17/12/08 패스트캠퍼스 북 스터디

그림으로 배우는 HTTP & Network Basic

프론트엔드 개발 스쿨 7기, 마가렛트

- 담당 부분
  - 제 5장 HTTP와 연계하는 웹 서버
  - 제 6장 HTTP 헤더
  - 제 7장 웹을 안전하게 하는 HTTPS

제 5장 HTTP와 연계하는 웹 서버

5.1 가상 호스트 기능은 서버 한 대로 멀티 도메인을 가능하게 한다

- 가상 호스트(Virtual Host)
    - 하나의 HTTP 서버에서 여러 웹 사이트 실행이 가능
- 클라이언트 =>
    - 도메인 명은 DNS에 의해 IP주소로 변환된 후 액세스하게 된다.
    - 같은 IP주소에 여러 개의 가상 호스트가 있기 때문에 호스트 명과 도메인 명을 완전히 포함한 URI를 지정하거나 Host 헤더 필드에서 지정해야 한다.

5.2 통신을 중계하는 프로그램: 프록시, 게이트웨이, 터널

HTTP는 클라이언트-서버 사이 외에 중간 통신 중계 프로그램과의 연결도 가능하다.

- 프록시: 클라이언트와 서버를 중계.
  - 캐시를 사용해 네트워크 대역 등을 효율적으로 사용할 수 있다.
  - 특정 웹사이트 액세스 제한 등의 목적을 가질 때도 있다.
  - 캐시 여부에 따라 (캐싱 프록시) 또 메시지 변경 여부에 따라 (투명 프록시) 구분할 수 있다.

- 게이트웨이: 데이터베이스, 결제 시스템 등 안전하게 접속하기 위한 목적으로 사용된다.
    - 동작 자체는 프록시와 비슷하다.
    - 클라이언트에서 HTTP 리퀘스트를 했을 때 해당 서버가 HTTP 서버 이외의 서비스를 제공하는 서버이다.
- 터널: 서버와 안전한 통신로를 확보한다.
    - 암호화 통신 등을 위해 사용된다.

5.3 리소스를 보관하는 캐시

- 캐시: 데이터 값을 미리 복사해놓는 임시 장소. 통신에서는 프록시 서버와 클라이언트 로컬 디스크에 보관된 리소스 사본을 말한다.
    - 캐시 서버가 있으면, (오리진) 서버에 액세스하는 작업을 줄이므로 통신량과 통신 시간을 절약한다.

5.3.1 캐시는 유효기간이 있다

오리진 서버가 변할 수 있기 때문!

5.3.2 클라이언트 측에도 캐시가 있다

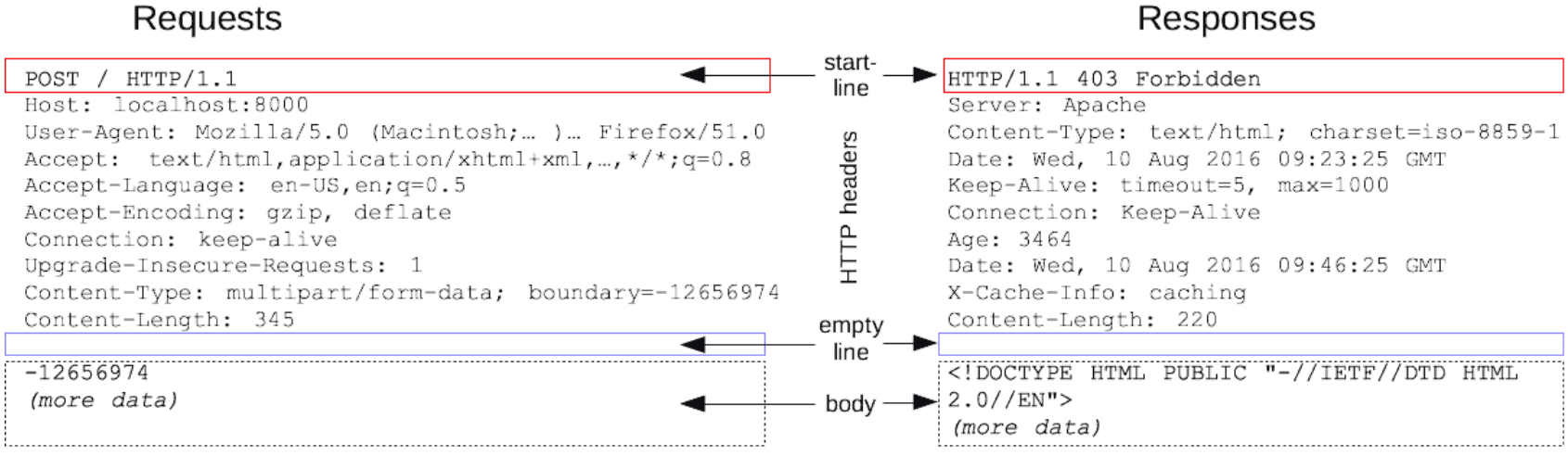
클라이언트가 사용하는 브라우저 상에도 캐시가 있는데, 이를 `인터넷 임시 파일`이라고 한다.

