> HTTPS의 자세한 프로토콜을 복습한다.

✓ SSL/TLS 보안

saseungmin · 2020년 5월 28일

애플리케이션 보안 웹 보안 정보보안

정보 보안

목록 보기 9/9

📌 SSL/TLS 개요

1. 기본 개념

- 클라이언트/서버 환경에서 TCP 기반의 Application에 대한 종단간(End-TO-End) 보안서비스를 제공하기 위해 만들어진 **전송계층 보안 프로토콜**이다.
- SSL/TLS에서는 **대칭키 암호, 공개키 암호, 일방향 해시함수, 메시지 인증코드, 의사난수 생서기, 전자서명**을 조합해서 안전한 통신을 수행한다.
- SSL/TLS는 **암호 스위트를 변경**해서강력한 알고리즘을 사용할 수 있다.

2. SSL/TLS상의 HTTP

- 통신내용을 암호화해주는 프로토콜로 SSL 혹은 TLS를 이용한다. 그리고 SSL/TLS상에 HTTP를 올린다.
- 이것을 프로토콜 이중 구조라고 하고 이로 인해 HTTP의 통신은 암호화되어 도청을 방지할 수 있고, 통신을 수행할 때 URL은 https:// 로 시작한다.

3. SSL/TLS 보안 서비스

- 기밀성 서비스: DES, RC4와 같은 대칭키 암호화 알고리즘을 사용하고 이때 사용되는 비밀키는 **Handshake Protocol**을 통해 생성된다.
- 클라이언트와 서버 상호 인증: **인증**에는 RSA와 같은 비대칭키 암호 알고리즘, DSS와 같은 **전자서명** 알고리즘과 X.509 **공개키 인증서**가 사용된다.
- 메시지 무결성 서비스: 해시 알고리즘을 사용해서 메시지 인증코드를 만들어 메시지에 포함시키기 때문에 **신뢰성** 있는 통신이 가능하다.

4. 암호 스위트(cipher suite)

- SSL/TLS에서 사용하는 대칭키 암호, 공개키 암호, 전자서명, 일방향 해시함수 등 사용하고 있던 **암호 기술에 결함이 발견되면 부품과 같이 교환할 수 있다.**
- 하지만 클라이언트와 서버가 같은 암호 기술을 사용해야 한다.

5. SSL과 TLS의 차이

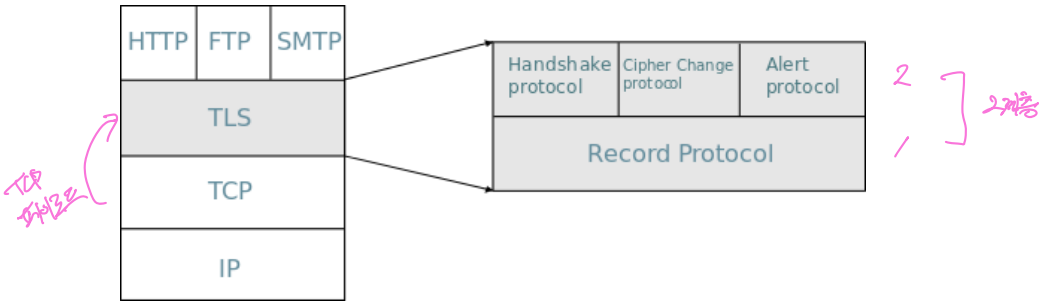
- SSL은 1995년 Version 3.0이 되었는데, 2014년 SSL 3.0 사양에 **POODLE 공격**이 가능한 취약점(CVE 2014-3566)이 발견되었기 때문에 SSL 3.0은 안전하지 않다.

> - **POODLE 공격 ❓** 프로토콜 버전을 다운그레이드 시키는 공격이다.
> - **CVE ❓** 보안 취약점과 기타 정보 보안 노출 사항을 기록한 규격화된 목록으로 발견년도와 일련번호를 포함한다.

- TLS는 SSL 3.0을 기초로 해서 IETF가 만든 프로토콜이다. (TLS 1.0 == SSL3.1)
- TLS 1.1에는 대칭 암호 알고리즘으로서 AES가 추가되었고, TLS 1.2에는 인증 암호로 GCM과 CCM을 사용할 수 있게 되었고, HMAC-SHA 256이 추가되었고 IDEA와 AES가 삭제 되었다.

