

# SWE 363: Web Engineering & Development

## Module 9

## Security



# Objectives

---

- ☐ Learn about Web security

# Outline

---

- ❑ Security Principles
- ❑ Authentication
- ❑ Cryptography
- ❑ Security Best Practices
- ❑ Common Threat Vectors

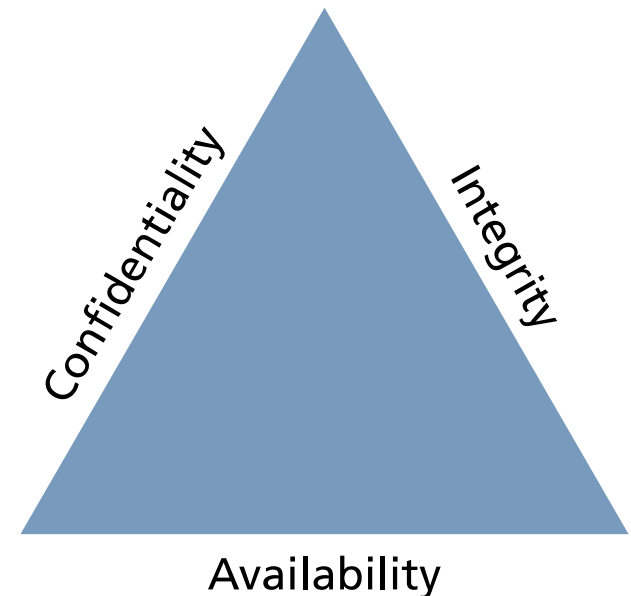
# Introduction

- ❑ The **ideal way** of addressing security is right **from the beginning and all along the way** so that you can plan for a secure system and hopefully have one in the end.
- ❑ The principal challenge with security is that threats exist in so many different forms.
- ❑ Since websites are an **application of networks and computer systems**, you must draw from those disciplines to learn many foundational security ideas.

# Security Principles

# Information Security

- ❑ There are many different areas of study that relate to security in computer networks.
- ❑ **Information security** is the holistic practice of protecting information from unauthorized users.
- ❑ At the core of information security is the **CIA triad**: confidentiality, integrity, and availability
  - **Confidentiality** is the principle of maintaining **privacy** for the data you are storing, transmitting, etc.
  - **Integrity** is the principle of ensuring that data is **accurate and correct**. This includes preventing unauthorized access and modification, but also extends to disaster preparedness and recovery.
  - **Availability** is the principle of making information available when needed to authorized people.



# Risk Assessment & Management

---

- ❑ The ability to assess risk is crucial to the web development world.
- ❑ Risk is a measure of **how likely an attack is**, and how costly the **impact** of the attack would be if successful.
- ❑ Risk assessment uses the concepts of **actors**, **impacts**, **threats**, and **vulnerabilities** to determine where to invest in defensive countermeasures.

- ❑ The term “**actors**” refers to the **people who are attempting to access your system**. They can be categorized as internal, external, and partners.
- 1. **Internal actors** are the people who work for the organization. They can be anywhere in the organization from the cashier through the IT staff, all the way to the CEO.
- 2. **External actors** are the people outside of the organization. They have a wide range of intent and skill, and they are the most common source of attacks.
- 3. **Partner actors** are affiliated with an organization that you partner or work with. If your partner is somehow compromised, there is a chance your data is at risk as well because quite often partners are granted some access to each other's systems (to place orders, for example).



# Impact

- ❑ The impact of an attack depends on **what systems** were infiltrated and **what data** was stolen or lost.
- ❑ The impact relates back to the CIA triad since impact could be the loss of availability, confidentiality, and/or integrity.
  - a) **A loss of availability** prevents users from accessing some or all of the systems.
  - b) **A loss of confidentiality** includes the disclosure of confidential information to a (often malicious) third party.
  - c) **A loss of integrity** changes your data or prevents you from having correct data. This might manifest as an attacker hijacking a user session, perhaps placing fake orders or changing a user's home address.

# Threats

- ❑ A **threat** refers to a particular path that a hacker could use to exploit a vulnerability and gain unauthorized access to your system.
- ❑ **Broadly, threats can be categorized using the STRIDE mnemonic**
  - ❑ **Spoofing**—The attacker uses someone else's information to access the system.
  - ❑ **Tampering**—The attacker modifies some data in nonauthorized ways.
  - ❑ **Repudiation**—The attacker removes all trace of their attack, so that they cannot be held accountable for other damages done.
  - ❑ **Information disclosure**—The attacker accesses data they should not be able to.
  - ❑ **Denial of service**—The attacker prevents real users from accessing the systems.
  - ❑ **Elevation of privilege**—The attacker increases their privileges on the system thereby getting access to things they are not authorized to do.

