국방망 보안채널 구현에 관한 연구

A Study of a Secure Channel Implementation on the Military Computer Network

이 준*
Lee, Jun

ABSTRACT

In this paper we suggest a protocol and an algorithm which connect a secure channel between a server and a client over a TCP layer. To make key exchange time the most quickly, the protocol adopts ECC Diffie Hellman(ECCDH) algorithm. And the protocol does not use Hello message for key exchanges and state changes. We also implement this protocol over an open TCP/IP program and check the secure channel connecting time over the military computer network. The suggested protocol could be practically used on the military computer network without a hardware implementation.

주요기술용어(주제어) : Key Exchange, ECCDH, Secure Channel

1. 머리말

현대전의 개념은 전장에 참여하는 모두가 임무에 따른 정보를 국방망을 통해 정확히 공유함으로 전력을 효율적으로 극대화하는 것이다. 특히 주요 임무 중 타 부대와 협의를 통해 중대한 결심을 하는 지휘관의 경우 대화에 대한 보안이 요구된다. 하지만 TCP/IP에서 운영되는 국방망은 특성상 악의적인 제 3자의 도청을 막을 수 없다. 또한 작전지역의 유동성에 따라 국방망이 유선망만으로 구성이 곤란하여 무선LAN과 혼합하여 구성하게 될 경우 무선 AP(Access Point)에서의 도청은 심각하게 취약하므로 이에 대한 보안 대책도 필요하다. 본 연구는 국방망에서 일대일로 연결하는 서버와 클라이언트 간의 보안 채널을 구축하기 위한 효율적인 프로토콜과 알고리즘을 제안하고, 보안성이 없는 TCP/IP 공개 프로그램 위에 제안된 프로토콜을 구현하고 실험함으로 국방망에서 실시간 보안 채널형성 가능을 보인다.

† 2008년 4월 3일 접수~2008년 5월 16일 게재승인
* 공군사관학교(Korea Air Force Academy)
주저자 이메일 : jlee@afa.ac.kr

2. 관련연구

서버와 클라이언트를 연결하는 보안 채널 형성에서 가장 널리 사용되는 기술은 IP계층 위와 TCP상에서 동작하는 것들로 모두 공개키 알고리즘을 기반으로 하고 있으며, 인증서를 이용한 PKI(Public Key Infrastructure)의 요소를 포함하고 있지만 다음과 같은 취약점을 내포하고 있다.

가. IPsec

IPsec은 인터넷 상에서 VPN(Virtual Private Network)을 구현하는데 개발된 프로토콜이다. 네트

워크상의 IP 계층에서의 보안에 중점을 두었으며 클라이언트와 VPN 서버 간에만 보안채널을 형성한다. 일단 IPsec VPN이 형성되면 internal network 상의 돌아다니는 모든 암호화되지 않은 data를 획득할 수 있어 진정한 end-to-end security는 제공하지 못하는 단점이 있다. IPsec은 통신하는 서버에 맞는 특정 Client S/W를 인스톨해야 한다. 이것이 호환성에 걸림돌이 된다[1]. 또한 IPsec은 IP 계층에서 data를 암호화/복호화해서 보안성을 강화하는 프로토콜이지만 모든 라우터, 스위치 등의 장비가 모두 IPsec을 지원하지 않기 때문에 IPsec의 모든 기능이 구현되기 힘들다[2].

나. TLS/SSL

네트워크상의 TCP 층의 보안에 중점을 두었으며 서버와 클라이언트 간에만 보안채널을 형성한다. 일단 TLS/SSL VPN이 형성되면 network 상의 돌아다니는 모든 data는 암호화되었으므로 자격이 없는 제3자는 획득 할 수 없다. 진정한 end-to-end security를 제공한다. 하지만 TLS/SSL은 서버와 클라이언트 세션 설정과정에서 Cipher suite rollback attack, Key exchange algorithm rollback, Dropping the change cipher spec message attack 등의 취약성[3]을 드러낸다. 이 모두는 Hello message 전송과정에서 발생하는 취약점으로 공개키 암호와 규격을 다양하게 지정할 수 있는 문제와 state 변화를 자동적으로 전환하지 않음에 있다.

다. 기타

VoIP 비화기 프로토콜[4]에 대한 제안은 별도의 하드웨어를 요구하지 않으면서 ECC ElGamal 알고리즘을 사용하여 서버와 클라이언트가 블록암호에 사용할 비밀키를 합의한 후 개인키 소유를 확인한다. 확인하는 절차는 블록암호를 통해 받은 인증키를 확인한 결과와 ECC 공개키로부터 유도한 해시 값이 일치하는 경우에만 보안대화 채널을 형성한다.