📌 TLS(Transport Layer Security) 프로토콜

1. TLS 구조

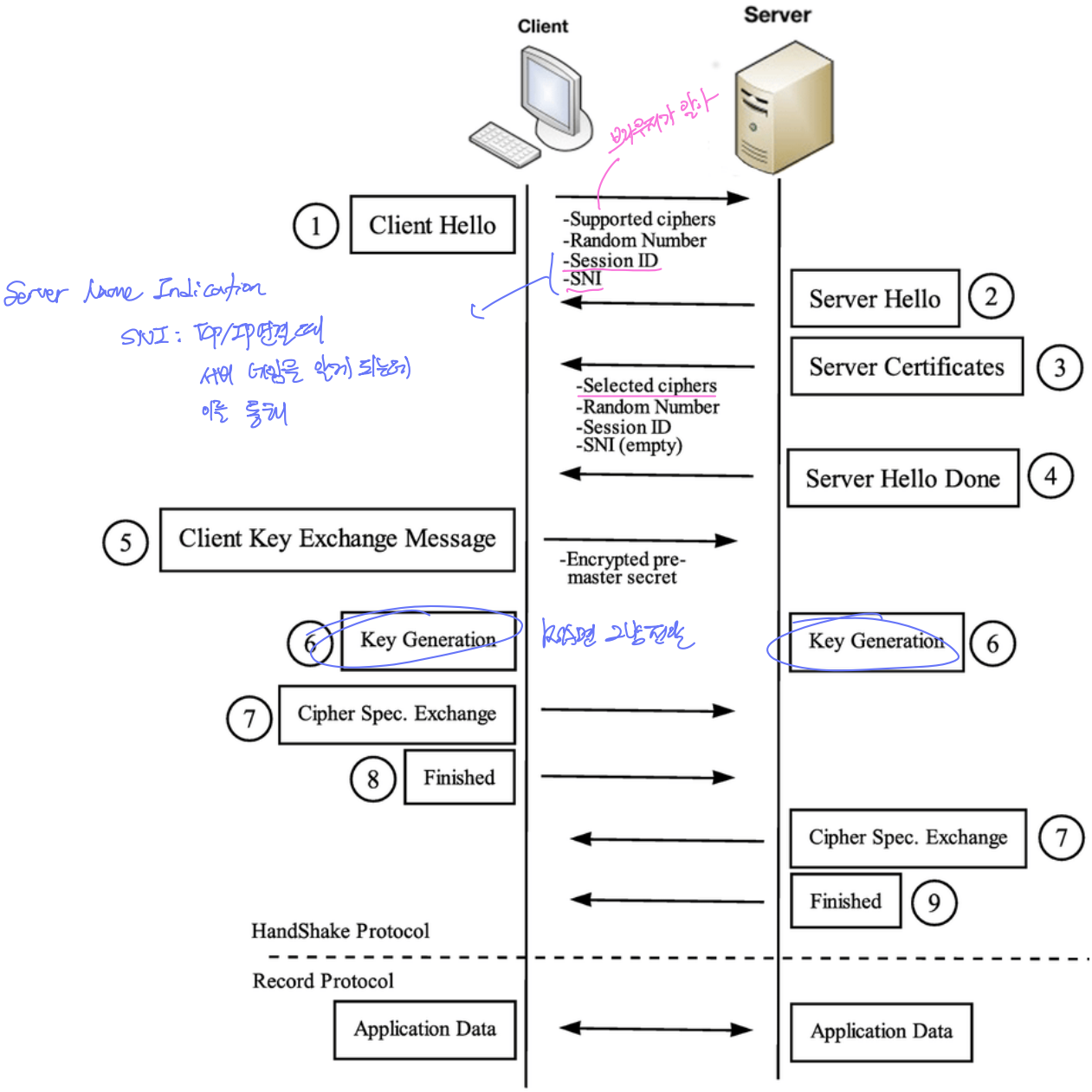


> - TLS는 단일 프로토콜이 아니고 **2계층에 걸친 프로토콜이다.**
> - **Record 프로토콜**은 운반자이며, 응용 계층으로부터 오는 데이터뿐만 아니라 TLS의 상위 프로토콜로부터 오는 메시지를 전송한다. Record 프로토콜에서 오는 메시지는 보통 TCP인 전송 계층의 페이로드이다.
> - **Handshake 프로토콜**은 Record 프로토콜에 대한 보안 매개변수를 제공한다.
>   - 암호 집합을 설정하고 키와 보안 매개변수를 제공한다
>   - 또한, 필요하다면 클라이언트가 서버에 대해 그리고 서버가 클라이언트에 대해 인증된다.
> - **ChangeCipherSpec 프로토콜**은 암호학적 비밀을 신속하게 보내는 데 사용된다.
> - **Alert 프로토콜**은 비정상 조건을 알리는데 사용된다.
> - **Heartbeat 프로토콜**은 프로토콜 객체의 가용성을 모니터링 할 때 사용된다.

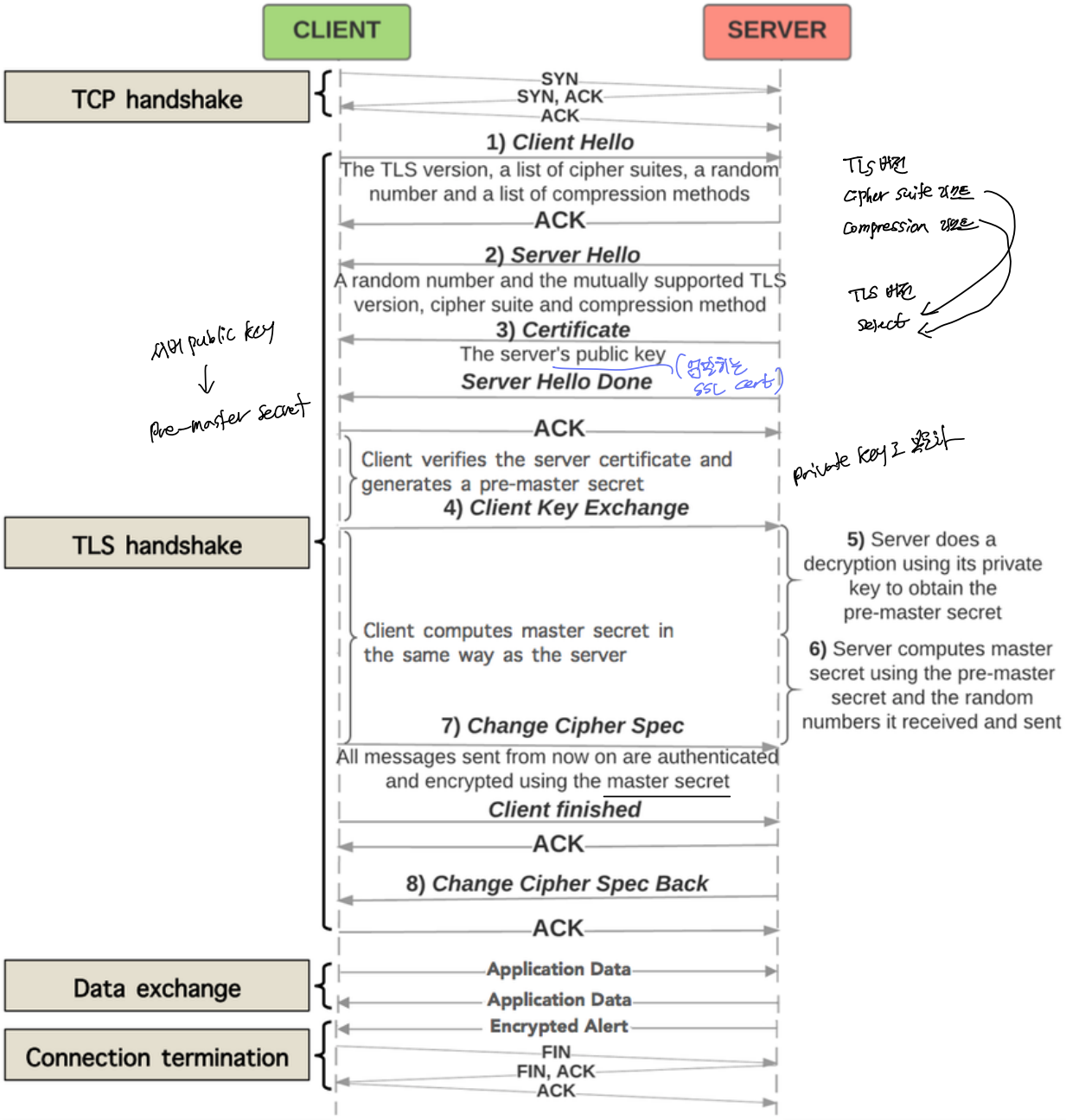
2. Handshake 프로토콜

- 서버와 클라이언트가 서로 인증하고 암호화 MAC 알고리즘 그리고 TLS 레코드 안에 보낸 데이터를 보호하는 데 사용할 암호키를 협상할 수 있다.
- Handshake 프로토콜은 모든 응용 데이터를 전송하기 이전에 사용한다. 각 메시지는 다음과 같은 3개의 필드로 구성된다