# Vulnerabilities

- ❑ **Vulnerabilities** are the security holes in your system.
- ❑ The top five classes of vulnerability from the Open Web Application Security Project are:
  - ❑ **Injection**: Injection flaws, such as SQL injection, occur when an attacker sends untrusted data to an interpreter that is executed as a command without proper authorization.
  - ❑ **Broken authentication and session management**: Incorrectly configured user and session authentication could allow attackers to compromise passwords, keys, or session tokens, or take control of users' accounts to assume their identities.
  - ❑ **Cross-site scripting**: Cross-site scripting (XSS) flaws give attackers the capability to inject client-side scripts into the application, for example, to redirect users to malicious websites.
  - ❑ **Insecure direct object references**: Insecure deserialization flaws can enable an attacker to execute code in the application remotely, tamper or delete serialized (written to disk) objects, conduct injection attacks, and elevate privileges.
  - ❑ **Security misconfiguration**

# Security Policy

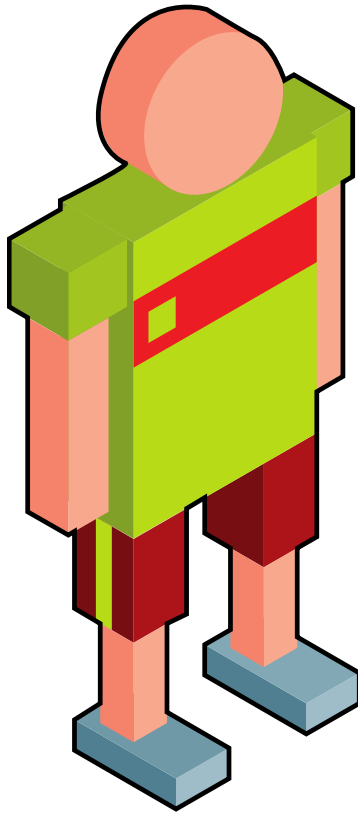
- ❑ One often underestimated technique to deal with security is to **clearly articulate policies to users of the system** to ensure they understand their rights and obligations.
- ❑ **Usage policy:**
  - Defines what systems users are permitted to use, and under what situations.
  - Usage policies are often designed to reduce risk by removing some attack vector from a particular class of system.
  - A company may, for example, prohibit social networking while at work, even though the IT policies may allow that traffic in.
- ❑ **Authentication policy:**
  - Controls how users are granted access to the systems.
- ❑ **Legal policies:**
  - Define a wide range of things including data retention and backup policies as well as accessibility requirements (like having all public communication well organized for the blind).

# Authentication

# Authentication..

- ❑ Authentication is the process by which you decide that someone is who they say they are and therefore permitted to access the requested resources.
  - To achieve both *confidentiality* and *integrity*, the user accessing the system must be who they purport to be.
  
- ❑ Authentication factors are the things you can ask someone for in an effort to validate that they are who they claim to be.
  
- ❑ Three categories of authentication factor, *knowledge*, *ownership*, and *inherence*, are commonly thought of as the things you know, the things you have, and the things you are.

