Boston Identity

AN IDENTITY COMPANY

IAM Handbook

A technical guide for IAM professionals

Revision History

Version	Date	Author	Comments
0.1	06/18/2023	Le Deng	Initial Draft
0.2	06/19/2024	Le Deng	Added LDAP Chapter
0.3	03/11/2025	Le Deng	Added Key Points for each chapter

Contents

Forward

In today's interconnected world, where data breaches and cyber threats loom large, safeguarding digital assets has become paramount. Organizations face the formidable challenge of protecting their sensitive information while ensuring that authorized individuals have seamless access to the resources they need. This is where Identity and Access Management (IAM) steps in as a crucial discipline that underpins modern security practices.

IAM lies at the heart of establishing and maintaining a secure digital ecosystem. It encompasses a set of principles, processes, and technologies that govern the identification, authentication, authorization, and auditing of users and their interactions within an organization's information systems.

At its core, IAM seeks to answer the fundamental questions of "Who has access to what?" and "What can they do with it?" By effectively managing identities and controlling access to digital resources, organizations can mitigate risks, comply with regulations, enhance operational efficiency, and build trust with their stakeholders. This book delves deep into the core concepts of Identity and Access Management, exploring its key components, best practices, and real-world implementation strategies, to equip both security professionals and decision-makers with the knowledge necessary to navigate this dynamic field.

This book serves as an introductory book focusing on the fundamentals and core concepts that an IAM professional needs to understand.

The first part is from chapter 1 to 5, which focuses on some of the basic knowledge of modern Web Applications and Networks. Specifically, chapter1 talks about general web structure and REST. Chapter 2 starts to look at LDAP and chapter 3 is about Database. Chapter 4 talks about HTTP. Chapter 5 and 6 look at different aspects of networks.

The second part is from chapter 7 to 12, which focuses on IAM essentials. Chapter 7 describes the basics of cryptography. Chapter 8 and 9 describe OAuth2 and various grant flows. Chapter 10 introduces OIDC which is on top of OAuth2.0. Chapter 11 and 12 focuses on SAML.

I hope you enjoy this book and find it useful in the world of IAM.

Boston Identity
AN IDENTITY COMPANY

PART I - Modern Web Apps and Networks

Chapter 1 – Web Applications

This chapter lays the groundwork for understanding the pivotal role web applications play in our fast-evolving digital world. This chapter highlights their significance and the need to grasp their core concepts for effective participation in today's interconnected landscape.

Key Points

- **Web Application Fundamentals:**
  - Web applications are software accessed via a web browser, requiring a server for hosting and transaction handling.
  - They serve diverse purposes, from e-commerce to social media.

- **Client-Server Architecture:**
  - A model where clients request services from a central server.
  - Three-tier architecture separates presentation, application logic, and data tiers for enhanced scalability and performance.

- **Modern Web App Architecture:**
  - Consists of frontend (client-side) and backend (server-side) components.
  - Frontend: Uses HTML, CSS, and JavaScript for user interface and interactivity.
  - Backend: Handles business logic, data processing, and database interactions.

- **Frontend Technologies:**
  - HTML: Structures content.
  - CSS: Styles content.
  - JavaScript: Enables interactivity and dynamic features.
  - JavaScript frameworks such as React, Angular, and Vue.js enhance development efficiency.

- **Backend Architecture:**
  - Layered architecture including model, controller, service, and persistence layers.
  - Handles user requests, business logic, and database interactions.

- **REST (REpresentational State Transfer):**
  - An architectural style for creating scalable and maintainable web services.
  - Utilizes HTTP methods (GET, POST, PUT, DELETE) for resource manipulation.
  - RESTful design decouples frontend and backend development.
  - Guiding principles include client-server, statelessness, cacheability, uniform interface, layered system, and code on demand (optional).

- **REST Requests and Responses:**
  - Requests: Include endpoint, method, headers, parameters, and data body.

- Responses: Deliver data (JSON, XML) or status codes.
- Postman is a tool used to test REST API's.

Sources and related content

Why Web Application?

In today's rapidly evolving world of information technology, web applications have emerged as an essential component of our digital landscape. These applications are at the heart of our online experiences, enabling us to access and interact with a wide range of services, from social media platforms to e-commerce websites and productivity tools.

Having a basic understanding of web application concepts is crucial for individuals and businesses alike. It allows us to navigate and harness the power of the digital realm, empowering us to communicate, collaborate, and conduct transactions more efficiently.

Furthermore, as web applications continue to shape various industries and sectors, possessing knowledge about their concepts ensures individuals can adapt and innovate in this digital era.

Whether it's understanding the fundamentals of client-server architecture, grasping key web development technologies, or comprehending security considerations, a basic understanding of web applications equips us with the necessary skills to fully participate in today's interconnected world.

What is a Web Application?

An application can be considered as a complex piece of software, which consists of many different components.

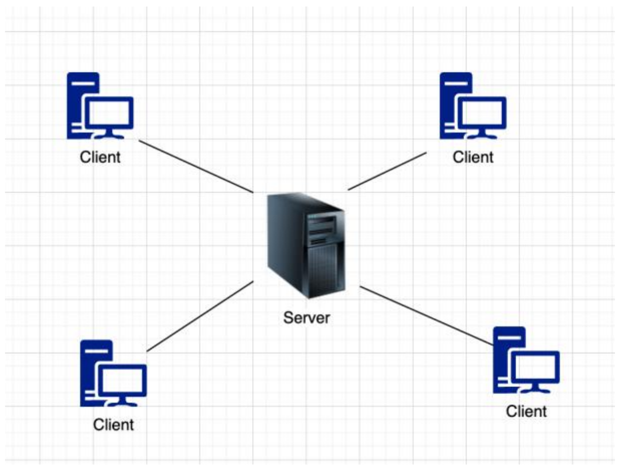
Web apps can be designed for a wide range of use cases. For example, an E-Commerce Web App will have such components as user registration, product display, login management, product management, user checkout etc.

Web apps are usually accessed through a network with a web browser such as Google Chrome, Mozilla Firefox, Safari etc.

For a web app to operate, it needs a server which hosts the web app and handles the transactions between browser and the app.

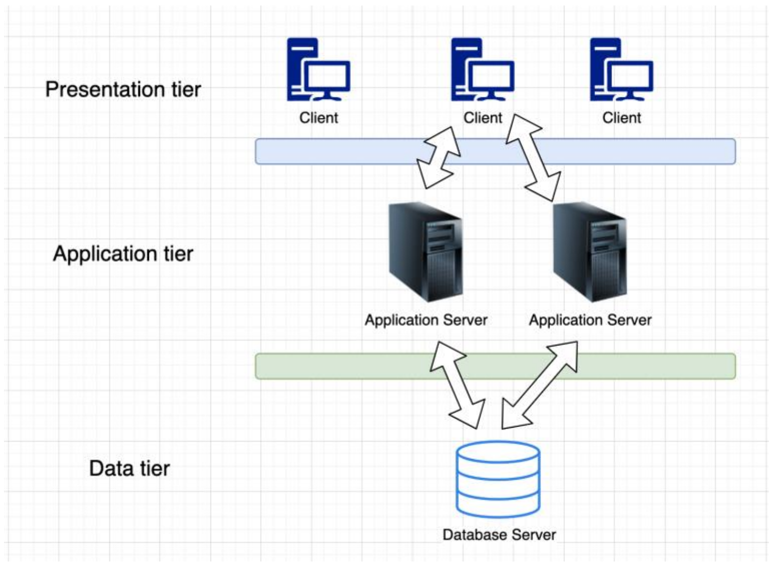
Client Server Architecture

Client Server Architecture is a computing model where the server hosts, delivers and manages most of the resources and services to be consumed by the client. One or more client computers will connect to a central server over a network or internet connection.



In this case, the server is always running and waiting for requests to come. Clients are often situated at workstation or personal computer, while servers are located remote on the network, such as in an on-premise data center or on the cloud. For clients to communicate with the server, the former usually sends out a request to the server, once the server receives the request, finished processing it, a response will be sent back to the client.

Three-tier Client Server Architecture



The 3-tier architecture is extended from the basic client server architecture.

A three-tier client/server is a type of multi-tier computing architecture in which an entire application is distributed across three different computing layers or tiers. It divides the presentation, application logic and data processing layers across client and server devices.

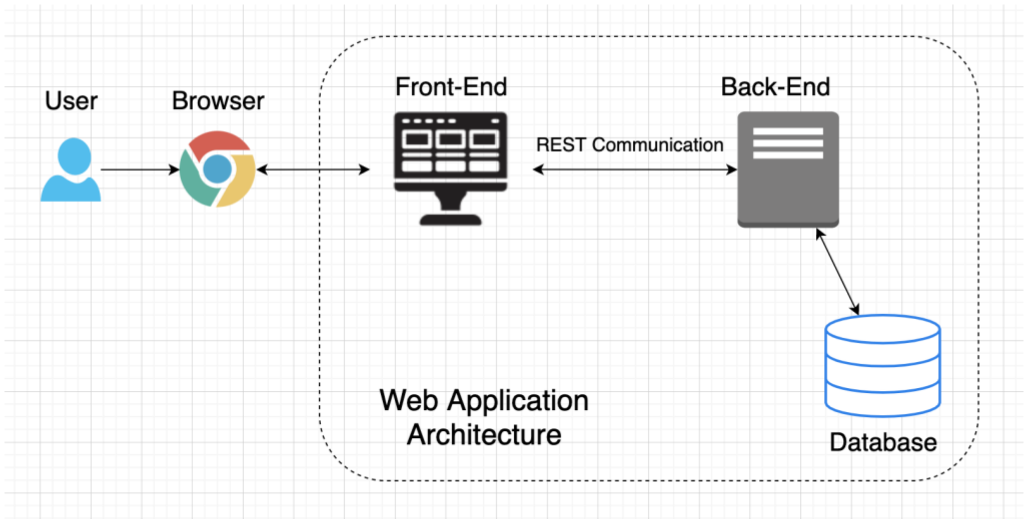
- **Presentation tier** – the frontend layer and consists of user interface
- **Application tier** – contains the functional business logic which drives an application's core capabilities
- **Data tier** – comprises of database/data-storage system and data access

There are many benefits to using a 3-layer architecture including speed of development, scalability, performance, and availability.

Modern Web App Architecture

Web app development has experienced several stages in the history, from tightly coupled monolithic development to lately REST-based architecture.

Here is a general architecture diagram for a web app.



Taking an E-commerce web app as an example.

A user opens a browser from his laptop and he types a URL to access a web app. The browser connects to the frontend server of the web app through HTTP and displays the content in it. The content here can be roughly divided into two parts: Graphics and Data.

The graphical part is rendered by your browser based on the content served from the frontend server. The data part is not served by the frontend directly; instead, it will communicate with the backend to fetch the proper data. Meanwhile, the backend server may or may not have that data handy. If not, it will need to communicate with the database to fetch the data and pass back to the frontend, which will again send to your browser. Based on the graphics and data the browser gets, it renders them and displays them as what you see day-to-day.

Let's take a look at some more details on each part.

Frontend

Frontend developer uses three primary coding languages to code web apps:

- **HTML**
- **CSS**
- **JavaScript**

The code written will be running inside a browser, this is in contrast to a backend developer, whose code runs on a server.

Modern frontend development will leverage frameworks like Angular, ReactJS etc. Yet, no matter what fancy framework is used, eventually the code will be built into HTML, CSS and

JavaScript or a part of the three, as those are the only elements that can run in a browser. A JavaScript framework cannot run directly in a browser.

HTML & CSS

HTML (Hyper Text Markup Language) gives content structure and meaning by defining content as, for example, headings, paragraphs, or images.

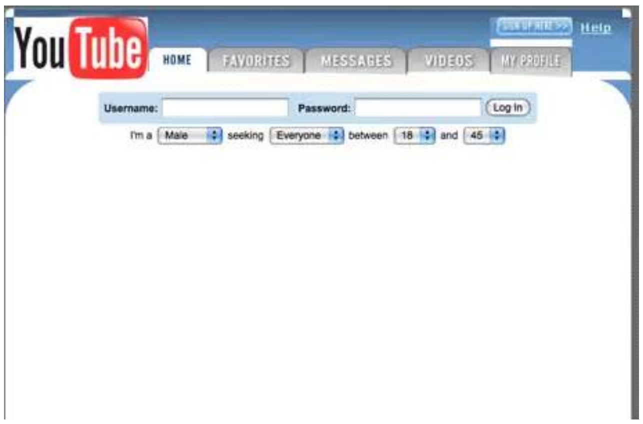
CSS (Cascading Style Sheet) is a presentation language created to style the appearance of content using such elements as fonts, colors, etc.

Technically speaking, HTML and CSS are actually not programming languages. In a broad sense, a programming language is a notation for writing programs, which are specifications of a computation or algorithm. HTML & CSS are not “executed” by computer; rather, they are parsed by an engine from the browser.

JavaScript

JavaScript is a programming language that allows you to implement complex features on web page. Without JavaScript, a web page will just sit there and display static content information for you to look at.

This is YouTube in 2005.



Why is it called JavaScript?
When JavaScript was created, Java was very popular, so it was decided to name after Java. But as it evolved, JavaScript became a fully independent language with its own specification called ECMAScript, which has no relation to Java now.

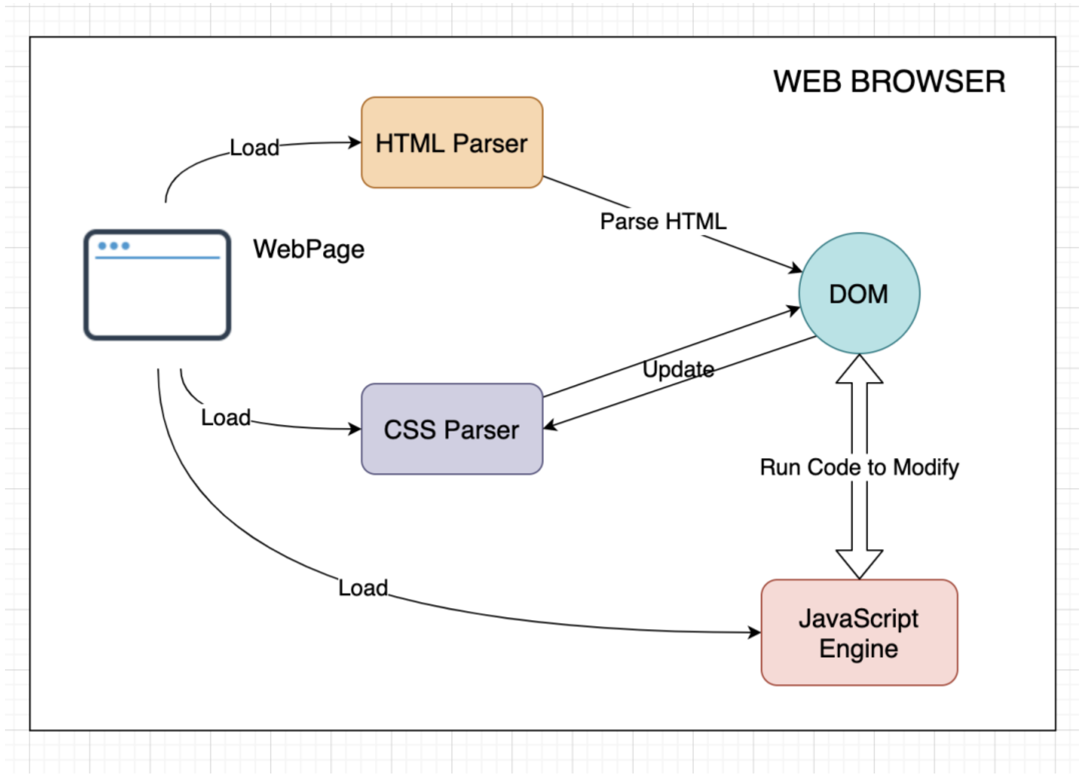
How JavaScript Works

First, let's understand how browsers work with HTML and CSS.

A web browser loads a web page, parses the HTML content sent by the Frontend server and creates something called Document Object Model (DOM) from those HTML contents. This DOM allows the web content to interact with your JavaScript code.

Also, the browser will grab everything linked to the HTML, such as images and CSS files and then send them to CSS Parser which will then update DOM content.

HTML and CSS are put together by DOM to create the web page first. After that, the browser's JavaScript engine loads JavaScript code and executes in the order the code is written. Executing JavaScript code after HTML and CSS finish loading is important and developers usually write code to enforce that.



JavaScript Framework

Framework is not a must for development but they are more adaptable and flexible for the designing of web apps, so most developers prefer to use it.

A JavaScript Framework differs from a JavaScript library in its control flow: A library offers functions to be called by its parent code, whereas a framework defines the entire application design.

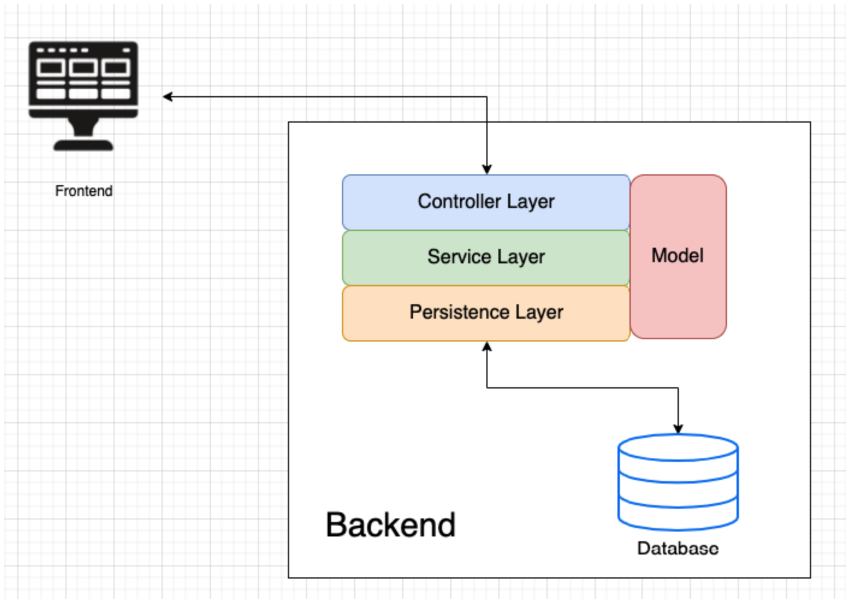
Some of the most popular JavaScript frameworks are:

- Angular
- React
- Vue.js
- Ember.js
- Meteor

Backend

Backend provides services based on business logic and workflows.

The backend architecture could be very complicated sometimes. From a web app perspective, backend can be considered as a layered architecture like below.



Model Layer

You’ve probably heard about the MVC (**Model**, **View** and **Controller**) pattern. It is one of the most frequently used industry-standard specific development aspects of an application. Basically, it’s an architectural pattern that separates an application into three main logical components, each of which are built to handle specific development aspects of an application.

The Model layer can be related to the M in MVC pattern.

- **Model** – corresponds to all the data-related logic. This can represent the data that is being transferred across other layers or any other business logic-related data.

Controller Layer

Controllers can be considered the uppermost layer of a web application. It is responsible for processing the user's input and returning the correct response back to the user.

Service Layer

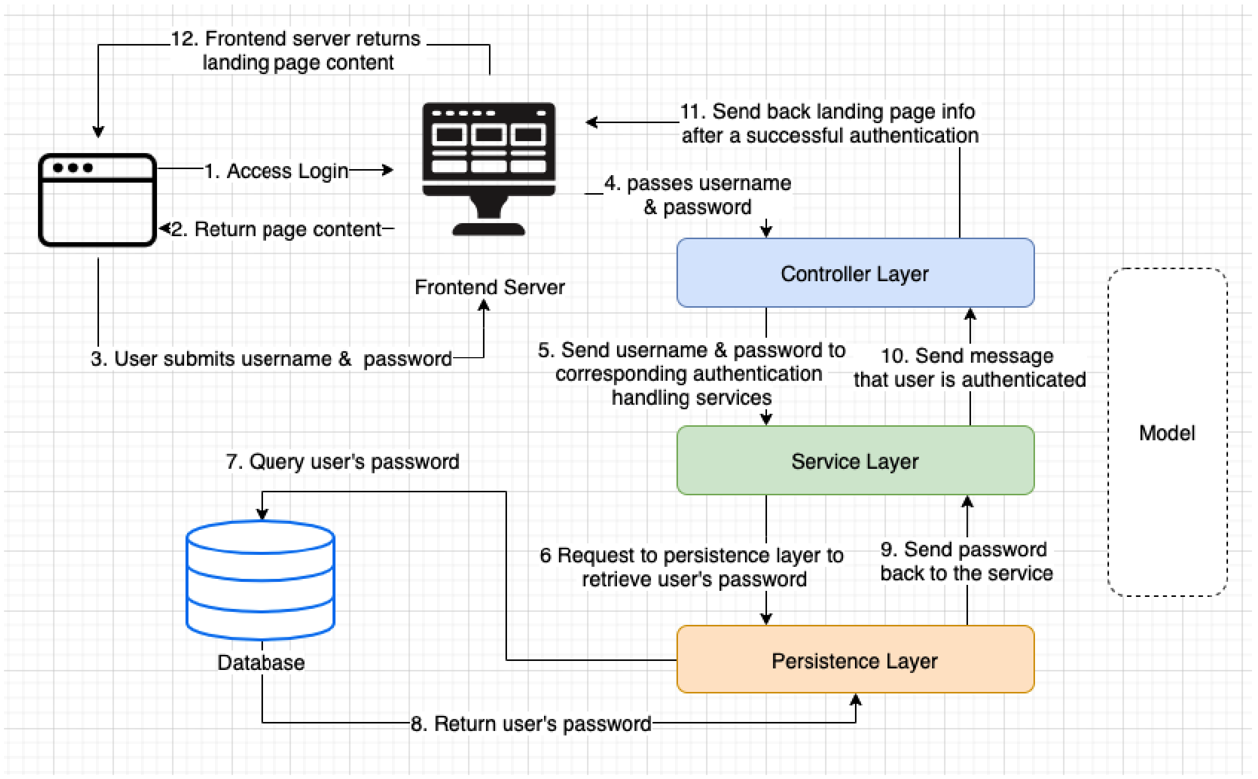
Service layer resides below the controller layer. It acts as a transaction boundary and contains both application and infrastructure services.

Persistence Layer

Persistent layer is the lowest layer of a web application. First of all, data cannot just sit in the application. They need to be saved somewhere so that you don't lose them when something bad happens with your application. Furthermore, application resources are limited and they are mainly used for processing and data should be stored elsewhere, mainly in a database. The persistent layer is responsible for communicating with data storage.

An Example Workflow

To better understand how the backend works, let's take a look at a user login scenario. User starts by typing a web app URL into the browser, and the app home page prompts the user to login.


12. Frontend server returns landing page content
11. Send back landing page info after a successful authentication
1. Access Login
2. Return page content
4. passes username & password
Controller Layer
Frontend Server
3. User submits username & password
5. Send username & password to corresponding authentication handling services
10. Send message that user is authenticated
Model
7. Query user's password
Service Layer
6 Request to persistence layer to retrieve user's password
9. Send password back to the service
Database
Persistence Layer
8. Return user's password

Steps:

1. A user accesses a login page of a web app
2. Frontend server returns the page content to the user browser
3. User inputs user name and password and clicks submit button
4. Frontend passes username and password information to controller at backend
5. Controller send username and password to corresponding authentication handling services
6. User inputted password needs to be compared to the existing password, so service requests persistence layer to retrieve the user's password
7. Persistence layer component sends a query to database
8. Database returns the password to persistence layer component
9. Persistent layer sends password back to the service with the original request
10. After the service compares the user inputted password with the existing password, some additional logic process will happen. Then, the service sends back a result, say, the user is authenticated.
11. Controller sends back the landing page info after a successful user login
12. Frontend server sends back the landing page content to user browser and browser display the landing page information

Of course, this is a relatively high-level workflow and a lot of details are not mentioned here. The point here is to show how a backend works in general.

REST (REpresentational State Transfer) Protocol

Representational state transfer (REST) is a software architectural style that defines a set of constraints to be used for creating web services.

REST is used to build web services that are lightweight, maintainable, and scalable in nature. A service built on REST architecture is called a RESTful service. The pre-assumed underlying protocol for REST is HTTP, as it fits the REST spec well.

Advantages of REST for Development

RESTful design decouples frontend from backend and this has plenty of advantages in terms of development and maintenance. Some of them are:

1. **Separation between the client and the server**. This lets developers treat Frontend development and backend development as two separate parts. For example, the frontend team can use different frameworks to suit their needs as long as they communicate with the backend through REST.
2. **Scalability**. As client and servers are separated, we can have the product scaled without much of the difficulty, compared to when client and servers are tightly coupled.
3. **Independent of Platform or languages**. Since REST leverages HTTP, you can use different kinds of languages, PHP, Java, Python or Node.js as your backend development tool. This gives you a lot of freedom on the backend development.

Relationship Between REST and HTTP

In simple words, HTTP is an application protocol while REST is a set of rules that enable you to build an application that has a specific set of desirable constraints.

The reason we use HTTP as a means for RESTful communication is that the infrastructure, servers and client libraries for HTTP are widely available and HTTP as a protocol fits the rules of REST very well.

Guiding Principles of REST

REST has its own 6 guiding constraints which must be satisfied if an interface needs to be referred as RESTful.

1. **Client-server** – By separating User Interface concerns from Data Storage concerns, we improve the portability of the User Interface across multiple platforms and scalability.
2. **Stateless** – Each request from client to server must contain all information necessary to understand the request. Session state is therefore kept entirely on the client.
3. **Cacheable** – All resources should allow caching unless explicitly indicated that caching is not possible.
4. **Use of a Uniform Interface**. Resources should be uniquely identifiable through a single URL, and only by using the underlying methods of the network protocol, such as DELETE, PUT, and GET with HTTP, should it be possible to manipulate a resource.
5. **Layered Syste**m. REST allows for an architecture composed of multiple layers of servers.
6. **Code on Demand (Optional)**. Most of the time a server will send back static representations of resources in the form of JSON or XML. However, when necessary, servers can send executable code to the client.

REST Request and Response

For a REST Request, it is important to know:

- The endpoint – a unique URL that contains the data objects or collection of objects
- The method – functions that perform corresponding operations which usually involve CRUD (Create, Reading, Updating and Deleting)
- The headers – fields providing instructions to the server
- The params – user specifications passed to the endpoint that affects the type of response generated
- The data (or body) – contains information you want to be sent to the server

For a REST Response, it can be data in the format of JSON or XML or any other media type. When something is not good, for example server is down, a response with an appropriate HTTP status code will be returned. We will look at more in the later chapters.

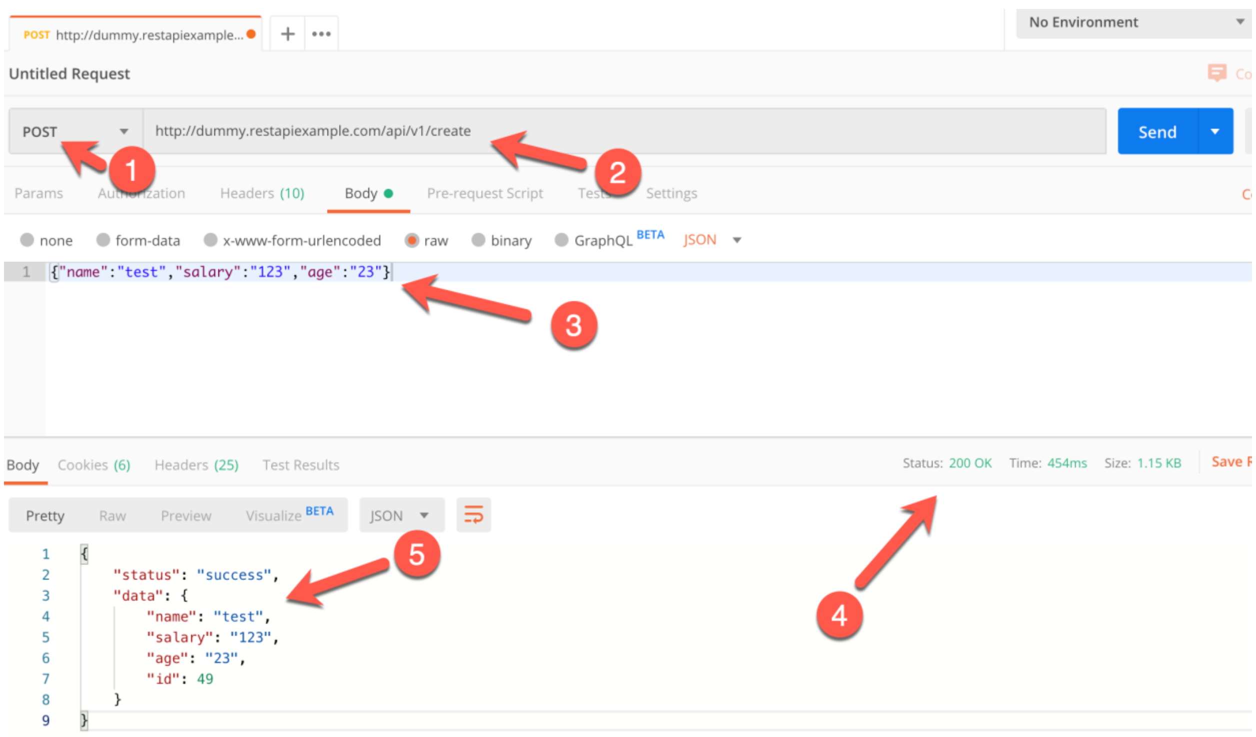
Postman and Example

Postman is a REST client tool that can be used for REST API testing. When we access a REST server by typing a URL in the browser, the method is GET by default. In order to test out different methods, headers, data body and other features of a REST service, we need to use a REST client like Postman.

The screenshot is an example of a REST request and response.

- 1. Http method set as POST
- 2. The target URL
- 3. Data body

- 4. Response status, 200 means success
- 5. Response data



POST http://dummy.restapiexample.com/api/v1/create

```
{"name":"test","salary":"123","age":"23"}
```

Status: 200 OK Time: 454ms Size: 1.15 KB

```
{
    "status": "success",
    "data": {
        "name": "test",
        "salary": "123",
        "age": "23",
        "id": 49
    }
}
```

Chapter 2 – LDAP

Chapter 2, "LDAP," provides an in-depth exploration of the Lightweight Directory Access Protocol (LDAP), a cornerstone protocol for managing and accessing directory information over networks, as standardized in RFC 4511.

This chapter underscores LDAP's pivotal role in Identity and Access Management (IAM) by enabling centralized authentication, authorization, and related functions such as user provisioning and role-based access control, making it indispensable in enterprise environments like Active Directory and OpenLDAP.

It outlines the evolution of LDAP from its inception as a simplified alternative to X.500's DAP in the early 1990s (LDAPv1), through its enhanced LDAPv2, to the widely adopted LDAPv3 of 1997, which introduced robust security features like SASL and TLS. The chapter also details LDAP's hierarchical Directory Information Tree (DIT) structure, key terminology (e.g., DN, RDN, attributes, object classes), and operational mechanisms like search requests and authentication workflows.

Additionally, it evaluates LDAP's strengths—such as its efficiency in read-heavy workloads and standardized interoperability—and its limitations, including weaker write performance and schema inflexibility.

Key Points

- **LDAP's Purpose:**
  - LDAP is a protocol for managing and accessing directory information, optimized for read-heavy workloads.
  - It is used for centralized authentication and authorization, and stores data in a hierarchical tree structure.

- **LDAP Basics:**
  - LDAP is a crucial component in IAM, providing centralized authentication and authorization.
  - Directory services, like OpenDS, can be implemented using LDAP.
  - LDAPv3 is the current prevalent version.

- **LDAP Structure:**
  - Data is organized in a Directory Information Tree (DIT) with entries identified by Distinguished Names (DNs).
  - Key terms include entries, attributes, object classes, and schemas.

- **LDAP Components:**
    - DN (Distinguished Name): Unique identifier for an entry.
    - RDN (Relative Distinguished Name): Component of a DN.
    - Attributes: Data held within an entry.
    - Object Classes: Define collections of attributes.
    - OIDs: Unique identifiers for LDAP elements.

- **LDAP Operations:**
    - DNs and RDNs define the hierarchical structure.
    - Attributes store entry data.
    - Object classes define entry types.
    - Search requests retrieve entries based on criteria.

- **LDAP Authentication and Authorization:**
    - LDAP clients interact with LDAP servers.
    - Authentication verifies user identity through binding (anonymous, simple, SASL).
    - Authorization determines access rights through role/group lookups and Access Control Lists (ACLs).

This chapter establishes LDAP as a foundational protocol in modern IAM frameworks, balancing its enduring relevance with an analysis of its constraints.

Why LDAP

LDAP is an old protocol and even if LDAPv3 came out in 1997, it is still quite some time back. It is prevalent in the industry right now, but will it be the same in the future?

One major catch is that LDAP is optimized for read-heavy workloads. LDAP directories are designed to handle a high volume of read operations efficiently. This makes them ideal for use cases where data is read frequently but updated less often, such as user authentication and authorization, an essential role of IAM.

LDAP stores data in a hierarchical tree structure, which naturally represents organizational structures (e.g., company departments, employee hierarchy). This makes it easier to manage and navigate relationships between entries.

LDAP is a standardized protocol (defined by RFCs) specifically designed for accessing and maintaining directory information. This ensures compatibility and interoperability between different LDAP implementations.

Nevertheless, there are benefits like centralized authentication and authorization, attribute-based access, replication and redundancy, etc.

Besides its benefits, there are constraints of LDAP too.

Since it's optimized for read heavy operation, it compromises the writing performance. Applications with heavy write demands may find LDAP less suitable compared to traditional databases.

LDAP schemas define the structure of directory entries and can be inflexible. Changing the schema can be complex and may require downtime or significant reconfiguration.

LDAP does not provide robust transaction support like traditional relational databases, which can be a limitation for applications requiring complex transactional integrity.

What is LDAP

Lightweight Directory Access Protocol (LDAP) is a **protocol**, a widely adopted protocol (https://datatracker.ietf.org/doc/html/rfc4511) that enables the management and access of directory information over a network, playing a critical role in IAM.

LDAP serves as a crucial component in IAM by providing centralized authentication and authorization services. When a user attempts to access a network resource, an LDAP server can verify the user's credentials and determine their permissions based on the information stored within the directory.

This centralized approach enhances security by ensuring consistent access control policies and simplifies administrative tasks. In addition to authentication and authorization, LDAP supports key IAM functions such as user provisioning and deprovisioning, password management, and role-based access control.

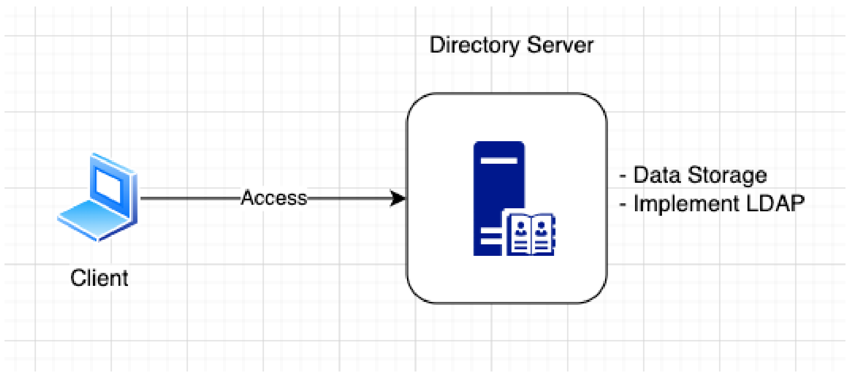
As a result, LDAP is integral to many enterprise environments, supporting systems like Microsoft's Active Directory, OpenLDAP, and various other directory services. Through its efficient and standardized method of handling directory information, LDAP plays a vital role in maintaining the integrity and accessibility of data within an organization's IAM framework.

Directory Service

In a general sense, a Directory Service is a database for **storing and maintaining information about users and resources**. This information, such as usernames, passwords, and user preferences, allows system and network administrators to control access to applications and resources.

As LDAP is a type of protocol regarding management and access of directory information over a network, a Directory Service can be implemented based on LDAP protocol. For example,

OpenDS, a directory service project started in 2005 by Sun Microsystems, is written in Java following the LDAP standard.



Some Terms
People use LDAP and Directory Service interchangeably. LDAP is the protocol and Directory Service usually implements LDAP for standard compliance.

LDAP Versions

LDAP has evolved over the years with various versions introduced to enhance its functionality and compatibility.

LDAPv1

LDAP was developed in the early 1990s by Tim Howes and his colleagues at the University of Michigan, which is used to access a directory service called X.500. LDAP is basically a simplified or lightweight version of another protocol - DAP (Directory Access Protocol) that was used to access X.500 directory service at that time. As the protocols functionality is limited, it was not widely adopted

LDAPv2

A couple of years later, LDAP version 2 was introduced as an enhanced version of LDAPv1. It provides client-server directory access capabilities and supports simple authentication as well as basic operations like search, add, delete and modify.

LDAPv3

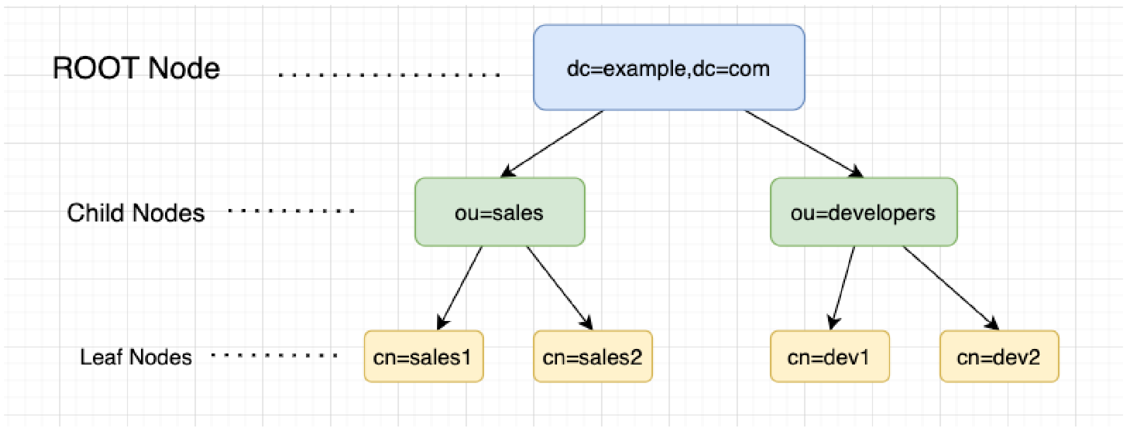
Published in 1997, the third version **LDAPv3** became popular and widely adopted. It features some major improvements over LDAPv2, such as enhanced security features supporting SASL (Simple Authentication and Security Layer) and TLS (Transport Layer Security), schema definition and manipulation, extended operations and controls, etc.

Then LDAPv3 was accepted as the internet standard for directory services and remains the latest and most prevalent version of LDAP today. In other words, whenever you see people mentioning LDAP nowadays, it is most likely LDAPv3.

LDAP Directory Structure

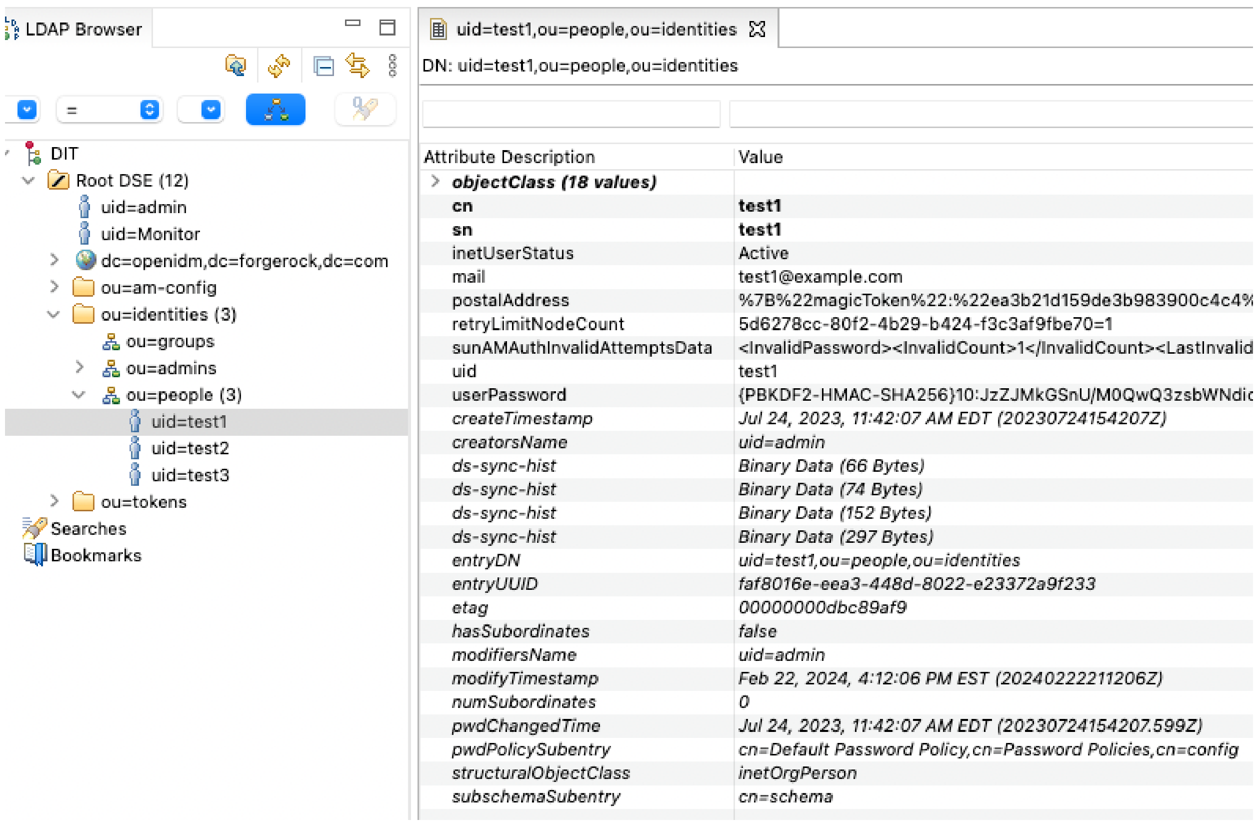
LDAP organizes information in a hierarchical tree structure called **Directory Information Tree** (**DIT**, and DIT can vary based on the software or directory service provider you use) and consists of one or more entries. Entries represent real-world objects such as organizations, users, non-user entities (e.g. printer), etc.

Here is an example of an LDAP tree structure.



The tree structure always starts with a root node (also called **root DSE**, Directory Service Entry), which usually represents the organization. It will then be followed by further levels of entries. Each node is an entry and can be identified by its DN.

The following highlighted a user entry called ‘test1’.



LDAP Browser

- DIT
  - Root DSE (12)
    - uid=admin
    - uid=Monitor
    - dc=openidm,dc=forgerock,dc=com
    - ou=am-config
    - ou=identities (3)
      - ou=groups
      - ou=admins
      - ou=people (3)
        - uid=test1
        - uid=test2
        - uid=test3
    - ou=tokens
- Searches
- Bookmarks

uid=test1,ou=people,ou=identities

DN: uid=test1,ou=people,ou=identities

Attribute Description	Value
objectClass (18 values)	
cn	**test1**
sn	**test1**
inetUserStatus	Active
mail	test1@example.com
postalAddress	%7B%22magicToken%22:%22ea3b21d159de3b983900c4c4%
retryLimitNodeCount	5d6278cc-80f2-4b29-b424-f3c3af9fbe70=1
sunAMAuthInvalidAttemptsData	<InvalidPassword><InvalidCount>1</InvalidCount><LastInvalid
uid	test1
userPassword	{PBKDF2-HMAC-SHA256}10:JzZJMkGSnU/M0QwQ3zsbWNdic
createTimestamp	*Jul 24, 2023, 11:42:07 AM EDT (20230724154207Z)*
creatorsName	*uid=admin*
ds-sync-hist	*Binary Data (66 Bytes)*
ds-sync-hist	*Binary Data (74 Bytes)*
ds-sync-hist	*Binary Data (152 Bytes)*
ds-sync-hist	*Binary Data (297 Bytes)*
entryDN	*uid=test1,ou=people,ou=identities*
entryUUID	*faf8016e-eea3-448d-8022-e23372a9f233*
etag	*00000000dbc89af9*
hasSubordinates	*false*
modifiersName	*uid=admin*
modifyTimestamp	*Feb 22, 2024, 4:12:06 PM EST (20240222211206Z)*
numSubordinates	*0*
pwdChangedTime	*Jul 24, 2023, 11:42:07 AM EDT (20230724154207.599Z)*
pwdPolicySubentry	*cn=Default Password Policy,cn=Password Policies,cn=config*
structuralObjectClass	*inetOrgPerson*
subschemaSubentry	*cn=schema*

Key Terms

Let's look at some key terms used by LDAP.

Entry

A single record in the LDAP directory. Each entry is identified by a **Distinguished Name** (DN) and contains a set of attributes.

Attributes

Properties or characteristics of an entry. Each attribute has a type (or name) and one or more values. Common attributes include ***cn*** (common name), ***sn*** (surname), ***mail*** (email address), and ***uid*** (user ID).

Object Class

A definition of a type of entry in the directory. Object classes determine the mandatory and optional attributes that an entry must or can have. For example, the ***person*** object class might require ***cn*** and ***sn*** attributes and allow ***mail*** and ***telephoneNumber*** attributes.

Object Identifiers (OIDs)

An object identifier (**OID**) is a string that is used to uniquely identify various elements in the LDAP protocol, as well as in other areas throughout computing. OIDs consist of a sequence of numbers separated by periods (e.g., “1.2.840.113556.1.4.473” is the OID that represents the server-side sort request control).

Schema

The set of rules that define the **structure and content of the directory**. The schema includes definitions of object classes, attribute types, syntax rules, and matching rules.

LDIF (LDAP Data Interchange Format)

A **standard plain text data** interchange format for representing LDAP directory content and update requests. LDIF conveys directory content as a set of records, one record for each object (or entry). It also represents update requests, such as Add, Modify, Delete, and Rename, as a set of records, one record for each update request.

Bind

An operation that authenticates a client to the LDAP server. The bind operation establishes the identity of the client and sets the security context for subsequent operations.

Search

An operation that retrieves entries from the directory based on specified criteria. The search operation can include a base DN, a search scope (base, one-level, or subtree), a filter, and a list of attributes to return.

ACL

A list that specifies permissions or access rights to directory entries and attributes.

SASL (Simple Authentication and Security Layer)

A framework for adding authentication support to connection-based protocols. LDAP can use SASL for various authentication mechanisms, providing greater security and flexibility.

LDAPS

When LDAP is added with TLS (Transport Layer Security), it adds a layer for securing communications over a computer network. LDAP can use TLS to encrypt data transmitted between the client and server, ensuring confidentiality and integrity.

Acronyms

There are acronyms like dn, cn, rdn, ou, etc used in LDAP and it’s important to know them as well.

DIT (Directory Information Tree)
The hierarchical structure in which directory entries are organized. The tree starts from a single root and branches out into various organizational units and entries.

DN (Distinguished Name)
A unique identifier for an entry in the directory. The DN specifies the entry's position in the DIT and is composed of a series of Relative Distinguished Names (RDNs). For example, uid=test1,ou=people,ou=identities

RDN (Relative Distinguished Name)
A component of a DN that identifies an entry relative to its parent entry. An RDN typically consists of one or more attribute-value pairs.

CN (Common Name)
An attribute type typically used to represent the name of a person or object. Example: cn=John Doe.

UID (User ID)
An attribute type representing a unique identifier for a user. Example: uid=jdoe.

DC (Domain Component)
An attribute type used to represent components of a domain name. Example: dc=example,dc=com.

OU (Organizational Unit)
An attribute type used to represent a division within an organization. Example: ou=Marketing.

O (Organization)
An attribute type representing an organization. Example: o=Example Corp.

C (Country)
An attribute type representing a country. Example: c=US.

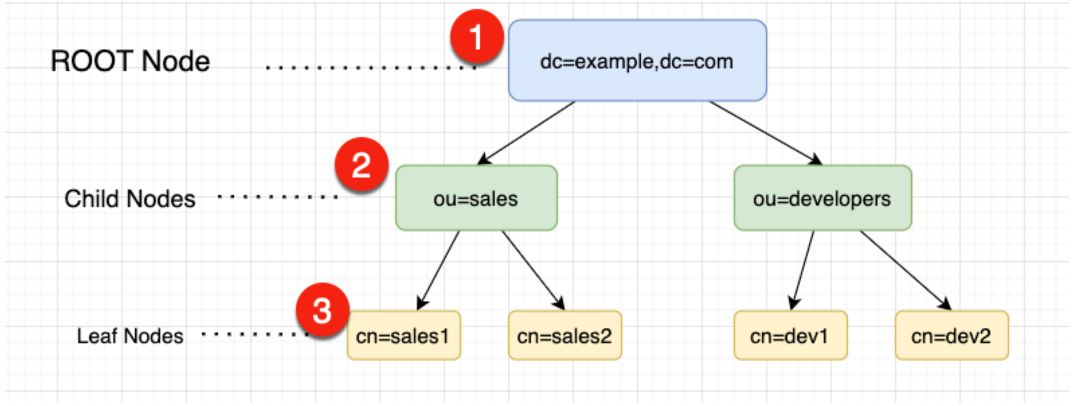
L (Locality)
An attribute type representing a locality, such as a city or town. Example: l=San Francisco.

How LDAP Works

DN and RDN

To identify an entry in the LDAP tree, we use what is called the Distinguished Name (DN). The DN is globally unique within a directory and represents the exact position of an object in the tree. It is constructed by combining the name of the entry with the names of its parent nodes, continuing all the way up to the root. In other words, the DN is the full path of the object within the tree.

Let's revisit the tree diagram as shown below and take a leaf node '*sales1*' (also an entry) as an example.



To traverse from root entry (*dc=example,dc=com*) to the target entry (*cn=sales1*), we will first hit '*dc=example,dc=com*', then '*ou=sales*', and lastly '*cn=sales1*'.

Here '*dc=example,dc=com*', '*ou=sales*', and '*cn=sales1*' are RDNs (Relative Distinguished Name).

To build the DN of '*sales1*', we simply reverse the order of traversed entries and concatenate them with commas (no space in between), which becomes '***cn=sales1,ou=sales,dc=example,dc=com***'. This will be DN of that '*sales1*' entry and it's **globally unique**, because there is only one way to traverse from root DSE to the target entry based on the LDAP tree structure.

Each RDN in a DN represents a level in the hierarchy and if you remove an RDN from a DN (starting from left side), you get the DN of the entry considered parent of the former DN.

For example, '*ou=sales,dc=example,dc=com*' is the parent entry of '*cn=sales1,ou=sales,dc=example,dc=com*'; vice versa, '*cn=sales1,ou=sales,dc=example,dc=com*' is the child entry of '*ou=sales,dc=example,dc=com*'.

Attributes
Attributes hold the data for an entry. Each attribute consists of an attribute type, zero or more attribute options, and a set of values that represent the actual data. Attribute types are schema elements that define how attributes should be managed by LDAP clients and servers.

Every attribute type must have an object identifier (OID) and may have one or more names that can be used to reference attributes of that type. They also require an attribute syntax, which specifies the type of data that can be stored in attributes of that type, and a set of matching rules, which determine how comparisons should be made against attribute values.

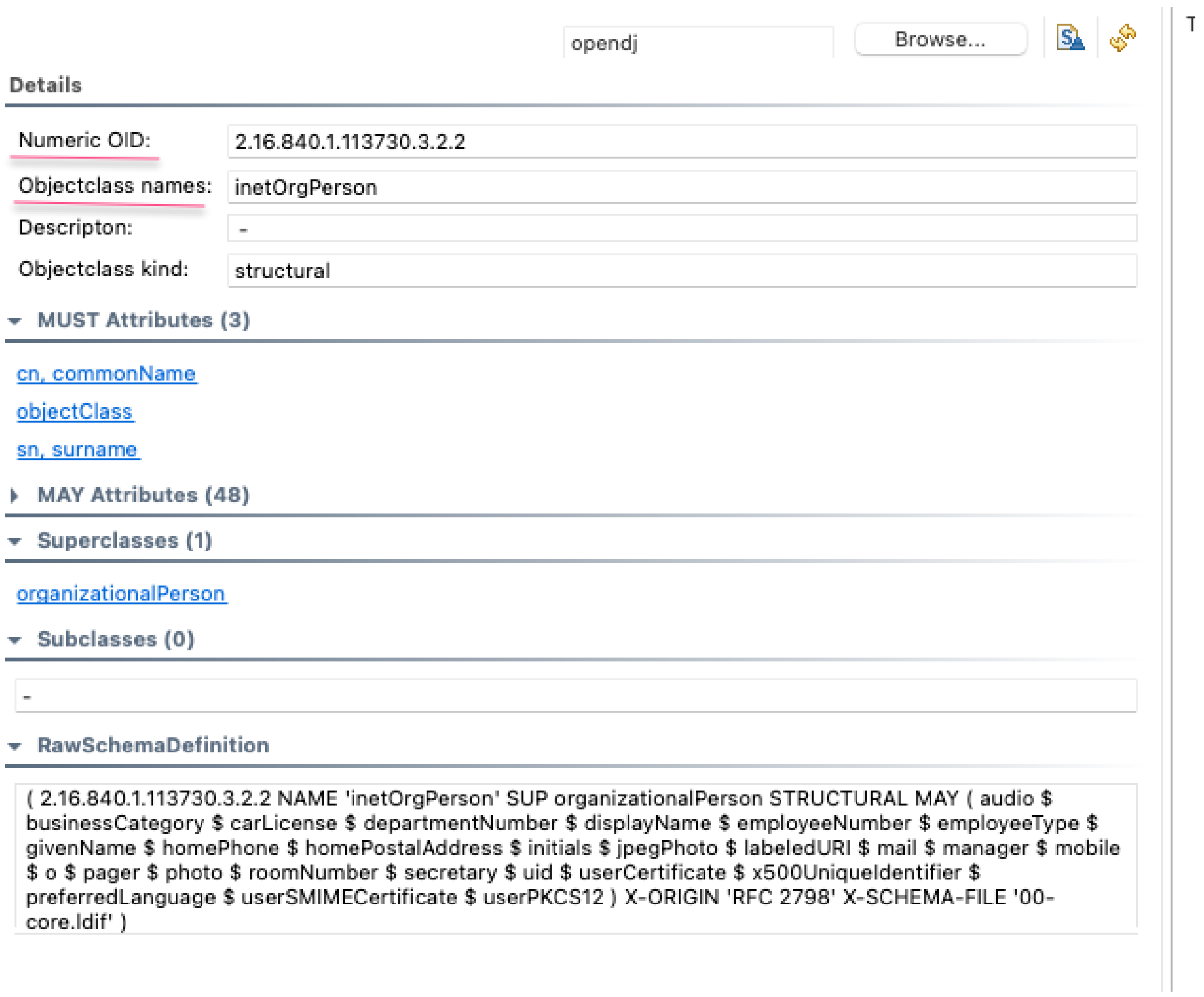
Additionally, attribute types may indicate whether an attribute can have multiple values within the same entry and whether the attribute is intended to store user data (a user attribute) or to be used for the server's operation (an operational attribute).

Operational attributes are typically used for **configuration and state information**. Attribute options are less commonly used but can provide metadata about an attribute. For instance, attribute options can be used to offer different versions of a value in various languages.

Object Classes
Object classes are schema elements that define **collections of attribute types** associated with a particular type of object, process, or other entity. Each entry has a structural object class that specifies the kind of object it represents, such as information about a person, a group, a device, or a service. Additionally, entries may have zero or more auxiliary object classes that provide extra characteristics.

Like attribute types, object classes must have an object identifier (OID) and can also have one or more names. Object classes may specify a set of required attribute types, meaning any entry with that object class must include those attributes. They may also list a set of optional attribute types, which entries with that object class may include at their discretion.

Below is an example of an object class named ‘*inetOrgPerson*’.


opendj

Browse...

Details

Numeric OID:	2.16.840.1.113730.3.2.2
Objectclass names:	inetOrgPerson
Descripton:	-
Objectclass kind:	structural

MUST Attributes (3)

cn, commonName
objectClass
sn, surname

MAY Attributes (48)

Superclasses (1)

organizationalPerson

Subclasses (0)

-

RawSchemaDefinition

```
( 2.16.840.1.113730.3.2.2 NAME 'inetOrgPerson' SUP organizationalPerson STRUCTURAL MAY ( audio $
businessCategory $ carLicense $ departmentNumber $ displayName $ employeeNumber $ employeeType $
givenName $ homePhone $ homePostalAddress $ initials $ jpegPhoto $ labeledURI $ mail $ manager $ mobile
$ o $ pager $ photo $ roomNumber $ secretary $ uid $ userCertificate $ x500UniqueIdentifier $
preferredLanguage $ userSMIMECertificate $ userPKCS12 ) X-ORIGIN 'RFC 2798' X-SCHEMA-FILE '00-
core.ldif' )
```

Object Identifiers (OIDs)

An object identifier (OID) is a **unique string** used to identify various elements in the LDAP protocol and other areas of computing.

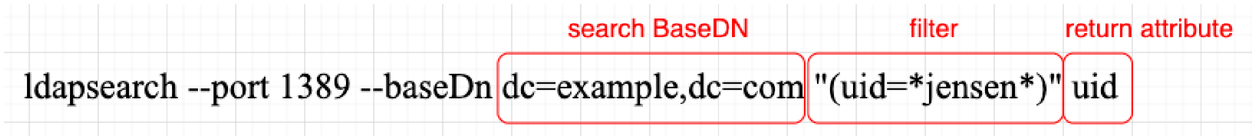
OIDs consist of a sequence of numbers separated by periods (e.g., "1.2.840.113556.1.4.805" is the OID for the subtree delete control). In LDAP, OIDs identify schema elements (such as attribute types, object classes, syntaxes, and matching rules), controls, and extended requests and responses.

For schema elements, user-friendly names may also be used in place of OIDs.

Search Request

A search request in LDAP usually consists of *Search Base DN*, *Scope*, *Search Filter* and *Attributes to Return*. Additional search parameters include size limit, time limit, etc.

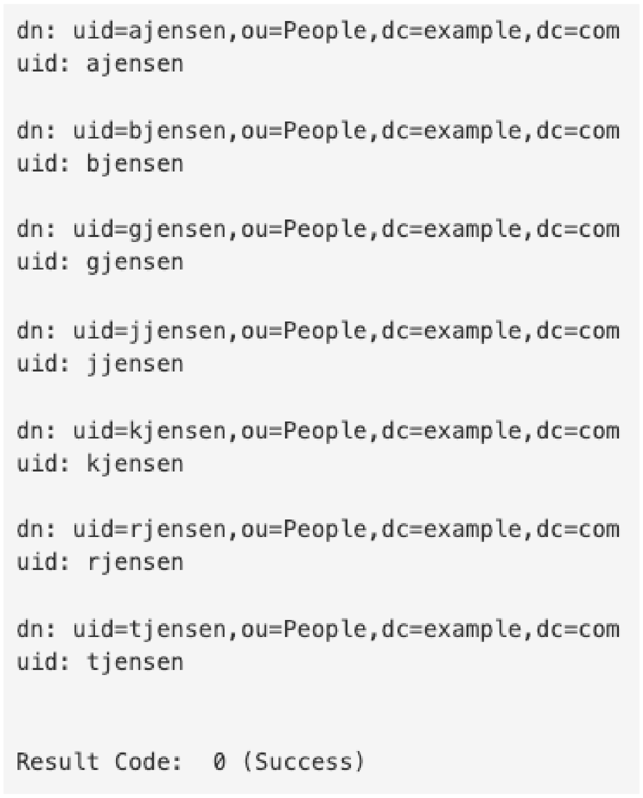
For example, here is an LDAP search request command



```
ldapsearch --port 1389 --baseDn dc=example,dc=com "(uid=*jensen*)" uid
```

The scope is omitted from the above command and can default to 'subtree' depending on the LDAP provider.

Assuming the scope is *'subtree'*, meaning all entries under the specified BaseDN, this search will return all entries under BaseDN '*dc=example,dc=com*' with uid value containing *'jensen'*, where only uid will be displayed for each entry (DN included by default here).



```
dn: uid=ajensen,ou=People,dc=example,dc=com
uid: ajensen

dn: uid=bjensen,ou=People,dc=example,dc=com
uid: bjensen

dn: uid=gjensen,ou=People,dc=example,dc=com
uid: gjensen

dn: uid=jjensen,ou=People,dc=example,dc=com
uid: jjensen

dn: uid=kjensen,ou=People,dc=example,dc=com
uid: kjensen

dn: uid=rjensen,ou=People,dc=example,dc=com
uid: rjensen

dn: uid=tjensen,ou=People,dc=example,dc=com
uid: tjensen


Result Code:  0 (Success)
```

LDAP Authentication and Authorization

LDAP Client

An LDAP client is a software application or utility that interacts with an LDAP server to perform operations on the directory service. These operations typically include searching, adding, modifying, and deleting directory entries. The LDAP client uses the LDAP protocol to communicate with the LDAP server.

LDAP Authentication

LDAP authentication involves verifying a user's identity against an LDAP directory.

Typical steps include:

1. Connection Establishment:
    - The client establishes a connection to the LDAP server over LDAP (port **389** by default) or LDAPS (port **636** by default)

2. Binding:
    - **Anonymous Binding**: Some LDAP servers allow anonymous binding, where no authentication is required. This is generally used for querying public information.
    - **Simple Binding**: The client sends a distinguished name (DN) and a password in plain text. This method is not secure unless used with TLS/SSL.
    - **SASL Binding**: Simple Authentication and Security Layer (SASL) allows for more secure and flexible authentication mechanisms, such as Kerberos, Digest-MD5, or GSSAPI.

3. Credential Validating:
    - The client searches the LDAP directory for the user's DN based on the provided credentials (e.g., username).
    - If the user is found, the LDAP server attempts to bind with the user's DN and password.
    - If the credentials are correct, the LDAP server grants access, and the client is considered authenticated.

LDAP Authorization

LDAP authorization determines what an authenticated user is allowed to do.

This typically involves the following steps:

1. User Role or Group Membership Lookup:
    - Once authenticated, the LDAP server or the application queries the directory to determine the user's roles or group memberships. This information is often stored

in attributes like memberOf for group membership.

2. Access Control:
    - **Access Control Lists** (ACLs): LDAP directories often use ACLs to define permissions. ACLs specify which users or groups have access to specific directory entries or attributes.
    - **Role-Based Access Control** (RBAC): Applications can implement RBAC by assigning permissions based on the user's role(s) retrieved from the LDAP directory.

3. Policy Enforcement:
    - The application or service enforces access control policies based on the retrieved roles or group memberships. This determines what resources or operations the authenticated user can access.

Chapter 3 – Database

In the IT industry, database concepts and knowledge play a pivotal role in ensuring efficient and effective management of data. Databases serve as the backbone of virtually every information system, allowing organizations to store, organize, retrieve, and analyze vast amounts of data. Understanding database concepts is essential for professionals working with data, as it enables them to design, develop, and maintain robust and scalable database systems.