> - 유형(type)1바이트 : 10개 메시지 중 하나를 나타낸다.
> - 길이 3바이트 : 메시지의 길이를 바이트로 나타낸다.
> - 내용 : 이 메시지와 연관된 매개변수이다.



📌 이미지 출처 : https://www.researchgate.net/figure/TLS-handshake-protocol_fig1_298065605


CLIENT
SERVER
TCP handshake
SYN
SYN, ACK
ACK
1) Client Hello
The TLS version, a list of cipher suites, a random number and a list of compression methods
ACK
2) Server Hello
A random number and the mutually supported TLS version, cipher suite and compression method
3) Certificate
The server's public key
Server Hello Done
ACK
Client verifies the server certificate and generates a pre-master secret
4) Client Key Exchange
TLS handshake
5) Server does a decryption using its private key to obtain the pre-master secret
Client computes master secret in the same way as the server
6) Server computes master secret using the pre-master secret and the random numbers it received and sent
7) Change Cipher Spec
All messages sent from now on are authenticated and encrypted using the master secret
Client finished
ACK
8) Change Cipher Spec Back
ACK
Data exchange
Application Data
Application Data
Encrypted Alert
Connection termination
FIN
FIN, ACK
ACK

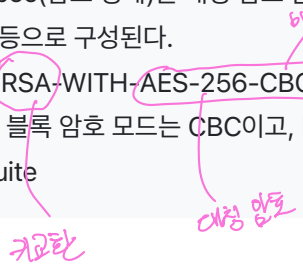
🌈 단계별 세부 내용

1. HelloRequest

> - 서버가 아무 때나 보낼 수 있는 메시지로 클라이언트에게 Client Hello 메시지를 보내어 협상을 시작하자고 요구하는 메시지이며 현재 협상이 진행 중이면 클라이언트는 이 메시지를 무시한다.
> - 서버는 HelloRequest를 보내고 이에 대한 Handshake가 끝나지 않은 상태에서 또 다시 HelloRequest를 보내면 안된다.

2. ClientHello

> - 클라이언트는 서버에 처음으로 연결을 시작하거나 HelloRequest 메시지에 대한 응답으로 ClientHello 메시지를 보낸다.
> - ClientHello 메시지는 세션 식별자, CipherSuite 리스트, 클라이언트가 지원하는 압축 알고리즘 리스트, 클라이언트 SSL 버전, 클라이언트가 생성한 난수를 서버에 전달한다.
>   - cipher_suites : CipherSuite 리스트는 키 교환 알고리즘, 암호 알고리즘, MAC 알고리즘을 포함한다.
>   - 형식은 SSL/TLS-(A)-WITH-(B) : A는 키 교환 및 인증 알고리즘, B는 cipher spec
>   - cipher spec(암호 명세)는 대칭 암호 알고리즘, 암호키 길이, 블럭 암호 모드, HMAC용 해시 알고리즘 등으로 구성된다.
>   - ex) TLS-RSA-WITH-AES-256-CBC-SHA256 : AES를 사용하며 암호키 길이는 256비트, 블록 암호 모드는 CBC이고, HMAC용 해시 알고리즘은 SHA-256을 사용하는 cipher suite



3. ServerHello

> - 서버는 ClientHello 메시지에 대한 응답으로 ServerHello 메시지를 보낸다. 실패 할 경우에는 HandShake Failure Alert 메시지를 띄운다.
> - ServerHello 메시지는 세션 식별자, 선택한 CipherSuite, 선택한 압축방법, 서버 SSL 버전, 서버가 생성한 난수를 클라이언트에 전달한다.

🔶 1 단계 : 보안 기능 확립 후에 클라이언트와 서버는 다음 내용을 알게 된다.

- SSL 버전
- 키 교환, 메시지 인증과 암호화를 위한 알고리즘
- 압축 방법
- 키 생성을 위한 2개의 난수

4. Server Certificate

- 서버의 인증이 필요한 경우에 서버는 ServerHello 뒤에 서버의 인증서가 포함된 Certificate을 보내야 한다.
- 이 때 서버의 인증서는 선택된 cipher suite의 키 교환 알고리즘에 맞는 타입이어야 한다.

5. Server Key Exchange

- ServerKeyExchange는 서버의 인증서를 보내지 않았거나, 보낸 인증서에 키 교환에 필요한 정보가 부족하면 전송되는 메시지이다.

6. Certificate Request

- 서버는 자신을 클라이언트에게 인증함과 동시에 클라이언트에게 클라이언트의 인증서를 통한 인증을 요구할 수 있다.
- 이 메시지는 선택적으로 사용되고 사용할 시엔 서버와 클라이언트의 상호 인증이 이루어진다.

7. ServerHelloDone

- 서버가 보낼 메시지를 다 보냈음을 알려주는 메시지이다.

🔶 2 단계 : 서버 인증과 키 교환 후에 서버는 클라이언트에 대해 인증되고 클라이언트는 필요하다면 서버의 공개키를 알 수 있다.

8. Client Certificate

- 서버가 클라이언트의 인증을 요구할 경우 클라이언트가 보내는 메시지이다.

9. Client Key Exchange

- 클라이언트는 세션키를 생성하기 위해 임의의 비밀 정보인 48바이트 pre-master-secret을 생성한다.
- 그 후 선택된 알고리즘(RSA, Fortezza, Diffi-Hellman 중 하나)을 이용하여 pre-master-secret을 서버와 공유하게 된다.

10. Certificate Verify

- 클라이언트 인증서의 명백한 확인을 위해서 클라이언트는 핸드셰이크 메시지를 전자 서명하여 전송한다.

