DF&C Lab.
Digital Forensic & Cryptanalysis
KOOKMIN UNIVERSITY

2023 2학기 보안프로토콜

팀 프로젝트

국민대학교 금융정보보안학과, DF&C 연구실
https://dfnc.kookmin.ac.kr

조교 : 김기윤 (gi0412@kookmin.ac.kr)
오픈톡 : https://open.kakao.com/o/gnMXxOHf

보안프로토콜

Contents

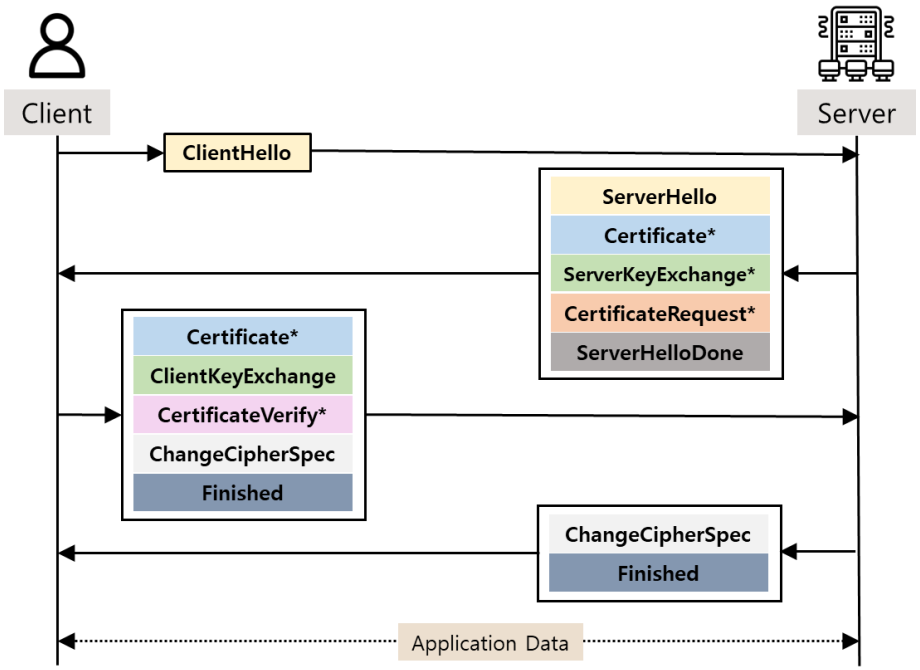
01

MINI TLS

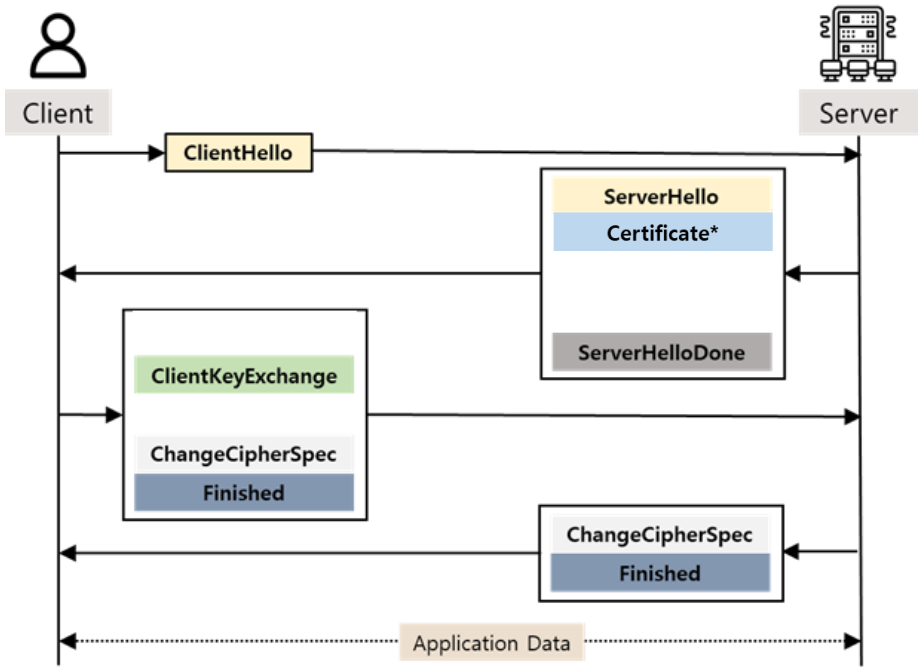
보안프로토콜

보안프로토콜을 활용한 ECHO 클라이언트 구현

- TLS 1.2



- MINI TLS (Ours)