제 6장 HTTP 헤더

6.1 HTTP 메시지 헤더

MDN: HTTP 메시지

HTTP 리퀘스트와 리스폰스의 구조



1. 시작 줄: 실행해야 할 요청 혹은 요청에 대한 성공 혹은 실패를 나타내는 상태를 기술. 항상 한 줄로 이뤄짐. `HTTP 메서드(ex. GET, POST 등) + request target(ex. URL) + HTTP 버전`으로 구성됨.
2. HTTP 헤더: 요청을 적거나, 메시지 내에 포함될 본문을 부연설명. 클라이언트와 서버 처리에 필요한 주요 정보가 거의 다 들어있다. 리퀘스트와 리스폰스에는 반드시 포함되어 있다.
3. 빈 줄: 요청 관련 메타 정보가 모두 전송되었음을 알림.
4. 바디: 요청과 연관된 데이터 (ex. HTML 폼) 혹은 응답과 관련된 문서를 포함. 사용자와 리소스를 필요로하는 정보가 있다. 바디 존재 여부와 크기는 시작줄과 헤더에 적어둔다. 바디가 꼭 있어야 하는 건 아님.

6.2. HTTP 헤더 필드

6.2.1 HTTP 헤더 필드는 중요한 정보를 전달한다

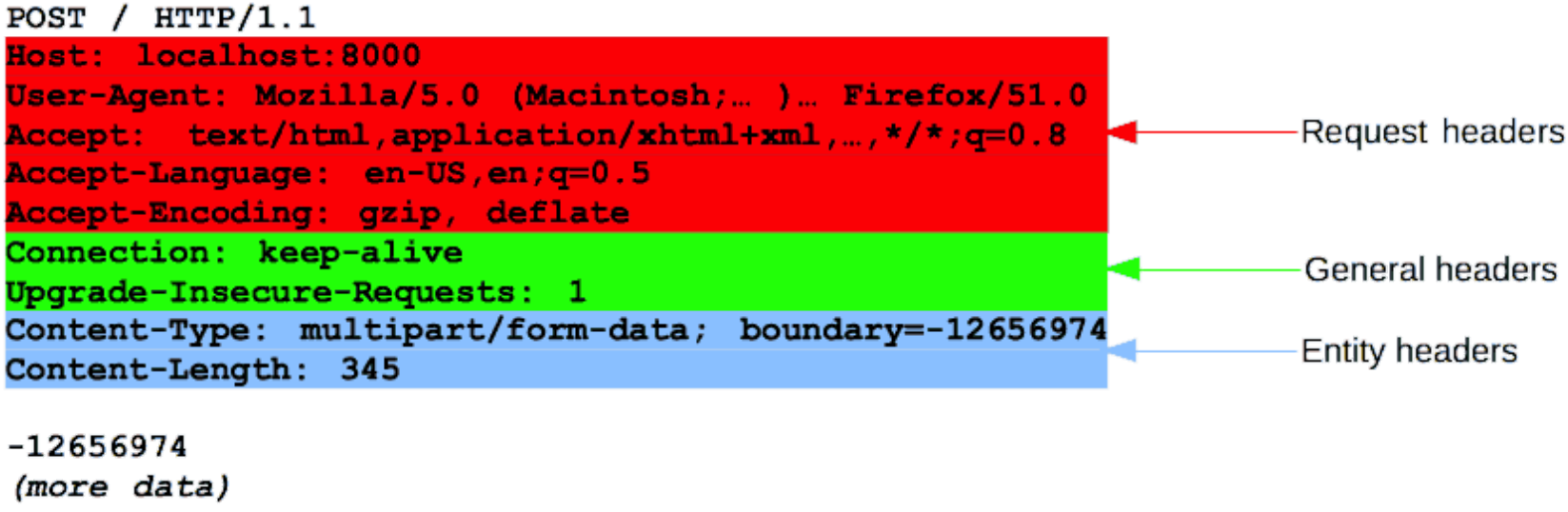
6.2.2. HTTP 헤더 필드의 구조

```
// 헤더 필드 명 : 필드 값
Content-Type:text/html
```

