인터넷 인증 프로토콜

컴퓨터시스템보안

금오공과대학교 컴퓨터공학부
최태영

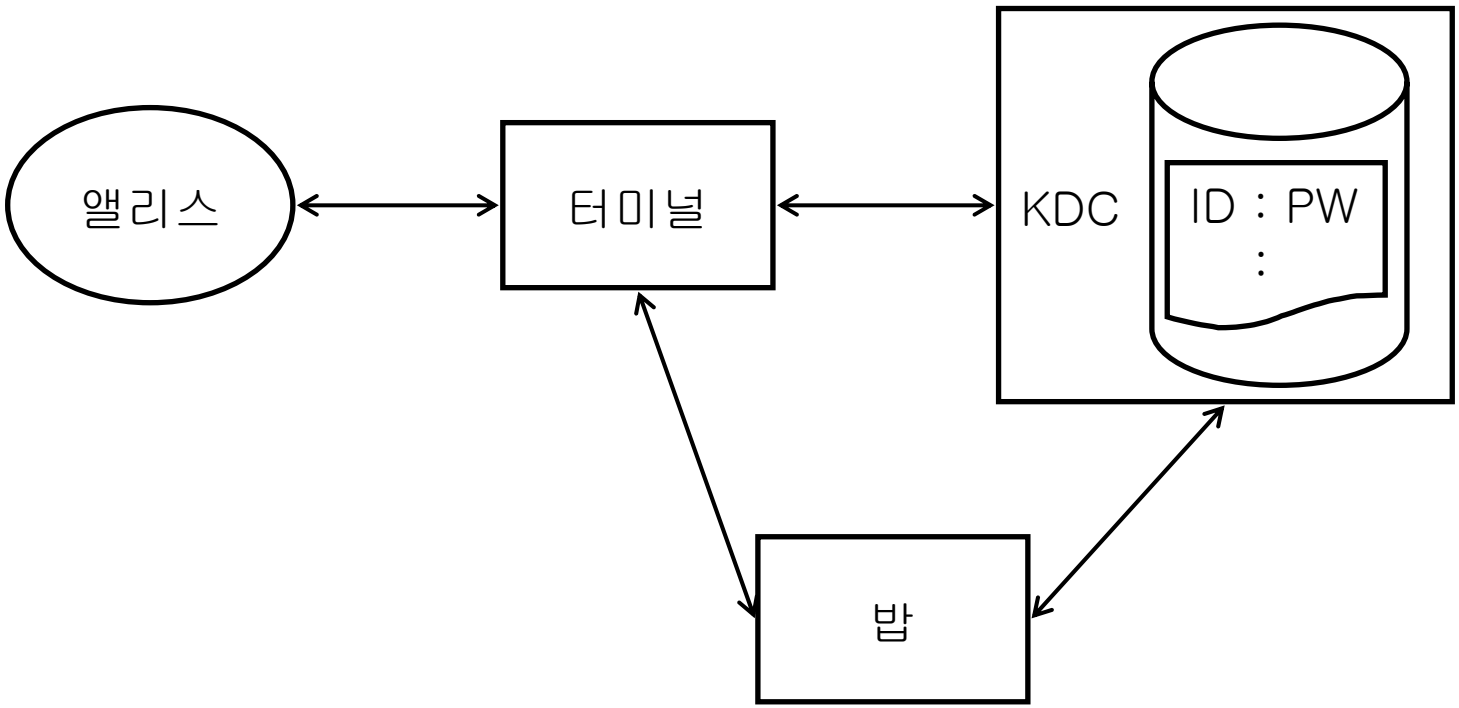
목차

- 커베로스
- GSM 보안구조
- TLS/SSL
- OpenSSL을 이용한 SSL 접속
  - Df
    - df

커베로스 (Kerberos)

- MIT에서 개발한 비밀키 암호 기반 인증 시스템
- 범위 : 학교나 기업과 같은 LAN
- 관리 대상 : 프린터, 서버, 그리고 컴퓨터 터미널과 같은 자원 또는 서버
- 행동 : 중앙 인증 센터에서 사용자를 인증하고 각 자원에 대한 인가
- 각 서버나 자원은 사용자들의 비밀 (password, hash)을 기억할 필요가 없음
  - 시스템 관리자의 부담을 줄임

커베로스 시스템의 예



커베로스 시스템의 비밀키

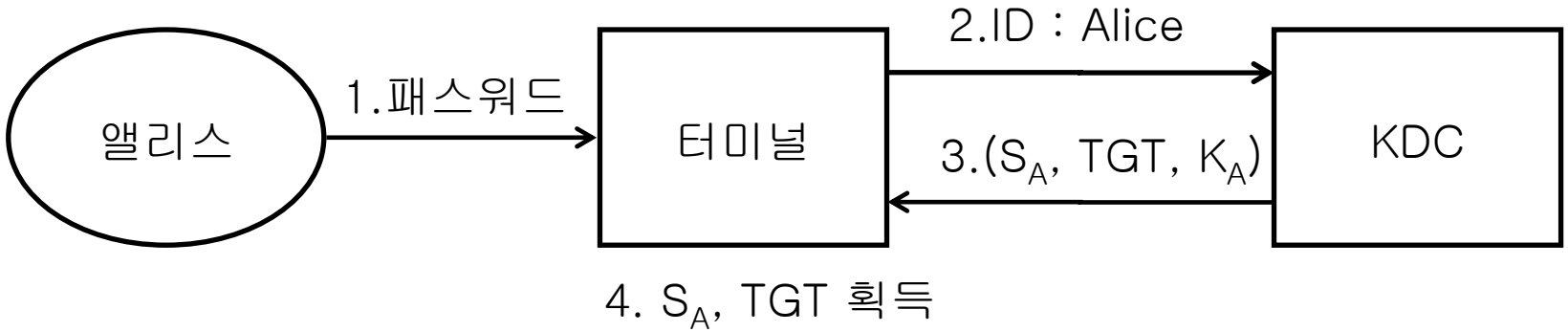
- 키 관리 센터 (KDC)는 각 사용자와의 비밀키 공유
  - 예) 사용자 앨리스와의 공유키 : K_A
  - KDC가 공격 당하면 전체 시스템이 위험해짐
- KDC는 각 서버와의 비밀키도 공유함
  - 예) 서버 밥과의 공유키 : K_B
- KDC 만이 가지는 마스터 키 : K_{KDC}
- 커베로스 단계
  1. 로그인 과정
  2. 서버 접속을 위한 티켓 확보
  3. 서버 접속