유사하지만 최대한 쉽게 구현 예정

보안프로토콜을 활용한 ECHO 클라이언트 구현

- 주고받는 데이터의 구조

  - 기존 :

Protocol (1바이트)	Version (2바이트)	Length (2바이트)	Protocol Message (n바이트)	MAC (옵션)

  - 변경 :

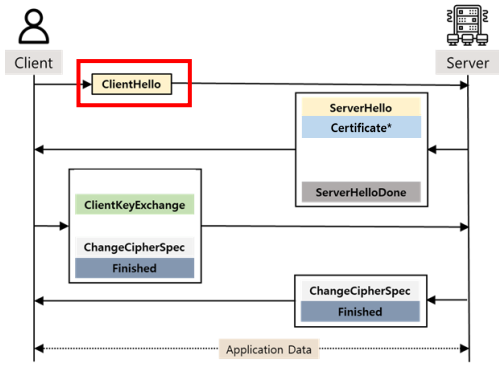
Protocol (1바이트)	Length (4바이트)	Protocol Message (n바이트)	MAC (핸드셰이크 이후)

- Protocol

: 0x00 – 'ClientHello'
: 0x01 – 'ServerHello'
: 0x02 – 'Certificate'
: 0x03 – 'ServerHelloDone' (1바이트 데이터 0x01 전송)
: 0x04 – 'ClientKeyExchange'
: 0x05 – 'ChangeCipherSpec' (1바이트 데이터 0x01 전송)
: 0x06 – 'Finished' (1바이트 데이터 0x01 전송)
: 0xFF – Error code

(핸드셰이크 이후 사용 가능)
: 0xFE – 'ECHO Mode'
: 0xFD – 'Data Decryption Mode'

보안프로토콜을 활용한 ECHO 클라이언트 구현



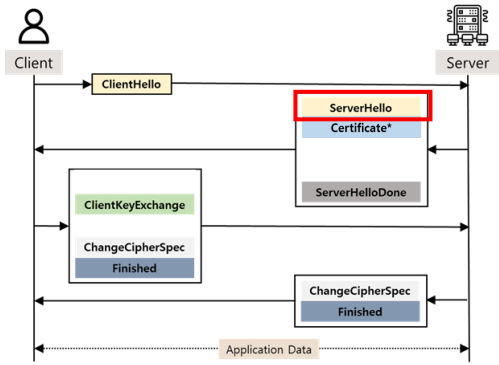
- 통신 1번 (Client -> Server)

Tag:ClientHello

: 클라이언트가 서버로 보내는 핸드셰이크 시작 메시지

- 담겨있는 메시지
  - ~~client_version: 클라이언트가 지원하는 TLS 버전~~
  - client_random: ~~타임 스템프 4바이트 + 28바이트 난수~~ 32바이트 난수 사용
  - ~~session_id: 이전에 설정된 세션이 있는 경우 이를 재사용 가능한지 질의 하기 위한 값~~
  - ~~cipher_suites: 클라이언트가 지원하는 Ciphersuite 목록 AES-CBC-PKCS#7, HMAC-SHA 만 사용 예정~~
  - ~~compression_methods: 클라이언트가 지원하는 압축 알고리즘 목록~~

보안프로토콜을 활용한 ECHO 클라이언트 구현



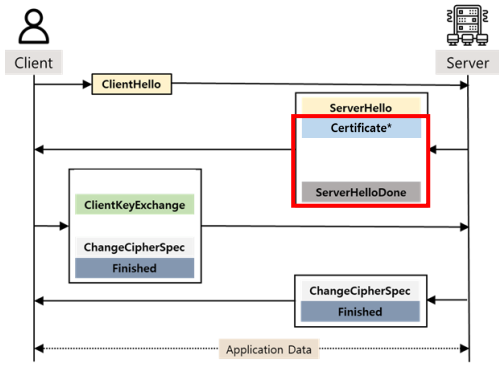
- 통신 2번 (Server -> Client)

Tag:ServerHello

: 서버가 클라이언트에게 보내는 핸드셰이크 시작 메시지

- 담겨있는 메시지
  - ~~server_version: 양측이 지원하는 최신 TLS 버전~~
  - server_random: ~~타임 스템프 4바이트 + 28바이트 난수~~ 32바이트 난수 사용
  - ~~session_id: 새로운 세션 사용 또는 이전 세션 사용 동의 용도~~
  - ~~cipher_suites: 양측이 지원하는 ciphersuite 사용 및 통보(공통 암호가 없는 경우 핸드셰이크 실패)~~
  - ~~compression_methods: 양측이 지원하는 압축 알고리즘 목록~~