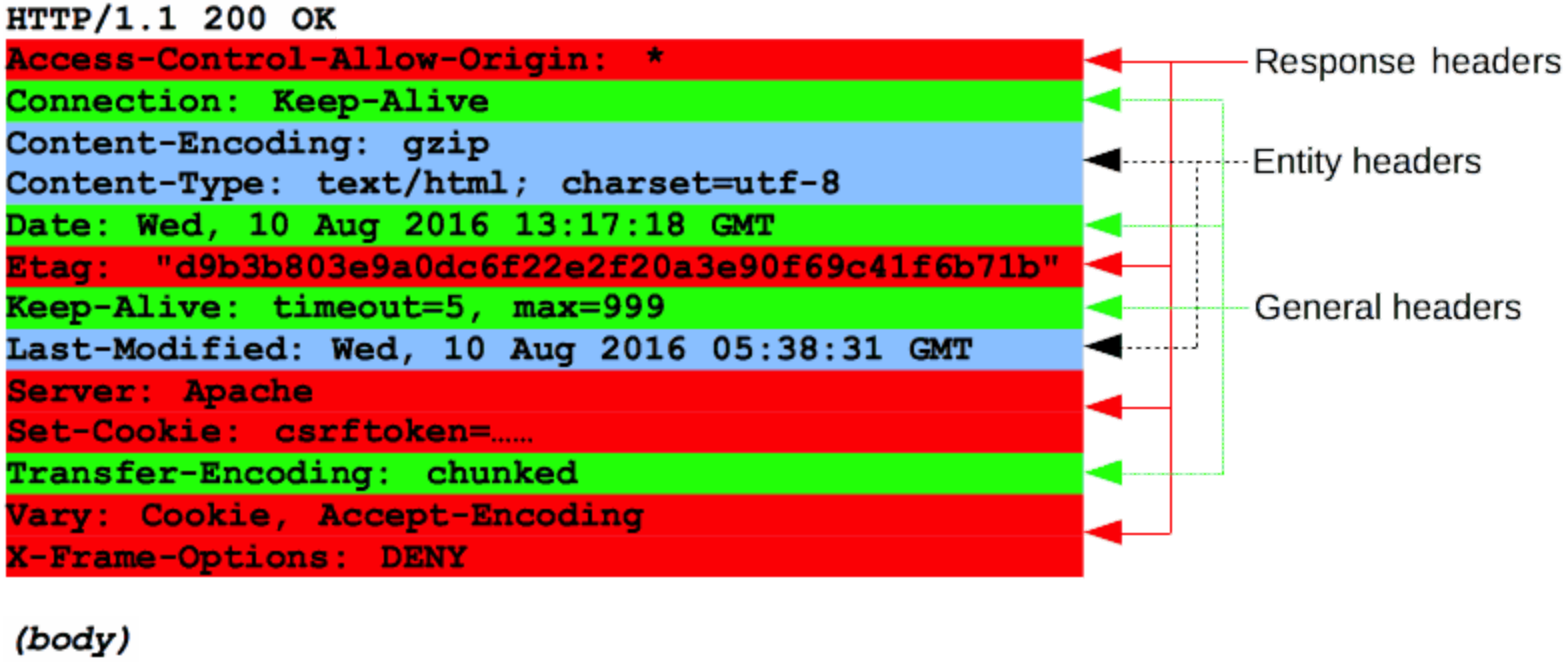
6.2.3 헤더 필드의 네 종류

General(ex. Via) / Request / Response / Entity(ex. Content-Length) 네 개로 구분.

- 리퀘스트 헤더 필드



- 리스폰스 헤더 필드


HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Connection: Keep-Alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Wed, 10 Aug 2016 13:17:18 GMT
Etag: "d9b3b803e9a0dc6f22e2f20a3e90f69c41f6b71b"
Keep-Alive: timeout=5, max=999
Last-Modified: Wed, 10 Aug 2016 05:38:31 GMT
Server: Apache
Set-Cookie: csrftoken=.....
Transfer-Encoding: chunked
Vary: Cookie, Accept-Encoding
X-Frame-Options: DENY
(body)
Response headers
Entity headers
General headers

6.2.4. HTTP/1.1 헤더 필드 일람

(책 참고)

6.2.5. HTTP/1.1 이외의 헤더 필드

비표준 헤더 필드도 사용되고 있음.

