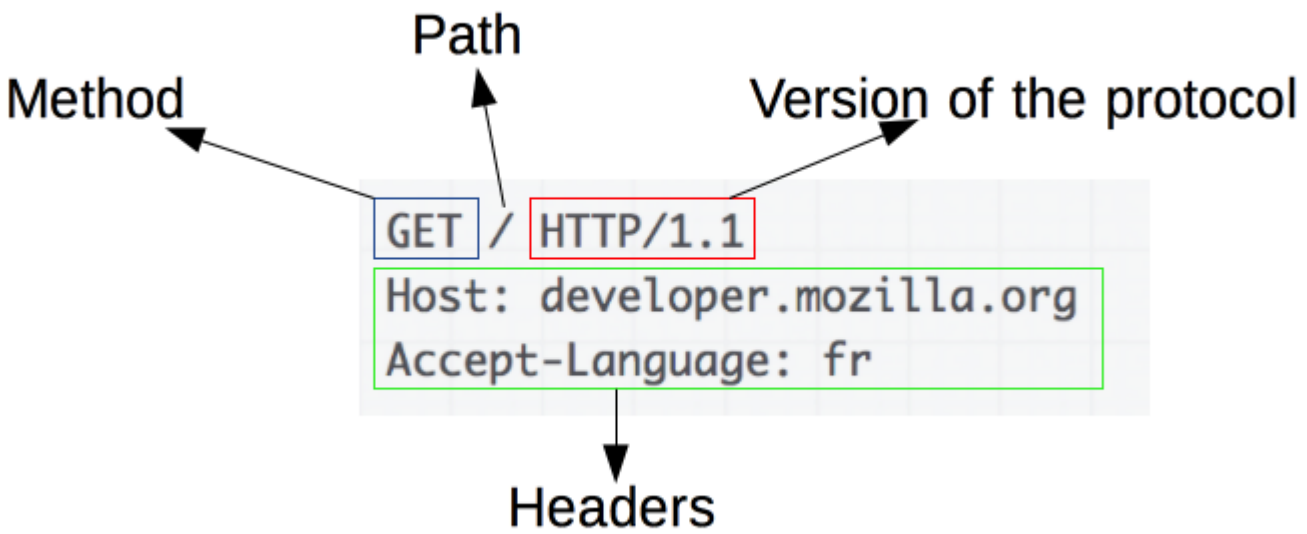
HTTP, HTTPS, REST, RESTful API

HTTP

- HTTP(Hyper Text Transfer Protocol)란 **서버/클라이언트 모델을 따라 데이터를 주고 받기 위한 프로토콜**
- **인터넷에서 하이퍼텍스트를 교환하기 위한 통신 규약**
- **80번 포트**를 사용한다. HTTP 서버가 80번 포트에서 요청을 기다리고 있으며, 클라이언트는 80번 포트로 요청을 보내게 된다.
- **애플리케이션 레벨**의 프로토콜로 **TCP/IP 위에서 작동**
- 상태를 가지고 있지 않는 **Stateless** 프로토콜
- **Method, Path, Version, Headers, Body** 등으로 구성



- HTTP는 **암호화가 되지 않은 평문 데이터를 전송**하는 프로토콜이었기 때문에, HTTP로 비밀번호나 주민등록번호 등을 주고 받으면 제3자가 정보를 조회할 수 있었다.

HTTPS

- HTTP에 **데이터 암호화**가 추가된 프로토콜 (네트워크 상에서 중간에 제 3자가 정보를 볼 수 없도록 암호화를 지원)
- **443번 포트**를 사용