커베로스 로그인 과정

- 사용자 앨리스는 터미널에 접속해야 함
  - 터미널을 통해 암/복호화 과정이 수행됨
  - 사용자는 TGT (Ticket Granting Ticket)을 얻어야 함
- 앨리스가 TGT를 얻는 과정
  1. 앨리스는 터미널에 ID와 PW_A를 입력
  2. 터미널은 앨리스의 ID를 KDC로 전해주며 TGT를 요청
  3. KDC는 앨리스의 ID로부터 앨리스와의 공유키 K_A를 찾아냄
     1. KDC는 앨리스와의 세션키 S_A를 랜덤하게 생성
     2. KDC는 TGT = E("Alice" | S_A, K_{KDC})를 생성
  4. KDC는 앨리스의 터미널에 E(S_A | TGT, K_A)를 전달
  5. 앨리스의 터미널은 K_A = h(PW_A)를 통해 K_A를 획득하여 KDC로부터의 메시지 복호화
     1. 앨리스의 터미널은 S_A와 TGT를 얻음

커베로스 로그인 과정

- KDC는 앨리스에 대한 명시적인 인증 과정을 진행하지 않음
- 앨리스는 K_A를 알아야 S_A와 TGT를 획득함
- TGT : 앨리스 ID와 세션키 S_A를 담고 있음
  - 앨리스는 그 내용을 볼 수 없음



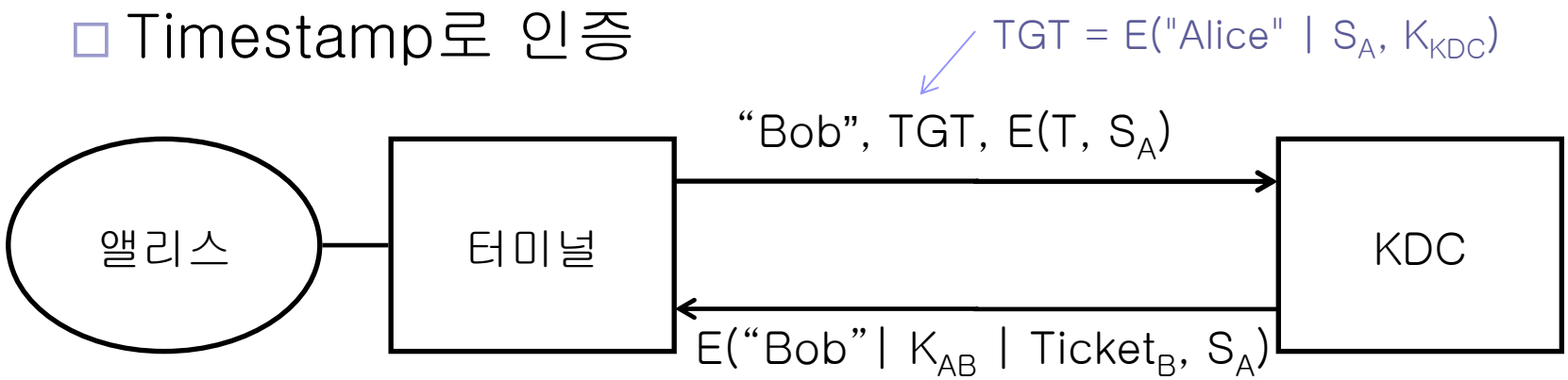
서버 접속을 위한 티켓 확보

- 앨리스가 서버 밥과 통신하려는 경우

1. 앨리스는 KDC에 “Bob", TGT, E(T, S_A)를 전송
   1. KDC는 TGT를 자신의 K_{KDC}로 복호화하여 "Alice"와 S_A를 획득
   2. KDC는 타임스탬프 E(T, S_A)를 복호화하여 T를 얻은 후 현재 시각과 비교
   3. KDC는 앨리스가 밥에게 접속할 권한이 있는 지 확인
   4. KDC는 앨리스와 밥 사이의 세션 키 K_{AB}를 생성
   5. KDC는 $Ticket_B$ = E("Alice" | K_{AB}, K_B) 생성
2. KDC는 앨리스에게 E("1. Bob" | K_{AB} | $Ticket_B$, S_A)를 전송

서버 접속을 위한 티켓 확보

- 앨리스는 서버 밥에게 접속할 티켓이 필요
- KDC는 상대방이 앨리스인가를 확인할 정보가 필요
  - TGT뿐만 아니라 세션키 SA를 가지고 있는지 확인 필요
  - Timestamp로 인증



서버 접속을 위한 티켓 확보

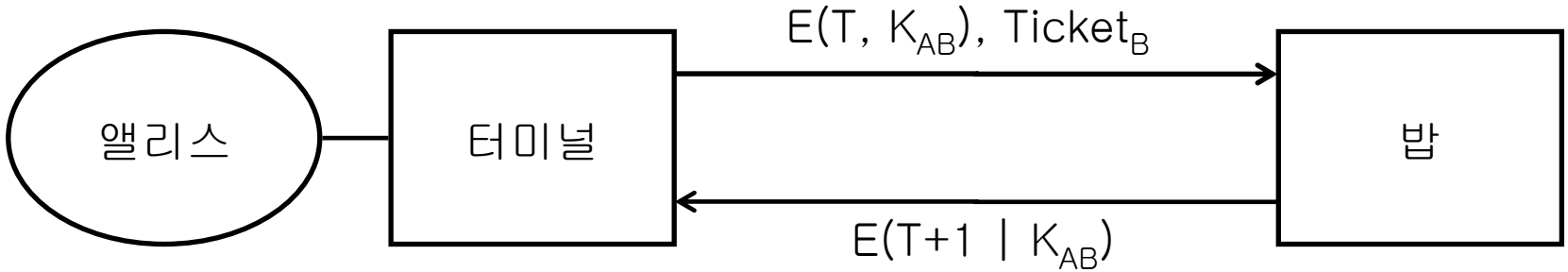
- 첫 번째 메시지의 손상 여부
  - “Bob” 손상 : 두 번째 프로토콜에서 앨리스가 확인
  - TGT 손상 : KDC가 복호화한 뒤 “Alice”가 손상되었는지 확인
  - $E(T, S_A)$ 손상 : KDC가 복호화한 뒤 손상된 T’를 인증하지 않음
- 두 번째 메시지의 손상 여부
  - 복호화한 후 문자열 “Bob”을 앨리스가 읽을 수 있는지 여부로 판단