보안프로토콜을 활용한 ECHO 클라이언트 구현



- 통신 3번 (Server -> Client)

Tag:Certificate

: 서버가 클라이언트에게 인증서를 보냄

- 담겨있는 메시지 내용
  - ~~인증서의 상세 정보를 전송~~
  - Base64로 인코딩 된 공개키

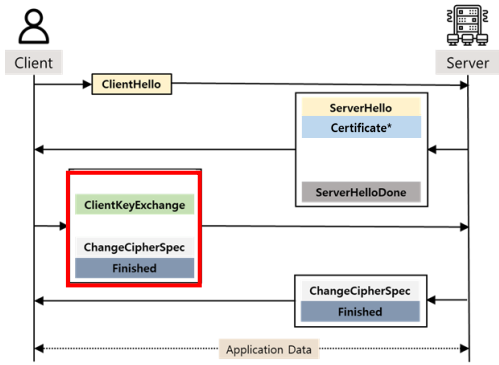
- 통신 4번 (Server -> Client)

Tag:ServerHelloDone

: ServerHello 절차가 종료됨을 알림

- 담겨있는 메시지 내용
  - 1바이트 데이터 0x01

보안프로토콜을 활용한 ECHO 클라이언트 구현



- 통신 5번 (Client -> Server)

Tag:ClientKeyExchange

: 서버의 인증서로 암호화한 난수 전송

- 담겨있는 메시지 내용
  - PreMasterSecret(Client가 생성한 난수)를 Server의 공개키로 암호화 하여 전송
  - ~~공개 키 또는 NULL값을 송신하여 서버측에서 PreMasterSecret을 생성하게 함~~

- 통신 6번 (Client -> Server)

Tag:ChangeCipherSpec

: 이제 설정한 암호로 통신하겠다고 알림

- 담겨있는 메시지 내용
  - 1바이트 데이터 0x01

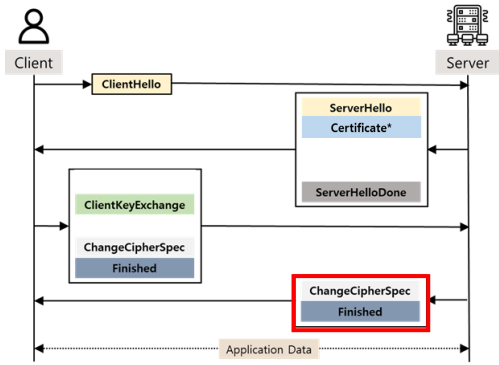
- 통신 7번 (Client -> Server)

Tag:Finished

: 핸드셰이크 절차가 종료되었음을 알림

- 담겨있는 메시지 내용
  - 1바이트 데이터 0x01

보안프로토콜을 활용한 ECHO 클라이언트 구현



- 통신 8번 (Server -> Client)

Tag:ChangeCipherSpec

: 이제 설정한 암호로 통신하겠다고 알림

  - 담겨있는 메시지 내용
    - 1바이트 데이터 0x01

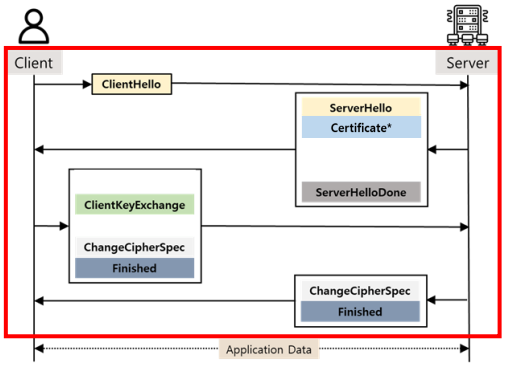
- 통신 9번 (Server -> Client)

Tag:Finished

: 핸드셰이크 절차가 종료되었음을 알림

  - 담겨있는 메시지 내용
    - 1바이트 데이터 0x01

보안프로토콜을 활용한 ECHO 클라이언트 구현



- 주고 받은 데이터
  - 96바이트 난수
    - Client_random (32바이트 난수) // 평문으로 전송
    - Server_random (32바이트 난수) // 평문으로 전송
    - PreMasterSecret (32바이트 난수) // 공개키로 암호화 하여 전송