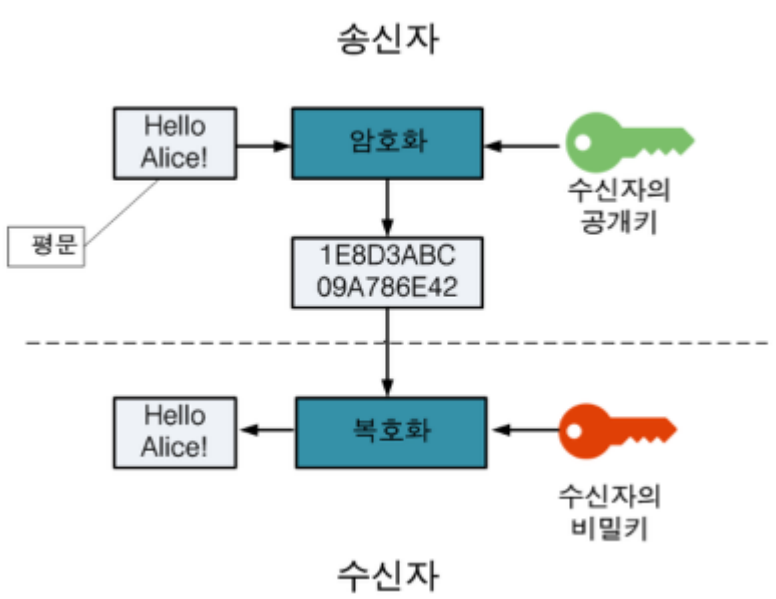
대칭키 암호화와 비대칭키 암호화

- 대칭키 암호화
    - **클라이언트와 서버가 동일한 키**를 사용해 암호화/복호화를 진행함
    - **키가 노출되면 매우 위험**하지만 **연산 속도가 빠름**

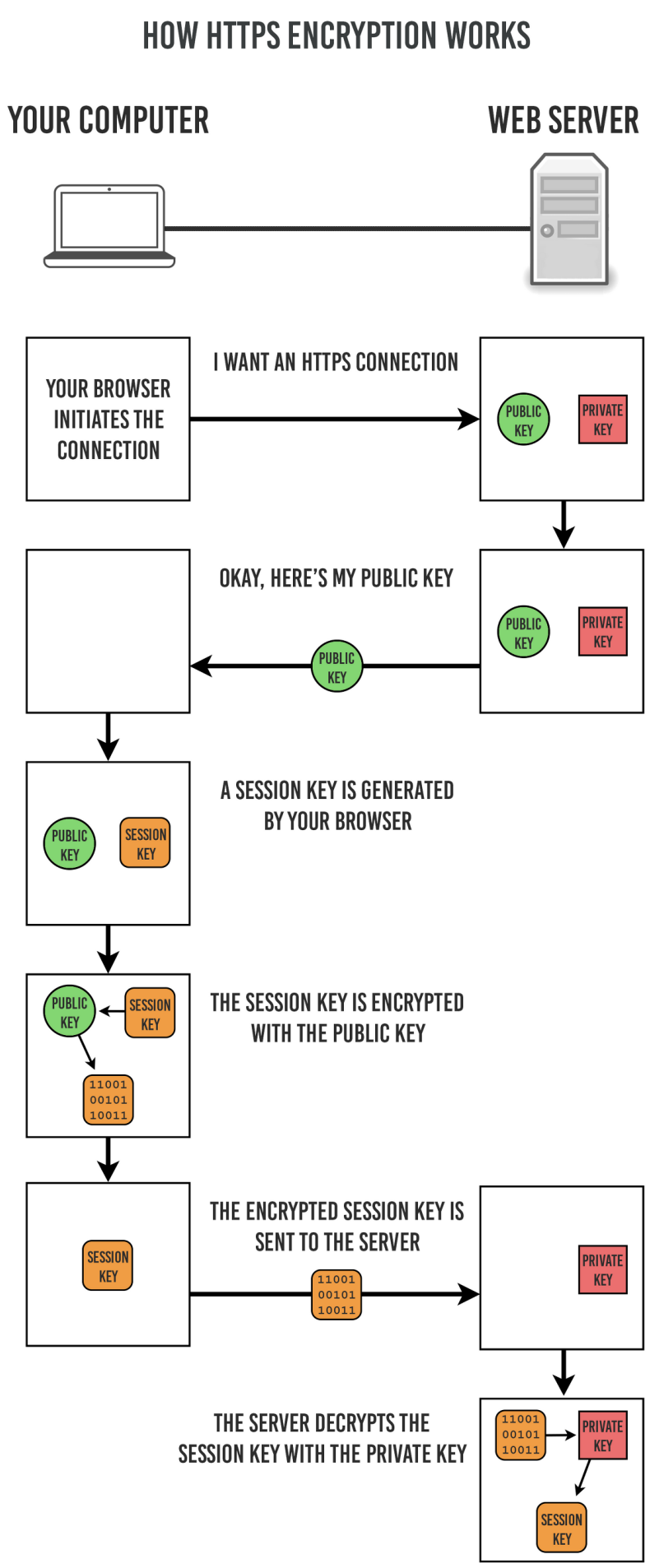
- 비대칭키 암호화
  - 1개의 쌍으로 구성된 **공개키와 개인키**를 암호화/복호화 하는데 사용함
  - **키가 노출되어도 비교적 안전**하지만 **연산 속도가 느림**
- HTTPS는 대칭키 암호화와 비대칭키 암호화를 모두 사용하여 빠른 연산 속도와 안정성을 모두 얻고 있다.