# Authentication factors



## What you **know** (Knowledge)

*Passwords, PIN, security questions, ...*



**only belong to a single person**

## What you **have** (Ownership)

*Access card, cell phone, cryptographic FOB, ...*



## What you **are** (Inherence)

*Retinas, fingerprints, DNA, walking gait, ...*

**These factors are much more difficult to forge**

# Authentication factors

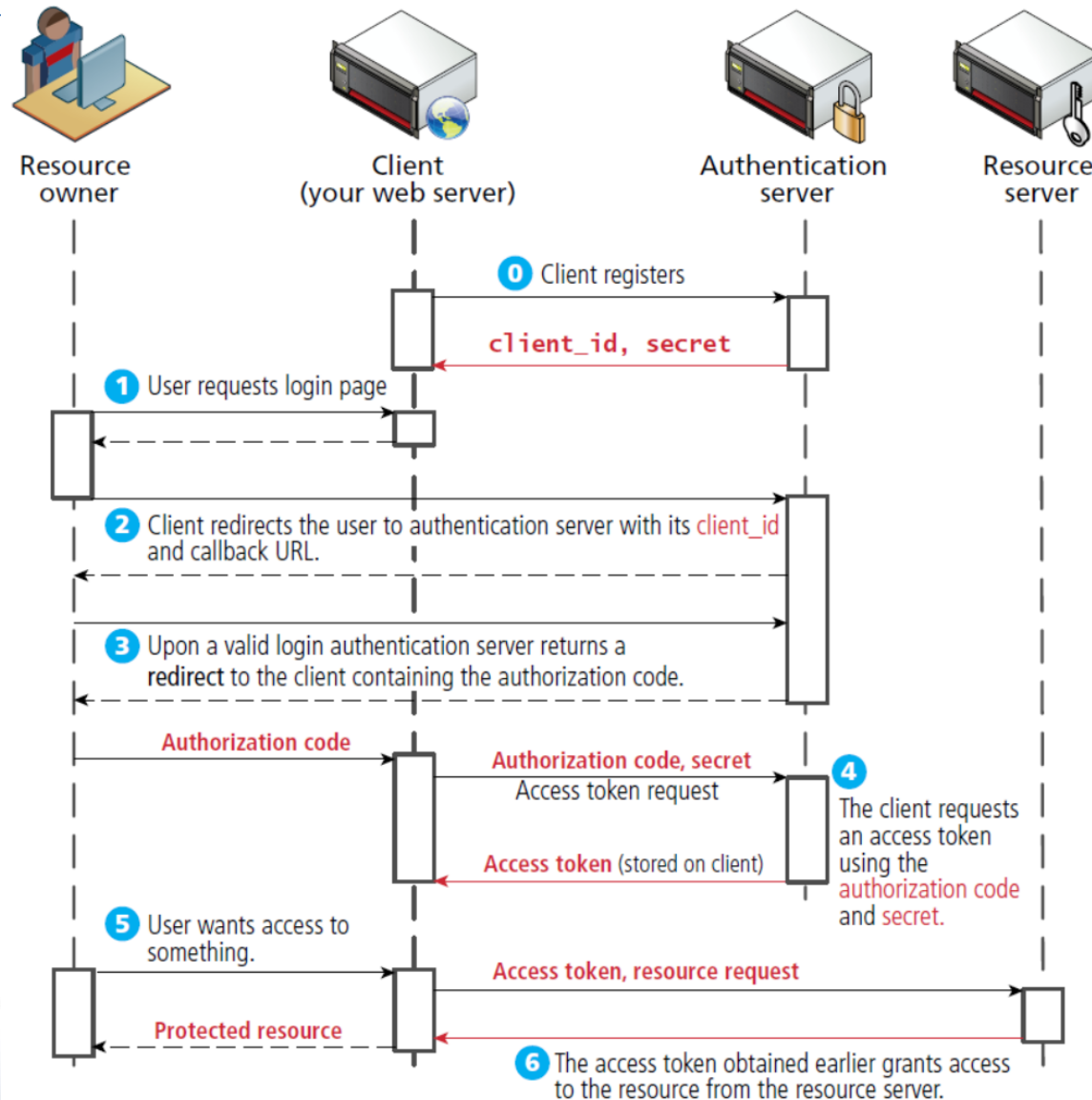
- ❑ **Single-factor authentication** is the weakest and most common category of authentication system where **you ask for only one of the three factors**.
  - Single-factor authorization relies on the strength of passwords and on the users being responsive to threats
- ❑ **Multifactor authentication** is where **two distinct factors** of authentication must pass before you are granted access. Ex. ATM machine
  - The inherent factors are still very costly to implement although they can provide better validation.
- ❑ **Third-party authentication**- many popular services allow you to use their system to authenticate the user and provide you with enough data to manage your application.
  - Third-party authentication schemes like **OpenID** and **oAuth**



# Open Authorization (OAuth)

- ❑ OAuth is a popular authorization framework that allows users to use credentials from one site to authenticate at another site
- ❑ OAuth 2.0 provides a rich authorization framework with well-defined security properties.
- ❑ OAuth uses four user roles in its framework:
  - The **resource owner** is normally the end user who can gain access to the resource (though it can be a computer as well).
  - The **resource server** hosts the resources and can process requests using access tokens.
  - The **client** is the application making requests on behalf of the resource owner.
  - The **authorization server** issues tokens to the client upon successful authentication of the resource owner. Often this is the same as the resource server.

# The steps required to register and authenticate a user using OAuth



# Authorization

- ❑ **Authorization** defines what rights and privileges a user has once they are authenticated.
- ❑ **Authentication** and **authorization** are sometimes confused with one another, but are two parts of a whole.
  - **Authentication** grants access, and
  - **Authorization** defines what the user with access can (and cannot) do.