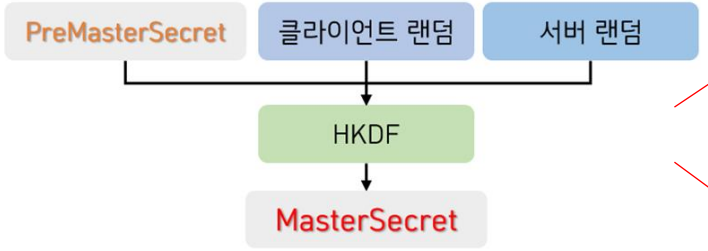
  - TLS 1.2 기준 암호화 키 생성 방식 활용

보안프로토콜을 활용한 ECHO 클라이언트 구현

- 암호키 생성 과정

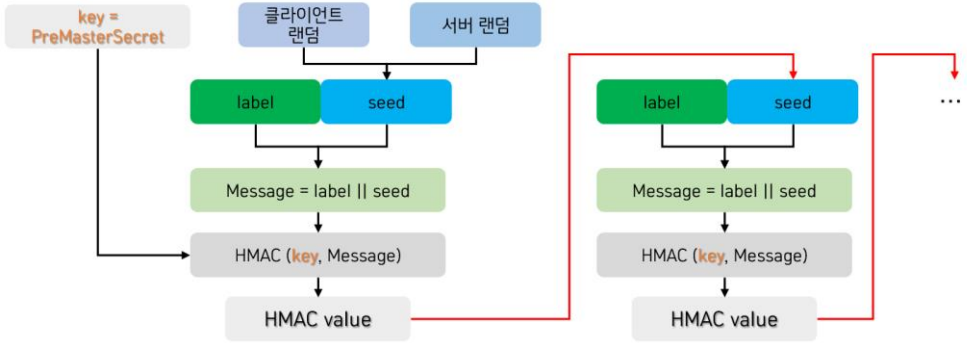
1. 48바이트의 공유 비밀 값(MasterSecret) 생성

- HMAC 기반 키 생성 알고리즘 HKDF 활용(PreMasterSecret, Client_Random, Server_Random 입력 시 **자동 생성 코드 제공**)



- MasterSecret = PRF(PreMasterSecret, "master secret", ClientHello.random + ServerHello.random)[0..47];
- P_hash(secret, label + seed) = HMAC_hash(secret, A(1) + label + seed) +
  HMAC_hash(secret, A(2) + label + seed) +
  HMAC_hash(secret, A(3) + label + seed) + ...

키 메시지



MasterSecret = HMAC value || HMAC Value || ⋯

보안프로토콜을 활용한 ECHO 클라이언트 구현

- 암호키 생성 과정

2. {MAC key, Cipher key, IV} BLOB 생성

- MasterSecret, Client_Random, Server_Random 입력 시 **자동 생성 코드 제공**

- MasterSecret 생성 후 연결동안 데이터 암호화에 사용되는 **키 데이터** (IV, 대칭 암호화 키, MAC 키) 생성
- key_block = PRF(master_secret, "key expansion", ClientHello.random + ServerHello.random);
- P_hash(master_secret, label + seed) = HMAC_hash(master_secret, A(1) + label + seed) +
  HMAC_hash(master_secret, A(2) + label + seed) +
  HMAC_hash(master_secret, A(3) + label + seed) + ...

키 / 메시지

보안프로토콜을 활용한 ECHO 클라이언트 구현

- 암호키 생성 과정

3. {MAC key, Cipher key, IV} BLOB을 용도에 맞게 분리

- Client_write_MAC_key = BLOB[0:16] // 클라이언트가 보내는 메시지의 MAC값 계산용 키
- Server_write_MAC_key = BLOB[16:32] // 서버가 보내는 메시지의 MAC값 계산용 키

- Client_write_key = BLOB[32:48] // 클라이언트가 메시지를 보낼 때 사용하는 암호화 키
- Server_write_key = BLOB[48:64] // 서버가 메시지를 보낼 때 사용하는 암호화 키