Moreover, a solid understanding of database concepts is critical for ensuring data integrity, security, and privacy, as it allows IT professionals to implement proper access controls, backup and recovery mechanisms, and data normalization techniques. In a world driven by data-driven decision-making, possessing knowledge of database concepts equips individuals with the expertise to harness the power of information and contribute to the growth and success of organizations across various industries.

This chapter highlights why understanding database concepts is essential for IT professionals to design scalable, secure, and robust systems while addressing challenges like data integrity, security, and concurrency. I

t explores the necessity of databases as data volumes grow, introduces Database Management Systems (DBMS) like Oracle and MySQL, and categorizes databases into relational (SQL) and non-relational (NoSQL) types. The chapter provides a detailed examination of relational database components—tables, rows, columns, and keys—alongside relationships, SQL syntax, normalization, and design processes. It concludes with a comparison of SQL and NoSQL databases, offering guidance on their contextual applications.

Key Points

- **Database Necessity:** Databases address data management challenges like size, accuracy, security, redundancy, concurrency, and transactions.

- **DBMS:** Database Management Systems (e.g., Oracle, MySQL, MongoDB) are software that manage databases.

- **Relational Databases (SQL):**

    - Data is organized in tables with rows and columns.
    - Database keys (primary, foreign, candidate, super) establish relationships and ensure data integrity.
    - Table relationships: One-to-one, one-to-many, many-to-many.
    - SQL (Structured Query Language) is used for data manipulation and querying.

- Database design involves requirement analysis, logical design (ER diagrams, normalization), physical design, implementation, and maintenance.
- Normalization reduces data redundancy and anomalies (1NF, 2NF, 3NF, BCNF).

- **Non-Relational Databases (NoSQL):**
    - Flexible schemas for unstructured or semi-structured data.
    - Types: Key-value, graph, document, wide-column.
    - Scalable and adaptable to evolving data requirements.

- **SQL vs. NoSQL:**
    - SQL: Structured data, predefined schemas, vertical scalability, complex queries, ACID compliance.
    - NoSQL: Unstructured data, dynamic schemas, horizontal scalability, fast data operations.
    - Hybrid systems can leverage both SQL and NoSQL for optimal performance.

Why do we need a database?

When you have some data, which could be anything like customers, products, employees…, and you want to store this data somewhere. This data could be in text format, numbers, dates, documents, images, audios, videos etc. This is easy to manage when they are small. Yet, this suddenly starts to become a problem when the amount of data is growing fast. Let's consider some potential perspectives here.

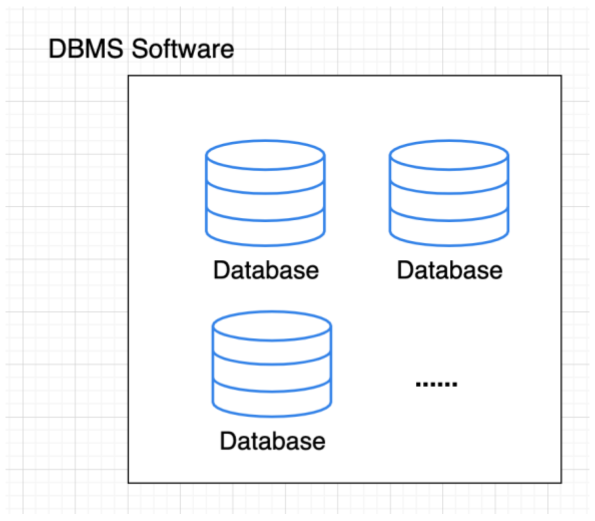
- Size – Large amount of data, hundreds of thousands of data, and still growing
- Accuracy – Are the data being inputted correct?
- Security – Different users with different roles should be granted different levels of privileges
- Redundancy – redundancy (multiple copies of data) provides backup in case something bad happens. What if you have conflicts among the copies of your data.
- Concurrency – how do you handle multiple users writing to the same data?
- Transaction – When a workflow is halfway finished and you want to cancel the workflow due to some reason, how do you handle the part which has already taken place?

The list could go longer and the real-world scenario is very complicated.

Database Management System (DBMS)

We often refer to our database as Oracle, MySQL, SQL Server, MongoDB etc. Actually, they are Database Management Systems (DBMS).

We can simply think DBMS is the software that would be installed on your personal computer or server, which is then used to manage one or more databases.



Database Types

There are different ways to categorize databases, but in general, databases can be divided into two types: **Relational** and **Non-relational**. They are also called SQL Database and Nosql Database.

We will first look at Relational Database, which is more traditional.

Tables

Table is the basic building block of a database and a database can be considered as a collection of tables. It is where you put your data, define data type, and relationship with other tables.

Rows & Columns

A table consists of rows, and columns.

- Row – represent entries of data
- Column – defines what kind of data it is specifically

For example, in the below table, it describes the temperature of different cities in the US. Each row represents an entry of the cities and each column defines the meaning of the values.

city	state	high	low
Phoenix	Arizona	105	90
Tucson	Arizona	101	92
Flagstaff	Arizona	88	69
San Diego	California	77	60
Albuquerque	New Mexico	80	72

Database Keys

A database key is an attribute (column) or set of attributes which allows you to establish a relationship between and identify the relation between tables.

In a real-world application, a table could contain thousands of records. Moreover, the records could be duplicated. Keys ensure that you can uniquely identify a table record despite these challenges. It helps to enforce identity and integrity in the relationship.

There are various keys in database system:

- Primary Key
- Foreign Key
- Super Key
- Candidate Key
- Alternate Key
- Compound Key
- Composite Key
- Surrogate Key

We'll take a look at these – **Primary Key**, **Foreign Key**, **Super Key** and **Candidate Key**.

Primary Key

For a table, you need something unique to identify a row and that's called "primary key". A column called "name" may have duplicate values since people can have the same name. This will create a problem when we refer to entries with names, as it will return more than one record which will create ambiguity.

- **Primary Key** – a minimal set of columns in a table that uniquely identifies rows in that table

To deal with that issue, there is often a column called "ID" in the table, which is used as the primary key for that table. The primary key is usually generated by DBMS.

Another note is that **a primary key does NOT have to be one column**. It can be the combination of several columns, as long as the combination of them uniquely identifies each row in the table and that combination is minimal.

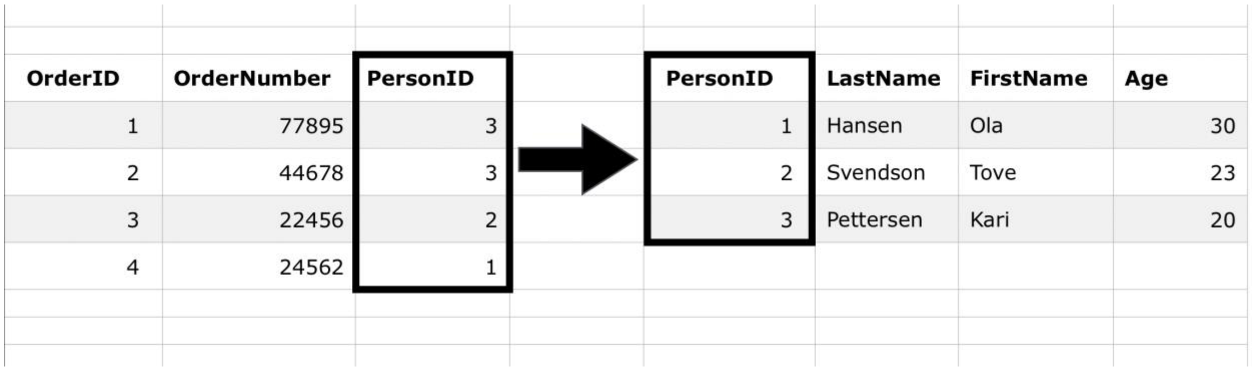
Using the combination of columns as the primary key might not be that often, since we can always add an ID column.

Foreign Key

- **Foreign Key** - a column in one table that refers to a Primary Key in another table.

One table can have more than one foreign key.

For example, in the following two tables, the left table has "OrderID" as primary key and the right table has "PersonID" as primary key. Furthermore, the "PersonID" is also a column in the left table. In this case, we say "PersonID" is a foreign key in the left table, since it is not the primary key for that table but is a primary key for another table.



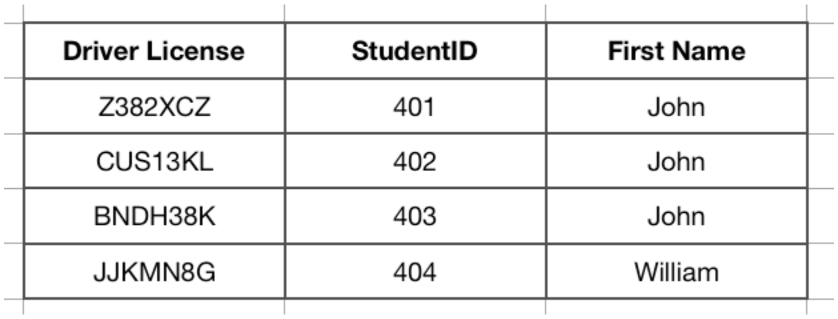
OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

Super Key

- **Super key** – a combination of one or more columns, which can uniquely identify a row in a table

For example, let's look at the below table:



Driver License	StudentID	First Name
Z382XCZ	401	John
CUS13KL	402	John
BNDH38K	403	John
JJKMN8G	404	William

Assume **DL** – Driver License, **SID** – StudentID, **FN** – First Name, the super keys are:

- {DL}
- {SID}
- {DL, SID}
- {DL, FN}
- {SID, FN}
- {DL, SID, FN}

Because all of them can uniquely identify each row of the table.

However, this is NOT super key:
- {FN}

Because First Name alone doesn't uniquely identify each row of the table.

Candidate Key

- Candidate Key – A super key with no redundant column.

We need to understand what redundant means here. It basically means minimal super key. Let's take the above super key and examine each of them here.

- {DL}, Not redundant
- {SID}, Not redundant
- {DL, SID}, Redundant because either DL or SID can uniquely identify each row
- {DL, FN}, Redundant because FN is not needed and just DL is enough to identify each row
- {SID, FN}, Redundant because FN is not needed
- {DL, SID, FN}, Redundant

In this case, the candidate keys are:
- {DL}

- {SID}

A primary key can be selected from candidate keys and decisions can be made by Database Admin.

Table Relationship

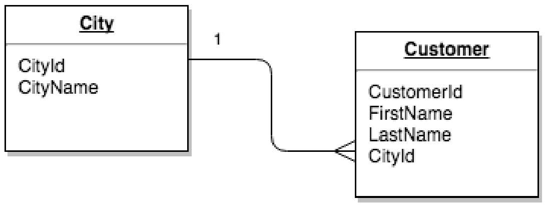
Tables don't just exist in the database separately. A lot of times, they are connected and the relationship between each table plays a vital role in a database. For example, establishing consistent relationships between database tables helps ensure data integrity as well as contributing to database normalization.

There are 3 types of relationships in a relational database:

- One-to-One
- One-to-Many (or Many-to-One)
- Many-to-Many

One-to-Many

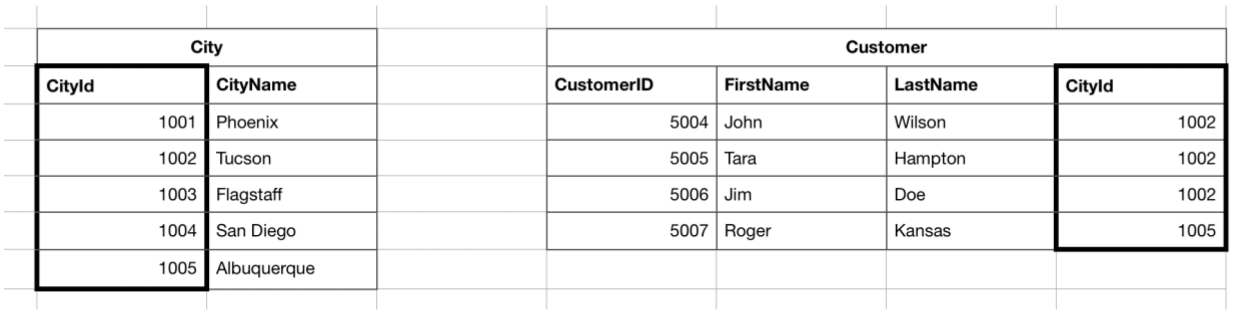
Let's look at One-to-Many first, as this is most common. In this type of relationship, a row in table A can have many matching rows in table B, but a row in table B can have one matching row in table A, as illustrated in the below **ER Diagram** (Entity-Relationship Diagram).



What is ER Diagram?
An entity relationship diagram (ERD) shows the relationships of entity sets stored in a database. One entity usually corresponds one table.

By defining the entities, their attributes, and showing the relationships between them, an ER Diagram illustrates the logical structure of databases.

Below table could be the two tables (entities) in the database.



City	
CityId	CityName
1001	Phoenix
1002	Tucson
1003	Flagstaff
1004	San Diego
1005	Albuquerque

Customer			
CustomerID	FirstName	LastName	CityId
5004	John	Wilson	1002
5005	Tara	Hampton	1002
5006	Jim	Doe	1002
5007	Roger	Kansas	1005

In the “City” table, “CityId” is its primary key. In the “Customer” table, “CustomerID” is its primary key and “CityID” is its foreign key, as it is referring to another primary key in the “City” table.

Since the “CityID” is uniquely identifying an entry in the “City” table, we can find the referenced city information for each of the entries in “Customer” table.

The interesting thing is that we see there are multiple customers with the same city record. That makes sense because we can have several different customers in one city. In other words, **One city can have Multiple Customers**. This is looking from “City” to “Customer”.

Now, let’s look from “Customer” to “City”. Can one customer have multiple cities? That is, can one customer reside in multiple cities? This is becoming even more interesting, as we can simply think that a customer as a person can travel from one city to another.

From this perspective, one customer can correspond to multiple cities. However, this may not fit the business context. The business context indicates that one customer will reside in one city and if the customer moves to another city, we can change the value to another city. In this sense, **One customer will correspond to One city**.

We look at it from both sides, one is One-to-Many and the other is One-to-One. Hence, the relationship between the two entities is **One-to-Many** (or Many-to-One).

Now, what if both sides are One-to-One? Then, the relationship is **One-to-One**. What if both sides are Many-to-Many? Then, the relationship is **Many-to-Many**.

The important thing here is that you can always justify a relationship by your own perspective, but a lot of times, we should find the best case fitting the business context. Even more, sometimes, there is not only proper design for the entity relationships. They are all good as long as they are “reasonable”. By “reasonable”, I mean it follows the principles and best practices for database design.

One-to-One

One-to-One relationships are not very common but are still valid.

For example, let’s consider driver and license. To verify that, we look from “driver” to “license” and we know that one specific driver corresponds to (own) one license; from “license” to “driver”, one specific license corresponds to (belong) to one driver. This is a One-to-One relationship.

Be aware that we are referring to a specific driver and that specific license. Although there are many drivers and many licenses out there, the relationship between them is One-to-One.

Many-to-Many

Many-to-Many relationships are more common compared to One-to-One.

For example, let’s consider an online bookstore, where buyers buy books from that store. First, we look from “buyer” to “book” and we know that one specific buyer can buy multiple books. Then, we look from “book” to “buyer”, one specific book can be bought by multiple different buyers. The relationship between “buyer” and “book” is Many-to-Many.

The interesting thing again comes with context. When we say “specific” book, are we physically referring to that copy buyer A has bought? Or, are we referring to the book considered as a product with the same title? In this business context, we would be better off considering it as the latter case.

SQL (Structured Query Language)

SQL is the standard language for dealing with Relational Databases. SQL can be used to insert, search, update and delete database records. SQL can do lots of other operations including optimizing and maintenance of databases.

SQL Example

```
SELECT * FROM Members WHERE Age > 30
```

SQL syntaxes used in different databases are similar to each other. However, a specific database will still have its own ‘Dialect’ when you are writing for that.

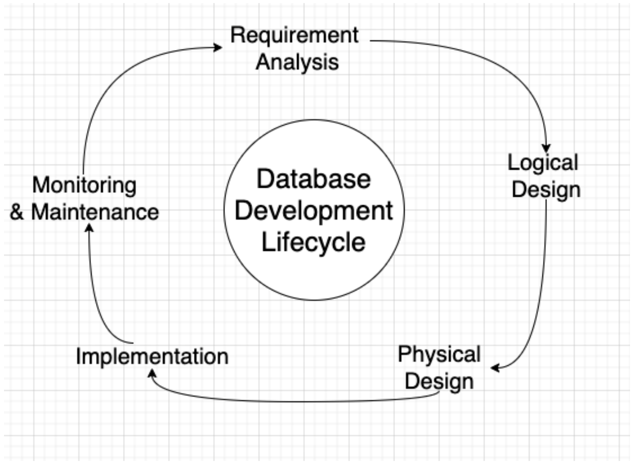
Why divide tables?
We know a foreign key is a column from one table acting as primary key in another table. Have you thought about why we divide the table into two tables?

Just think about it. Jamming data in one table will make the table larger and thus inflexible. Also, it would be more difficult to maintain. For example, if we extract repeated values from one table and put it into another table. Then we simply use IDs from other table instead of real values. In this case, when we need to update the value, we change need to change the value in the reference table; otherwise, we need to update every field that has this value.

On the other hand, dividing tables creates more tables and thus the complexity of relationships in the database. A well-designed database usually finds a balance between the two sides of a coin.

Database Design Flow

There are generally 5 stages to design a database.



Database Design Steps:

1. **Requirement Analysis** – first and most important stage. It involves assessing the informational needs of an organization so that a database can be designed to meet the needs

2. **Logical Design** – this involves 3 sub-steps
    a. Conceptual model – description of the structure of a database
    b. ER (Entity Relationship) Diagram – illustrate the entities and their relationships among each other

   c. Normalization – the process that organizes tables in a manner that reduces redundancy and dependency of data

3. **Physical Design** – make decisions about database environments, application environments, etc. The result of this step is a physical design specification.

4. **Implementation** – implement the proposed database, activities involve database creation, table creation, load data into tables etc.

5. **Monitor & Maintenance** – monitor and evaluate the database is running as expected and update it as needed

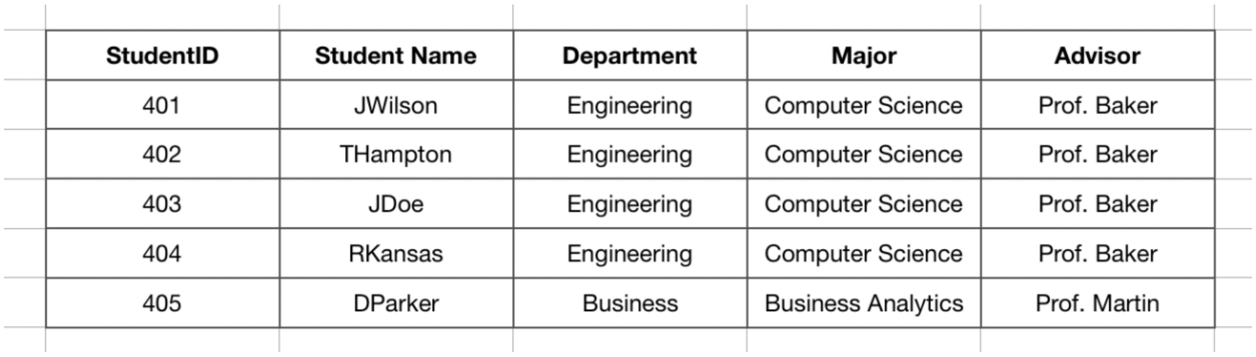
Database Normalization

Normalization is a process to decompose tables to eliminate data redundancy as well as undesirable characteristics such as insertion, update and deletion anomalies. It involves 4 major steps.

Problem Before Normalization

Let's take a look at an example.

We have a table here with student information. As we can see, Department and Major values are repeated and this repetition will probably happen for other students as well. This is **Data Redundancy**. We might just have another Department table and Major table and reference their values here.



StudentID	Student Name	Department	Major	Advisor
401	JWilson	Engineering	Computer Science	Prof. Baker
402	THampton	Engineering	Computer Science	Prof. Baker
403	JDoe	Engineering	Computer Science	Prof. Baker
404	RKansas	Engineering	Computer Science	Prof. Baker
405	DParker	Business	Business Analytics	Prof. Martin

Anomalies - Insertion, Updating and Deleting

There are different types of anomalies which can occur in referencing and referenced relation:

- **Insertion Anomaly** - If we have to insert 100 new students of the same department, then the department information will be repeated for all those 100 students.
- **Updating Anomaly** – Suppose professor Baker is replaced with another one, we need to update for all those records
- **Deletion Anomaly** – If we consider Student and Department as two different information and we delete the last record, the Department information will be lost as well

Normalization Rule

The process of database normalization consists of following several normalization rules:

- First Normal Form
- Second Normal Form
- Third Normal Form
- BCNF

The number of forms can still be developed. Yet, in many practical cases, normalization achieves its best in 3rd Normal Form.

First Normal Form (1NF)

First normal form requires:

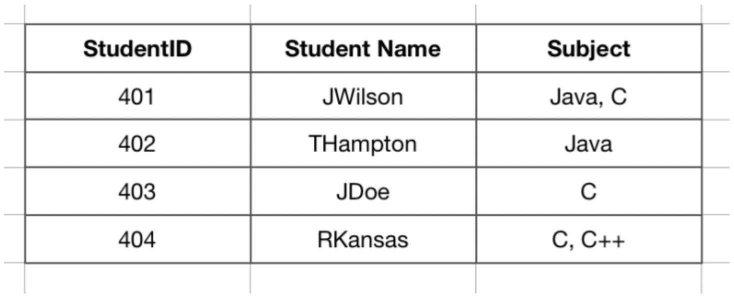
1. Single (atomic) valued attributes/columns
2. Values in a column should be the same type
3. All columns should have unique names

First rule basically means every value in a table cell should be a single (atomic) value. That is, there should not be multiple values.

Second rule requires values in the same column should be the same kind of type, which is kind of intuitive.

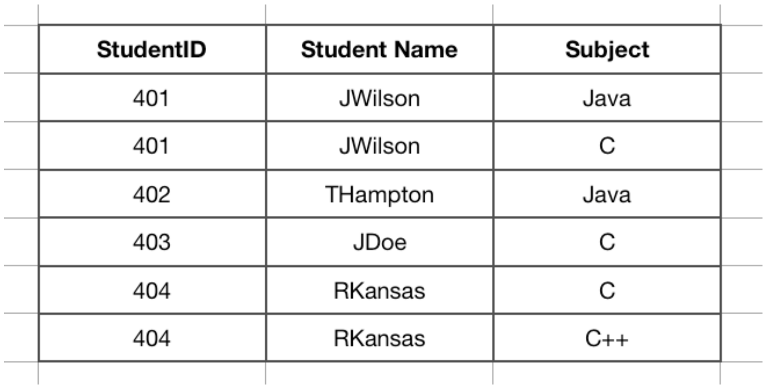
Third rule expects each column in a table should be unique. That is, no duplicate column names.

For example, in the below table, the second and third rules are satisfied but the first rule is violated, that is there are multiple values in some cells.



StudentID	Student Name	Subject
401	JWilson	Java, C
402	THampton	Java
403	JDoe	C
404	RKansas	C, C++

To fix that issue, we can break the cell into atomic values like this.



StudentID	Student Name	Subject
401	JWilson	Java
401	JWilson	C
402	THampton	Java
403	JDoe	C
404	RKansas	C
404	RKansas	C++

Let’s continue to look at the next form.

Second Normal Form (2NF)

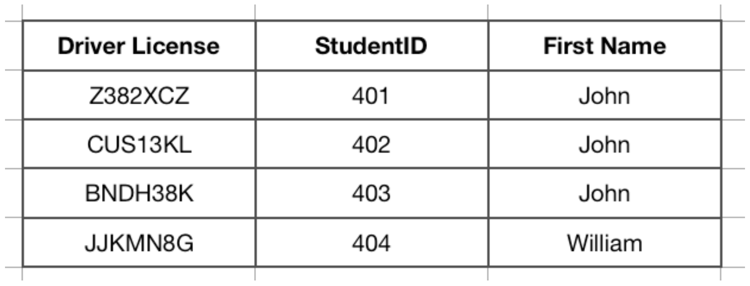
Second Normal Form requires:

1. First Normal form should be satisfied
2. No Partial Dependency

Let’s first look at what **Prime Attribute** and **Non-prime Attribute** means. Here, attribute is equivalent to column.

- **Prime Attribute** – Any attribute which is used to build candidate keys. That is to say, the building blocks of candidate keys.
- **Non-prime Attribute** – Attributes that are NOT Prime Attributes.

Consider this table again mentioned in Super Key section:



Driver License	StudentID	First Name
Z382XCZ	401	John
CUS13KL	402	John
BNDH38K	403	John
JJKMN8G	404	William

From the previous Candidate Key section, we know the candidate keys are: DL and SID. Since DL and SID are building blocks of candidate keys, DL and SID are prime attributes. Hence, FN is a non-prime attribute.

Now, for the definition of Partial Dependency:

- **Partial Dependency** – A non-prime attribute is dependent on part of the candidate keys.

Simply put, if {A, B} are candidate keys and {C} is a non-prime attribute. In this case, if {A, B} -> {C} and also {A} -> {C}, then {C} is partially dependent on {A, B}.

Looking at the previous table, we know that {DL, SID} is the candidate key and {FN} is a non-prime attribute. {DL, SID} can obviously determine {FN} for each row. Furthermore, {SID} can determine {FN} for each row as well. Hence, {FN} is partially dependent on {DL, SID}. In fact, {DL} can determine {FN} too.

On the other hand, a dependency, which isn't partial, is full. In other words, if we don't have {A} -> {C} or {B} -> {C}, that is to say, only {A, B} -> {C}, then we say {C} is fully dependent on {A, B}.

Why {SID} determines {FN}?
When we say {SID} can determine {FN}, that is because we know from a business perspective, every student in a school system will be assigned a unique StudentID and that StudentID alone can identify a student and hence his first name.

The point here is again, database design largely depends on the scenario and business context. A thorough analysis on the business situation will be paramount to the success of database design. A well-executed database design flow will effectively result in a proper delivery.

To fix the partial dependency issue in the previous case, we can remove the "duplicated" column and put it in a separate table. The following is one option.

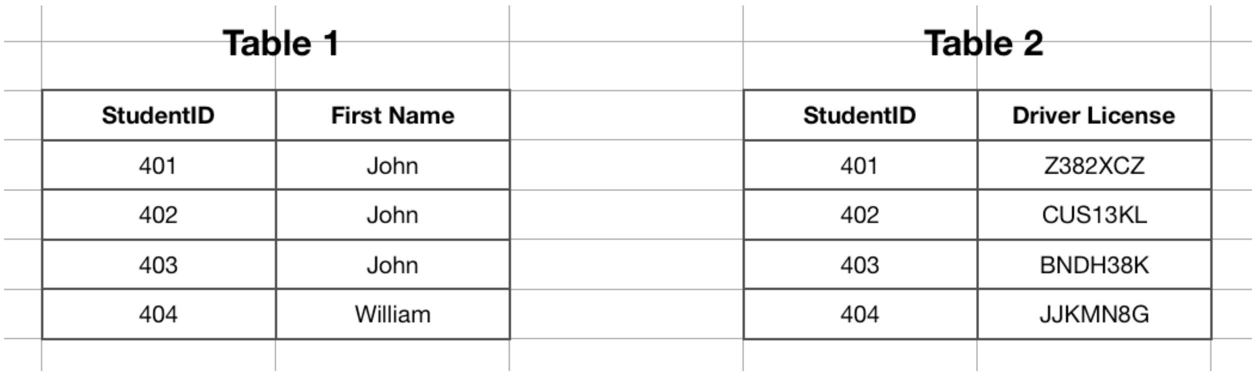


Table 1

StudentID	First Name
401	John
402	John
403	John
404	William

Table 2

StudentID	Driver License
401	Z382XCZ
402	CUS13KL
403	BNDH38K
404	JJKMN8G

Let's look at the Third Normal Form.

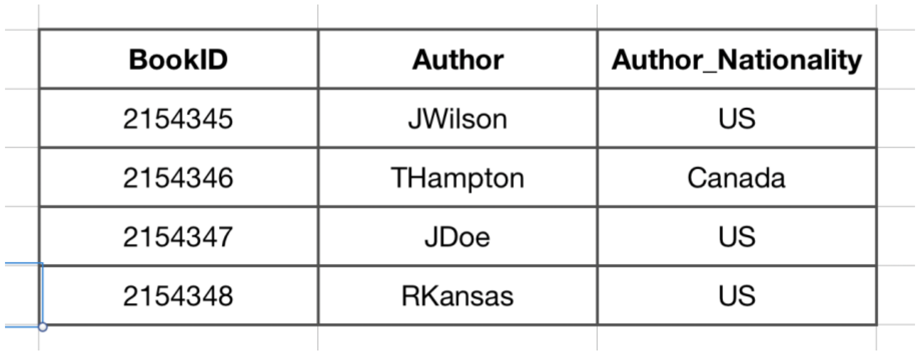
Third Normal Form (3NF)

Third Normal Form requires:

1. Second Normal form should be satisfied
2. No Transitive Dependency

A transitive dependency can be seen as: A -> B and B -> C; therefore A -> C.

A transitive dependency happens with at least three items.
An example would be



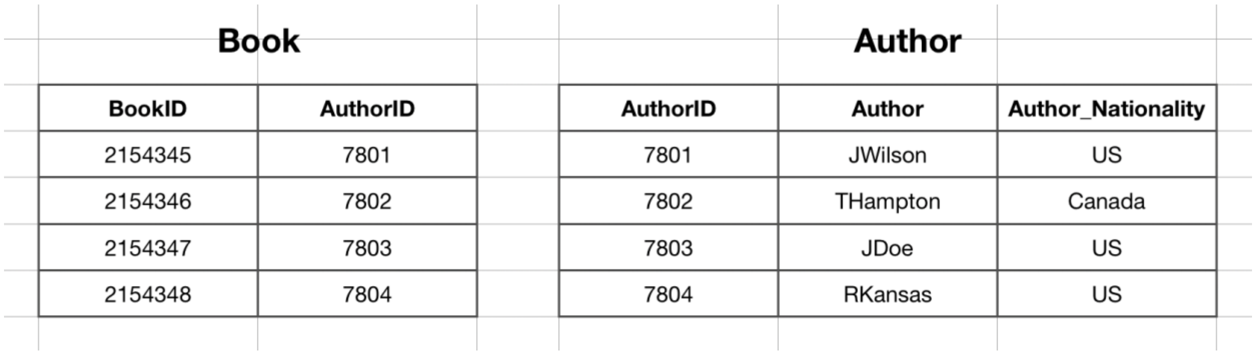
BookID	Author	Author_Nationality
2154345	JWilson	US
2154346	THampton	Canada
2154347	JDoe	US
2154348	RKansas	US

Here BID (BookID) doesn't determine AN (Author_Nationality) directly. However, BID determines Author and Author determines AN. This is a transitive dependency.

Transitive dependencies are BAD database design because it can contribute to data anomalies and inconsistencies, for example:

- If you delete second row, you would delete “THampton” and “Canada” from database, which is deletion anomaly
- You can’t add a new author to database unless you also add a book
- If “JDoe” changes Nationality, you would need to change every place where it appears
- You can’t delete an author such as “RKansas” without deleting the BookID “2154348”

To fix this, we can remove AN and create it in a separate table.



Book

BookID	AuthorID
2154345	7801
2154346	7802
2154347	7803
2154348	7804

Author

AuthorID	Author	Author_Nationality
7801	JWilson	US
7802	THampton	Canada
7803	JDoe	US
7804	RKansas	US

Now we have two tables, “Book” and “Author”. AuthorID in the Book table is a foreign key, as it is the primary key in the Author table.

This will provide better data flexibility and integrity. For example:

- If you delete “2154348” from Book table, the author information with AuthorID “7804” won’t be lost
- If you want to update an author’s information, you can just find the author in the Author table and update it. You don’t need to search for every place
- You can add a new author as you want in Author table and reference it later

Let’s look at the next form.

Boyce-Codd Normal Form (BCNF)

BCNF Normal Form is sometimes referred as 3.5NF, and it requires:

1. Third Normal form should be satisfied
2. For **every** dependency A -> B, A should be the super key of the table

Simply put, the second point means, for a dependency A -> B, if B is a **prime attribute**, A must be a **prime Attribute** too.

It’s rare if a table satisfies 3NF but does not satisfy BCNF. BCNF is a slightly stronger version of the third normal form.

For example:

StudentID	Subject	Professor
401	Java	Professor Campbell
402	Java	Professor Campbell
403	C	Professor Gray
404	C++	Professor Johnson
401	C	Professor Gray

Here two different students can select the same subject and one student can select two different subjects. That is to say, ‘StudentID + Subject’ can be used as the primary key, as the two together uniquely identifies each row.

In this case, StudentID and Subject are prime attributes (See what prime attributes is from 2NF section) while Professor is non-prime. Yet, as professors determine subjects (Professor -> Subject), we have A -> B while B is a prime attribute but A is a non-prime attribute. This doesn’t satisfy BCNF and might trigger anomalies.

To fix that, we can remove the Professor column and put it into a separate table, which forms:

StudentID	ProfessorID
401	1901
402	1901
403	1902
404	1903

ProfessorID	Professor	Subject
1901	Professor Campbell	Java
1902	Professor Gray	C
1903	Professor Johnson	C++

BCNF Normal Form is a more restricted form of normalization and provides even better database integrity.

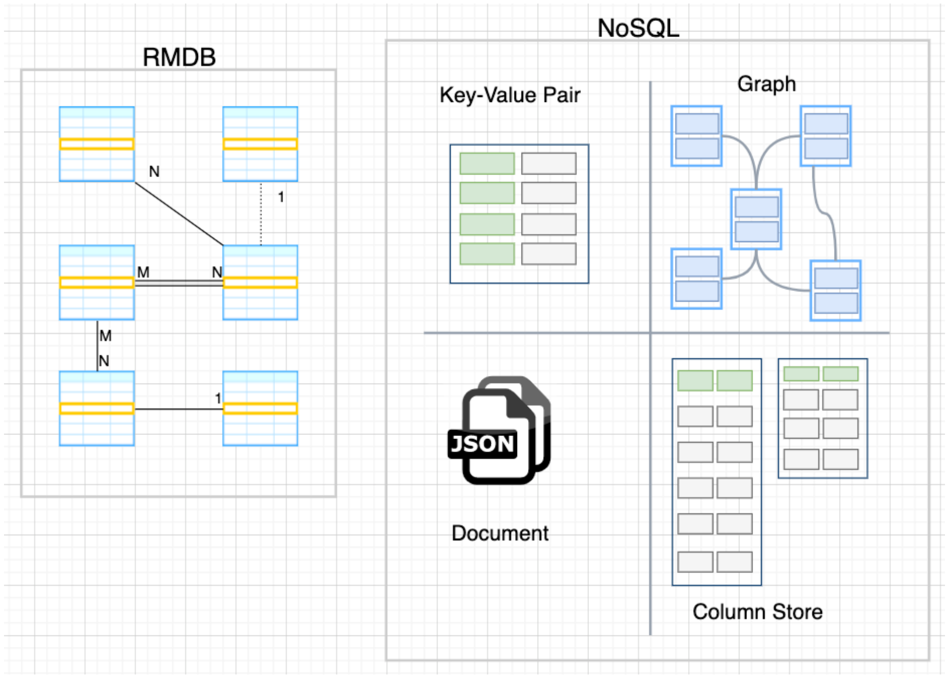
NoSQL Database

NoSQL, which stands for “Non-SQL” or “Not Only SQL”, is a database design approach that provides flexible schemas for the storage and retrieval of data beyond the traditional table structures found in relational databases.

Even if we call it NoSQL, it in general still supports SQL like queries and in a modern world of microservices, NoSQL and traditional SQL databases are commonly used together in a single application.

The fundamental differences between SQL and NoSQL are not all that complicated. Each just has a different way of storing and retrieving data.

With SQL databases, all data has an inherent structure, a schema – a formal definition of how data should be inserted into a database. For example, a column in a table is integer type and will accept only integer values. This clear definition is somewhat rigid, but on the other hand provides easy aggregation as well as reliability.



In the world of NoSQL, data can be stored in a schema-less or free-form fashion. That is, there will be no rigid rules around, rather flexible data models and operations. Generally, there will be four common types of NoSQL models:

- **Key-value** store: Stores data with simple indexed keys and values. Examples include Oracle NoSQL database, Redis, Amazon DynamoDB

- **Graph** database: Presents interconnected data as a logic path. Examples include GraphDB, Neo4j
- **Document** database: A more complex and structured version of key-value model, where each document has its retrieve key. Examples include: MongoDB, Couchbase
- **Wide Column** store: Uses tables, rows and columns. But the format and naming of the columns can vary in different rows within the same table. Examples include Apache Cassandra, Apache HBase

SQL VS NoSQL

Here is a summary with a comparison between SQL and NoSQL.

	SQL	**NoSQL**
Design	Uses SQL syntax for queries and analyze data	Various kinds of database technologies in response to the modern application demands
Type	Table-based	Key-value pair, graph, documents, wide column
Schema	Predefined schema	Dynamic schema for unstructured data
Scalability	Vertical scalable	Horizontally scalable
Suited Query	Good for complex query intensive environment	Not a good fit for complex queries
Hierarchical Data Storage	Not suitable for hierarchical data storage	Better fit for hierarchical data store as it supports key-value pair
Consistency	Strong consistency	Various by DB technologies used
Importance	Should be used when data validity is important	Should be used when fast data operations is more important

Use SQL if:

- You are working with complex queries and reports. With SQL, you can build one script that retrieves and presents your data. NoSQL doesn't support relation between data types. Running queries in NoSQL is doable, but slower.

- You have a high transaction application. SQL databases are a better fit for heavy duty or complex transactions because it’s more stable and ensures data integrity.

- You need to ensure ACID compliance (Atomicity, Consistency, Isolation, Durability) or defining exactly how transactions interact with a database.

- You don’t anticipate a lot of changes or growth. If you are not working with a large volume of data or many data types, NoSQL would be an overkill.

Use NoSQL if:

- You are constantly adding new features, functions, data types and it’s difficult to predict how the application will grow over time. Changing data models in SQL is troublesome which requires code changes. A lot of time is invested designing the data model because changes will impact all or most of the layers in the application. Yet, in NoSQL, we are working with a highly flexible schema design or no predefined schema. The data modeling process is iterative and adaptive. Changing the structure or schema will not impact development cycles or create any downtime for the application.

- You are not concerned about data consistency and 100% data integrity is not the top goal.

- You have a lot of data, many different data types, and your data needs will only grow over time. NoSQL makes it easy to store all different types of data together and without having to invest time into defining what type of data you’re storing in advance.

- Your data needs to scale up, out, and down. NoSQL provides much greater flexibility and the ability to control costs as your data needs change.

Using both SQL and NoSQL together in a system is doable when you want to leverage advantages of both databases. For example, you can have a fat, non-distributed, unscalable relational database as the single source of truth in your system, where you use NoSQL as the layer to intercept the bulk of the traffic for database operations. The point is to fit the right database for your contextual needs.

Chapter 4 – HTTP

In the realm of IT development, a deep understanding of the HTTP (Hypertext Transfer Protocol) protocol is of utmost importance.

HTTP forms the foundation of communication on the World Wide Web, facilitating the transfer of information between web servers and clients. It serves as the protocol through which web browsers request resources, such as web pages or multimedia content, and servers respond with the requested data. Proficiency in HTTP enables developers to effectively design and build web applications, ensuring smooth and efficient data exchange.

Understanding the intricacies of HTTP methods, status codes, headers, and cookies allows developers to troubleshoot issues, optimize performance, and implement secure and reliable web solutions. Furthermore, with the rise of API-driven development, knowledge of HTTP is indispensable for working with RESTful APIs, as they often utilize HTTP methods and status codes for interaction.

Key Points

- **The Web:** A global client/server network using protocols like HTTP for communication.

- **HTTP:** Application protocol for data transfer on the internet.

- **HTTP Communication:**

  - Client-server, request-response model.
  - Stateless (each request independent) but cookies enable stateful sessions.
  - Versions: HTTP/1.1, HTTP/2, HTTP/3.

- **Browsers:** Translate URLs into requests and render responses.

- **URLs:** Unique resource identifiers with syntax: `protocol://hostname:port/path-file-name`.

- **URIs:** More general than URLs, can include parameters and fragments.

- **HTTP Messages:**

  - Request: Request line (method, URI, version), headers, optional body.
  - Response: Status line (version, code, phrase), headers, optional body.

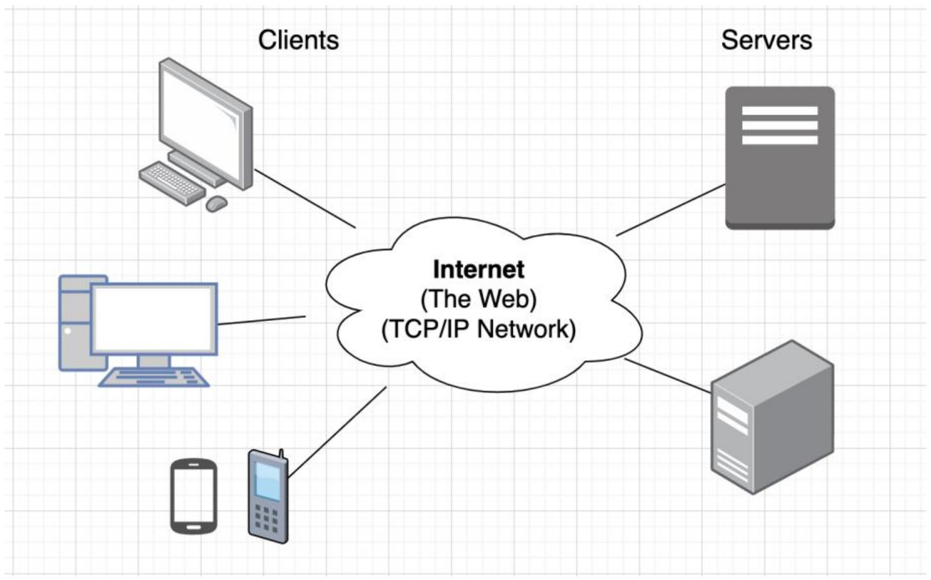
- **HTTP Methods:** GET, POST, PUT, DELETE, HEAD, OPTIONS.
- **GET vs. POST:** GET for retrieving data, POST for sending data.
- **Status Codes:** Indicate request outcome (e.g., 200 OK, 404 Not Found).
- **Request Headers:** Provide client information (e.g., Host, Accept, User-Agent).
- **Content Negotiation:** Selecting the best data format for responses using headers (Accept) or URL patterns.

The Web

Internet (or The Web) is a global distributed client/server system.

In order for proper communication to take place between the client and the server, web applications must agree on a specific protocol.

There are various protocols such as HTTP, FTP, SMTP, POP etc. But perhaps, HTTP is the most popular application protocol used on the Internet.

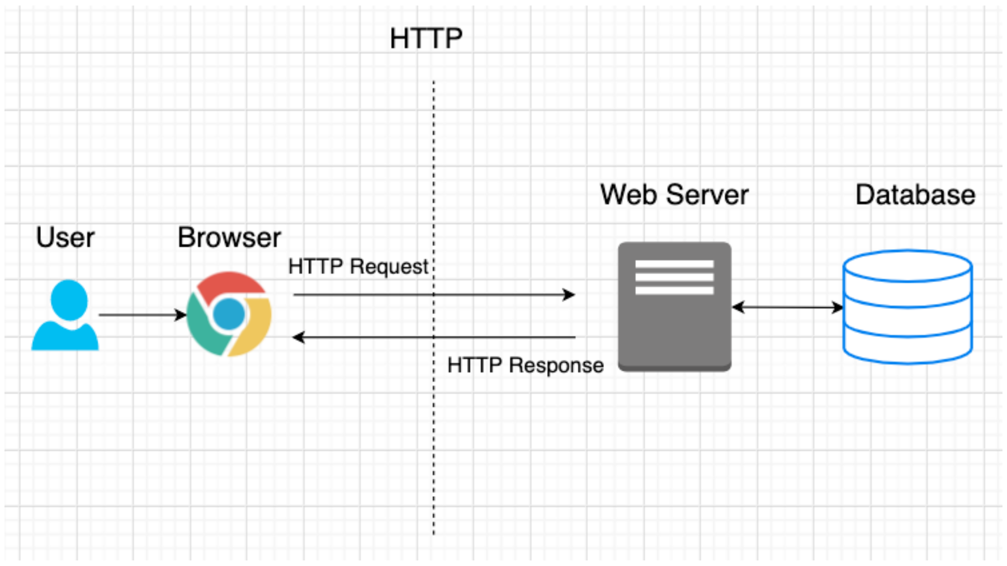


HTTP Protocol

HTTP (Hypertext Transfer Protocol) is the foundation for data communication for the Global Internet since 1990.

HTTP is based on TCP/IP to deliver data (HTML files, image files, query results etc.) on the Internet. The **default port is 80** while other ports are used sometimes.

Http specification specifies how clients' request data will be constructed and sent to the server, and how the servers respond to these requests.



Request and Response

HTTP is an asymmetric request-response client-server. This means an HTTP client (browser) initializes a request message to an HTTP server and the server, in turn, returns a response message, not the other way around.

Stateless but NOT Session-less

The HTTP protocol is a **stateless** protocol. That means the server isn't required to store session information and each request is independent of another. The server and client are aware of each other only during a current request. After that, both sides forget about each other.

While the core of HTTP itself is stateless, HTTP cookies allow stateful sessions. Using header extensibility, HTTP Cookies are added to the workflow, allowing session creation on each HTTP request to share the same context, or the same state.

Version

Two HTTP versions are currently being used widely: HTTP/1.0 and HTTP/1.1. HTTP/2.0 is also underway.

HTTP/1.0 was introduced in 1996 and HTTP/1.1 in 1997. HTTP/1.1 can be considered an upgrade for HTTP/1.0:

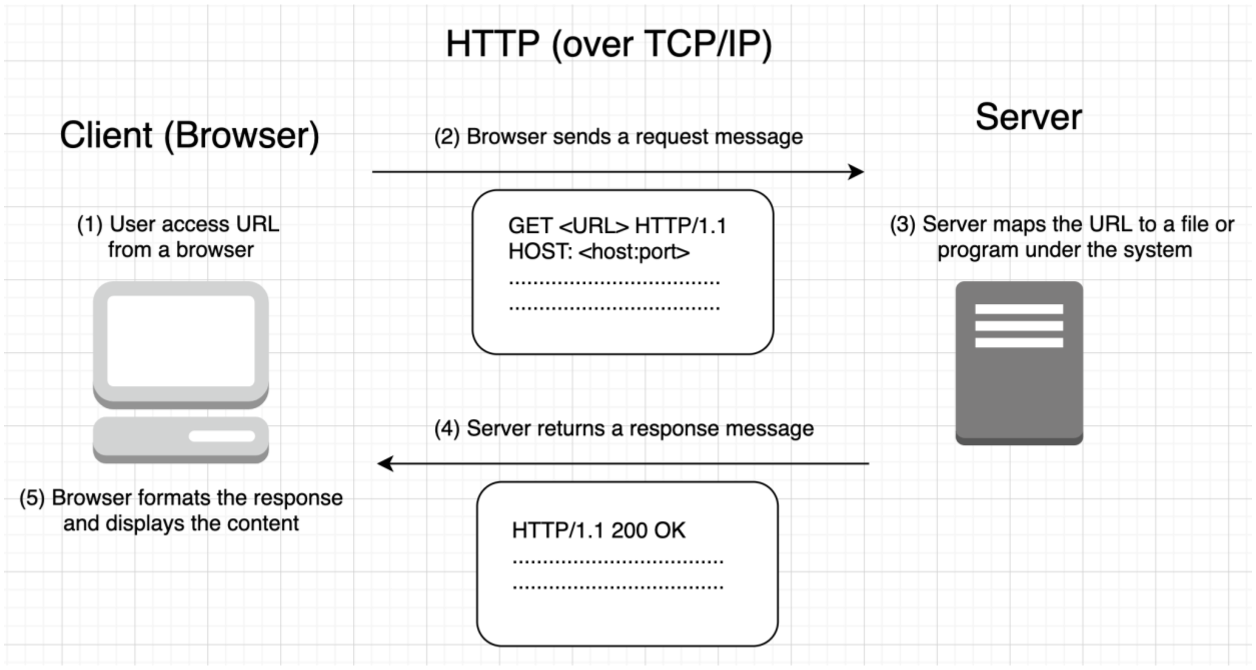
- HTTP/1.0 defines 16 status codes
- HTTP/1.1 defines 24 status codes
- HTTP/1.1 has a warning header capable of producing many secondary status alerts
- HTTP/1.0 authentication is somewhat unsafe as it is not encrypted
- HTTP/1.1 is safer as it uses a checksum of username, password and one-time value

Let's not step into the details for now.
HTTP/2.0 was introduced in 2015. It switches from ASCII to Binary. The general purpose is to make our applications faster, simpler and more robust.

HTTP/3.0 was introduced in 2018. It switches underlying transport from TCP to UDP. It's still young but Google Chrome has added support for it.

Browser

When you open a browser to visit a URL, you type in something like http://www.myhost.com/index.html, the browser turns the URL into a request message and sends it to the HTTP server. The server interprets the request message, and returns you an appropriate response message, which is either the resource you requested or an error message.


HTTP (over TCP/IP)
Client (Browser)
Server
(2) Browser sends a request message
(1) User access URL from a browser
GET <URL> HTTP/1.1
HOST: <host:port>
(3) Server maps the URL to a file or program under the system
(4) Server returns a response message
(5) Browser formats the response and displays the content
HTTP/1.1 200 OK

Uniform Resource Locator (URL)

A **URL** (Uniform Resource Locator) is used to uniquely identify a resource over the web.

URL has this syntax: *`protocol://hostname:port/path-file-name`*

- **Protocol**: The application level protocol used by the client and server, e.g. HTTP, FTP, telnet etc.
- **Hostname**: The DNS domain name (e.g. www.google.com) or IP address (e.g. 172.217.10.228)
- **Port**: The TCP port number that the server is listening for incoming request from the clients
- **Path-and-file-name**: The name and location of the requested resource

Some examples would be:

- ftp://www.ftp.org/docs/doc.txt
- mailto:john.wilson@gmail.com
- telnet://www.somehost.com/

Encoded URL

URLs cannot contain special characters, e.g. space or '~'. Special characters are encoded, in the form of *%xx*, where *xx* is the ASCII hex code for the character.

For example, '~' is encoded as *%7e*, a space is encoded as *%20*.

Uniform Resource Identifier (URI)

URI is more general than URL, which can even locate a fragment within a resource.

URI has this syntax: *`protocol://hostname:port/path?request-parameters#nameAnchor`*

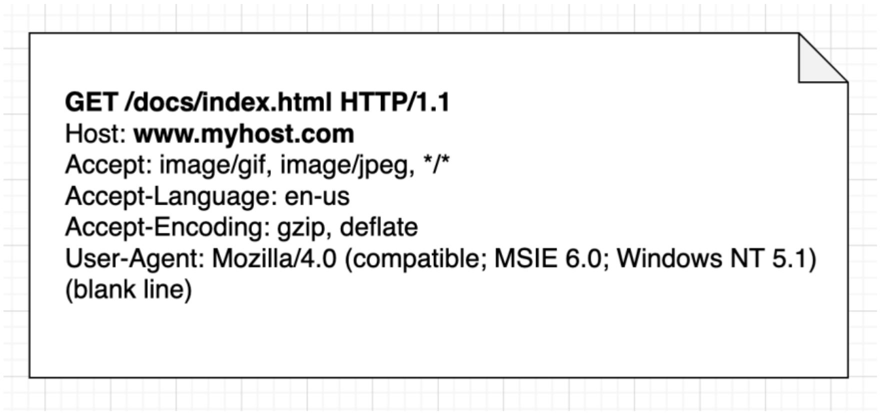
- The request parameters are in the form of *name=value* pairs. Parameters start with a '?' question mark and are separated by '&' ampersand.
- *#namerAnchor* identifies a fragment within an HTML document

HTTP Request/Response Message

HTTP client and server communicate by sending text messages.

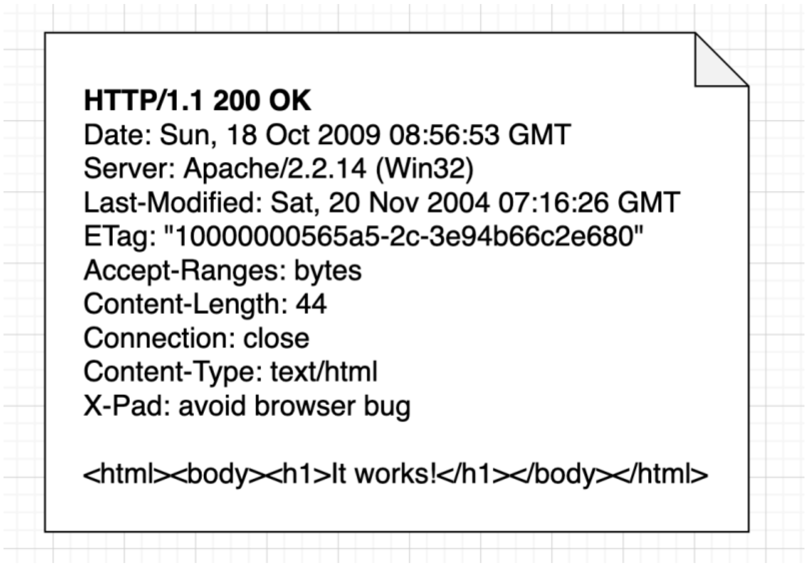
Whenever you enter a URL in the browser address, it translates that URL into a request message based on the protocol specified and sends that request message to the target server.

Here is a sample request message for accessing http://www.myhost.com/index.html in the browser (row numbers are added for reference purpose):



```
GET /docs/index.html HTTP/1.1
Host: www.myhost.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
(blank line)
```

A response will be sent back once the server has processed the request:



```
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
ETag: "10000000565a5-2c-3e94b66c2e680"
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug

<html><body><h1>It works!</h1></body></html>
```

When the browser receives the response message, it interprets the message and displays the contents of the message according to the media type of the response (Content-Type: text/html).

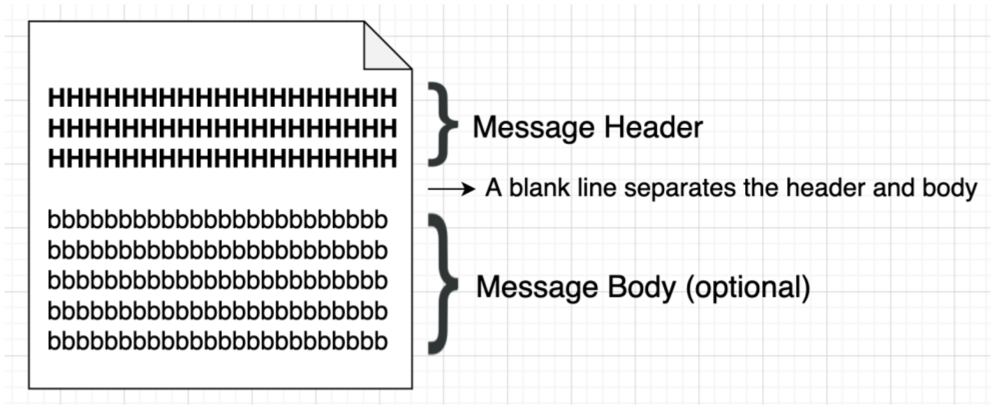
Common media type includes:

- “text/plain”
- “text/html”
- “image/gif”
- “image/jpeg”
- “audio/mpeg”
- “video/mpeg”
- “application/msword”
- “application/pdf”

Request/Response Messages

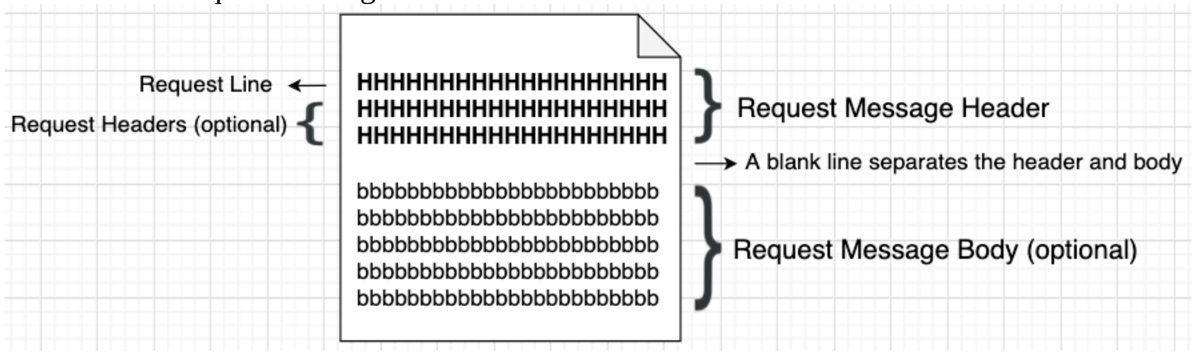
Let’s take a closer look at the message request/response messages.

An HTTP message (request and response) consists of a message *header* and an optional message *body*, which are separated by a *blank line*.



The Request Message

The format of a request message looks like this:



Request Line

The first line in the header is "Request Line", which has this syntax:

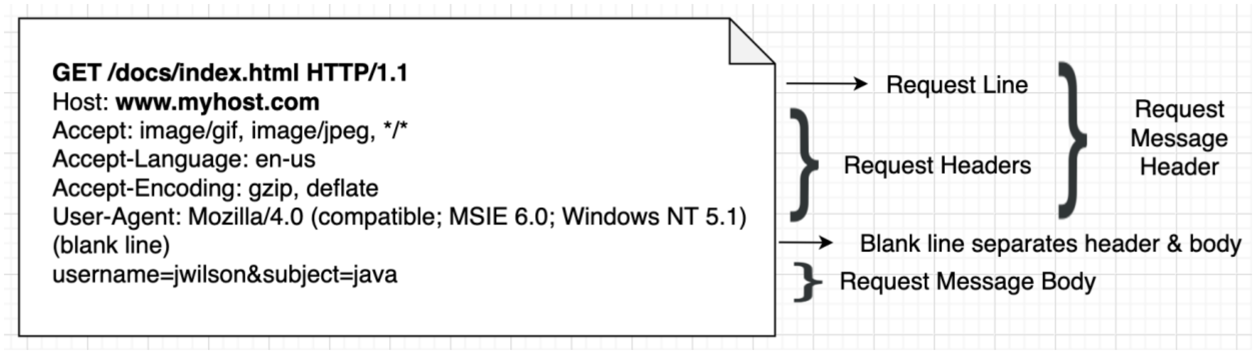
request-method-name<space>request-URI<space>HTTP-version

- **Request-method-name**: HTTP defines a list of request methods, e.g. GET, POST, HEAD, OPTIONS etc. Clients can use one of these methods to send a request to the server.
- **Request-URI**: the resource requested
- **HTTP-version**: two versions are currently in use: HTTP/1.0 or HTTP/1.1

Request Headers

Request headers are in the format of *name:value* pairs, where value can be multiple ones separated by commas.

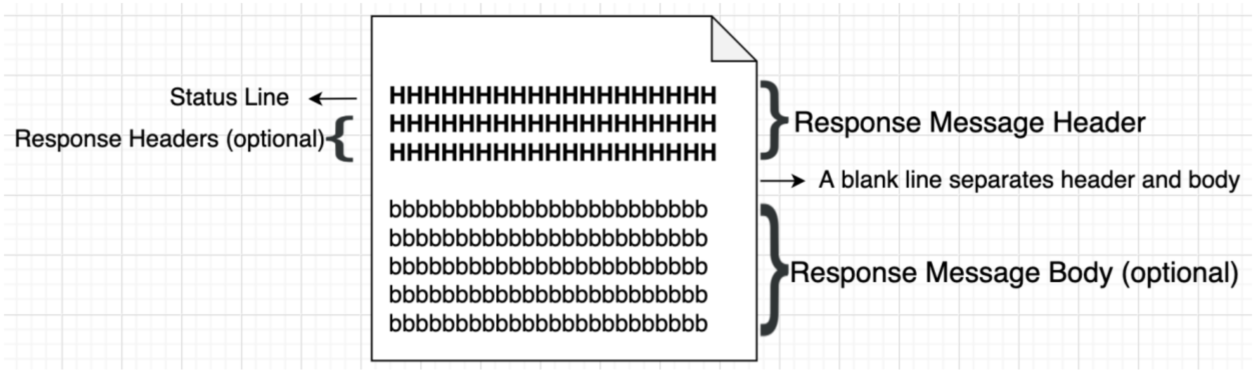
Request Example



```
GET /docs/index.html HTTP/1.1
Host: www.myhost.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
(blank line)
username=jwilson&subject=java
```

The Response Message

The format of a response message looks like this:



Status Line

The first line in the header is "Status Line", which has this syntax:

HTTP-version<space>status-code<space>reason-phrase

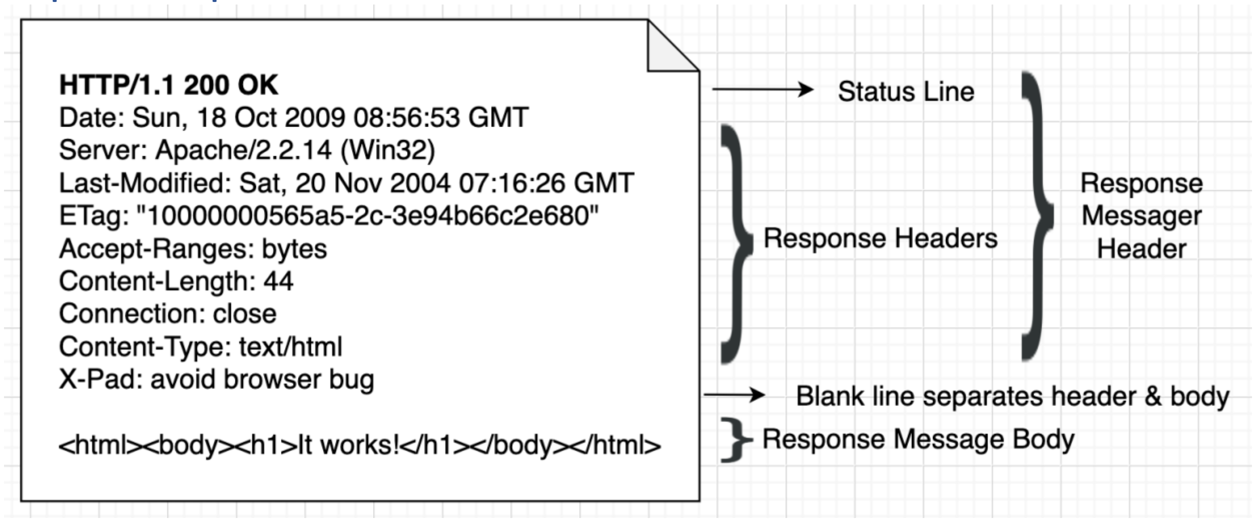
- **HTTP-version**: HTTP/1.0 or HTTP/1
- **Status-code**: a 3-digit number reflecting the outcome status of the request
- **Reason-phrase**: a short explanation for the status code returned

Example status code and reasons: "200 OK", "404 Not Found", "403 Forbidden", "500 Internal Server Error".