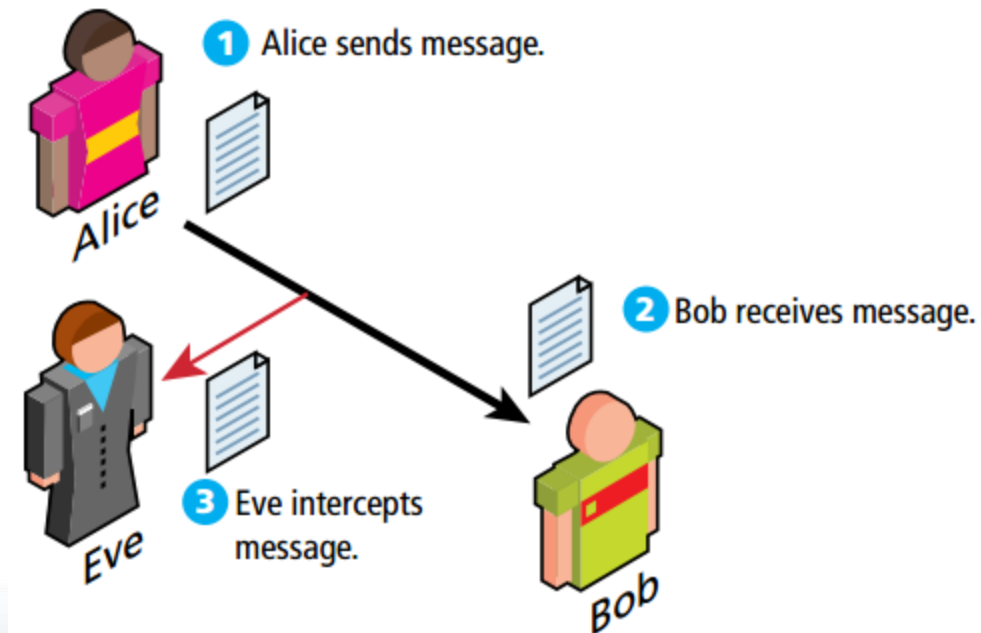
# Authorization

- ❑ Some examples in web development where **proper authorization increases security** include:
  - Using a separate database user for read and write privileges on a database
  - Providing each user an account where they can access their own files securely
  - Setting permissions correctly so as to not expose files to unauthorized users
  - Using Unix groups to grant users permission to access certain functionality rather than grant users admin access
  - Ensuring Apache is not running as the root account (i.e., the account that can access everything)