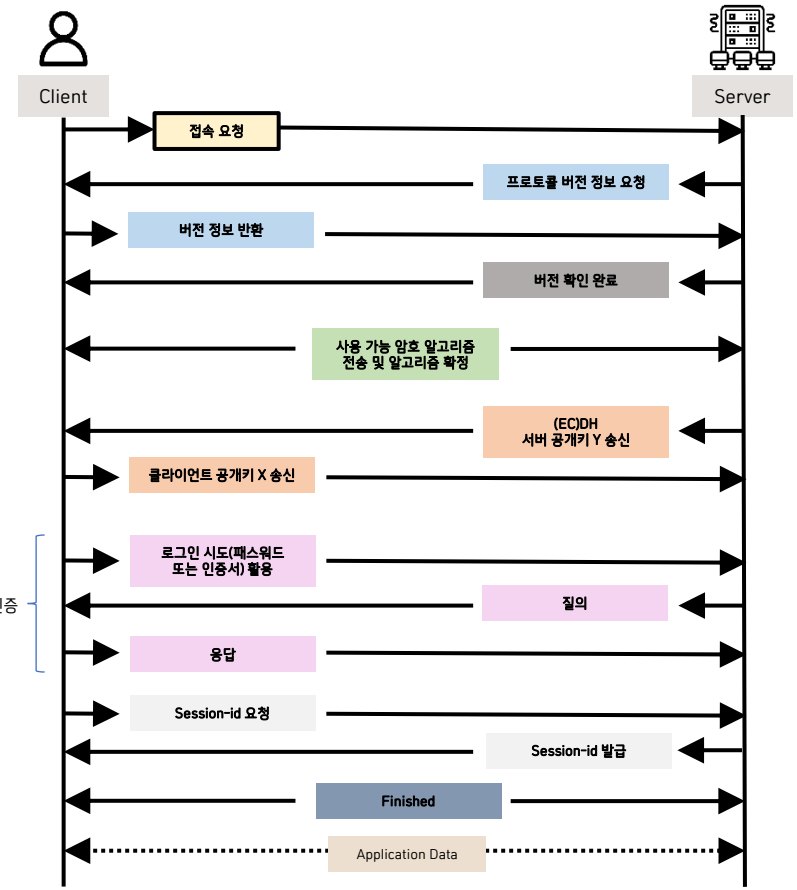
- Client_write_IV = BLOB[64:80] // 클라이언트가 메시지를 보낼 때 사용하는 IV
- Server_write_IV = BLOB[80:96] // 서버가 메시지를 보낼 때 사용하는 IV

02

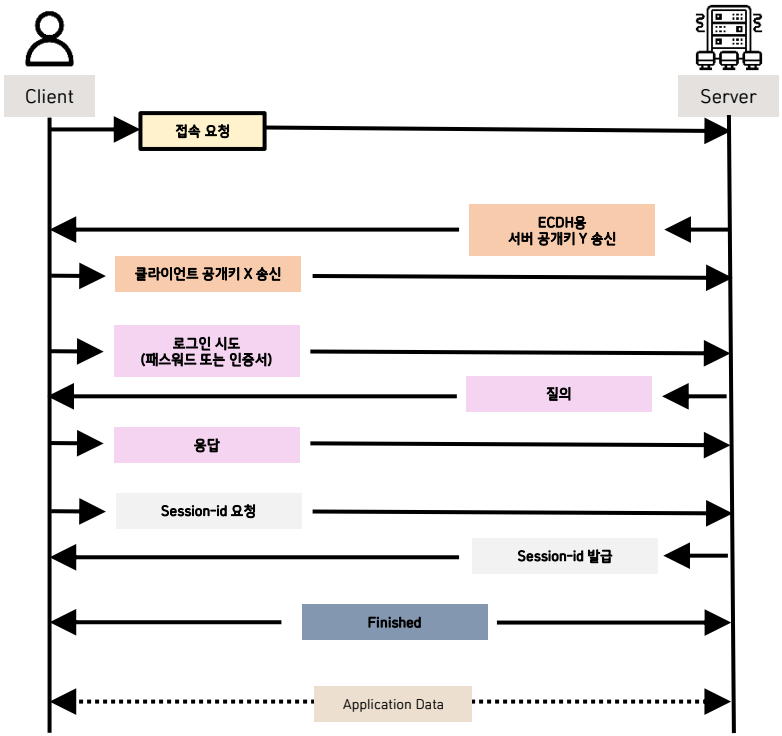
MINI SSH(구현대상x)