Response Headers

Like request headers, response headers are in the format of *name:value* pairs as well, where values can be multiple ones separated by commas.

Response Example



```
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
ETag: "10000000565a5-2c-3e94b66c2e680"
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug

<html><body><h1>It works!</h1></body></html>
```

HTTP Methods

HTTP has defined a list of request methods. Clients can use one of the methods to send a request to HTTP servers.

Here is a list of them:

- **GET**: get a web resource from the server
- **POST**: post data to the server
- **PUT**: update data at the server
- **DELETE**: delete data at the server
- **HEAD**: get the header as a GET method would obtain
- **OPTIONS**: let the server return the list of request methods supported
- Other methods

The GET method is most frequently used and then POST, PUT, DELETE and OPTIONS.

When you type in a URL in your browser and hit enter to access a web page, the HTTP method is GET. In other words, a browser address field can only support the GET method. In order to use other methods, you can use a HTTP REST client. There are online test clients and desktop application clients such as POSTMAN.

GET VS POST

Generally speaking, when you are not sending data to the server but just accessing the server resource, you will use the GET method. Instead, if you are sending data to the server, you probably want to use POST instead of GET.

GET method can also send data through URL query parameter, which is different from POST method, as the data is contained in the POST body instead of URL parameter. This has some advantages:

- The amount of data is unlimited
- The data is not shown on the address box of the browser

A POST Method Example

We use https://apitester.com/, an online API testing tool, to show the example here. You can find plenty of similar tools online.

Besides, we use https://reqres.in/, an online testing server for API testing with fake data.

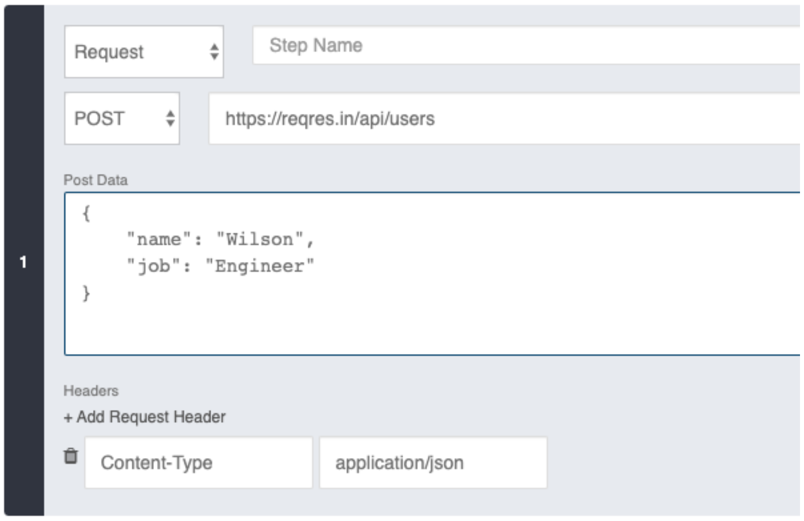
Test Scenario

The test scenario is fairly straightforward. The testing server has an endpoint /api/users, which accepts POST method for creating a new user. The posted data can be in JSON format and should have Name and Job attributes.

API Name	Base URL	Endpoint	Method	Sample Data
User creation	https://reqres.in	/api/users	POST	{ "name": "morpheus", "job": "leader" }

Here is the client setup.

- HTTP method has been set to POST
- Target URL is https://reqres.in/api/users
- Post Data is
  ```
  {
    "name": "Wilson",
    "job": "Engineer"
  }
  ```
- A header is added to specify content-type
  Content-Type : application/json



POST https://reqres.in/api/users

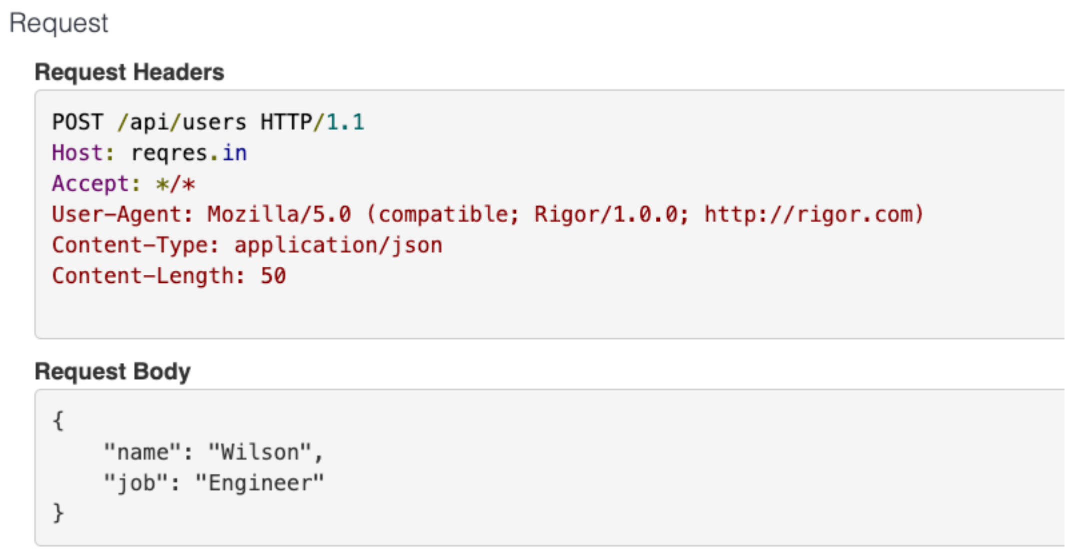
Post Data
```
{
    "name": "Wilson",
    "job": "Engineer"
}
```

Headers

Content-Type application/json

After clicking the “Test” button, the client sends the request to the target server and receives a response from the server.

Request Information


Request

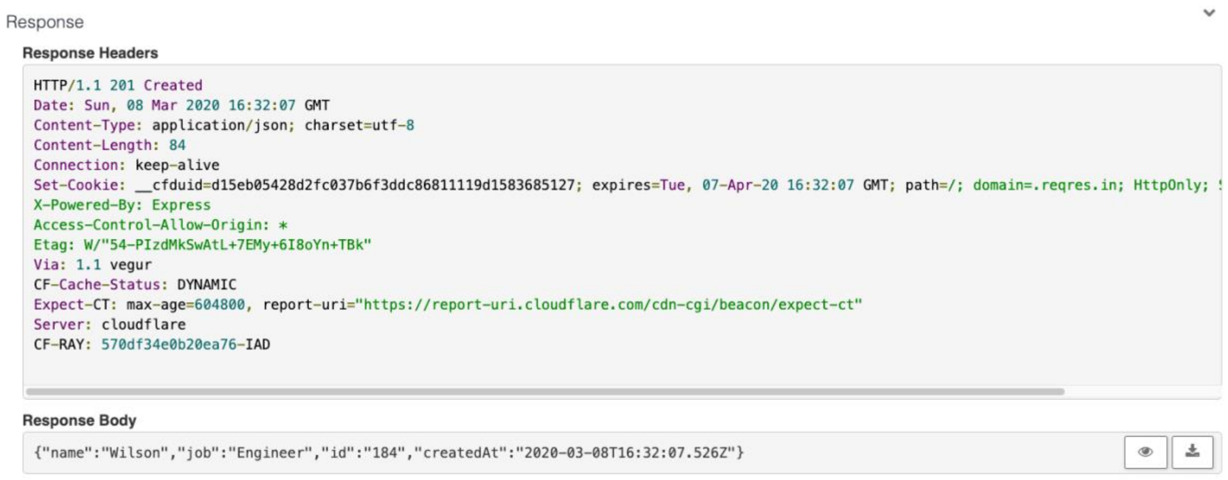
Request Headers

```
POST /api/users HTTP/1.1
Host: reqres.in
Accept: */*
User-Agent: Mozilla/5.0 (compatible; Rigor/1.0.0; http://rigor.com)
Content-Type: application/json
Content-Length: 50
```

Request Body

```
{
    "name": "Wilson",
    "job": "Engineer"
}
```

Response Information


Response

Response Headers

```
HTTP/1.1 201 Created
Date: Sun, 08 Mar 2020 16:32:07 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 84
Connection: keep-alive
Set-Cookie: __cfduid=d15eb05428d2fc037b6f3ddc86811119d1583685127; expires=Tue, 07-Apr-20 16:32:07 GMT; path=/; domain=.reqres.in; HttpOnly;
X-Powered-By: Express
Access-Control-Allow-Origin: *
Etag: W/"54-PIzdMkSwAtL+7EMy+6I8oYn+TBk"
Via: 1.1 vegur
CF-Cache-Status: DYNAMIC
Expect-CT: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
Server: cloudflare
CF-RAY: 570df34e0b20ea76-IAD
```

Response Body

```
{"name":"Wilson","job":"Engineer","id":"184","createdAt":"2020-03-08T16:32:07.526Z"}
```

Response Status Code

The status code will be included in the response from the server and it consists of a 3-digit number.

The first digit in the status code defines the category of the status in the response:

- **1xx (Informational):** request received, server is continuing the process

- **2xx (Success):** the request received successfully, understood, accepted and serviced
- **3xx (Redirection):** redirection takes place to continue further action
- **4xx (Client Error):** request has some error, e.g. bad syntax
- **5xx (Server Error):** server failed to fulfill the valid request

It is important to know the commonly encountered status code.

- **100 Continue:** server has received the request and is in the process of giving the response
- **200 OK:** the request looks good and has been fulfilled
- **301 Moved Permanently:** the requested resource has moved to a new location; the URL of the new location is given in the response header called "*Location*"
- **302 Found & Redirect** (or Move Temporarily): similar to 301, but the new location is temporary
- **400 Bad Request**: Server could not interpret or understand the request, probably syntax error in the request message
- **401 Authentication Required**: authentication is required (e.g. username/password) to access the resource
- **403 Forbidden**: authorization is required to access the resource (usually means client user doesn't have enough privileges)
- **404 Not Found**: the requested resource cannot be found in the server
- **405 Method Not Allowed**: the request method used is not allowed from the server side
- **408 Request Timeout**: the request to the server took longer than the server was prepared to wait (slow connection or wrong URL)
- **414 Request URI Too Long**: the URL requested by client is longer than the server's expectation
- **500 Internal Server Error:** server-side processing error, often caused by server-side programming issue
- **501 Method Not Implemented:** invalid request method (e.g "GET" mistakenly spelled as "Get")
- **502 Bad Gateway**: the requested server is acting as a gateway and it received an invalid response from its upstream
- **503 Service Unavailable**: server cannot respond due to overloading or maintenance
- **504 Gateway Timeout**: the requested server is acting as a proxy or gateway, and it received a timeout from upstream

Request Headers

Let's see some of the commonly-used request headers.

- **Host: <*domain-name*> -**

- as HTTP/1.1 support virtual hosts, that is to say multiple DNS names (www.myhost1.com, www.myhost2.com) can be attached to the same physical server
  - Host header is required in HTTP/1.1 to select a host
- **Accept**: **mime-type1, mime-type2** … -
  - client can let server know what MIME types it can handle or prefer by using Accept header
  - if the server has multiple versions of the document requested, it can check this header to decide which version to deliver to the client and this process is called **content-type negotiation**

What is MIME?
MIME stands for **Multipurpose Internet Mail** Extensions, as this is originally created for emails but nowadays used a lot in other protocols.

It's a way of identifying files on the internet and its format consist of two parts: "type/subtype". For example, "text/html" indicates file type as html, "image/png" indicates file type as png image.

The "x-" prefix in subtype simply means the subtype is non-standard.

- **Accept-Language: language-1, language-2 … -**
  - client can let server know what language it can handle or prefer by using Accept-Language header; this is called **language negotiation**
- **Accept-Charset: Charset-1, Charset-2 … -**
  - which charset is supported by client, e.g. *UTF8*
- **Accept-Encoding: encoding-method1, encoding-method2 … -**
  - which encoding method is supported by client; server can choose encode or compress the document before returning to the client, where server must set the response header "Content-Encoding" to inform the client the encoding type
  - e.g. x-gzip, x-compress
- **Connection: Close|Keep-Alive** –
  - tell server whether to close or keep alive the client-server connection after this request
  - HTTP/1.1 uses persistent connection (keep-alive) by default while HTTP/1.0 close the connection by default
- **Referrer: referer-URL** –
  - client uses this header to indicated the original URL page
  - for example, you click a link from page1 which opens page2, the page1 URL will be the referer-URL in the request to visit page2
  - all major browsers set this header, which can be used to track where the request is coming from

- please note this header can be easily spoofed

- **User-Agent: *browser-type*** –
    - the type of the browser used for the request
- **Content-Length: *number-of-bytes*** –
    - used by POST method to inform server the length of the request body
- **Content-Type: *mime-type*** –
    - used by POST method to inform server the media type of the request body
- **Cache-Control: *no-cache|others*** –
    - used by client to specify how pages are cached
    - "no-cache" (not recognized by HTTP/1.0) means obtaining a fresh copy from server
    - "Pragma: no-cache" is used by HTTP/1.0
- **Authorization** –
    - used by client to supply credentials to access protected resources
- **Cookie: *cookie-name1=cookie-value1, cookie-name2=cookie-value2…*** -
    - client uses this header to send cookies back to the server, which was set by the server earlier
- **If-Modified-Since: *data*** –
    - let the server to send the page only if it has been modified after the specified date

Content Negotiation

Generally, resources can have multiple presentations, mostly because there may be multiple different clients expecting different representations.

Server-driven VS. Agent-driven Content Negotiation

If the selection of the best representation for a response is made by an algorithm at the server, it is called **Server-driven negotiation**. If that selection is made by agent (client) side, it's called **Agent-driven content negotiation**.

In most cases, server-driven negotiation is NOT used, as the server side will have to make a lot of assumptions about client expectations. It would rather be easier to let the client tell the server what it prefers. Hence, agent-driven content negotiation is more popular.

Content Negotiation using HTTP headers

As previously mentioned, a request can have data attached to it. To specify its type, the request can have a "Content-Type" header to tell the server what format the attached data is. Some common examples are "text/plain", "application/xml", "text/html", "application/json", "image/gif", and "image/jpeg".

```
Content-Type: application/json
```

Besides the content format being sent, a client can tell the server what format is preferred in the response with the “Accept” header. The values will be similar to the ones mentioned above.

```
Accept: application/json
```

Multiple values can be specified in the “Accept” header.

```
Accept: application/json,application/xml;q=0.9,*/*;q=0.8
```

If there is no “Accept” header, the server can send the response with the default type.

Content Negotiation using URL patterns

Besides passing the “Accept” header, the client can use file extensions specified in the URL to indicate the preferred format, if the target server supports that. For example,

```
http://api.myhost.com/v1/users/1281.xml
http://api.myhost.com/v1/users/1281.json
```

The first request will return an XML response and the second one will return a JSON response.

Chapter 5 – Computer Network Basics

In the IAM world, possessing a comprehensive understanding of networks is crucial. Networks provide the infrastructure that connects software applications, devices, and users, enabling efficient data exchange and communication.

Knowledge of network protocols, such as TCP/IP, UDP, and HTTP, allows software developers to design and implement applications that can effectively communicate over networks.

Understanding network architectures, such as client-server and peer-to-peer models, helps design software solutions that align with the underlying network infrastructure. Furthermore, being well-versed in network security concepts, such as encryption, authentication, and firewalls, enables developers to build secure software systems and protect sensitive data from unauthorized access.

Additionally, as software applications often rely on network-based services, such as APIs or cloud platforms, understanding networking principles facilitates seamless integration with these services.

Key Points

- **Networks:** Connect devices for data transmission.
- **Data Communication:** Involves message, sender, receiver, medium, and protocol.
- **Transmission Modes:** Simplex (one direction), half-duplex (both directions, not simultaneous), full-duplex (simultaneous).
- **Network Components:** Work devices (workstations, servers), control devices (hubs, switches, routers, gateways).
- **Network Types:** PAN (personal), LAN (local), MAN (metropolitan), WAN (wide), Internet.
- **Intranet/Extranet:** Private networks for internal (intranet) or controlled external (extranet) access.
- **LAN Technologies:** Ethernet (various speeds), VLANs (segment LANs).
- **Network Topologies:** Physical/logical arrangement of devices (point-to-point, bus, star, ring, mesh, tree, hybrid).
- **OSI Model:** 7-layer framework for network communication:

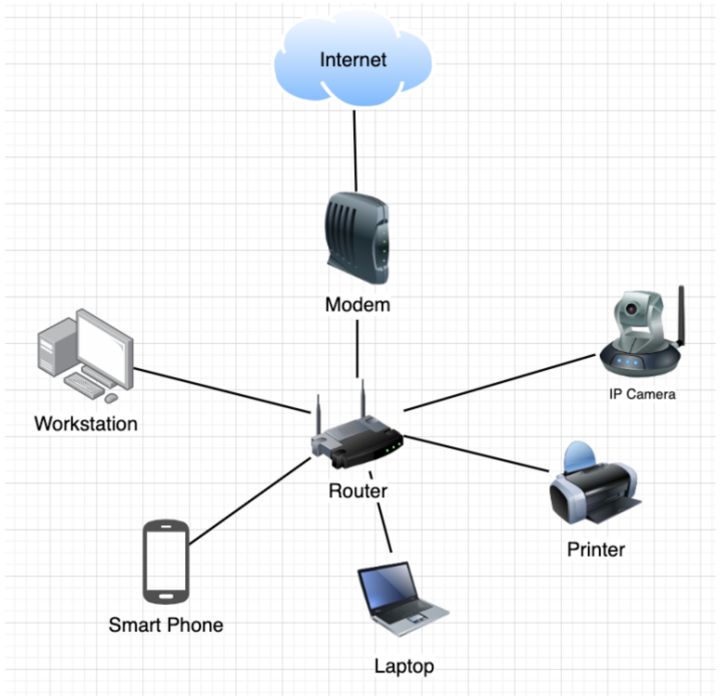
- Application: User interaction (HTTP, SMTP).
- Presentation: Data formatting (encryption, compression).
- Session: Communication management.
- Transport: Data segmentation and flow control (TCP).
- Network: Routing and addressing (IP).
- Data Link: Intra-network communication.
- Physical: Hardware (cables, bits).

- **Internet Model (TCP/IP):** Simplified 4-layer model.

Computer Networking

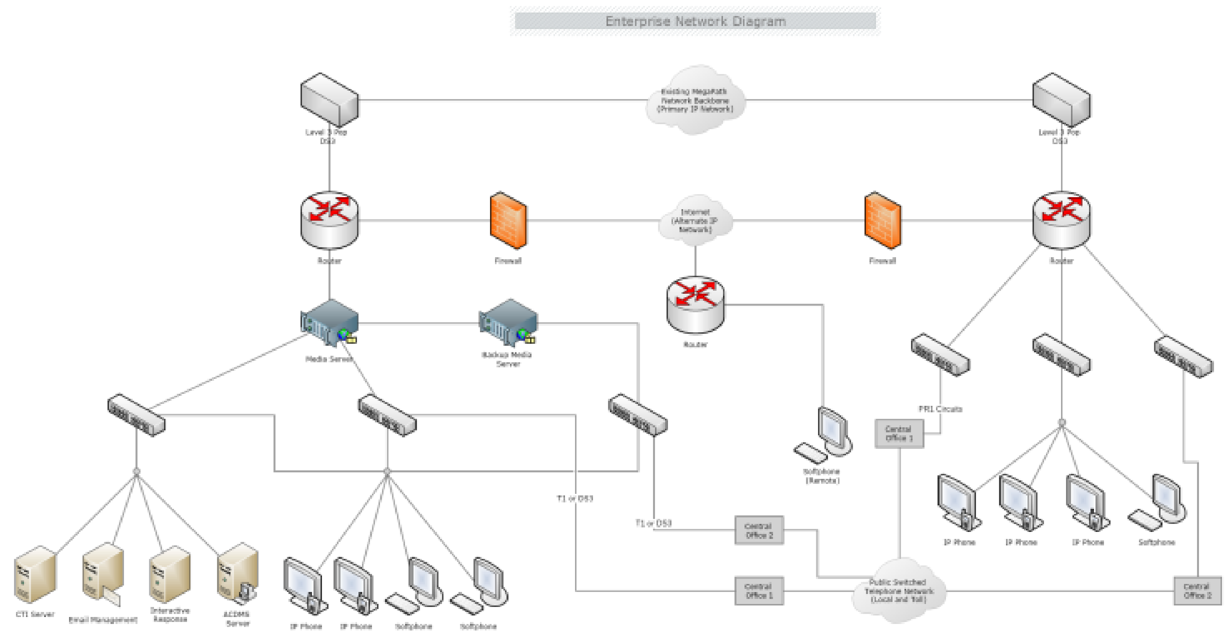
A Computer Network is a set of two or more computers and devices that will transmit or receive data through cable or wireless media.

Here is an example of a simple home network.



A router is used to route traffic to various network end devices and a modem is used to allow home networks to connect to the Internet through telephone lines.

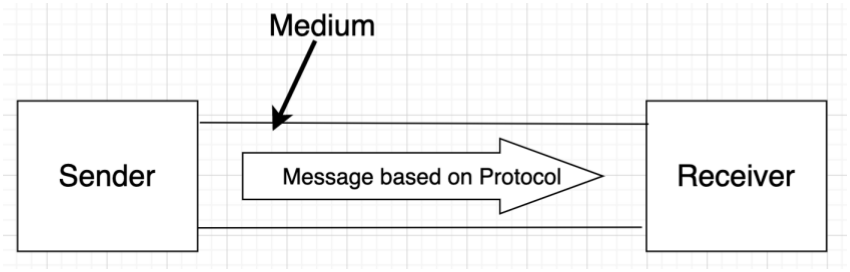
On the other hand, a network system could be complex as well.



We'll look at more details in the later part. Let's first start with a basic data communication model.

Data Communication Model

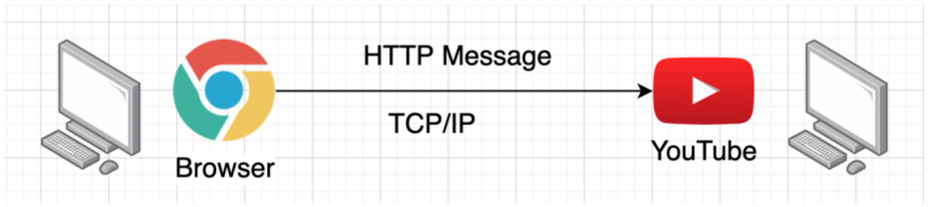
This is a basic model for two nodes in a network to communicate:



There are 5 components here:

- Message: the information or data being delivered
- Sender: the node which sends out the message
- Receiver: the node where the message is being sent to
- Medium: the medium through which the message travels
- Protocol: a set of rules define how message should be communicated

Here is a quick example. You open a browser from your laptop and type in the URL of the YouTube homepage.



- Message: the HTTP request to access the YouTube home page
- Sender: your laptop
- Receiver: YouTube server
- Medium: your local cable or wireless, global optical fiber
- Protocol: TCP/IP

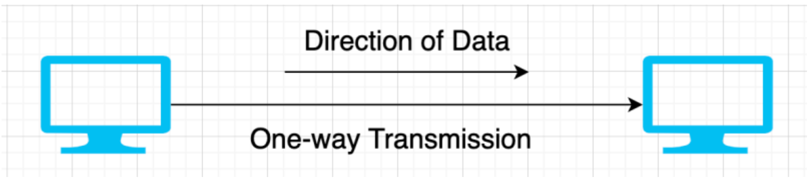
Transmission Modes

The transmission mode decides the direction of data flow and has three categories:

- Simplex
- Half-Duplex
- Full-Duplex

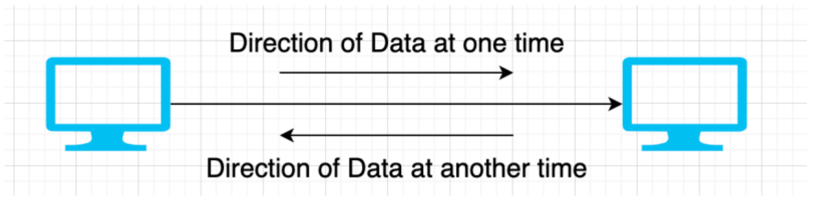
Simplex Mode

In Simplex Mode, data transmits in one direction only.



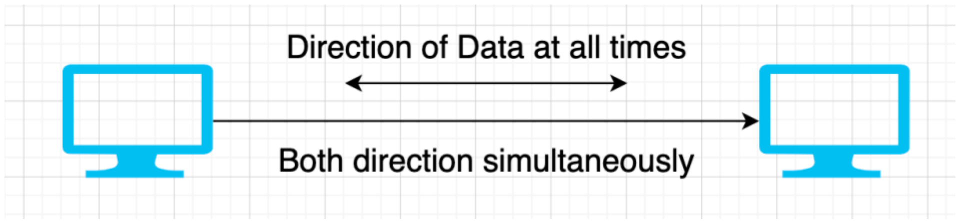
Sender can only send data and receiver can only receive data. Television is an example of simplex mode transmission.

Half-Duplex Mode
In Half-Duplex Mode, transmission happens in both directions, but they cannot happen at the same time.


Direction of Data at one time
Direction of Data at another time

If one device is sending data, then the other device cannot send data until the transmitted data is received. A walkie-talkie works in this mode.

Full-Duplex Mode
In Full-Duplex Mode, both devices can send and receive data simultaneously. Mobile phones we use are an example of this mode.



Typical Network Components
A network can contain various devices and those devices in general can be two categories: work devices and control devices.

Work devices can be workstations, laptops, servers, printers etc.

Control devices can be hubs, routers, switches, gateways, bridges, etc.

Here are some typical ones:

- **Workstation** – individual computer or group of computers, used by a single user to perform work
- **Server** – a computer provides functionality for other devices
- **Modem** – a device that allows computer to connect to the internet over existing telephone line

- **Hub** – most basic device that connects multiple computers or other network devices together; replaced by switches nowadays
- **Switch** – does what a hub does, but more efficiently
- **Router** – does what a switch does, but even more intelligently
- **Gateway** – does what a router does, plus translation between networking technologies
- **Bridge** – used to connect two or more hosts or network segments together
- **Wireless Access Points** – creates a wireless local area network (WLAN)

Modem VS Router

A modem (modulator-demodulator) is a device used to modulate and demodulate signals. It is used to connect you to an ISP (Internet Service Provider) such as Xfinity. It does not help to examine the data packet and is usually placed between a telephone line and a router.

A router is a networking device that allows you to create LAN or WAN. Internet comes to Modem and then to router and then to your devices via either Ethernet cable or WIFI signal. Router examines all data packets before forwarding it. It is usually placed between a computer network and modem.

Router VS Switch

A network router serves two primary functions: 1. Create and maintain a local area network; 2. Manage the data moving inside the network as well as entering and leaving the network. Router operates at Layer 3 (Network) in the OSI model and can perform NAT. Router operations revolve around IP Addresses. Routers can work with both wired and wireless networks.

A network switch connects various devices together on a single computer network. Since the process of linking network segments is also called bridging, switches are also referred to as bridging devices. Switch operates at Layer 2 (Data Link Layer) in OSI model and can’t perform NAT. Switches work with MAC addresses as it operates within the limit of a single network. Switches are restricted to wired network connections.

What is an Internet Service Provider (ISP)?
An ISP is usually a company that can provide you with access to the internet. Without subscription to ISP, even if you have a shiny computer with premium modem and router, you won’t have a connection to Internet.

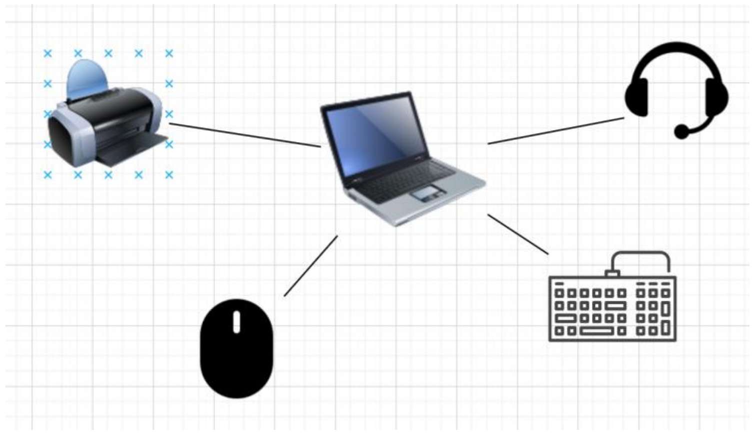
Whereas anyone can have a website, not everyone can be an ISP. It takes money, infrastructure and a great number of technicians to maintain miles of cables as well as services for hundreds of thousands of subscribers.

Computer Network Types

Generally speaking, networks are recognized based on their geographical span. It can be as small as the distance between your mobile phone and its Bluetooth headphone and as large as the internet itself.

Personal Area Network (PAN)

A PAN is the smallest network which is personal to a user, which may include devices like Bluetooth keyboard, mouse, headphone, printers, etc. PAN has connectivity range up to 10 meters.

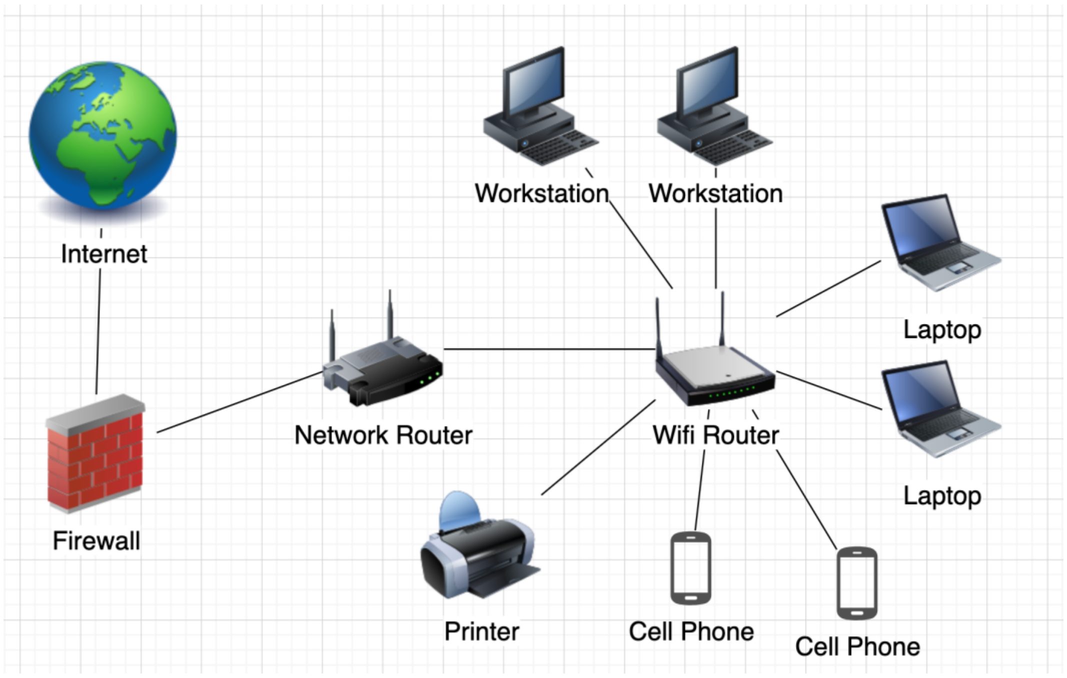


Local Area Network (LAN)

A computer network spanned inside a building and operated under a single administrative system is usually referred to as LAN. Number of systems connected in LAN may vary from two to as many as hundreds of thousands.

Regardless of size, a LAN’s single defining characteristic is that it connects devices that are in a single, limited area.

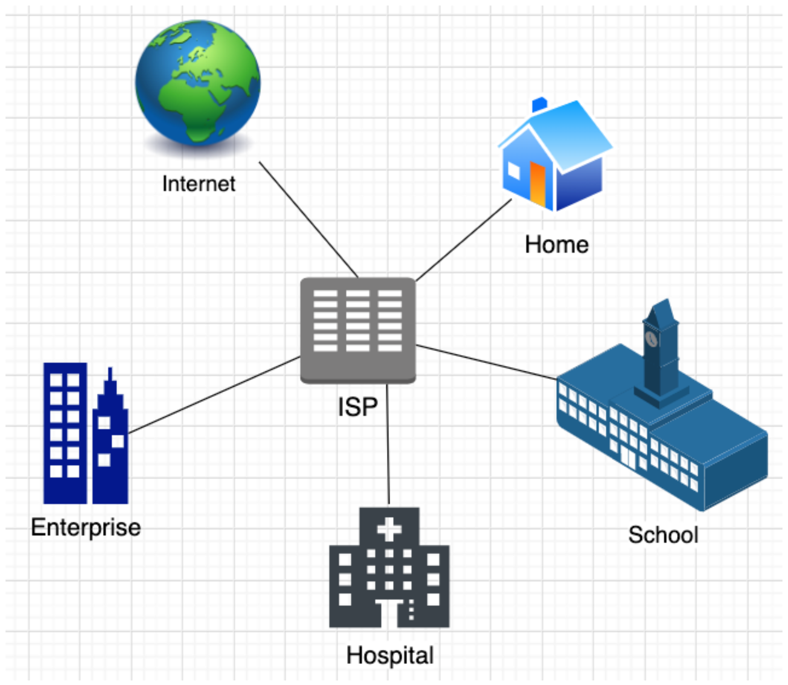
The advantages of a LAN are the same as those for any group of devices networked together. The devices can use a single internet connection, share files with one another, print to shared printers, and be accessed and even controlled by one another.



Metropolitan Area Network

MAN covers a larger area than LAN while smaller area as compared to WAN. It connects two or more computers that are apart but reside in the same or different cities.

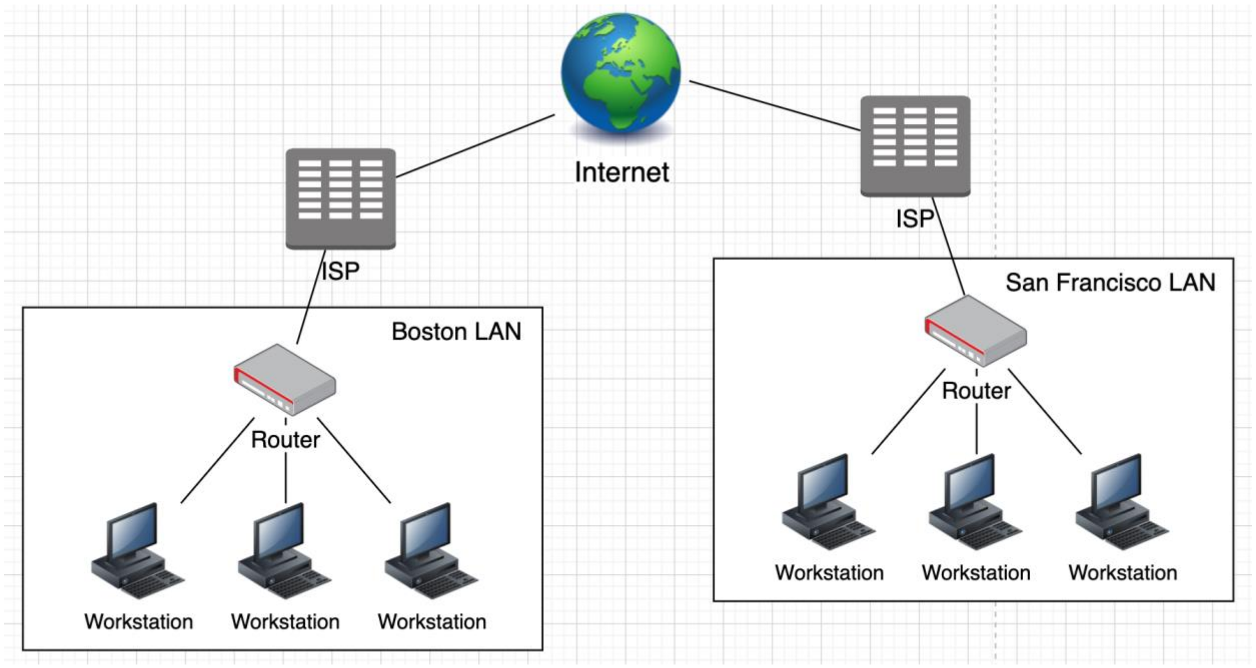
In general, a MAN is either owned by a user or by a network provider who sells service to users, rather than a single organization as in LAN.



In the above MAN, an ISP (Internet Service Provider) provides network service to different groups within the city.

Wide Area Network

WAN covers a wide area which may span across provinces or even countries. Generally speaking, telecommunication networks are WAN. These networks provide connectivity to MANs and LANs.



The above diagram shows two LANs in two different cities. They are connected through the internet to form a WAN.

Internet

The Internet can be seen as a WAN, but the largest WAN on planet earth.

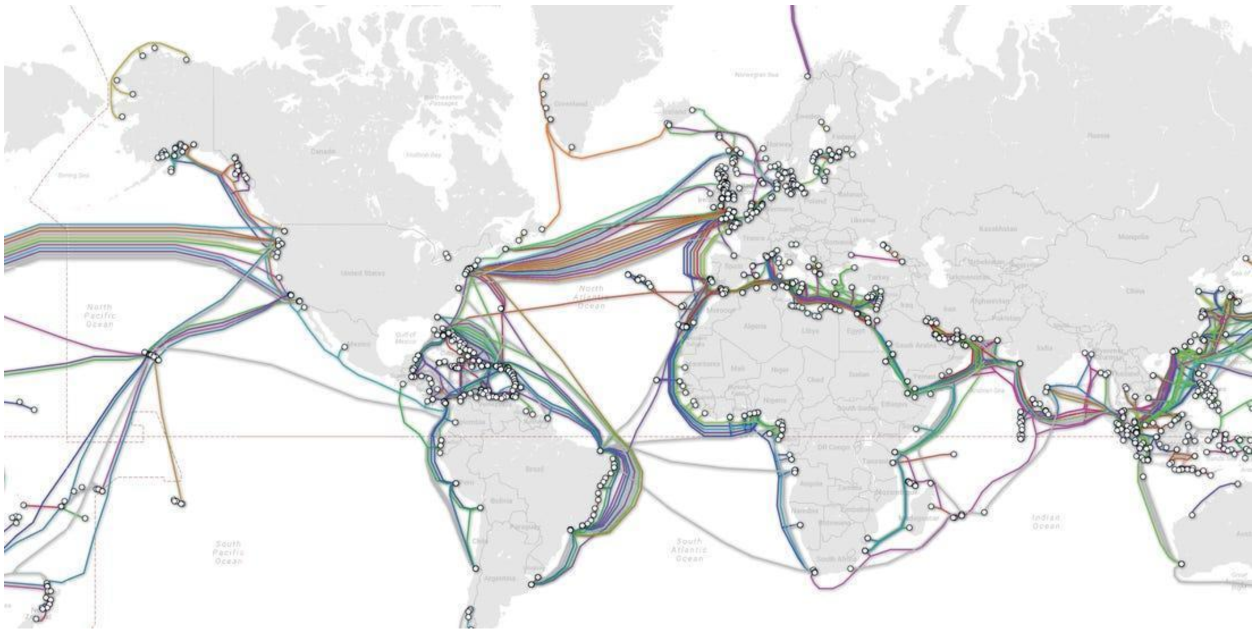
The Internet enables users to share and access enormous amounts of information worldwide. At a grand level, the internet works on the Client-Server model.

The Internet uses fiber optic cable as its infrastructure to connect around the world. To connect various continents, fiber optic cables are laid under the sea as submarine communication cables.

Internet is tightly involved in many aspects of our life, as we know:

- Web sites
- Email
- Instant Messaging
- Blogging
- Social Media
- Marketing
- Resource Sharing
- Audio and Video Streaming

The following map shows global fiber optic cable under the sea.



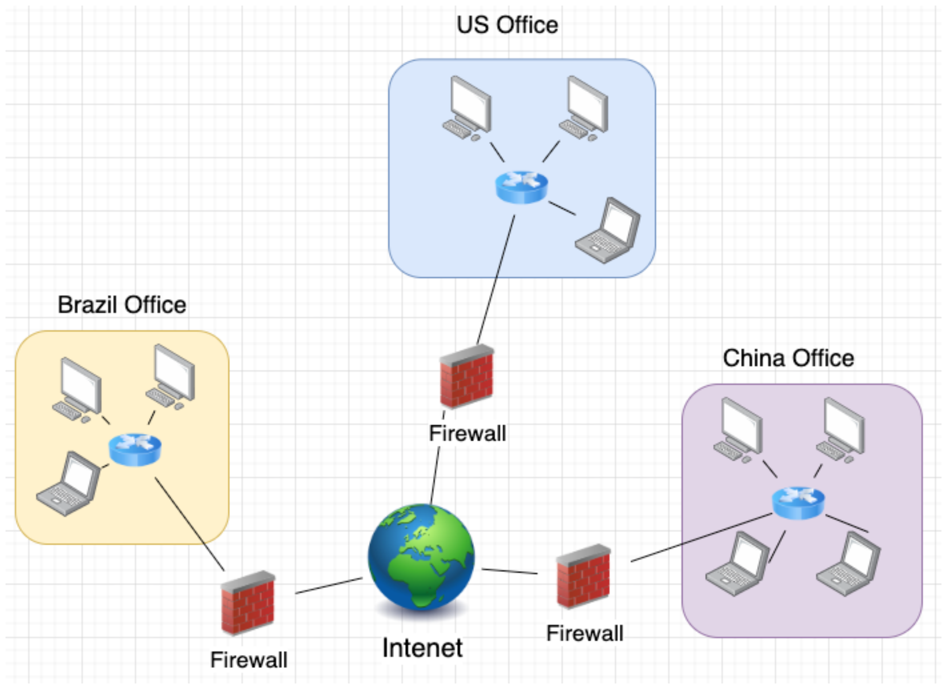
Intranet

Intranet is defined as a private network of computers within an organization with its own server and firewall.

The general difference between Internet and Intranet is that the Internet is a public network which is not owned by any entity, while an intranet is privately owned and not always accessible to the public. Oftentimes, a company or organization has their own intranet network and members or employees will have access to it.

Intranet can be a LAN or a WAN. We just wouldn’t use the term Intranet for a LAN with a home network of a couple of computers. More specifically, we use Intranet in regards to an enterprise or organization.

For example, a company has three offices globally, one in the US, one in China and One in Brazil. Each office has its own building and each building forms a LAN. Then this LAN is connected through the internet to form a WAN. However, even though the three offices are connected through the internet, the WAN network is not accessible to the public, as the company uses a firewall to block all unauthorized traffic. This WAN acts as an Intranet for the company.

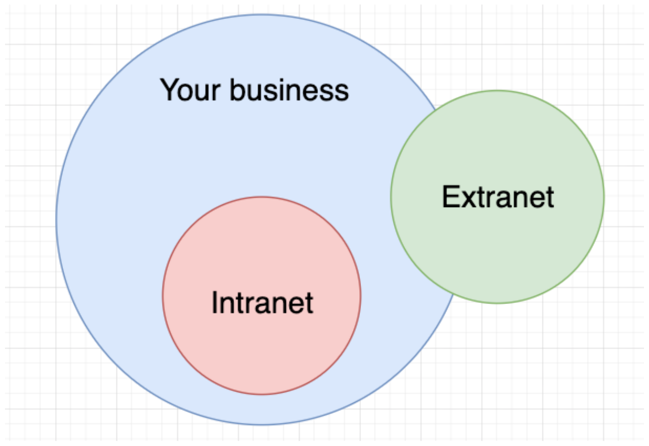


Intranet provides such advantages as easy storage of files and information, easy ways to communicate among employees, more security for organization's information and properties, etc.

Extranet

An Intranet is a network where employees can create content, communicate, collaborate, get stuff done and develop the company culture. On the other hand, an Extranet is like an Intranet, but provides controlled access to authorized customers, vendors, partners, or others outside the company.

Conceptually, the relationship looks like this:

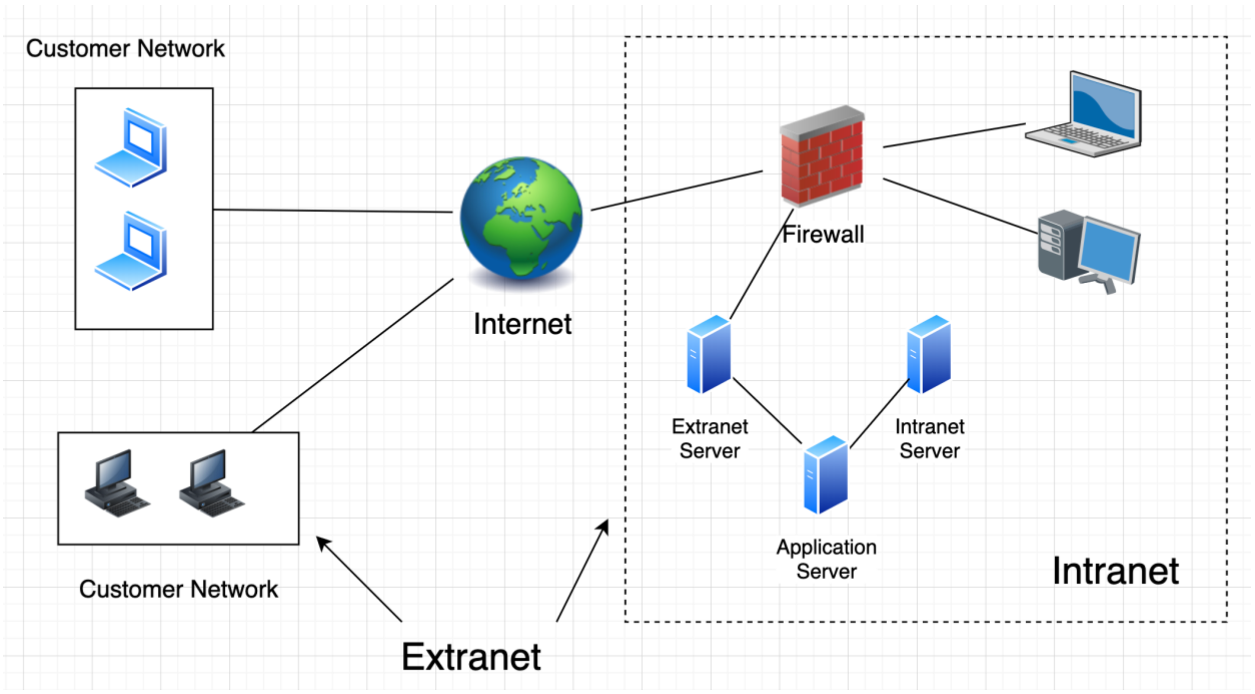


Intranet and extranet support two different areas within a business, but have similar goals: to improve how employees work with clients and each other.

Intranet VS Extranet VS Internet

Simply put:

- Intranet is shared content accessed by members within a single organization
- Extranet is shared content access by groups through cross-enterprise boundaries
- Internet is global communication accessed through the Web



The Internet, extranet, and intranet all rely on TCP/IP technologies. The difference lies in the level of access allowed inside and outside the organization.

Intranet allows organization members to access an extensive amount of resources. Extranet expands the scope of the network by allowing non-members such as vendors and customers to use company resources. Yet, as they are not internal members, they will have limited privileges accessing organization resources. Then, the Internet expands the scope of the network even further to the global facing public. Here the organization is part of the internet and all internet end computers may access its resources, but the privileges will be limited even further, as it is public facing.

LAN Technology Types

Ethernet

Ethernet is the most common type of LAN in use today. It is popular because it strikes a good balance between speed, cost and ease of installation.

A standard Ethernet network can transmit data at a rate up to 10 Megabits per second (10 Mbps). IEEE developed an Ethernet standard known as IEEE Standard 802.3, which defines rules for configuring an Ethernet network and also specifies how the elements in Ethernet network interact with one another.

Fast Ethernet

As the development of technology and performance, there is a need for higher transmission speed and hence an upgraded version of Ethernet – Fast Ethernet (IEEE 802.3u). The speed limit has been raised from 10 Mbps to 100 Mbps with only minor changes to the previous cable structure.

Gigabit Ethernet

Gigabit Ethernet was developed to meet the need for even faster communication with such applications as multimedia. It is defined in the IEEE 802.3 standard and is currently used as an enterprise backbone.

10 Gigabit Ethernet

10 Gigabit Ethernet is the fastest and so far, the most recent of Ethernet standards. IEEE 802.3ae defines a version of Ethernet with a nominal rate of 10 Gbits/s that makes it 10 times faster than Gigabit Ethernet.

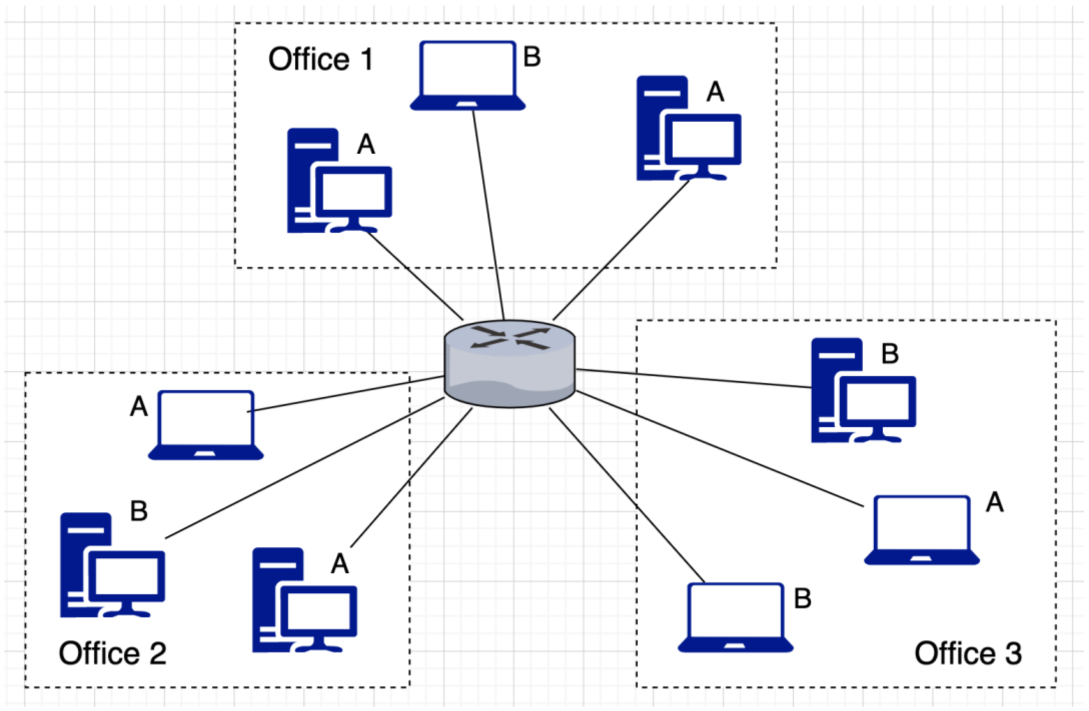
Unlike other Ethernet systems, 10 Gigabit Ethernet is based entirely on the use of optical fiber connections.

This developing standard is stepping away from the LAN design where it broadcasts to all the nodes within the network. Instead, it goes toward a system which includes some elements of wide area routing.

Virtual LAN (VLAN)

VLAN is a solution to divide a single Broadcast domain (or a LAN) into multiple Broadcast domains. Devices in one VLAN domain cannot speak to a device in another VLAN domain.

Let's take a look at the below example.



We have a LAN from an enterprise, which connects office 1, office 2 and office 3. Yet, we have two departments here: A and B. We would like to divide them into two domain groups so that work machines won't talk across departments. Hence, we set up two VLAN with Domain A and Domain B.

VLAN can be configured through network switches.

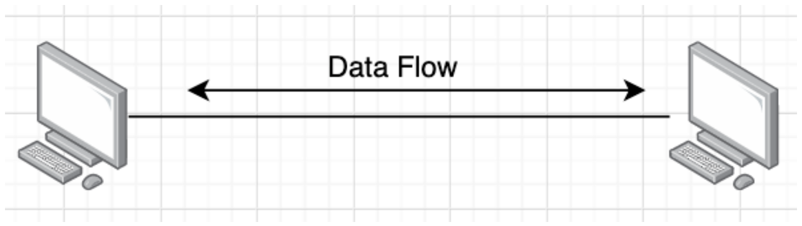
Network Topologies

Network Topology is how devices are arranged and connected in a network. The concept of network topology can be either physical or logical.

Point-to-Point

Point-to-point networks contain just two hosts.

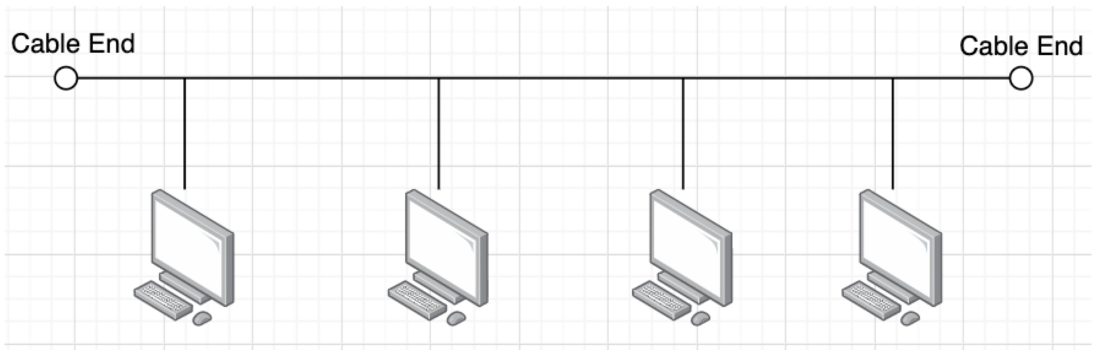
Often, computers or devices are connected back-to-back using a single piece of cable.



Again, this Point-to-Point connection can be either physical or logical.

Bus Topology

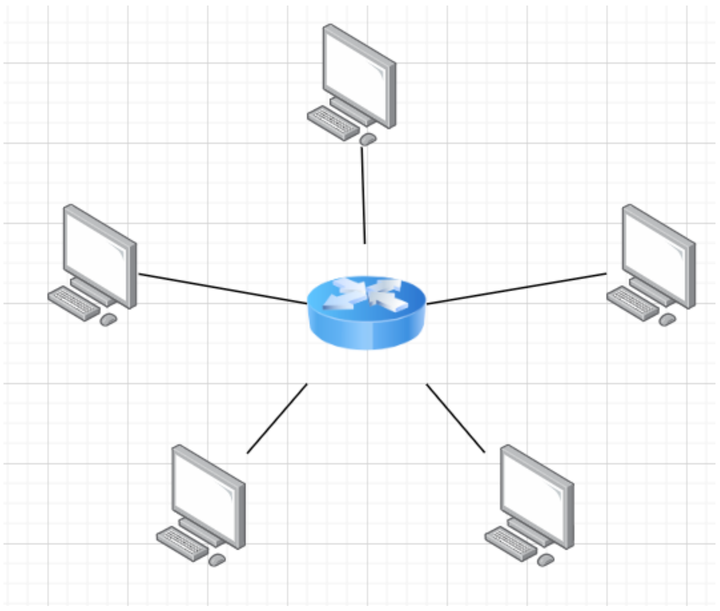
In the Bus topology, all hosts (devices) are connected to a single communication line.



Bus topology is fairly easy to understand and set up. It can be used in small networks. The issue is that as the number of connected hosts increases, the performance will decrease. Furthermore, a cable failure could result in the entire network failure.

Star Topology

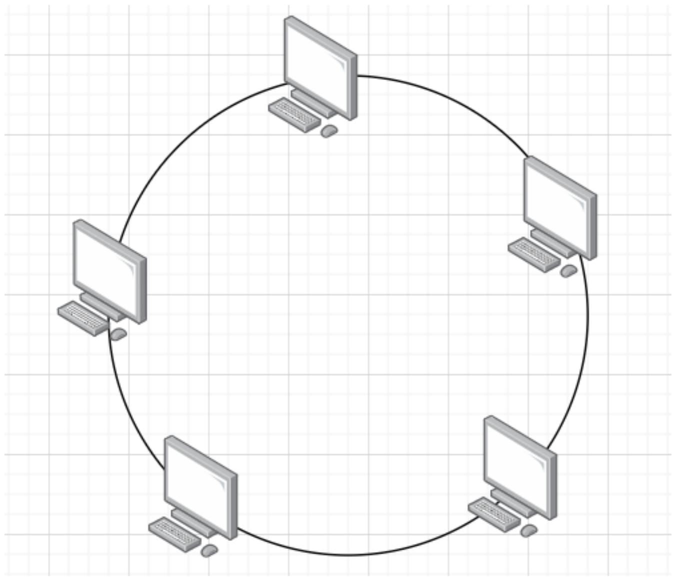
In Star Topology, all hosts are connected to a central host, known as the hub. The connection between hub and end host can be considered as a Point-to-Point connection.



Hub plays an important role in this topology. It has a central network control point which is easier to maintain and troubleshoot. The setup is fairly straightforward. Yet, a premium hub could still be expensive and it's acting as a single point of failure. Performance is also dependent on the capacity of the hub.

Ring Topology

In Ring Topology, each host connects to exactly two other hosts, creating a circular network structure.

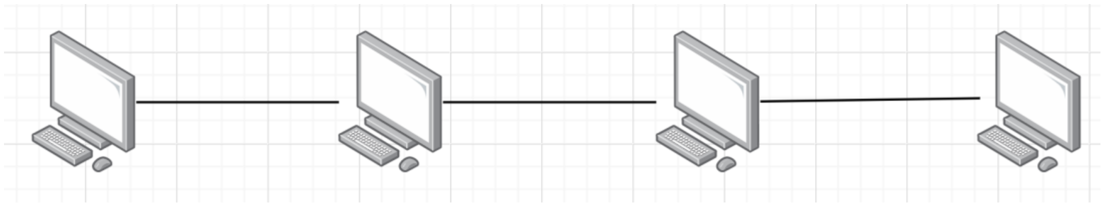


When one host tries to communicate or send a message to a host that is not adjacent to it, the data travels through all intermediate hosts.

Transmitting in ring topology is not quite affected by high traffic or adding more nodes and it doesn't cost a lot to install. Yet, troubleshooting could be problematic and adding or removing a host will disturb the network.

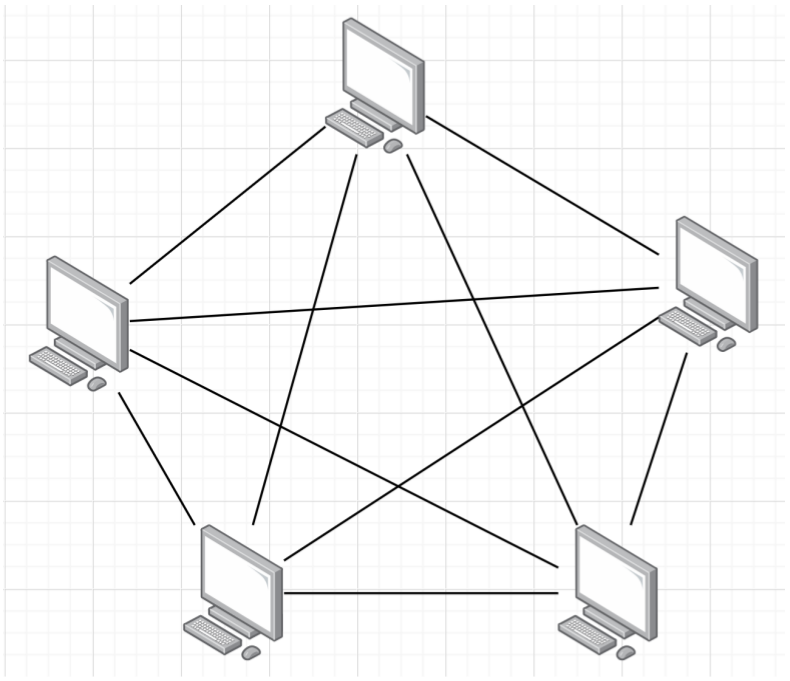
Daisy Chain Topology

This is similar to the Ring Topology, all hosts are connected to the adjacent host, except the two end hosts.



Mesh Topology

In Mesh Topology, one host is connected to one or multiple hosts.



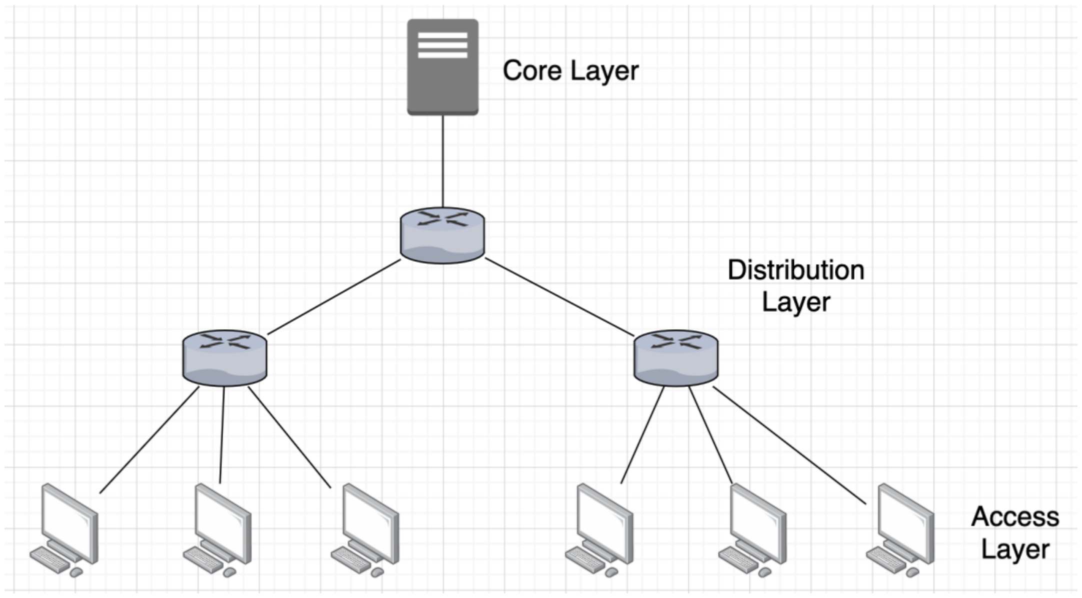
Mesh topology has two types:

- **Full Mesh**: All hosts have a point-to-point connection to every other host in the network. This provides the most reliable network structure among all network topologies.
- **Partial Mesh**: Not all hosts have point-to-point connection to every other host. This means hosts connecting to each other can be in some arbitrary fashion.

For a mesh topology, it is usually robust and easy to troubleshoot. The down side includes difficult setup, higher cable cost and bulk wiring.

Tree Topology

Tree Topology is also called Hierarchical Topology. It is the most common form of network topology in use so far.