🔶 3 단계 : 클라이언트 인증과 키 교환 후에 클라이언트는 서버에 대해 인증되고 클라이언트와 서버 양측은 사전 마스터 비밀을 알게 된다.

10. ChangeCipherSpec, Finished

> - ChangeCipherSpec 메시지는 핸드셰이크 프로토콜에 포함되는 것은 아니고, 이 메시지 이후에 전송되는 메시지는 새롭게 협상된 알고리즘과 키를 이용할 것임을 나타낸다.
> - Finished 메시지는 ChangeCipherSpec 메시지 이후에 전송되며, 협상된 알고리즘과 키가 처음으로 적용되고 상대편에서 이 메시지를 통해서 협상한 결과를 확인하게 된다.
> - 이후 애플리케이션 데이터가 전송된다.

🔶 4 단계 : 종료 단계 이후 클라이언트와 서버는 데이터를 교환할 준비가 된다.

3. Record 프로토콜

- 레코드 프로토콜은 신뢰하는 전송 서비스를 필요하므로 TCP 프로토콜을 사용한다.
- 기밀성을 제공 : 핸드셰이크 프로토콜은 TLS 페이로드를 관용 암호화 하는 쓸 공유 비밀키를 정의
- 메시지 무결성 제공 : MAC을 생성하는데 사용할 공유 비밀키를 정의한다.
- 레코드 프로토콜은 전송할 응용 메시지를 다룰 수 있는 크기의 블록으로 잘라내어 단편화한다.
- 옵션으로 데이터를 압축하고, MAC을 적용하고, 암호화 하고, 헤더를 추가하고 그 결과를 단편으로 전송한다.
- 수신된 데이터는 복호화 하고, MAC을 확인하고, 압축을 풀고 재조립하여 상위 계층 사용자에게 전달한다.

4. ChangeCipherSpec 프로토콜

- ChangeCipherSpec 메시지는 바로 직전에 협상된 CipherSpec과 키에 의하여 보호될 후속 레코드를 상대에게 알리기 위하여 클라이언트 또는 서버에 의해 전송된다.
- 종단 간에 협상된 보안 파라미터를 이후부터 적용/변경함을 알리기 위해 사용하는 프로토콜이다.
- ChangeCipherSpec 프로토콜은 한 바이트로 구성되고 값 1을 갖는 한 개의 메시지로 구성된다.

5. Alert 프로토콜

- Alert 프로토콜은 SSL/TLS 통신 과정에서 발생하는 오류를 통보하기 위해 경고 할 때 사용한다.
- 이 프로토콜 안의 각 메시지는 2바이트로 구성된다.

6. HeartBeat 프로토콜

- HeartBeat 프로토콜은 프로토콜 개체의 가용성을 모니터링 할 때 사용하는 프로토콜이다.

📌 SSL/TLS 공격

1. OpenSSL의 HeatBleed 취약점(2014년 4월)

- HeatBleed 취약점은 TLS의 하트비트 확장이라고 하는 기능에 **요구 데이터 길이에 대한 점검이 결여**되었기 때문에 메모리 상의 정보까지 상대방에게 넘어가 버리는 것이다.
- 대응책은 HeatBleed 취약점 대책이 실행된 버전으로 OpneSSL을 갱신하거나 하트비트 확장을 사용하지 않는 옵션을 부착해 재컴파일하는 등 조치가 필요하다.
- 또한 인증서를 재발급 받고, 취약점 조치가 완료된 후 사용자들의 비밀번호를 재설정한다.

2. SSL 3.0의 취약점과 POODLE 공격(2014년 10월)

- TLS를 사용하고 있다고 해도 SSL 3.0으로 다운그레이드 당하여 POODLE 공격을 받을 취약점이 있다.
- POODLE 공격을 통해 SSL 3.0을 사용하도록 강제한 후 MITM(중간자) 공격을 통해 암호화되어 송수신되는 쿠키정보나 데이터를 추출하는 공격이 가능하다.
- 블록 암호화 기법인 CBC 모드를 사용하는 경우 발생하는 패딩된 암호블록이 MAC에 의해 보호되지 않는 취약점을 이용한다.

3. FREAK 공격과 암호 수출 규제 (2015년 2월)

- SSL을 통해 강제로 취약한 RSA로 다운 그레이드 시킬 수 있는 취약점이 있다.
- SSL/TLS 서버에 대한 RSA Export Suites라고 불리는 약한 암호 스위트를 사용하게 하는 공격으로 MITM 공격을통해 RSA를 다운 그레이드 시켜서 정보를 유출시킨다.
- 대응책으로는 OpenSSL 버전을 최신버전으로 업그레이드 하고 OS및 브라우저를 업그레이드 시킨다.

4. 완전 순방향 비밀성(PFS, Perfect Forward Secrecy)

- 서버 공개키가 개인키를 이용하여 키 교환을 수행할 경우 공격자는 MITM 공격을 통해 트래픽을 가로채고 서버 개인키를 이용해 세션키/비밀키 및 송수신 데이터를 복호화할 수 있다.
- 희생자는 유출된 서버 인증서를 폐기해도(CRL, OCSP)유출된 서버 개인키로 보호되는 이전 트래픽 정보를 공격자가 보관하고 있다면 모두 복호화되는 문제점이다.
- 위 문제를 해결하기 위해 PFS라고 한다.
- 서버 개인키가 노출되어도 이전 트래픽 정보의 기밀성은 그대로 유지되는 암호학적 성질을 말한다.