TCP 계층 위에서 작동하는 이 프로토콜은 SSL처럼 Hello message를 교환하여 키를 합의하는 대신 절차적으로 진행하여 키를 합의함으로 man in middle의 문제를 해결할 수 있는 장점을 제시하고 있으나 실제적으로 구현되지 않아 실용적 가능성을 가늠하기 어렵다. 또한 제안된 프로토콜은 ElGamal 알고리즘보다 Diffie Hellman 키교환 알고리즘이 계산량이 작아 속도가 빠른 이점을 이용하지 못하고 있다.

이에 대한 개선 프로토콜을 다음과 같이 제안하고 구현한다.

3. 프로토콜

가. 유한체 F_p위에서의 ECC 공개키

유한체 F_p위에서의 타원곡선 군의 구성 요소는 다음과 같이 정의된다.

$$\{(x,y) \mid y^2 \bmod p = (x^3+ax+b) \bmod p\} \cup O$$
$$O(\text{infinite}) \quad a,b \in F_p \qquad 4a^3+27b^2 \neq 0 \bmod p$$

위와 같은 타원방정식에 대한 ECC의 공개키 **P**는 **P** = { p, a, b, Q, kQ }로 구성된다. 이때 k는 개인키로 비공개한다. ECC 구현은 여러 종류가 있으나 추가적인 하드웨어 구현 없이 소프트웨어 구현은 소수 p가 언더라잉인 유한체 위에서 구현하는 것이 효율적이다[5].

1) ECC-DH 암호화

① 공개키 kQ에 m을 곱하여 S1 = m(kQ)을 얻는다.
② 공개키 Q에 m을 곱하여 S2 = mQ를 얻는다.
③ 송신자는 S2를 공개자(수신자)에게 송신한다.
④ 수신자는 S1을 공유키로 사용한다.

2) ECC-DH 복호화

① 공개자는 S2를 수신한다.
② 공개자 개인키 k를 S2에 곱하여 k(S2)를 얻는다.
③ 공개자는 k(S2)을 얻음으로 송신자와 동일한 키(S1)를 공유한다.

나. 인증서

1) 인증서 구조

인증서는 네 부분으로 구성된다. 공개키 P, 공개키 소유자의 개인정보 P_I, 인증서 유효시간 V_Time과

위 세 가지 사항을 해시함수로 처리한 값에 대한 인증기관의 서명된 인증키 A로 구성된다. 인증기관의 서명은 일반적으로 RSA를 사용하며 1024비트를 권한다[6]. 인증서는 인증키를 제외한 세 부분을 문자열로 구성하여 해시함수로 처리하고 그 값을 인증기관의 개인키로 서명하여 발행한다.

$$str = q + a + b + G.x + G.y + kG.x + kG.y + P_I + V\_time \quad (1)$$

$$H = h(str) \quad (2)$$

$$A = H^e \quad (3)$$

2) 인증키 신청 및 발급

인증기관은 해시 값 H를 만드는 절차와 인증기관 공개키 d로부터 인증키를 확인하는 절차($H = A^d$)를 공개한다.

① 신청자는 자신의 공개키 P를 인증기관에 제출한다. 이때 신청자는 개인키 k를 공개하지 않는다.
② 인증기관은 신청 자격, 개인정보를 확인한 후 적격자에 대해서 인증키에 포함될 공개키 P, 개인정보 P_I 및 인증서 유효기간 V_Time을 결정한다.
③ 신청자의 정보 및 유효기간에 대하여 식 (1)을 사용하여 문자열 Str을 만든다.
④ 문자열 Str에 대해 식 (2)로 해시 값 H를 구한다.
⑤ 인증기관 개인키 e로 식 (3)의 인증키 A를 구한다.
⑥ 인증서 D = { P, P_I, V_Time, A }를 발급한다.

3) 인증서 확인

대화채널 형성 초기단계에서 서버와 클라이언트는 서로에게 자신의 P, P_I, V_Time를 제출하여 키 교환 전에 인증서 D의 유효기간과 개인정보를 확인받는다.

키 교환으로 확정된 송수신 키를 블록암호 비밀키로 사용하여 인증키를 교환함으로 공개키에 대한 개인키 소유를 확인하고, 서명된 인증키와 식 (2)로부터 계산된 해시 값의 일치를 확인하여 공개키 소유, 개인정보 확인, 유효기간을 실질적으로 확인한다.