서버 접속

   1. 앨리스는 현재 시각 T를 밥과의 세션 키 K_{AB}로 암호화한 $E(T, K_{AB})$를 생성
1. 앨리스는 $E(T, K_{AB})$와 $Ticket_B$를 밥에게 전송
   1. 밥은 $Ticket_B$를 KDS와 자신간의 비밀키 K_B로 복호화하여 상대방이 “Alice"이라는 사실을 알고 앨리스와의 세션 키 K_{AB}를 얻음
   2. 밥은 K_{AB}로 $E(T, K_{AB})$를 복호화하여 T를 얻고, 자신의 현재 시각과 비교하여 시각오차 범위 이내이면 상대방이 앨리스임을 인증함
2. 밥은 T+1을 세션 키 K_{AB}로 암호화한 $E(T+1, K_{AB})$를 앨리스에게 전송
   1. 앨리스는 암호문을 세션 키 K_{AB}로 복호화하여 자신이 보낸 T+1인지 확인

서버 접속

- 각 메시지의 무결성과 인증을 다음과 확인
- 첫 번째 메시지 검증
  - $E(T, K_{AB})$: timestamp 검증으로 확인
  - $Ticket_B$: 복호화하여 문자열 “Alice”를 확인
- 두 번째 메시지 검증
  - 복호화 후 T+1을 읽을 수 있는가를 확인



$Ticket_B = E(\text{"Alice"} \mid K_{AB}, K_B)$

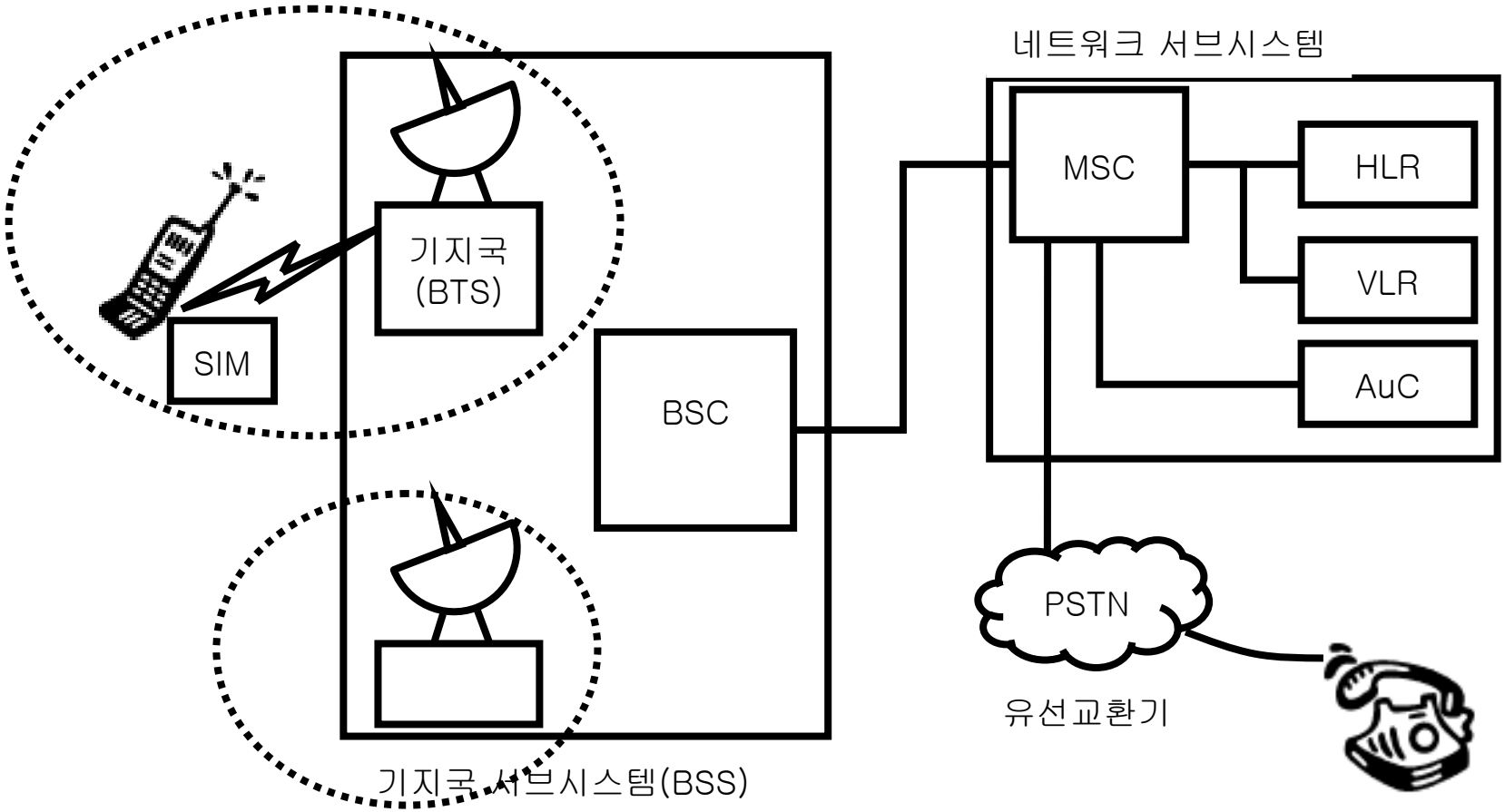
커베로스의 안전성 및 성능

- 무상태성 (stateless)
  - KDC는 사용자 로그인을 기억하지 않음
  - 로그인은 TGT에 저장되어 있음
  - 성능저하
  - 서비스 거부 공격에 강함
- 사용자와 서버와의 통신
  - 세션키로 암호화되어 메시지가 전송되므로 인증이 이루어진 이후로는 공격자가 세션을 뺏을 수 없음
- $Ticket_B$는 사용자를 거쳐 서버에게 전달
  - $E(T, K_{AB})$의 복호화를 위해 필요

