL injection.
3. **Escape Special Characters**: If you're not using prepared statements, escape special characters in user input before embedding them in SQL queries. This prevents SQL injection by neutralizing malicious characters. Most programming languages offer functions for escaping characters, such as `mysqli_real_escape_string()` in PHP.
4. **Least Privilege Principle:** Limit the privileges of database users to only those required for their intended tasks. Avoid using database accounts with excessive permissions, as compromised accounts can be more harmful.
5. **Stored Procedures:** Use stored procedures to encapsulate SQL logic and enforce parameterization. Stored procedures can help prevent SQL injection by abstracting SQL queries away from user input.
6. **Database Access Control:** Implement strict access controls and permissions at the database level to restrict access to sensitive data and operations. This includes limiting access to database tables, views, and procedures based on user roles.
7. **Regular Security Audits**: Conduct regular security audits and penetration testing to identify and mitigate SQL injection vulnerabilities. Automated tools and manual code reviews can help uncover potential vulnerabilities in your application.
8. **Use Web Application Firewalls (WAFs): I**mplement a WAF to filter and block malicious requests before they reach your web application. WAFs can detect and prevent SQL injection attacks by inspecting incoming HTTP traffic.

9. **Update and Patch:** Keep your database management system (DBMS), web server, and application frameworks up to date with the latest security patches and updates to mitigate known vulnerabilities.