# Cryptography

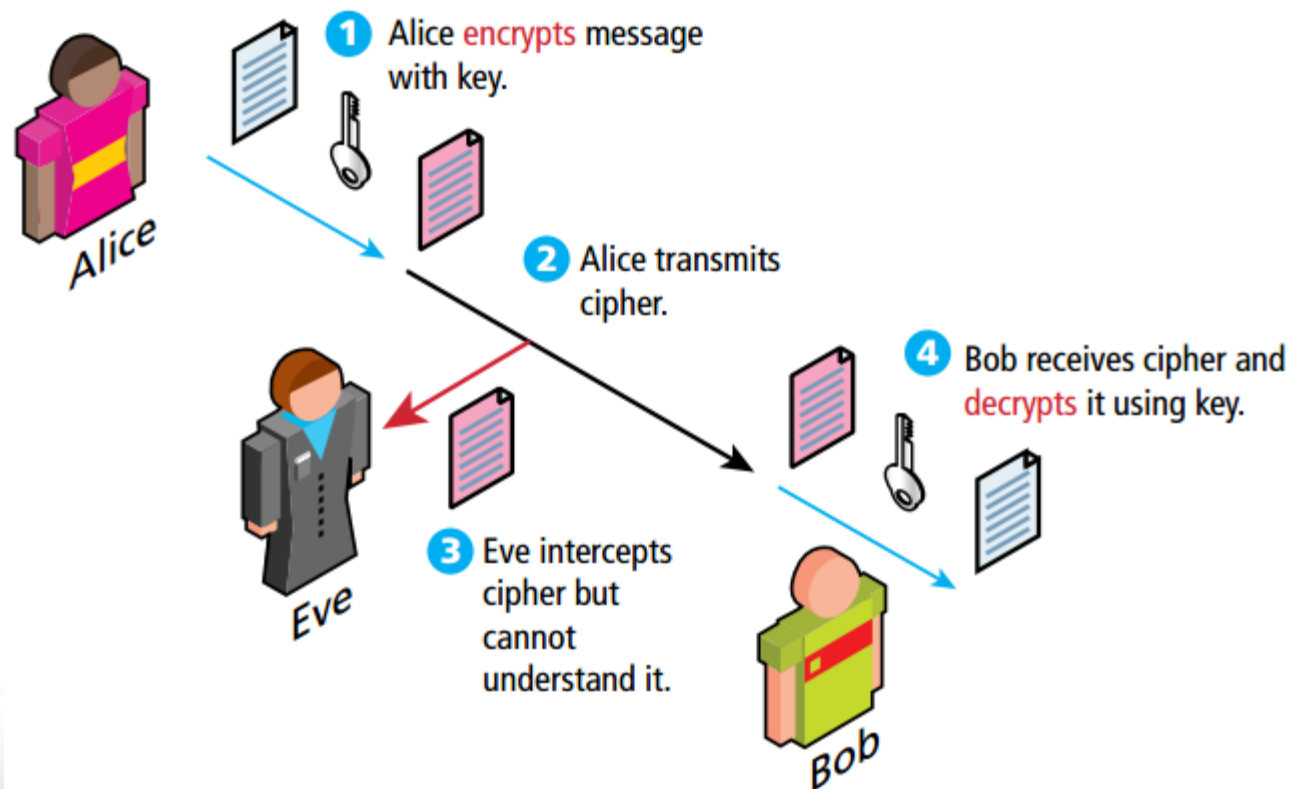
# Cryptography

- ❑ **Eavesdropping** could allow someone to get your credentials while they are being transmitted.
- ❑ A **cipher** is a message that is complicated so that it cannot easily be read, unless one has some secret knowledge (**key**).
  - keep the key a secret between the sender and the receiver



# Cryptography

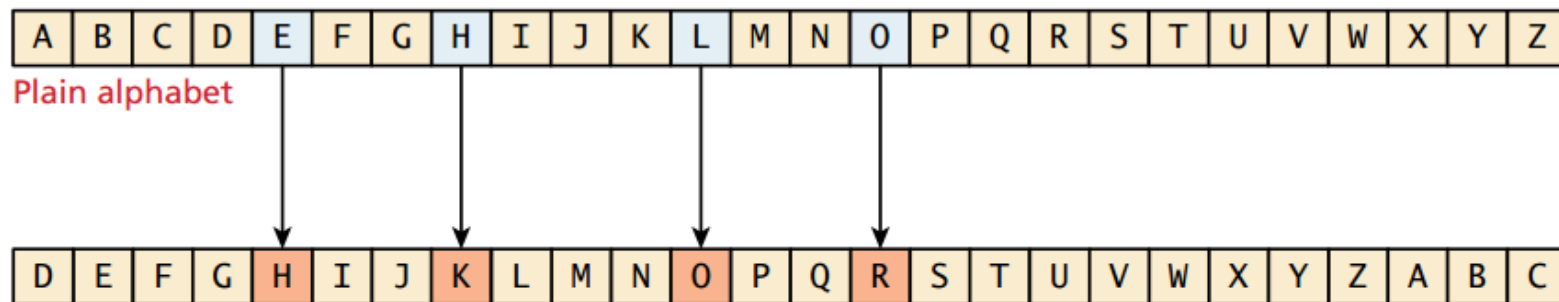
- ❑ Alice encrypts the message (**encryption**) and Bob, the receiver, decrypts the message (**decryption**) both using their keys



# Substitution Ciphers

- ❑ A **substitution cipher** is one where each character of the original message is replaced with another character according to the encryption algorithm and key.
- ❑ The **Caesar cipher**- is a substitution cipher where every letter of a message is replaced with another letter, by shifting the alphabet over an agreed number (from 1 to 25).

e.g The message HELLO, for example, becomes KHOOR when a shift value of 3 is used