▼ 참고 (공개키 / 개인키)

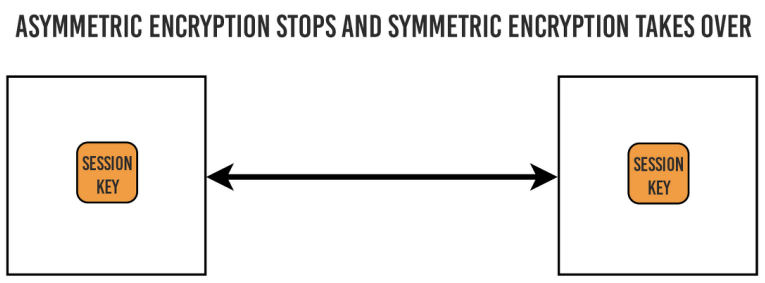
- 공개키와 개인키는 서로를 위한 1쌍의 키
- 공개키: 모두에게 공개가능한 키
- 개인키: 나만 가지고 알고 있어야 하는 키
- 공개키 암호화: 공개키로 암호화를 하면 개인키로만 복호화할 수 있다. -> 개인키는 나만 가지고 있으므로, 나만 볼 수 있다.
- 개인키 암호화: 개인키로 암호화하면 공개키로만 복호화할 수 있다. -> 공개키는 모두에게 공개되어 있으므로, 내가 인증한 정보임을 알려 신뢰성을 보장할 수 있다.



동작과정



1. 클라이언트(브라우저)가 서버로 최초 연결 시도를 함
2. 서버는 공개키(엄밀히는 인증서)를 브라우저에게 넘겨줌
3. 브라우저는 인증서의 유효성을 검사하고 세션키를 발급함
4. 브라우저는 세션키를 보관하며 추가로 서버의 공개키로 세션키를 암호화하여 서버로 전송함
5. 서버는 개인키로 암호화된 세션키를 복호화하여 세션키를 얻음
6. 클라이언트와 서버는 동일한 세션키를 공유하므로 데이터를 전달할 때 세션키로 암호화/복호화를 진행함



HTTPS의 장단점

- 장점
  - 네트워크 상에서 열람, 수정이 불가능하므로 **안전하다**.
- 단점

- **암호화**를 하는 과정이 **웹 서버에 부하**를 준다.
- HTTPS는 설치 및 인증서를 유지하는데 **추가 비용이 발생**한다.
- 암호화/복호화의 과정이 필요하기 때문에 **HTTP보다 속도가 느리다**. (물론 오늘날에는 거의 차이를 못느낄 정도이다.)
- 인터넷 **연결이 끊긴 경우 재인증 시간이 소요**된다.
  - HTTP는 비연결형으로 웹 페이지를 보는 중 인터넷 연결이 끊겼다가 다시 연결되어도 페이지를 계속 볼 수 있다.
  - 그러나 HTTPS의 경우에는 소켓(데이터를 주고 받는 경로) 자체에서 인증을 하기 때문에 인터넷 연결이 끊기면 소켓도 끊어져서 다시 HTTPS 인증이 필요하다.