보안프로토콜을 활용한 ECHO 클라이언트 구현

- SSH (Secure shell)프로토콜



- Ours



https://datatracker.ietf.org/doc/html/rfc4253

보안프로토콜을 활용한 ECHO 클라이언트 구현

- MINI SSH (Secure shell)프로토콜
  - 데이터 구조

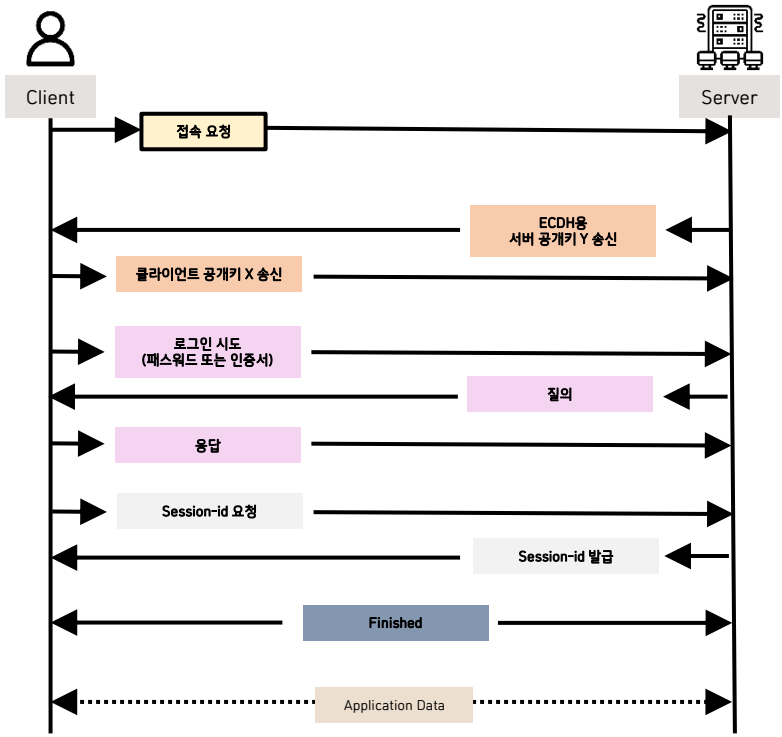
Protocol (1바이트)	Length (4바이트)	Protocol Message (n바이트)	MAC (핸드셰이크 이후)

- Protocol

: 0x00 : 접속 요청(1바이트 데이터 0x01 전송)
: 0x01 : ECDH용 서버의 공개키 전송
: 0x02 : ECDH용 클라이언트의 공개키 전송
: 0x04 : 로그인 시도(1바이트 데이터 0x01 전송)
: 0x05 : 질의
: 0x06 : 응답
: 0x07 : Session id 요청(1바이트 데이터 0x01 전송)
: 0x08 : Session id 발급(32바이트 난수)
: 0x09 : Finished(1바이트 데이터 0x01 전송)

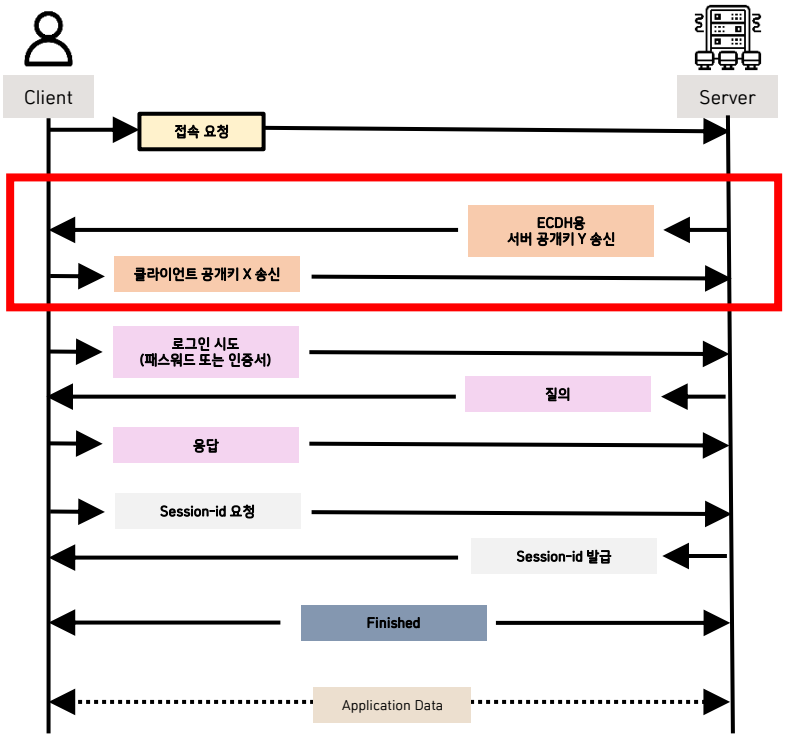
(핸드셰이크 이후 사용 가능)

: 0xFF – 'ECHO Mode'
: 0xFE – 'Data Encryption Mode'
: 0xFD – 'Data Decryption Mode'