GSM 보안 구조

- Global System for Mobile communications, Groupe Special Mobile)
- 휴대전화 통신 표준, 212개 국가에서 약 30억 사용자들이 있음 (2007년)
- Second Generation (2G) 휴대전화 표준
  - First generation은 아날로그 통신
- Time Division Multiple Access (TDMA) 방식
  - 한국은 Code Division MA 방식
- 휴대전화의 표준은 IMT-2000 (International Mobile Telecommunications-2000, 3G) 방식으로 이동중

GSM의 구성


네트워크 서브시스템
MSC
HLR
VLR
AuC
기지국
(BTS)
SIM
BSC
PSTN
유선교환기
기지국 서브시스템(BSS)
유선전화

- SIM (Subscriber Identity Module) : 가입자 식별 모듈
  - 사용자가 휴대전화 서비스 업체와 계약을 체결하면서 자신의 대표하는 도구
  - Smart card의 일종이며 휴대폰에 장착
  - Ki : ISIM과 같이 사용자 인증을 위한 정보
- ISIM (International Mobile Subscriber ID)
  - 국제 이동국 식별번호(요금 청구의 대상)
- BTS (Base Transfer Station) : 기지국
- BSC (Base Station Controller)
- BSS (Base Station Subsystem) : 기지국 서브시스템
- MSC (Mobile Switching Center) : 이동교환센터
  - HLR (Home Location Register) : 사용자가 등록된 서브시스템
  - VLR (Visitor Location Register) : 다른 서브시스템을 사용하는 사용자의 등록 정보가 저장된 서버
  - AuC (Authentication Center) : 사용자 인증 수행
- PSTN (Public Switch Telephone Network) : 아날로그 전화 교환 네트워크

GSM 보안 목표

- 1G 무선통신은 보안에 대한 고려 없이 설계되었음
- 2G 무선통신은 보안 기능 수립이 목표
  - 유선교환망 PSTN 수준의 보안
  - 복제방지
- 보안 서비스
  - 익명성
    - 문제점 : 휴대전화가 접속하는 BTS로 사용자의 위치 파악 가능
    - 임시 이동가입자 식별번호 (Temporary Mobile Subscriber ID, TMSI) 할당
  - 인증 : 인증 프로토콜을 사용하여 사용자 인증
    - 중간자 공격은 고려하지 않았음
  - 비밀성 : 스트림 암호 사용
    - 블록암호는 높은 컴퓨팅 파워 요구
    - 오류율이 높은 경우 스트림 암호가 오류 파급이 적음

GSM 인증 프로토콜

1. 휴대전화 → 기지국 : 자신의 SIM 카드에 있는 IMSI
2. 기지국 → 이동교환센터 (MSC) : IMSI
3. MSC내의 인증센터 (AuC)는 IMSI에 대응되는 비밀키 Ki를 저장장소에서 꺼낸다.
    1. AuC는 난수 RAND를 만들고, XRES = A3(RAND, Ki)를 계산한다. 여기서 A3은 해시함수이다. A3은 SIM 카드 내에도 존재한다.
    2. AuC → 기지국 : RAND, XRES
4. 기지국 → IMSI를 보낸 휴대전화 : RAND
5. 휴대전화의 SIM 카드는 SRES = A3(RAND, Ki)를 계산
    1. 휴대전화 → 기지국 : SRES
6. 기지국은 SRES = XRES를 확인

※ 비밀키 Ki가 휴대전화나 네트워크 서브시스템 바깥으로 노출되지 않음

GSM 비밀키 생성

1. 네트워크 서브시스템은 인증과정에서 Ki를 저장장치에서 꺼낸 뒤 Kc = A8(RAND, Ki)를 생성한다.
   - A8은 해시 함수
2. 서브시스템은 (RAND, XRES)를 기지국에 보낼 때 Kc를 같이 보낸다.

① 휴대전화는 Kc = A8(RAND, Ki)를 계산한다.

3. 기지국과 휴대전화는 Kc를 시드로 사용하여 A5(Kc) 키 스트림을 생성한다.
   - A5는 랜덤 스트림 생성기
4. 모든 메시지는 키 스트림과 bitwise xor연산으로 암호화되어 전송되고 bitwise xor연산으로 복호화된다.