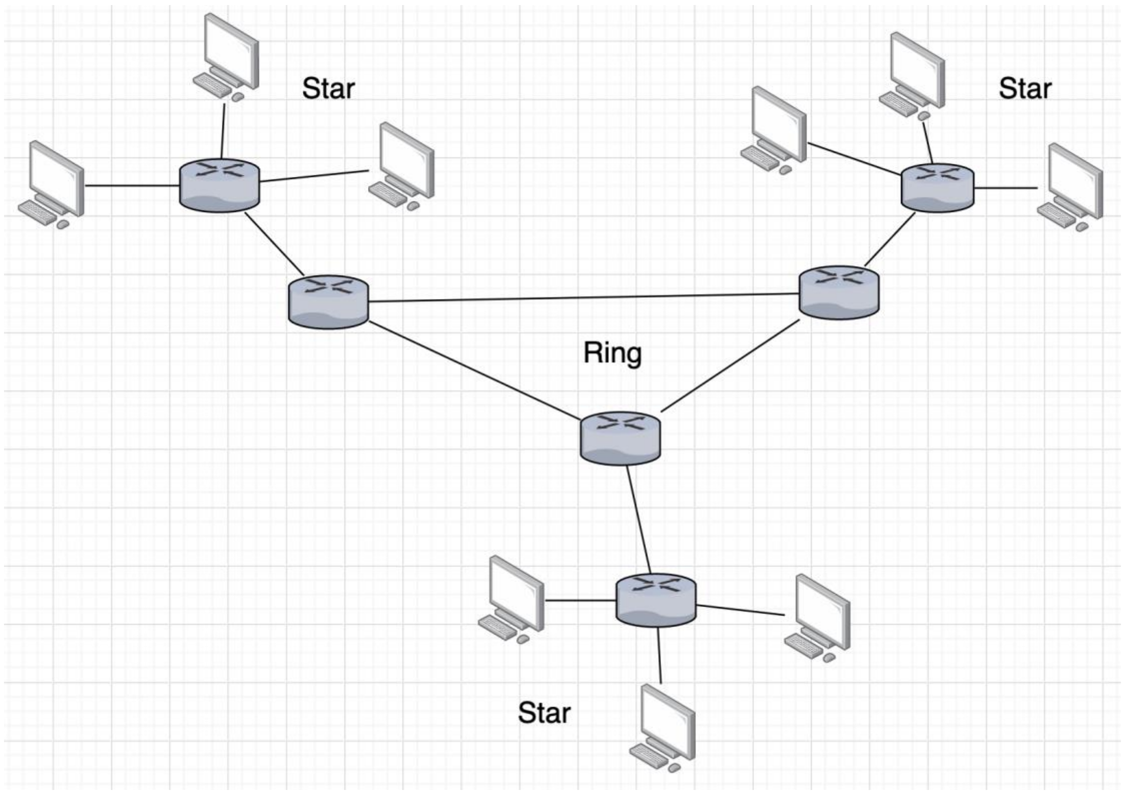
This topology divides the network into multiple layers.

- The root of the tree is the core layer, which serves as the central point of the network.
- The middle layer is the distribution layer, which works as mediator between upper layer and lower layer
- The lower layer is the access layer, where computers are connected

The good side of a tree topology is easy node addition/deletion, easy management and troubleshooting. The downside would be high cost, the core layer acts as a single point of failure.

Hybrid Topology

A Hybrid Topology consists of more than one topology.



A hybrid topology usually inherits the merits from the prime topologies. The above diagram shows the combination of Ring Topology and Star Topology.

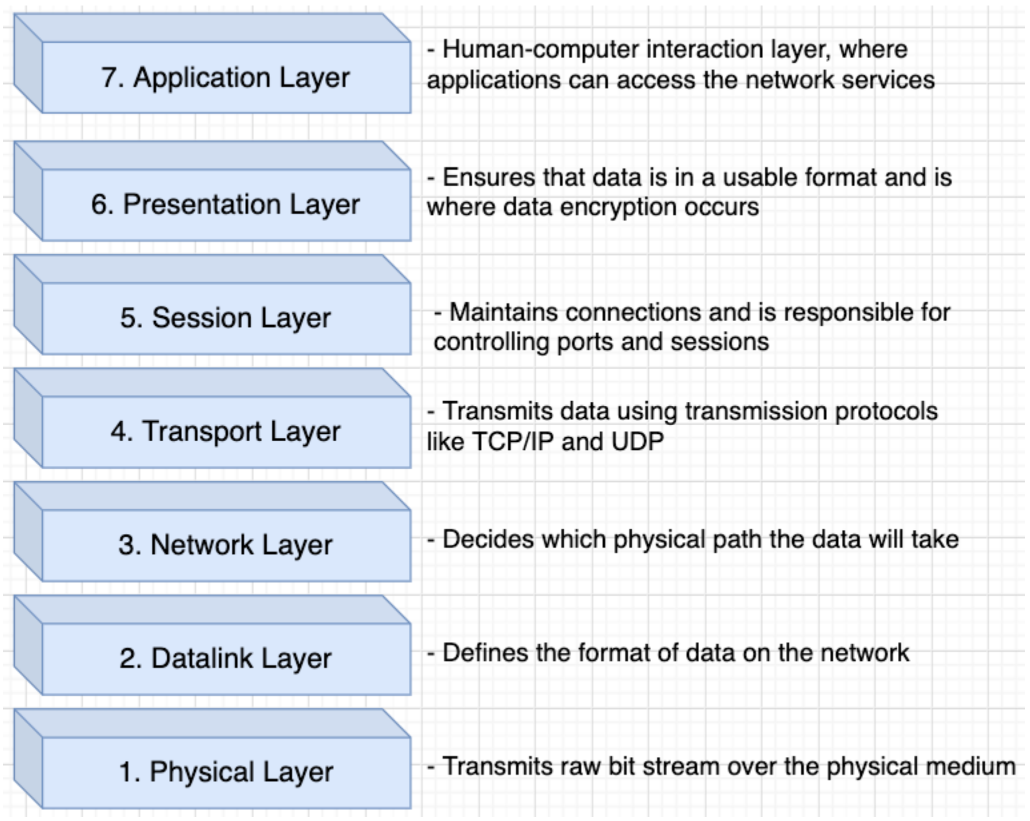
The Internet can be considered as the largest Hybrid Topology.

OSI Model

OSI (Open Systems Interconnection) is a conceptual framework for how applications communicate over a network. The OSI model can be seen as a universal language for computer networking.

The model partitions the system into 7 different abstract layers and each layer performs a particular network function as well as communicating with the layers above and below itself.

Although the modern Internet doesn't strictly follow the OSI model, the OSI model is still very useful as a reference for understanding network behaviors. For example, if we encounter a web app server down, the OSI model can help to break down the problem and isolate the source of the trouble. If we can narrow down to one specific layer of the model, a lot of unnecessary work can be avoided.

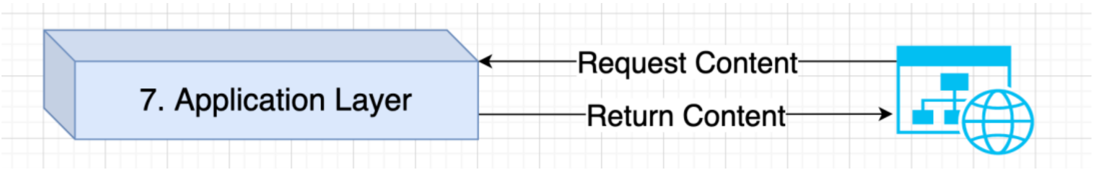

7. Application Layer
- Human-computer interaction layer, where applications can access the network services
6. Presentation Layer
- Ensures that data is in a usable format and is where data encryption occurs
5. Session Layer
- Maintains connections and is responsible for controlling ports and sessions
4. Transport Layer
- Transmits data using transmission protocols like TCP/IP and UDP
3. Network Layer
- Decides which physical path the data will take
2. Datalink Layer
- Defines the format of data on the network
1. Physical Layer
- Transmits raw bit stream over the physical medium

Let's take a look at each layer more closely from top to bottom.

Application Layer (Layer 7)

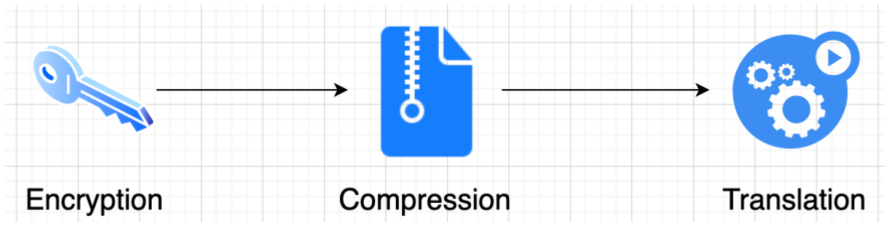
This is the only layer where the user directly interacts with data.

Web applications accessed by browser or email client rely on Layer 7 for communication, though browsers and clients are NOT part of the application layer. The application layer is more responsible for the protocols and data transmission. Application layer protocols include HTTP, SMTP etc.



Presentation Layer (Layer 6)

Layer 6 is primarily responsible for preparing data so that it can be used by Layer 7 (Application Layer). That is to say, Layer 6 makes the data presentable for applications to consume. This layer is responsible for data translation, encryption and compression.



If two communicating nodes are using different encoding methods, Layer 6 will translate incoming data into proper syntax that the receiving node can understand.

If the communication is over an encrypted connection, Layer 6 will encrypt data before sending and decrypt data after receiving.

Layer 6 can also help to compress data for transmission which improves the speed and efficiency of communication.

Session Layer (Layer 5)

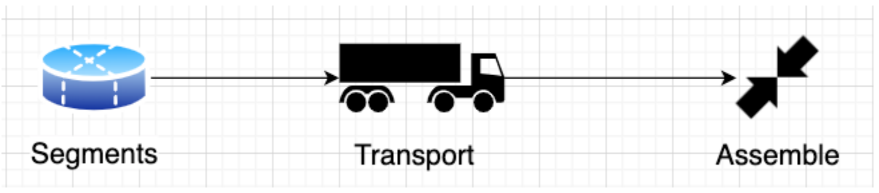
Layer 5 is responsible for opening and closing communication between two nodes.

When a connection is established between two nodes, a session is created. This session will have its duration which will depend on various factors. Layer 5 ensures the session stays open long enough to transfer all the data being exchanged, and then closes the session in a timely manner.

Session
Channel Established

Transport Layer (Layer 4)

Layer 4 deals with data transport. This includes taking data from Layer 5 and breaking it up into chunks (segments) before sending it to Layer 3. The transport layer on the receiving end will reassemble the segments into original data where Layer 5 can consume.

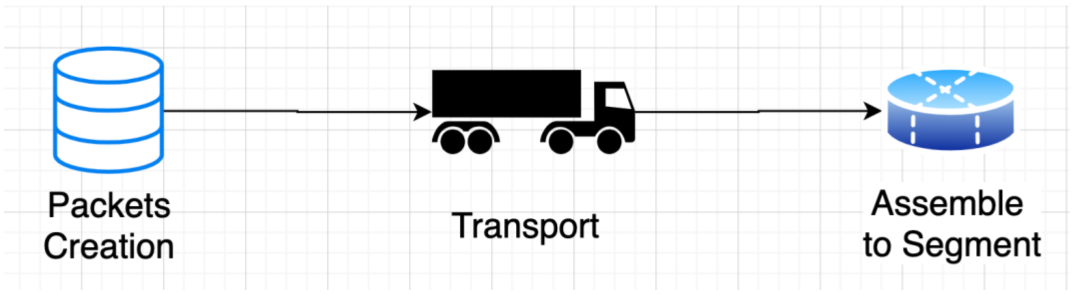


Layer 4 is also responsible for flow control and error control. For example, a fast connection shouldn't overwhelm a receiver which has a slow connection; the data assembled should be complete on the receiving end.

Network Layer (Layer 3)

Layer 3 is responsible for facilitating data transfer between two different networks. This layer is unnecessary if the two nodes are communicating in the same network.

Layer 3 breaks up segments from Layer 4 into smaller units, called "packets", on the sending node and then routes and transports them to the receiving end, where those packets are assembled.

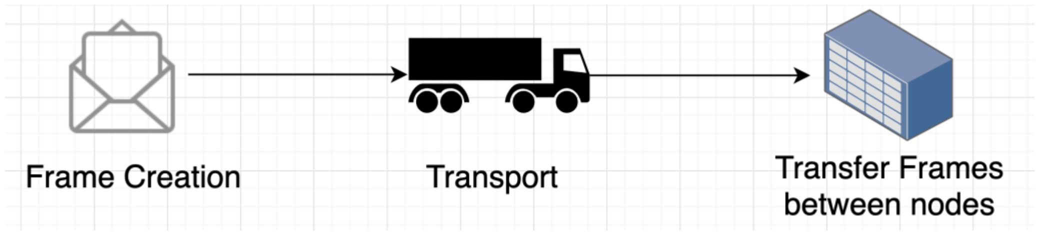


Data Link Layer (Layer 2)

Layer 2 is similar to Layer 3, except this layer facilitates data transfer between two nodes within the SAME network.

Layer 2 takes packets from Layer 3 and breaks them into smaller pieces called "Frames".

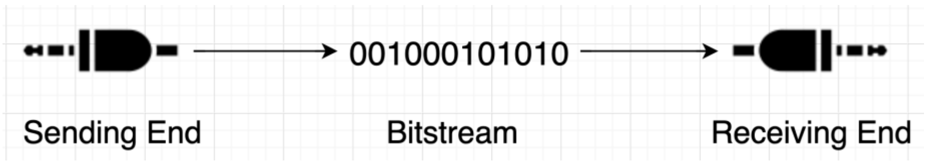
Similar to Layer3, the data link layer is responsible for flow control and error control in intra-network communication.



Physical Layer (Layer 1)

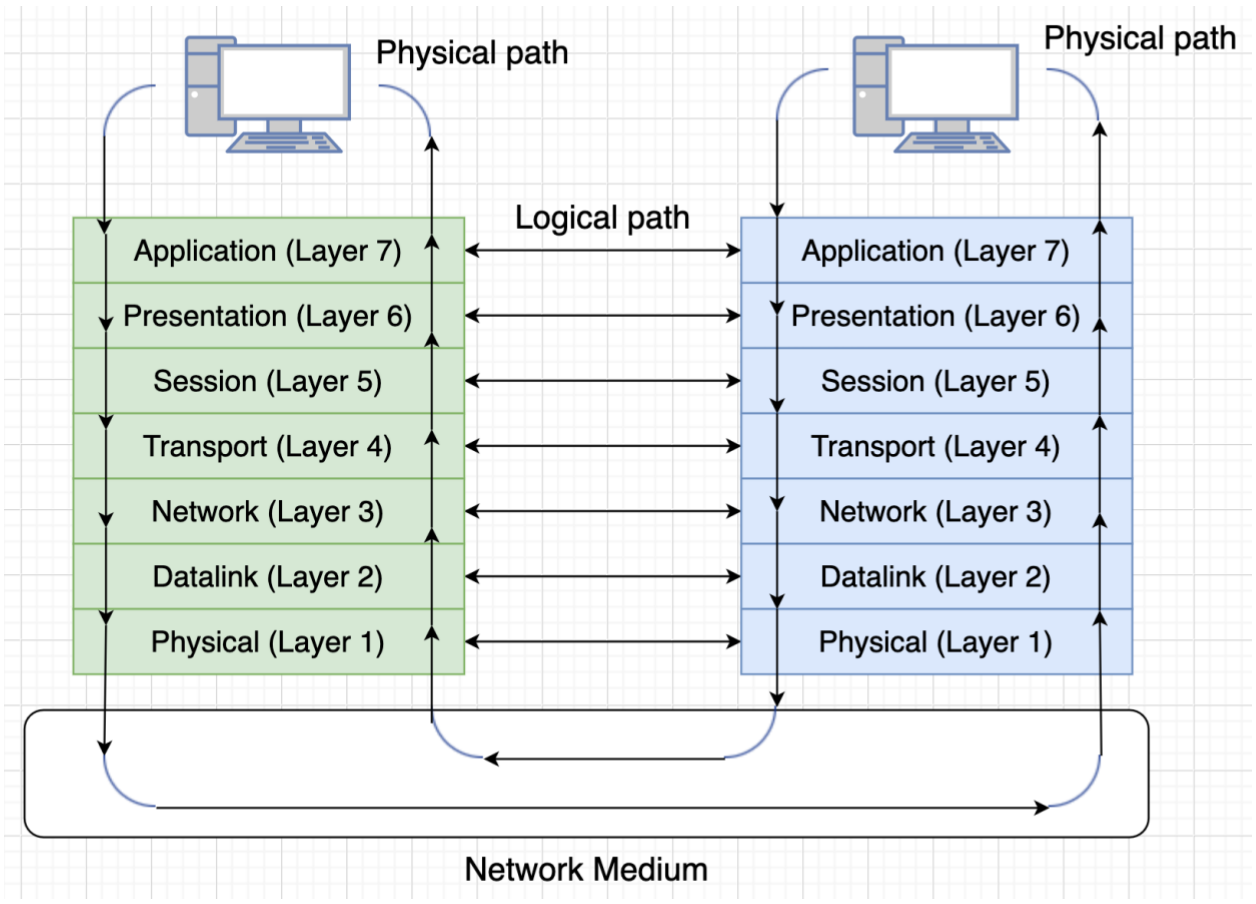
Layer 1 refers to the physical equipment involved in the data transfer, such as cables and switches.

With layer 1, data will be converted into streams of bits, which are strings of 1s and 0s.



How Data Flows through the OSI Model

We know the layer for user computer interaction is layer 7 – the Application layer; however, in order for data to travel from one node to another in the network, the data needs to go through layer 1 – the Physical layer. In this case, data will travel the 7 levels from layer 7 to layer 1 and then from layer 1 to layer 7, as shown in the diagram.



Let's take a look at an example. John wants to send an email to William.

Sender operations start with John composing his message in an email application and clicking the "send" button.

- The email application will pass the email message over Layer 7 (Application Layer) with SMTP (Simple Mail Transfer Protocol) and transfer the data to Layer 6 (Presentation Layer).
- Layer 6 compresses the data and sends to Layer 5 (Session Layer)
- Layer 5 initializes the communication session with the receiver
- Then data will come to Layer 4 (Transport Layer) where the data will be segmented and passed to Layer 3 (Network Layer)
- Layer 3 breaks down segments into packets and passes to Layer 2 (Data Link Layer)
- The data segments will be broken down even further into frames in Layer 2 and then sent to Layer 1 (Physical Layer)
- Layer 1 converts the data into a bitstream of 0s and 1s and send them through a physical medium, e.g. network cable to the receiver

Receiver operations starts with William's computer receiving the bitstream through a physical medium, e.g. WIFI; the data will flow through the same series of 7 layers on his device, but in the opposite order

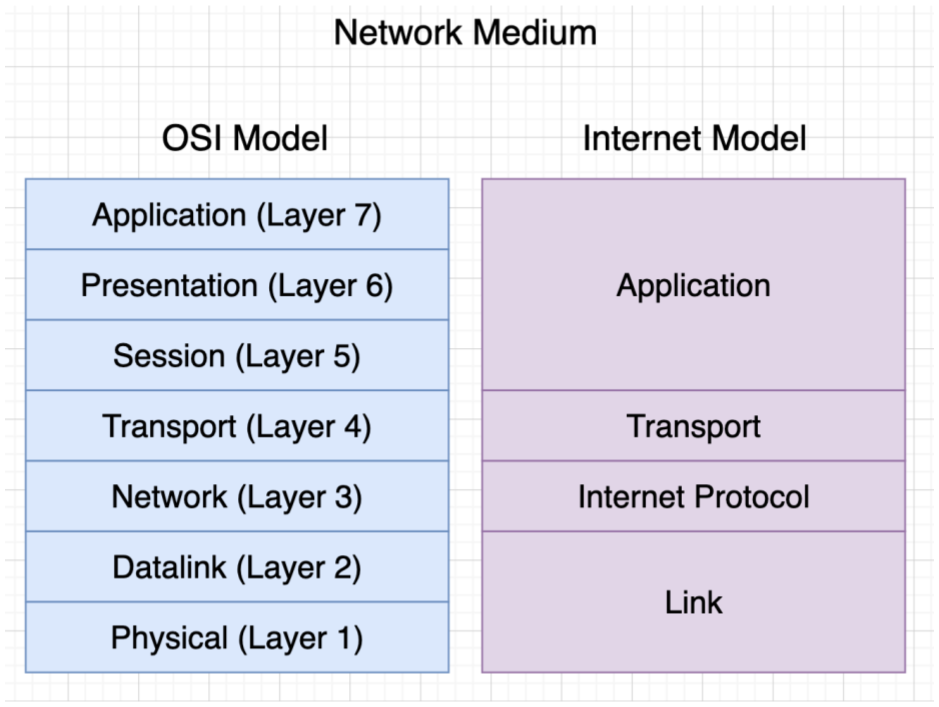
- Layer 1 will convert the bitstream from 0s and 1s to frames and pass to Layer 2
- Layer 2 reassembles frames into packets for Layer 3
- Layer 3 will create segments out of the packets and pass to Layer 4
- Layer 4 will assemble the segments into one piece of data and pass to Layer 5
- Layer 5 will pass the data to Layer 6 and end the connection
- Layer 6 then removes the compression and pass the raw data to Layer 7
- Layer 7 then feed the human-readable data to William's email application, which allow him to read John's email

Next, let's take a look at the Internet Model, which dominates the world right now.

Internet Model (TCP/IP Model)

The OSI model is good at letting people concentrate on the specifics of a network implementation. Yet, implementation based directly on the OSI model is not that popular, mainly due to its complexity – Sticking rigidly to the layers and their isolation will add extra work which is sometimes unnecessary.

The Internet Model, or the TCP/IP model (as the Internet is based on TCP/IP), is a 4-layer model instead of 7. Compared to the OSI model, TCP/IP model is a simpler, easy-to-understand, more practical proposition.



- **Application Layer**: defines the protocol which enables user interaction with the network, e.g. HTTP, FTP, SMTP etc.
- **Transport Layer**: defines how data should flow between hosts. The major protocol at this layer is TCP (Transmission Control Protocol). This ensures that the data delivered is in correct order.
- **Network Layer**: IP (Internet Protocol) works on this layer. The functionality of this layer is host addressing and recognition. It also defines routing.
- **Link Layer**: sending and receiving actual data. Unlike the OSI model, the Link Layer here is independent of underlying network hardware and architecture.

Chapter 6 – Computer Network: A Further Look

In this chapter, we will continue exploring some more important concepts of networks.

Key Points

- **DNS:**
  - Translates domain names to IP addresses.
  - Works through a hierarchy of servers (local cache, recursive, root, TLD, authoritative).
  - DNS records provide instructions to servers.

- **IP Addresses:**
  - Unique identifiers for devices on a network.
  - IPv4 (32-bit) and IPv6 (128-bit).
  - Classes (A, B, C, D, E) and subnetting.
  - CIDR (Classless Inter-Domain Routing) for efficient address allocation.

- **Domain Names:**
  - FQDN (Fully Qualified Domain Name): Complete domain name (e.g., www.google.com).
  - URLs: Web addresses (absolute and relative).

- **TCP/IP:**
  - TCP: Connection-oriented, reliable, ordered data transmission.
  - IP: Addressing and routing packets.
  - TCP 3-Way Handshake: Establishes connection.

- **UDP:**
  - Connectionless, faster but less reliable than TCP.
  - Suitable for time-sensitive applications (e.g., video streaming).

- **Routing:**
  - Routers forward packets between networks.
  - Routing types: Static and dynamic.
  - Routing schemes: Unicast, broadcast, multicast, anycast.
  - Routing protocols: OSPF, BGP, IGRP, EIGRP, EGP, RIP.

- **VPN:**
  - Creates a secure, private network over a public connection.
  - Benefits: Privacy, anonymity, security.
  - Scenarios: Remote access, site-to-site.
  - Tunneling: Encapsulates and encrypts data.

- **Application Layer Protocols:**
  - SMTP: Email transfer.
  - MIME: Extends SMTP for various data types.
  - HTTP: Web communication.
  - RADIUS: Centralized authentication.
  - FTP: File transfer.
  - DNS: Domain name resolution.

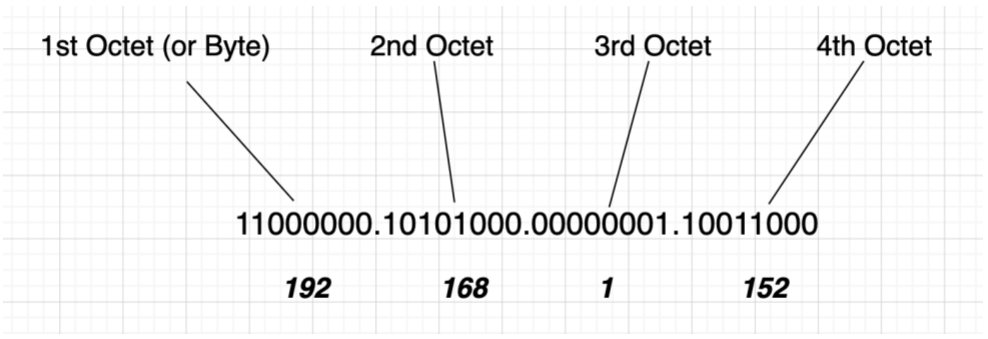
Internet Domain Name System

DNS (Domain Name System) is a naming database where the domain names are located and translated into IP (Internet Protocol) addresses.

IP Address

Computers in a network use IP address to find where they are in the network. It is a unique logical address assigned to a machine over the network:

- IP address is the unique address assigned to each host present on internet
- IP address (IPv4) is 32 bits (4 bytes) long
- IP address consists of two parts: **Network Part** and **Host Part**
- Each of the 4 bytes is represented by a number from 0 to 255, separated with dots, e.g. 192.168.1.152



IP addresses are difficult for people to memorize, so DNS is introduced to resolve this issue. For example, when we want to visit Google, we don't type in its IP address in the browser, instead we type in its domain name google.com into the browser URL address field. That domain name will be resolved by DNS servers into its corresponding IP address so that your computer knows where to find Google's server on the Internet.

Types of IP Addresses

IP addresses can be classified into two categories: **Static IP addresses** and **Dynamic IP addresses**.

Static IP address – As its name implies, the static IP address usually doesn't change unless a network administrator changes it. They serve as a permanent Internet address and provide a simple and reliable way for communication.

Dynamic IP address – Dynamic IP addresses are temporary IP addresses. These IP addresses are assigned to a computer when they get connected to the Internet each time. These are borrowed from a pool of IP addresses, and are shared across various computers.

IPv4 and IPv6

When we say IP address, we are referring to IPv4. It is widely used and accounts for up to 90% of Internet traffic.

IPv6 is the most recent version of Internet Protocol. It was initiated in 1994 by IETF (Internet Engineer Task Force) to replace IPv4 and is being deployed to fulfill the need for more internet addresses.

IPv6 addresses are 128-bit IP address written in hexadecimal and separated by colons, e.g. **3fae:1d00:4525:31:200:f8ff:fe21:67cf**

While increasing the pool of addresses is the most obvious benefit of IPv6, there are other important technological benefits as well:

- No more NAT (Network Address Translation)
- Auto-configuration
- No more private address collisions
- Better multicast routing
- Simpler header format
- More efficient routing
- Built-in authentication and privacy support
- Easier administration

Since the appearance of IPv6, we are starting to migrate to it from IPv4. Yet, the progress is slow. Legacy technologies take a long time to die off and this is due to a list of different factors. It is the future and it will happen, but just very slowly.

Classes of IP Address

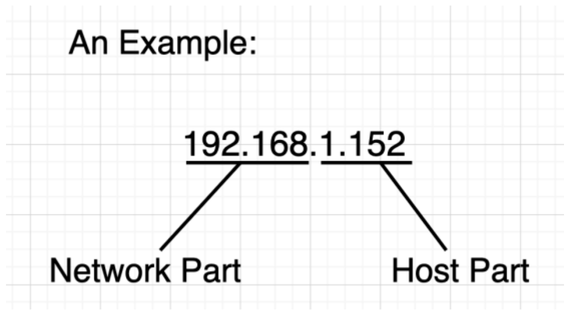
The original designers of the IP stack came up with a hierarchical addressing scheme with five classes, from Class A to Class E. Each class has a range of IP addresses.

- Class A: IP range from 0-127 in the first byte, designed for very large companies
- Class B: IP range from 128-191 in the first byte, designed for medium-size companies
- Class C: IP range from 192-223 in the first byte, designed for small companies
- Class D: IP range from 224-239 in the first byte, reserved for multicast addressing, Not used in public sector
- Class E: IP range from 240-255 in the first byte, reserved for scientific studies, Not used in the public sector

Note this addressing scheme based on Class is being replaced by CIDR (Classless Inter-Domain Routing) which has an IP address like 192.168.5.152/24. We'll look at it later.

Network Part and Host Part

IP address bytes can be divided into two parts: Network Part and Host Part.



Network Address is used for routing on the Internet and once the data packets have arrived at the local network, Host Address will be used.

Network part specifies the unique number assigned to your network. Host part is the IP address that is assigned to each host. The bits where they are separated are dynamic and can depend on the IP class.

This can be considered as a local network connected to the Internet and this network is assigned a unique number to be identified in the Internet. Then this number forms the network part of each host's IP address within the network.

Subnetting (Subnetwork)

If you work at a large organization or business, especially one spread across multiple branches, it's quite possible that it deploys a subnet.

Subnetting, or subnetworking, is the process of splitting a single large network into two or more smaller ones.

Why do we need subnetting? Within the IP system, even a target network can be found, it could still be difficult to route a data packet to the right machine to that network. This becomes even more difficult when networks reach a large scale. By breaking the network up into smaller parts, subnets help alleviate this network congestion.

An organization can use IP subnets for logical reasons (firewall, etc.) or physical requirements (small broadcast domains, etc.). Routers use subnets to make routing choices.

Subnetting can improve network security. With subnets, security incidents can be better contained.

Subnet Mask

A subnet mask is selected by a network administrator to divide a network into smaller subnetworks.

For example, a common subnet mask is 255.255.255.0 which in binary is "1111 1111.1111 1111.1111 1111.0000 0000". Note a subnet mask will always have a certain number of zeros at the end, which leaves a range of IPs that can be assigned. The example subnet mask has 8 binary bits of 0 at end and this provides 2^8=256 possible IP addresses in the subnetwork. If the administrator uses 255.255.255.127 as a mask, which in binary is "1111 1111.1111 1111.1111 1111.1111 0000". This will allow 2^4=32 IP addresses in the subnetwork.

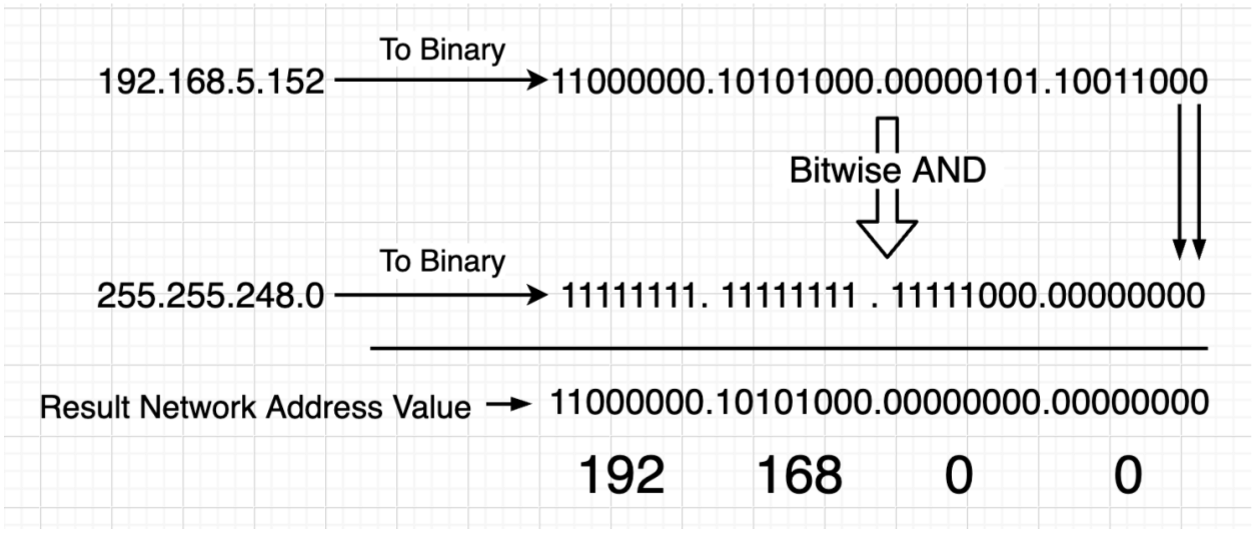
Calculate Network Address and Host Address

Suppose we have a host IP 192.168.5.152 and the network has a mask of 255.255.248.0.

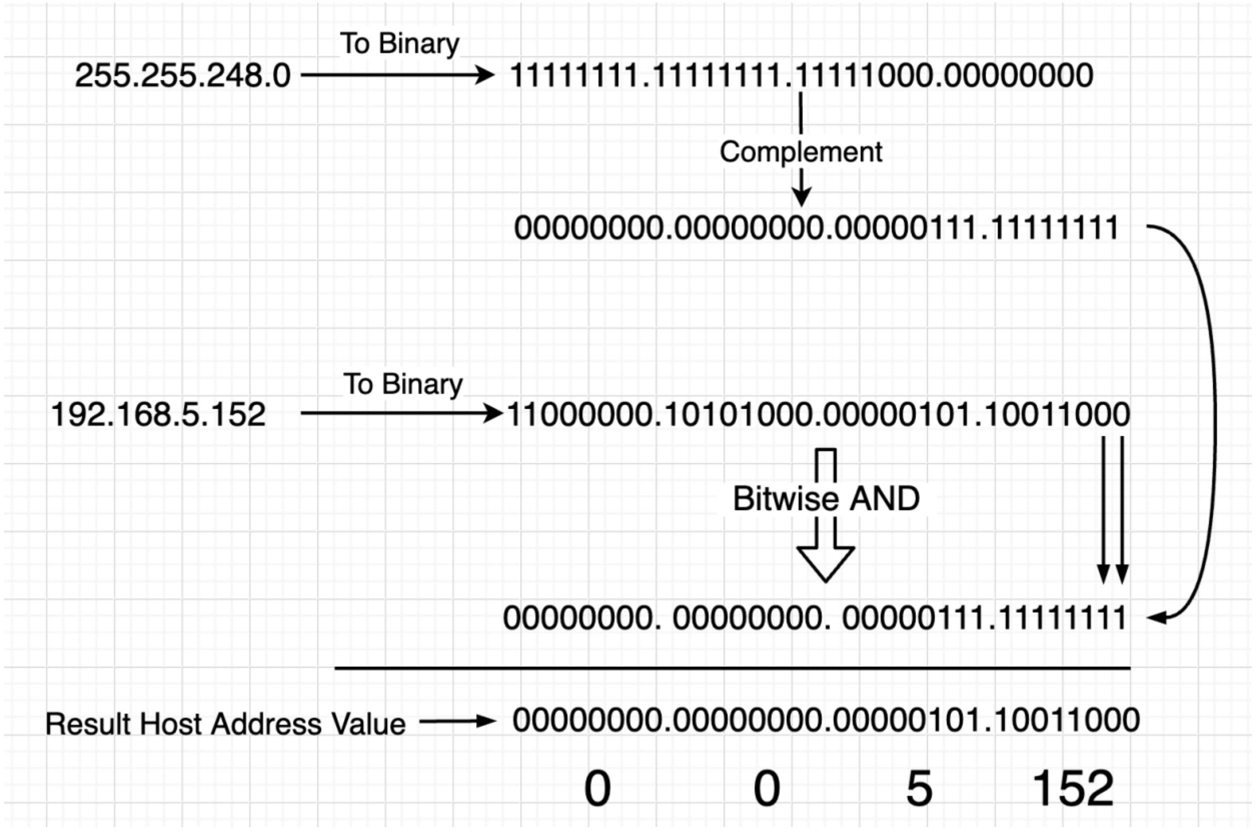
First, let's calculate the allowed IP range with this subnet mask for some fun. Converting the subnet mask to binary gives us 11111111.11111111.11111000.00000000. So, the allowed IP range is from 00000000.00000000.00000000.00000000 to

00000000.00000000.00000111.11111111, which in decimal is 0.0.0.0 to 0.0.7.255. That is 256*8 = 2048 available IP addresses.

Next, to calculate the Network address, we first convert the subnet mask to binary. After that, we convert host IP to binary and do a bitwise AND operation. The result binary is the Network address.



Then, we calculate the Host address, we first convert the subnet mask to binary and Apply binary complement to it. After that, we convert host IP to binary and do a bitwise AND operation. The result binary is the Host address.



CIDR (Classless Inter-Domain Routing)

CIDR is an IP addressing scheme to replace the old system based on classes A, B and C. CIDR is for subnetting just like Class-based scheme, but with a better efficiency.

Problem with Class-based IP Addressing

The old class-based IP addressing came with inefficiencies that exhaust IPv4 addresses faster than it needs to.

- Class A – Over 16 million host identifiers
- Class B – 65,535 host identifiers
- Class C – 254 host identifiers

If an organization has 1000 host machines, it can't go for Class C license and will probably use a Class B license. In this case, a very large number of IP addresses are wasted. This will become a problem when a lot of organizations are doing it. To solve this issue, CIDR is introduced and it is based on a variable-length subnet masking scheme.

How CIDR works

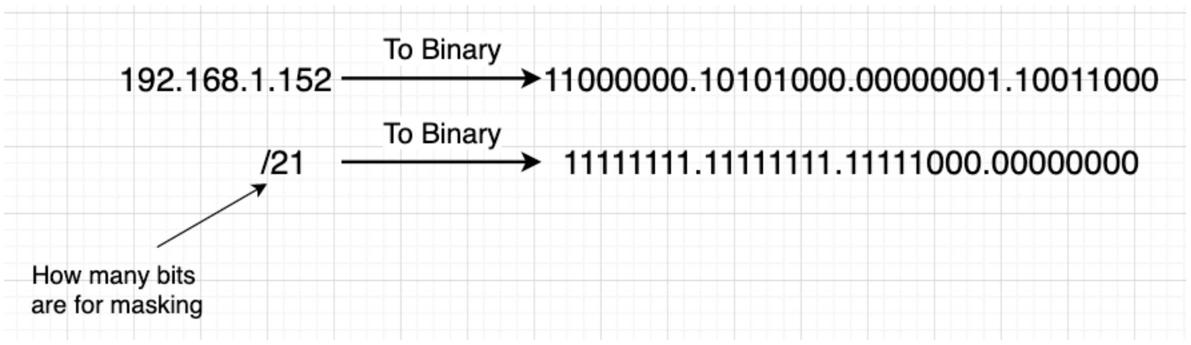
CIDR is based on VLSM (Variable-Length Subnet Masking). This allows it to define prefixes of arbitrary lengths, making it much more efficient than the Class-based system.

CIDR IP addresses consist of two parts. The network part is written as a prefix, like a normal IP address, e.g. 192.168.1.152. The second part is the suffix which indicates how many bits are masked in the address (e.g. /21). Putting it together, it will look like 192.168.1.152/21.

Calculate Network Address and Host Address with CIDR

We can calculate the Network address and Host address with a CIDR IP just like we did previously.

The point is to know the number after slash (/) in the CIDR address representing how many significant bits (starting from left) is value 1.



'/21' here indicates 21 bits of 1s. The remaining part is the same as what we have done previously. We do bitwise AND to find the Network address as 192.168.0.0 and a bitwise AND for mask's binary complement to find the Host address as 0.0.1.152. In other words, '/21' here means 21 bits of the original IP address is the Network address and the other 11 bits is the Host address.

CIDR is a great way to improve the efficiency of IP address distribution. It was vital with IPv4 as IP addresses were quickly being exhausted.

FQDN (Fully Qualified Domain Name)

FQDN is the fullest possible domain name of a host or a computer on the internet. The syntax is:

[hostname].[domain].[tld]

For example, www.google.com, where ".com" is the top-level domain and then it follows the domain name "google", and the last is the hostname "www".

The hostname here can show a specific service or protocol for that domain, such as “mail”, that is mail.google.com.

URL (Uniform Resource Locator)

We talked about URLs in the HTTP chapter. Let’s revisit it here.

URL refers to a web address which uniquely identifies a resource over the internet. This resource can be a web page, image, audio, video or other types presented on the Web.

There are two forms of URL: Absolute URL and Relative URL.

Absolute URL

Absolute URL is a complete address of a resource on the Web. It can comprise four parts: protocol, server name, path name and file name. Note protocol and server name are usually required.

For example, https://en.wikipedia.org/wiki/URL,

- http is the protocol
- en.wikipedia.org is the server name
- /wiki/URL is the path name

The protocol tells the browser how to handle the request. Some other protocols are:

- FTP
- Https
- mailto

Relative URL

Relative URL is a partial address of a webpage, where the protocol and server name are omitted. For example, when we write html code

```
<a href = "/xyz.html">
```

The above is a link to the xyz.html page, which will be in the same directory of your home page.

Absolute URL VS Relative URL

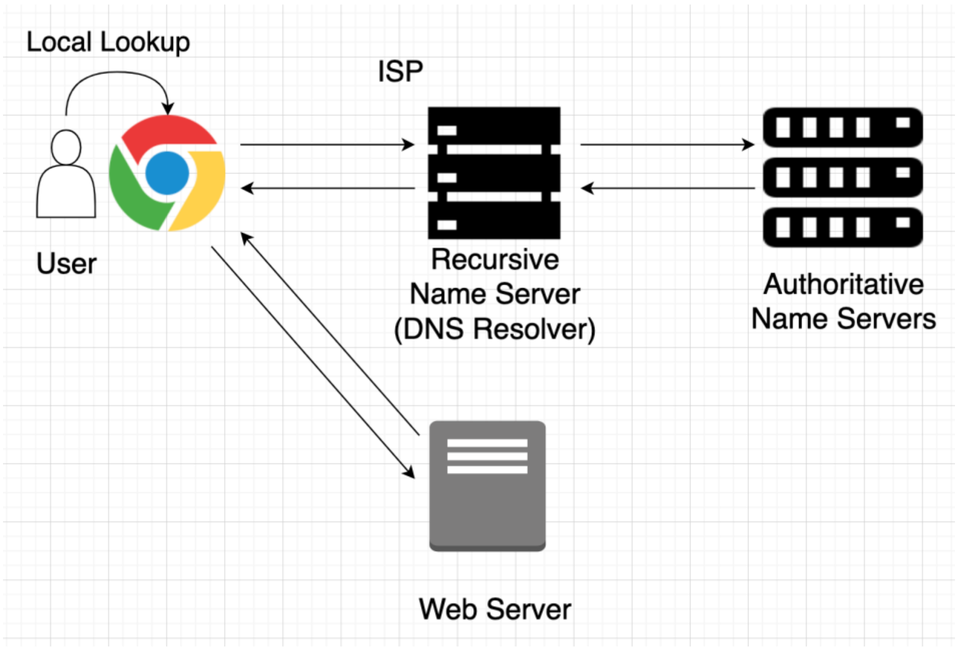
Absolute URL	Relative URL

Link web pages across different websites	Link web pages within the same website
Sometimes difficult to manage	Easier to manage
Need to change when server name changes	No need to change when server name changes

How DNS (Domain Name System) Works

When you visit a domain such as google.com, your computer follows a series of steps to turn a human-readable web address into a machine-readable IP address. This happens each time you use a domain name, whether it's viewing websites, sending email or uploading a file. The whole flow takes a couple of steps and is happening behind the scenes.

Let's take a look at the process.



Step 1: Request information

When you type in a URL in the browser, it will try to resolve the hostname. The first place your computer looks for is its local DNS cache, which stores information that your computer has recently viewed. This DNS cache is a temporary database, and is maintained by a computer's operating system.

If your computer can't find the record for the requested hostname, it will go to the next step – query recursive DNS servers.

Step 2: Ask the Recursive DNS servers

If the information can’t be found locally, your computer queries the recursive DNS servers (DNS resolvers) from your ISP (Internet Service Provider). These recursive DNS servers perform the DNS query on your behalf.

The recursive DNS servers have their own caches which are shared among ISP’s customers, and hence you will have a reasonable chance the domain name you are visiting can be found.

Besides the DNS resolvers run by ISP, some companies provide public DNS resolvers as well, e.g. Google Public DNS, OpenDNS etc.

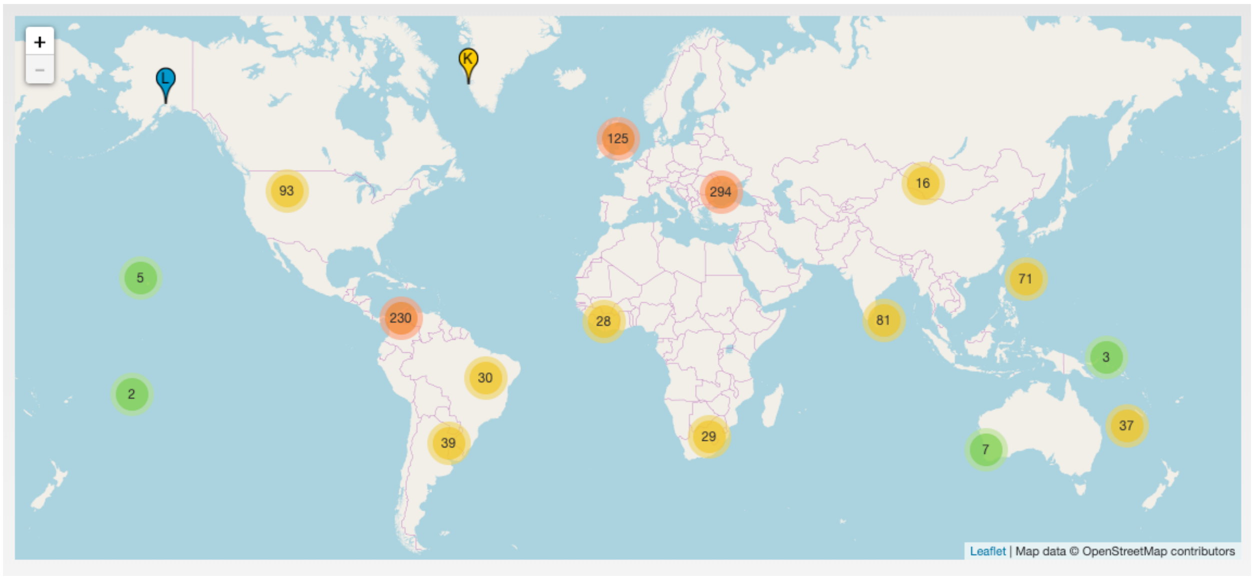
If the recursive servers don’t have the records, it will go to the next step – root name servers.

Step 3: Ask the Root Name servers

If the recursive servers don’t know the answer, the query will happen at the Root Name servers. Root Name servers, or DNS root servers, are name servers that are responsible for the functionality of DNS for the entire Internet. The Root Name servers don’t know the answer directly, but they can direct DNS queries to someone that knows where to find it.

There are 13 Root Name servers in the world. Each of the servers has a bunch of instances to load-balancing the traffic.

Below is a location map from root-servers.org for Root Name servers.



These Root Name servers are operated by 12 organizations (by now) and many of them have been operating those Root Name servers since the creation of DNS.

When the Root Name server receives the query request, it will look at the first part of the request, reading from right to left – google.com, and then direct the query to other servers. In this case, since it is a .com, it will go to TLD (Top-Level Domain) Name servers.

Step 4: Ask the TLD (Top-Level Domain) Name servers

TLD Name servers maintain information for all the domain names that share a common domain extension, such as .com, .net, or other extension name.

For example, a .com TLD Name server contains information for every website that ends in '.com'.

Yet, again, these TLD Name servers still don't have the information we need, but they will point us to the next step – Authoritative DNS servers. To do that, the TLD Name servers review the next part from right to left of the request – google.com, and direct the query to the name servers responsible for this specific domain.

Step 5: Ask the Authoritative DNS servers

Authoritative name servers are responsible for knowing all the information about a specific domain, which is stored in the DNS records. There are different types of records, each of which contain specific kinds of information.

In this case, we want to know the IP address of google.com, so we ask the authoritative name server for the address record ('A' record).

Step 6: Retrieve the record

The recursive server retrieves the 'A' record for google.com from the authoritative name servers and stores the record in its local cache for later use.

All local records have a time-to-live value, which tells when the cached record expires. If the record expires, the recursive servers will fetch a new copy to make sure the information is up-to-date.

Step 7: Receive the answer

Once the recursive server gets the answer, the address record of the request hostname, it returns that information to your computer. Your computer stores it in the computer local cache, reads the IP address from the record, and then passes that information to your browser. The browser then opens a connection to the webserver and reaches the website.

The whole process takes a very short amount of time to complete (in the order of milliseconds).

The job of resolving a hostname is simple: get its IP address; yet, it takes some complexity to finish the work. This is probably due to the large number of servers and registered records from the entire world. This well-designed DNS architecture helps to handle the large volume and its complexity so that we can get the resolved hostname with good performance and reliability.

DNS Records

DNS records act as instructions for the DNS server, so it knows which domain names each IP address is associated with.

DNS records contain different syntax and commands for how the server should respond to the request. Some of the common ones are:

- "**A**" Record: this record refers to the actual IP address that's associated with the domain; e.g. "74.125.224.147"
- "**AAAA**" Record: this one operates the same as "A" record, except the record points to an IPv6 address
- "**CNAME**" Record: this record indicates subdomains that might be listed under or associated with your current domain; for example, you can link blog.example.com with a CNAME record to an A record set on example.com, and they would both point to the same server
- "**MX**": this record refers to any mail servers that might be used in accordance with your domain; for example, example.com can be linked to mail.example.com through MX record, and then mail.example.com has a A record of 98.76.54.321
- "**SOA**": this record has crucial information about your domain, such as when your domain was last updated and relevant contact information
- "**TXT**": this can be edited to include any additional information about the domain that isn't currently listed

TCP/IP (Transmission Control Protocol / Internet Protocol)

We know there are vast numbers of users communicating using various devices in different languages, which also includes various ways of transmitting data along with different software implementations. That being said, if there are no pre-agreed 'standards' that govern the way users communicate, communicating world-wide would not be possible.

TCP/IP are the dominating transport and network layer protocol in the Internet today. Let's take a look at it first.

TCP and IP are actually two different protocols, but most of the time they are used together.

TCP/IP is used as the basic communication protocol suite for the Internet. Sometimes, TCP/IP can also be used as a communication protocol in a private network (intranet or extranet).

TCP defines how applications can create channels of communication across a network. It also manages how a message is divided into smaller packets before they are transmitted over the internet and reassembled in the right order at the destination.

IP defines how to address and route each packet to make sure it reaches the right destination. IP is the primary way in which network connections are made, and it establishes the basis of the Internet. IP does not handle packet ordering or error checking, as they are handled by TCP.

Common Types of TCP/IP

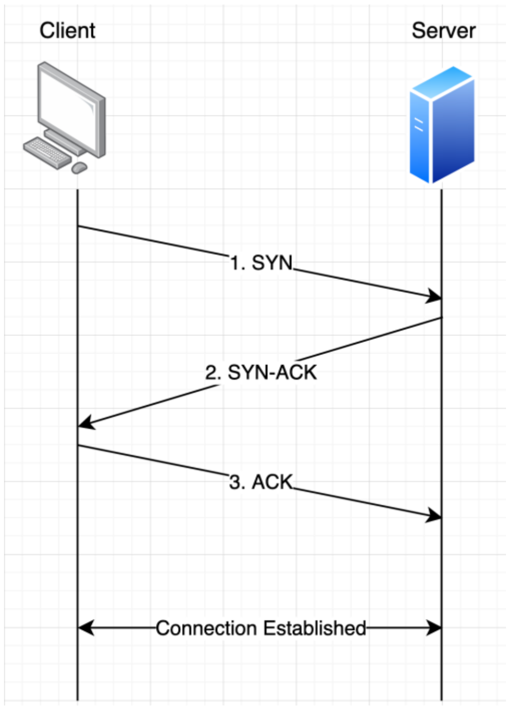
Common types of TCP/IP include:

- **HTTP** (HyperText Transfer Protocol): handles communication between a web server and a web browser
- **HTTPS** (Secure HTTP): handles secure communication between a web server and a web browser
- **FTP** (File Transfer Protocol): handles transmission of files between computers

TCP/IP are at Transport Layer and Network Layer while HTTP, HTTPS and FTP are at Application Layer.

TCP 3-Way Handshake

In order for a client to communicate with a server, a connection needs to be established. This is usually known as the TCP 3-Way Handshake.



To open a connection:

1. The client sends an SYN ‘initial request’ packet to the target server to start the dialogue
2. The target server then sends a SYN-ACK packet to agree to the process
3. The client sends an ACK packet to the target to confirm the process and the connection is established

After a connection is established, message content can be sent.

TCP Features

Here are some of the TCP features:

- TCP is reliable. As the receiver always sends positive or negative acknowledgement about the data packet to the sender, the sender always knows whether the data has reached the destination. If not, the sender can resend it.
- TCP ensures the data reach the destination in the same order it was sent
- TCP is connection oriented. TCP requires that connection between two remote points be established before sending actual data.
- TCP provides error-checking and recovery mechanism
- TCP provides end-to-end communication
- TCP provides flow control
- TCP operates in Client/Server point-to-point mode

- TCP provides full duplex server, i.e. an endpoint can play both receiver and sender

UDP (User Datagram Protocol)

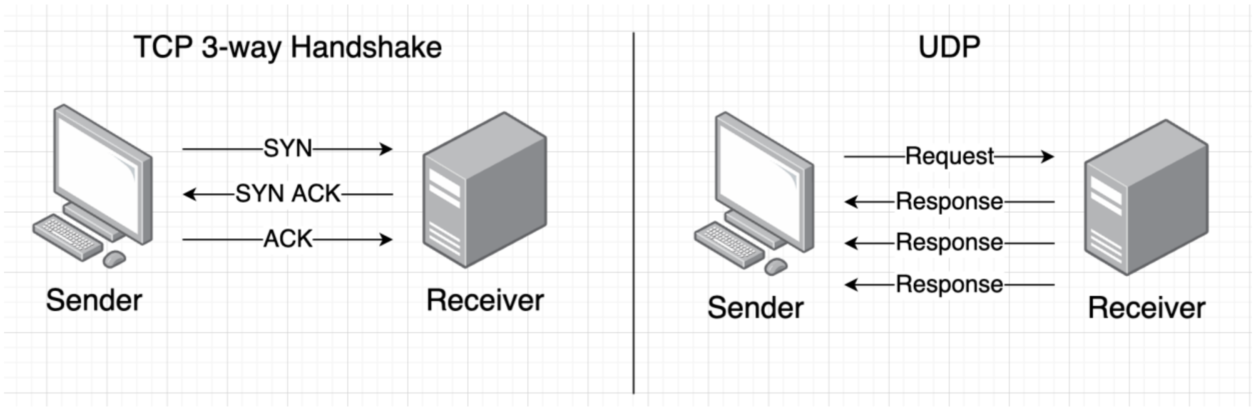
UDP is a communication protocol used across the Internet for time-sensitive transmissions such as video playback or DNS lookups. Compared to TCP, it is faster by not requiring a 3-way handshake, which allows data to be transferred before the receiving party agrees to the communication.

UDP doesn't have the error checking and ordering functionality as of TCP. In other words, the data transmission is not as reliable as TCP. If some packets are dropped due to any network reason, those dropped data won't be resent. While this sounds less ideal, there are certain situations 'timing' is preferred over 'reliability'.

When we say UDP is commonly used where occasionally dropping packets is more important than waiting, it is based on the fact that the data will be transferred to the destination to a very large extent and only a very small portion of data is dropped. There wouldn't be any sense if we have dropped, say, half of the data during transmission (well, you might find some extreme cases).

As an example, video streaming can be based on UDP. With limited network bandwidth, we would prefer a fluent video playback with some pixel glitch to crystal clear image but dropping frames.

TCP VS UDP



Connection

TCP is a connection-oriented protocol; UDP is a connectionless protocol. TCP establishes a connection between a sender and receiver before data can be sent. UDP does not need to establish a connection to send data.

Reliability

TCP is reliable. Data sent over TCP is guaranteed to be delivered to the receiver. TCP will check packets for errors and track packets so that data is not lost or corrupted. UDP is relatively not reliable in this sense, as it does not provide guaranteed delivery.

Flow Control

TCP uses a flow control mechanism to ensure a sender is not overwhelming a receiver by sending too many packets at once. UDP does not provide flow control.

Ordering

TCP does ordering and sequencing to guarantee that packets sent will be delivered to the client in the same order they were sent. UDP sends packets in any order.

Speed

TCP is slower than UDP as the former does more things.

Usage

TCP is commonly used for applications that require high reliability where timing is less of a concern:

- HTTP
- SSH
- FTP
- SMTP

UDP is used for applications that requires speed

- Video Streaming
- Live Broadcast
- VoIP (Voice over IP)

Routing

Data is transferred in the form of packets. A router is a network layer device and will have the ability to create multiple paths for data transmission and choose the best route.

Router connects two or more networks which can be either nearby or very far away and hence, it works with LAN, MAN, WAN or Internet. Routers come in different sizes and functions. Small routers are used in homes and small businesses. Mid-sized routers are used to connect several buildings and large routers are used in big organizations.

Router Functions

The router features depend on its intelligent level and may include:

- Transfer data to destination by selecting the best path
- Connects different types of networks
- Shares information with other routers in the network
- Contains routing table that keeps track of the routes to networks

Routing Type

There are mainly two types of routing:

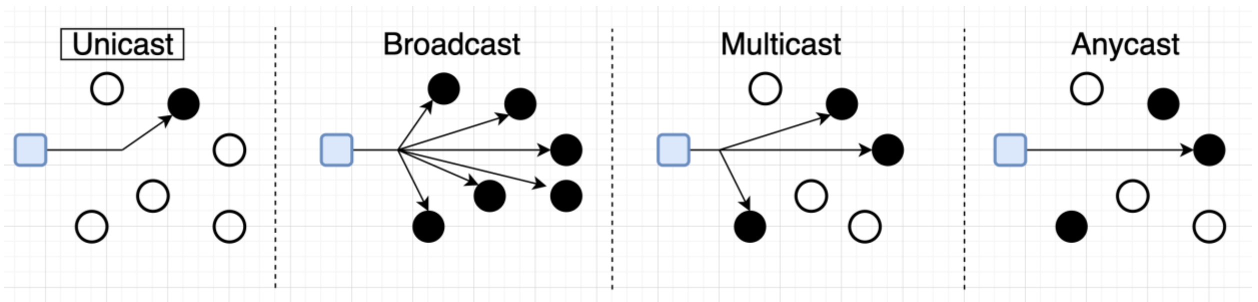
- **Static routing**: for small networks, which enables the network administrator to enter the routing information manually in the routing table. This is time-consuming and difficult to scale
- **Dynamic Router**: for large networks, which updates the routing table automatically according to the changes in network topology and information received from other routers. The work is completed by the help of routing protocols such as RIP, OSPF, BGP and IGRP.

Routing Schemes

Data is transported over a network in different routing schemes:

- Unicast: from one source to one destination, i.e. One-to-One; mostly used
- Broadcast: from one source to all possible destinations, i.e. One-to-All
- Muticast: from one source to multiple destinations stating an interest in receiving the traffic, i.e. One-to-Many

- Anycast: from one source to any one of a group of potential destinations which are all identified by the same destination address, i.e. One-to-Any; selection is often based on distance



Most of the traffic on the Internet or Intranet use Unicast, as the destination is already known in those cases. Router just needs to look up the routing table and forward the packet to the next hop (router).

Router Protocols

Routing protocols determine how a router identifies other routers in the network, keeps track of all possible destinations and makes dynamic decisions for where to send each network message.

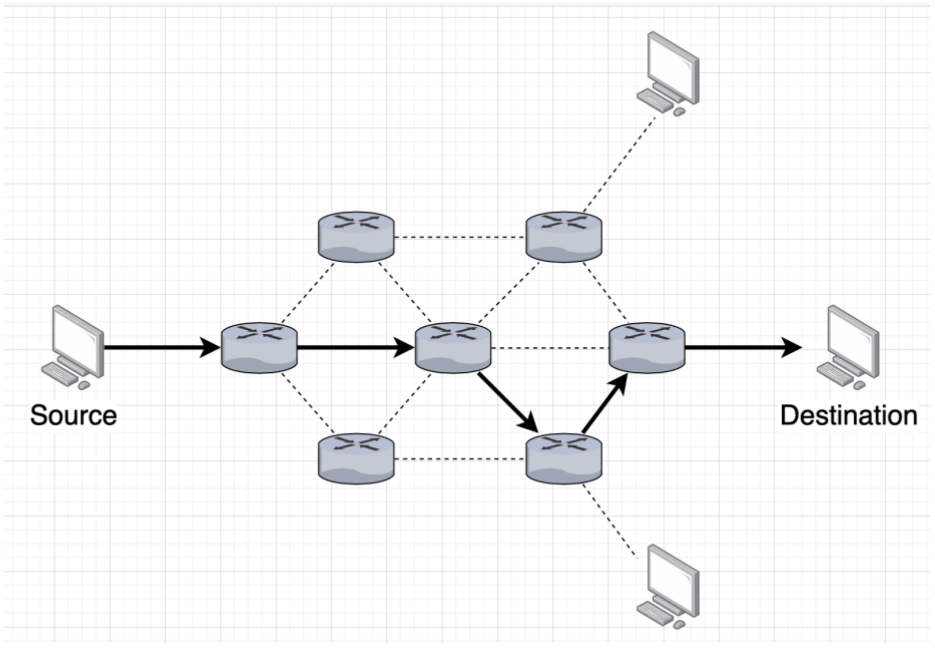
Common protocols include:

- **OSPF** (Open Shortest Path First): find the best path for packets as they pass through a set of connected networks
- **BGP** (Border Gateway Protocol): manage how packets are routed across the internet through the exchange of information between edge routers; this ensures router can quickly adapt to send packets through another reconnection if one path goes down
- **IGRP** (Interior Gateway Routing Protocol): determine how routing information between gateways will be exchanged within an autonomous network
- **EIGRP** (Enhanced Interior Gateway Routing Protocol): ask for help from neighbor router if it can’t find a route to a destination
- **EGP** (Exterior Gateway Protocol): determine how routing information between two neighbor routers is exchanged
- **RIP** (Routing Information Protocol): the original protocol for defining how routers should share information when moving traffic among an interconnected group of local area networks

How Router Works

Data is first broken down into packets at the sender side and then forward to the router for delivery. When the router receives a packet, it will check the destination of the packet.

If the destination address lies in the same network, then the router will directly transfer the packet to its destination. If the packet does not belong to the same network, then it will check its routing table and pass the packet to the next router specified in the table, which forwards it again until the packet finally reaches its destination network.



Routing tables list directions for forwarding data to particular network destinations, sometimes in the context of other factors, like cost. Eventually, it will base on an algorithmic set of rules that calculate the best way to transmit traffic toward any given IP address.

A routing table often specifies a default route, which is used whenever it fails to find a better forwarding option for a given packet, e.g. home network or small business network router directs all traffic along a single default route to its ISP (Internet Service Provider).