6.2.6. End-to-end 헤더와 Hop-by-hop 헤더

캐시 프록시 / 비캐시 프록시 동작 정의를 위해 카테고리가 나뉘어져 있다. **End-to-end 헤더** 는 최종 수신자에게 전송되며, **Hop-to-hop 헤더** 는 한 번의 전송만이 유효하고 캐시와 프록시까지만 전송되기도 한다.

6.3. HTTP/1.1 General Header Fields(일반 헤더 필드)

일반 헤더 필드는 메시지 전체에 적용된다.

- Cache-control: 캐시의 동작을 지정한다. (상세 디렉티브 내용은 책 참조.)

```
// 여러 개의 디렉티브를 지정하는 경우
Cache-Control: private, max-age=0, no-cache
```

- Connection: **프록시에 더 이상 전송하지 않는 헤더 필드**(hop-to-hop 헤더)를 지정하고, **지속접속을 관리**하는 역할을 한다.

```
//프록시에 전송하지 않을 헤더필드 지정
Connection: Upgrade

//접속 끊기
Connection: Close
```

- Date: HTTP 메시지 생성 날짜를 표시한다.
- Pragma: 구버전(HTTP/1.0)의 흔적으로 호환성만을 위해 정의되어 있다.
- Trailer: 메시지 바디에 기술되어 있는 내용을 미리 알려준다.

```
HTTP/1.1 200 OK
Date: Tue, 03, Jul 2012 04:11:42 GMT
...
Trailer: Expires

...메시지 바디...
Expires: Tue, 28 Sep 2004 23:59:59 GMT
```

- Transfer-Encoding: 메시지 바디의 전송코딩 형식을 지정한다.
- Upgrade: HTTP 및 다른 프로토콜의 새로운 버전이 통신에 이용되는 경우 사용된다. 이때, Connection도 Upgrade 로 지정해줘야 한다.
- Via: 프록시 혹은 게이트웨이 이용 시에 해당 서버 정보를 Via 에 추가한다. 경로를 알기 위해 사용된다.

- Warning: 기본적으로 7개의 경고 코드가 미리 정의되어 있다.
  - 110: response is state (프록시가 유효기간 지난 리소스를 반환)
  - 111: revalidate failed (프록시가 리소스 유효성 재확인에 실패)
  - 112: Disconnection operation (프록시가 네트워크로부터 끊겨있음)
  - 113: Heuristic expiration (리스폰스가 유효기간(24시간)을 경과하고 있는 경우)
  - 214: Transformation applied (프록시가 인코딩, 미디어타입 등에 어떤 처리를 한 경우)

6.4. 리퀘스트 헤더 필드

- Accept: 처리할 수 있는 미디어 타입과 우선순위를 전달.
    - Accept-Charset
    - Accept-Encoding
    - Accept-Language
- Authorization: 유저 에이전트의 인증 정보 전달.
    - Proxy-Authorization: 클라이언트와 프록시 사이에 인증이 이뤄진다.
- Expect: 특정 동작을 서버에 요구. 해당 요청에 대해 서버가 응답하지 못하면 417 Expection Failed 를 반환한다.
- From: 메일주소 전달

- **Host**: 헤더에서 필수적인 필드. 요청한 리소스의 인터넷 호스트와 포트 번호를 전달한다. 가상 호스트 때문에 같은 IP주소로 여러 도메인이 적용되어 있을 때, 호스트명을 명확하게 해줘야 하여 이용한다. 호스트명이 없을 경우에는 값을 비워 보낸다.
- If-Match: 조건에 맞으면 보낸다.
    - If-Modificed-Since
    - If-None_Match
    - If-Range
    - If-Unmodified-Since
- Max-Forwards: 전송해도 좋을 서버 수의 최대치를 지정한다.
- Range: 리소스의 지정 범위를 전달.
- Referer: 리퀘스트가 발생한 원래 리소스의 URI 전달. (원래 Referrer가 맞는 철자이나 잘못된 철자 대로 사용 중.)
- User-Agent: 리퀘스트를 생성한 브라우저 종류 전달.