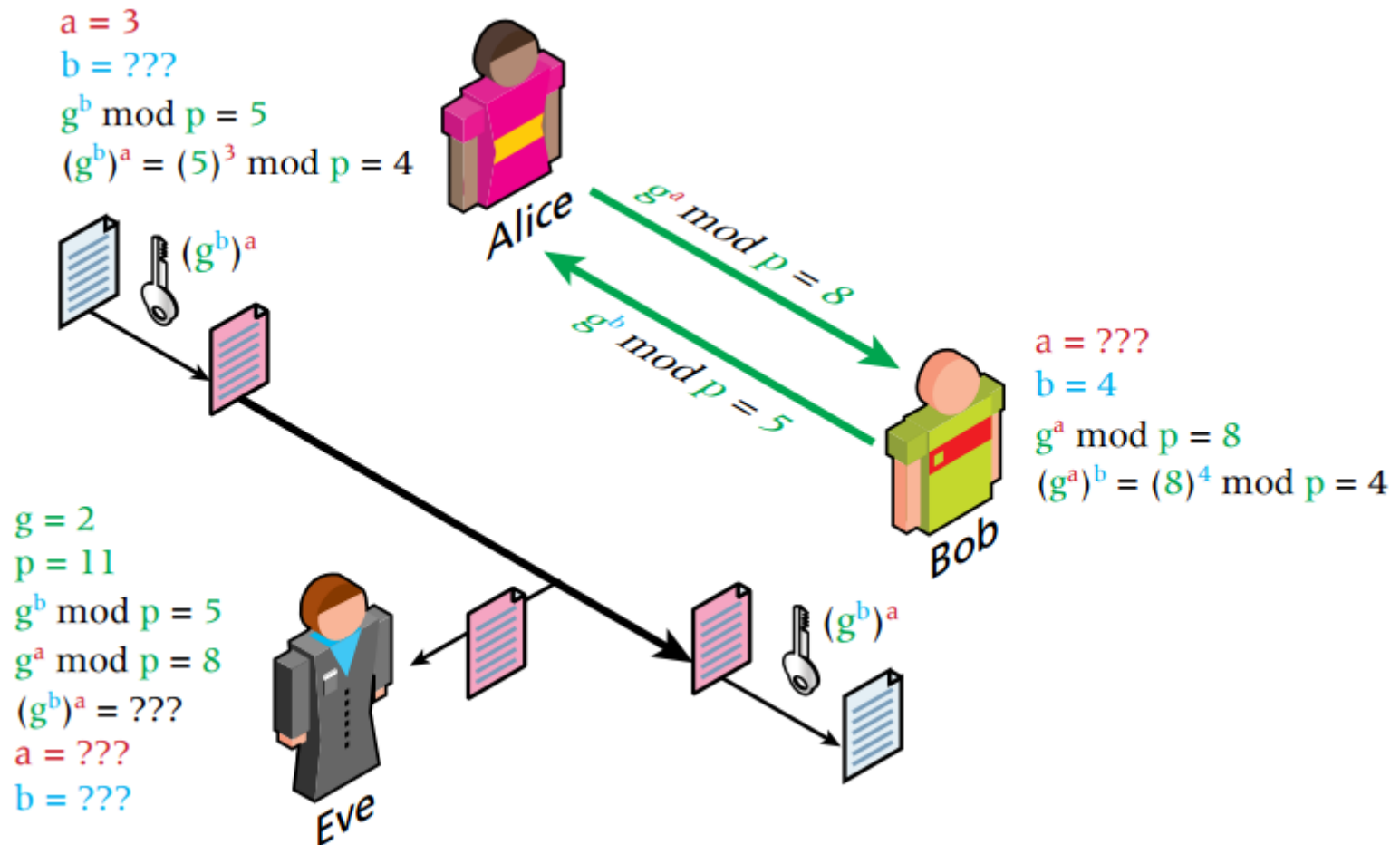
- ❑ **Caesar cipher** uses the same key to encode and decode, so we call it **symmetric cipher**. The problem is that we have to have a **shared private key**.



# Public Key Cryptography

- ❑ The challenge with symmetric key ciphers is that the secret must be exchanged before communication can begin.
- ❑ >> How do you get that information exchanged?
- ❑ **Public key cryptography** (or **asymmetric cryptography**) solves the problem of the secret key by using two distinct keys: a **public** one, widely distributed and another one, kept **private**.