SeungMin_Sa

HTTPS 통신과정 쉽게 이해하기 #3(SSL Handshake, 협상)

[도움되는 네트워크 엔지니어 환영]

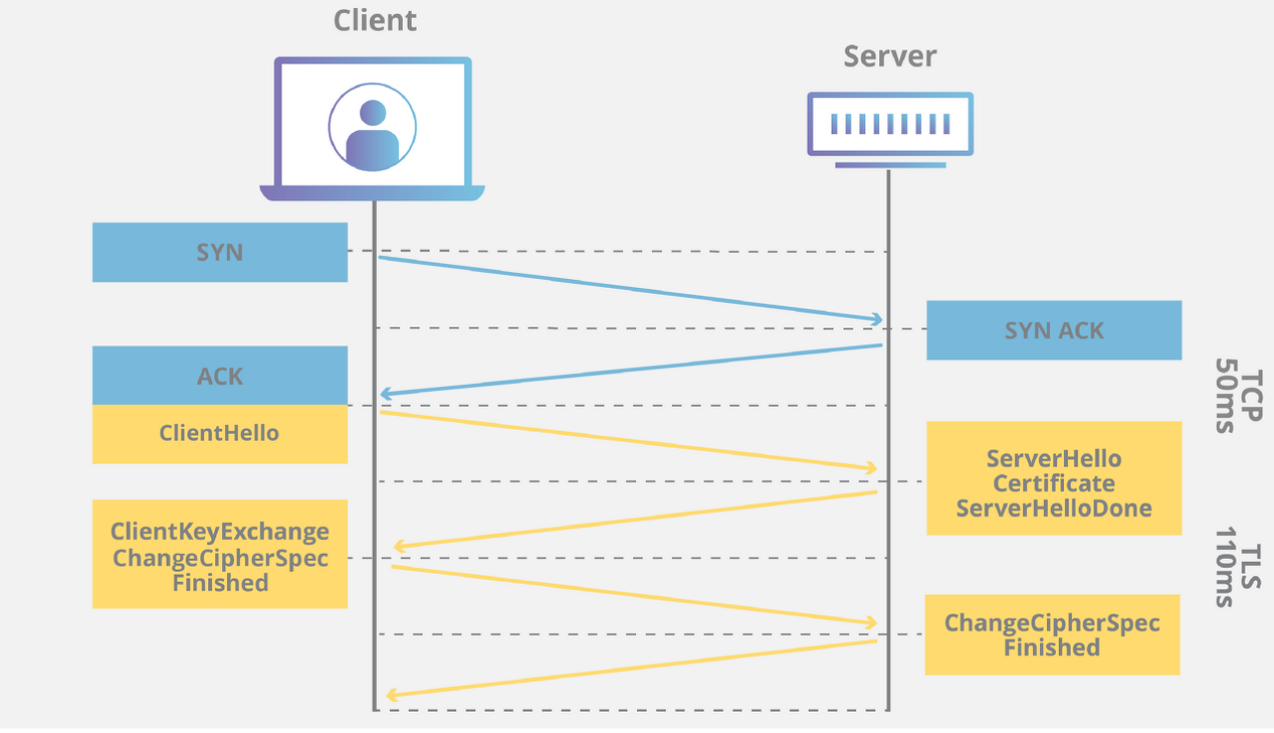
고대 그리스에서는 타인에게 노출되어서는 안 될 중요한 정보를 보낼 때, 전달하는 이(사자)의 머리를 빡빡 깎아서 중요한 정보를 적은 후 머리가 자라서 글이 보이지 않으면 그제야 상대방에게 보냈다고 합니다. 그리하게 되면 그 사실을 모르는 사람은 정보를 볼 수 없고 사실을 아는 수신자는 다시 머리를 깎은 후 정보를 볼 수 있을 테니까요. 이 암호 기법을 **스테가노그래피**라고 합니다.

<고대 그리스의 정보 전달방법, 스테가노그래피(출처 : EBS MATH)>

그 당시로서는 정말 획기적인 방법이 아닌가 싶습니다. 누가 두피에 글을 적어서 보낼 거라고 예상을 할까요? 물론 머리가 다 자라라면 한참 걸린다는 단점이 존재하지만 들키지 않는 데는 이만한 방법이 없는 것 같습니다. 그만큼 중요한 정보를 안전하게 보내는 것이 가장 중요했기 때문이겠지요. 세 번째 그림에서 보니 **동그라미** 속 수신자는 반가운 얼굴로 사자를 맞이하네요. 그게 의미하는 것이 뭘까요? **바로 수신자와 송신자가 전달방법에 대해 이미 공유하고 있으며, 사자가 온 이유를 알고 있다는 뜻입니다.**

HTTPS 통신과정에서도 송신자와 수신자가 암호화 통신을 하기 위한 방법과 수단에 대해 공유합니다. 즉 데이터를 암호화할 대칭키(비밀키)를 타인에게 노출시키지 않고 Client가 Server에게 전송하기 위해 협상을 벌이는 것이죠. 이번에 다룰 **SSL Handshake(TLS Handshake)**가 그 방법으로, 송신자와 수신자가 암호화된 데이터를 교환하기 위한 일련의 협상과정을 의미합니다. 협상과정에는 SSL

인증서 전달, 대칭키(비밀키) 전달, 암호화 알고리즘 결정, SSL/TLS 프로토콜 결정 등이 포함됩니다.



<SSL handshake의 과정(출처 : Cloudflare)>

SSL Handshake의 과정을 그린 그림을 가져왔습니다. 여기 **파란색** 칸과 **노란색** 칸은 네트워크 상에서 전달되는 IP Packet을 표현한 것입니다. 맨 윗줄의 **SYN, SYN ACK, ACK**는 TCP layer의 3-way handshake로 HTTPS가 TCP 기반의 프로토콜이기 때문에 암호화 협상(SSL Handshake)에 앞서 연결을 생성하기 위해 실시하는 과정이고 아래 **노란색 상자의 패킷**들이 SSL Handshake입니다.

1. 암호화 알고리즘(Cipher Suite) 결정

2. 데이터를 암호화할 대칭키(비밀키) 전달

위 두 가지를 기억하면서 순서대로 보시길 권장합니다.

