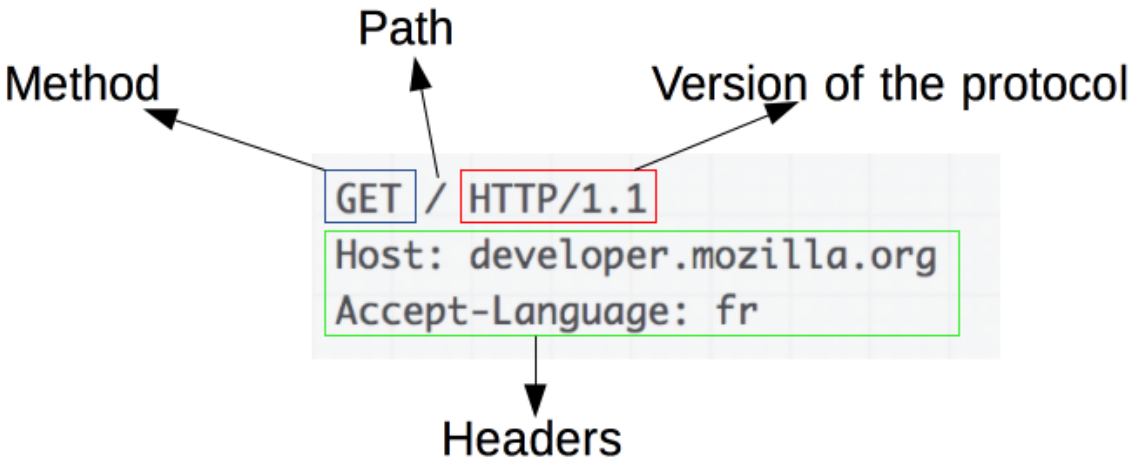
HTTP

HTTP&HTTPS

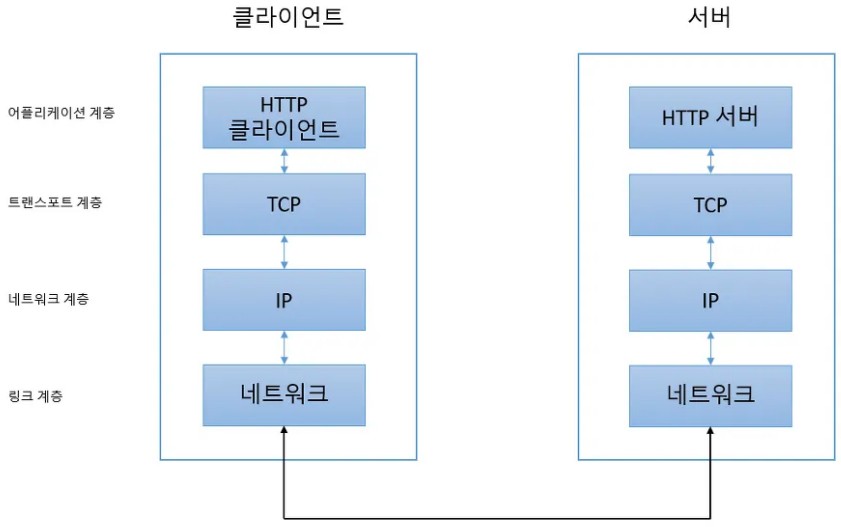
태그	NETWORK
주차	7주차

📌 목차

1. HTTP



- HyperText Transfer Protocol
  - HTML과 같은 HyperText 문서를 주고받기 위해 만들어짐
  - 최근에는 모든 웹 관련 API 통신에 이용하는 통신 프로토콜
- 서버와 클라이언트 간의 TCP/IP 통신 위에서 메세지를 교환하기 위해 사용되는 프로토콜
- 포트번호 80



- **비연결성**과 **무상태성**의 특징을 가지는 통신

비연결성(Connectionless)

- 처음 연결을 맺은 후 요청(Request)와 한번의 응답(Response) 이후 연결이 종료됨
- 매 요청마다 다시 연결을 맺음
- 매번 연결을 맺어 느려지는 것을 보완하기 위해 KeepAlive와 같은 속성 활용 가능

무상태성(Stateless)

- 프로토콜에서 client의 상태를 기억하지 않음
- client의 상태를 보관하기 위해 쿠키나 세션, JWT 토큰 등을 이용해 client의 상태 유지