HTTP 메소드

- 클라이언트와 서버 사이에 이루어지는 요청(Request)과 응답(Response) 데이터를 전송하는 방식
- 서버에 주어진 리소스에 수행하길 원하는 행동, 서버가 수행해야 할 동작을 지정하는 요청을 보내는 방법
- HTTP 메소드의 종류는 총 9가지가 있다.
  - **GET : 리소스 조회**
  - **POST: 요청 데이터 처리, 주로 등록에 사용**
  - **PUT : 리소스를 대체(덮어쓰기), 해당 리소스가 없으면 생성**
  - **PATCH : 리소스 부분 변경 (PUT이 전체 변경, PATCH는 일부 변경)**
  - **DELETE : 리소스 삭제 기타 메소드**
  - HEAD : GET과 동일하지만 메시지 부분(body 부분)을 제외하고, 상태 줄과 헤더만 반환
  - OPTIONS : 대상 리소스에 대한 통신 가능 옵션(메서드)을 설명(주로 CORS에서 사용)
  - CONNECT : 대상 자원으로 식별되는 서버에 대한 터널을 설정
  - TRACE : 대상 리소스에 대한 경로를 따라 메시지 루프백 테스트를 수행

GET VS POST

처리 방식	GET 방식	POST 방식
URL에 데이터 노출 여부	O	X
URL 예시	http://localhost:8080/boardList?name=제목&contents=내용	http://localhost:8080/addBoard
데이터의 위치	Header(헤더)	Body(바디)
캐싱 가능 여부	O	X

Caching(캐싱)이란? 캐싱이란 한번 접근 후, 또 요청할 시 빠르게 접근하기 위해 레지스터에 데이터를 저장시켜 놓는 것

HTTP 상태코드

- **1XX: Informational(정보 제공)**
  - 임시 응답으로 현재 클라이언트의 요청까지는 처리되었으니 계속 진행하라는 의미
- **2XX: Success(성공)**
  - 클라이언트의 요청이 서버에서 성공적으로 처리되었다는 의미
- **3XX: Redirection(리다이렉션)**
  - 완전한 처리를 위해서 추가 동작이 필요한 경우. 주로 서버의 주소 또는 요청한 URI의 웹 문서가 이동되었으니 그 주소로 다시 시도하라는 의미
- **4XX: Client Error(클라이언트 에러)**
  - 없는 페이지를 요청하는 등 클라이언트의 요청 메시지 내용이 잘못된 경우를 의미

- **5XX: Server Error(서버 에러)**
    - 서버 사정으로 메시지 처리에 문제가 발생한 경우. 서버의 부하, DB 처리 과정 오류, 서버에서 익셉션이 발생하는 경우를 의미

상태코드	
200	클라이언트의 요청을 정상적으로 수행함
201	클라이언트가 어떠한 리소스 생성을 요청, 해당 리소스가 성공적으로 생성됨(POST를 통한 리소스 생성 작업 시)
301	클라이언트가 요청한 리소스에 대한 URI가 변경 되었을 때 사용하는 응답 코드
	(응답 시 Location header에 변경된 URI를 적어 줘야 합니다.)
400	클라이언트의 요청이 부적절 할 경우 사용하는 응답 코드
401	클라이언트가 인증되지 않은 상태에서 보호된 리소스를 요청했을 때 사용하는 응답 코드
	(로그인 하지 않은 유저가 로그인 했을 때, 요청 가능한 리소스를 요청했을 때)
403	유저 인증상태와 관계 없이 응답하고 싶지 않은 리소스를 클라이언트가 요청했을 때 사용하는 응답 코드
	(403 보다는 400이나 404를 사용할 것을 권고. 403 자체가 리소스가 존재한다는 뜻이기 때문에)
405	클라이언트가 요청한 리소스에서는 사용 불가능한 Method를 이용했을 경우 사용하는 응답 코드
500	서버에 문제가 있을 경우 사용하는 응답 코드