4) 인증서 재발급

D의 유효기간이 경과하거나 개인정보 P_I가 변경된 경우 재발급을 요청하고, 인증기관은 신청자에 관한 개인정보와 유효일자를 수정한 후 발급하는 절차에 따라서 D를 재발급한다.

다. 프로토콜

서버를 S, 클라이언트를 C로 표기하고, 서버의 공개키와 인증서는 아래첨자 S를 사용하여 다음과 같이 표시한다. 공개키 $\mathbf{P_S}$ = { p_s, a_s, b_s, Q_S, k_sQ_S }, 인증키 A_S, 인증서 D_S, 개인키는 k_s로 표시한다. 클라이언트 역시 같은 방법으로 아래첨자 C를 사용하여 표시한다.

① C가 S에 접속하면 S는 인증서 D_S에서 A_S를 제외한 정보를 전송한다.
② C는 D_S의 유효기간과 개인정보를 우선 확인한 후 유효하면, 인증서 D_C에서 A_C를 제외한 정보를 S에게 전송한다. 전송 후 난수(random number) m_c를 생성하여 S의 공개키 Q_S와 k_sQ_S에 m_c를 곱한다.
③ S는 C로부터 수신한 내용에서 D_C 유효기간과 개인정보를 우선 확인한다. 인증서 기간이 유효하면 S는 난수 m_s를 생성한 후, C의 공개키 Q_C와 k_cQ_C에 m_s를 곱한다. S는 $m_s(Q_C)$를 C에 전송한다. 이때 S는 $m_s(k_cQ_C)$를 기억하고 C로 보내는 블록암호의 비밀키로 사용한다.
④ C는 S로부터 m_sQ_C 수신한 후 ②단계에서 계산한 m_cQ_S를 S에게 보낸다. 그 후 수신한 m_sQ_C에 자신의 개인키 k_c를 곱하여 $k_c(m_sQ_C)$를 얻어 S로부터 받는 블록암호의 비밀키로 사용한다.
⑤ S는 수신한 m_cQ_S에 k_s를 곱하여 $k_s(m_cQ_S)$얻는다. S는 C로부터 받는 블록암호의 비밀키로 $k_s(m_cQ_S)$를 사용한다.
⑥ C는 인증키 A_C를 비밀키 $m_c(k_sQ_S)$로 암호화하여 S에게 전송하고, S역시 인증키 A_S를 대칭키 $m_s(k_cQ_C)$로 암호화하여 C에게 전송한다.
⑦ 암호문을 수신한 S와 C는 각각의 비밀키 $k_s(m_cQ_S)$, $k_c(m_sQ_C)$로 암호문을 해독한 평문 A_S와 A_C를 얻는다.

⑧ S와 C는 각각 ①과 ②단계에서 얻은 정보에서 해시 값 H_C와 H_S를 얻은 후 각각 인증기관의 공개키 d를 이용해서 $H_C = A_C^d$와 $H_S = A_S^d$ 일치를 확인한다. 일치하지 않을 경우 연결을 차단한다.

⑨ 인증키가 유효할 경우 S와 C는 각각의 비밀키 $k_s(m_cQ_S)$, $k_c(m_sQ_C)$로 대칭키 암호화한 메시지를 주고받는 절차로 보안채널을 형성한다.