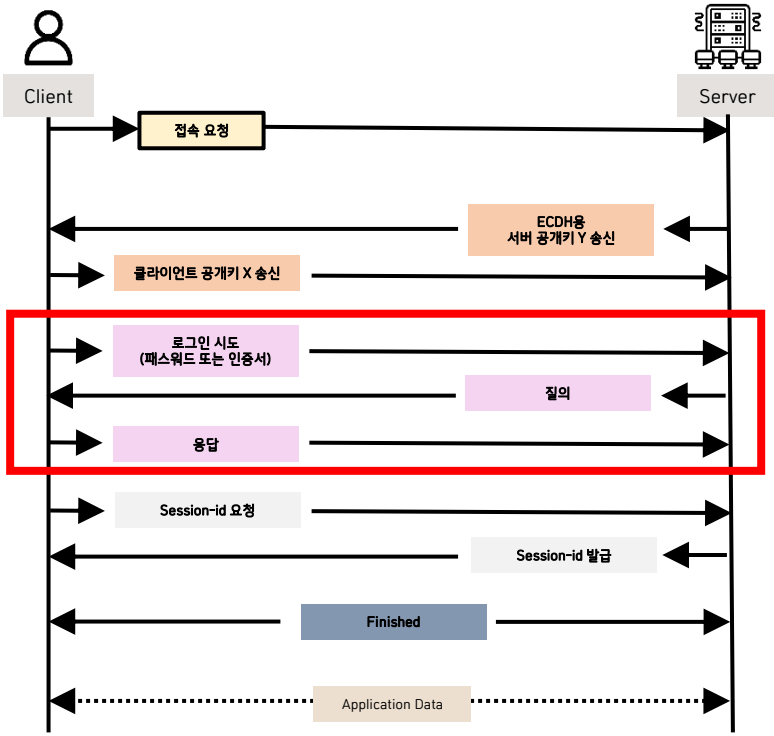
보안프로토콜을 활용한 ECHO 클라이언트 구현

- 비밀값(Shared Secret) 공유 과정
  - ECDH : 타원곡선을 활용한 디피-헬먼 키 교환 방식
    - P256곡선 사용
    - 랜덤한 개인키/공개키 생성 코드 제공
    - 클라이언트의 개인키와 서버의 공개키 입력 시 비밀 값 계산 코드 제공



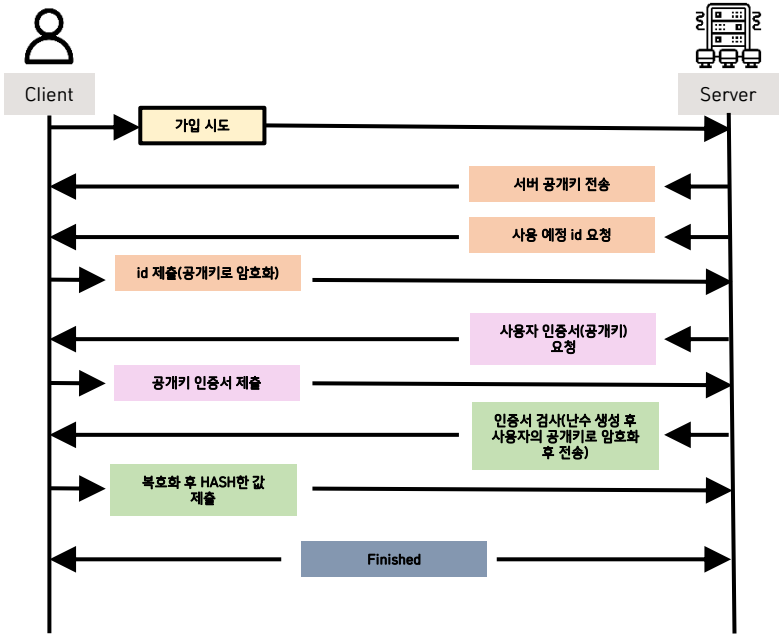
보안프로토콜을 활용한 ECHO 클라이언트 구현

- 사용자 인증 과정
  - 사전에 등록한 계정 또는 인증서 사용
  - 질의/응답은 총 2회 이루어짐
    - 질의1 : ID 요구
    - 응답1 : ID 입력
    - 질의2 : 난수 생성 후 ID에 매칭된 공개키로 암호화 하여 송신
    - 응답2 : 복호화 후 SHA256 해시함수로 해시한 값 송신