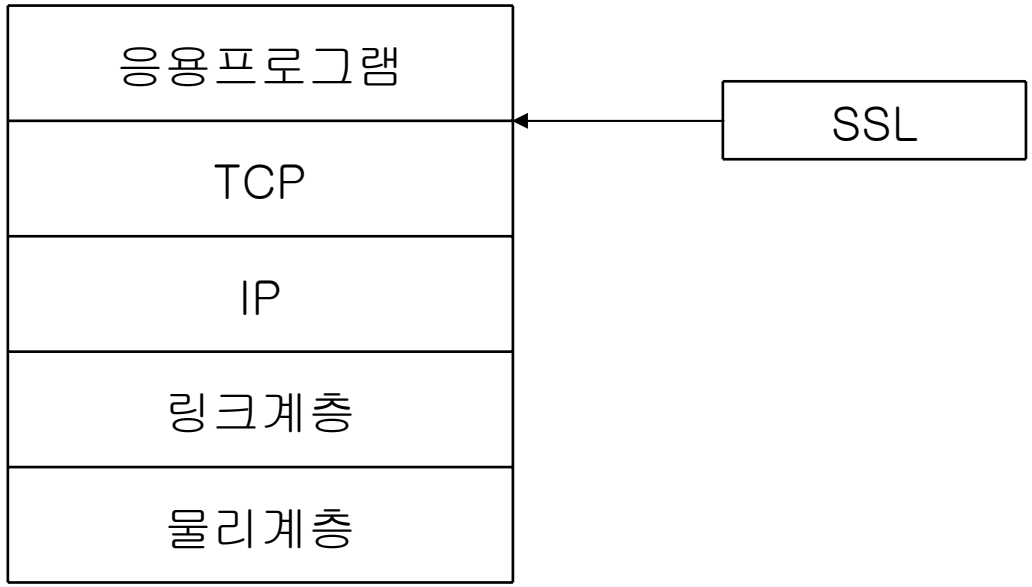
GSM의 안전성

- GSM의 보안상 문제점
  - A3과 A8 해시 함수가 1998년에 문제가 있음이 알려짐
  - 비밀키 Ki은 휴대전화 등록 시에 복제될 수 있음
  - 기지국과 네트워크 서브시스템 사이에는 암호화하지 않은 통신을 사용함
  - SIM카드 내의 Ki는 출력되지 않도록 설계되었지만 광학적 자극이나 전력소모의 차이를 분석해서 추측하는 방법이 있음이 알려짐
  - 중간자 공격에 대응책이 없음
    - 기지국을 인증하는 프로토콜이 존재하지 않음
    - 공격자는 위장 기지국을 설치하여 인증 4 단계에서 자신의 RAND를 휴대전화에 전달
    - 기지국은 암호화를 하지 말 것을 요구할 수 있음
    - 사용자의 의심을 막기 위해 상대방을 연결하고 통화내용을 forwarding할 수 있음
- GSM의 보안성 미흡 ? : 설계 조건은 만족함 (PSTN 수준의 보안)

TLS/SSL

- TCP/IP 네트워크 통신 계층에 존재하는 보안 프로토콜의 일종
- IPsec : IP layer와 link layer 사이에서 IP packet들에 대한 무결성과 비밀성 보장
- SSL (Secure Socket Layer) : application layer와 TCP layer 사이에서 메시지에 대한 무결성과 비밀성 보장
  - 라이브러리의 형태로 제공됨 (다음 슬라이더)
  - Web browser와 web server 사이의 통신 지원
  - 대부분의 web browser에서 지원
  - 많은 통신 응용프로그램에서 사용 : ssh
- TLS (Transport Layer Security)가 SSL을 계승

SSL의 구조적 위치



TLS/SSL의 개요

- 인증 수행, 세션 키 분배, 그리고 세션 키로 암호화된 메시지 송수신
- SSL session : session key가 정해지면 시작됨
- 지원하는 기능 (암호화를 이용)
  - 비밀성 : 노출되어서는 안 되는 정보를 인터넷을 통해 보내기 위해
  - 무결성 : 결재 금액과 같은 예민한 정보의 변경을 막기 위해
- 부인 방지 기능은 제공되지 않음 (전자 서명을 사용해야 함)
- 서버 인증은 중요함 : fishing site의 공격을 막기 위해
- 사용자 인증은 부차적임 : session이 시작된 이후 id/pw로 인증 가능

SSL의 서버 인증

- 공개키 암호화를 통한 인증 프로토콜 사용
  - 사용자는 난스를 서버의 공개키로 암호화하여 전송
  - 서버는 난스를 볼 수 있음을 사용자에게 입증
- 서버는 사용자에게 자신의 공개키 인증서를 제공해야 함
  - 사용자는 서버의 공개키 인증서를 검증할 수 있어야 함
  - 서버는 컴퓨터 운영체제에서 제공하는 인증 기관을 통해 공개키 인증서를 발급받아야 함
- 인증할 수 없는 공개키 인증서를 사용자가 받았을 경우 SSL은 경고 메시지를 발생시킴

TLS/SSL Protocol

- Handshake protocol
  - 서버 인증과 session key 교환 과정
- Record Protocol
  - 암호화된 메시지 송수신 과정
- Alert protocol
  - 에러가 발생했을 때 시작되는 에러 처리 과정

Handshake Protocol

- 3개의 필드로 이루어진 메시지 교환
  - type [1 byte] : 메시지 종류 명시
  - length [3 bytes] : 메시지 길이
  - content [≥ 1 byte] : 메시지 종류에 따른 값들
- 4 단계 진행
  - 1단계 : security capabilities establishment
  - 2단계 : server authentication and key exchange
  - 3단계 : [client certificate and key exchange]
  - 4단계 : application phase

메시지 종류들

메시지 종류	파라미터
Hello request	없음
Client hello	버전, 난수 R_A, Session id, 암호 알고리즘 목록, 압축 알고리즘 목록
Server hello	버전, 난수 R_B, Session id, 암호 알고리즘, 압축 알고리즘
Certificate	X.509v3 인증서 체인
Server key exchange	Parameters, 전자 서명
Certificate request	Type, authorities
Server hello done	없음
Certificate verify	전자 서명
Client key exchange	Parameters, 전자 서명
Finished	해시 값

Handshake 1 단계

- 사용자 → 서버 : client hello
- 서버 → 사용자 : server hello
- Parameters (client hello)
  - version : 사용자가 지원 가능한 최신 TLS 버전 (2009.6월 현재 TLS1.2)
    - major와 minor field로 구성
  - 난수 : 32 비트의 타임스탬프와 28 바이트의 난수로 구성, R_A
  - 세션 ID : 0일 경우 새로운 세션, 다른 값이면 이미 연결되어 있는 세션임을 의미함
  - 암호 알고리즘 목록 : 사용자가 사용할 수 있는 암호 알고리즘들의 목록, 앞의 것일 수록 선호됨
  - 압축 방법 : 사용자가 사용할 수 있는 압축 알고리즘들의 목록

Handshake 1 단계 (계속)

- Parameters (server hello)
  - 버전 : min (사용자 제안 버전, 서버 최신 버전)
  - 난수 : 서버가 제시하는 새로운 값, R_B
  - 세션 ID : client hello의 세션 ID가 0이 아니면 동일한 값 사용, 0이면 새로운 값 부여
  - 암호 알고리즘 : client hello의 목록에서 한 개 선택
  - 압축 방법 : client hello의 목록에서 한 개 선택

Handshake 2 단계

- 서버 → 사용자

1. Certificate
   - 서버의 인증서와 검증을 위한 인증서 체인
2. Server key exchange
   - 서버가 인증서를 가지지 않은 경우에 여기에 공개키를 보냄
3. Certificate request (선택적)
   - 사용자에게 인증서를 요구하는 메시지
4. Server hello done
   - 2 단계 메시지를 모두 보냈음을 알림