VPN (Virtual Private Network)

A VPN gives you online privacy and anonymity by creating a private network from a public internet connection.

VPN masks your IP address so your online actions are virtually untraceable. Furthermore, VPN services establish secure and encrypted connections to provide more privacy.

VPN Benefits

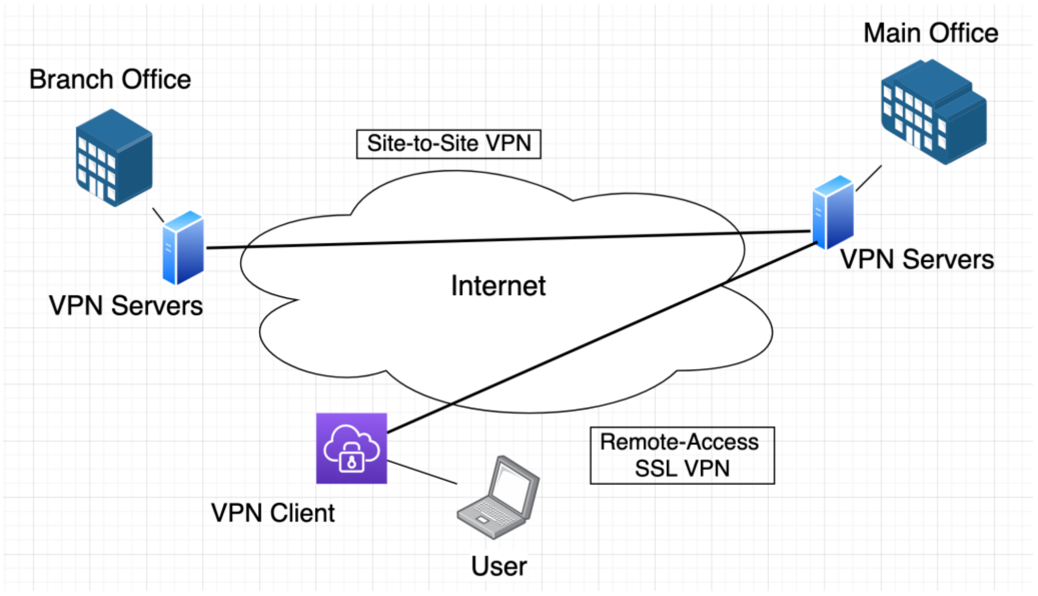
Things VPN can do includes:

- Hide your IP address: connect to a VPN conceals your IP
- Change your IP address: you will get a different IP when using VPN
- Encrypt data transfer: VPN will protect data you transfer on public
- Mask your location: with a VPN, users can choose country of origin for their internet connection
- Access blocked websites: get around websites blocked by governments

VPN Scenarios

VPNS are generally used in two scenarios:

- Remote Access
- Site-to-Site Access



Remote Access

In a Remote-Access VPN scenario, a VPN client, e.g. laptop or smartphone, establishes a remote connection to a VPN server, in order for the client to communicate with the LAN from the public, such as the Internet.

Site-to-Site Access

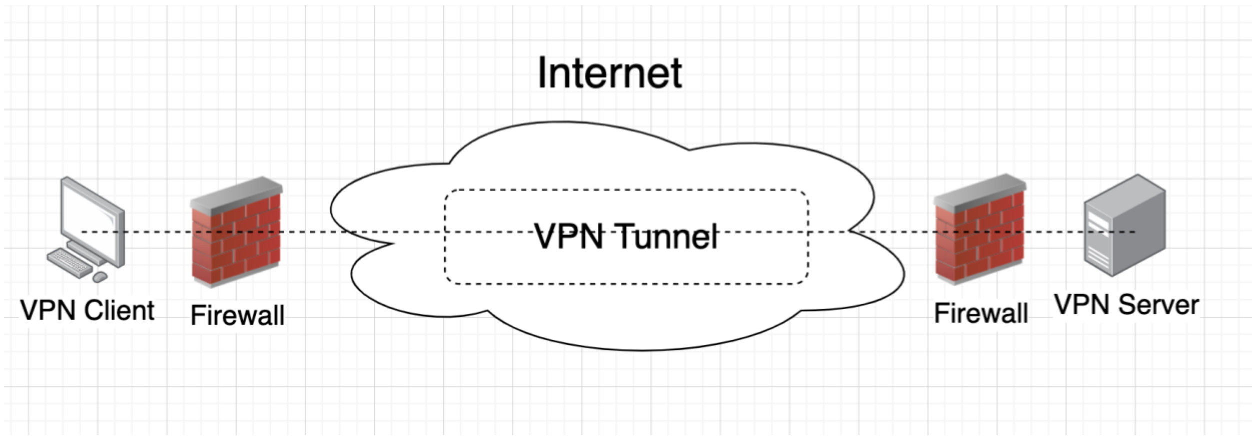
In Site-to-Site VPN scenario, two or more VPN servers create end-to-end connections in order to allow LAN traffic to traverse the WAN (Internet).

Tunneling
In both scenarios of VPN, VPN traffic reaches the intended Network Endpoint through a process called Tunneling.

The term VPN tunneling describes a process whereby data is securely transported from one device or network to another through a non-secure environment (such as the internet) without compromising privacy. Tunneling is protecting data by repackaging it into a different form.

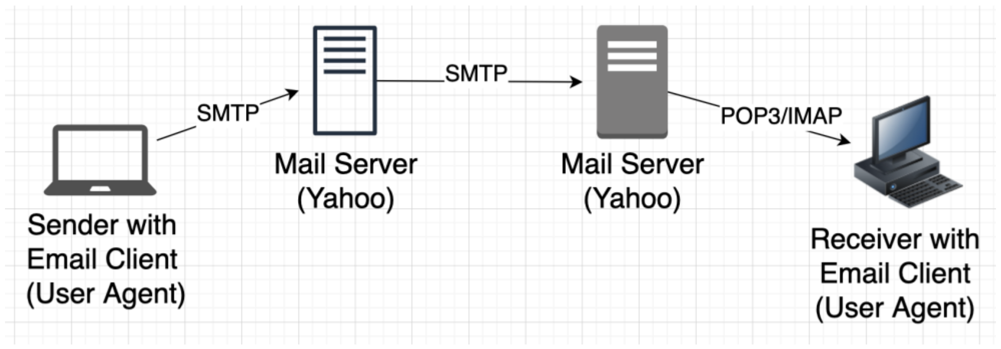
Tunneling is more of a conceptual idea. That is to say, in reality, no physical tunnel exists. The data still needs to travel through the same wires as any other data passing through the public network.

VPN Tunneling uses data encapsulation and encryption to safely carry data traffic through the non-secure environment. Encapsulation isolates the data from other data traveling over the same network, and encryption makes the data unreadable to surveillance or criminals. Hence it is like the data is traveling inside a tunnel.



Application Protocols

SMTP (Simple Mail Transfer Protocol)
SMTP is a protocol used to transfer email from one user to another.


SMTP
SMTP
POP3/IMAP
Mail Server (Yahoo)
Mail Server (Yahoo)
Sender with Email Client (User Agent)
Receiver with Email Client (User Agent)

A user will compose an email through an email app and send via that app to its mail server through SMTP. The first mail server receives the message and establishes a connection with the destination mail server and forwards the message to the destination server via SMTP. Once the message has arrived, the receiver will use an email app to download the message via POP3 or IMAP.

What are POP3 and IMAP?
POP3 (Post Office Protocol version 3) is a standard mail protocol used to receive emails from a remote server to a local email client. POP3 allows you to download email messages on your local computer and read them even when you are offline. When you use POP3 to download messages, the messages will be removed from email server.

IMAP (Internet Message Access Protocol) is a mail protocol for accessing email on a remote web server from a local client.

IMAP and POP3 are the two most commonly used Internet mail protocols for retrieving emails. The difference between the two lies in that POP3 assumes your email is being accessed only from one app while IMAP allows simultaneous access by multiple clients. That’s why IMAP is more popular than POP3.

MIME (Multipurpose Internet Mail Extensions)

MIME is an extension to SMTP that allows users to exchange different kinds of data files over the Internet, e.g. images, audios and videos. MIME is required if the text is using a character set other than ASCII. Virtually all human-written internet email is transmitted via SMTP in MIME format.

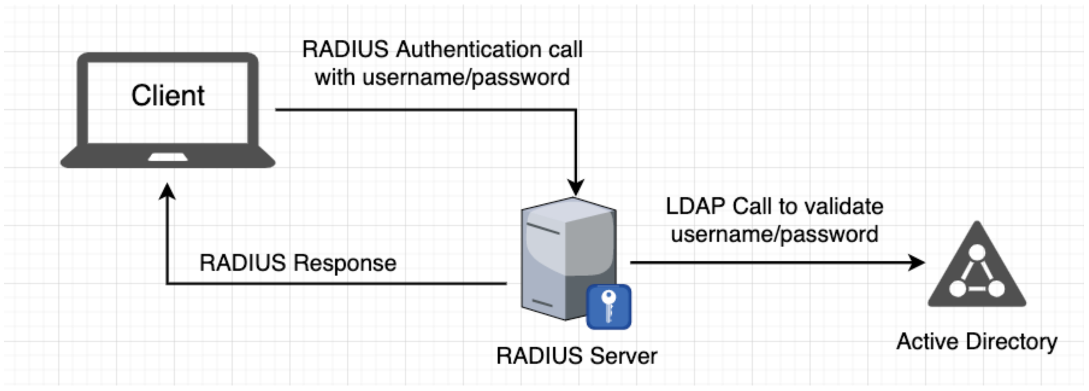
HTTP (HyperText Transfer Protocol)

HTTP is covered in the previous sections but I want to revisit it after we have gone through some network basics. It is one of the application layer protocols. It is the foundation of all websites, that is WWW or World Wide Web.

RADIUS (Remote Authentication Dial-In User Service)

RADIUS is a legacy authentication mechanism. It's a client-server protocol that enables a remote server to communicate with a central server to authenticate users and authorize their access to the request system or service.

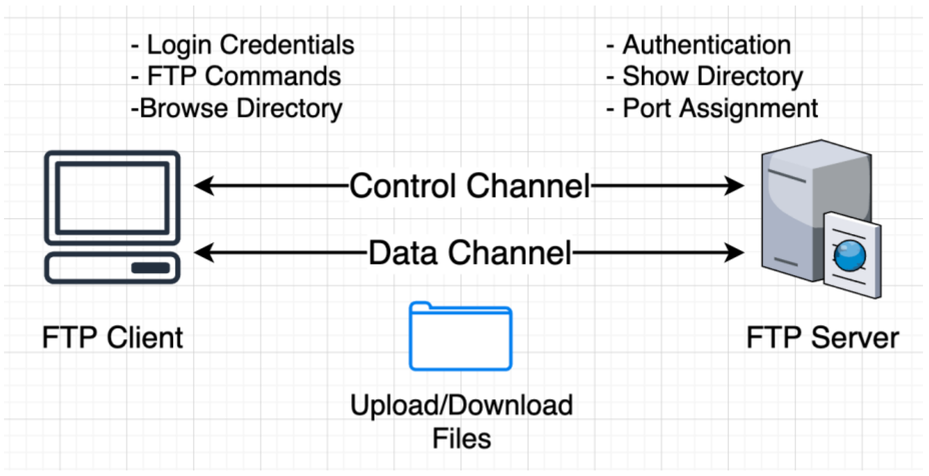
RADIUS allows a company to maintain user profiles in a central database that all remote servers can share.


Client
RADIUS Authentication call with username/password
LDAP Call to validate username/password
RADIUS Response
RADIUS Server
Active Directory

FTP (File Transfer Protocol)

FTP is the most widely used protocol for file transfer over a network. Although a lot of file transfer is now handled using HTTP, FTP is still commonly used to transfer files "behind the scenes".

FTP is a client-server protocol that relies on two communication channels between client and server: Command Channel and Data Channel. The former is for controlling the conversation and the latter is for transmitting file content.



DNS (Domain Name System)

A quick revisit on DNS. We know it is used to translate a domain name into the corresponding IP.

DNS works under Client-Server Model and it primarily uses UDP to serve requests. Sometimes, TCP will be used as well.

Common Protocols, Ports and Functions

Here is a summary of some common protocols, ports and their functions. The port number listed is the default one or a common port that people will use. Yet, port numbers can be changed in some cases.

Name	**Protocol**	**Port**	**Description**
FTP	TCP	20/21	File transfer; FTP control on port 21; FTP data transfer on port 20
SSH	TCP	22	Primary method to manage remote devices and machines on a command level
Telnet	TCP	23	Basic insecure connection to manage remote devices on a command level

SMTP	TCP	25	Transfer email from sender to mail server; also used to transfer email from source to destination between mail servers
DNS	TCP/UDP	53	Translate domain names into IP addresses
DHCP	UDP	67/68	Used on networks that do not use static IP address assignment
HTTP	TCP	80	Main protocol used by web browser
HTTPS	TCP	443	Provide a secure layer on HTTP with either TLS or SSL
IMAP	TCP	143	Main protocol used to retrieve email from a server
POP3	TCP	110	Retrieve email from SMTP servers
SNMP	TCP/UDP	161/162	Used for network monitoring, configure and control.
LDAP	TCP/UDP	389	Used for accessing and maintaining distributed directory information
LDAPS	TCP/UDP	636	Provide a secure layer on LDAP with either TLS or SSL
RDP	TCP	3389	Establish a connection with a remote computer
RADIUS	UDP	1812	Provides centralized authentication, authorization and accounting (AAA)

PART II - IAM Core Concepts

Chapter 7 – Cryptography Basics

For IAM (Identity and Access Management) professionals, having a foundational understanding of cryptography is crucial.

Cryptography plays a vital role in ensuring the confidentiality, integrity, and authenticity of sensitive data within IAM systems. Understanding cryptographic algorithms, such as symmetric encryption, asymmetric encryption, and hashing, allows IAM professionals to implement secure authentication and authorization mechanisms.

Encryption techniques are used to protect sensitive user data, such as passwords and personal information, while hashing algorithms help ensure the integrity of stored data by verifying its integrity against tampering. Familiarity with cryptographic protocols, such as SSL/TLS, enables IAM professionals to establish secure communication channels for transmitting authentication and access control information.

Moreover, knowledge of cryptographic key management practices, such as key generation, distribution, and revocation, is essential for IAM professionals to design and implement secure key management systems.

A fundamental understanding of cryptography equips IAM professionals with the necessary knowledge and skills to implement robust security measures, safeguard user identities, and protect sensitive data within IAM systems.

Key Points

- **Encryption/Decryption:**
    - Transforms data to protect confidentiality.
    - Uses algorithms and keys.
    - Different from encoding (data transformation for compatibility).
- **Symmetric Encryption:**
    - One key for both encryption and decryption.
    - Algorithms: AES, DES, IDEA, Blowfish, RC4 (stream cipher), RC5, RC6.
- **Asymmetric Encryption:**
    - Two keys: Public key for encryption, private key for decryption.
    - Enables secure communication without pre-shared keys.

- Used in TLS/SSL, digital signatures.
- RSA is a common algorithm.

- **Hashing:**

  - One-way function producing a fixed-size output (hash) from variable-size input.
  - Used for data integrity verification, password storage, digital signatures.

- **Digital Signatures:**

  - Authenticates the sender and ensures data integrity.
  - Uses asymmetric encryption and hashing.

- **Digital Certificates:**

  - Links a public key to an entity.
  - Issued by Certificate Authorities (CAs).
  - Used for secure connections (SSL/TLS) and authentication.

- **PKI (Public Key Infrastructure):**

  - Framework for managing digital certificates.
  - Certificate chain: Root CA -> Intermediate CAs -> End-user certificates.
  - Chain of trust: Verifying certificates through a hierarchy.

- **CRL (Certificate Revocation List):**

  - List of revoked certificates.

- **Key and Certificate Management:**

  - Securely managing cryptographic keys and certificates.

- **SSL/TLS:**

  - Cryptographic protocols for secure communication.
  - TLS is the successor to SSL.
  - Provides encryption, authentication, and data integrity.

Cryptography Intro

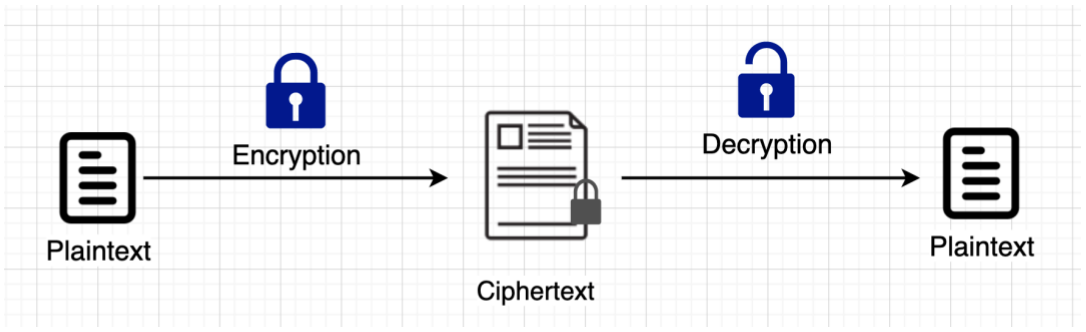
Cryptography is about a set of methods which enable storing and transmitting information while safeguarding it from intruders. In other words, we use cryptography methods to keep information private and only intended recipients can read the message.

Cryptography has been long. The oldest application of cryptography is military secrets protection. One of the best-known ciphers is the Caesar Cipher, attributed to Julius Caesar. It is used to send confidential information to his commanders and soldiers in the field.

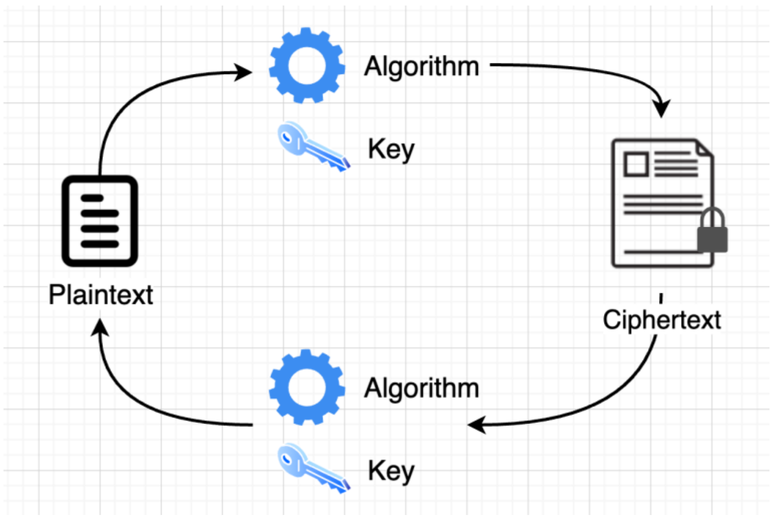
A massive use of cryptography in the modern world is on the Internet. HTTPS use cryptographic ciphers to protect information flowing between client and servers. Passwords are stored in encrypted format to protect from hackers. Other sensitive commercial information is encrypted to avoid disclosing to the public.

Encryption and Decryption

Cryptography is achieved by converting data into a different form called **ciphertext** or **code** which is often incomprehensible. This process is called **encryption**. The reverse process, which converts ciphertext into original data, is called **decryption**.



To perform encryption and decryption, we will need **Algorithms** and **Keys**. The algorithm here can be considered as a mathematical procedure needed to convert data into ciphertext or recover from it. The key here is a piece of information used with an algorithm to fulfill that process.



Encryption VS Encoding

You probably have heard about encoding, which is a different process from encryption. The purpose of encoding is to transform data so that it can be properly stored or consumed by a different type of system.

For example, a message encoded by Morse code does not have privacy protection, as anyone who knows the Morse code standard can understand it; yet, the encoded message can be transferred from one place to another through telecommunication.

On the other hand, encryption is to transform data in order to keep it secret from others.

Key Length and Encryption Strength

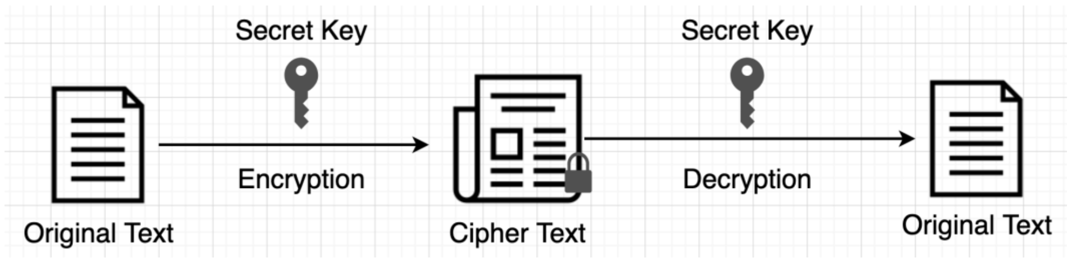
The strength of encryption is related to the difficulty of discovering the key, which in turn depends on both the cipher and the length of the key. Key length is measured in bits.

Generally speaking, longer keys provide stronger encryption. Then the question comes: should we always use a very long key? A longer key may be stronger for encryption; yet, it will also require more calculation during encrypting and decrypting. Thus, a proper key length helps finding a balance between encryption strength and its operability.

Furthermore, different algorithms have different compute procedures. Given the same key length, different algorithms will have different encryption strengths. In other words, to achieve the same level of encryption strength, different algorithms might need to use different key lengths.

Symmetric Key Encryption

Symmetric Key Encryption is a type of encryption where only one key is used to both encrypt and decrypt data. This is in contrast to Asymmetric Key Encryption, where two different keys are used: one for encryption and the other for decryption. The entities communicating via symmetric encryption must exchange the key in a secretive way.



The secret key being used can be a specific password/code or just a random string. Besides the secret key, an encryption algorithm is needed as well. This algorithmic procedure includes ways of both encryption and decryption.

Some examples of symmetric encryption algorithms include:

- AES (Advanced Encryption Standard), this is the most popular one used today
- DES (Data Encryption Standard)
- IDEA (International Data Encryption Algorithm)
- Blowfish (Drop-in replacement for DES or IDEA)
- RC4 (Rivest Cipher 4)
- RC5 (Rivest Cipher 5)
- RC6 (Rivest Cipher 6)

AES, DES, IDEA, Blowfish, RC5 and RC6 are block ciphers. RC4 is stream ciphers.

Block cipher uses a block algorithm, where original data is divided into blocks and encrypted with the secret key. During the encryption process, data will be held in memory before the entire encryption process is done.

Stream cipher uses stream algorithm, where data is encrypted as streams instead of blocks being retained in system memory.

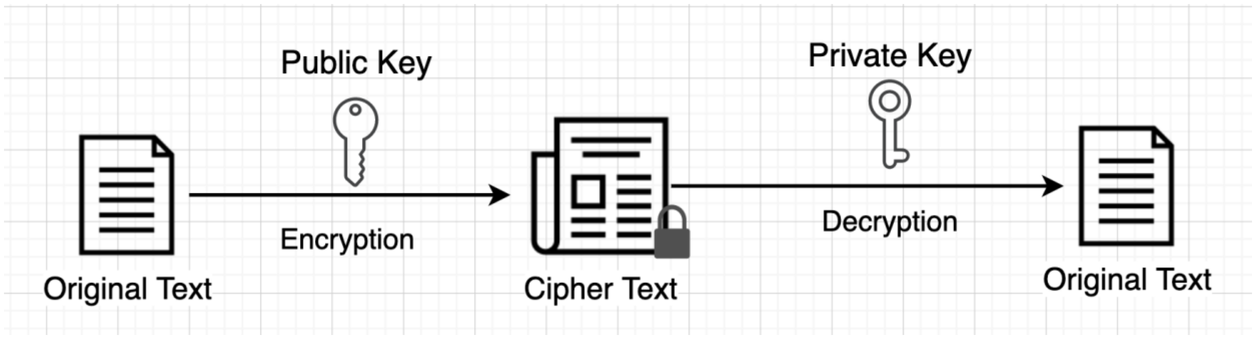
AES is the symmetric encryption choice for most applications today. It is very widely used, mostly with 128-bit or 256-bit keys. 256-bit keys are considered strong enough to protect military top-secret data.

Asymmetric Key Encryption

Asymmetric key encryption, also known as **Public-Key Encryption**, uses a pair of keys (Public Key and Private Key) for encryption/decryption in contrast to just one key in Symmetric Key encryption.

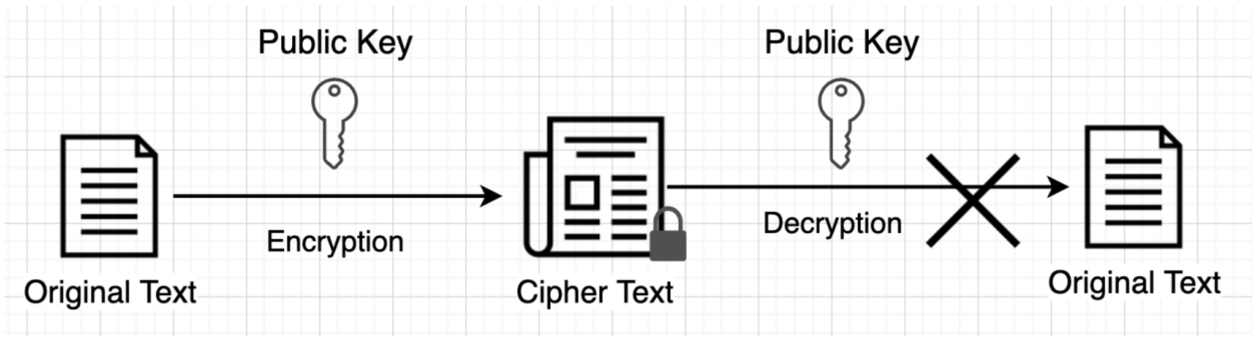
A public key is a key used by any person to encrypt a message so that it can only be decrypted by the intended recipient with the related private key. An encrypted message with a public key can Only be decrypted by the related private key when the pair of keys were initiated. A private key is usually held by the key initiator and is kept from the public.

Public keys are usually published online for users to download. In this case, when a user wants to send an encrypted message to a recipient, he can download the public key and encrypt the message with it and then send the encrypted message to the recipient. As the encrypted message can only be decrypted by the related private key, which is held by the recipient. The information contained will be protected from external access.



On the other hand, if a private key holder encrypts the message with the private key, any user with a public key can decrypt it. A scenario for this use case is to authenticate the message is really sent by the private key holder.

Because of the one-way nature of the encryption function, a sender is unable to read the message of another sender, even though they have the same public key. In other words, a public key cannot decrypt a message which is encrypted by that public key.



Asymmetric Key Encryption is widely used today and many protocols rely on it, such as TLS (Transport Layer Security) and SSL (Secure Sockets Layer) which makes HTTPS possible. The encryption process is also used when a secure network connection needs to be established. Besides, Asymmetric Encryption can also be used to authenticate data using digital signatures.

RSA (Rivest-Shamir-Adleman) Algorithm is usually used for Asymmetric encryption and it has become pretty much an industry standard, which also offers choices of key size and digest algorithm.

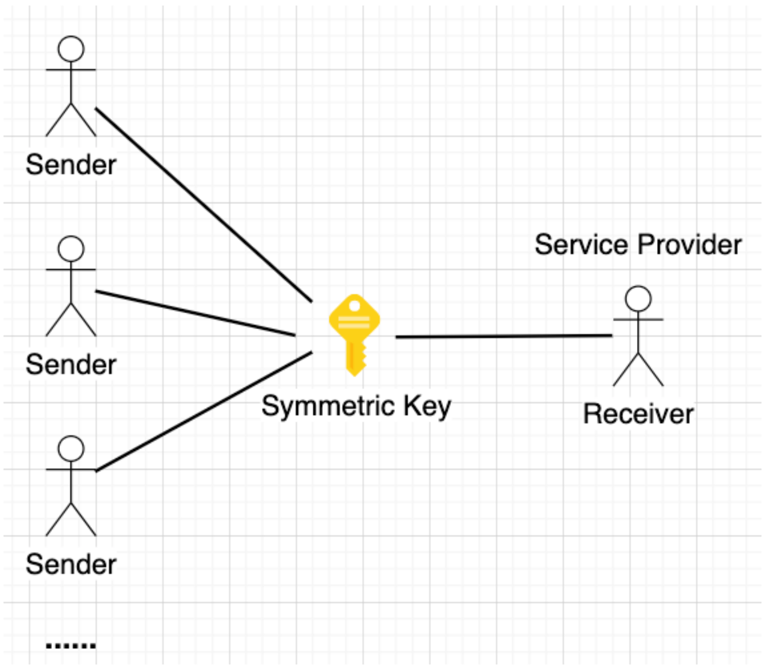
Symmetric VS Asymmetric Encryption

Asymmetric encryption was developed to replace Symmetric encryption, as Symmetric encryption has an inconvenient drawback – the need to share the symmetric key between sender and receiver.

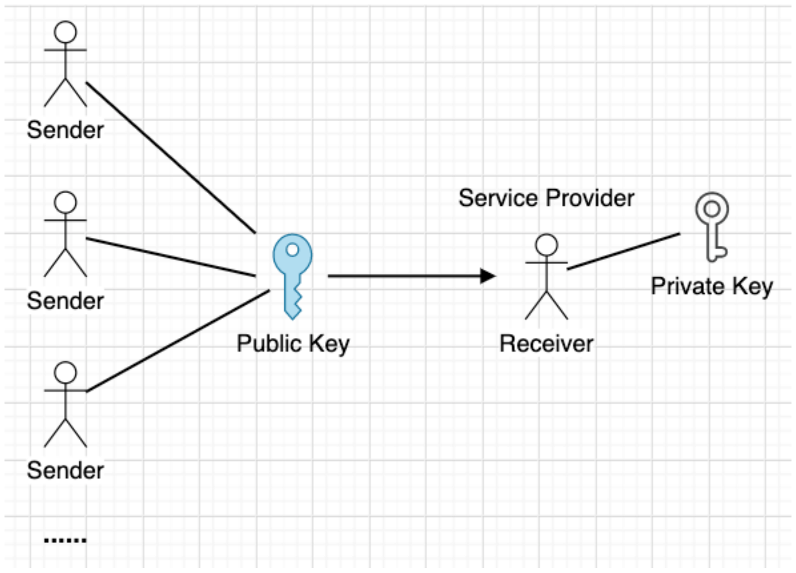
For example, suppose we have a service provider who is also a message receiver. He will be communicating with various service users. Specifically, those users will be sending messages to the receiver and those messages need to be encrypted.

In this case, using Symmetric Key encryption requires that key to be shared among users. This is not a secure way to do it. If one user compromises the key, the encryption could be compromised. As more users are starting to use the service, the risk of compromising the key will increase as well. Furthermore, if a user who possesses the symmetric key is a hacker and intercepts the encrypted message, he will be able to decrypt the message as well.

On the other hand, a lot of times, it is not practical for service providers to use different symmetric keys for different users, e.g. 1000 users will result in 1000 different keys, and that would be difficult to manage.



Asymmetric encryption resolves this issue. First, a public key is published to all the users, any user can download the public key, so there is no such thing as 'compromising the public key'. Second, when a user encrypts a message using the public key, it can't be decrypted with that public key. In other words, other users possessing this public key can't decrypt the message. Only the private key holder can decrypt the message. Third, regarding user scaling, since the public key is already published, increasing user number is not an issue.

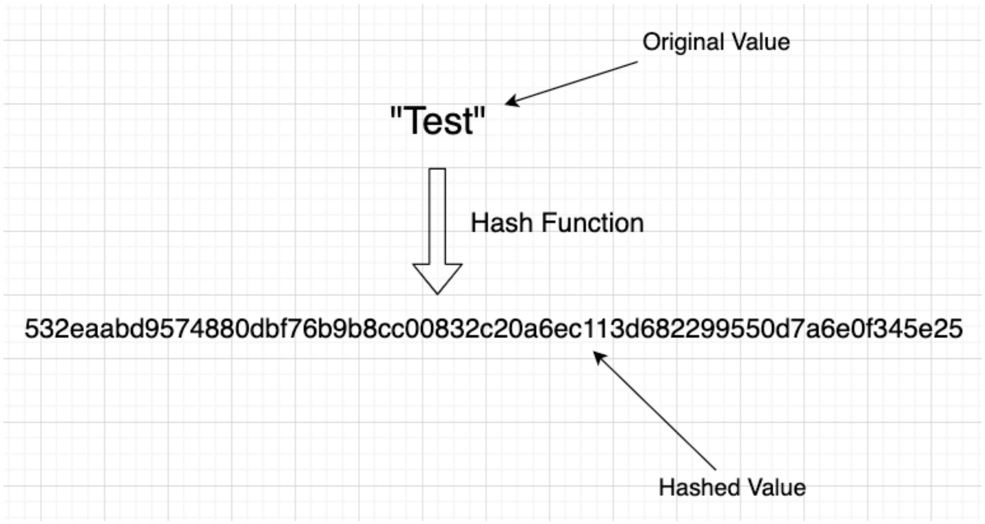


As Asymmetric encryption involves two keys, the processing time is increased. That is, Asymmetric encryption usually takes longer than Symmetric encryption.

Factor	**Symmetric Encryption**	**Asymmetric Encryption**
Number of Keys	One Key	Two keys – Public Key and Private Key
Complexity	Simple technique as only one key is used to carry out both operations	Contribution from separate keys for encryption and decryption makes it a fairly more complex process
Swiftness of Execution	Fast	Relatively slow
Algorithms	AES	RSA

Hashing

Hashing in the cryptographic context usually refers to the process of generating a fixed-size output from an input of variable size. This is done via the use of hash functions, which are mathematical algorithms.

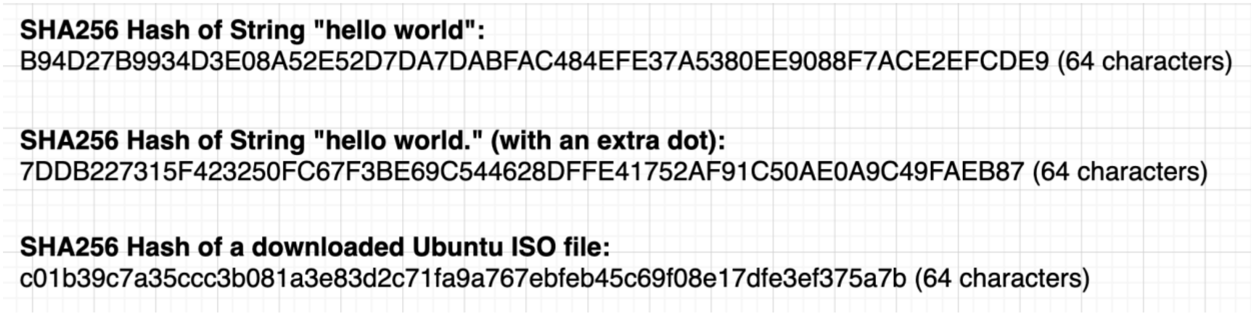

Original Value
"Test"
Hash Function
532eaabd9574880dbf76b9b8cc00832c20a6ec113d682299550d7a6e0f345e25
Hashed Value

Hashing is a one-way method. In other words, an input value through a hash function will generate a hashed string, but the hashed string can’t be reversed to recover the original input value. This is different from Symmetric/Asymmetric encryption and is designed on purpose.

Besides one-way feature, there are two important features regarding hashing:

1. Whether it's a giant file or a small amount of input, the output hash string is fixed length
2. Even if it is a huge file, a tiny change will result in a very different hash string

Here are some examples:



SHA256 Hash of String "hello world":
B94D27B9934D3E08A52E52D7DA7DABFAC484EFE37A5380EE9088F7ACE2EFCDE9 (64 characters)

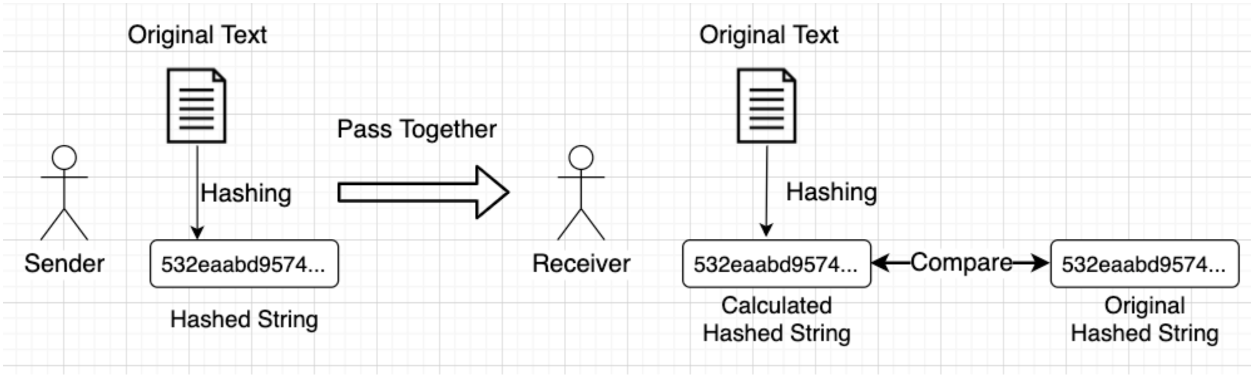
SHA256 Hash of String "hello world." (with an extra dot):
7DDB227315F423250FC67F3BE69C544628DFFE41752AF91C50AE0A9C49FAEB87 (64 characters)

SHA256 Hash of a downloaded Ubuntu ISO file:
c01b39c7a35ccc3b081a3e83d2c71fa9a767ebfeb45c69f08e17dfe3ef375a7b (64 characters)

Adding a dot to the "hello world" string will result in a very different string. The hash string of a downloaded Ubuntu ISO file is still 64 characters.

Seeing those features of hashing, now the question becomes where we should use hashing. While Symmetric/Asymmetric encryption is to protect the information from being accessed from others, the purpose of hashing is to verify the integrity of the data received.

For example, when a user receives a data pack from a sender, how does he know the received data is not missing fragments or tampered?



To help with that, the sender can generate a hashed value of the data and send both data and generated hash string to the receiver. Once the receiver receives the data and hashed value, the receiver can do the same as what the sender has done – generate the hashed value from the original data and then compare the two hashed strings. If the data received has changed, the newly generated hash string will be different from the original hash string, and hence the receiver knows the data is missing fragments or tampered. He can then inform the sender and find another way to send the data.

Real world applications of Hashing could involve:

Password Verification – Storing password in plain text is insecure, so nearly all passwords are stored as hashes. When a user inputs a password, it is hashed and the result is compared to the stored hash value to authenticate the user.

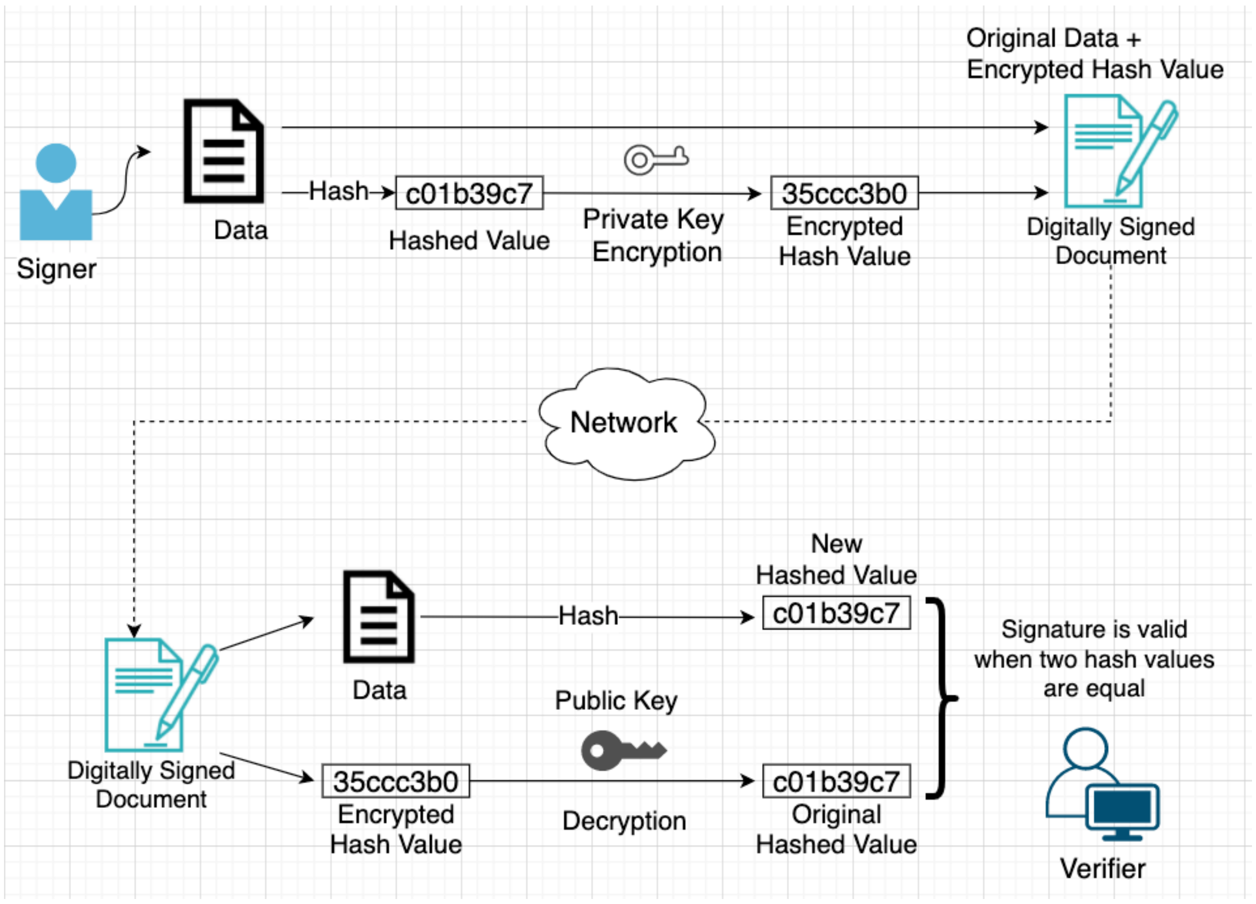
Signature Generation and Verification – Verifying signatures is a process to verify the authenticity of a digital document or message. A valid digital signature gives its receiver strong proof that the message was created by a known sender and that message was not altered in transit. We will look at digital signature and verification later.

Verify File and Message Integrity – Hashes can be used to make sure messages and files transmitted from sender to receiver are not tampered with during transit. This practice builds a "chain of trust".

Digital Signature

Digital signatures can be considered as an electronic version of "fingerprints". In other words, the digital signature securely associates a signer with a document in a transaction.

Here is the flow on how digital signature is created and verified.



First, the signer generates a hash value from the data and then encrypts the hash value with his private key. The encrypted hash value will be "packaged" together with data and sent to the receiver. After the receiver receives the "package", he generates a new hash value with the data received. Besides that, the receiver will use a public key published by the signer to decrypt the encrypted hash value from the "package", which will give us the original hash value. The two hash values are then compared for their values. Any discrepancy means the "package" is not valid and should be discarded.

Let's take a closer look to see why this is a fail-safe delivery. First, if the encrypted value is not using the signer's private key, the verifier's decryption will fail. As the public key is published from the Signer's website, the public key guarantees its originality. This means the private key used to encrypt the data was incorrect. This could be due to a mistake or a hacker's interception. The data will be discarded in this case. In this case, the public and private keys guarantee the source of the data (the sender) is valid. Next, we need to ensure the validity of the data. Any modification on the data, even so very tiny, will result in a different hash value due to the feature of hashing. If the two hash values are different, it means the data has been tampered and the "package" should be discarded as well.

The failure on the verifier's side will bring up exceptions in the system which can then be handled by the system. For example, verifiers can ask the signer to send another "package" or some other methods will be used to tackle the situation.

The point here, with the digital signature which leverages Asymmetric Encryption and Hashing, is that message delivery is guaranteed to be successful.

Digital Certificate

A Digital Certificate, also known as Public Key Certificate or PKI Certificate, is used to link ownership of a public key with the entity that owns it.

The majority digital certificates serve two main functions:

1. It verifies the identity of the applicant. With digital certificates, you can be ensured that the entities (website, companies, individuals, etc.) you are interacting with are really who they say they are.
2. It encrypts the data transferred between two systems so data can only be interpreted by the intended receiver.

The format of digital certificates is defined by X.509, which is an industry standard. It is defined by ITU-T (International Telecommunication Union's Standardization sector).

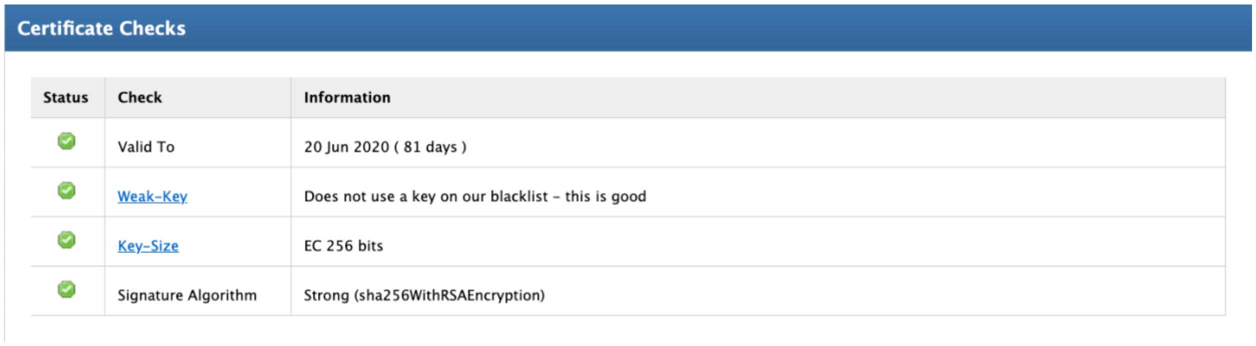
Certificate Sample

Here is a sample certificate which is the SSL cert for Wikipedia.org. Note it is encoded so it is not human-readable. The "-----BEGIN CERTIFICATE-----" and "-----END CERTIFICATE-----" parts are usually required when you use the certificate.

```
-----BEGIN CERTIFICATE-----
MIIHNzCCBh+gAwIBAgISAwKzQ/VZCy5Vym5HWaTTD98LMA0GCSqGSIb3DQEBCwUA
MEoxCzAJBgNVBAYTAlVTMRYwFAYDVQQKEw1MZXQncyBFbmNyeXB0MSMwIQYDVQ
QD
ExpMZXQncyBFbmNyeXB0IEF1dGhvcml0eSBYMzAeFw0yMDAzMjIwNzAxNDFaFw0y
MDA2MjAwNzAxNDFaMBoxGDAWBgNVBAMMDyoud2lraXBlZGlhLm9yZzBZMBMGByqG
SM49AgEGCCqGSM49AwEHA0IABHS1tRzfLiZGLFCszVEV5laIDrq0vzi2GEEKkdge
zXtmCqEW848lPnECin6IlK3rF/COA6MmzAk5v2hldzc3X0+jggUQMIIFDDAOBgNV
HQ8BAf8EBAMCB4AwHQYDVR0lBBYwFAYIKwYBBQUHAwEGCCsGAQUFBwMCMAwG
A1Ud
EwEB/wQCMAAwHQYDVR0OBBYEFKR4ErppVuDO2hwUpywj+jMrZbDGMB8GA1UdIwQY
MBaAFKhKamMEfd265tE5t6ZFZe/zqOyhMG8GCCsGAQUFBwEBBGMwYTAuBggrBgEF
BQcwAYYiaHR0cDovL29jc3AuaW50LXgzLmxldHNlbmNyeXB0Lm9yZzAvBggrBgEF
BQcwAoYjaHR0cDovL2NlcnQuaW50LXgzLmxldHNlbmNyeXB0Lm9yZy8wggLFBgNV
HREEggK8MIICuIIRKi5tLm1lZGlhd2lraS5vcmeCESoubS53aWtpYm9va3Mub3Jn
```

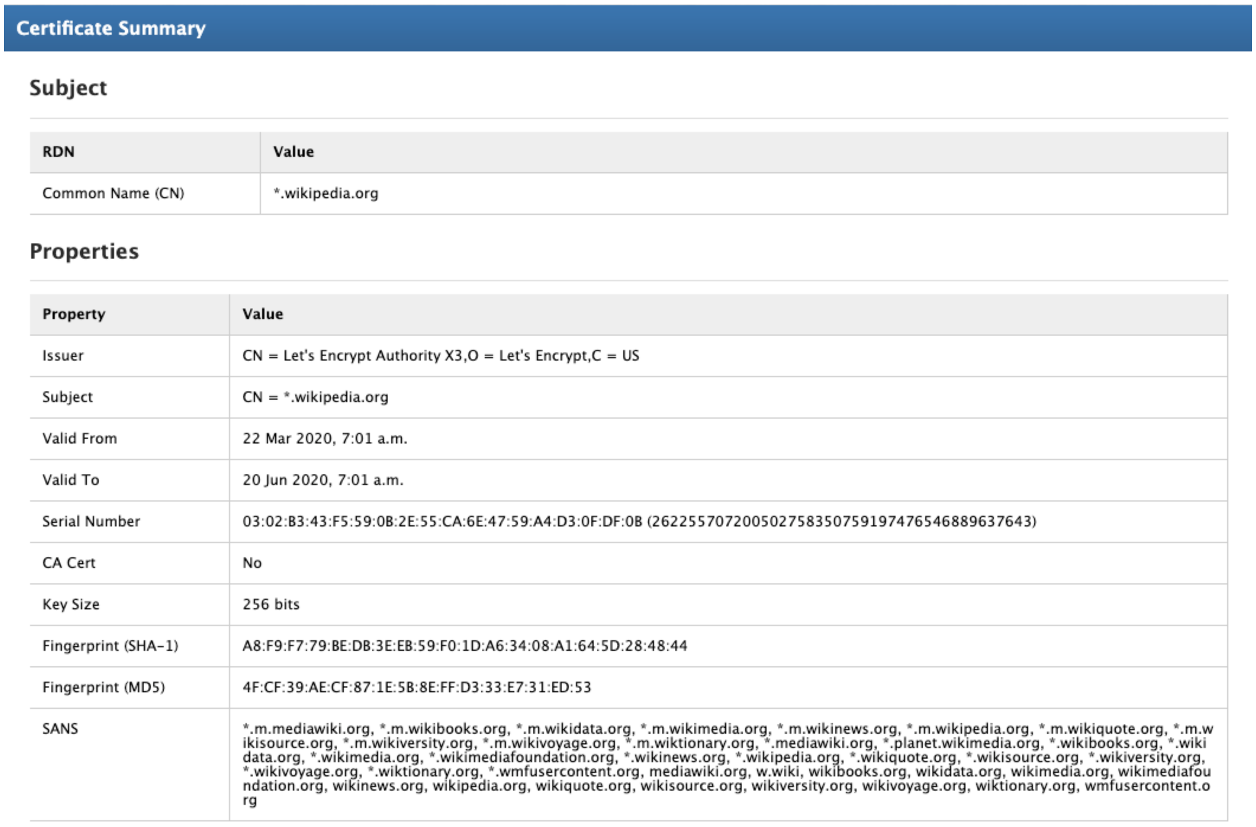
```
ghAqLm0ud2lraWRhdGEub3JnghEqLm0ud2lraW1lZGlhLm9yZ4IQKi5tLndpa2lu
ZXdzLm9yZ4IRKi5tLndpa2lwZWRpYS5vcmeCESoubS53aWtpcXVvdGUub3JnghIq
Lm0ud2lraXNvdXJjZS5vcmeCEyoubS53aWtpdmVyc2l0eS5vcmeCEioubS53aWtp
dm95YWdlLm9yZ4ISKi5tLndpa3Rpb25hcnkub3Jngg8qLm1lZGlhd2lraS5vcmeC
FioucGxhbmV0Lndpa2ltZWRpYS5vcmeCDyoud2lraWJvb2tzLm9yZ4IOKi53aWtp
ZGF0YS5vcmeCDyoud2lraW1lZGlhLm9yZ4IZKi53aWtpbWVkaWFmb3VuZGF0aW9u
Lm9yZ4IOKi53aWtpbmV3cy5vcmeCDyoud2lraXBlZGlhLm9yZ4IPKi53aWtpcXVv
dGUub3JnghAqLndpa2lzb3VyY2Uub3JnghEqLndpa2l2ZXJzaXR5Lm9yZ4IQKi53
aWtpdm95YWdlLm9yZ4IQKi53aWt0aW9uYXJ5Lm9yZ4IUKi53bWZ1c2VyY29udGVu
dC5vcmeCDW1lZGlhd2lraS5vcmeCBncud2lraYINd2lraWJvb2tzLm9yZ4IMd2lr
aWRhdGEub3Jngg13aWtpbWVkaWEub3Jnghd3aWtpbWVkaWFmb3VuZGF0aW9uLm9y
Z4IMd2lraW5ld3Mub3Jngg13aWtpcGVkaWEub3Jngg13aWtpcXVvdGUub3Jngg53
aWtpc291cmNlLm9yZ4IPd2lraXZlcnNpdHkub3Jngg53aWtpdm95YWdlLm9yZ4IO
d2lrdGlvbmFyeS5vcmeCEndtZnVzZXJjb250ZW50Lm9yZzBMBgNVHSAERTBDMAgG
BmeBDAECATA3BgsrBgEEAYLfEwEBATAoMCYGCCsGAQUFBwIBFhpodHRwOi8vY3Bz
LmxldHNlbmNyeXB0Lm9yZzCCAQMGCisGAQQB1nkCBAIEgfQEgfEA7wB1AF6nc/nf
VsDntTZIfdBJ4DJ6kZoMhKESEoQYdZaBcUVYAAABcQFDBZwAAAQDAEYwRAIgWFVq
AFQ0hGMgOHvMM8FrOHBmb0//NUcbdi+oO7wBnqMCIAaypfL8Fns/12wtFsG+zOYG
8xirKVaxgHbDk2R5WFo+AHYAB7dcG+V9aP/xsMYdIxXHuuZXfFeUt2ruvGE6GmnT
ohwAAAFxAUMFxAAABAMARzBFAiEAvsYnQ0hZG+R6lZ4nw1Ni9H2QeFxWwFZCrY6d
FHjcXtgCIFU02965kJ8OiuUrn/dXJYx1JjQbRzrHF0QcXZFDbqD2MA0GCSqGSIb3
DQEBCwUAA4IBAQAdUKHKIE8P3qJS/rWSWE83DUdbznFURQ/UBbDK/Z0r5/BRDUI+
DhTD5bfC2eE1ceC33Pi4KPANM3SNZFrW2w95acuuzQ0pMVKtCJ3gvMyBOOM+ZECc
QXh5acO3wNpC6qvRz+GPnZ+8lwFDbVw6UVNRPGA0wXPTojHiVaMyX/hOhFjKjY7Y
5CbBYGSBKTNnEwVlonV6rUDzYPpSDQ1N8EFUOWaITWQFo1rAflw8b4Ks3F3IWW6+
O8XZanNMhwdfpCvmGHUCYDzRIjObFMevk4YWJo+Feu5wZobLAbHmsW8JoG38EICs
atwcyATUWd8Z1kUmTD2j1xc+Z81Mqfp1QulW
-----END CERTIFICATE-----
```

The decoded information is as below:



Certificate Checks

Status	Check	Information
✓	Valid To	20 Jun 2020 (81 days)
✓	Weak-Key	Does not use a key on our blacklist - this is good
✓	Key-Size	EC 256 bits
✓	Signature Algorithm	Strong (sha256WithRSAEncryption)



Certificate Summary

Subject

RDN	Value
Common Name (CN)	*.wikipedia.org

Properties

Property	Value
Issuer	CN = Let's Encrypt Authority X3,O = Let's Encrypt,C = US
Subject	CN = *.wikipedia.org
Valid From	22 Mar 2020, 7:01 a.m.
Valid To	20 Jun 2020, 7:01 a.m.
Serial Number	03:02:B3:43:F5:59:0B:2E:55:CA:6E:47:59:A4:D3:0F:DF:0B (262255707200502758350759197476546889637643)
CA Cert	No
Key Size	256 bits
Fingerprint (SHA-1)	A8:F9:F7:79:BE:DB:3E:EB:59:F0:1D:A6:34:08:A1:64:5D:28:48:44
Fingerprint (MD5)	4F:CF:39:AE:CF:87:1E:5B:8E:FF:D3:33:E7:31:ED:53
SANS	*.m.mediawiki.org, *.m.wikibooks.org, *.m.wikidata.org, *.m.wikimedia.org, *.m.wikinews.org, *.m.wikipedia.org, *.m.wikiquote.org, *.m.wikisource.org, *.m.wikiversity.org, *.m.wikivoyage.org, *.m.wiktionary.org, *.mediawiki.org, *.planet.wikimedia.org, *.wikibooks.org, *.wikidata.org, *.wikimedia.org, *.wikimediafoundation.org, *.wikinews.org, *.wikipedia.org, *.wikiquote.org, *.wikisource.org, *.wikiversity.org, *.wikivoyage.org, *.wiktionary.org, *.wmfusercontent.org, mediawiki.org, w.wiki, wikibooks.org, wikidata.org, wikimedia.org, wikimediafoundation.org, wikinews.org, wikipedia.org, wikiquote.org, wikisource.org, wikiversity.org, wikivoyage.org, wiktionary.org, wmfusercontent.org

And the detail information is listed here:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            03:02:b3:43:f5:59:0b:2e:55:ca:6e:47:59:a4:d3:0f:df:0b
    Signature Algorithm: sha256WithRSAEncryption
        Issuer:
            commonName                = Let's Encrypt Authority X3
            organizationName          = Let's Encrypt
            countryName               = US
        Validity
            Not Before: Mar 22 07:01:41 2020 GMT
            Not After : Jun 20 07:01:41 2020 GMT
        Subject:
            commonName                = *.wikipedia.org
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:74:b5:b5:1c:df:2e:26:46:2c:50:ac:cd:51:15:
                    e6:56:88:0e:ba:b4:bf:38:b6:18:41:0a:91:d8:1e:
                    cd:7b:66:0a:a1:16:f3:8f:25:3e:71:02:8a:7e:88:
                    94:ad:eb:17:f0:8e:03:a3:26:cc:09:39:bf:68:65:
                    77:37:37:5f:4f
                ASN1 OID: prime256v1
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Subject Key Identifier:
                A4:78:12:BA:69:56:E0:CE:DA:1C:14:A7:2C:23:FA:33:2B:65:B0:C6
            X509v3 Authority Key Identifier:

keyid:A8:4A:6A:63:04:7D:DD:BA:E6:D1:39:B7:A6:45:65:EF:F3:A8:EC:A1

            Authority Information Access:
                OCSP - URI:http://ocsp.int-x3.letsencrypt.org
                CA Issuers - URI:http://cert.int-x3.letsencrypt.org/