6.5. 리스폰스 헤더 필드

- Accept-Ranges: 리소스를 일부분만 취득할 수 있을지 여부를 전달한다.
- Age: 얼마나 오래 전에 리스폰스가 생성되었는지 전달하며, 초 단위로 사용한다.
- ETag: 엔티티 태그라 불리며, 일시적으로 리소스를 특정하기 위한 문자열을 전달한다. 서버는 리소스마다 ETag를 할당한다.
- Location: Request-URI 이외의 액세스를 유도하는 경우 사용되며 주로 Redirect 시의 해당 URI를 사용한다.
- Proxy-Authenticate
    - WWW-Authenticate

- Retry-After: 일정 시간 후에 리퀘스트를 다시 수행해달라는 요청을 전달한다.
- Server: HTTP 서버의 소프트웨어와 버전, 옵션을 전달한다.
- Vary: 캐시를 컨트롤하기 위해 사용된다. 예를 들자면, 요청과 캐시의 언어가 같을 때는 캐시의 리소스를 전달하되 다른 언어일 경우에는 오리진 서버로 리소스를 가지러 오도록 한다.

6.6. 엔티티 헤더 필드

콘텐츠 갱신기간 등 엔티티에 관한 정보를 포함한다.

- Allow: 사용가능한 메소드 종류를 전달.(ex. GET, POST)
- Content-Encoding
    - Content-Language
    - Content-Length
    - Content-Location
    - Content-MD5
    - Content-Range
    - Content-Type
- Last-Modified

6.7. 쿠키를 위한 헤더 필드

- 쿠키: 유저 식별과 상태관리에 사용되고 있는 기능이다. 유저 상태를 관리하기 위해, 유저 컴퓨터에 일시적으로 데이터를 기록해둔다. 쿠키가 호출되었을 때는 쿠키의 유효기간과 송신지 도메인, 경로 등을 체크하는 것이 가능하다.
- Set-Cookie: 상태관리 개시를 위한 쿠키 정보 (리스폰스)
    - Expire
    - Path
    - Domain
    - Secure
    - HttpOnly
- Cookie: 서버에서 수신한 쿠키 정보 (리퀘스트)

6.8. 그 이외의 헤더 필드

7장. 웹을 안전하게 지켜주는 HTTPS

7.1. HTTP의 약점

- 암호화하지 않은 평문 통신이라 도청이 가능하다.
  - TCP/IP 구조의 통신 내용은 전부 경로에서 엿볼 수 있다.
  - (SSL(Secure Socket Layer), TLS(Transport Layer Security) 등 다른 프로토콜을 조합해 통신을 암호화하거나, 통신하고 있는 콘텐츠 자체를 암호화하는 방법이 있다.
    - SSL 등을 이용해 안전한 통신로를 확립하고 난 후 그 통신로를 이용해 통신하는 HTTP를 HTTPS(HTTP Secure) 혹은 HTTP over SSL이라고 부른다.

- 통신 상대를 확인하지 않기 때문에 위장이 가능하다.
  - 리퀘스트나 리스폰스에서는 통신 상대를 확인하지 않으며 누구나 리퀘스트할 수 있다.
  - SSL는 상대를 확인하는 수단으로 증명서를 제공하는데, 증명서를 확인하여 통신 상대를 판단할 수 있다. 제 3자가 제공해주어야 한다.

- 정보의 정확성을 증명할 수 없기 때문에 변조가 가능하다.
    - 중간에 변조되었더라도 상대가 수신하기 전까지는 그 사실을 알 수가 없다. (중간자 공격)
    - MD5나 SHA-1 등 해시값을 확인하거나 파일의 디지털 서명을 확인하는 방법이 있을 수 있다. 다만 확실히 확인할 수 있는 것은 아니다.
- 확실한 방어를 위해서는 **HTTPS**를 사용하는 것이 최선이다!

7.2. HTTP + 암호화 + 인증 + 완전성 보호 = HTTPS

7.2.1. HTTP 암호화와 인증과 완전성 보호를 더한 HTTPS

HTTP Secure(HTTPS): HTTP에 암호화나 증명서 인증 등의 구조를 더한 것을 의미한다.

7.2.2. HTTPS는 SSL을 껍질을 덮어쓴 HTTP

보통 HTTP는 TCP와 직접 통신하지만, SSL을 사용한 경우에는 SSL이 그 중간 단계에 들어간다. 즉, HTTP 통신을 하는 소켓 부분을 SSL 혹은 TLS라는 프로토콜로 대체한 것이다. 새로운 애플리케이션 계층(가장 바깥쪽)에서의 프로토콜이 아니며, 또한 HTTP와는 독립된 프로토콜이다.