라. 알고리즘

```
void main(state)
{
int mode = 0                    //클라이언트 시작 모드
socket svr_socket, clnt_socket  // 소켓 선언
if ( state == server )
{       svr_socket = socket(초기화변수)   // 서버 소켓 생성
        bind(서버 관련변수)     // 소켓에 주소할당
        listen(서버와 대기큐)   // 연결요청 대기상태 진입
        clnt_socket = accept(서버와 연결할 주소)// 연결요청 수락
        close(svr_socket)       // 다른 클라이언트의 서버 연결 요청 차단
        msg = str(PS)           // 서버 정보(공개키, 개인정보, 유효기간)
        send(msg)               // 서버 정보를 클라이언트에게 보내기
        mode = 1                // 서버 시작 모드}
else
{         clnt_socket=socket(초기화변수)  // 서버접속 소켓 생성
          connect(접속소켓과 서버주소정보)          // 서버로 연결요청
}
while (1)
  {
    if(receive(msg)==TRUE) // 패킷 수신
        {
          switch(mode)
          {
                case 0 : PS = msg     // 서버 정보 받기
                        if( PS 유효기간 )
                        {
                              msg = str(PC)              // 클라이언트 정보(공개키, 개인정보, 유효기간)
                           send(msg)                  // 클라이언트 정보를 서버에게 보내기
                           msg_S = Enc_PS(PS)         // 서버 공개키 DH 암호화(mckskQs, mcQs)
                           mode = mode + 2
                          }
                        else
                         close(clnt_socket)           // 연결종료
                         break
                case 1 : PC = msg     // 클라이언트 공개키 받기
                        if( PC 유효기간 )
                        {
                             msg = Enc_Pc(MC)          // 클라이언트 공개키 DH 암호화(mskcQc, msQc)
                           send(msg)
                           mode = mode + 2
                          }
                        else
                         close(clnt_socket)           // 연결종료
                         break
                case 2 : MC = Enc_PC-1(msg)           // 클라이언트 공개키 DH 복호화
                         send(msg_S)                 // 서버의 공개키 DH 암호화 보내기
                         mode = mode + 2
```

```
                    break
            case 3 : MS = Enc_PS-1(msg)              // 서버 공개키 DH 복호화
                    send( )                          // 클라이언트에게 빈 메시지 보내기
                    mode = mode + 2
                    break
            case 4 : msg = Enc_Sec(mc(ksQs), AC)     // 대칭키로 클라이언트 인증키 암호화
                    send(msg)                        // 서버에게 암호화한 인증키 보내기
                    mode = mode + 2
                    break
            case 5 : AC = Enc_Sec-1(ks(mcQs), msg)   // 서버가 대칭키로 복호화하여 클라이언트 인증키 받기
                    msg = Enc_Sec(ms(kcQc), AS)
                    send(msg)
                    Hc = Hash(Pc+namec)
                            if ( AC is not correct )
                                    close(clnt_socket) // 연결종료
                            else
                                    mode = 100                  // 키 교환 완료
                    break
            case 6 : AS = Enc_Sec-1(kc(msQc), msg)   // 클라이언트가 대칭키로 복호화하여 서버 인증키 받기
                    HS = Hash(PS+nameS)
                            if ( AS is not correct )
                                    close(clnt_socket) // 연결종료
                            else
                                    mode = 100      // 키 교환 완료
                    break
            case 100 : if ( state == server )        // 수신 비밀키 선택
                            key = ks(mcQs)
                    else
                            key = kc(msQc)
                    msg = Enc_Sec-1(key, msg)   // 비밀키로 복호화하여 받기

        }
    }
  else
    {
        if(mode == 100 and T is not empty)
        {       if ( state == server )                  // 송신 비밀키 선택
                    key = ms(kcQc)
                else
                    key = mc(ksQs)
                msg = Enc_Sec(key, T)                   // 비밀키로 암호화하여 보내기
                send(msg)   // 패킷 송신
        }
    }
  if (close(clnt_socket)==TRUE ) break;   // 연결종료
 }
}
```

4. 구현

공개된 TCP/IP 채팅 프로그램[7] 위에 160비트 ECC[8], 1024비트 RSA[9]와 표준 블록암호로 개발된 ARIA[10], 공개된 해시 함수 MD5[11]를 포함시켜 제안된 프로토콜을 Visual C++로 구현[12]하였다. 본 구현의 주요 목표는 키교환, 인증키 인증, 보안채널 형성 시간의 실용성과 보안채널에서 전송되는 메시지 확인이다.

가. 실험장비

실험장비는 일반적인 컴퓨터를 활용하였다. 장비성능은 윈도우XP, Intel Core(TM)2 CPU T7200 @2.00GHz, 1.99GHz 0.99GB RAM이다.

나. 인증기관

인증서에 표기되는 모든 수는 2^{16}진법으로 표기한다.

1) 공개키와 개인키

인증기관의 공개키와 개인키는 1024비트이고, 기존의 활용된 수를 사용하였다[13]. 이 때 인증기관의 공개키는 유사소수 512비트를 택하였다.

2) 인증서 표현

인증서 공개키는 160비트 ECC[14]에서 개발된 수를 활용하였으며, 개인정보는 유니코드로 표기하였고, 유효기간은 함수 c_time을 이용할 수 있도록 1970년 1월 1일 0시를 기준으로 유효기간까지 초단위로 표시했다.