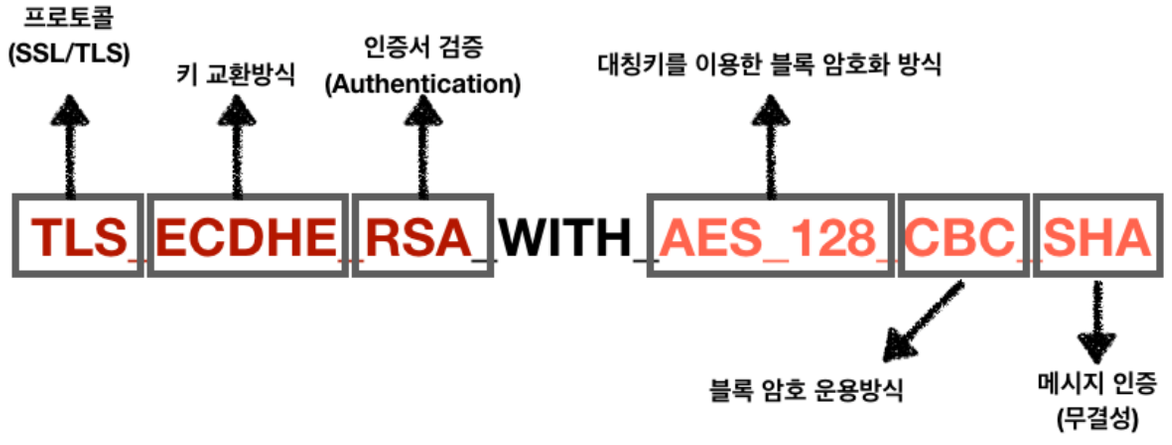
Client Hello

Client가 Server에 연결을 시도하며 전송하는 패킷입니다. 자신이 **사용 가능한 Cipher Suite 목록, Session ID, SSL Protocol Version, Random byte** 등을 전달합니다. Cipher Suite는 SSL Protocol version, 인증서 검정, 데이터 암호화

프로토콜, Hash 방식 등의 정보를 담고 있는 존재로 선택된 Cipher Suite의 알고리즘에 따라 데이터를 암호화하게 됩니다. 아래 사진을 보면 Client가 사용 가능한 Cipher Suite를 Sever에 제공하는 것을 알 수 있습니다.

```
Transport Layer Security
  TLSv1.2 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 512
    Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 508
        Version: TLS 1.2 (0x0303)
      > Random: 1396873af8d56db07f55a31afba6c98a04e00025005764fe…
        Session ID Length: 32
        Session ID: fe329526917d48c5af72228bdcb801142894fe91f4a548f7…
        Cipher Suites Length: 34
      v Cipher Suites (17 suites)
          Cipher Suite: Reserved (GREASE) (0x3a3a)
          Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
          Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
          Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
          Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
          Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
          Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)
```

<Clienthello Packet>



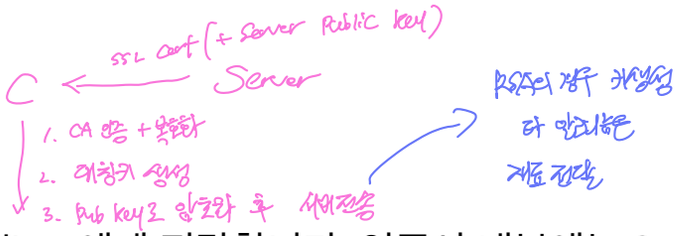
<Cipher Suite의 구성>

Server Hello

Client가 보내온 ClientHello Packet을 받아 Cipher Suite 중 하나를 선택한 다음 Client에게 이를 알립니다. 또한 자신의 SSL Protocol Version 등도 같이 보냅니다. 아래 사진을 보면 ClientHello에서 17개였던 Cipher Suite와 달리 아래에서는 Server가 선택한 한 줄(Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256)만이 존재하는 것을 알 수 있습니다. Client가 보낸 리스트에 있는 Cipher Suite 중에 한 개를 선택한 것이지요.

```
▼ TLSv1.2 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 78
  ▼ Handshake Protocol: Server Hello
      Handshake Type: Server Hello (2)
      Length: 74
      Version: TLS 1.2 (0x0303)
    ▶ Random: 3896a769b30ae8f9cd0dcd3eb1d58aa4d7a12e2c5ca8847b…
      Session ID Length: 0
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
      Compression Method: null (0)
      Extensions Length: 34
    ▶ Extension: renegotiation_info (len=1)
    ▶ Extension: server_name (len=0)
    ▶ Extension: ec_point_formats (len=4)
    ▶ Extension: session_ticket (len=0)
    ▶ Extension: application_layer_protocol_negotiation (len=5)
    ▶ Extension: extended_master_secret (len=0)
```