            X509v3 Subject Alternative Name:
                DNS:*.m.mediawiki.org, DNS:*.m.wikibooks.org,
DNS:*.m.wikidata.org, DNS:*.m.wikimedia.org, DNS:*.m.wikinews.org,
DNS:*.m.wikipedia.org, DNS:*.m.wikiquote.org, DNS:*.m.wikisource.org,
DNS:*.m.wikiversity.org, DNS:*.m.wikivoyage.org, DNS:*.m.wiktionary.org,
DNS:*.mediawiki.org, DNS:*.planet.wikimedia.org, DNS:*.wikibooks.org,
DNS:*.wikidata.org, DNS:*.wikimedia.org, DNS:*.wikimediafoundation.org,
DNS:*.wikinews.org, DNS:*.wikipedia.org, DNS:*.wikiquote.org,
DNS:*.wikisource.org, DNS:*.wikiversity.org, DNS:*.wikivoyage.org,
DNS:*.wiktionary.org, DNS:*.wmfusercontent.org, DNS:mediawiki.org,
DNS:w.wiki, DNS:wikibooks.org, DNS:wikidata.org, DNS:wikimedia.org,
DNS:wikimediafoundation.org, DNS:wikinews.org, DNS:wikipedia.org,
DNS:wikiquote.org, DNS:wikisource.org, DNS:wikiversity.org,
DNS:wikivoyage.org, DNS:wiktionary.org, DNS:wmfusercontent.org
            X509v3 Certificate Policies:
                Policy: 2.23.140.1.2.1
                Policy: 1.3.6.1.4.1.44947.1.1.1
                  CPS: http://cps.letsencrypt.org
```

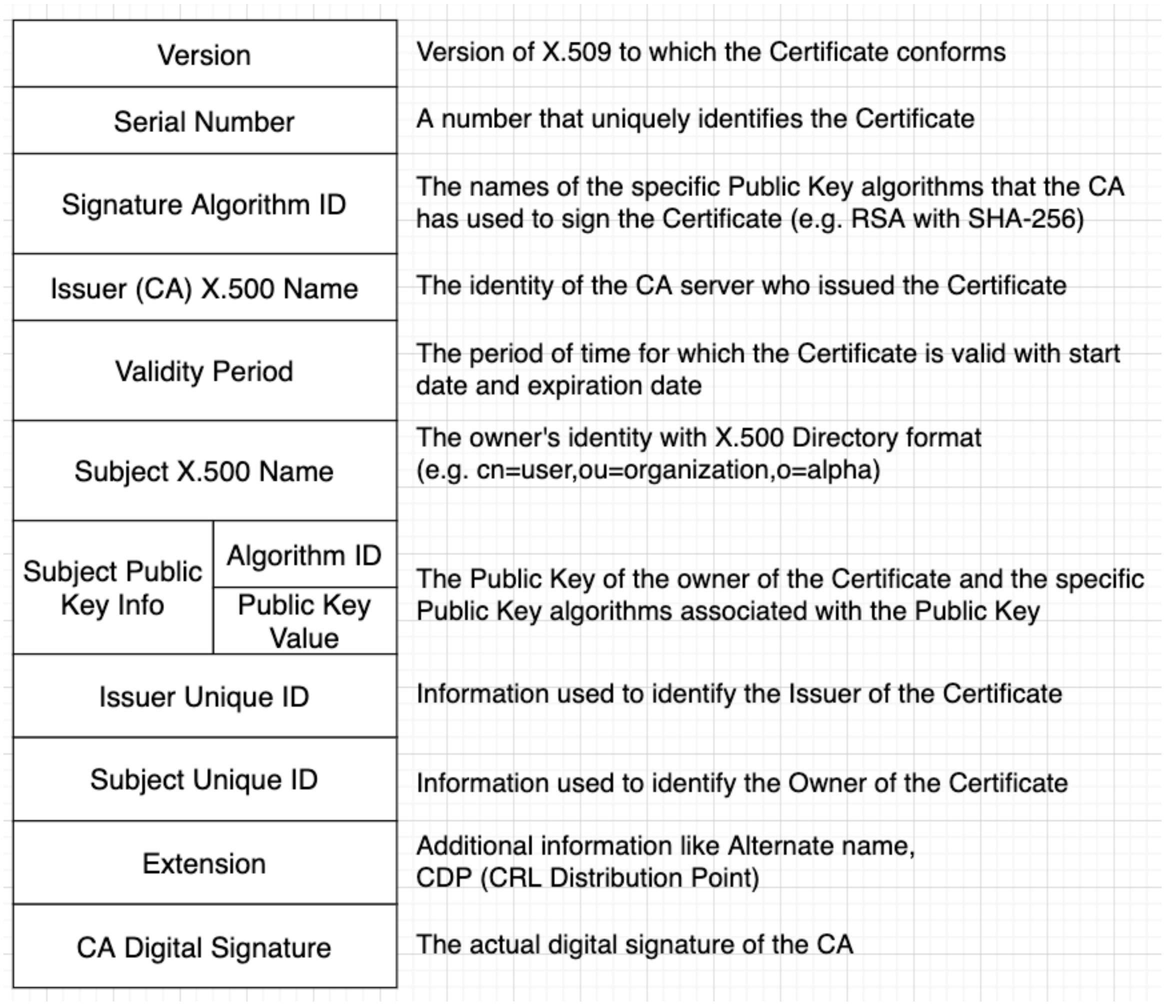
Digital certificates are most commonly used for initializing secure connections (SSL/TLS). Digital certificates are also used for sharing public keys which are used for public key encryption and authentication of digital signatures.

A digital certificate mainly includes:

- Public key being certified
- Identifying information about the entity that owns the public key
- Metadata related to the digital certificate
- Digital signature created by the issuer of the certificate

Format

Usually, the format of a digital certificate will contain the following information.

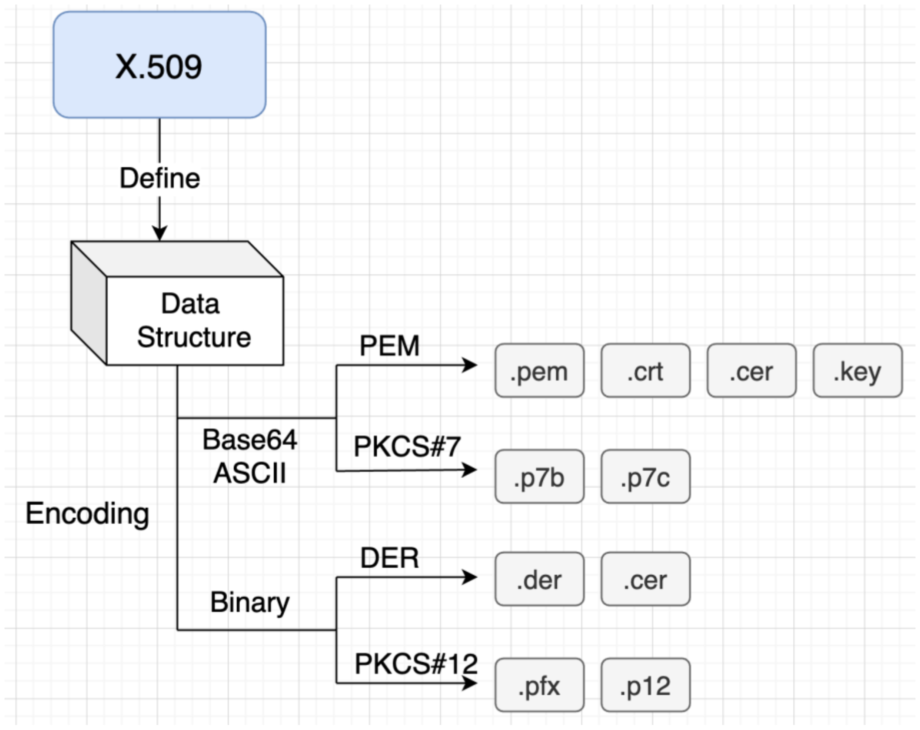


Version		Version of X.509 to which the Certificate conforms
Serial Number		A number that uniquely identifies the Certificate
Signature Algorithm ID		The names of the specific Public Key algorithms that the CA has used to sign the Certificate (e.g. RSA with SHA-256)
Issuer (CA) X.500 Name		The identity of the CA server who issued the Certificate
Validity Period		The period of time for which the Certificate is valid with start date and expiration date
Subject X.500 Name		The owner's identity with X.500 Directory format (e.g. cn=user,ou=organization,o=alpha)
Subject Public Key Info	Algorithm ID	The Public Key of the owner of the Certificate and the specific Public Key algorithms associated with the Public Key
	Public Key Value	
Issuer Unique ID		Information used to identify the Issuer of the Certificate
Subject Unique ID		Information used to identify the Owner of the Certificate
Extension		Additional information like Alternate name, CDP (CRL Distribution Point)
CA Digital Signature		The actual digital signature of the CA

Certificate Format Types

There are different formats of X.509 certificates: PEM, PCKS#7, DER and PCKS#12.

PEM and PCKS#7 formats use Base64 ASCII encoding while DER and PKCS#12 use binary encoding. The certificate files have different extensions based on the format and encoding they use.



PEM Format

Most CAs provide certificates in PEM format in Base64 ASCII encoded files. The certificate file types can be .pem, .crt, .cer, or .key. The .pem file can include the server certificate, the intermediate certificate and the private key in a single file. The server certificate and intermediate certificate can also be in a separate .crt or .cer file. The private key can be in a .key file.

PEM files use ASCII encoding, so they can be opened in text editor. The certificate is contained between ---- BEGIN CERTIFICATE ---- and ---- END CERTIFICATE ---- statements.

PKCS#7 Format

PKCS#7 format is a cryptographic message syntax standard. PKCS#7 certificates use Base64 ASCII encoding with file extension .p7b or .p7c. Only certificates can be stored in this format, not private keys. The P7B certificates are contained between ---- BEGIN PKCS7 ---- and ---- END PKCS7 ---- statements.

DER Format

The DER certificates are in binary form, contained in .der or .cer files. These certificates are mainly used in Java-based web servers.

PKCS#12 Format

The PKCS#12 certificates are in binary form, contained in .pfx or .p12 files

The PKCS#12 can store server certificates, the intermediate certificate and the private key in a single .pfx file with password protection. These certificates are mainly used on the Windows platform.

Certificate Usages

TLS/SSL Server Certificate

In TLS (Transport Layer Security), which is a replacement for SSL, a server is required to present a certificate as part of the initial connection setup. A client connecting to that server will perform the certification validation:

- The subject of the certificate matches the hostname, i.e. domain name, to which the client is trying to connect
- The certificate is signed by a trusted Certificate Authority

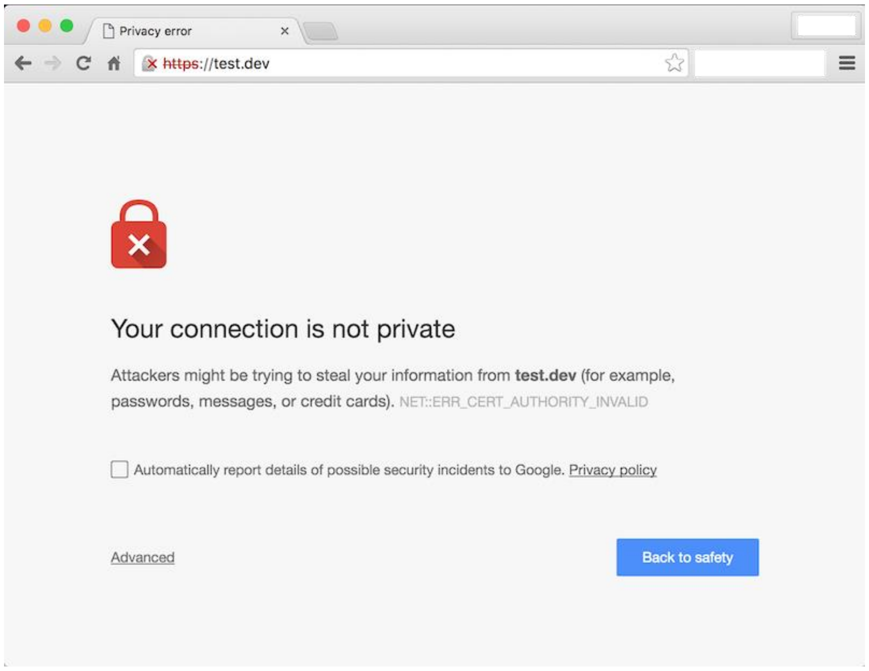
The primary hostname (domain name of the website) is listed as the Common Name in the Subject field of the certificate. A certificate might be valid for multiple hostnames, that is multiple websites.

A TLS/SSL server may be configured with a self-signed certificate. In that case, clients generally won't be able to verify the certificate. Depending on the context, sometimes clients can proceed with access while sometimes they are forced to terminate the connection.

Self-Signed Certificate

A Self-Signed Certificate is a certificate that is not signed by a CA. These certificates are easy to generate and cost-free. Yet, they lack some security properties compared to the ones signed by a CA.

For example, when a web server uses a Self-Signed Certificate to provide HTTPS services, users will see a warning sign in the browser. Sometimes users will be forced by the browser to terminate the connection.


Privacy error

https://test.dev

Your connection is not private

Attackers might be trying to steal your information from **test.dev** (for example, passwords, messages, or credit cards). NET::ERR_CERT_AUTHORITY_INVALID

☐ Automatically report details of possible security incidents to Google. Privacy policy

Root Certificate

A Root Certificate is a certificate that identifies a root CA. Root certificates are self-signed and form the basis of X.509-based PKI (Public Key Infrastructure). We will be looking at more details for PKI later.

Intermediate Certificate

An Intermediate Certificate is a certificate used to sign other certificates. It's within the middle tier in the PKI hierarchy. An intermediate certificate must be signed by another intermediate certificate, or a root certificate.

Leaf Certificate

Leaf certificate is any certificate that cannot be used to sign other certificates. For example, TLS/SSL server certificates.

TLS/SSL Client Certificate

Client certificates are less common compared to server certificates. They are used to authenticate a client connecting to the TLS server as well as providing access control.

Client certificates are more common in RPC systems, where they are used to authenticate devices to ensure that only authorized devices can make certain calls.

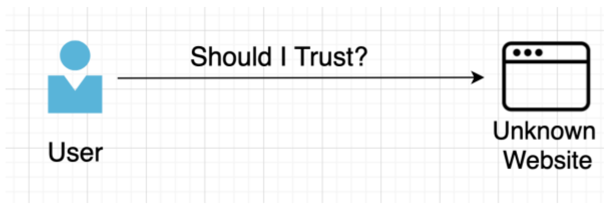
Email Certificate

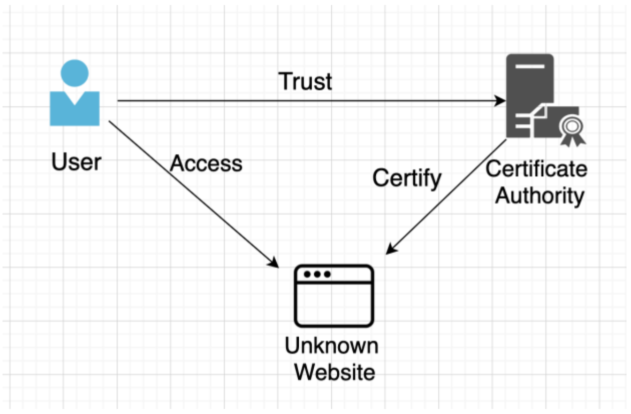
In the secure email protocol, senders need to discover which public key to use for a given recipient. They get this information from an email certificate.

CA (Certificate Authority)

A CA is an entity that is trusted to sign, issue, distribute and revoke digital certificates.

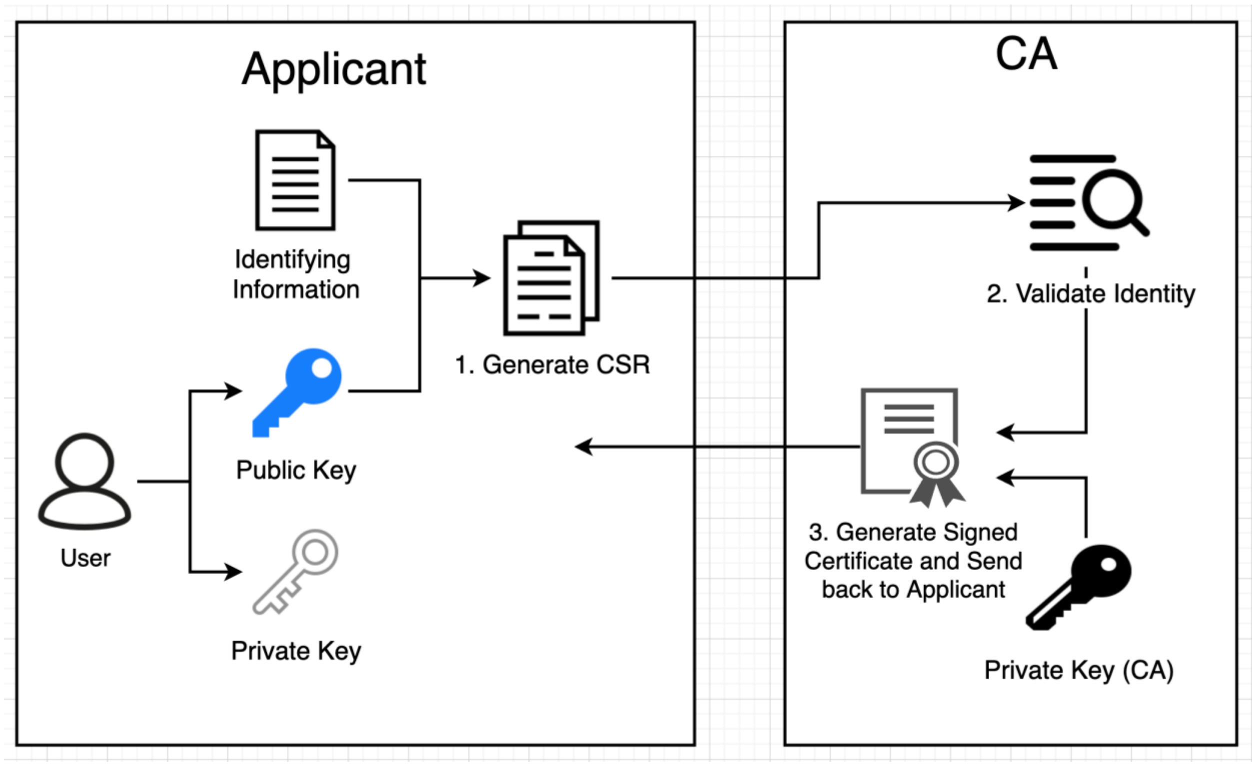
When a user accesses a website, how does he know if the website is safe or not? It would be difficult to tell just between the user and the unknown host.





This issue can be resolved by introducing a third party – Certificate Authority. Basically CA, as its name, is acting as an authority to certify an unknown website (unknown to the user) for its legibility. Since CA is usually an official institute or a trusted entity, the user can trust the website as well.

Requesting a Certificate from CA



To request a digital certificate from CA, an applicant will first generate a pair of keys and then put the public key together with the applicant's personal identifying information to generate a CSR (Certificate Signing Request). A CSR is an encoded text file that includes the public key and other information such as domain name, organization, email address, etc. The private key is held by the applicant and will not be shown to CA.

Once the CSR has been sent to CA, CA will verify that information the CSR contains is correct. If so, CA will use its own private key to generate a signed certificate and send it back to the applicant.

Verifying CA Signed Certificate

When the signed certificate is presented to a third party, CA's signature presented in the signed certificate will be verified. This can be done with the public key CA has published on its website. This is because if the signature was encrypted with CA's private key, it should be able to be decrypted and thus verified by CA's public key. On the contrary, if the signature can't be decrypted with the CA's public key, the verification fails.

PKI (Public Key Infrastructure)

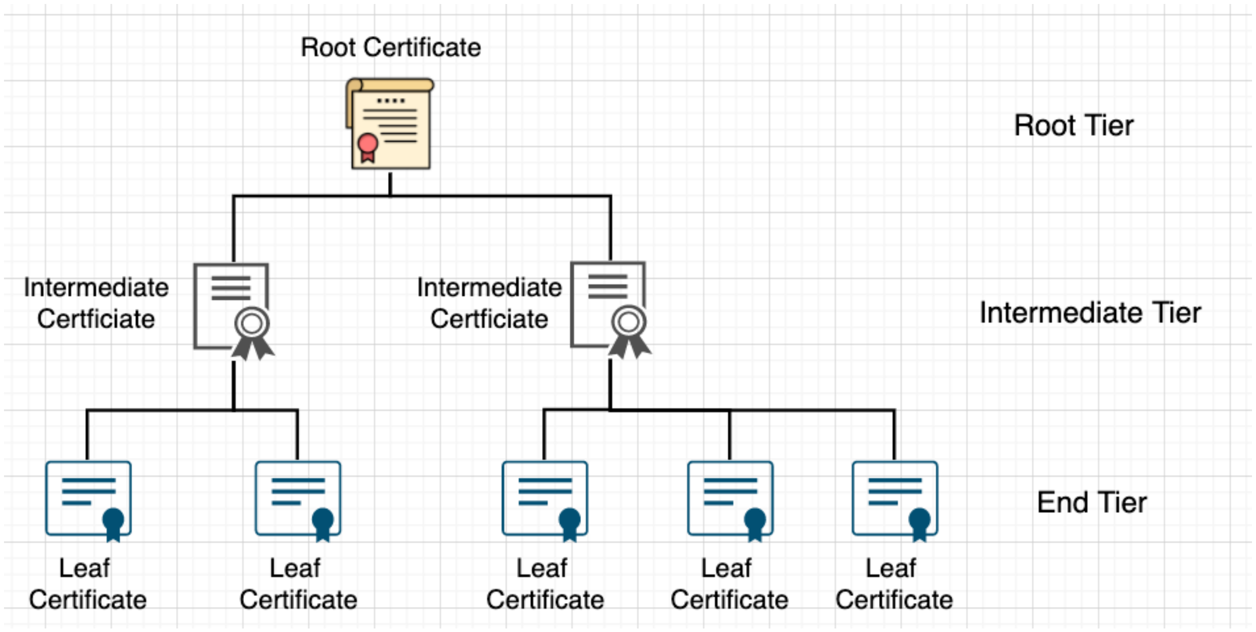
So far, we have seen a list of tools and methods, e.g. asymmetric encryption, hashing, digital signature, digital certificate, which are used to fulfill various security needs in the modern world. To properly manage those, we will need a structure or a framework.

The concept of PKI (Public Key Infrastructure) can be considered as the infrastructure involving the set of hardware, software, policies, processes, and procedures required to create, manage, distribute, use, store, and revoke digital certificates and public-keys.

One of the main elements in PKI is asymmetric encryption which uses a public key and a private. Without PKI, sensitive information can still be encrypted, ensuring confidentiality, and then exchanged between two parties. However, there would be no assurance of the identity of the other party. A public key certificate helps to establish an association between an identity and a public key.

For example, suppose you receive a file from a sender, which can only be opened with a password. Yet, even though you have the password, how would you know the sender's identity or legibility? The sender might be a camouflaged hacker or an illegit group. With the help of a trusted third-party- Certificate Authority, you have a much better confidence level on the source of the information.

Certificate Chain



There are three kinds of certificates here: Root Certificate, Intermediate Certificate and Leaf Certificate (Server Certificate).

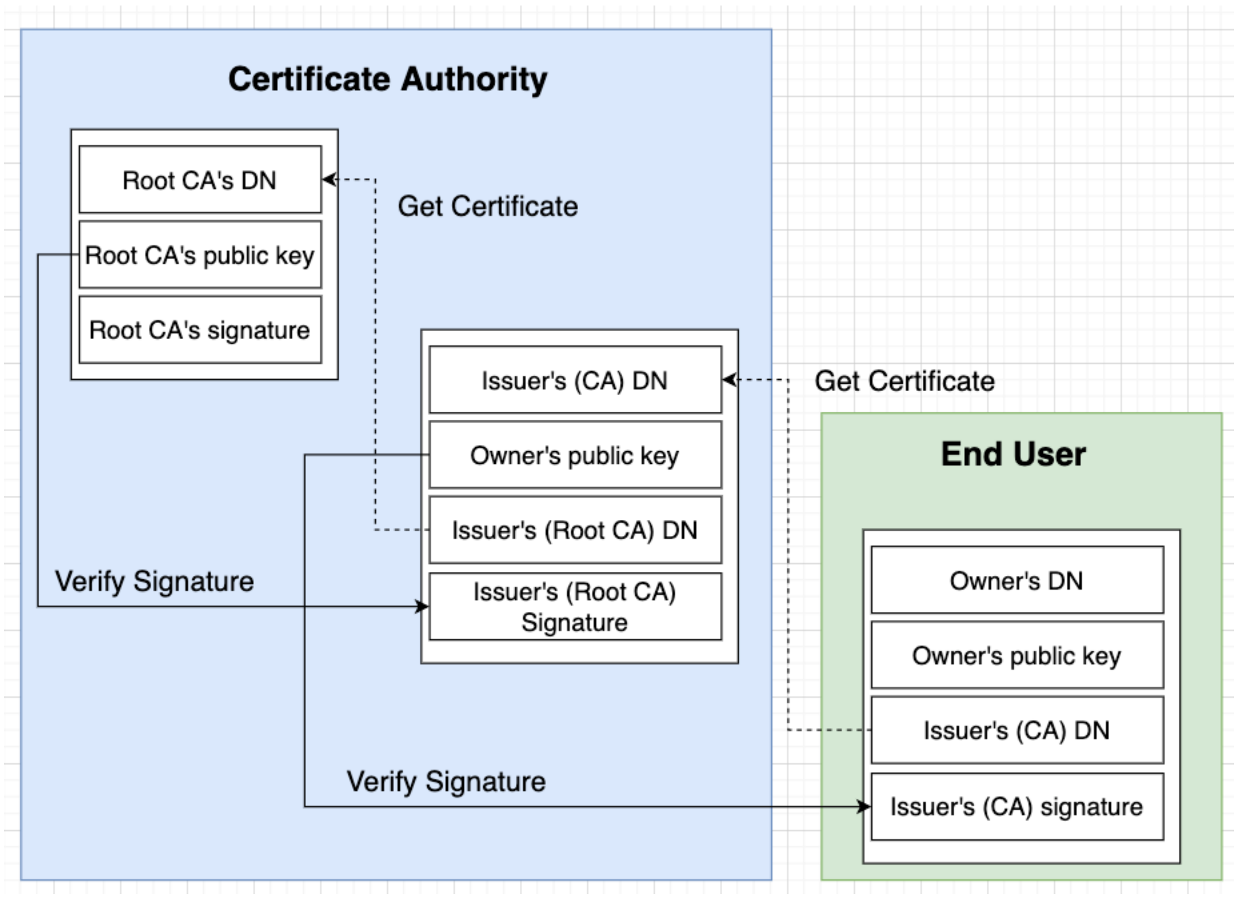
- **Root Certificate**: A certificate that belongs to the issuing Certificate Authority. A number of Root Certificates come pre-downloaded with most web browsers and are stored in a "trust store". The Root Certificates are closely guarded by Certificate Authorities.
- **Intermediate Certificate**: A certificate signed by Root Certificate and is also used to sign other Intermediate Certificate or Leaf Certificate. It acts as a middleman between Root Certificate and Leaf Certificate. There will always be at least one intermediate certificate in a chain, but can be more than one as well.
- **Leaf Certificate**: A certificate signed by an intermediate certificate and can be used by end users.

The benefit of this hierarchical structure is that an intermediate certificate will create a separator between the root certificate and the end certificate. Even an end certificate is compromised in a way that allows malicious access to the next level of certificate, an intermediate certificate is acting as a layer protecting the root certificate.

Root Certificates are self-signed by CA as they are the top level and CAs follow very strict security guidelines to ensure those Root Certificates are well-guarded. Precious Root Certificates will not be used to sign the end certificate directly. Instead, root certificates will sign intermediate certificates which are then used to sign end certificates.

Chain of Trust

The Chain of Trust refers to how an end-user certificate is linked back to a trusted CA. As from the certificate chain, we know that each end certificate has one or more intermediate certificates and a root certificate.



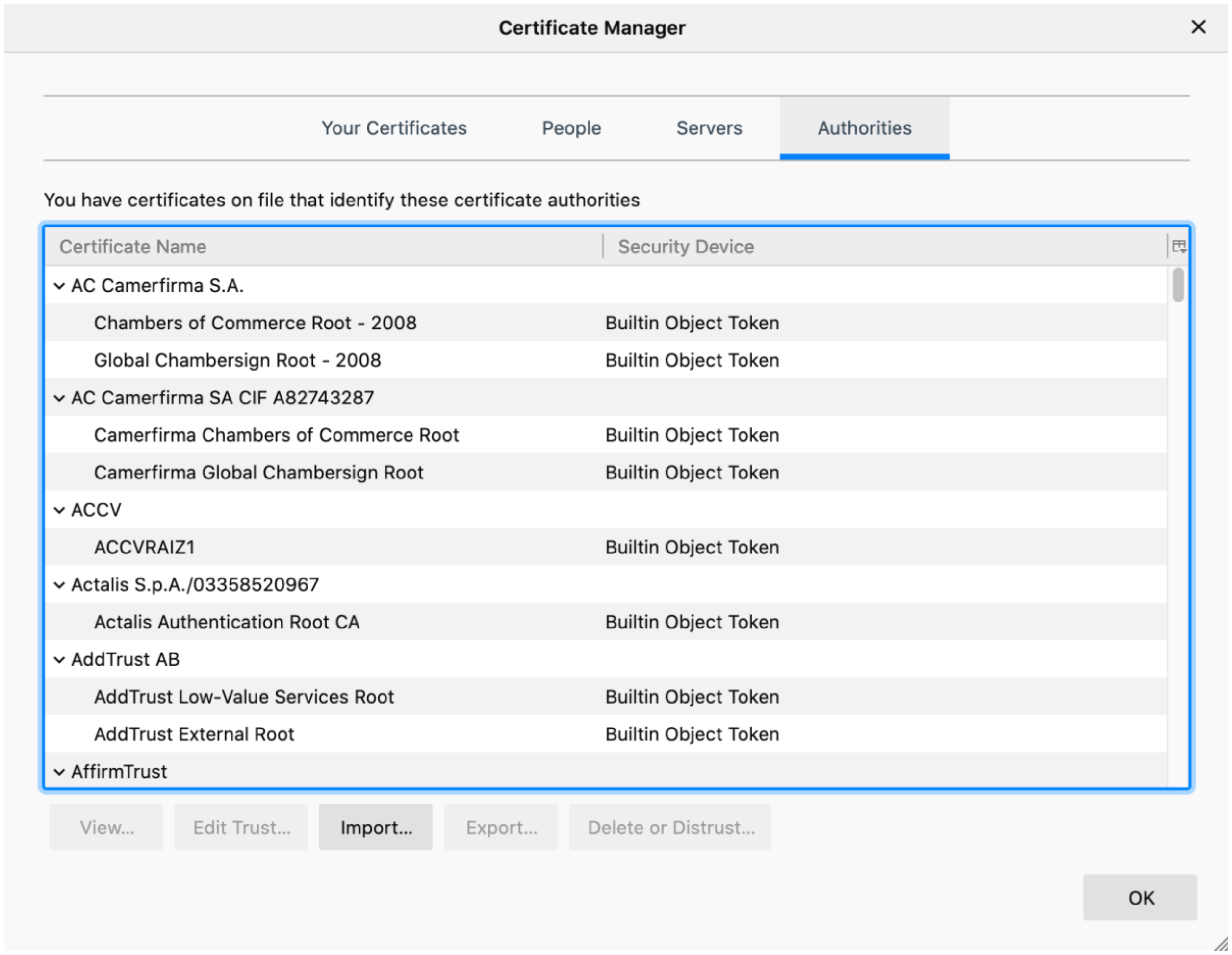
For example, when you use your web browser accessing a website with HTTPS, that SSL/TLS certificate will be verified. It will first check the end certificate's signature with the public key from the issuer. If the signature is verified on the end certificate, the question then is whether the intermediate certificate used to sign the end certificate is legitimate. The web browser will then check the intermediate certificate signature with the public key from the root certificate. This builds up the chain of trust.

Yet, you may wonder how a root certificate is trusted here, as we know root certificate is self-signed. Solving this issue will need to involve third parties.

For a workstation or a mobile device, there is usually a place called 'root store', where a list of trusted root certificates is stored. If not provided by the device itself, a public-facing root store is usually maintained under the authority of a major software provider, which distributes their root store along with software which depends upon it to determine the trust.

Many providers of browsers and operating systems operate their root store programs (usually very strict) and CAs may apply to be accepted into a root store using the criteria of that program. Note each root store is independent of the other. For example, Mozilla, Microsoft, Apple and Google maintain their own root stores.

Here is a list of trusted CAs from Firefox.



You have certificates on file that identify these certificate authorities

Certificate Name	Security Device
AC Camerfirma S.A.	
Chambers of Commerce Root - 2008	Builtin Object Token
Global Chambersign Root - 2008	Builtin Object Token
AC Camerfirma SA CIF A82743287	
Camerfirma Chambers of Commerce Root	Builtin Object Token
Camerfirma Global Chambersign Root	Builtin Object Token
ACCV	
ACCVRAIZ1	Builtin Object Token
Actalis S.p.A./03358520967	
Actalis Authentication Root CA	Builtin Object Token
AddTrust AB	
AddTrust Low-Value Services Root	Builtin Object Token
AddTrust External Root	Builtin Object Token
AffirmTrust	

CRL (Certificate Revocation List)

A CRL is a list of certificates that have been revoked by the CA before their scheduled expiration date and should no longer be trusted. CRLs can be considered as a type of blacklist to verify whether a certificate is still valid and trustworthy.

For example, when a browser makes a connection to a site using TLS, the server’s certificate is checked for anomalies, and part of the process involves checking that the certificate is not listed in a CRL.

A certificate can be revoked for many reasons. The most common reason for revoking a certificate occurs when a certificate’s private key has been compromised. Other reasons could be that if a CA finds a certificate was issued improperly, it may revoke the original and issue a new one; if a certificate is found to be counterfeit, the CA will revoke it and add it to the CRL.

Note that the difficult part for CRL to work properly is that they are difficult to maintain and difficult to be distributed efficiently. When a CA receives a CRL request from a browser, it returns a complete list of all the revoked certificates that the CA manages. The browser must then parse the list to determine if the certificate of the requested site has been revoked. Even though the list is updated as often as hourly, there could still be a latency. Furthermore, to avoid the download overhead, a lot of times CRL is cached, which could also potentially post the risk of not having the latest list.

Certificate Lifecycle

So far, we know a certificate's life span involves different stages. Let's take a look at its lifecycle. The lifecycle of a certificate can be broken down into multiple steps.

1. **Certificate Enrollment** – An entity submits a request for a certificate to CA.
2. **Certificate Issuance** – CA validates the identity of the applicant.
3. **Certificate Validation** – Every time the certificate is used, it will be checked with CA ton confirm that it's still valid
4. **Certificate Revocation** – When a certificate is expired or actively revoked by CA for other reasons
5. **Certificate Renewal** – A certificate can be renewed once it expires, where you can choose whether to generate a new pair of keys or not.

Key and Certificate Management

Key Management

As data encryption and protection has grown dramatically during the recent digital era, there is a large increase in the number of encryption keys used throughout an enterprise.

In essence, key management deals with generating, exchanging, storing, using and replacing cryptographic keys as needed at the user level. Controlling and maintaining data encryption keys is an essential part of any data encryption strategy.

In an organization or enterprise, key management is usually done through a CKMS (Centralized Key Management System). CKMS is an integrated approach for generating, distributing and managing cryptographic keys for devices and applications.

There are several key management standards, some of which are still underway. The best known is the KMIP (Key Management Interoperability Protocol) developed by vendors and OASIS

(Organization for the Advancement of Structured Information Standards). KMIP's goal is to allow users to attach any encryption device to a key management system.

Certificate Management

Similar to key management, certificates, as another essential asset to data encryption and protection, will need to be properly managed as well. This management process can include creation, storage, dissemination, suspension and revocation.

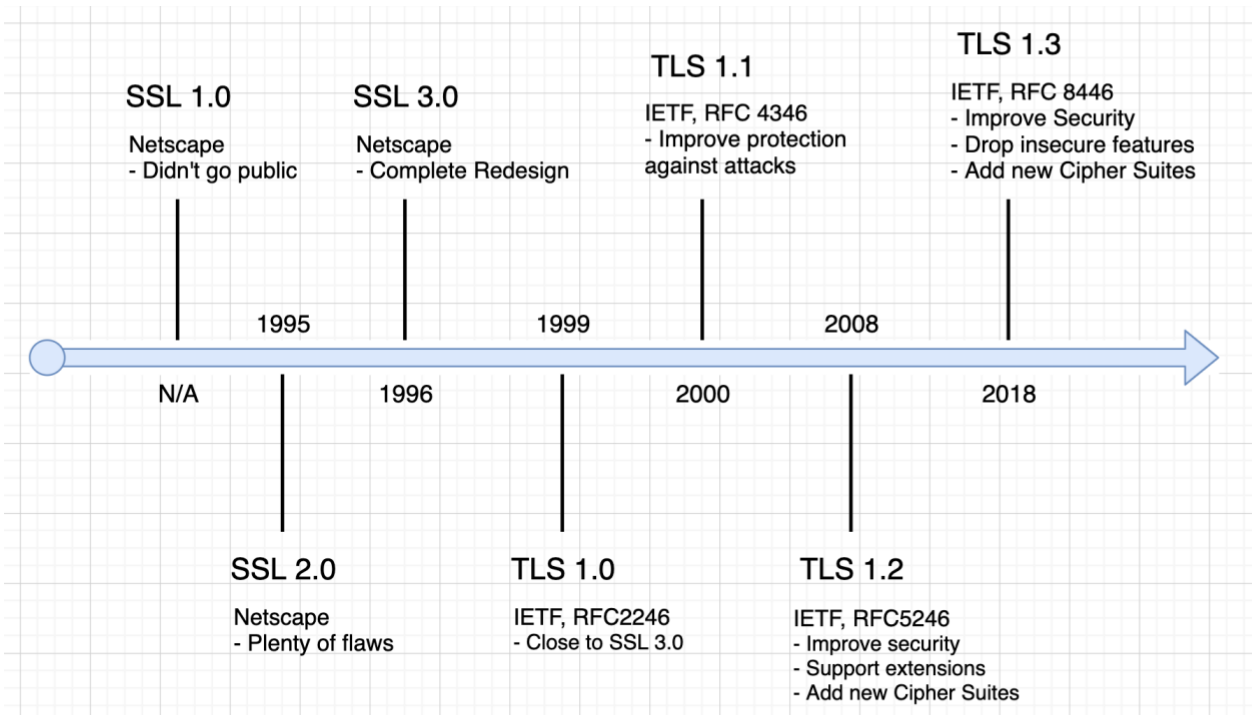
One thing from key management is that the certificate has an expiration date and once that is passed, the certificate can no longer be used. Hence, certificate management requires close monitoring on the valid period of each certificate and proper alert message should be delivered to the right staff member.

SSL/TLS (Secure Socket Layer / Transport Layer Security)

SSL and TLS are both cryptographic protocols that provide authentication and data encryption between servers, machines and applications operating over a network, e.g. client server connection.

When we talk about SSL/TLS, most likely we are referring to TLS.

SSL has been around for about 25 years and TLS is actually a new version of SSL. When IETF adopted SSL, it is renamed to TLS.



Why TLS

As we have seen previously, digital certificates are used in TLS to fulfill some security purpose. Specifically, there are three essential functions: Encryption, Authentication and Data Integrity.

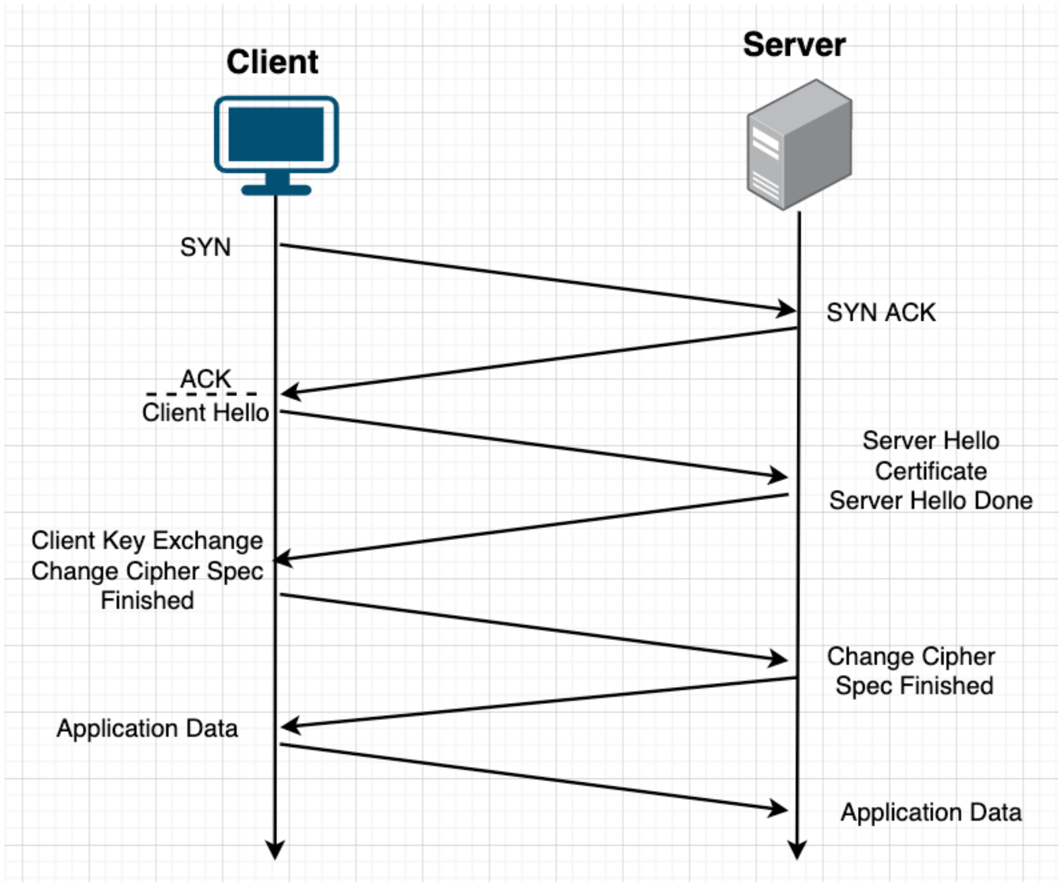
- **Encryption**: A mechanism to protect what is sent from one host to another
- **Authentication**: A mechanism to verify the validity of the identified service provider
- **Integrity**: A mechanism to detect message tampering or forgery

These three functions are crucial during network communication, especially when we are sending sensitive or valuable information.

How TLS works

In order to establish a cryptographically secure data channel, the connection peers must agree on which cipher suite to use and the keys used to encrypt the data.

The exact steps within a TLS handshake will vary depending upon the kind of key exchange algorithm used and the cipher suites supported by both sides. RSA key exchange (asymmetric encryption) is most often used and the following steps describe it.



TLS runs over TCP, which means we must first complete the TCP three-way handshake (SYN, SYN ACK and ACK). With the TCP connection in place, the client sends a hello message including a number of specifications in plain text, such as the version of TLS protocol it is running, the list of supported cipher suites, and a string of random bytes known as the "client random".

When client message received, the server then selects the TLS version and cipher suite from the list provided by the client as well as attaching the certificate (contains public key), a "server random", another random string of bytes that's generated by the server, and sending the hello done response back to the client. Optionally, the server can send a request for the client's certificate.

When the server response is accepted by the client, both sides have agreed on a common version and cipher, and the service certificate has been verified as well. This confirms that the server is who it says it is, and that the client is interacting with the actual owner of the domain.

Client will then initiate a key exchange, which is used to establish the symmetric key for the session. In the case of RSA, the client sends one more random string of bytes, the "premaster secret”, which is encrypted with the server’s public key and can only be decrypted by the server’s private key. This premaster secret will be used by both client and server to generate master secret and session key. The session key is a temporary key used to symmetrically encrypt the data communication within the client server connection session.

The “Change Cipher Spec” message from the client lets the server know that it has generated the session key and is going to switch symmetric encryption with the generated session key. The “Finished” message sent by client to server indicates that the handshake is complete on the client side.

The server processes the key exchange parameters sent by the client, decrypts the premaster secret and computes the session key. Then it sends its “Change Cipher Spec” message to indicate it is switching to symmetric encryption using the generated session key as well. Note both client and server will generate the same session key based on the shared premaster secret. Then the server sends its “Finished” message to the client.

After these steps, both parties now have a session key and can exchange application data over the secured channel they have established. All messages sent from client to server or server to client are encrypted using the session key.

Chapter 8 – OAuth 2.0

A thorough understanding of OAuth 2 is indispensable for professionals operating in IAM. OAuth 2 has emerged as a widely adopted authorization framework, enabling secure and delegated access to resources on behalf of users. IAM professionals need to comprehend the OAuth 2 workflow, including the roles of various entities such as clients, resource owners, authorization servers, and resource servers.

Understanding OAuth 2 grants, such as authorization code, implicit, client credentials, and refresh tokens, allows IAM professionals to design and implement secure authentication and authorization mechanisms for their systems.

Additionally, familiarity with OAuth 2 scopes and scopes management is vital for effectively controlling the level of access granted to different resources and APIs. Being knowledgeable about OAuth 2 best practices ensures IAM professionals can mitigate common security risks and vulnerabilities, such as token leakage or unauthorized access.

So, a strong grasp of OAuth 2 equips IAM professionals with the expertise to implement robust, secure, and user-centric access control mechanisms, facilitating seamless integration and interoperability across various applications and platforms.

First, before we dive into the nitty gritty of OAuth 2.0, it's very helpful to have a brief understanding of why OAuth was created.

Key Points

- **OAuth 2.0 Motivation:**
  - Addresses the issue of secure privilege delegation.
  - Eliminates the need to share user credentials with third-party applications.

- **Key Concepts:**
  - Resource Owner: The user owning the protected resources.
  - Client: The application requesting access to resources.
  - Authorization Server: Issues access tokens.
  - Resource Server: Stores and protects resources.
  - Access Token: Credential for accessing resources (often a JWT).
  - Grants: Workflows to obtain access tokens.
  - Scope: Defines the extent of access granted.

- **Workflow:**

    1. Client redirects the user to the Authorization Server.
    2. User authenticates and grants consent (if required).
    3. Authorization Server issues an access token to the Client.
    4. Client uses the access token to access resources on the Resource Server.

- **Client Registration:**

    - Clients must register with the Authorization Server, providing Client ID, Client Secret, and Redirect URIs.

- **Grant Types:**

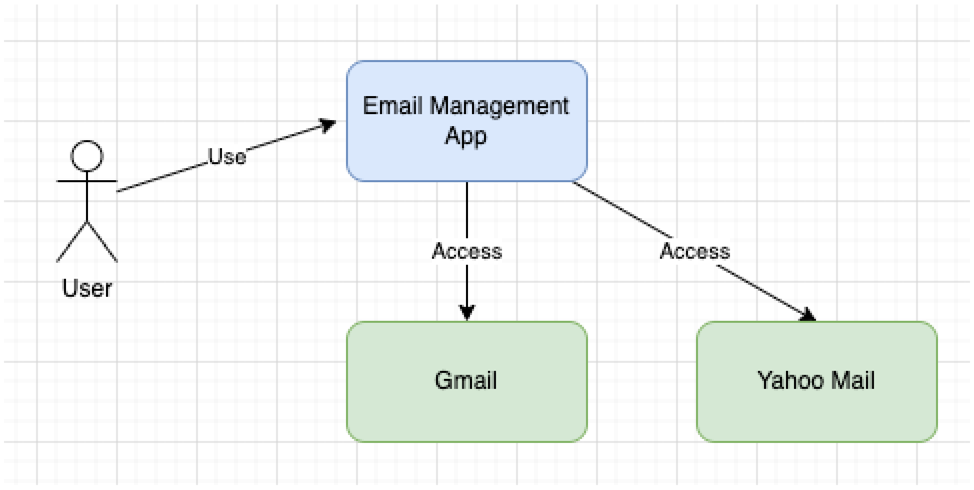
    - Authorization Code: Browser-based flow for user authorization.
        1. User authenticates and grants authorization.
        2. Authorization Server issues an authorization code.
        3. Client exchanges the code for an access token (and refresh token).
    - Client Credentials: For machine-to-machine communication.
        1. Client authenticates using Client ID and Client Secret.
        2. Authorization Server issues an access token.
    - Other grant types: PKCE, Device Code, Refresh Token, Implicit (deprecated), User Password (deprecated).

Issues Before OAuth?

OAuth (Open Authorization) 1.0 was invented in 2007 to fix the above issue and later in 2012, OAuth 2.0 was published to fix some critical issues in OAuth 1.0. Note OAuth 2.0 is not backward compatible with OAuth 1.0. Also, OAuth 2.1 is being drafted to consolidate the functionality in OAuth 2.0 and other new usage patterns.

So what causes the invention of OAuth protocol? The OAuth protocol was invented to solve the problem of privilege delegation. Let's first look at an example to understand what privilege delegation is.

As in the below diagram, a user wants to use an Email Management Application to manage his two email accounts so that he doesn't need to login each email account separately. In this case, the Email Management App will need to access his two email accounts (Gmail and Yahoo Mail), but the question is how.



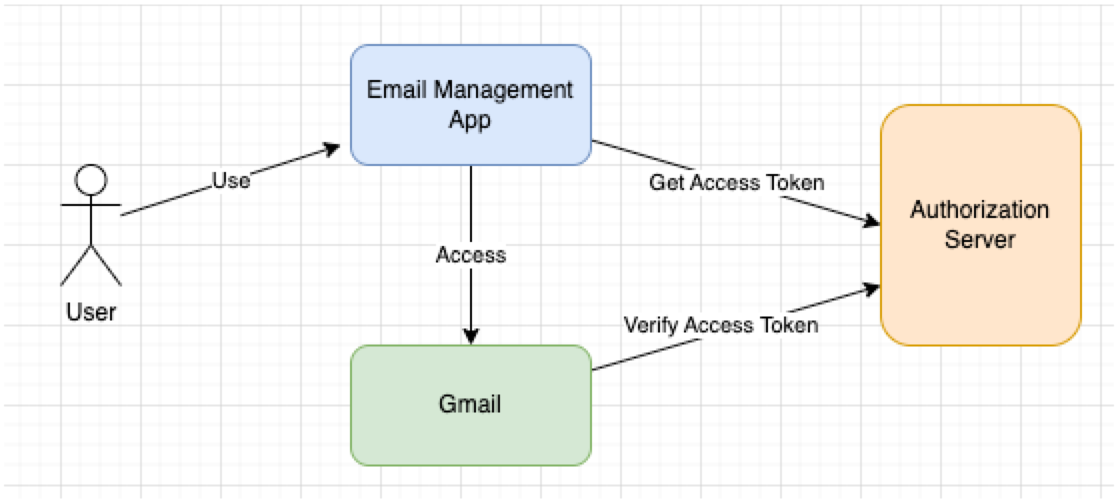
In the old days, the user would need to give his username and password for each of the two email accounts to the Email Management App, so that the app can login and retrieve emails from those two email providers. This is obviously NOT a good way considering user privacy and security. For example:

- Password could be breached
- User doesn't know whether the Email Management App is doing other things rather than just retrieving the emails
- When the password of Gmail and Yahoo Mail is updated, the user needs to update the password information in the Email Management App as well, which is difficult to maintain when the user has more email accounts

When the above problem became prevalent online, some internet standard organizations started to design a protocol to fix the issue.

How OAuth Fixes the Issue

The genius idea here is to add another entity or party (Authorization Server) to the above diagram (see below).



An Authorization Server is added to the flow and in this case, the user doesn't need to give the credentials to the Email Management App anymore. Why? Because the Authorization Server will issue a token (called Access Token) to the Email Management App and the app will use that token as credential/proof to access Gmail and Yahoo Mail. Of course, the Access Token needs to be verified.

So instead of exposing the personal password to the Email Management App, the end user authenticates into the Authorization Server. Then the question is why is this a better situation? As you are still exposing the personal password to the Authorization Server.

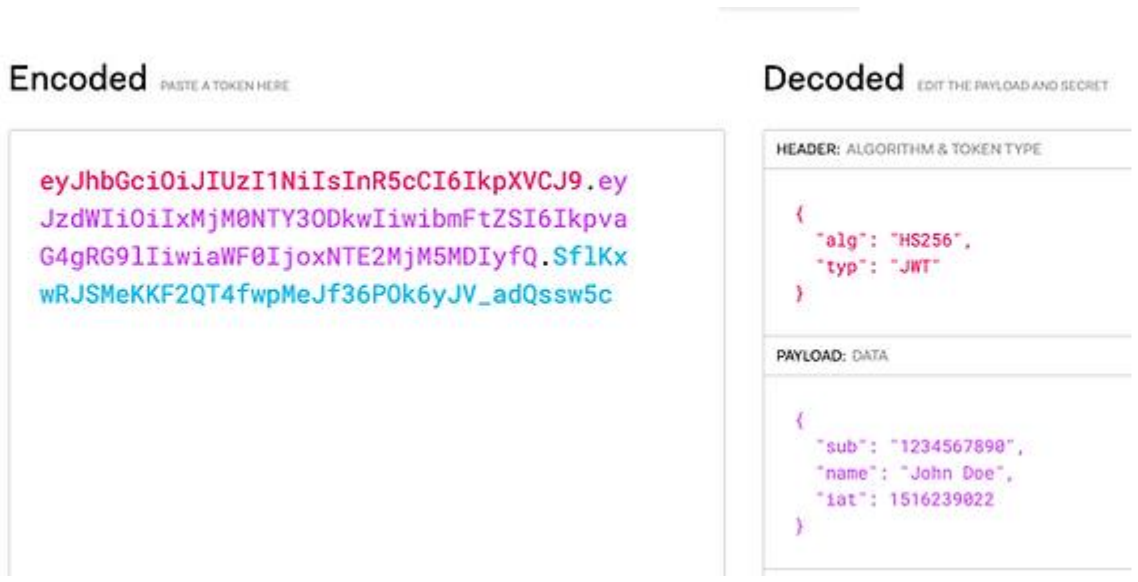
From a view with a larger scope, this Authorization Server is usually not some random server, but rather one with Authority in a given context. For example, in a company's internal network, the company can establish a formal Authorization Server for all OAuth related flows and in this case, we can trust this Authorization Server within the scope of the company's internal network.

Furthermore, another crucial benefit is that when the user authenticates to the Authorization Server, he could just use one user account (such as his company credential) for accessing different Applications. Here we are talking about the Email Management App, but in the real world, there could be hundreds of apps like this and they can all rely on this Authorization Server with the one user account for each user.

As you can see, the critical part is the introduction of the Authorization Server that changes the privilege delegation game and this is why a lot of times, people will say OAuth is a delegation protocol.

Some Terminology

Note Resource Owner, Client, Authorization Server and Resource Server are the four main roles in the OAuth 2.0 protocol and we will talk more about them later.



Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKx
wRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

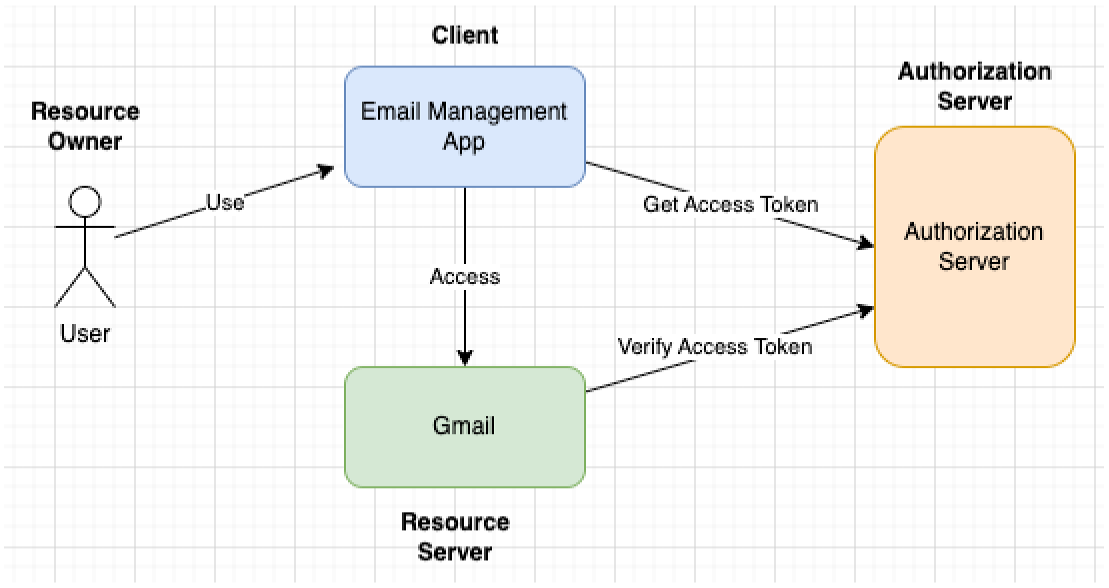
```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

- JWT - A JWT (JSON Web Token) is a method for representing claims securely between two parties as defined in RFC 7519.
- Resource Owner - The end user who owns some resources (e.g. The user owns Gmail and Yahoo Mail accounts)
- Client - An application that will access protected resources (e.g. Email Management App)
- Authorization Server - The server that issues Access Token
- Resource Server - A server that hosts protected resources (e.g. Gmail server and Yahoo Mail server)
- Access Token - A self-contained string used to access protected resources. JWT Token is the most popular. In the pic below, the left side is a JWT token and the right side are decoded values.
- Grants - Different workflows to get Access Token
- Scope - A way to limit the amount of access to the protected resources

Roles and High-level Workflow

Understanding roles will help us to better understand OAuth 2.0.

There are four roles in OAuth 2.0: Resource Owner, Client, Authorization Server and Resource Server, as shown below.



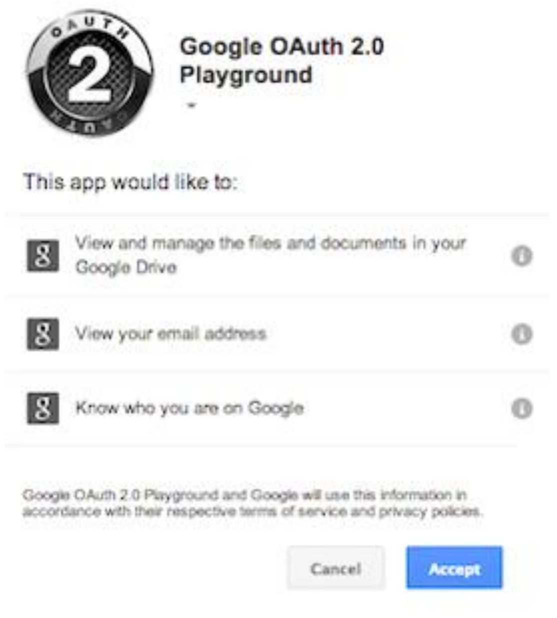
Let's look at the OAuth 2.0 workflow at a high level.

The OAuth 2.0 workflow starts with the end user (Resource Owner) trying to use Email Management App (Client). In order for the Email Management App to work or fulfill its functionalities, it needs to access Gmail and Yahoo Mail server (Resource Server) for the user's email resources.

However, without proper credentials, the Email Management App just can't access those resources, as they are protected. The credential that the client needs is the Access Token and in order for the client to get the Access Token, the client redirects (usually a browser http 302 redirect) the user to Authorization Server, where the user will get authenticated.

Note OAuth 2.0 standard does NOT specify what kind of authentication method to be used, so any method that suits the context will be fine. This is another reason why people say OAuth 2.0 is a Delegation Protocol, but NOT an Authentication Protocol.

After the user is authenticated, the Authorization Server might present a consent page asking the user whether the user will give the listed permissions to the Client App to access the resources. Below is an example. The workflow will be terminated if the user chooses deny or cancel.


Google OAuth 2.0 Playground

This app would like to:

View and manage the files and documents in your Google Drive

View your email address

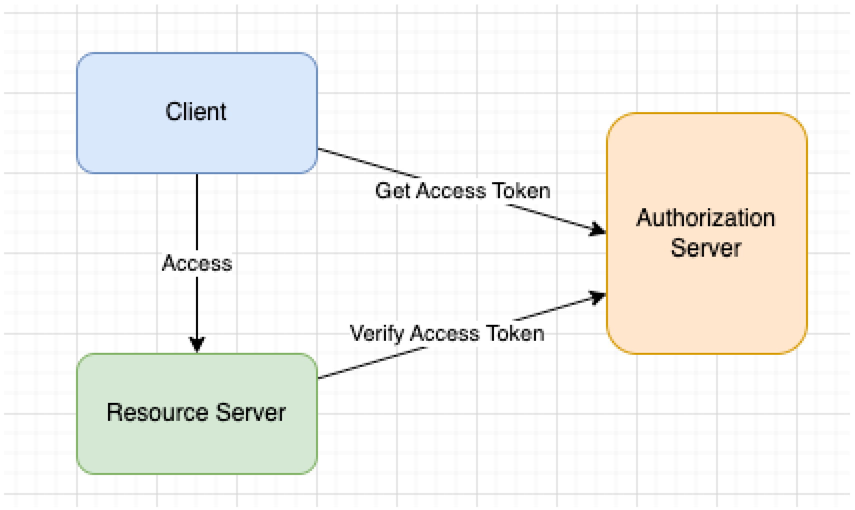
Know who you are on Google

Google OAuth 2.0 Playground and Google will use this information in accordance with their respective terms of service and privacy policies.

This consent page is NOT required in the flow. Many Authorization Servers support 'implicit-consent' mode so that only user authentication is required.

After user authentication and optional consent is finished, the Authorization Server will return an Access Token (The Access Token in practice is usually a JWT token, but it doesn't have to be) to the Email Management App (Client). Then the Email Management App sends a request to the Gmail Server (Resource Server) with the Access Token telling that it needs those emails the end user (Resource Owner) possesses. The Gmail Server checks the request and verifies the Access Token. After that, the Gmail Server returns those resources to the Email Manage App and the workflow is finished.

One thing to mention is that the OAuth workflow doesn't have to involve a user. In this case, the client triggers the OAuth 2.0 flow by itself and will get an access token for its own usage. This is often named as machine-to-machine call.



From an even more abstract view, the whole point of the workflow is to get an Access Token from the Authorization Server so that the client can access the Resource Server for the protected resources.

Please also note, this is just a high-level process of the OAuth 2.0 workflow and in practical situations, there are more details and tweaks going on.

Client Registration

Before the client can interact with the Authorization Server, it needs to be registered in the Authorization Server to provide the following details:

- Client ID
- Client Secret
- Redirect URI or Callback URL (could be multiple)

Client ID and Client Secret are similar to Username and Password. The former is used for Application and the latter is used for human users. By providing the client id and secret to the Authorization Server, it knows the client is a legitimate one to trigger the OAuth flow.

The Redirect URI is where the Authorization Server will do a http browser redirect to after the user authenticates in the Authorization Server.

OAuth 2.0 Grant Types

In part I, we talked about the OAuth 2.0 high level workflow. Now we will dive in with more details and one important part is to understand different grant types. Different grant types are just different workflows to get an Access Token and based on the scenario, we should pick the proper grant type.

Here is the list of Grant Types:

- Authorization Code
- Client Credentials
- PKCE
- Device Code
- Refresh Token
- Implicit (deprecated)
- User Password (deprecated)

The first two, Authorization Code and Client Credentials, are the most often used. Authorization Code is used when an End User is involved (user-to-machine flow) while Client Credentials is used when NO End User is present (machine-to-machine flow).

The last two, Implicit and User Password are legacy flows and will be deprecated in the future.

In this post, we will go through the Authorization Code and Client Credentials workflow, as those two are crucial to understand OAuth 2.0.

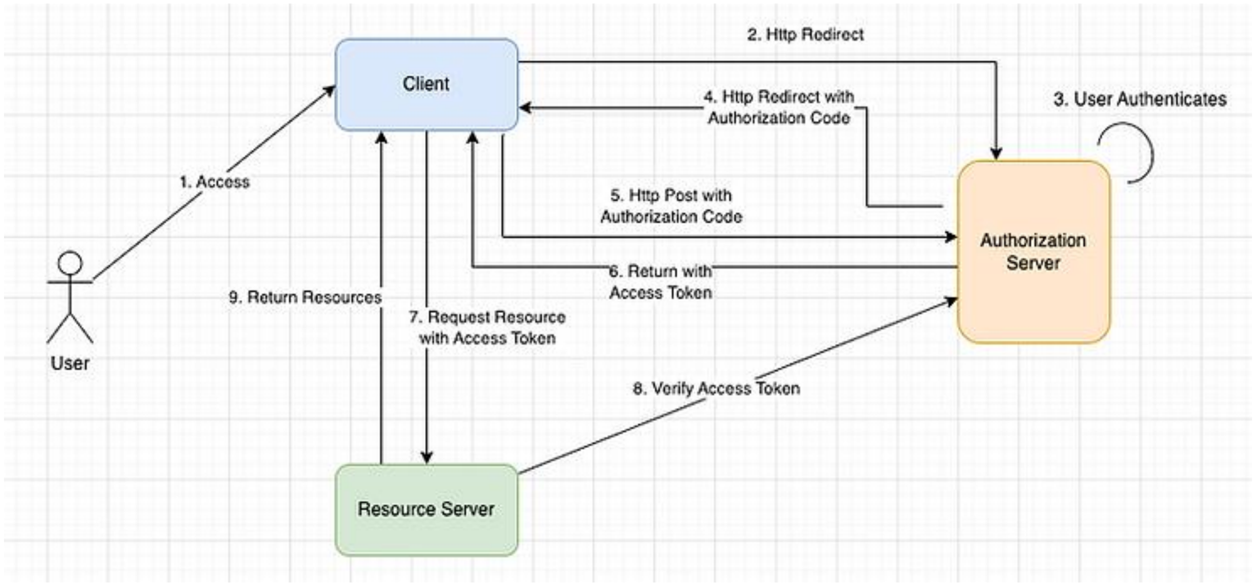
Authorization Code Grant

The Authorization Code grant type is the most commonly used. There are mainly two parts in the process:

User authenticates to Authorization server and Authorization Server returns Authorization Code to the Client (Browser based)
Client uses Authorization Code to exchange an Access Token (No Browser)

Part one will be based on browser redirection and part two will be an Http Post where NO browser is involved.

Note, some Authorization Server (such as ForgeRock) supports step 1 - getting Authorization Code without Browser, meaning that Client can make a REST call to its endpoint providing the same parameters as payload and get back the authorization code. This depends on the Authorization Server.


Client
2. Http Redirect
4. Http Redirect with Authorization Code
3. User Authenticates
1. Access
5. Http Post with Authorization Code
Authorization Server
6. Return with Access Token
9. Return Resources
7. Request Resource with Access Token
User
8. Verify Access Token
Resource Server

Step 1. User Accesses the Client.

User interacts with the Client app, which will trigger an Authorization Code grant workflow.

Step 2. Client Starts the Authorization Code

Remember the first part of the flow is browser-based, so it will start with an URL with some parameters.

https://authorization-server.com/oauth/authorize?response_type=code&client_id=test-client123&redirect_uri=https://oauth-app.com/callback&scope=read&state=PPKswuzDbVL06ankyPO6FoVwi1CdysR5

Notice that https://authorization-server.com/oauth/authorize is the Authorization Server address with the endpoint to start the flow. This is defined by the Authorization Server itself.