Handshake 3 단계

- 사용자 → 서버

1. Certificate
   - 2 단계에서 서버가 인증서를 요구한 경우
   - 없으면 No certificate를 리턴
2. Server key exchange
   - 48 byte 크기의 pre-master secret S를 생성
   - $\{S\}_B$를 보냄
3. Certificate verify
   - 사용자가 보낸 인증서를 검증하는 과정
   - 지금까지의 내용에 대한 해시를 개인키로 서명한 내용

Handshake 4 단계

- 사용자 → 서버
  1. Change cipher specs
  2. Finished
- 서버 → 사용자
  1. Change cipher specs
  2. Finished
- 지금까지의 메시지로 *MS*와 세션 키 key_block을 생성
  - MS = MD5(S | SHA('A' | S | R_A | R_B)) |
    MD5(S | SHA('BB' | S | R_A | R_B)) |
    MD5(S | SHA('CCC' | S | R_A | R_B))
  - key_block = MD5(MS | SHA('A' | MS | R_B | R_A)) |
    MD5(MS | SHA('BB' | MS | R_B | R_A)) |
    MD5(MS | SHA('CCC' | MS | R_B | R_A))

Record Protocol

1. 메시지 단위 : 최대 16KB 크기의 fragmentation
2. Fragmentation에는 무손실 압축
3. MAC 생성
   - Key_block을 키로하는 HMAC 유사 알고리즘
   - Fragmentation 뒤에 붙음
4. Fragmentation은 key_block을 비밀키로 하여 암호화
5. Header
   - 패킷 종류, major 버전, minor 버전, 압축되었을 때의 메시지의 길이

Alert Protocol

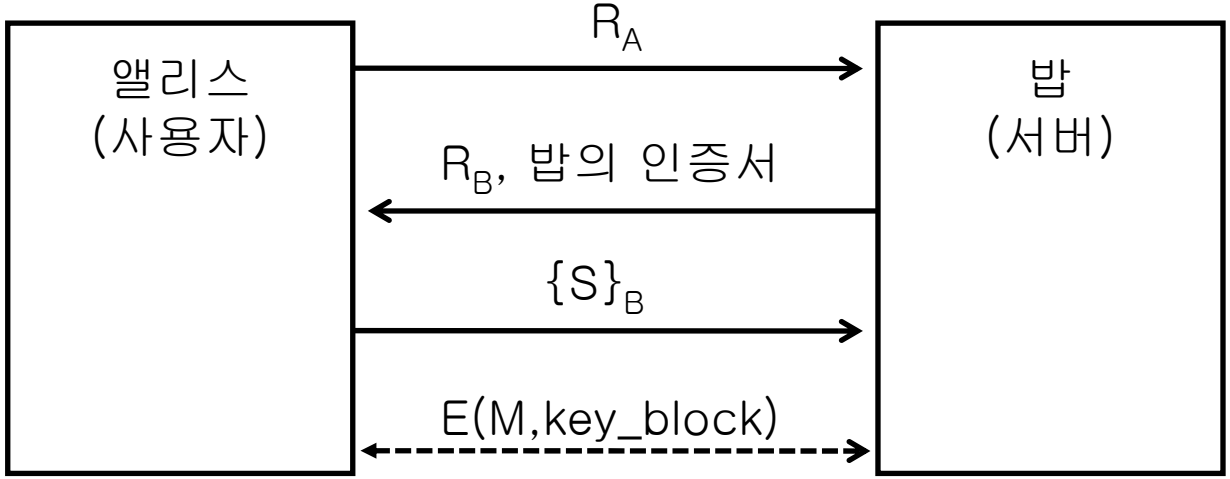
- Fatal alert : 연결을 끊음
- Non-fatal alert : 에러처리는 하고 연결은 유지함

치명적 경고종류	설명
Unexpected message	예상치 않은 메시지 도착
Bad record MAC	MAC이 잘못된 경우
Decompression failure	압축 해제 시 에러 발생
Handshake failure	협상 시 알맞은 알고리즘이 없을 경우
Illegal parameters	파라미터들이 지정된 범위의 값을 벗어난 경우

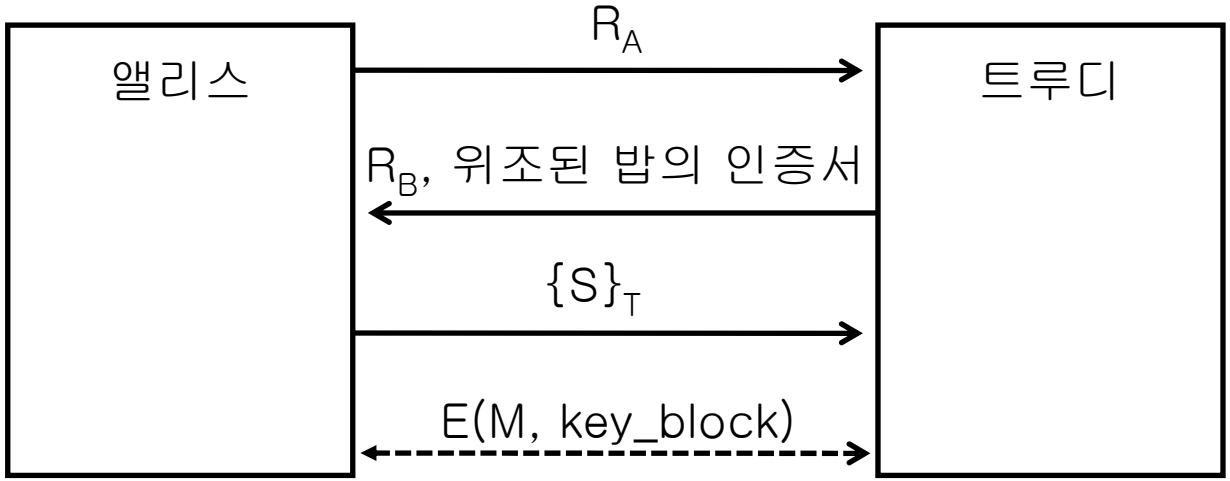
치명적이지 않은 경고

경고 종류	설명
No certificate	인증서를 요청했는데 상대방이 가지지 않은 경우
Bad certificate	인증서 검증에 실패한 경우
Unsupported certificate	버전 등으로 인해 처리할 수 없는 인증서 양식인 경우
Certificate revoke	철회된 인증서인 경우
Certificate expired	유효기간이 지난 인증서인 경우
Certificate unknown	검증과정에서 에러가 생긴 경우
Close notify	더 이상 보낼 메시지가 없음을 알림