# Diffie-hellman Key exchange

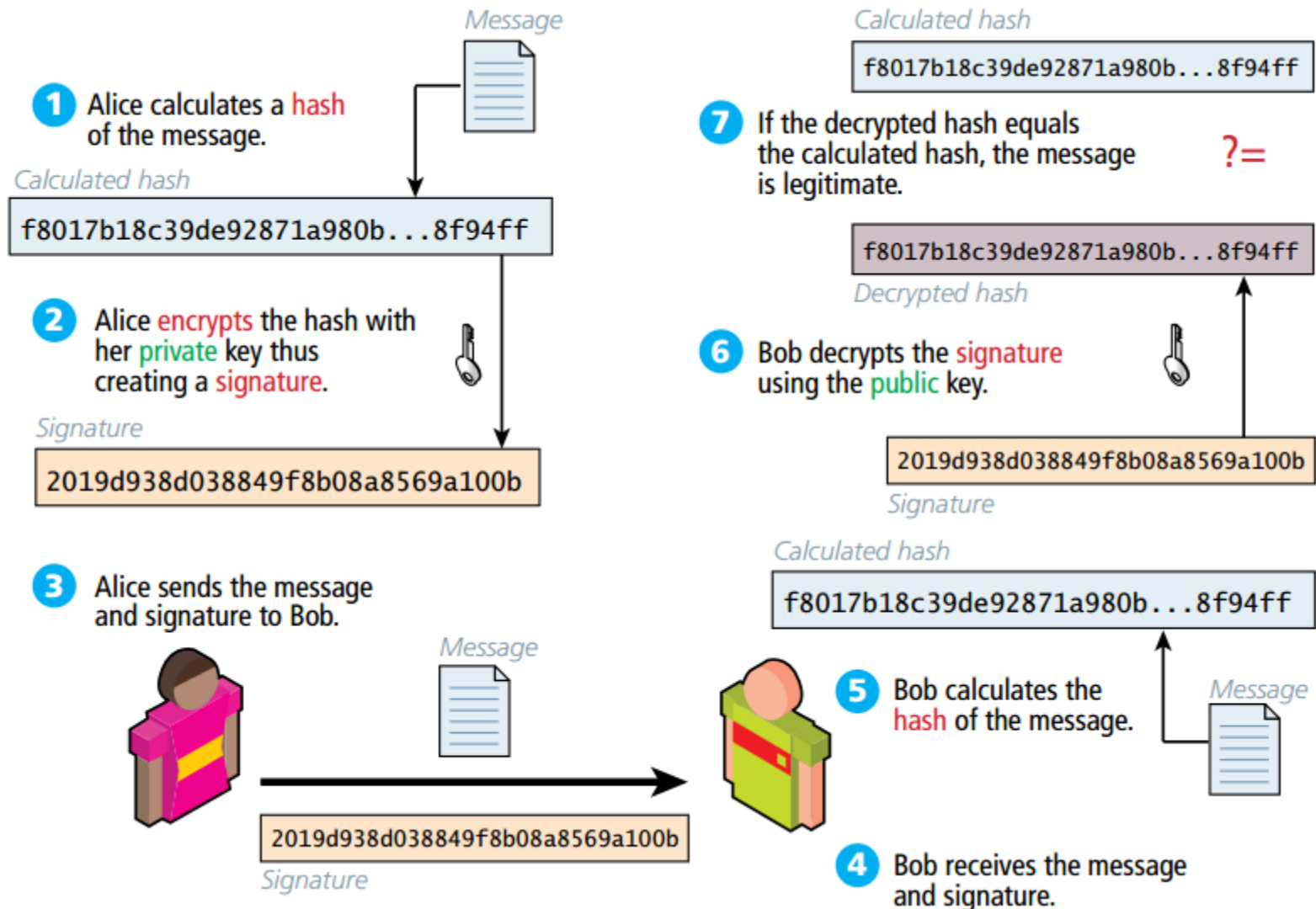


Diffie-Hellman Key Exchange for  $g = 2$  and  $p = 11$

# Digital signatures

- ❑ Cryptography can also help in **validating** that the sender is really who they **claim to be**, through the use of **digital signatures**.
- ❑ A **digital signature** is a mathematically secure way of validating that a particular digital document was created by the person claiming to create it (**authenticity**), was not modified in transit (**integrity**), and cannot be denied (**non-repudiation**).
- ❑ Using the concepts from public key cryptography, we can consider the process of **signing a digital document to be as simple as encrypting a hash of the transmitted message**.
  - The receiver can then apply the same technique, by creating a hash of the message, and then decrypting your signature using the public key to make sure the two messages are equal

# Illustration of a digital signature and its validation



# Security Best Practices

UserID (int)	Username (varchar)	Password (varchar)
1	ricardo	password
2	randy	password

# Data storage

```
//Insert the user with the password
function insertUser($username,$password){
    $link = mysqli_connect("localhost", "my_user", "my_password",
                           "Login");
    $sql = "INSERT INTO Users(Username,Password)
            VALUES('$username','$password')";
    mysqli_query($link, $sql);           //execute the query
}

//Check if the credentials match a user in the system
function validateUser($username,$password){
    $link = mysqli_connect("localhost", "my_user", "my_password",
                           "Login");
    $sql = "SELECT UserID FROM Users WHERE Username='$username' AND
            Password='$password'";
    $result = mysqli_query($link, $sql); //execute the query
    if($row = mysqli_fetch_assoc($result)){
        return true; //record found, return true.
    }

    return false; //record not found matching credentials, return false
}
```

- The confidentiality of the data. (passwords in plain text)
- The *integrity* of the system and the data. (if one access the DB can see others psw)

# Secure Hash

UserID (int)	Username (varchar)	Password (varchar)
1	ricardo	5f4dcc3b5aa765d61d8327deb882cf99
2	randy	5f4dcc3b5aa765d61d8327deb882cf99

```
//Insert the user with the password being hashed by MD5 first.
function insertUser($username,$password){
    $link = mysqli_connect("localhost", "my_user", "my_password",
                           "Login");
    $sql = "INSERT INTO Users(Username,Password)
           VALUES('$username',MD5('$password'))";
    mysqli_query($link, $sql); //execute the query
}

//Check if the credentials match a user in the system with MD5 hash
function validateUser($username,$password){
    $link = mysqli_connect("localhost", "my_user", "my_password",
                           "Login");
    $sql = "SELECT UserID FROM Users WHERE Username='$username' AND
           Password=MD5('$password')";

    $result = mysqli_query($link, $sql); //execute the query
    if($row = mysqli_fetch_assoc($result)){
        return true; //record found, return true.
    }
    return false; //record not found matching credentials, return false
}
```

PHP code can access implementations of MD5 and SHA algorithms through the md5() or sha1() functions. MySQL also includes implementations.