REST

- HTTP 통신에서 어떤 자원에 대한 CRUD 요청을 Resource와 Method로 표현하여 특정한 형태로 전달하는 방식

REST의 특징

1. **Uniform Interface(일관된 인터페이스)**
    - 요청을 통일되고, 한정적으로 수행하는 아키텍처 스타일
    - 요청을 하는 Client가 플랫폼(Android, Ios, Jsp 등) 에 무관하며, 특정 언어나 기술에 종속받지 않는 특징
    - 이러한 특징 덕분에 Rest API는 HTTP를 사용하는 모든 플랫폼에서 요청 가능하며, Loosely Coupling(느슨한 결함) 형태를 가짐
2. **Stateless(무상태성)**
    - 서버는 각각의 요청을 별개의 것으로 인식하고 처리해야하며, 이전 요청이 다음 요청에 연관되어서는 안됨
    - 그래서 Rest API는 세션정보나 쿠키정보를 활용하여 작업을 위한 상태정보를 저장 및 관리하지 않는다.
    - 무상태성 때문에 Rest API는 서비스의 자유도가 높으며, 서버에서 불필요한 정보를 관리하지 않으므로 구현이 단순
    - 서버의 처리방식에 일관성을 부여하고, 서버의 부담을 줄이기 위함
3. **Cacheable(캐시 가능)**
    - Rest API는 결국 HTTP라는 기존의 웹표준을 그대로 사용하기 때문에, 웹의 기존 인프라를 그대로 활용할 수 있다. 그러므로 Rest API에서도 캐싱 기능을 적용할 수 있음
    - HTTP 프로토콜 표준에서 사용하는 Last-Modified Tag 또는 E-Tag를 이용하여 캐싱을 구현할 수 있고, 이것은 대량의 요청을 효율적으로 처리할 수 있게 도움
4. **Client-Server Architecture (서버-클라이언트 구조)**
    - Rest API에서 자원을 가지고 있는 쪽이 서버, 자원을 요청하는 쪽이 클라이언트에 해당
    - 서버는 API를 제공하며, 클라이언트는 사용자 인증, Context(세션, 로그인 정보) 등을 직접 관리하는 등 역할을 확실히 구분시킴으로써 서로 간의 의존성을 줄임
5. **Self-Descriptiveness(자체 표현)**

- Rest API는 요청 메세지만 보고도 이를 쉽게 이해할 수 있는 자체 표현 구조로 되어있다.
- 아래와 같은 JSON 형태의 Rest 메세지는 http://localhost:8080/board 로 게시글의 제목, 내용을 전달하고 있음을 손쉽게 이해할 수 있다. 또한 board라는 데이터를 추가(POST)하는 요청임을 파악할 수 있다.

```
HTTP POST , http://localhost:8080/board
{
        "board":{
                "title":"제목",
                "content":"내용"
        }
}
```

6. **Layered System(계층 구조)**
   - Rest API의 서버는 다중 계층으로 구성될 수 있으며 보안, 로드 밸런싱, 암호화 등을 위한 계층을 추가하여 구조를 변경할 수 있다.
   - Proxy, Gateway와 같은 네트워크 기반의 중간매체를 사용할 수 있게 해줌
   - 하지만 클라이언트는 서버와 직접 통신하는지, 중간 서버와 통신하는지 알 수 없다.

RESTful API

REST의 규칙

1. URI는 명사를 사용한다.
2. 슬래시로 계층 관계를 표현한다.
3. URI의 마지막에는 슬래시를 붙이지 않는다.
4. URI는 소문자로만 구성한다.
5. 가독성이 떨어지는 경우 하이픈을 사용한다.

RESTful API의 구성요소

- **Resource**
  - 서버는 Unique한 ID를 가지는 Resource를 가지고 있으며, 클라이언트는 이러한 Resource에 요청을 보낸다. 이러한 Resource는 URI에 해당함
- **Method**
  - 서버에 요청을 보내기 위한 방식으로 GET, POST, PUT, PATCH, DELETE가 있다. CRUD 연산 중에서 처리를 위한 연산에 맞는 Method를 사용하여 서버에 요청을 보내야 한다.
- **Representation of Resource**
  - 클라이언트와 서버가 데이터를 주고받는 형태로 json, xml, text, rss 등이 있다. 최근에는 Key, Value를 활용하는 json을 주로 사용

장단점

1. 장점
   - 독립성: **클라이언트와 서버가 분리**되어 있기 때문에 **독립적으로 개발 및 유지보수** 가능
   - 단순성: 표준 HTTP를 사용하기 때문에 사용하고 **이해하기 쉬움**. 기존 HTTP 인프라 사용가능
   - 자원 효율성: 다른 API와 달리 JSON과 같은 더 가벼운 포맷을 사용하고, 필요 없는 요소들은 배제해서 **더 적은 자원과 대역폭을 사용함**
   - 확장성 및 유연성: 클라이언트와 서버가 분리되어 있기 때문에 **확장 용이**. 리소스의 표현을 다양하게 제공하기 때문에 **다양한 클라이언트에 대응 가능**
2. 단점
   - **표준 규약이 없음**
   - HTTP 메서드 형태가 제한적: HTTP 메서드를 사용해 URI를 표현하기 때문에 **HTTP 메서드에 제한됨**