TLS/SSL Protocol Summary



TLS/SSL에 대한 공격 시도



OpenSSL을 이용한 SSL 접속 (서버)

```
SSL_library_init();
SSL_load_error_strings();
bio_err = BIO_new_fp(stderr,BIO_NOCLOSE);
meth = SSLv23_server_method();
ctx = SSL_CTX_new(meth);
res = SSL_CTX_use_certificate_chain_file(ctx, "BobCert.pem");
assert(res);
res = SSL_CTX_use_PrivateKey_file(ctx,"BobPriv.pem“, SSL_FILETYPE_PEM);
assert(res);
res = SSL_CTX_check_private_key(ctx);
assert(res);
sbio=BIO_new_ssl(ctx,0);
BIO_get_ssl(sbio, &ssl);
assert(ssl);
SSL_set_mode(ssl, SSL_MODE_AUTO_RETRY);
```

SSL 초기화 및 bio 설정

SSL 통신을 위한 데이터 입력

SSL로부터 Bio 채널 생성

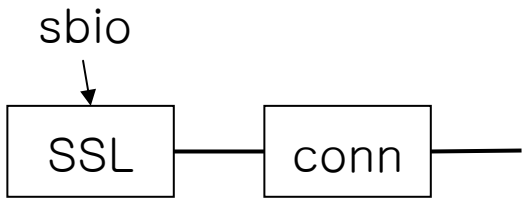
```
bbio = BIO_new(BIO_f_buffer());
sbio = BIO_push(bbio, sbio);
acpt=BIO_new_accept("4433");
BIO_set_accept_bios(acpt, sbio);
out = BIO_new_fp(stdout, BIO_NOCLOSE);
if(BIO_do_accept(acpt) <= 0) {
  fprintf(stderr, "Error setting up accept BIO₩n");
  ERR_print_errors_fp(stderr);
  return 0;
}
if(BIO_do_accept(acpt) <= 0) {
  fprintf(stderr, "Error in connection₩n");
  ERR_print_errors_fp(stderr);
  return 0;
}
sbio = BIO_pop(acpt);
BIO_free_all(acpt);
```

```
if (BIO_do_handshake(sbio) <= 0) {
  fprintf(stderr, “Error in SSL handshake₩n”);
  ERR_print_errors_fp(stderr);
  return 0;
}
```

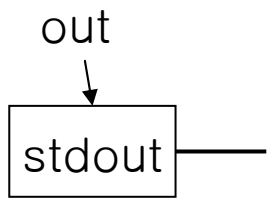
```
BIO_puts(sbio, "HTTP/1.0 200 OK₩nContent-type:text / plain ₩n₩n");
BIO_puts(sbio, "₩nConnection Established₩nRequest headers:₩n");
BIO_puts(sbio, "--------------------------------₩n");
for(;;) {
  len = BIO_gets(sbio, tmpbuf, 1024);
  if(len <= 0) break;
  BIO_write(sbio, tmpbuf, len);
  BIO_write(out, tmpbuf, len);
  if((tmpbuf[0] == '₩r') || (tmpbuf[0] == '₩n')) break;
}
BIO_puts(sbio, "--------------------------------₩n");
BIO_puts(sbio, "₩n");

BIO_flush(sbio);
BIO_free_all(sbio);
return(0);
```

BIO Method and Bio Object

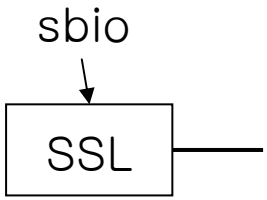
- BIO method : TCP/IP connection이나 buffering 등의 기능을 결정함
- BIO object : 1개의 BIO method로 구성되어 있으며
- BIO chain : 여러 개의 BIO object들이 chain처럼 연결된 것
- BIO object 생성 함수들
  - sbio = BIO_new_ssl_connect(ctx)
    - SSL method를 가진 BIO object sbio와 TCP connect method를 가진 BIO object 생성
  - out = BIO_new_fp(stdout, …)
    - 콘솔 (stdout) method를 가진 BIO object out 생성
  - sbio = BIO_new_ssl(ctx, 0)
    - SSL method를 가진 BIO object sbio 생성
  - bbio = BIO_new(…)
  - acpt = BIO_new_accept("port.no") : accept method를 가진 BIO



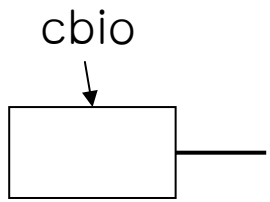
sbio = BIO_new_ssl_connect()



out = BIO_new_fp(stdout,…)



sbio = BIO_new_ssl()

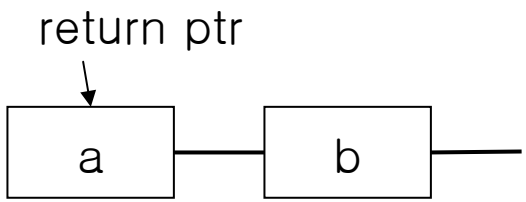


cbio = BIO_new()