<ServerHello Packet>

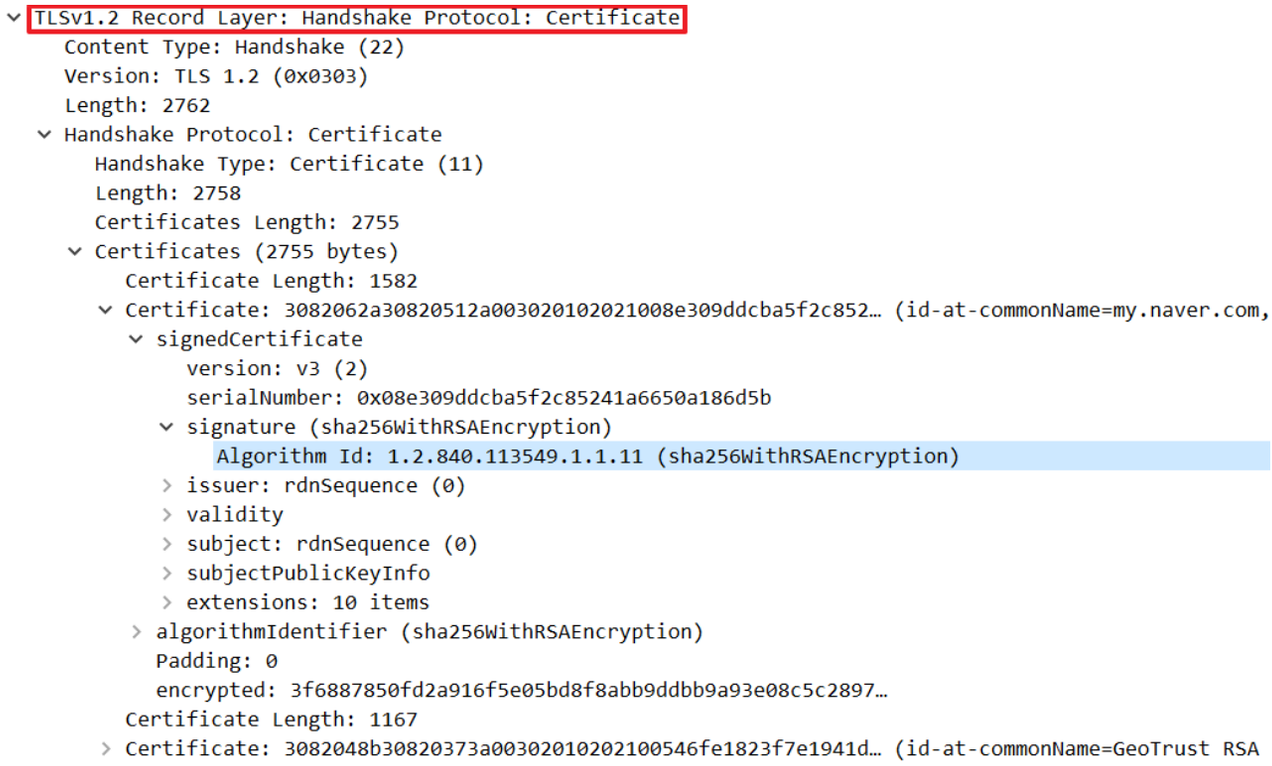


Certificate

Server가 자신의 SSL 인증서를 Client에게 전달합니다. 인증서 내부에는 Server가 발행한 **공개키(+개인키는 Server가 소유)**가 들어있습니다. Client는 Server가 보낸 **CA(Certificate Authority, 인증기관)의 개인키**로 암호화된 이 SSL 인증서를 이미 모두에게 공개된 **CA(Certificate Authority, 인증기관)의 공개키**를 사용하여 복호화합니다. 복호화에 성공하면 이 인증서는 CA(Certificate

Authority, 인증기관)가 서명한 것이 맞는 것이니 진짜임이 증명되는 것이죠. 즉, 인증서를 검증하는 것입니다.

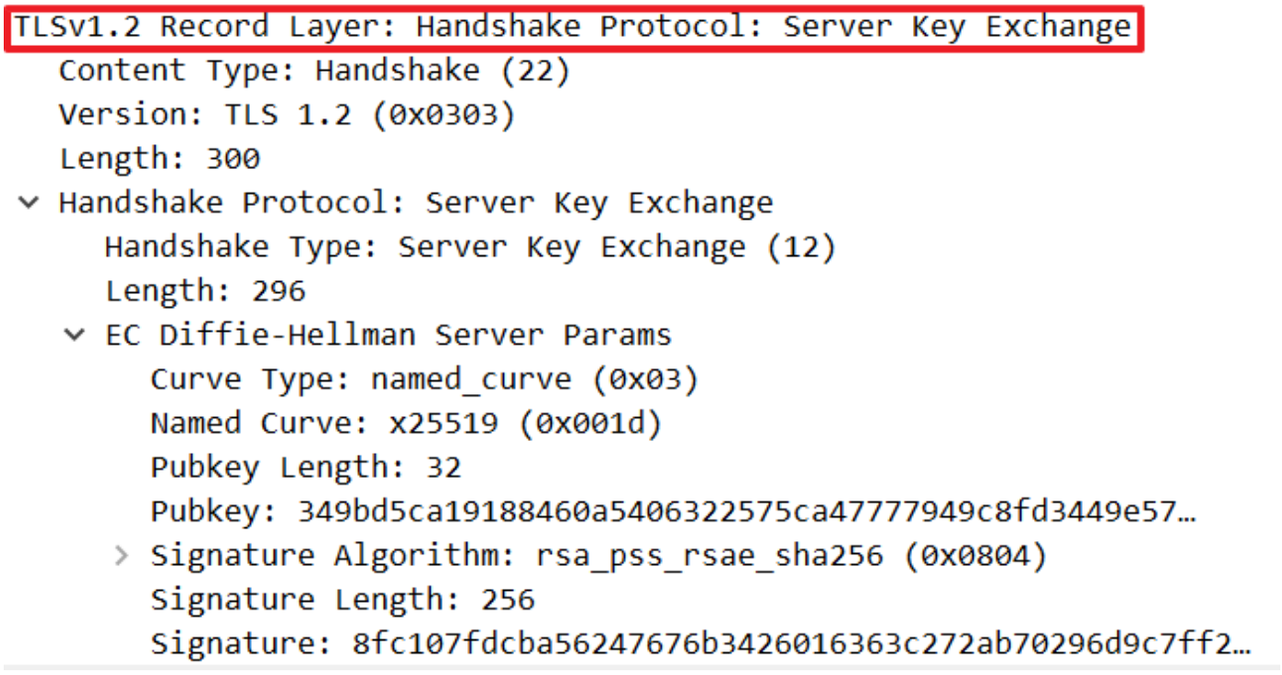
또한 Client는 데이터 암호화에 사용할 **대칭키(비밀키)**를 생성한 후 SSL 인증서 내부에 들어 있던 (Server가 발행한) 공개키를 이용해 암호화하여 Server에게 전송할 것입니다. 아래를 보면 SSL 인증서가 무슨 알고리즘(RSA)으로 암호화되었고, 무슨 Hash 알고리즘(SHA256)으로 서명되었는지 확인 가능합니다.



<Certificate Pacekt>

Server Key Exchange / ServerHello Done

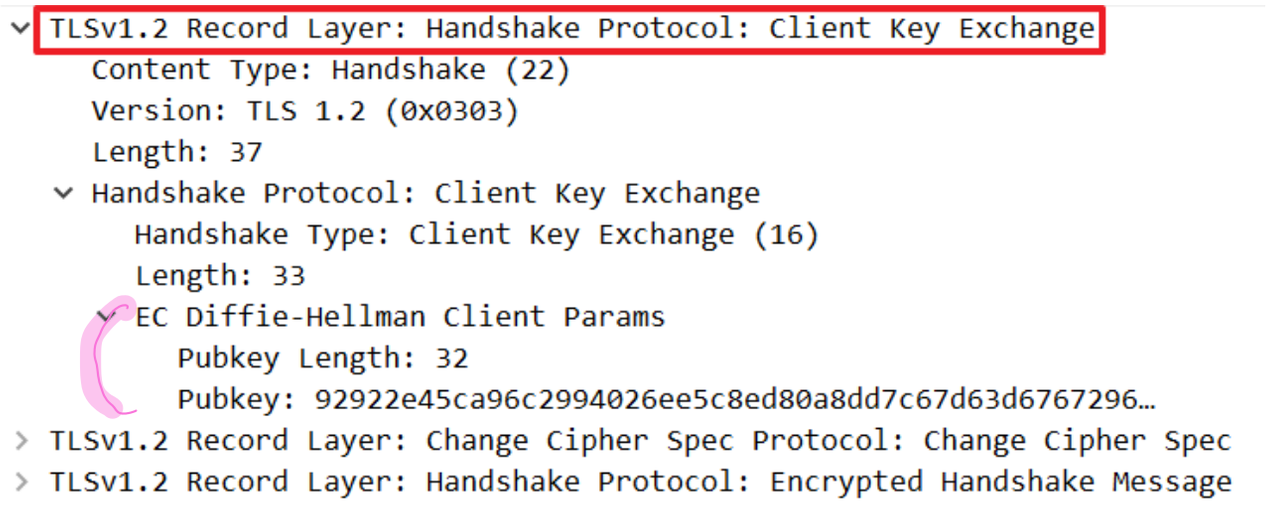
'Server Key Exchange'는 Server의 **공개키**가 SSL 인증서 내부에 없는 경우, Server가 직접 전달함을 의미합니다. **공개키**가 SSL 인증서 내부에 있을 경우 Server Key Excahnge는 생략됩니다. 인증서 내부에 공개키가 있다면 Client가 CA(Certificate Authority, 인증기관)의 공개키를 통해 인증서를 복호화한 후 Server의 공개키를 확보할 수 있습니다. 그리고 Server가 행동을 마쳤음을 전달합니다.