1-2. Request 구조

1) start-line

- **HTTP Method**
  - `GET` / `POST` / `PUT` / `PATCH` / `DELETE`와 같은 http method를 적는 부분
  - 서버에서 요청을 routing할 때 사용되며, 각각의 method는 차례로 READ/CREATE/UPDATE(all)/UPDATE(part)/DELETE를 의미
- **URL**
  - 리소스를 요청하는 주소
  - 전체 주소를 입력하기도 하며, 같은 host에서 요청을 보낼 경우 `/{리소스 경로}`와 같이 path만 요청 가능
- **protocol version**
  - http의 버전
  - HTTP/1.1을 가장 많이 사용하며 최근에는 HTTP/2의 사용이 늘고 있음

2) Header

- 요청에 대한 정보, 응답에 대한 요청, 인증 정보, 접속 정보 등 요청에서 필요한 다양한 요소를 담을 수 있는 공간
- key, value로 구성되어 있음
- **Content-Type**
  - Body에 들어가는 요청 전문의 Type

파일 타입	content-type
JSON	`application/json`
이미지/파일	`multipart/form-data`
바이너리가 없는 form	`application/x-www-form-urlencoded`
타입이 없는 text	`plain/text`

- **Accept**
  - 응답 받을 메세지 타입을 명시
- **Connection**
  - 주로 keep-alive로 셋업
  - keep-alive로 셋업하면 매 요청 시 커넥션을 다시 맺지 않고 커넥션을 유지하므로 성능 향상을 기대할 수 있음
- User-Agent
  - 사용자의 기기 식별 가능
  - 기기/OS/브라우저 별 예외를 처리할 때 많이 사용되며, 사용자 통계 수집 용도로도 사용
- Authorization
  - 인증 정보를 담을 때 사용하는 헤더
  - 주로 인증 토큰을 Authorization 헤더에 담아 보냄
- Cookie
  - 개인 브라우저에 저장되는 Cookie 정보를 보낼 때 사용하는 헤더
- Session-id
  - Session에 대한 id값을 지정하는 부분

☝🏻 Cookie나 Session-id의 경우 프레임워크나 라이브러리에서 처리해주기 때문에 직접 셋업할 일이 많지는 않지만 해당 헤더의 존재를 알고 있다면 postman이나 restclient 사용 시 해당 정보를 header에 넣어 시뮬레이션 하는데 사용할 수 있음

3) Empty-line

- Header와 Body를 구분하는 부분

4) Body

- 메세지 전문이 들어감
- Header의 Content-type과 꼭 Type을 맞춰줘야 함
- HTTP Method 중 GET/DELETE는 body를 쓰지 않음

1-3. Response 구조

1) start-line

- **protocol version**
- **HTTP Status Code**
- **HTTP Status**
  - 요청에 대한 응답 상태
  - 1XX(정보): 요청을 받았으며 프로세스를 계속 진행
  - 2XX(성공): 요청을 성공적으로 받았으며 인식했고 수용함
  - 3XX(리다이렉션): 요청 완료를 위해 추가 작업 조치가 필요
  - 4XX(클라이언트 오류): 요청의 문법이 잘못되었거나 요청을 처리할 수 없음
  - 5XX(서버 오류): 서버가 명백히 유효한 요청에 대한 충족을 실패