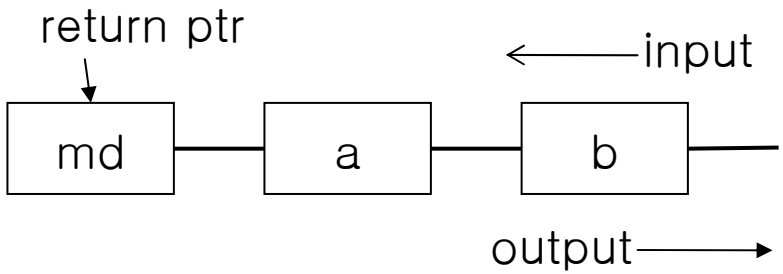
BIO_push, BIO_pop

- BIO object들의 chain을 관리하는 함수
- BIO_push(*a*, *b*): BIO object *b*앞에 BIO object *a*를 붙여 2개의 object로 구성된 BIO chain 형성
- BIO_push(*md*, *a*): BIO chain *a*앞에 object *md*를 붙임
- BIO_pop(*a*): BIO object *a*를 자신이 포함된 chain으로부터 끄집어냄
- BIO chain의 가장 오른편으로부터 데이터가 입력되며 왼편으로 이동함
  - 데이터는 각 object를 경유하면서 해당 method의 처리를 받음
- 데이터는 왼편에서 오른편으로 이동하면서 가장 오른편으로 출력됨
  - 역시 경유하는 object에서 각 method의 처리를 받음

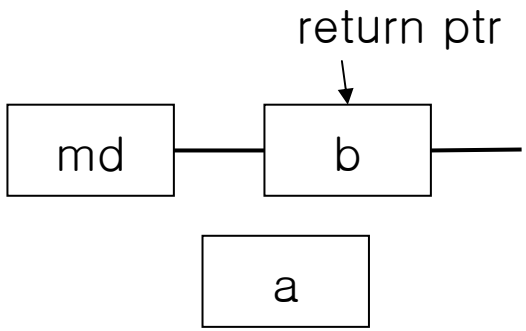
BIO_push(a, b)



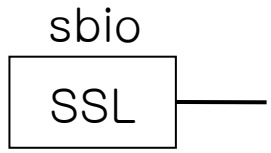
BIO_push(md, a)



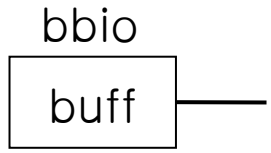
BIO_pop(a)



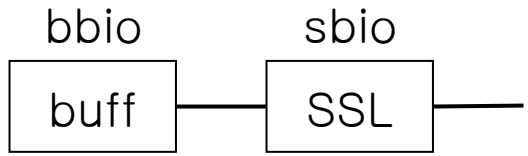
sbio = BIO_new_ssl()



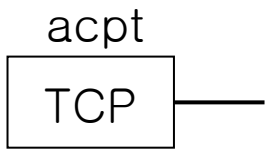
bbio = BIO_new(BIO_f_buffer())



sbio = BIO_push(bbio, sbio)

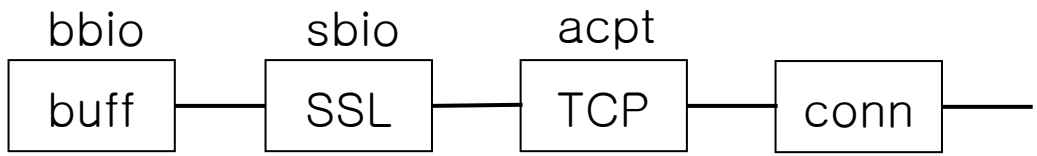


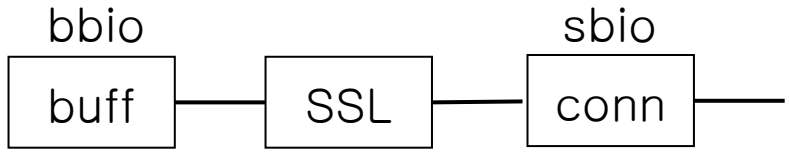
acpt = BIO_new_accept("4433")



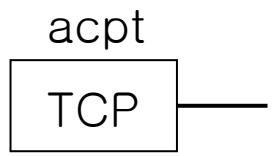
BIO_set_accept_bios(acpt,sbio)

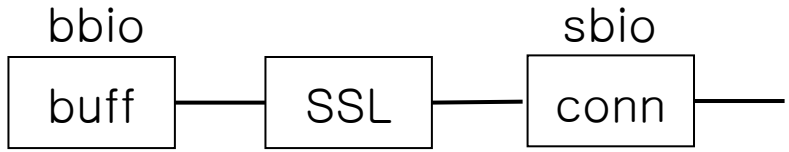
BIO_do_accept(acpt)



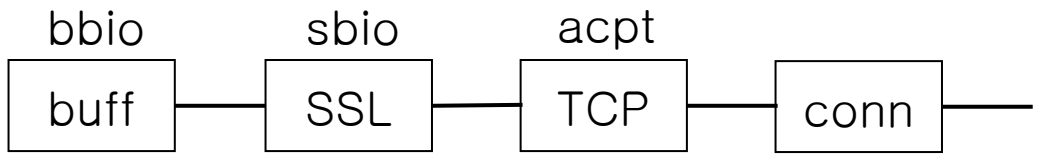


sbio = BIO_pop(acpt)





BIO_do_accept(acpt)



OpenSSL을 통한 클라이언트 접속

```
if (!bio_err){
  SSL_library_init();
  SSL_load_error_strings();
  bio_err=BIO_new_fp(stderr,BIO_NOCLOSE);
}
meth = SSLv23_client_method();
ctx = SSL_CTX_new(meth);
sbio = BIO_new_ssl_connect(ctx);
BIO_get_ssl(sbio, &ssl);
if(!ssl) {
  fprintf(stderr, "Can't locate SSL pointer₩n");
  exit(1);
}
SSL_set_mode(ssl, SSL_MODE_AUTO_RETRY);
```

```
  BIO_set_conn_hostname(sbio, "127.0.0.1:4433");
  out = BIO_new_fp(stdout, BIO_NOCLOSE);
  res = BIO_do_connect(sbio);
  if (res <= 0) {
    fprintf(stderr, "Error connecting to server₩n");
    ERR_print_errors_fp(stderr);    exit(1);
  }
  res = BIO_do_handshake(sbio);
  if (res <= 0) {
    fprintf(stderr, "Error establishing SSL connection ₩n");
    ERR_print_errors_fp(stderr);    exit(1);
  }
  BIO_puts(sbio, "GET / HTTP/1.0₩n₩n");
  for(;;) {
    len = BIO_read(sbio, tmpbuf, 1024);
    if(len <= 0) break;
    BIO_write(out, tmpbuf, len);
  }
  BIO_free_all(sbio);
  BIO_free(out);
  return(0);
}
```