<Server Key Excahnge>

Client Key Exchange

대칭키(비밀키, 데이터를 실제로 암호화하는 키)를 Client가 생성하여 SSL 인증서 내부에서 추출한 Server의 공개키를 이용해 암호화한 후 Server에게 전달합니다. 여기서 전달된 '대칭키'가 바로 **SSL Handshake의 목적**이자 가장 중요한 수단인 데이터를 실제로 암호화할 **대칭키(비밀키)**입니다. 이제 키를 통해 Client와 Server가 교환하고자 하는 데이터를 암호화합니다.



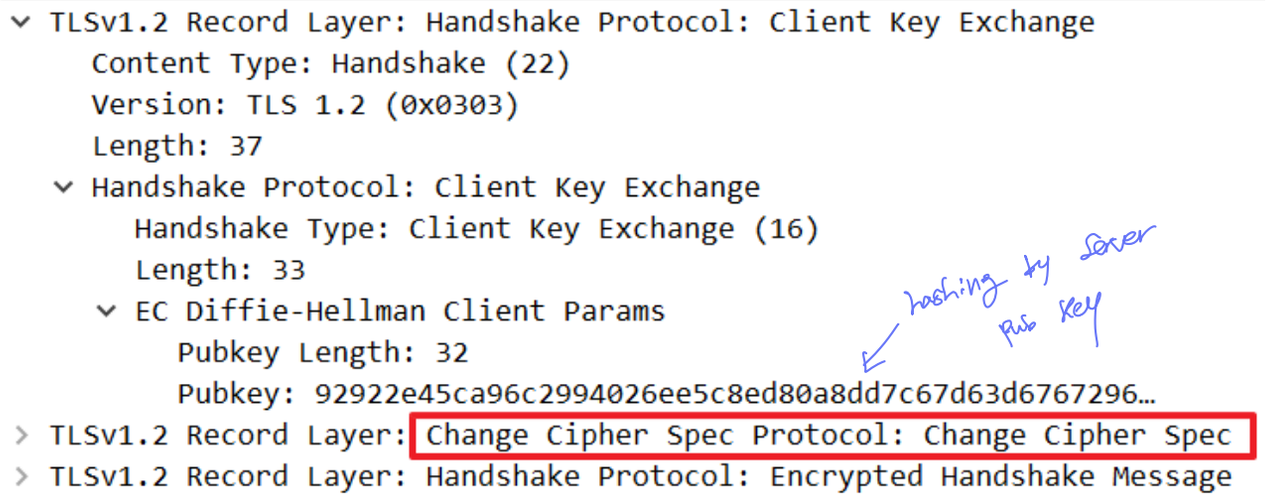
<Client Key Exchange>

위의 캡쳐화면을 보고 의아한 생각이 드실겁니다. Client가 데이터를 암호화할 대칭키(비밀키)를 보낸다고 했는데 'EC Diffie-Hellman Client Params', 즉 키를

생성할 재료를 보낸다는 것으로 보이네요. 키교환 알고리즘을 RSA가 아닌 DH, DHE 알고리즘과 타원곡선 암호인 ECDHE을 사용하게 된다면 Client가 데이터를 암호화할 대칭키(비밀키)를 보내는 것이 아니라 대칭키(비밀키)를 생성할 재료를 Client와 Server가 교환하게 됩니다. 그리고 서로 교환한 각자의 재료를 합쳐 동일한 대칭키를 만들 수 있답니다. DH, DHE 알고리즘과 타원곡선 암호인 ECDHE의 특징입니다. RSA 키교환 알고리즘을 사용하게 되면 Client가 대칭키(비밀키)를 생성하여 인증서 내부에 들어 있던 서버의 공개키로 암호화 한 후 전달하게 됩니다.

ChangeCipherSpec / Finished

Client, Server 모두가 서로에게 보내는 Packet으로 교환할 정보를 모두 교환한 뒤 통신할 준비가 다 되었음을 알리는 패킷입니다. 그리고 'Finished' Packet을 보내어 SSL Handshake를 종료하게 됩니다.



<Change Cipher Spec>

여기까지가 SSL Handshake입니다. 요약해보면 ClientHello(암호화 알고리즘 나열 및 전달) - > Serverhello(암호화 알고리즘 선택) - > Server Certificate(인증서 전달) - > Client Key Exchange(데이터를 암호화할 대칭키 전달) - > Client / ServerHello done (정보 전달 완료) - > Fisnied (SSL Handshake 종료) 이정도로 나열할 수 있습니다. 과정이 끝나면 Client와 Server는 데이터를 암호화할 동일한 대칭키(비밀키)를 갖게 되며, 서로에게 각자 갖고 있는 동일한 대칭키를 통해 데이터를 암호화하여 전송하거나 데이터를 복호화합니다. Client와 Server는 이제 성공적으로 비밀친구가 되었습니다.

여기까지가 SSL Handshake의 과정에 대한 전반적인 내용입니다. 사실 RSA나 DHE와 같은 알고리즘에 대한 언급이 없다보니 약간 나사가 빠진 기분이 드네요. 그래서 다음 편에 **Cipher Suite**를 다룰 예정인데요. Cipher Suite를 이해하시고 나면 좀더 쉽게 느껴지지 않을까 생각합니다. 감사합니다.