- **response_type=code** - This tells the Authorization Server that the Client App is initializing the Authorization Code flow
- **client_id** - The registered Client Name/ID from the Authorization Server
- **redirect_uri** - The URI where Authorization Server will redirect to after authenticating the user
- **scope** - One or more strings (separated by space by default) indicating which permissions the application is requesting
- **state** - A random string to keep track of current OAuth 2 flow transaction; the same value will be returned by the Authorization Server; this is to prevent CSRF attacks

Step 3. User Authentication

In this step, User needs to log in to the Authentication Server. OAuth 2.0 does NOT specify what kind of authentication method is required, so it all depends on the need. Sometimes a simple username/password will do, while other times, MFA (Multi-Factor Authentication) is needed.

After the user is authenticated, Authorization Server might present a consent page to list out permission that the Client App is requesting to access. If the user denies it, the flow will be terminated.

The consent page is optional and a lot of times, the Authorization Server can turn it off to indicate an 'implicit-consent'. This suits some use cases well.

Step 4. Authorization Server Returns the Authorization Code to the Client App

The return of the Authorization Code happens in the URL parameter and will be an Http browser redirect.

https://oauth-app.com/callback?code=2ummcKJvTt3ocgeI3RtdUOEWtZ6wMhmT&state=PPKswuzDbVL06ankyPO6FoVwi1CdysR5

- **code** - This is the authorization code generated by the Authorization Server; it is usually short-lived (between 1-10 mins) and will be invalid once used
- **state** - This will be the same value from the request; if the Client App sees a different value, the flow should be terminated

Step 5. Exchange Authorization Code for an Access Token

Once the Authorization Code is retrieved by the Client App, it will make an Http POST request to the corresponding Authorization Server endpoint to exchange the Authorization Code for an Access Token.

To send out the POST request, you can use CURL from the command line or REST Client Tool like POSTMAN. The POST request will look like below:

```
POST /oauth/token HTTP/1.1
Host: authorization-server.com

grant_type=authorization_code
&code=2ummcKJvTt3ocgeI3RtdUOEWtZ6wMhmT
&redirect_uri=https://oauth-app.com/callback
&client_id=test-client123
&client_secret=xxxxxxxxxx
```

- **grant_type=authorization_code** - This tells the Authorization Server that the Client app is using the Authorization Code flow; do NOT confuse with 'response_type=code' in step 2
- **code** - The same as the value returned by the Authorization Server
- **redirect_uri** - The same redirect URI that was used when requesting the code; this may not be required depending on the Authorization Server
- **client_id** - The same client id that is used to request the code
- **client_secret** - The secret with the registered Client to make sure it is a legitimate Client

Step 6 Authorization Server Returns an Access Token

All the parameters sent by the Client App will be verified by the Authorization Server and then it will send an Http response with the Access Token in the response body.

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache

{
  "access_token":"eyJ0eXAiOiJKV1QiLCciOiJSUzI1NiIs...",
  "token_type":"Bearer",
  "expires_in":3599,
  "scope":"read",
  "refresh_token":"eyJ0eXAiOiJKV1QiAqdFSBzjqfTGAM4B4..."
}
```

Note a Refresh Token is included in the response as well. Access Token usually expires in a couple of hours and then the Client App can use the Refresh Token to call another Authorization Server endpoint to get a new Access Token, without having the same End User to log in the Authorization Server again.

Step 7 Client App Access Resource Server with Access Token

Next, the Client App requests access to the protected resources on the Resource Server.

The Access Token is a bearer token, meaning it is self-contained. Any client possessing this token will be authenticated as long as the token itself is still valid.

The Access Token is put in the Http request Header with header name 'Authorization' and value 'Bearer ' (word 'Bearer' plus a space) inserted before the actual token. The word 'Bearer' is case-insensitive.

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCciOiJSUzI1NiIs...
```

Step 8 Resource Server Verifies the Access Token

Resource Server must always verify the Access Token before granting the Client App the access. Depending on the type of the Access Token, there are different ways to verify it.

One way is to send the Access Token back to the Authorization Server endpoint (introspection) and let it do the work. Another way is that if the Access Token is a JWT (a lot of times is), the Resource Server can decode the Access Token payload and verify expiration, scope, issuer as well as signature.

Token Introspection
By default, a Client can only introspect its own Access Token. Depending on the Authorization Server, a Client can introspect an Access Token issued to other Clients with some special scope values.

```
POST /oauth/introspect HTTP/1.1
Host: authorization-server.com
```

token=eyJ0eXAiOiJKV1QiLCciOiJSUzI1NiIs...
The Authorization Server will respond with the Access Token information.

```
{
   "active": true,
   "scope": "read",
   "client_id": "test-client123",
   "token_type": "Bearer",
   "exp": 1419356238
}
```

There are other endpoints as well such as revoking an Access Token, Jwks and .*wellknown* endpoints. We'll look more on this in OIDC.

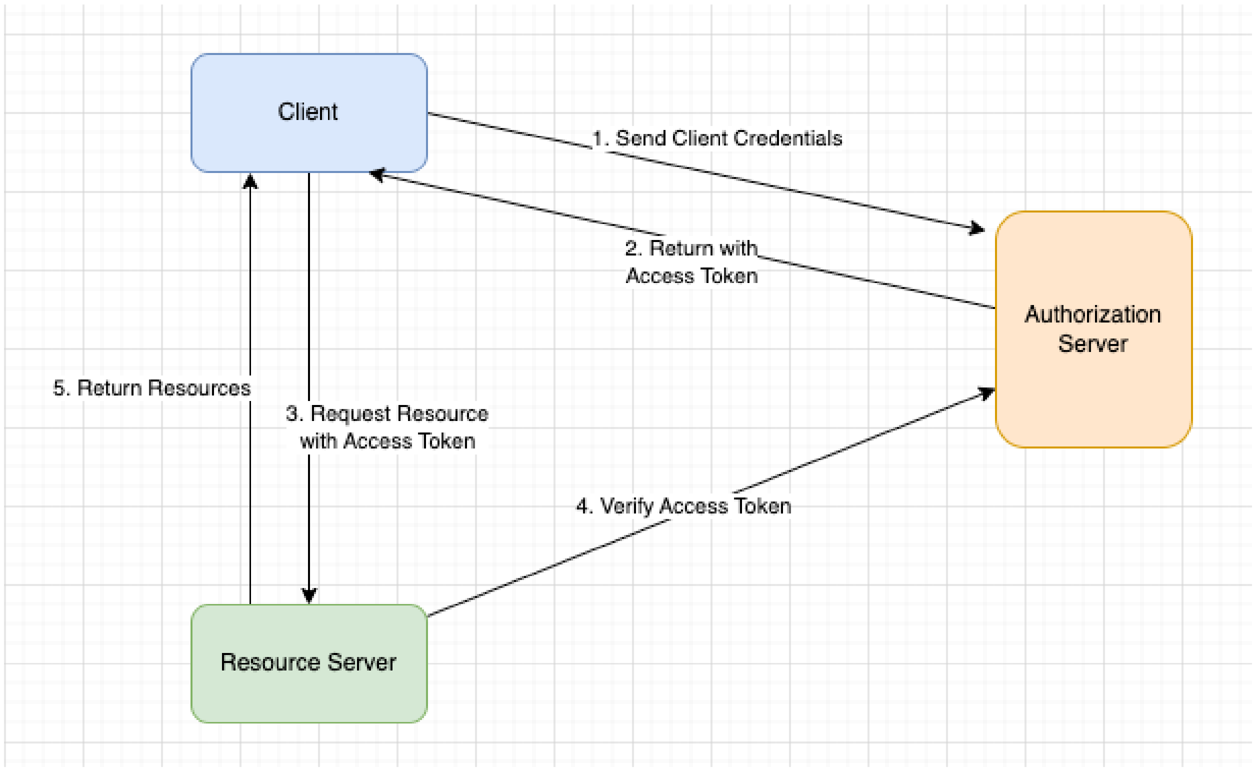
Step 9 Resource Server Returns Resources to the Client App

Once the Access Token is verified, the Resource Server will send the requested resources back to the Client App and the flow is finished.

Client Credentials Grant

The client credentials grant is used when Client App is calling some API resources hosted by the Resource Server. This is considered a machine-to-machine (or server-to-server) call as opposed in the Authorization Code grant where the user is involved.

Client Credentials Grant process is simpler than Authorization Code, where NO redirect is needed. Client App sends the request with Http POST to the Authorization Server and after verification, it will return the Access Token.


Client
1. Send Client Credentials
2. Return with Access Token
Authorization Server
5. Return Resources
3. Request Resource with Access Token
4. Verify Access Token
Resource Server

Step 1. Client Initializes the Client Credentials Flow

This will be a straight Http POST call to the Authorization Server.

POST /oauth/token HTTP/1.1
Host: authorization-server.com

grant_type=client_credentials&client_id=test-client123&client_secret=xxxxxxxxxx&scope=read

Notice that *https://authorization-server.com/oauth/token* is the Authorization Server address with the endpoint to start the flow. This is defined by the Authorization Server itself.

- **grant_type=client_credentials** - This tells the Authorization Server that the Client app is initializing the Client Credentials flow
- **client_id** - The same client id that is used to request the code
- **client_secret** - The secret with the registered Client to make sure it is a legitimate Client
- **scope** - One or more strings (separated by space by default) indicating which permissions the application is requesting

Step 2. Return with Access Token

All the parameters sent by the Client App will be verified by the Authorization Server and then it will send an Http response with the Access Token in the response body.

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache

{
  "access_token":"eyJ0eXAiOiJKV1QiLCJhbGSUzI1NiIs...",
  "token_type":"Bearer",
  "expires_in":3599,
  "scope":"read"
}
```

Note NO Refresh Token will be included in the Client Credentials flow.

The remaining steps are the same as in the Authorization Code grant flow.

Chapter 9 – OAuth 2.0: PKCE

OAuth 2.0 with Proof Key for Code Exchange (PKCE) is an enhanced security mechanism designed to protect against code interception attacks in authorization flows.

PKCE is an extension to the OAuth 2.0 protocol and provides an additional layer of security for native and mobile applications. The PKCE flow ensures that even if an attacker intercepts the authorization code, they cannot exchange it for an access token without possessing the original cryptographic proof.

This flow adds a dynamic secret called the code verifier to the OAuth 2.0 authorization request, which is later verified during the token exchange process. By using PKCE, developers can significantly enhance the security of their applications, mitigating the risk of authorization code interception and unauthorized access to protected resources.

Key Points

- **Client Types:**
  - Confidential Clients: Can securely store client secrets.
  - Public Clients: Cannot store secrets securely (e.g., SPAs, mobile apps).
- **PKCE Motivation:**
  - Prevents authorization code interception attacks in public clients.
  - Eliminates the need for client secrets in public clients.
- **PKCE Flow:**
  1. Client generates a `code_verifier` (random string) and `code_challenge` (hashed `code_verifier`).
  2. Client includes `code_challenge` and `code_challenge_method` in the authorization request.
  3. Authorization Server stores the `code_challenge` and issues an authorization code.
  4. Client exchanges the code for an access token, including the `code_verifier`.
  5. Authorization Server verifies the `code_verifier` against the stored `code_challenge`.
  6. If verified, the Authorization Server issues an access token.
- **Benefits:**
  - Enhances security for public clients.
  - Enables the use of the Authorization Code grant type without client secrets.

Authorization Code with PKCE (Proof Key for Code Exchange)

Authorization Code with PKCE (pronounced 'pixy'), or simply PKCE grant, is designed to be used with SPA (Single Page Apps) or Mobile Apps as a modern solution. Before we dive in, let's look at some pre-knowledge to better understand it.

Public Client and Confidential Client

According to OAuth specs, there are two types of Clients that can be registered with an Authorization Server: Public Client and Confidential Client

Confidential Client - Client Applications that are able to securely store Client Secret and keep it safe
Public Client - Client Applications that NOT able to store Client Secret and hence, can NOT use secret to authenticate a Client

Based on the definition, when registering a Confidential type Client, a Client Secret will be provided together with the Client ID by the Authorization Server, while in contrast, NO secret will be provided to the Public Client.

The reason for the above is that Confidential Client refers to Applications within a company, which usually have a backend using technologies like Java, .NET, NodeJS etc. These applications can store a password or secret securely e.g. in a database.

On the other hand, Public Client refers to SPA or Mobile Application. SPA is written in JavaScript and usually has NO way to store a password/secret other than source code. Native Mobile Applications can NOT store password/secret securely either. The binaries from one's cellphone can be decompiled and credentials could be compromised.

The Issue with Public Client

Remember the standard Authorization Code flow is a two-step workflow, where the first step is a browser redirect to get an Authorization Code and second step is to exchange the code for Access Token with Http POST. Specifically, the second step needs to POST Client ID and Client Secret in addition to the Authorization Code.

If the Client App is a SPA or Native Mobile App (meaning a Public Client), it can NOT store secrets and hence can NOT use Client ID and Secret to authenticate itself in the Authorization Server.

To avoid the issue of storing secrets, one option is to use Implicit Grant instead of Authorization Code grant, where only Client ID and Redirect URI are used to authenticate the client in

Authorization Server (NOT a secure way). Furthermore, since Access Token is returned in the URL, it has added risk of exposing the Access Token. This is why Implicit Grant is deprecated.

Now, if we go back to the Authorization Code grant and remove the Client Secret, only relying on Authorization Code, Client Id and Redirect URI to exchange for an Access Token. It looks plausible but actually is vulnerable to a type of attack called Authorization Code Interception Attack.

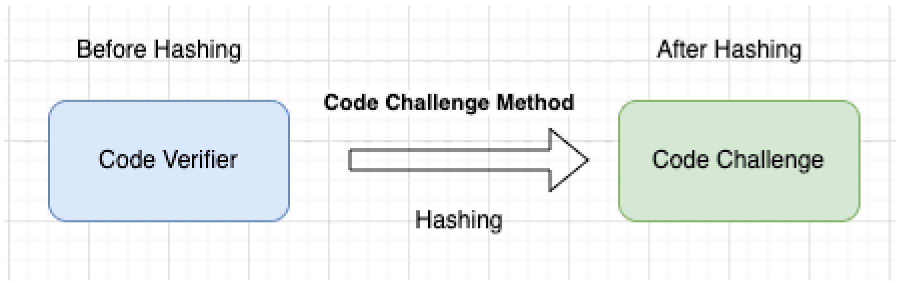
To fix this, some additional parameters: Code Verifier, Code Challenge and Code Challenge Method are added to the flow and this is the PKCE flow.

How PKCE Flow Works

Let's first understand what area Code Verifier, Code Challenge and Code Challenge Method.

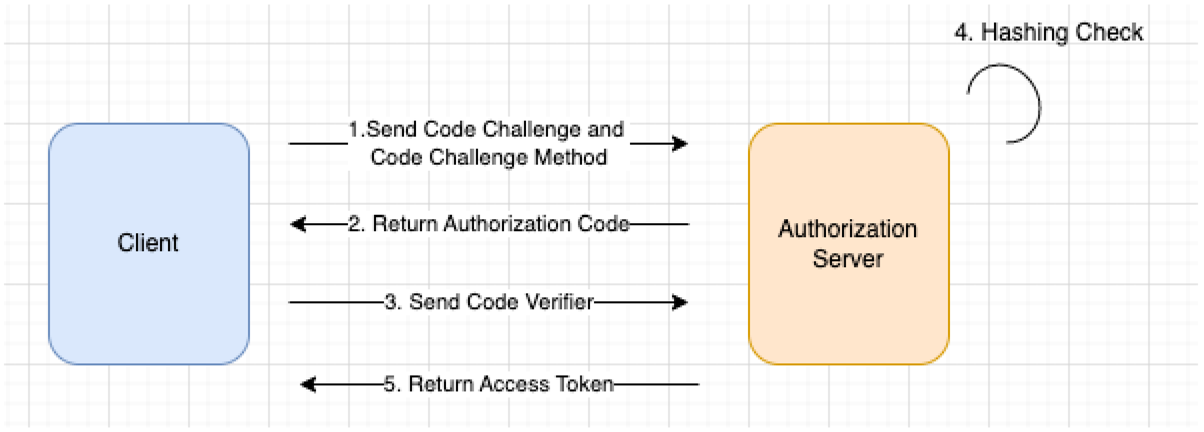
- **Code Verifier** - A cryptographically random string generated by the Client App. This value will be different each time a PKCE flow is initialized.
- **Code Challenge** - This is the hashed value from Code Verifier (You need to understand what Hashing is).
- **Code Challenge Method** - The hash method/function used to hash Code Verifier to generate Code Challenge

Note the default Code Challenge Method is 'plain', which does NOT do any hashing on the original Code Verifier and it is recommended to replace it with some hashing algorithm such as S256 (SHA256).



On a high level, the flow is:

1. Client generates a random string (Code Verifier), and then hash it with a certain hash function(Code Challenge Method). Then the Client sends the hashed value (Code Challenge) and hashing method to the Authorization server.
2. Authorization Server stores the Code Challenge value and Code Challenge method and returns the Authorization Code to Client.
3. In exchanging Authorization Code for Access Token, the Client sends the Authorization Code together with, this time, Code Verifier to the Authorization Server.
4. Authorization Server runs a hashing check with the Code Verifier and the Code Challenge & Code Challenge Method received in the first step.
5. If the check is valid, Access Token will be returned.



PKCE Flow Steps

1. Client sends (browser redirect) code_challenge and code_challenge_method to Authorization Server

https://authorization-server.com/oauth/authorize?client_id=test-client123&response_type=code&scope=read&redirect_uri=https://oauth-app.com/callback&code_challenge=bc6c6abccd6fec88f1772fa263f824208db050306562e80f3f8684a82f6b9818&code_challenge_method=S256&state=abc123

Notice that *https://authorization-server.com/oauth/authorize* is the Authorization Server address with the endpoint to start the flow. This is defined by the Authorization Server itself. Also some Authorization Server (such as ForgeRock) supports getting Authorization Code without Browser, meaning that Client can make a REST call to its endpoint providing the same parameters as payload and get back the authorization code. This depends on the Authorization Server.

- **client_id** - The registered Client Name/ID from the Authorization Server
- **response_type=code** - This tells the Authorization Server that the Client App is initializing the Authorization Code flow (PKCE is a modified Authorization Code flow)
- **scope** - One or more strings (separated by space by default) indicating which permissions the application is requesting
- **redirect_uri** - The URI where Authorization Server will redirect to after authenticating the user
- **code_challenge** - Hashed value of original Code Verifier
- **code_challenge_method** - Hash algorithm used to generate the code_challenge; 'S256' means SHA256
- **state** - A random string to keep track of current OAuth 2 flow transaction; the same value will be returned by the Authorization Server; this is to prevent CSRF attacks

2. Authorization Server returns Authorization Code

Based on the Authorization Code flow, the user needs to authenticate in the Authorization Server and consent might be needed based on the server configuration.

The return of the Authorization Code happens in the URL parameter and will be an Http browser redirect.

https://oauth-app.com/callback?code=LtwzXMel17MNjrel4KIcaZoz24eGsLtQ&state=abc123

- **code** - This is the authorization code generated by the Authorization Server; it is usually short-lived (between 1-10 mins) and will be invalid once used
- **state** - This will be the same value from the request; if the Client App sees a different value, the flow should be terminated

3. Exchanging Authorization Code for an Access Token

Client sends out the Authorization Code and Code Verifier with an Http POST. You can see NO Client Secret is included in this request.

To send out the POST request, you can use CURL from the command line or REST Client Tool like POSTMAN. The POST request will look like below:

```
POST /oauth/token HTTP/1.1
Host: authorization-server.com

grant_type=authorization_code &code=LtwzXMel17MNjrel4KIcaZoz24eGsLtQ
&redirect_uri=https://oauth-app.com/callback
&client_id=test-client123
&code_verifier=LtwzXMel17MNjrel4KIcaZoz24eGsLtQ
```

The client_id and redirect_uri must match the previous call.

- **grant_type=authorization_code** - This tells the Authorization Server that the Client app is using the Authorization Code flow(again, PKCE is still a modified Authorization Code flow); do NOT confuse with 'response_type=code' in step 2
- **code** - The same as the value returned by the Authorization Server
- **redirect_uri** - The same redirect URI that was used when requesting the code; this may not be required depending on the Authorization Server
- **client_id** - The same client id that is used to request the code
- **code_verifier** - The original string used to generate the code_challenge

4. Authorization Server Check code_verifier, code_challenge and code_challenge_method

Besides checking the code, redirect_uri and client_id, the Authorization Server will do a hashing on code_verifier with code_challenge_method and compare the generated value with the code_challenge value sent by the client. If the two values do NOT match, Authorization Server will reject the request and terminate the flow.

5. Authorization Server Returns Access Token

After validation of all parameters and hashing, Authorization Server will return the Access Token in the payload.

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache

{
  "access_token":"eyJ0eXW13A8OZ5AbigKfDmKAPHHGTU7cUbNdrw...",
  "token_type":"Bearer",
  "expires_in":3599,
  "scope":"read"
}
```

Similar to Authorization Code flow, Refresh Token can be issued by Authorization Server as well.

Chapter 10 – OIDC

OpenID Connect (OIDC) is an identity layer built on top of the OAuth 2.0 protocol, designed to provide secure and standardized user authentication and authorization in modern web applications.

OIDC enables applications to verify the identity of users, obtain user profile information, and securely authenticate users using various identity providers, such as Google, Facebook, or Microsoft.

With OIDC, developers can delegate the responsibility of user authentication to trusted identity providers, reducing the complexity of managing user credentials. By leveraging JSON Web Tokens (JWTs) and standardized protocols, such as OpenID Connect Discovery, OIDC simplifies the integration of identity services into applications and ensures interoperability between different identity providers and relying parties.

With its emphasis on user-centric authentication and secure identity federation, OIDC has become a widely adopted standard for implementing robust and scalable identity solutions in the digital era.

Key Points

- **OIDC Overview:**
    - Extends OAuth 2.0 with an identity layer.
    - Returns an ID Token (JWT) containing user information alongside the Access Token.
    - Enables clients to verify user identity.
- **Key Differences from OAuth 2.0:**
    - Renamed roles: Authorization Server -> OIDC Provider (IDP), Client -> Relying Party (RP).
    - ID Token is always a JWT.
    - Standardized scopes: `openid`, `profile`, `email`.
    - ID Token claims: `sub`, `iss`, `aud`, `iat`, `exp`, `acr`, email.
- **OIDC Grant Flows:**
    - Uses the same grant types as OAuth 2.0.
    - ID Token is primarily used with Authorization Code and PKCE flows.

- **OIDC Endpoints:**
    - Userinfo Endpoint: Retrieves user claims using an Access Token.
    - JWKS Endpoint: Provides public keys for JWT verification.
    - /.wellknown Endpoint: Returns OIDC metadata about the IDP.
- **Single Sign-On (SSO):**
    - Allows users to access multiple applications with a single set of credentials.
    - Benefits: Enhanced security, increased productivity, reduced IT costs, improved user experience.
    - OIDC is a popular standard for implementing SSO.
- **Modern SSO with OIDC:**
    - OIDC provider acts as the central authentication authority.
    - Applications (clients) interact with the OIDC provider for authentication.
    - Sessions are managed at both the client application and OIDC provider levels.
    - Achieves SSO by leveraging the OIDC provider's session.

What OIDC (OpenID Connect) is

OIDC stands for "OpenID Connect". It is built on top of OAuth 2.0 protocol with an added Identity Layer, meaning OIDC is a superset of OAuth 2.0. In simple words, it allows another token - ID Token (Identity Token) to be returned together with the Access Token. While Access Token is used to access some resources, ID Token itself contains the User Information who has authenticated to the Authorization Server.

Why OIDC

Generally speaking, the whole point of OAuth 2.0 is for clients to get an Access Token so that it can access some resources. This works well for privilege delegation like explained in OAuth 2.0 chapters.

However, that Access Token does NOT contain any User Information (e.g. username, email, role...) who has authenticated to the Authorization Server and this User Information is often critical for a lot of functions in a system. Hence, after the OAuth 2.0 protocol came out and became popular, technology groups started to realize that there is a strong need to obtain that User Information somewhere during the process of running OAuth 2.0 flows.

Then the idea is why not just add another token that contains the User Information and return it together with the Access Token, as the OAuth 2.0 protocol is already popular? So the ID Token is added when Access Token is returned. Beyond just adding the ID Token, we need to make it more comprehensive and become a standard so that everyone can follow it and hence, the birth of OIDC.

Understanding OIDC

Since OIDC is OAuth 2.0 under the hood, the core parts are the same for both two (such as grant types). Yet, there are still some naming changes and extensions in OIDC compared to OAuth 2.0.

1. Roles

The Authorization Server in OAuth 2.0 is renamed as OIDC Provider (also called Identity Provider or simply IDP).

The Client in OAuth 2.0 is renamed as Relying Party (RP).

Some of the terms are used interchangeably as in nature, they are the same.

2. Use of JWT

In OAuth 2.0, JWT is pretty much the go-to format for Access and Refresh Tokens. In OIDC, an ID Token is always a JWT token. Nowadays, all tokens are pretty much JWT tokens and that includes Access Token, ID Token and Refresh Token.

3. Standardized Scope

Scope in OAuth defines the level of privileges of an Access Token. When a Client registers with Authorization Server, it needs to define the available scope list and there is no standard on that, meaning any client can define whatever values they want.

OIDC addresses scopes to the ID Token specifically. It provides some standard frequently used scopes while allowing custom scopes. Remember scope can be a list of string values and whenever a scope value is specified, the corresponding claims (or attributes) are returned in the ID Token.

Here are a list of the predefined standard scopes:

Scope	Required	Description

openid	Yes	default mandatory scope; returns a 'sub' claim which represents a unique identifier for the authenticated user
profile	No	returns User's profile claims such as name, first_name, last_name...
email	No	returns email claim and email_verified, which is a boolean

4. ID Token

The ID Token is JWT format and its payload includes a list of claims (attributes). Here are some basic ones.

ID Token Claim	Meaning
sub	Subject - Identifier for the User at the Issuer
iss	Issuer - which authority issued this ID Token
aud	Audience - which Client the ID Token was generated for
iat	Issue Time - when the ID Token was generated
exp	Expiration Time - when the ID Token is set to expire
acr	Authentication Context Reference - The security context values requested for Authentication
Email	User email

For a more detailed list, you can check https://openid.net/specs/openid-connect-core-1_0.html#StandardClaims

5. OIDC Grant Flows

The grant flow for OIDC is the same with OAuth 2.0 as described previously.

- Authorization Code
- Client Credentials
- PKCE
- Device Code
- Refresh Token

- Implicit (deprecated)
- User Password (deprecated)

Those flows can be categorized into two groups: human-to-server and server-to-server

Since ID Token is addressed to a User, the ID Token won't be returned in the server-to-server call, as there is no User authentication happening.

From the list above, Authorization Code and PKCE grant flows are the main ones used to get an ID Token. Although Device Code flow involves user authentication, ID Token is NOT usually used in the flow. Client Credentials and Refresh Token flows are server-to-server calls, so no ID Token will be returned. Implicit and User Password flows involve User, but is deprecated due to security concerns and shouldn't be used for modern platforms.

6. Endpoints

Besides the original OAuth 2.0 endpoints (e.g. token introspection), there are some other endpoints worth mentioning as well.

Userinfo Endpoint

New to OIDC this endpoint allows you to make a request using an appropriate Access Token to retrieve identity information (claims) about the authenticated User. Here is an example:

```
{
   "sub":"uid4BTXdCDI6TV4m3g3",
   "name":"John Doe",
   "first_name":"John",
   "last_name":"Doe",
   "locale":"en-US",
   "updated_at":1311280970,
   "email":"john.doe@example.com",
   "email_verified":true,
   "address":{
      "street_address":"123 xxx Blvd.",
      "locality":"Boston",
      "region":"MA",
      "postal_code":"02115",
      "country":"US"
   },
   "phone_number":"+1 (777) 888-9999"
}
```

JWKS (Json Web Key Set) Endpoint

JWKS (pronounced 'jawks') is a set of keys containing the public keys used to verify any JWT issued by the Authorization Server and signed using the asymmetric encryption algorithm (e.g. RS256). Below is an example where the first entry is shown:

```
{
   "keys":[
      {
         "alg":"RS256",
         "kty":"RSA",
         "use":"sig",
         "x5c":["MIIC+DCCAeCgAwIBABIGjYW6hFpn2VBAMTGGN1c3Rvb..."],
         "n":"yeNlzlub9codqEztjfU_SPCT1Axz6if3lHpq_g4Sz-cbe4....",
         "e":"AQAB",
         "kid":"NjVBRjY5MDlCMUIwNzU4RTA2QzZFM",
         "x5t":"NjVBRjY5MDlCMUIwNzU4RTA2QzZFMDQ4QzQ2MDA"
      }
   ]
   ...
}
```

Each property in the key is defined by the JWK spec

Property Name	Description
alg	The cryptographic algorithm used for the key
kty	The cryptographic family name for the key
use	How the key is supposed to be used; 'sig' means signature; 'enc' means encryption
x5c	The x.509 cert chain. The first entry is used to verify the JWT signature.
n	The modulus value for RSA the public key
e	The exponent value for the RSA public key
kid	The key id - a unique identifier for the key
x5t	The thumbprint of the x.509 cert

The /.wellknown Endpoint

The .*wellknown* endpoint (don't miss the dot) returns OIDC metadata about the Authorization Server or OIDC Server. This endpoint usually does NOT need any authentication. Here is an example:

```
{
   "issuer":"https://authorization-server.com",
       "authorization_endpoint":"https://authorization-
server.com/oauth/authorize",
   "token_endpoint":"https://authorization-server.com/oauth/token",
   "userinfo_endpoint":"https://authorization-
server.com/oauth/userinfo",
   "registration_endpoint":"https://authorization-
server.com/oauth/clients",
   "jwks_uri":"https://authorization-server.com/jwks",
   "response_types_supported":[
      "code",
      "code id_token",
      "code token",
      "code id_token token",
      "id_token",
      "id_token token"
   ],
   "response_modes_supported":[
      "query",
      "fragment",
      "form_post"
   ],
   "grant_types_supported":[
      "authorization_code",
      "implicit",
      "refresh_token",
      "password"
   ],
   "subject_types_supported":[
      "public"
   ],
   "id_token_signing_alg_values_supported":[
      "RS256"
   ],
   "scopes_supported":[
      "openid",
      "email",
      "profile",
      "address",
      "phone"
   ],
   "token_endpoint_auth_methods_supported":[
      "client_secret_basic",
      "client_secret_post",
      "client_secret_jwt",
      "none"
   ],
   "claims_supported":[
      "iss",
      "ver",
      "sub",
      "aud",
      "iat",
      "exp",
      "jti",
      "auth_time",
      "amr",
      "idp",
      "nonce",
      "name",
      "preferred_username",
      "first_name",
      "last_name",
      "email",
      "email_verified",
      "profile"
   ],
   "introspection_endpoint":"https://authorization-
server/oauth/introspect",
   "introspection_endpoint_auth_methods_supported":[
      "client_secret_basic",
      "client_secret_post",
      "client_secret_jwt",
      "none"
   ],
   "revocation_endpoint":"https://authorization-server/oauth/revoke",
   "revocation_endpoint_auth_methods_supported":[
      "client_secret_basic",
      "client_secret_post",
      "client_secret_jwt",
      "none"
```

```
    ]
}
```

SSO (Single Sign-On)

SSO has been prevalent in organizations for years, but its importance is a lot of times overlooked. As the big trend is moving to cloud and taking advantage of third-party services, seamless and secure access to multiple applications is essential for maintaining operation efficiency and a smooth customer experience. This is done via SSO.

SSO itself is an authentication mechanism to allow users to securely access multiple applications using just one set of credentials. Furthermore, with SSO, a user doesn't need to sign on repeatedly to different applications, meaning the user needs to sign on just once and he will gain access to all authorized applications, websites and data from an organization or multiple related organizations.

Why SSO is Important

There are many benefits of using SSO, such as enhanced security, increased productivity, lowered IT costs and improved user experience.

Enhanced Security

Cybercriminals usually target usernames and passwords. In other words, the more accounts a user has, the more management overhead and vulnerabilities it has. Since users only need to remember one credential, they tend to use more complex and secure passwords.

A usual misconception about SSO is that if this only account is stolen, the whole system is compromised. While that seems true at first glance, the reality is that with good practices, the probability of compromising that account is significantly lowered.

A great strategy is to use MFA (Multi-Factor Authentication) with SSO. MFA requires a user to present at least two different types of means to authenticate himself. For example, a username/password and a push notification to a cell phone.

As a result, the security benefits of SSO far outweighs its issue.

Increased Productivity

SSO increases employee productivity by reducing the time they spend signing on and managing passwords. More often than not, an employee needs to interact with multiple applications in the day-to-day work. While he is spending time logging into different accounts, remembering them and retrieving them could be some other overheads, plus changing and resetting passwords.

The time and effort can be saved with one account.

Lowered IT Costs

Studies have shown that more than 50% of help desk calls are about password issues. In other words, the more passwords a user has, the more work the IT help desk will have. Furthermore, if an organization has some specific password policy that requires a long and complicated password, the burden could be increased more. SSO will help in reducing this kind of cost.

SSO Improves User Experience

With the move to cloud, employees are starting to use more apps in the work. A lot of accounts could be frustrating to deal with. Signing-in once not only increases productivity, it enhances employees' job satisfaction as well.

SSO also improves customer experience. Sometimes, a customer abandons his cart simply because of a forgotten password or password reset issue. SSO alleviates this kind of issue and provides a seamless experience.

OIDC as a SSO Standard

There are two popular SSO standards that are popular for SSO: SAML and OIDC.

SSO doesn't have to be based on standards, but using standards has a lot of benefits and it has become a general practice in industry to use SAML or OIDC for SSO.

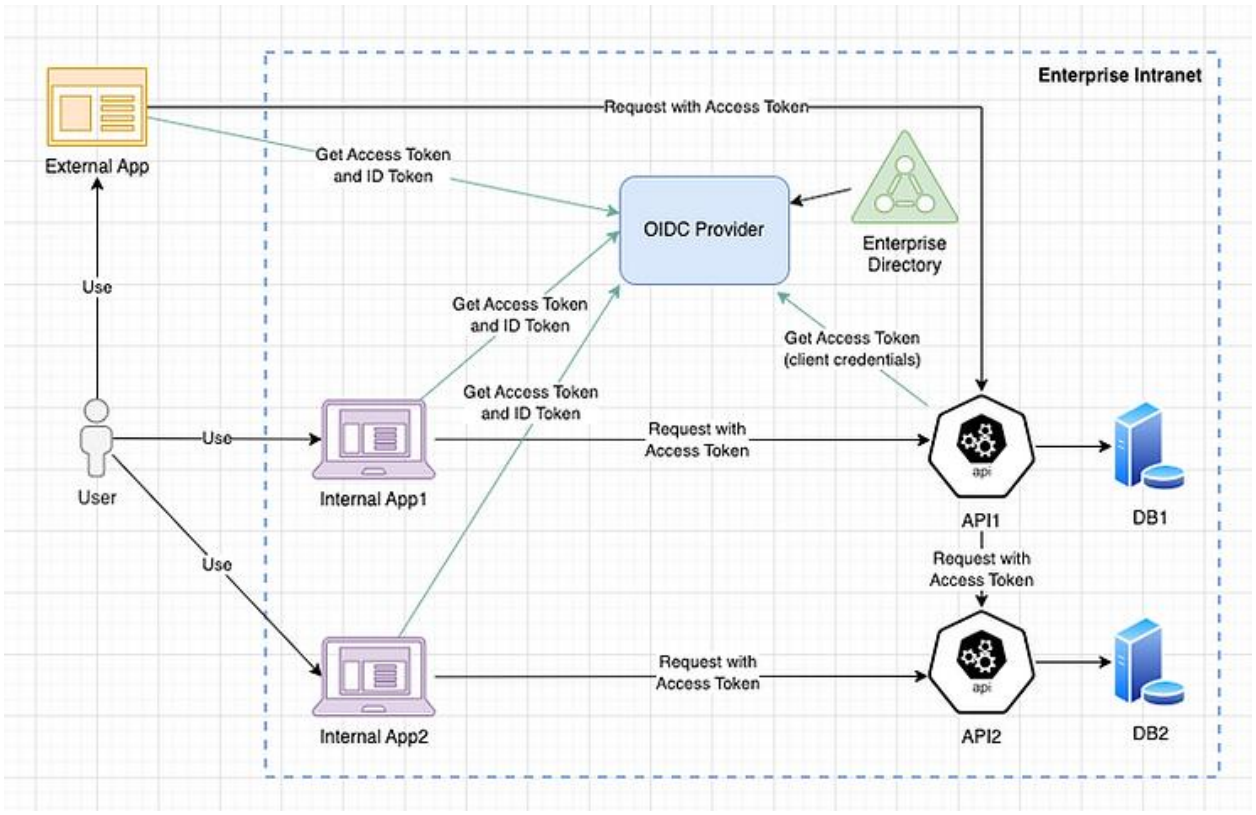
We know that OAuth 2.0 by itself is more of a privilege delegation protocol, and OIDC adds an identity layer on top of OAuth 2.0, which makes it very suitable for SSO. SAML is another popular SSO standard, especially in enterprise SSO. We will focus on OIDC in this post.

Modern SSO Integration Pattern with OIDC

Let's take a look at an example to understand how modern SSO works with OIDC.

In the below diagram:

- An OIDC provider acts as the core of the SSO system within an enterprise intranet
- There are internal and external apps (Client) that are interacting with this system
- API1 and API2 are API resources (Resource Server) that client application can access and they have their own databases
- The external/internal application can be either a public client or a confidential client depending on the settings of the context
- The OIDC provider is connected to an enterprise directory where the user accounts are stored
- Users can be either within intranet or external facing



Enterprise Intranet
External App
Request with Access Token
Get Access Token and ID Token
OIDC Provider
Enterprise Directory
Use
Get Access Token and ID Token
Get Access Token (client credentials)
Get Access Token and ID Token
User
Use
Internal App1
Request with Access Token
API1
DB1
Request with Access Token
Use
Internal App2
Request with Access Token
API2
DB2
api

Authorization Code Flow

An example Authorization Code flow would be that a user starts to use Internal App1, the latter redirects the user to OIDC Provider for authentication. After the user authenticates, the OIDC provider returns Access Token and ID Token. Internal App1 extracts user information from the ID Token and processes some business logic. Then the Internal App1 sends a request with the Access Token to the API1. API1 verifies the Access Token and returns the API resources to Internal App1. The app processes the resources and further assists user operations.

Client Credentials Flow

Another example with Client Credentials flow would be that API1 needs to request some resources from API2. This would be a server-to-server (or machine-to-machine) call. API1 first gets an Access Token (No ID Token) from the OIDC server and then sends the request along with the Access Token to API2. API2 verifies the Access Token and returns the resources.

Sessions and How SSO is Achieved

Usually the client applications manage user sessions by themselves, meaning each client application will maintain a user session for its own and separate client apps have separate user sessions. This is a pure client application session and has nothing to do with the OIDC provider. The good thing for this kind of app session is that whenever the user comes back to the same client app where a user session exists, the user doesn't need to authenticate anymore.

However, if no session exists for the client application, it will redirect the user to the OIDC provider for authentication and once authenticated, the OIDC server will create a user session too. Besides that, the OIDC provider will return an ID Token to the client application. Then the client app extracts user info from the ID Token so that it knows which user it is and hence creates a client app session for that user.

Next, when the user tries to access another client app, he is redirected to the OIDC provider again. But this time, since the user has already authenticated to the OIDC provider (in the last transaction) and a user session does exist, the user doesn't need to authenticate anymore. The Access Token and ID Token will be directly returned to the client app.

In all cases, as long as the apps are interacting with the same OIDC provider, only one account is needed and only one login is needed during a certain time. This is how SSO is achieved.

Chapter 11 – SAML

Similar to OAuth and OIDC, SAML (Security Assertion Markup Language) is another essential protocol used in the IAM world. Even though it was originally developed in 2006 and people say that OAuth/OIDC is replacing SAML, the reality is that SAML is still widely and actively being used and deployed in the industry today. The reason is that there are some features that SAML has but OAuth/OIDC doesn't. Hence, having a good understanding is crucial for any IAM professionals.

The most widely used version is SAML 2.0, which was developed to fix some security issues in SAML 1.0, and when people talk about SAML, it is usually SAML 2.0 they are referring to. We will follow that fashion in this post.

Key Points

- **SAML Overview:**
  - XML-based standard for exchanging authentication and authorization data.
  - Used for SSO, cross-domain SSO, and identity federation.
- **SSO vs. Cross-Domain SSO vs. Identity Federation:**
  - SSO: Single sign-on within the same domain.
  - Cross-Domain SSO: SSO across different domains (often different organizations).
  - Identity Federation: Enables users to access resources across different domains using the same digital identity.
- **IDP (Identity Provider) and SP (Service Provider):**
  - IDP: Authenticates users and issues assertions.
  - SP: Provides services and relies on IDP for authentication.
- **SAML Flows:**
  - SP-Initiated Flow: User tries to access SP, SP redirects to IDP for authentication, IDP responds to SP with assertion.
  - IDP-Initiated Flow: User authenticates at IDP, IDP forwards assertion to SP.
- **SAML Setup:**
  - Circle of Trust (COT): Establishes trust between IDP and SP.

- SAML Metadata: XML document containing information about SAML entities (IDP, SP).
    - Includes entityID, SSO descriptor (endpoints, certificates, keys, etc.).
- Metadata Exchange: IDP and SP exchange metadata to establish trust.

- **SAML Messages and Bindings:**

    - SAML Messages: XML-based messages (e.g., AuthnRequest, Response) exchanged between IDP and SP.
    - SAML Bindings: Define how messages are transported (HTTP POST, HTTP Redirect, HTTP Artifact, SOAP).

- **SAML Request:**

    - Sent from SP to IDP to initiate authentication.
    - Contains information like IssueInstant, Destination, ProtocolBinding, AssertionConsumerServiceURL, Issuer, NameIDPolicy, RequestedAuthnContext.

- **SAML Response and Assertion:**

    - SAML Response: Sent from IDP to SP after authentication.
    - Contains a SAML Assertion with user identity information (e.g., Subject, Issuer, Conditions, Audience, AuthnContext, Attributes).

SAML Overview

SAML was published by the open standard consortium OASIS (Organization for the Advancement of Structured Information Standards) in 2006 to address the issue of repeated password inputs and SSO (Single Sign-On) was introduced during that time.

At that time, JSON or YAML was not quite adopted yet and XML was popular, so SAML was developed based on XML.

The major use case scenarios for SAML are: SSO/Cross-Domain SSO and Identity Federation. Let's understand them first before diving into SAML.

SSO vs. Cross-Domain SSO vs. Identity Federation

When talking about SAML, sometimes you will encounter the so-called 'Cross-Domain SSO' and 'Identity Federation'. It's good to know a little bit about SSO, Cross-Domain SSO and Identity Federation.

SSO
SSO could be referring to just the general concept of single sign-on, that is when you have multiple applications and once you sign into one application, you don't have to sign into other applications. These applications could be within the same domain or from different domains. However, when SSO is used against Cross-Domain SSO, the former usually means SSO within the same domain.

The implementation for same-domain SSO is like a typical setup where you have a centralized IAM platform and other applications will redirect users to this centralized IAM platform for authentication. This redirect mechanism could be based on standard protocol like SAML or OIDC. It could also be something custom developed. The goal is to have all apps integrated with the centralized IAM platform so that SSO can be done with those Apps.

Cross-Domain SSO
Cross-Domain SSO, like its name, is for SSO from different domains. When applications are from different domains, they could be from different departments within an organization, but more often than not, they are applications from other organizations.

So here we are talking about SSO across different companies or organizations, whose infrastructure environment could be very different from same-domain SSO. Custom development is usually limited, as one company won't easily allow another company to access their APIs or other resources. In this case, it's better to use standard protocols like SAML or OIDC. For enterprise-to-enterprise SSO integration, SAML is sometimes preferred over OIDC.

Identity Federation
Identity Federation (or Federated Identity) is a system which allows identities from different enterprises (domains) to use the same digital identity to access all applications and networks.

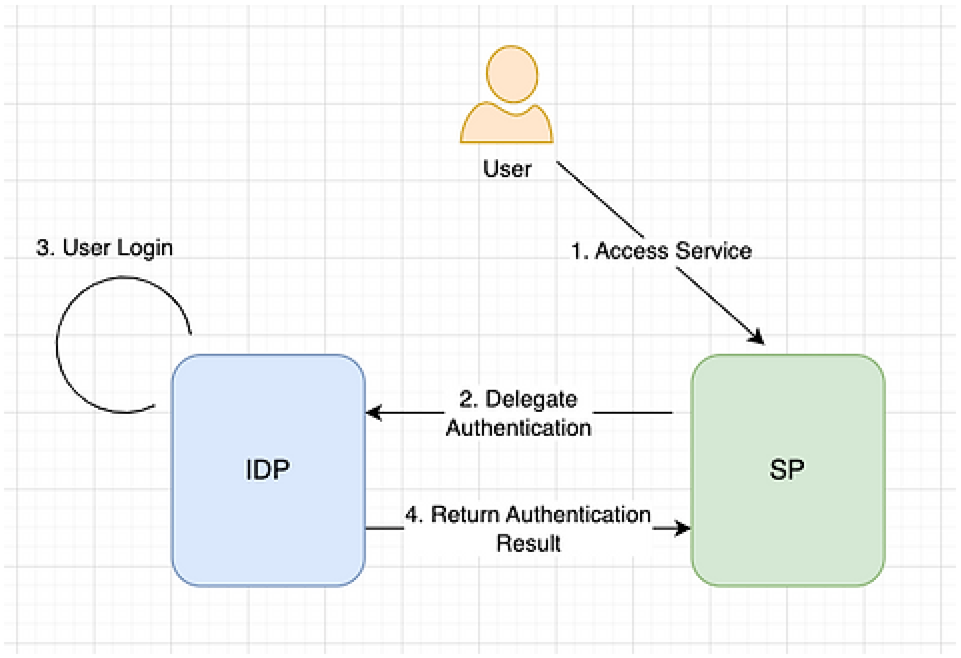
In a general sense, SSO is needed to fulfill Identity Federation, and since Identity Federate usually deals with different organizations/enterprises, it is closer to the concept of Cross-Domain SSO.

When identities are federated across multiple organizations, their profile information is shared to those organizations as well so that the organizations can operate more efficiently using that set of identity profiles. In order for that to happen, besides the trusted relationship established for Cross-Domain SSO, organizations need to have an agreement on what profile information to be shared as well. After that, protocols like SAML or OIDC can be leveraged again to implement it.

IDP (Identity Provider) and SP (Service Provider)

Now, let's jump on the boat of SAML. There are two major parties in SAML: IDP and SP, they are both called SAML entities.

IDP (Identity Provider) is the party responsible for Authenticating the user while SP (Service Provider) is the party responsible for providing services, who will NOT authenticate the user but delegate authentication to IDP.



SP-Initiated Flow

The reason for delegating authentication to IDP is that in this case IDP can act as a centralized platform to authenticate users, that is when there are multiple SPs, they can all rely on IDP for authentication. This will help to fulfill SSO, because once a user login to IDP, IDP will maintain an active user session (e.g. browser cookie) and next time when the same user is redirected to IDP from another SP, he doesn't need to login anymore.

IDP is usually connected to a centralized User Directory where all user profile information for authentication are present.

SP-Initiated Flow and IDP-Initiated Flow

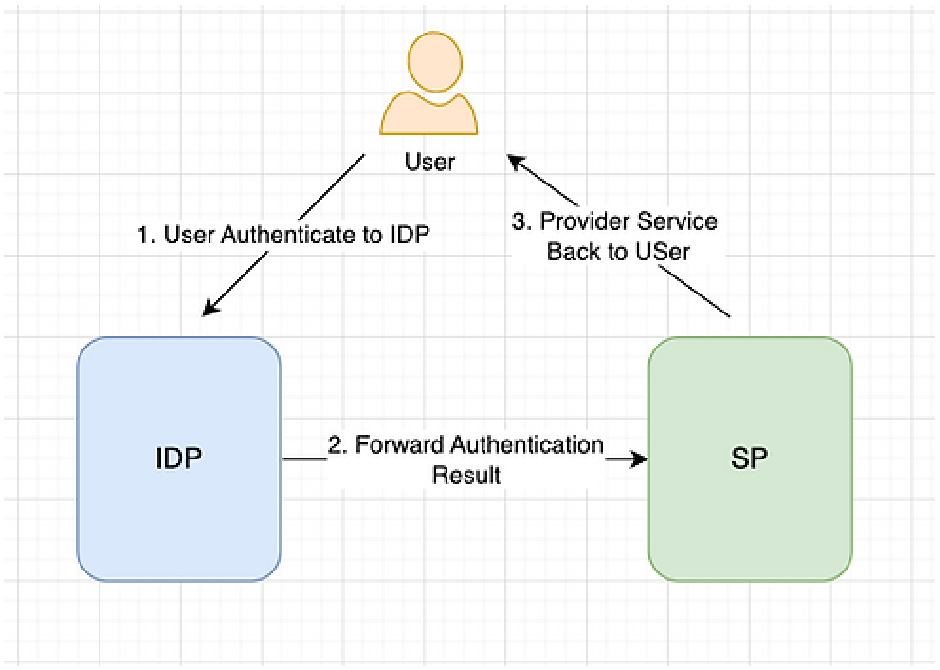
Once we understand what IDP and SP are, we can look at the workflows in SAML. There are two workflows in SAML: SP-Initiated flow and IDP-Initiated flow.

SP-Initiated flow is more frequently seen and the high level process is the same as shown in the above diagram. A user first tries to access some services provided by SP. In order to provide the service, SP needs to authenticate the user to know who he is and what profile information he possesses. Furthermore, to follow the pattern of SSO, SP redirects the user to IDP for authentication. If the user hasn't authenticated in IDP before or the authenticated session has expired, the user will be presented with a login page.

Once a user has finished authentication, IDP will pull the user profile information from the centralized user directory and generate a response to send back to SP. SP then parses and verifies the response to extract user information. Also, since the user is verified, a lot of times, SP will create a user session as well so that next time when the user comes, he doesn't need to be redirected to IDP again. Note this SP user session is specifically for this SP and will store locally on the SP side and it is different from the IDP user session.

IDP-Initiated flow is similar to SP-Initiated flow. The difference is that instead of SP redirecting users to IDP for authentication, the user directly accesses IDP and authenticates at the IDP side. Then IDP forwards the authentication result to SP.

One thing to note is that as IDP is usually integrated with multiple SP, it needs to know which SP to forward the authentication result. This is usually indicated when the user accesses IDP in the first step.


User
1. User Authenticate to IDP
3. Provider Service Back to USer
IDP
2. Forward Authentication Result
SP

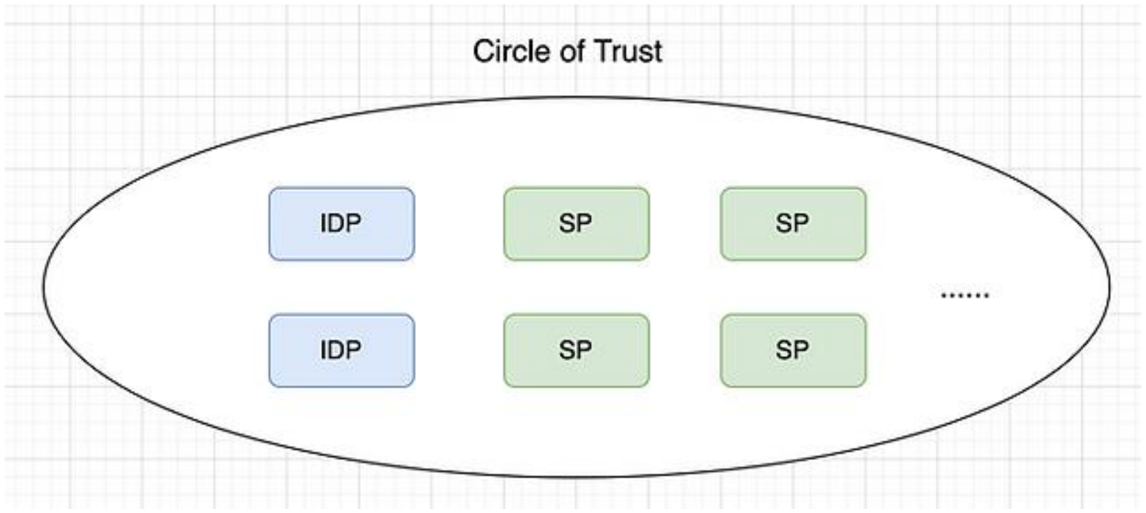
IDP-Initiated Flow

SAML Setup

Now that we have some general understanding of SAML, we can check on how SAML is set up. This setup is needed before the SAML flow can work.

COT (Circle of Trust)

Before SP can delegate authentication to IDP or use the authentication result from IDP, a trusted relationship needs to be established between IDP and SP. This is kind of intuitive, as a SP just can't delegate authentication to some random party or use the authentication result from it. Vice versa, IDP can't just forward authentication results to some random SP to disclose the user information. Some form of trusted-relationship needs to be established.



Circle of Trust

Before we look at how the trust is established, let's take a look at the 'end state'. Eventually at some point, multiple applications (SP) need to establish a trusted relationship with at least one IDP (could be more which is referred to as muti-IDP). As illustrated in the above diagram, all SPs and IDPs are in the same league (circle). This basically means, any SP can delegate authentication to an IDP in the circle and any authentication result sent by IDP is trusted by SP (though more validation is needed to make sure no tampering on the response which we will explore later).

The circle of trust (COT) is the prerequisite for either IDP-Initiated or SP-Initiated SAML flow, and a lot of times, a COT already exists and we just need to add the new SP to the COT.

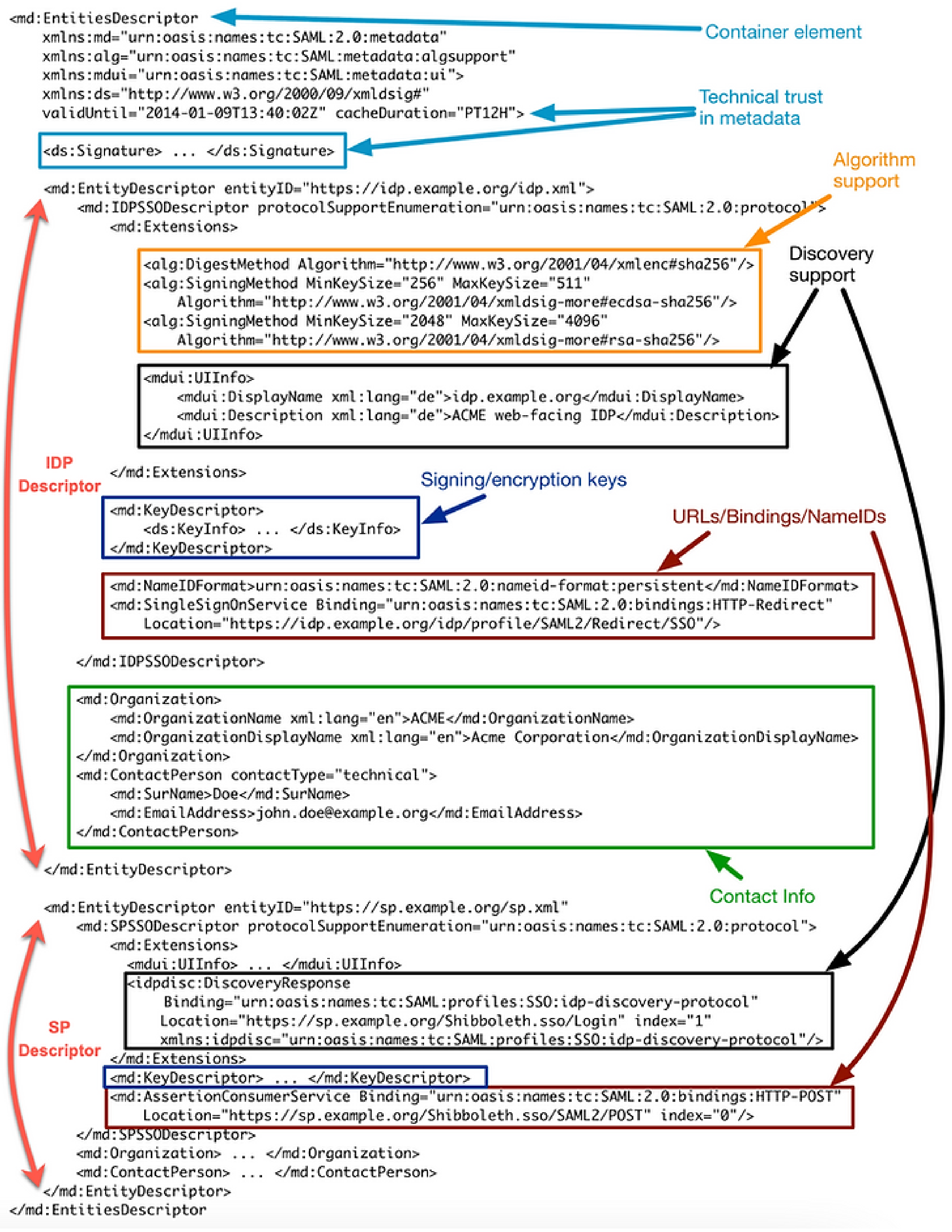
SAML Metadata

Now that we know COT is needed before SAML flow, the question comes as how to establish this handshake. The answer is through a method called Metadata Exchange. Let's first understand what SAML Metadata is.

In theory, SAML metadata is an XML-based configuration data file which includes various information about the SAML entity (IDP or SP) such as identifier, binding endpoints, certificates, keys, etc. Since IDP and SP are serving different purposes in a SAML flow, it's straightforward to tell that IDP and SP will contain different information. On the other hand, they share some similar format and content as well.

Here is an example of IDP and SP metadata combined. We can see the combined version of IDP and SP metadata from time to time. This is because an IDP platform can also serve as a SP. However, SP doesn't usually serve as an IDP. We show a combined version here for conciseness.

Boston Identity



```
<md:EntitiesDescriptor
    xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
    xmlns:alg="urn:oasis:names:tc:SAML:metadata:algsupport"
    xmlns:mdui="urn:oasis:names:tc:SAML:metadata:ui">
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    validUntil="2014-01-09T13:40:02Z" cacheDuration="PT12H">
    <ds:Signature> ... </ds:Signature>
    <md:EntityDescriptor entityID="https://idp.example.org/idp.xml">
        <md:IDPSSODescriptor protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
            <md:Extensions>
                <alg:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                <alg:SigningMethod MinKeySize="256" MaxKeySize="511"
                    Algorithm="http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256"/>
                <alg:SigningMethod MinKeySize="2048" MaxKeySize="4096"
                    Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
                <mdui:UIInfo>
                    <mdui:DisplayName xml:lang="de">idp.example.org</mdui:DisplayName>
                    <mdui:Description xml:lang="de">ACME web-facing IDP</mdui:Description>
                </mdui:UIInfo>
            </md:Extensions>
            <md:KeyDescriptor>
                <ds:KeyInfo> ... </ds:KeyInfo>
            </md:KeyDescriptor>
            <md:NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-format:persistent</md:NameIDFormat>
            <md:SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
                Location="https://idp.example.org/idp/profile/SAML2/Redirect/SSO"/>
        </md:IDPSSODescriptor>
        <md:Organization>
            <md:OrganizationName xml:lang="en">ACME</md:OrganizationName>
            <md:OrganizationDisplayName xml:lang="en">Acme Corporation</md:OrganizationDisplayName>
        </md:Organization>
        <md:ContactPerson contactType="technical">
            <md:SurName>Doe</md:SurName>
            <md:EmailAddress>john.doe@example.org</md:EmailAddress>
        </md:ContactPerson>
    </md:EntityDescriptor>
    <md:EntityDescriptor entityID="https://sp.example.org/sp.xml"
        <md:SPSSODescriptor protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
            <md:Extensions>
              <mdui:UIInfo> ... </mdui:UIInfo>
              <idpdisc:DiscoveryResponse
                  Binding="urn:oasis:names:tc:SAML:profiles:SSO:idp-discovery-protocol"
                  Location="https://sp.example.org/Shibboleth.sso/Login" index="1"
                  xmlns:idpdisc="urn:oasis:names:tc:SAML:profiles:SSO:idp-discovery-protocol"/>
            </md:Extensions>
            <md:KeyDescriptor> ... </md:KeyDescriptor>
            <md:AssertionConsumerService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
                Location="https://sp.example.org/Shibboleth.sso/SAML2/POST" index="0"/>
        </md:SPSSODescriptor>
        <md:Organization> ... </md:Organization>
        <md:ContactPerson> ... </md:ContactPerson>
    </md:EntityDescriptor>
</md:EntitiesDescriptor
```

General Format

The general format of metadata for IDP looks like:

- EntityDescriptor
    - IDPSSODescriptor
        - Extensions (optional, IdP discovery and algorithm support, not much used today)
        - KeyDescriptor (optional, but usually included)
        - ArtifactResolutionService (optional, legacy and not much used today)
        - SingleLogoutService (optional)
        - NameIDFormat (optional)
        - SingleSignOnService (required at least one)
    - Organization (Not Important)
    - ContactPerson (Not Important)

The general format of metadata for SP looks like:

- EntityDescriptor
    - SPSSODescriptor
        - Extensions (optional, service discovery, algorithm support and entity category, not much used today)
        - KeyDescriptor (optional, but usually included)
        - SingleLogoutService (optional)
        - NameIDFormat (optional)
        - AssertionConsumerService (required at least one)
    - Organization (Not Important)
    - ContactPerson (Not Important)

The most important part for both IDP and SP metadata are the entityID and SSO Descriptor. Organization and ContactPerson are more of reference information types which are usually ignored during SAML integration.

Unique Identifier of SAML Entity

Each SAML Entity (SP and IdP) that participates in SAML protocol should have a Unique Identifier. This identifier could be a URI with a maximum 1024 characters. In the above example, the IDP Unique Identifier (entityID) is "*https://idp.example.org/idp.xml*", while the SP Unique Identifier (entityID) is "*https://sp.example.org/sp.xml*". They are both within the <EntityDescriptor>, which is the root element of either IDP metadata or SP metadata.

It might look weird to use a URL as an identifier at first. But seeing that each application usually has a unique home URL and that URL represents the application well, it kind of makes sense to use URL as the entity identifier. Actually, that's what you see most of the time you see in practice.

SSO Descriptor

SSO Descriptor contains Extensions, KeyDescriptor, ArtifactResolutionService (IDP), SingleLogoutService, NameIDFormat, SingleSignOnService(IDP), AssertionConsumerService(SP).

Extensions

The extensions part can contain IDP Discovery, Algorithm Support etc. This is not much used today and is usually omitted.

Key Descriptor

This part of metadata includes information of one or more public keys used in the SAML flow. The SAML request/response can be signed as well as encrypted. In order to verify the signature or decrypt the request/response, a public key is needed. Usually the public key to verify that information is provided inline along with the request/response. However, pre-registered keys as provided in metadata can prevent invalid public keys from being presented in the SAML request/response.

