HandShake protocol

SHA256

https://github.com/openssl/openssl/blob/master/include/openssl/sha.h

```
#include <openssl/sha.h>
U8 digest[SHA256_DIGEST_LENGTH] = {0}; SHA256_CTX hs = {0};
SHA256_Init(&hs);
SHA256_Update(&hs, str, strlen(str)); SHA256_Update(&hs, str, strlen(str));
SHA256_Final(digest, &hs);
```

HMAC

https://github.com/openssl/openssl/blob/master/include/openssl/hmac.h

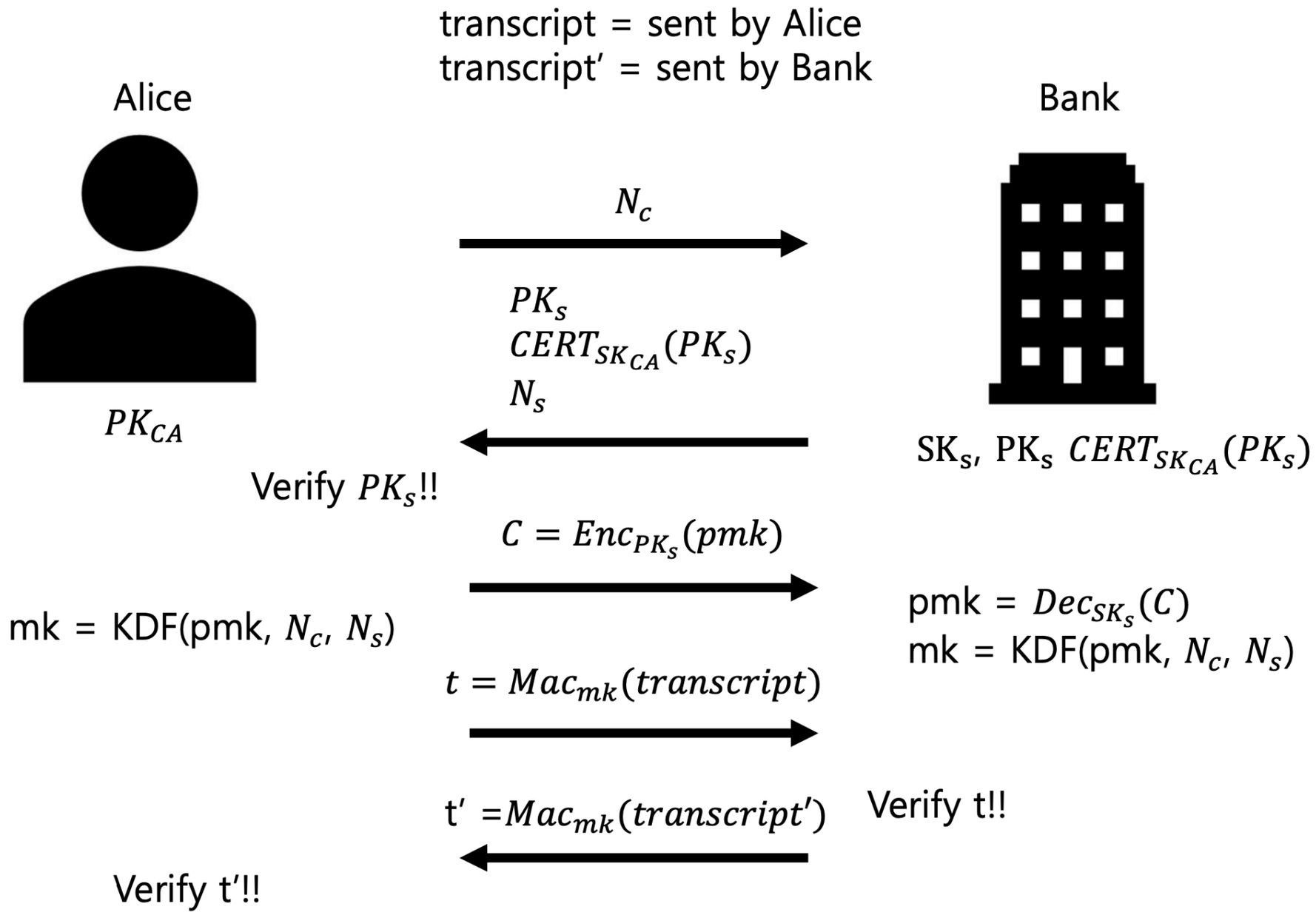
```
#include <openssl/hmac.h>
int mdLen;
EVP_MD* evpmd;
evpmd=EVP_get_digestbyname("SHA256"); //해쉬함수 선택 unsigned char md[EVP_MAX_MD_SIZE];

unsigned char *mk = “01234567890123456789012345678 9aa”;        //HMAC의 key
HMAC_CTX *hctx = HMAC_CTX_new();                                //HMAC_CTX 할당
HMAC_CTX_reset(hctx);                                           //CTX 초기화
HMAC_Init(hctx, mk, SHA256_DIGEST_LENGTH, evpmd);     //선택한 해쉬함수와 key로 초기 세팅
HMAC_Update(hctx, str, str_len);                                // update str

HMAC_Update(hctx, str, str_len);   //update str

HMAC_Final(hctx, md, &mdLen);   //결과물을 md에 return (binary_string)
```