2) Header

- Request와 유사하나, 서버에 대한 정보를 건내줌
- 사용자의 cookie나 session을 초기화 하는 데도 이용

3) Empty-line

- Request와 동일

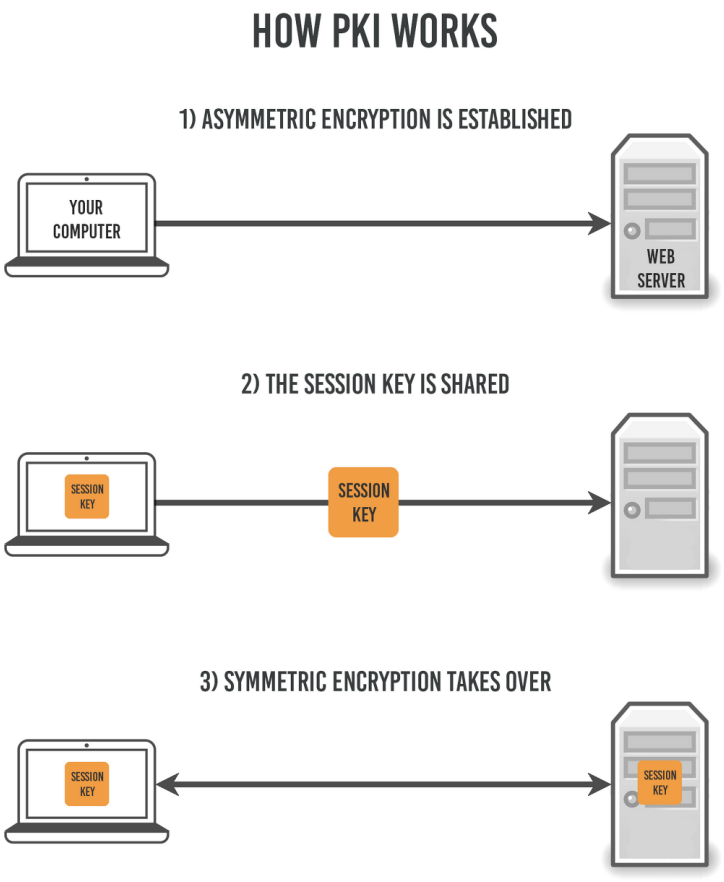
4) Body

- Request와 동일

2. HTTPS

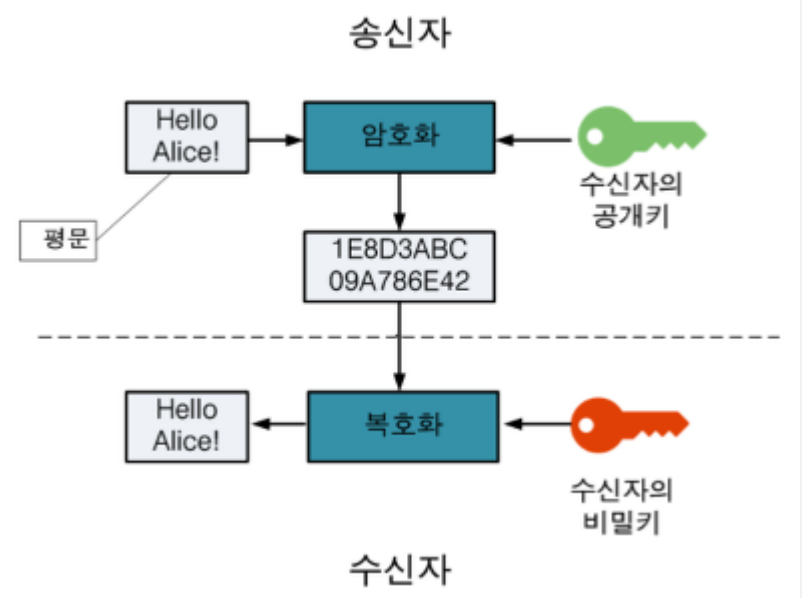
- HTTP의 보안 버전. 데이터 전송의 보안을 강화하기 위해 암호화됨
- 암호화 프로토콜을 사용해 통신을 암호화함
- 이전에는 SSL(Secure Sockets Layer)으로 알려졌지만, TLS(Transport Layer Security)라고 불림
    - 특정 공급자가 주장하는 실체가 맞는지 확인
- 포트번호 443