보안프로토콜을 활용한 ECHO 클라이언트 구현

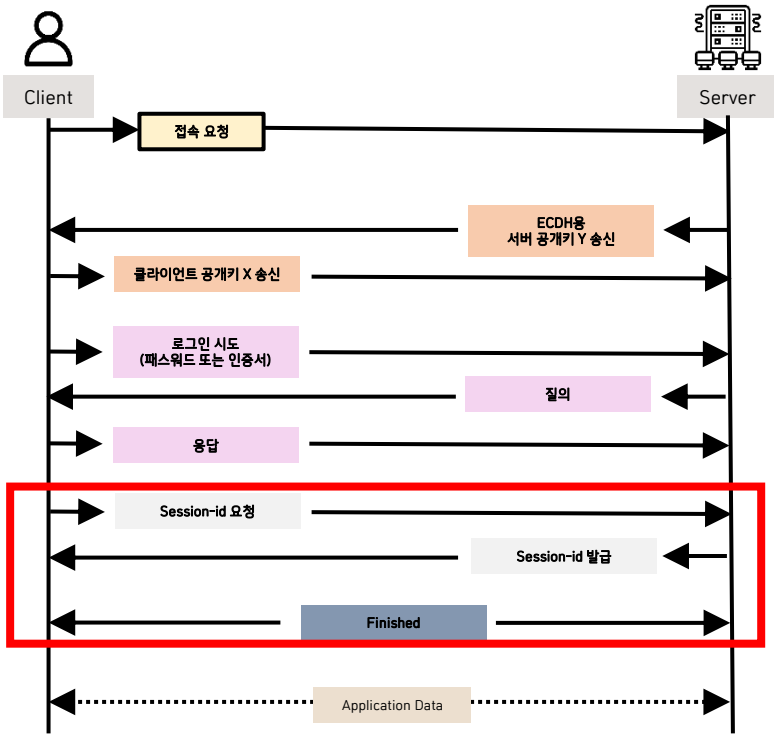
- 인증서 등록 과정



보안프로토콜을 활용한 ECHO 클라이언트 구현

- Session id값 요청
  - 클라이언트
    - 1바이트의 데이터 0x1 송신
  - 서버
    - 32바이트의 Session id 반환

- 세션 수립 및 암호화 시작을 알리는 Finished 송수신



보안프로토콜

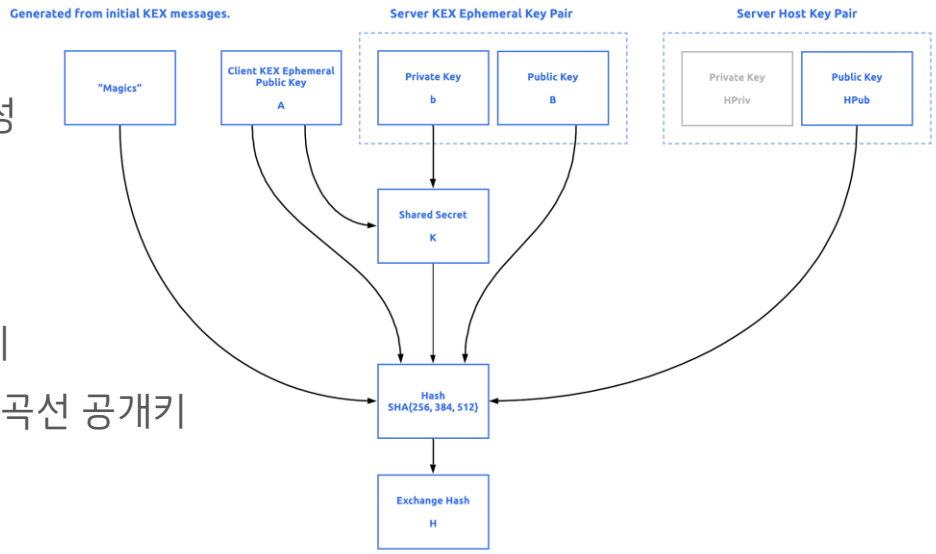
보안프로토콜을 활용한 ECHO 클라이언트 구현

- 암호키 생성 과정

1. 32바이트의 공유 비밀 값(Shared Secret) ‘K’ 생성

2. 공유된 비밀 값 ‘H’ 계산

- SHA256(“Magics” || 클라이언트 타원곡선 공개키 || 공유 비밀 값 K || 서버 타원곡선 공개키 || 클라이언트의 RSA 공개키)



보안프로토콜을 활용한 ECHO 클라이언트 구현

- 암호키 생성 과정

3. 키 생성

- 클라이언트의 초기 IV : SHA1(K || H || “A” || session_id) [0:16]
- 서버의 초기 IV : SHA1(K || H || “B” || session_id) [0:16]

- 클라이언트의 암호화 키 : SHA1(K || H || “C” || session_id) [0:16]
- 서버의 암호화 키 : SHA1(K || H || “D” || session_id) [0:16]

- 클라이언트의 MAC 키 : SHA1(K || H || “E” || session_id) [0:16]
- 서버의 MAC 키 : SHA1(K || H || “F” || session_id)[0:16]

Q&A