7.2.3. 상호간에 키를 교환하는 공개키 암호화 방식

현대 암호는 알고리즘이 공개되어 있고 사용자가 키를 가지고 있다. 이때 암호화와 복호화에 하나의 키를 같이 사용(공통키 혹은 대칭키 암호 방식)한다면 문제가 되는데, 누군가가 키를 손에 넣으면 암호를 해독할 수 있다는 점이다.

이를 해결하는 방법이 **공개키 암호(혹은 비대칭키 암호)**이다. 한 쌍의 키가 존재하며, 개인키로 암호화한 정보는 그 쌍이 되는 공개키로만 복호화가 되며 공개키로 암호화한 정보는 그 쌍인 개인키로만 복호화가 가능한 것을 의미한다. 정보 송신자가 상대의 공개키를 사용해 암호화를 하고, 수신자는 자신의 비밀키를 사용해 복호화를 한다.

HTTPS는 속도와 효율성을 위해 공통키 암호와 공개키 암호 방식을 둘 다 이용한다.

7.2.4. 공개키가 정확한지 아닌지를 증명하는 증명서

도중에 공개키를 공격자가 바뀌치기할 수도 있으므로, 인증기관의 증명서가 사용된다. 서버의 공개키를 인증기관에 등록하면, 인증기관은 자신의 비밀키로 공개키 증명서를 작성해 등록한다. 이후 클라이언트 요청에 따라 인증기관이 서버의 공개키 증명서가 진짜인지 확인하면, 서버의 공개키로 암호화하여 메시지를 송신기관에 등록한다. 서버는 비밀키로 메시지를 복호화한다.

인증기관의 공개키는 클라이언트에게 안전하게 전달되어야 하므로, 브라우저에서는 주요 인증기관의 공개키를 사전에 내장한 상태로 출시되는 경우가 많다.

- 상대방이 실제로 있는 기업인지를 확인하는 EV SSL 증명서가 있다. 주소창 옆에 따로 떠서 확인해볼 수 있다.
- 클라이언트 증명서도 HTTPS에서 이용이 가능한데, 서버가 통신하는 상대가 의도한 클라이언트인지를 증명할 수 있다. 다만, 증명서를 입수하고 배포하는 데 문제가 발생할 수 있다. 또 클라이언트의 현황만을 확인할 쭌 실제 유저를 확인할 수는 없다.
- SSL은 인증기관을 신용할 수 있는 전제 하에 가능한 기술인데, 자칫 인증기관을 사칭하는 경우가 발생할 수도 있다.
    - 독자적으로 구축한 '사짜' 인증기관을 자기인증기관이라 부르고, 경고창이 뜨게 된다.

7.2.5. 안전한 통신을 하는 HTTPS의 구조

1. Client to Server: 'Client Hello'
2. Server to Client: 'Server Hello'
3. Server to Client: Certificate 메시지 송신 (공개키 증명서 포함)
4. Server to Client: 'Server Hello Done'
5. Cleint to Server: 'Client Key Exchange' - 통신을 암호화하는 데 사용하는 Pre-Master secret 이 포함되어 있으며 공개키로 암호화되어 있음.
6. Client to server: 'Change cipher Spec' - 이 메시지 이후의 통신은 암호키를 사용해 진행한다는 의미.
7. Client to Server: 'Finished'
8. Server to Client: 'Change cipher Spec'
9. Server to Client: 'Finished'
10. SSL에 의해 접속이 확립되었고 통신 또한 보호됨.
11. HTTP 리퀘스트, 리스폰스 송수신

- MAC(Message Authentication Code): 메시지 다이제스트. 변조를 감지할 수 있어 완전성 보호 가능.
- TLS: SSL을 바탕으로 한 프로토콜이며, 총칭하여 SSL이라 부르기도 한다.
- SSL을 이용하면 통신속도가 떨어지고 리소스를 다량으로 소비하여 처리가 느려지게 된다. 또한 암호화 처리 때문에 부하가 발생한다. 근본적인 해결방법은 없고 SSL 엑셀러레이터라는 하드웨어를 사용하기도 한다.
    - 따라서 모든 페이지가 HTTPS를 사용하는 것은 아니다.