Handshake protocol



RSA

Sign =RSAsign
Verify = RSAverify
Enc = RSAEnc

Msg=“He/0llo”
Strlen(Msg)=2
주의!!
Sign, verify, enc 기존함수는 msg_len에 대한 정보를 주지 않았습니다.
(이전엔 전부 키보드 입력으로 받은 메시지를 입력으로 넣었기 때문에 strlen()함수로 문자열의 길이 를 구할 수 있다)
하지만, handshake에선 메세지중에서 랜덤으로 뽑은 binary_string이 존재해 중간에 0이 들어갈 수 있습니다.
즉, strlen() 함수론 정확한 msg_len을 구할 수 없습니다.(strlen은 기준점으로 부터 0까지 의 길이)
따라서 매개변수로 msg_len을 보내줘야 합니다.
(함수안에서 strlen(msg) 문장을 전부 msg_len 으로 바꾸기만 하면 됩니다.)

예시)
U8* RSA_sign(U8 *sign, int msg_len, const SK *priv);
int RSA_verify(U8 *msg, int msg_len, const U8 *sign, const PK *pub);
U8 * RSA_enc(U8 *msg, int msg_len,PK *pub);
U8* RSA_dec(U8* msg, int msg_len, SK* priv);

Transcript

HMAC은 server와 client가 주고 받은 transcript(메시지)를 입력으로 넣어서 태그를 출력합니다.

알고리즘에서 c_trans1, c_trans2 가 client가 보낸 transcript 이고, s_trans1는 서버가 보낸 transcript 입니다.

※ c_trans1, c_trans2, s_trans1 의 자료형은 unsigned char

—>unsigned char c_trans1 = json_object_to_json_string(json_object *obj);

Client는 자신이 보낸 transcript(c_trans1, c_trans2)를 HMAC의 입력으로 넣어서 태그 보내고

Server도 마찬가지로 자신이 보낸 transcript(s_trans1)를 HMAC의 입력으로 넣어서 보낸다.

C_HMAC = HMAC(c_trans1||c_trans2)

S_HMAC = HMAC(s_trans1)

RSA_SCHEME

: hex string (only “0~9” “A~F”)
: binary string (all ASCII CODE)
: BIGNUM

Client

$N_c \leftarrow random\ 256\ bits$

$PK_s = P\_N, P\_E$
$PK_{CA} = CA\_N, CA\_E$
p_n, p_e = PK_s

$H \leftarrow HASH($ p_n || p_e $)$
if $RSAVeft_{PK_{CA}}(H, CERT) == 1$:
 printf(“cert verify ok !\n”)
else
 printf(“cert verify fali !\n”)