2-1. 작동 원리



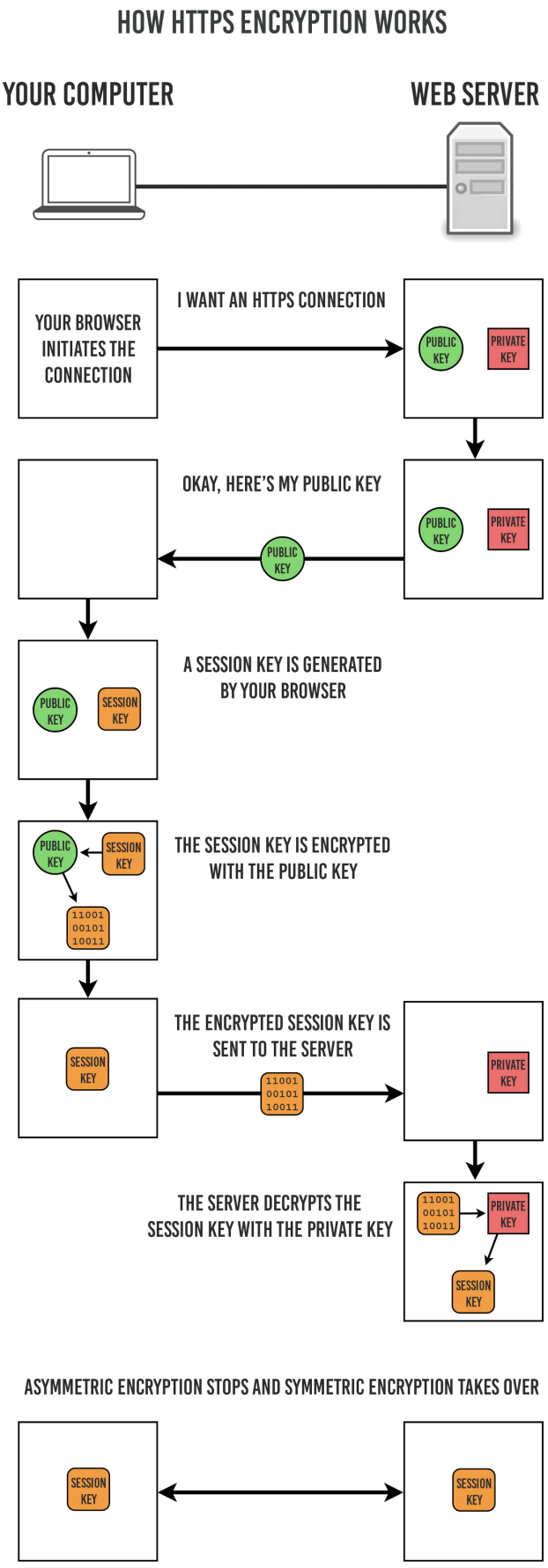
- 대칭키와 비대칭키를 모두 이용해 통신 보호
- 처음 연결을 성립하여 클라이언트와 서버가 세션키를 공유하는 과정에서 안전한 연결을 위해 **비대칭키** 사용
    - HTTPS 연결 과정(Hand-Shaking)에서 서버와 클라이언트간에 세션키 교환
    - 세션키는 대칭키로 만들어짐
- 이후 데이터를 교환하는 과정에서 빠른 연산을 위해 **대칭키** 사용

☝🏻 **비대칭키**



1. **개인 키(private key)**
    - 웹 사이트 소유자가 관리하며 비공개로 유지
    - 키는 웹 서버에 존재하며 공개 키로 암호화된 정보를 해독하는 데 사용
2. **공개 키(public key)**
    - 안전한 방식으로 서버와 상호작용하고자 하는 모든 사람들이 사용할 수 있음
    - 공개키로 암호화된 정보는 개인 키로만 해독 가능

2-2. 동작 흐름


HOW HTTPS ENCRYPTION WORKS
YOUR COMPUTER
WEB SERVER
YOUR BROWSER INITIATES THE CONNECTION
I WANT AN HTTPS CONNECTION
PUBLIC KEY
PRIVATE KEY
OKAY, HERE'S MY PUBLIC KEY
PUBLIC KEY
PRIVATE KEY
PUBLIC KEY
A SESSION KEY IS GENERATED BY YOUR BROWSER
SESSION KEY
PUBLIC KEY
SESSION KEY
11001 00101 10011
THE SESSION KEY IS ENCRYPTED WITH THE PUBLIC KEY
SESSION KEY
THE ENCRYPTED SESSION KEY IS SENT TO THE SERVER
11001 00101 10011
PRIVATE KEY
11001 00101 10011
PRIVATE KEY
THE SERVER DECRYPTS THE SESSION KEY WITH THE PRIVATE KEY
SESSION KEY
ASYMMETRIC ENCRYPTION STOPS AND SYMMETRIC ENCRYPTION TAKES OVER
SESSION KEY
SESSION KEY

1. 클라이언트(브라우저)가 서버로 최초 연결 시도
2. 서버는 공개키(인증서)를 클라이언트에게 넘겨줌
3. 클라이언트는 공개키(인증서)의 유효성을 검사한 뒤 세션키 발급
4. 클라이언트는 세션키를 보관하며 추가로 서버의 공개키로 세션키를 암호화하여 서버로 전송
5. 서버는 개인키로 암호화된 세션키를 복호화해 세션키를 얻음
6. 클라이언트와 서버는 동일한 세션키를 공유하므로 데이터를 전달할 때 세션키로 암호화/복호화 진행

1) HTTPS의 발급 과정

- 서버는 클라이언트와 세션키를 공유하기 위한 공개키를 생성해야 하는데, 일반적으로는 인증된 기관(Certificate Authority)에 공개키를 전송해 인증서를 발급받음

2-2. HTTP와의 차이점

- 엄밀히 말하면 별개의 프로토콜은 아님
  - 단순히 TLS/SSL 암호화를 사용하는 HTTP
- HTTPS가 안전
- HTTPS가 속도가 비교적 느림