# Salting the hash

- ❑ a hacker with access to a table of hashes could build data structures called **rainbow tables** that aid in breaking passwords given enough time and space.
- ❑ The technique of adding some noise to each password is called **salting** the password and makes your passwords very secure.

UserID (int)	Username (varchar)	Password (varchar)	Salt
1	ricardo	edee24c1f2f1a1fda2375828fbef6933	12345a
2	randy	ffc7764973435b9a2222a49d488c68e4	54321a



```

function generateRandomSalt(){
    return base64_encode(mcrypt_create_iv(12), MCRYPT_DEV_URANDOM));
}
//Insert the user with the password salt generated, stored, and
//password hashed
function insertUser($username,$password){
    $link = mysqli_connect("localhost", "my_user", "my_password",
                           "Login");
    $salt = generateRandomSalt();
    $sql = "INSERT INTO Users(Username,Password,Salt)
           VALUES('$username',MD5('$password$salt'), '$salt')";
    mysqli_query($link, $sql); //execute the query
}

//Check if the credentials match a user in the system with MD5 hash
//using salt
function validateUser($username,$password){
    $link = mysqli_connect("localhost", "my_user", "my_password",
                           "Login");
    $sql = "SELECT Salt FROM Users WHERE Username='$username'";
    $result = mysqli_query($link, $sql); //execute the query
    if($row = mysqli_fetch_assoc($result)){
        //username exists, build second query with salt
        $salt = $row['Salt'];
        $saltSql = "SELECT UserID FROM Users WHERE Username='$username'
                   AND Password=MD5('$password$salt')";
        $finalResult = mysqli_query($link, $saltSql);
        if($finalrow = mysqli_fetch_assoc($finalResult))
            return true; //record found, return true.
    }
    return false; //record not found matching credentials, return false
}

```

# Common Threat Vectors

# SQL Injection

- ❑ **SQL injection** is the attack technique of using reserved SQL symbols to try and make the web server execute a malicious query other than what was intended.
- ❑ The malicious attacker is not trying to log in, but rather, trying to insert rogue SQL statements to be executed that have nothing to do with the user authentication system.

1. `SELECT * FROM Users WHERE uname='';`
2. `TRUNCATE TABLE User;`

The second one (TRUNCATE) removes all the records from the Users table



- 0 A vulnerable form passes unsanitized user input directly into SQL queries.

User

Password

POST

```
...
$user = $_POST['username'];
$pass = $_POST['pass'];
$sql = "SELECT * FROM Users WHERE
       uname='$user' AND passwd=MD5('$pass')";
sql_query($sql);
...
```

```
SELECT * FROM Users WHERE
uname='alice' AND
passwd=MD5('abcd')
```

Users table



- 1 Hacker inputs SQL code into a text field and submits the form.

User

Password

POST

```
SELECT * FROM Users WHERE uname='';
TRUNCATE TABLE Users;
# ' AND passwd=MD5('')
```

Users table

--	--	--	--

- 2 PHP script puts the raw fields directly into the SQL query.

- 3 The resulting query is actually two queries.

*Rest of query  
is commented  
out*

- 4 All records in Users table are deleted.

# SQL Injection

- ❑ From a security perspective, you should never trust a user input enough to use it directly in a query, no matter how many HTML5 or JavaScript pre-validation techniques you use.
- ❑ There are two ways to protect against such attacks:
  - sanitize user input, and
  - apply the least privileges possible for the application's database user.

# 1. Sanitize input

To **sanitize** user input:

- ❑ before using the user input in a SQL query, you either
    - apply **sanitization functions** (mysqli\_real\_escape\_string) or
    - bind the variables in the query using parameters or **prepared statements**.
- [https://www.w3schools.com/php/php\\_mysql\\_prepared\\_statements.asp](https://www.w3schools.com/php/php_mysql_prepared_statements.asp)

## 2. Least possible privileges

- ❑ A properly secured system only **assigns users and applications the privileges they need** to complete their work, but no more.
- ❑ For instance, in a typical web application, one could define three types of database user for that web application:
  - one with **read-only privileges**,
  - one with **write privileges**, and
  - finally an **administrator with the ability to add, drop, and truncate tables**.
- ❑ The read-only user is used with all queries by nonauthenticated users.
- ❑ The other two users are used for authenticated users and privileged users, respectively.

# References

All the slides content are taken from

- ❑ “*Fundamentals of Web Development*” by Randy Connolly and Ricardo Hoar, 2015
  - Chapter 16