$pmk \leftarrow random\ 256bits$
$Mk = HASH(pmk||N_c||N_s)$
$C \leftarrow RSAEnc_{PK_s}(pmk)$

c_trans1= { “scheme” : “RSA_SCHEME”, “N”: “FF”}

s_trans1 =
{
“P_N” : “FF”, “P_E” : “FF”,
“CA_N” : “FF”, “CA_E” : “FF”,
“CERT” : “FF”, “N” : “FF”
}

c_trans2= { “C” : “FF” }

Server

$N, e, sk \leftarrow RSA\_setup$
$CA\_N, CA\_E, CA\_SK \leftarrow RSA\_setup$
$H \leftarrow HASH(N||e)$
$CERT \leftarrow RSAsign_{SK_{CA}}(H)$
$N_s \leftarrow random\ 256bits$

RSA_SCHEME

: hex string (only “0~9” “A~F”)
: binary string (all ASCII CODE)
: BIGNUM

Client

$C\_MAC = HMAC_{Mk}$(c_trans1 || c_trans2)

{ “MAC” : “FF” }

MAC` $\leftarrow MHAC_{Mk}$(s_trans1)
S_MAC = MAC

if S_MAC == MAC`
 printf(“Mac Verification Success\n”)
else
 printf(“Mac Verification Fail”)

$k_c \leftarrow HASH(0||mk)$
$k_c' \leftarrow HASH(1||mk)$
$k_s \leftarrow HASH(2||mk)$ // AES enc key
$k_s' \leftarrow HASH(3||Mk)$ // HMAC key

1 byte

Server

$pmk \leftarrow RSADec_{SK_s}(C)$
$Mk \leftarrow HASH(pmk||N_c||N_s)$
MAC` $\leftarrow HMAC_{Mk}$(c_trans1 || c_trans2)

if C_MAC == MAC`
 printf(“Mac Verification Success\n”)
else
 printf(“Mac Verification Fail”)

{ “MAC” : “FF” }

S_MAC $\leftarrow HMAC_{Mk}$(s_trans1)

$k_c \leftarrow HASH(0||mk)$ // AES enc key
$k_c' \leftarrow HASH(1||mk)$ // HMAC key
$k_s \leftarrow HASH(2||mk)$
$k_s' \leftarrow HASH(3||Mk)$

RSA_SCHEME

: hex string (only “0~9” “A~F”)
: binary string (all ASCII CODE)
: BIGNUM

Client

Msg = “How are u?”
AES_set_encrypt_key(ks, 128, s_enc_key);
CT = $AES\_encrypt_{s\_enc\_key}$(Msg)
CT_MAC = $HMAC_{k_s'}$(CT)

{ “CT” : “FF”, MAC: “FF” }

MAC’ = $HMAC_{k_c'}$(CT)
If MAC’ == CT_MAC
 printf(“Mac Verification Success\n”)
 AES_set_decrypt_key(kc, 128, c_dec_key);
 decMsg ← $AES\_decrypt_{c\_dec\_key}$(CT)
 Print dec Msg !!
else
 printf(“Mac Verification Fail”)

Server

MAC’ = $HMAC_{k_s'}$(CT)
If MAC’ == CT_MAC
 printf(“Mac Verification Success\n”)
 print decrypted message
else
 printf(“Mac Verification Fail”)

{ “CT” : “FF”, MAC: “FF” }

Msg = “I’m good!”
CT = $AES\_encrypt_{k_c}$(Msg)
CT_MAC = $HMAC_{k_c'}$(CT)

주의사항

1. Server 실행파일을 다운받은 후 실행을 했을 때
   bash: ./server: 허가 거부 란 오류 메세지가 뜬다면 chmod 777 server 명령어를 쳐주세요.

2. 실행시, ./server 로 명령어를 치면 됩니다. 내부포트가 **29292**로 고정되어 있습니다.

3. Key값이 다를 경우 간단한 오류메세지를 출력해줍니다.

4. 내부 포트번호는 바뀌지 않지만 외부 포트번호와 IP는 실시간으로 변동되니 실행이 안될 경우 바뀌었는지 확인해 주세요.