ArtifactResolutionService (ARS)

This is the IDP endpoint for resolving the artifact when Http Artifact Binding is used in the SAML flow. Binding method basically means how IDP sends the authenticated user information back to SP.

There are two ways of binding in SAML: Http Front-Channel (Redirect or Post) and Http Back-Channel (Artifact Binding). The most frequently used is the Http Front-Channel with Post method. We will look more into this later.

SingleLogoutService

This endpoint is for Single Logout Service. When Single Logout Service is implemented, both IDP and SP should present a SingleLogoutService endpoint. When a single logout action is triggered from the IDP SingleLogoutService endpoint, the IDP logs the user off and it also sends multiple requests to all other SP SingleLogoutService endpoints to log the user off on the SP side. Note not all SAML implementations support Single Logout.

NameIDFormat

The NameIDFormat is used to define the NameID value in the IDP response to SP. IDP presents a list of NameIDFormats in its metadata to show what kind of NameIDFormats it supports in the IDP Response. SP presents a list of NameIDFormats to show what kind of NameIDFormats it can request and parse in the IDP response.

A list of examples here:

- **urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified**: Indicates that the format of the NameID is not specified.
- **urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress**: Indicates that the NameID is an email address.
- **urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName**: Indicates that the NameID is an X.509 distinguished name.
- **urn:oasis:names:tc:SAML:1.1:nameid-format:kerberos**: Indicates that the NameID is a Kerberos principal name.
- **urn:oasis:names:tc:SAML:2.0:nameid-format:persistent**: Indicates that the NameID is a persistent identifier that is unique within the scope of the IDP.
- **urn:oasis:names:tc:SAML:2.0:nameid-format:transient**: Indicates that the NameID is a transient identifier that is unique within the scope of the IDP and is not intended to be persisted.

SingleSignOnService

This is IDP's endpoint, which is used as the entry URL to start the SSO service. For example, in the SP-Initiated flow, SP will send requests to this endpoint to start the flow. At least one value is required in the IDP metadata.

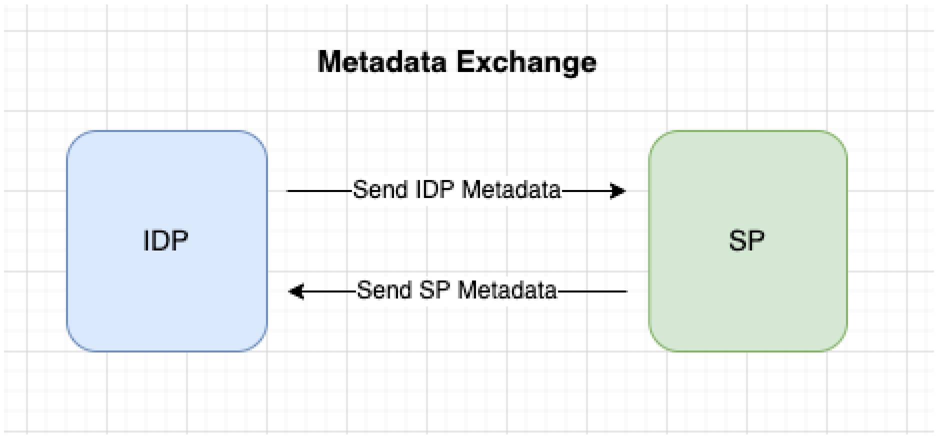
AssertionConsumerService (ACS)

This is SP's endpoint. It is used by IDP when forwarding its SAML response to the SP, where SP will consume the response. At least one value is required from SP metadata.

Exchange of Metadata

As seen from the previous post, IDP metadata and SP metadata are XML-based data containing important information for the SAML entity (IDP and SP).

IDP metadata should be loaded into SP so that IDP can be recognized by SP. Vice versa, SP metadata should be loaded into IDP so that SP can be recognized by IDP.



Metadata Exchange

In reality, there is no specific standard way required to load those information into each party. We can send the metadata to each party as an email attachment, or publish it online and let the other party retrieve it via URL. Once the metadata information is received, it needs to be updated in the IDP/SP system.

A lot of times, an IDP is a well developed IAM platform, an engineer can directly import the SP metadata file into it. On the other hand, SP could be an in-house developed application, and there is usually no straight-forward import function. In this case, a developer will extract the key values from IDP metadata and manually input into SP side.

All in all, the key information from IDP metadata needs to be configured in SP, and the key information from SP metadata needs to be configured in IDP.

How SAML Works

Now that we have established the trust relationship between IDP and SP, we can start to look at how SAML flow really works under the hood.

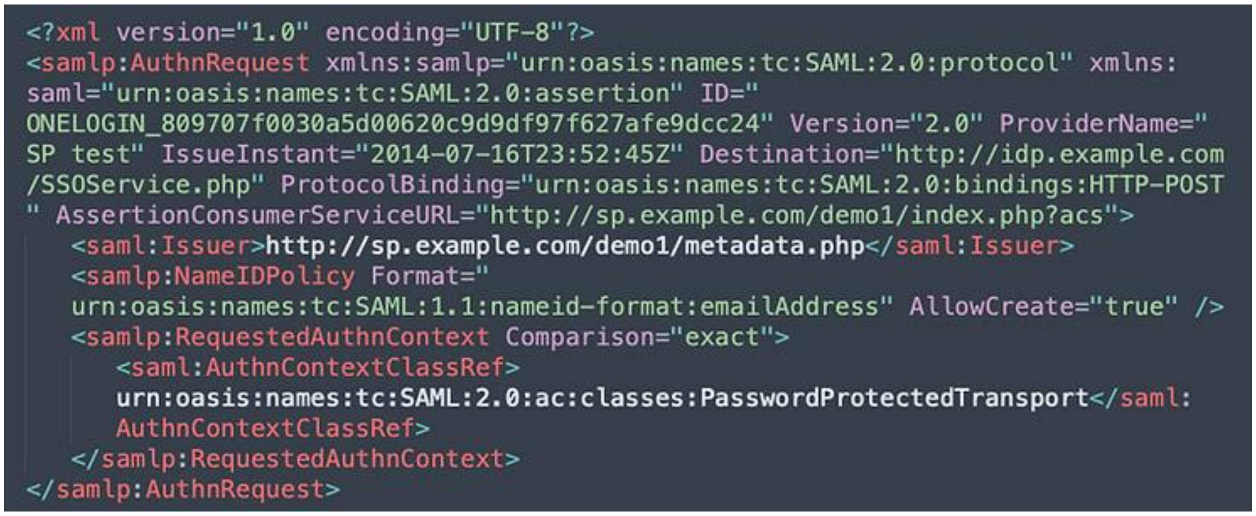
SAML Messages

SAML Messages in SAML are XML-based to convey SAML information between IDP and SP. For example, in the SP-initiated flow, SP will first send a SAML Request message to IDP and later IDP will send a SAML Response message back to SP. During the transport time, the XML-based content will be encoded using Base64.

Here is an example of an encoded sample SAML request:

*PHNhbWxwOkF1dGhuUmVxdWVzdCB4bWxuczpzYW1scD0idXJuOm9hc2lzOm5hbWVzOnRjOl
NBTUw6Mi4wOnByb3RvY29sIiB4bWxuczpzYW1sPSJ1cm46b2FzaXM6bmFtZXM6dGM6U0FN
TDoyLjA6YXNzZXJ0aW9uIiBJRD0iT05FTE9HSU5fODA5NzA3ZjAwMzBhNWQwMDYyMGM5
ZDlkZjk3ZjYyN2FmZTlkY2MyNCIgVmVyc2lvbj0iMi4wIiBQcm92aWRlck5hbWU9IlNQIHRlc3Qi
IElzc3VlSW5zdGFudD0iMjAxNC0wNy0xNlQyMzo1Mjo0NVoiIERlc3RpbmF0aW9uPSJodHRw
Oi8vaWRwLmV4YW1wbGUuY29tL1NTT1NlcnZpY2UucGhwIiBQcm90b2NvbEJpbmRpbmc9InV
ybjpvYXNpczpuYW1lczp0YzpTQU1MOjIuMDpiaW5kaW5nczpIVFRQLVBPU1QiIEFzc2VydGlv
bkNvbnN1bWVyU2VydmljZVVSTD0iaHR0cDovL3NwLmV4YW1wbGUuY29tL2RlbW8xL2luZGV
4LnBocD9hY3MiPg0KICA8c2FtbDpJc3N1ZXI+aHR0cDovL3NwLmV4YW1wbGUuY29tL2Rlb
W8xL21ldGFkYXRhLnBocDwvc2FtbDpJc3N1ZXI+DQogIDxzYW1scDpOYW1lSURQb2xpY3kg
Rm9ybWF0PSJ1cm46b2FzaXM6bmFtZXM6dGM6U0FNTDoxLjE6bmFtZWlkLWZvcm1hdDplb
WFpbEFkZHJlc3MiIEFsbG93Q3JlYXRlPSJ0cnVlIi8+DQogIDxzYW1scDpSZXF1ZXN0ZWRBd
XRobkNvbnRleHQgQ29tcGFyaXNvbj0iZXhhY3QiPg0KICAgIDxzYW1sOkF1dGhuQ29udGV4dE
NsYXNzUmVmPnVybjpvYXNpczpuYW1lczp0YzpTQU1MOjIuMDphYzpjbGFzc2VzOlBhc3N3b3J
kUHJvdGVjdGVkVHJhbnNwb3J0PC9zYW1sOkF1dGhuQ29udGV4dENsYXNzUmVmPg0KICA8
L3NhbWxwOlJlcXVlc3RlZEF1dGhuQ29udGV4dD4NCjwvc2FtbHA6QXV0aG5SZXF1ZXN0Pg=
=*

The decoded XML content is



```xml
<?xml version="1.0" encoding="UTF-8"?>
<samlp:AuthnRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol" xmlns:
saml="urn:oasis:names:tc:SAML:2.0:assertion" ID="
ONELOGIN_809707f0030a5d00620c9d9df97f627afe9dcc24" Version="2.0" ProviderName="
SP test" IssueInstant="2014-07-16T23:52:45Z" Destination="http://idp.example.com
/SSOService.php" ProtocolBinding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST
" AssertionConsumerServiceURL="http://sp.example.com/demo1/index.php?acs">
   <saml:Issuer>http://sp.example.com/demo1/metadata.php</saml:Issuer>
   <samlp:NameIDPolicy Format="
   urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress" AllowCreate="true" />
   <samlp:RequestedAuthnContext Comparison="exact">
      <saml:AuthnContextClassRef>
      urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport</saml:
      AuthnContextClassRef>
   </samlp:RequestedAuthnContext>
</samlp:AuthnRequest>
```

SAML Bindings

In SAML, bindings are methods used to transport SAML messages between IDP and SP. There are several bindings defined in the SAML specification:

- HTTP POST Binding
- HTTP Redirect Binding

- HTTP Artifact Binding
- SOAP Binding

HTTP POST Binding

This is the most frequently used binding method. The SAML messages are transported via HTTP POST requests, where the message content is in the payload of an HTML form.

HTTP Redirect Binding

This method is sometimes used as well. The SAML messages are transported via HTTP GET requests. The message is transported as a URL query string parameter called "SAMLRequest".

HTTP Artifact Binding

This binding is less used today. It doesn't send the full SAML message in the HTTP call, but instead, using an intermediary token as an artifact. And then, once SP receives the artifact, it calls the IDP's ArtifactResolutionService (ARS) endpoint to exchange the artifact for the full content.

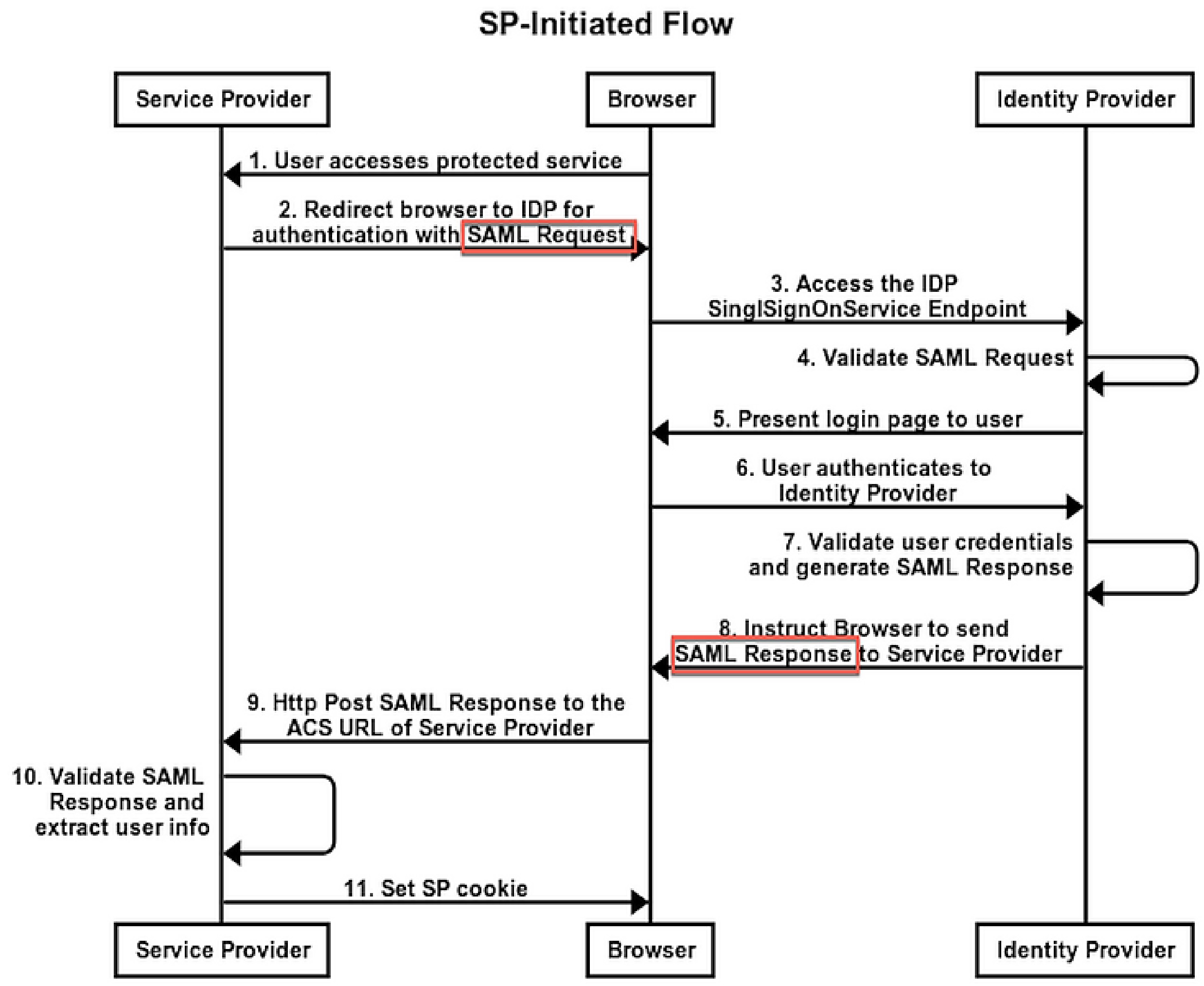
The artifact token is an opaque identifier (meaning some random string to represent an 'ID') transported in the HTTP response from IDP to SP. Then the exchange of artifacts for full response content happens on a backend socket channel, which is NOT HTTP.

SOAP Binding

This binding, as its name, uses SOAP (Simple Object Access Protocol) messaging framework to transport SAML messages. This is less used today as well.

SP-Initiated Flow with HTTP POST Binding

Since HTTP POST Binding is mostly used nowadays, let's take a look at the SP-Initiated flow again, but this time with more details.


SP-Initiated Flow
Service Provider
Browser
Identity Provider
1. User accesses protected service
2. Redirect browser to IDP for authentication with SAML Request
3. Access the IDP SinglSignOnService Endpoint
4. Validate SAML Request
5. Present login page to user
6. User authenticates to Identity Provider
7. Validate user credentials and generate SAML Response
8. Instruct Browser to send SAML Response to Service Provider
9. Http Post SAML Response to the ACS URL of Service Provider
10. Validate SAML Response and extract user info
11. Set SP cookie
Service Provider
Browser
Identity Provider

SP-Initiated Flow with HTTP POST Binding

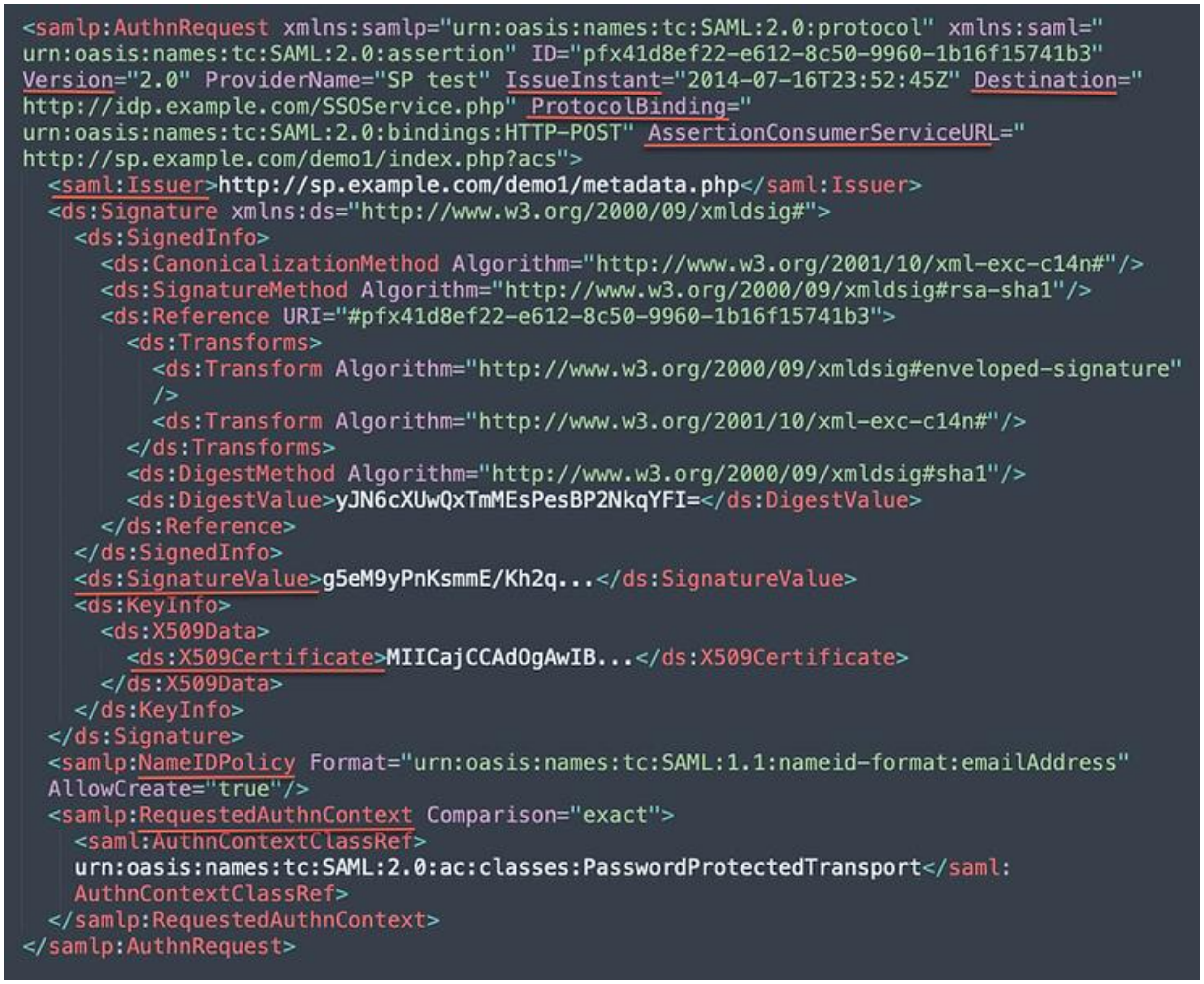
When a user accesses the service from SP, SP detects no active user session exists, so it redirects user to IDP for authentication. Note here a SAML Request is generated by SP and forwarded to IDP's SingleSignOnService Endpoint via HTTP POST.

SingleSignOnService Endpoint is the target at IDP to invoke the SAML flow and HTTP POST is a binding method, i.e. how SAML requests should be sent from SP to IDP. Although there are other binding methods, e.g. HTTP Redirect and Artifact Binding, it is HTTP POST that is mostly used. Then let's understand this SAML Request with more details.

SAML Request

A SAML Request (or AuthnRequest) is a message from SP to IDP, requesting IDP to issue a SAML Response describing the properties of a user after the user authenticates to the IDP. This happens in the SP-Initiated flow.

Here is an example SAML Request in the XML format:



```xml
<samlp:AuthnRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol" xmlns:saml="
urn:oasis:names:tc:SAML:2.0:assertion" ID="pfx41d8ef22-e612-8c50-9960-1b16f15741b3"
Version="2.0" ProviderName="SP test" IssueInstant="2014-07-16T23:52:45Z" Destination="
http://idp.example.com/SSOService.php" ProtocolBinding="
urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST" AssertionConsumerServiceURL="
http://sp.example.com/demo1/index.php?acs">
  <saml:Issuer>http://sp.example.com/demo1/metadata.php</saml:Issuer>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
      <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <ds:Reference URI="#pfx41d8ef22-e612-8c50-9960-1b16f15741b3">
        <ds:Transforms>
          <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"
          />
          <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        </ds:Transforms>
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <ds:DigestValue>yJN6cXUwQxTmMEsPesBP2NkqYFI=</ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>g5eM9yPnKsmmE/Kh2q...</ds:SignatureValue>
    <ds:KeyInfo>
      <ds:X509Data>
        <ds:X509Certificate>MIICajCCAdOgAwIB...</ds:X509Certificate>
      </ds:X509Data>
    </ds:KeyInfo>
  </ds:Signature>
  <samlp:NameIDPolicy Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress"
  AllowCreate="true"/>
  <samlp:RequestedAuthnContext Comparison="exact">
    <saml:AuthnContextClassRef>
    urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport</saml:
    AuthnContextClassRef>
  </samlp:RequestedAuthnContext>
</samlp:AuthnRequest>
```

Some key elements are highlighted.

- **Version** - Indicates it is SAML 2.0
- **IssueInstant** - The date and time the SAML request was issued. This field is used to ensure that the message is not replayed or reused after a certain period of time
- **Destination** - The entityID of the IDP that this message is supposed to be sent to
- **Protocol Binding** - The binding method used to transmit SAML messages

- **AssertionConsumerServiceURL** - The ACS URL at SP where IDP will return the SAML response to
- **Issuer** - The entityID of SP
- **SignatureValue** - The signature value (redacted in the example) of the SAML request
- **X509Certificate** - The public cert (redacted in the example) that can be used to verify the signature of this request
- **NameIDPolicy** - The NameIDFormat that IDP should use in the SAML Response
- **RequestedAuthnContext** - The level of authentication that is required by the SP. This may include information such as authentication method, strength of authentication and any additional context needed

Not all elements listed above are required and from time to time, some of the elements are omitted. For example, if AssertionConsumerServiceURL is not specified in the request, IDP usually has a default one that's preconfigured for this SP during metadata exchange, and when sending the SAML Response back, it will use the default AssertionConsumerServiceURL.

The signature information is not required in the SAML Request, but is always good to have. In this case, IDP can always verify the signature using the Public Cert provided (how to verify signature is outside the scope of this post) to make sure the request is not tampered.

The NameIDFormat is important too, as it depicts how the subject value (the value that identifies the user authenticated to IDP) returned in the SAML response should be interpreted by SP. For example, it's shown as 'emailAddress' in the example, so in the SAML response, the subject value should be interpreted as an email value.

RequestedAuthnContext provides a means for SP to ask IDP to authenticate the user with a specific authentication mechanism. For example, if SP specifies PasswordProtectedTransport in the SAML Request, the IDP knows it has to authenticate the user through login/password, protected by SSL/TLS. In turn, IDP will also show in the SAML response which mechanism it used to authenticate the user through AuthnContextClassRef. AuthnContextClassRef is required in the SAML Response.

SAML Response and Assertion

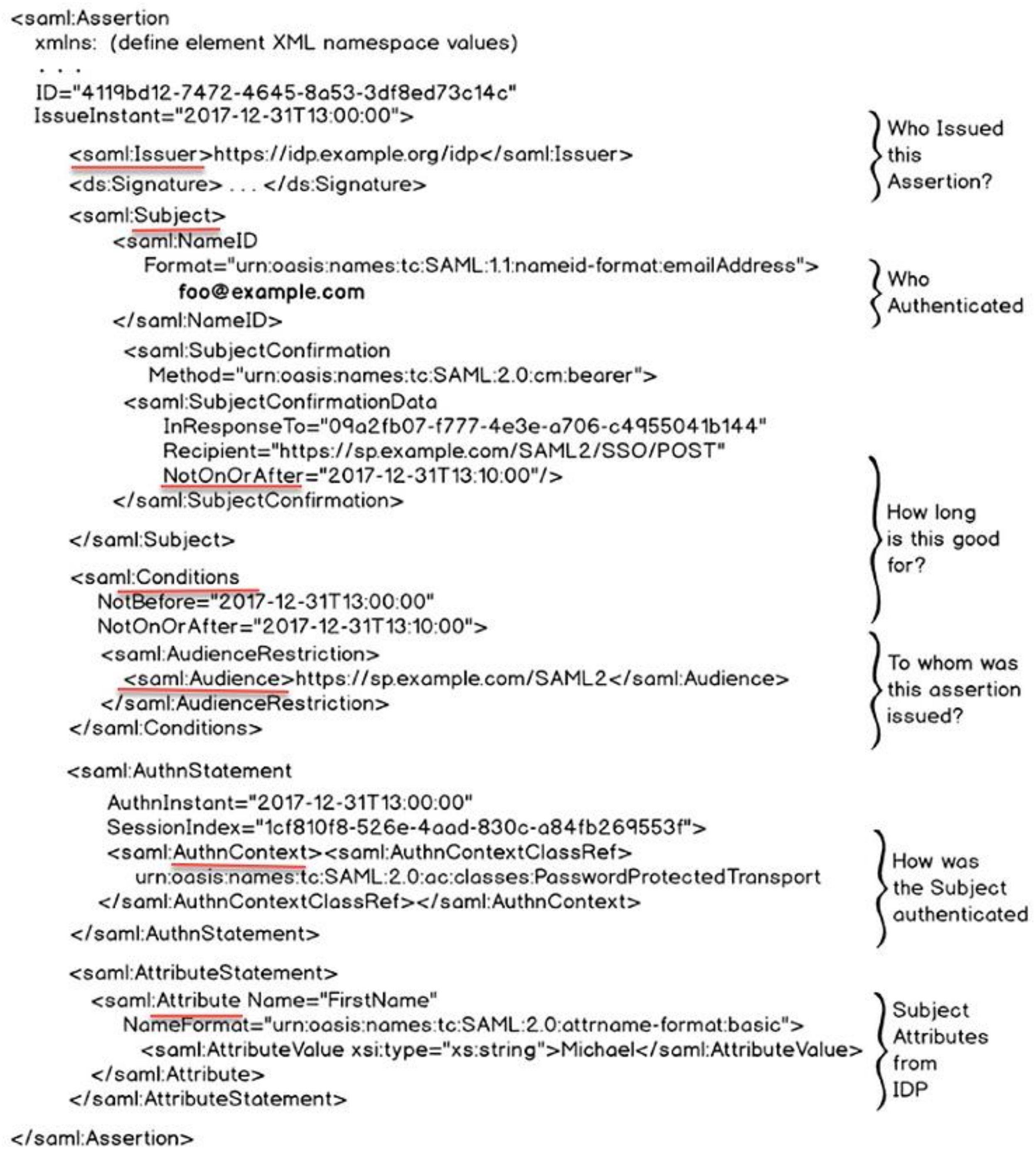
Once the browser sends the SAML Request to IDP, it will ask the user to login to IDP (if no active user session is presented). Once a user is authenticated to IDP, it will pull the user profile from the User Directory and generate a SAML Response, which will be sent back to SP's ACS endpoint.

The format of SAML Response is similar to SAML Request. However, the major component in SAML Response is the SAML Assertion.

SAML Assertion

A SAML Assertion is a part of the SAML where all necessary user profile information e.g. email, first name, last name, role etc, as well as the authentication context/status are present.

Here is an Assertion example:



```
<saml:Assertion
   xmlns:  (define element XML namespace values)
   . . .
   ID="4119bd12-7472-4645-8a53-3df8ed73c14c"
   IssueInstant="2017-12-31T13:00:00">

      <saml:Issuer>https://idp.example.org/idp</saml:Issuer>
      <ds:Signature> . . . </ds:Signature>
      <saml:Subject>
          <saml:NameID
              Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">
                  foo@example.com
          </saml:NameID>
           <saml:SubjectConfirmation
             Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
           <saml:SubjectConfirmationData
               InResponseTo="09a2fb07-f777-4e3e-a706-c4955041b144"
               Recipient="https://sp.example.com/SAML2/SSO/POST"
               NotOnOrAfter="2017-12-31T13:10:00"/>
          </saml:SubjectConfirmation>

      </saml:Subject>

      <saml:Conditions
         NotBefore="2017-12-31T13:00:00"
         NotOnOrAfter="2017-12-31T13:10:00">
          <saml:AudienceRestriction>
            <saml:Audience>https://sp.example.com/SAML2</saml:Audience>
          </saml:AudienceRestriction>
      </saml:Conditions>

      <saml:AuthnStatement
          AuthnInstant="2017-12-31T13:00:00"
          SessionIndex="1cf810f8-526e-4aad-830c-a84fb269553f">
          <saml:AuthnContext><saml:AuthnContextClassRef>
             urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
         </saml:AuthnContextClassRef></saml:AuthnContext>
      </saml:AuthnStatement>

      <saml:AttributeStatement>
        <saml:Attribute Name="FirstName"
           NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
             <saml:AttributeValue xsi:type="xs:string">Michael</saml:AttributeValue>
        </saml:Attribute>
      </saml:AttributeStatement>

</saml:Assertion>
```

Who Issued this Assertion?
Who Authenticated
How long is this good for?
To whom was this assertion issued?
How was the Subject authenticated
Subject Attributes from IDP

Some key elements are highlighted:

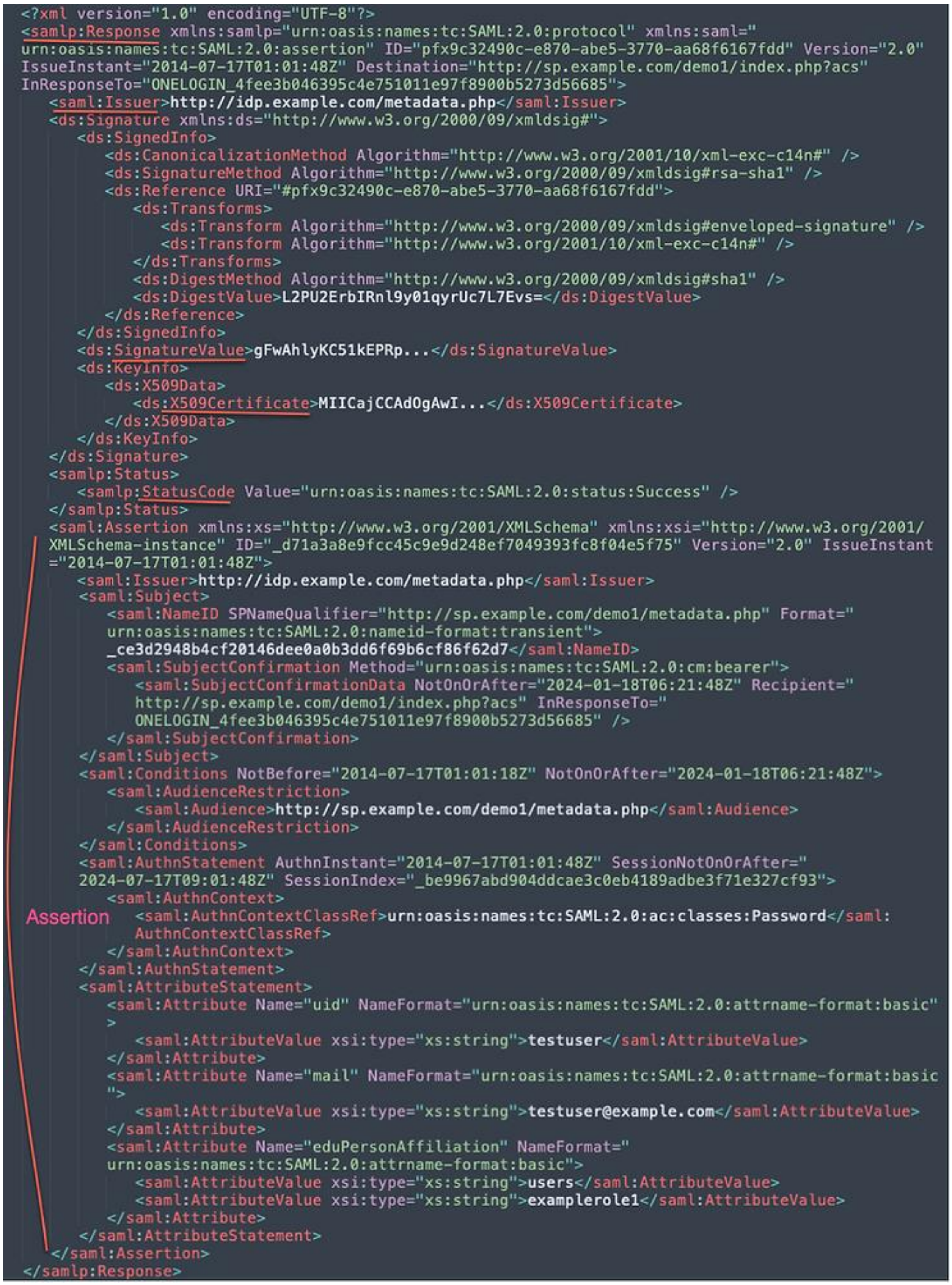
- **Issuer** - The entityID of IDP
- **Subject** - The user who has been authenticated, presented in a specified NameIDFormat

- **NotOnOrAfter** - Indicate how long this assertion will be valid for
- **Conditions** - Conditions that must be met for the assertion to be considered valid
- **Audience** - The entityID of the target SP who this message is intended for
- **AuthnContext** - The level of authentication context that happened with this user authentication
- **Attribute** - List of additional user attributes such as first name, last name, role etc.

As you can see, with the authentication contexts and identity profiles presented in the assertion, SP who receives that information has what is necessary to confirm a user identity. Yet, to properly send the SAML Assertion back to SP, it needs a wrapper following the SAML protocol.

SAML Response

An example SAML Response looks like below. Assertion is still the main part, but there are some other added entries to make it a valid SAML Response.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<samlp:Response xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol" xmlns:saml="
urn:oasis:names:tc:SAML:2.0:assertion" ID="pfx9c32490c-e870-abe5-3770-aa68f6167fdd" Version="2.0"
IssueInstant="2014-07-17T01:01:48Z" Destination="http://sp.example.com/demo1/index.php?acs"
InResponseTo="ONELOGIN_4fee3b046395c4e751011e97f8900b5273d56685">
   <saml:Issuer>http://idp.example.com/metadata.php</saml:Issuer>
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <ds:SignedInfo>
         <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
         <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
         <ds:Reference URI="#pfx9c32490c-e870-abe5-3770-aa68f6167fdd">
            <ds:Transforms>
               <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
               <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
            <ds:DigestValue>L2PU2ErbIRnl9y01qyrUc7L7Evs=</ds:DigestValue>
         </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>gFwAhlyKC51kEPRp...</ds:SignatureValue>
      <ds:KeyInfo>
         <ds:X509Data>
            <ds:X509Certificate>MIICajCCAdOgAwI...</ds:X509Certificate>
         </ds:X509Data>
      </ds:KeyInfo>
   </ds:Signature>
   <samlp:Status>
      <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success" />
   </samlp:Status>
   <saml:Assertion xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
   XMLSchema-instance" ID="_d71a3a8e9fcc45c9e9d248ef7049393fc8f04e5f75" Version="2.0" IssueInstant
   ="2014-07-17T01:01:48Z">
      <saml:Issuer>http://idp.example.com/metadata.php</saml:Issuer>
      <saml:Subject>
         <saml:NameID SPNameQualifier="http://sp.example.com/demo1/metadata.php" Format="
         urn:oasis:names:tc:SAML:2.0:nameid-format:transient">
         _ce3d2948b4cf20146dee0a0b3dd6f69b6cf86f62d7</saml:NameID>
         <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
            <saml:SubjectConfirmationData NotOnOrAfter="2024-01-18T06:21:48Z" Recipient="
            http://sp.example.com/demo1/index.php?acs" InResponseTo="
            ONELOGIN_4fee3b046395c4e751011e97f8900b5273d56685" />
         </saml:SubjectConfirmation>
      </saml:Subject>
      <saml:Conditions NotBefore="2014-07-17T01:01:18Z" NotOnOrAfter="2024-01-18T06:21:48Z">
         <saml:AudienceRestriction>
            <saml:Audience>http://sp.example.com/demo1/metadata.php</saml:Audience>
         </saml:AudienceRestriction>
      </saml:Conditions>
      <saml:AuthnStatement AuthnInstant="2014-07-17T01:01:48Z" SessionNotOnOrAfter="
      2024-07-17T09:01:48Z" SessionIndex="_be9967abd904ddcae3c0eb4189adbe3f71e327cf93">
         <saml:AuthnContext>
            <saml:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:Password</saml:
            AuthnContextClassRef>
         </saml:AuthnContext>
      </saml:AuthnStatement>
      <saml:AttributeStatement>
         <saml:Attribute Name="uid" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic"
         >
            <saml:AttributeValue xsi:type="xs:string">testuser</saml:AttributeValue>
         </saml:Attribute>
         <saml:Attribute Name="mail" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic
         ">
            <saml:AttributeValue xsi:type="xs:string">testuser@example.com</saml:AttributeValue>
         </saml:Attribute>
         <saml:Attribute Name="eduPersonAffiliation" NameFormat="
         urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
            <saml:AttributeValue xsi:type="xs:string">users</saml:AttributeValue>
            <saml:AttributeValue xsi:type="xs:string">examplerole1</saml:AttributeValue>
         </saml:Attribute>
      </saml:AttributeStatement>
   </saml:Assertion>
</samlp:Response>
```

Highlighted elements:

- **Response** - Indicate this is a SAML response instead of SAML request
- **Issuer** - The entityID of IDP
- **SignatureValue** - The signature value for the entire SAML response message
- **X509Certificate** - The public certificate that is used to verify the signature
- **StatusCode** - The status of the SAML flow, success, requestDenied and etc.

Once the SP receives the SAML response and verifies the signature to make sure of the data integrity, SP can extract identity information from the Assertion and create a user session and set browser cookie.

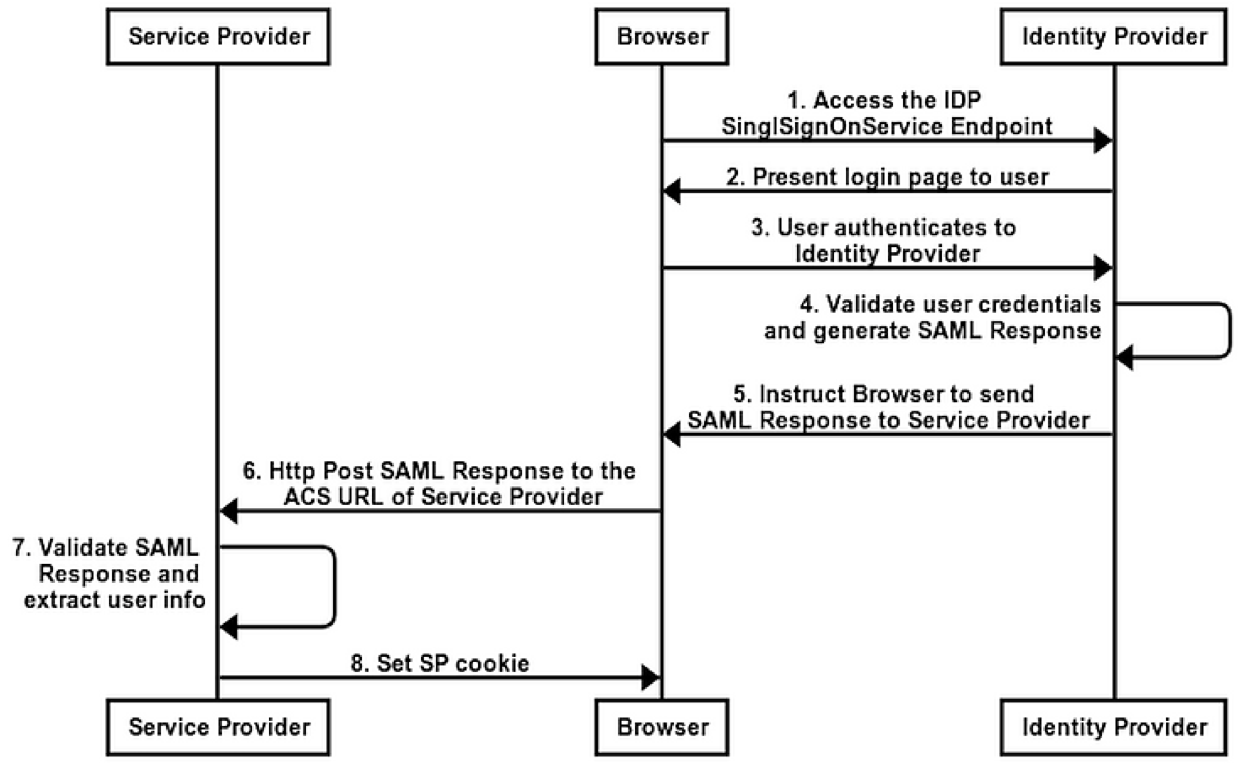
IDP-Initiated Flow with HTTP POST Binding

The IDP-Initiated Flow is similar to the SP-Initiated Flow, with less steps.

IDP-Initiated flow is used from time to time. This could be based on various reasons. For example, IDP just finishes some other function and would like to trigger the SAML flow, or SP doesn't support SP-Initiated flow well.

Regarding the flow, instead of accessing SP, the browser directly accesses IDP and a login page is present. Once a user finishes authentication at IDP, the remaining steps are the same as in SP-Initiated flow.

SP-Initiated Flow


Service Provider
Browser
Identity Provider
1. Access the IDP SinglSignOnService Endpoint
2. Present login page to user
3. User authenticates to Identity Provider
4. Validate user credentials and generate SAML Response
5. Instruct Browser to send SAML Response to Service Provider
6. Http Post SAML Response to the ACS URL of Service Provider
7. Validate SAML Response and extract user info
8. Set SP cookie
Service Provider
Browser
Identity Provider

One thing to note is that IDP needs to know which SP to forward the SAML Response to if there are multiple SP. This could be done by passing a URL parameter to IDP when triggering the IDP-initiated flow or other means that IDP supports.

Chapter 12 – SAML: A Further Look

In this chapter, we continue checking out SAML and concepts.

Key Points

- **SAML Core Components**

    - **Assertions:** Security statements about a subject (e.g., user), issued by an entity like an IDP. These assertions convey authentication, attribute, and authorization information.
    - **Protocols:** Define the request/response interactions between SAML entities (e.g., Authentication Request Protocol, Single Logout Protocol).
    - **Bindings:** Specify how SAML messages are encoded and transmitted over communication channels (e.g., HTTP POST, HTTP Redirect).
    - **Profiles:** Combine assertions, protocols, and bindings to support specific use cases (e.g., Web Browser SSO Profile).

- **Important Profiles**

    - **Web Browser SSO Profile:** The most common profile, enabling SSO within web browsers.
    - **Single Logout Profile:** Facilitates logging out of multiple applications through a single action.
    - **Artifact Resolution Profile:** (Rarely used) Allows retrieval of SAML messages using artifacts.

- **Common Bindings**

    - **HTTP POST Binding:** The most widely used binding, transmitting SAML messages within HTTP POST requests.
    - **HTTP Redirect Binding:** Sends SAML messages as URL parameters in HTTP GET requests.
    - **HTTP Artifact Binding:** (Rarely used) Employs an artifact (an opaque identifier) to represent the SAML message, which is retrieved separately.
    - **SOAP Binding:** (Legacy) Uses the SOAP protocol for message exchange.

- **Key Protocols**

    - **Authentication Request Protocol:** Used by SP to request user authentication from IDP.
    - **Authentication Response Protocol:** Used by IDP to respond to authentication requests with assertions.

- **Single Logout Protocol:** Enables coordinated logout across multiple applications.
- **Metadata Exchange Protocol:** Facilitates the exchange of metadata between SAML entities.
- **NameID Mapping Protocol:** Defines how user identifiers are mapped between different SAML entities.

- **Assertions**

  - Contain statements about a user's identity and security attributes.
  - Three types of statements:
    - Authentication Statements: Describe how the user was authenticated.
    - Attribute Statements: Provide specific attributes about the user.
    - Authorization Decision Statements: Specify what actions the user is authorized to perform.

- **Privacy and Security**

  - **Privacy Mechanisms:**
    - Identity Pseudonyms: Allow the use of pseudonyms to protect user privacy.
    - Transient Identifiers: Generate unique identifiers for each session to prevent user tracking.
    - Assurance Levels: Define the level of confidence in user authentication.
  - **Security Mechanisms:**
    - TLS: Use TLS for all HTTP communication to ensure confidentiality and integrity.
    - Signatures: Mandate signing of SAML responses (and optionally assertions) to verify message integrity.

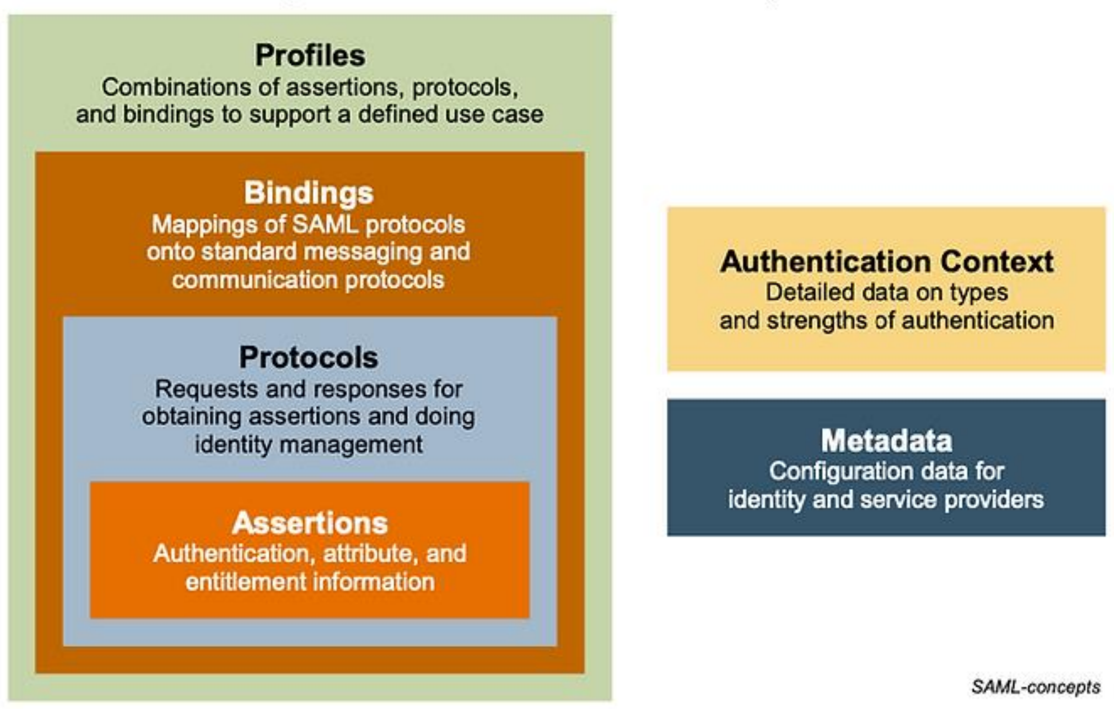
- **Best Practices**

  - **Assertion vs. Response Signing:** Sign the entire SAML response for stronger security, even though signing only the assertion is permissible in some cases.
  - **Encryption:** Encrypt the SAML response to protect sensitive user data.
  - **SAML Request Signature:** Include signatures in SAML requests whenever possible to ensure request integrity.
  - **Entity IDs and SAML IDs:** Include entity IDs and unique SAML IDs in requests and responses to prevent message mix-ups and replay attacks.

SAML Concept

We can basically divide SAML into four different parts or layers.

- **Assertions** - SAML Assertions are security information asserted by an entity party (e.g. IDP) in the form of statements about a subject
- **Protocols** - SAML defines some request/response protocols
- **Bindings** - SAML Bindings detail exactly how SAML messages can be sent/received underlying transport protocols
- **Profiles** - SAML Profiles define how SAML assertions, protocols, and bindings are combined to provide interoperability in different usage scenarios


Profiles
Combinations of assertions, protocols, and bindings to support a defined use case
Bindings
Mappings of SAML protocols onto standard messaging and communication protocols
Protocols
Requests and responses for obtaining assertions and doing identity management
Assertions
Authentication, attribute, and entitlement information
Authentication Context
Detailed data on types and strengths of authentication
Metadata
Configuration data for identity and service providers
SAML-concepts

SAML Concepts

Profiles

SAML defines several Profiles and some of them are not actively being used today. The relevant ones are:

- **Web Browser SSO Profile** - This is the most used Profile when SAML is present. It defines how SAML entities use the Authentication Request Protocol and SAML Response messages and assertions to achieve Single Sign-On with Browsers. It defines how messages are used in combination with HTTP Redirect, HTTP POST, and HTTP Artifact Bindings
- **Single Logout Profile** - This is sometimes used. It defines how SAML Single Logout Protocol can be used with Browser SSO Profile and SOAP (legacy)

- **Artifact Resolution Profile** - Rarely used. It defines how SAML entities can use the Artifact Resolution Protocol to retrieve the SAML message (e.g. IDP response) referred by an artifact

Bindings

SAML Bindings define how SAML messages are encoded, packaged, and transmitted over different communication channels. Some of the bindings are deprecated.

- **HTTP POST Binding** - Most used. Transmit SAML messages through HTTP POST requests. SAML messages are encoded as XML documents and sent in the body of the POST request.
- **HTTP Redirect Binding** - Transmit SAML messages through HTTP GET requests. SAML messages are encoded as URL parameters and sent in the query string of the redirect request.
- **HTTP Artifact Binding** - Rarely used. Transmit SAML messages using a combination of HTTP and a separate messaging protocol. SAML messages are encoded as a small identifier called an artifact that is transmitted over HTTP, and the actual SAML message is retrieved by the recipient using a separate messaging protocol.
- **SOAP Binding** - Legacy binding, not used today. This binding is used to transmit SAML messages over SOAP (Simple Object Access Protocol) using the SOAP message format.

Protocols

SAML defines some request/response protocols for operation.

- **Authentication Request Protocol** - Used by a service provider (SP) to request authentication of a user from an identity provider (IdP)
- **Authentication Response Protocol** - Used by an IdP to respond to a SAML authentication request with an assertion containing information about the user's identity
- **Single Logout Protocol** - Allow logout of active sessions associated with a principal. The logout can be directly initiated by the user, or initiated by an IDP or SP on session timeout, administrator action, etc.
- **Metadata Exchange Protocol** - Used to exchange metadata between SAML entities, including information about endpoints, certificates, and supported bindings
- **NameID Mapping Protocol** - It defines how to map the identifier used by one SAML entity to the identifier used by another entity

Assertions

An assertion contains information about a user's identity and the security attributes associated with that identity, which is generated by an IDP in response to a SAML authentication request from a SP. SAML Assertions include several elements.

SAML defines three kinds of statements that it can carry:

- **Authentication Statements** - Describe the means how the user is authenticated, e.g. authentication time
- **Attribute Statements** - Specific attributes associated to a subject (user)
- **Authorization Decision Statements** - Describe what the subject is entitled to do

Privacy and Security in SAML

With various regulations going-on, user privacy has become a determining factor for some organizations. SAML also supports a number of mechanisms that support deployment in privacy.

- **Identity Pseudonym** - SAML supports pseudonyms for an identity between IDP and SP. This helps protect the user's privacy by preventing the service provider from collecting or storing their personal information
- **One-time Transient Identifier** - A transient identifier (random opaque string) is generated as the subject identifier each time IDP sends response back to the SP, so that SP will NOT be able to recognize them as the same individual as might have previously visited
- **Assurance Level** - Authentication Context mechanism allows a user to be authenticated at a sufficient assurance level for the resource requested

Just providing assertions may not be enough to ensure a secure system and some security mechanism is needed as well.

- Use TLS for HTTP all the time
- The IDP response is mandatory to be signed and signature verified by SP

Some Best Practices

Assertion Signing vs Response Signing

The signature for SAML Response can be at either Assertion level or the entire Message level or both.

OASIS's SAML Specification points out that TLS should be used at all times to maintain confidentiality and message integrity. The Assertion MUST be signed when HTTP POST binding is used, and MAY be signed if the HTTP-Artifact binding is used.

The reason that not the entire SAML Response message needs to be signed is because it is relying on TLS to maintain data integrity and prevent man-in-the-middle attack. When HTTP-Artifact is used, it is even safer as it is a back channel connection.

However, I would recommend always sign the entire SAML response message and optionally sign the Assertion part. The reasons are:

1. Assertion is included in the Response Message and signing the entire response message will guarantee the integrity of Assertion as well.
2. The OASIS standard was published in 2006 and Information Technology has evolved a lot since then. Signing Assertion plus HTTP with TLS could be good in the old days, but may not be sufficient in current time.
3. Signing the entire message wouldn't put much overhead on the SAML flow

By signing the entire SAML Response message, it is guaranteed that the SAML message is not tampered.

Encryption

SAML supports encryption for the messages as well. Since SAML Requests usually don't have sensitive information, it's rare to encrypt SAML requests.

On the other hand, SAML Response could include some sensitive private information. In this case, it's better to encrypt the SAML response message so that even if the response message is intercepted, the information is not disclosed.

Note that encryption should be combined with other means like TLS for HTTP, Message Signature, and etc.

SAML Request Signature

In general, it is good practice to include signatures in the SAML Request, so that the integrity of the request can be verified.

Sometimes, SP might have limitations generating signed SAML requests. If no signature is present in SAML Request, in this case, the ACS URL presented in the SAML Request must have been pre-registered in the IDP. If the signature is present, depending on the IDP, the ACS URL in the request could be a new URL that is not pre-registered. This is helpful in some scenarios where ACS URL is dynamic or difficult to predict.

Include SAML ID in the Request and Response

It's common to include SP entityID and IDP entityID in the request and response. This will let the receiver verify the origin and target audience of the message and reject it if there is any mismatch.

Another important attribute is the ID which is a string uniquely identifying the SAML request. For example, ID="*809707f0030a5d00620c9d9df97f627afe9dcc24*" as below.

```
<samlp:AuthnRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"

xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"

ID="809707f0030a5d00620c9d9df97f627afe9dcc24" Version="2.0" ProviderName="SP test"

IssueInstant="2014-07-16T23:52:45Z" Destination="http://idp.example.com/SSOService.php"

ProtocolBinding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"

AssertionConsumerServiceURL="http://sp.example.com/demo1/index.php?acs">
```

This ID should be included in the SAML response as well and the response should be rejected if there is any mismatch.

Appendix I - Being an IAM Consultant

What is an IAM Consultant?

In a general sense, a consultant is a person who provides professional advice in a particular field of science or business to an organization or individual. In the context of IAM, a consultant utilizes its identity skills and experiences to advise or implement IAM solutions to enterprise customers.

Why Do Companies Need Consultant?

This question comes to why companies need consulting in the first place, and usually a defining reason is that consultants possess a knowledge advantage. A client hires a consulting firm that provides advice and implementations that leads to the resolution of an issue within the client's organization.

Other reasons could be:

- Consultants are objective on the issue
- Consultants are less likely to fall to internal politics
- Consultants are contractors and cost could be less than hiring permanent employees in the long run
- Consultants are more effective in terms of execution

Why IAM Consultant?

Bringing this back to the field of IAM, a mid-to-large enterprise has the issue of identity and access management needs to resolve and usually in-house development is very costly (time and money). A decent size of engineering team is needed and it takes a long time to do the development work. After production goes live, the IAM system will need maintenance as well. On the other hand, there are mature IAM vendors out there in the market and adopting them will save hugely for the enterprise.

So a good strategy is for companies to buy and use IAM products directly, but where do IAM consultants come into play? The main reason is that each system of a client is unique and could be very different from another, you just can't buy the product and plug-and-play right away. This process of implementation and integration needs to happen between the IAM vendor product and client company. That's where IAM consultants come into play.

Consulting Service Process

The IAM consulting service helps client companies to solve problems related to IAM. As each customer is unique, the first critical step is to understand the current state of the customer. Sometimes, a completely new IAM solution needs to be designed and implemented, while more often, an IAM system is already there and running, and customers are seeking to upgrade or migrate to new systems. In either case, a good understanding of the customer's current state is crucial for the success of the project.

After an agreed scope of work has been defined between consultants and customer, the next step is the plan and design of the solution. This is usually done among more experienced engineers and again requires collaborative efforts between consultants and customers. Sometimes, prototypes will be developed as the proof-of-concept of the design.

A lot of times, the starting point of implementation is not when design is completely done and they can go in parallel. In this case, design work goes a little ahead of the implementation work and runs in parallel. The implementation process follows typical SDLC (Software Development Lifecycle) and uses agile practices. The implementation phase could take months or even years to finish.

Once the implementation is done and delivered to the customer, the consulting goes to the closing phase. Customers can choose to close the work or bring in the requirement for the next step of the project.

IAM Expertise

The above process is actually pretty typical for general consulting service. The specific part for IAM consulting would be the expertise and experiences related to the IAM products from various IAM vendors. In order to give proper advice and implement the solution for clients, IAM consultants need to know their products well. Modern IAM solution platforms, such as ForgeRock, Ping Identity, Okta, Auth0 etc, are all different and complicated products. Having a good understanding of the function and feature that the product can provide is crucial to an IAM consultant.

Besides the IAM product itself, another important aspect is integration patterns. IAM platforms are fairly large systems and usually a mid-to-large company will have its own system as well and the challenge and gap come as how to integrate the IAM product into the client system. Understanding different integration patterns would be very helpful in this case, as a lot of times, while the IAM products and client systems are different, the integration pattern can be reused. For example, when dealing with SSO (Single-Sign On), one immediately thinks about OAuth/OIDC or SAML as the integration pattern and this pattern will lay a good foundation for solving the SSO issue.

Besides the above two, there are many other aspects of IAM expertise as well:

- Typical IAM concepts and frameworks
- Coding and scripting skill
- Knowledge on Networks
- Knowledge on Database and Directories
- Experience with Cloud Service platforms
- Agile process experience

Appendix II - How to Become an IAM Consultant

How People Become IAM Professionals

First of all, a lot of IAM professionals didn't start their IAM career right from the start. This could be due to the fact that IAM is not a popular field for new graduates as well as among those in the early years of their professional career. More often than not, people choose something that is popular as the starting point of their career, e.g. software developer, cloud engineer, and this is easy to understand. However, after several years into the industry, some people happened to be involved in IAM related projects, which is usually a crucial component of an organization's security infrastructures. Then, they stayed in the area and made a career out of it.

This pattern probably won't change for quite some time, which is one of the reasons that good IAM consultants will be difficult to find from the market in the long run, because the stream of incoming IAM candidates are still limited. This limitation will continue adding values to those already in the field for the next ten to twenty years, at least to some extent. (Ask ChatGPT about the IAM market and career).

A Roadmap to Get Started

On the other hand, if this field attracts you and you intend to chase this path as your career, there are a couple of steps you can follow to achieve it.

Have Good Understanding on the IAM Basics

There are a couple of important concepts and knowledge related to the IAM field. It's important to have a good grasp of them. You can do this by checking out books, tutorials, online courses, attending workshops and seminars. One book I recommend is Solving Identity Management in Modern Applications (2nd version available as of the writing of this post). This book talks about Identity basics, core frameworks such as OAuth 2.0, OpenID Connect, SAML 2.0, Authorizations, Single Sign-On etc.

The contents of the book lay a good map for important IAM topics that you can research as well. You don't have to dive deep in each topic; having some understanding of each topic should get you started.

Hands-on Practices

You don't have to wait until you understand all the IAM related topics to start working on some small tasks. Actually, getting your hands dirty working on some of the concepts is crucial to understand it. For example, when learning OAuth 2.0, you may just google online using keywords like 'OAuth 2.0 practice' and then you may find OAuth 2.0 Playground - a sandbox lab environment to play with OAuth 2.0. You may find some of the topics have more resources, while others have less. Don't just read the book when trying to learn some topic, but practice as well to enhance your understanding. Use your own judgment when you feel good to move on.

Programming Skills

Programming skills are NOT a must for an IAM career, but if you have it, it will greatly enhance your understanding as well as your career path in IAM. You don't have to understand many languages, but it is essential to have a deep understanding of one language. Here, I would like to recommend Java. Java is a preferred programming language in the world of security due to some of the merits related to security. Plus, popular products like ForgeRock and Ping Identity are written in Java and when developing custom plugins, you will be using Java for that.

Project Experiences

Project experiences could be difficult to acquire in general, especially if you are just starting to learn IAM yourself; after all, industry projects require candidates with experiences while new joiners are looking for projects to gain experience.

However, this doesn't necessarily mean you can't find any projects to work on. Search for books and online courses, and from time to time, you can find guided tutorials on doing hands-on projects. Even if it is difficult to find one, you can start to think about requirements yourself and use case scenarios that need to be addressed, and then implement solutions for it. This project experience can be later shown on your resume as well.

Get Certified

Certificates could be another good way to prepare for your IAM career. By passing exams, you have acquired a good amount of knowledge. Furthermore, certificates can be shown in your resume to add credibility to your IAM skills. Also, passing an exam as a target usually leads to more productivity and encouragement. Modern IAM vendors such as Ping Identity, ForgeRock and Okta all provide certification exams.

Network with IAM Professionals

Networking is an important aspect of any career development. Connect with IAM professionals on LinkedIn, join seminars and events and interact with IAM groups. This will help you get the latest from the field and make valuable connections for potential future opportunities.

Developing Consulting Skills

As issues usually with technology professionals, they are versatile on the technical aspects, but more of than not, lack of soft skills. Strong communication and interpersonal skills, as well as the ability to understand the business needs of your clients are of the same importance as the technical side. Consider taking courses or training programs in consulting and project management to help you develop these skills.

The steps mentioned above don't need to be done strictly in the order of sequence. Having a good understanding of the IAM theory will probably lay a good foundation for other work, but you may still work on some of the topics in parallel.

In general, by following these steps, it will get you started on the road of being a successful IAM consultant. Yet, this also takes time and dedication. Even after you successfully land a job as an IAM professional, you still need to stay active and continue learning to be up-to-date. This will help your IAM career advance and eventually be successful.