3) 인증키

인증키는 수식 (1)(2)(3) 절차에 따라서 작성하였다. 실험장비에서 인증서를 확인하는 10,000번 반복실험에서 1회당 소요되는 시간은 0.3429sec이다. 참가자 2인에 대하여 발행된 인증서는 표 1과 표 2이다.

다. 참가자

서버와 클라이언트 구분은 접속을 요구하는 측이 클라이언트 C가 되고 접속을 수락하는 측이 서버 S가 된다. 참가자는 접속하는 상황에 따라서 서버가 될 수 있고 클라이언트가 될 수 있다. 실험에는 공개키와 개인키, 참가자 개인정보를 임의로 선택하여 서버와 클라이언트 대표로 하였다.

인증키는 공개키, 개인정보, 유효기간을 해시함수로 처리하므로 개인정보 길이는 제한을 받지 않는다. 대표로 예시된 인증서에서 참가자의 개인정보는 직책과 이름으로 구분하였다.

참가자 1의 개인정보는 수군제독 이순신, 인증서 유효기간은 2012년 1월 1일 0시까지이며, 참가자 2는 육군장수 강감찬이고 인증서 유효기간은 2009년 5월 5일 0시까지이다.

1) 참가자 1의 인증서와 개인키

[표 1] 참가자1 인증서

* : V_Time

p	6526 0286 0642 498a 3a71 7c74 5bcc 779a 7e50 2af1
a	02cf 1636 6af7 0a34 1c38 6294 25a6 18ae 6921 0072
b	4f6b 06c3 108c 2e0c 1bd4 4c26 1e70 0af0 6be3 2d34
G.x	20f7 1669 11dc 0ccc 0ca8 3235 3072 19d8 616b 288e
G.y	548c d7ca 841b 494a a86d 258e acdb 293e c144 8bfd
kG.x	1d54 4580 818e 1e90 c59e edab 3aec 509c 6202 4822
kG.y	33e6 356b eb7e 19bf c561 8981 4b4e bfde b4c8 ff45
P_I	c218 ad70 c81c b3c5 20 c774 c21c c2e0
*	4eff 2370
A	9bae 6911 ae39 2bc7 b941 37ad 4c9 bf60 6fd0 9b8c 1f83 ead2 3e2b 3d31 a44e 86b2 7379 6cb8 e0ff 67b8 514f b29a babe b7bf 1723 6802 1b2 5aeb 87ba ad3c 1bdb 4d46 175b 3bf1 121e d9ee 2a35 af3b 7763 fbf8 ad 3c0c ab84 2a05 148a 6996 987 197f c565 7a21 e164 8ada 36b7 1fdc ecf5 7a3c e922 1a6a 9a16 a786 116a fb5e cfbe 1bf5

k = 15ab 1208 1ec8 7387 55ed 0896 09b1 4247 76da 3ff7 (개인키)

2) 참가자 2의 인증서와 개인키

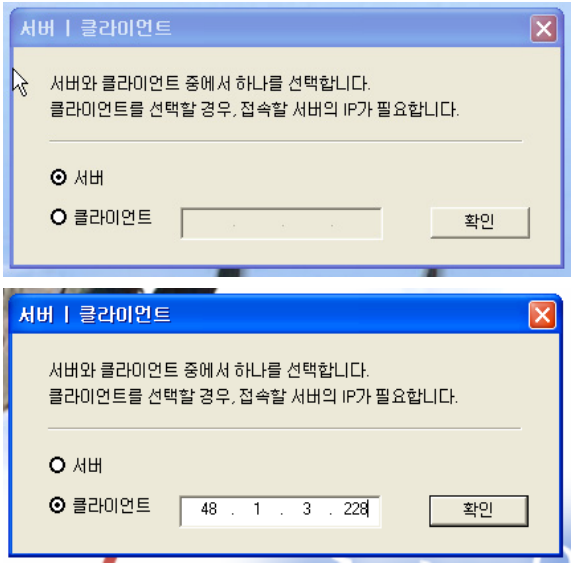
[표 2] 참가자2 인증서

* : V_Time

p	12cf 1216 00e4 39cd 2ef2 70f3 6bb7 4604 0059 7cab
a	0964 0cf8 565a 196f 6989 793e 1a93 523a 1abc 0caa
b	aba 288c 6352 1d03 35f2 7bc1 116c 31c8 479d 49c0
G.x	1283 2707 5a99 0983 6d98 3339 4363 07b7 3dca 72b6
G.y	93a 006e 27a2 0069c 52f5 a566 3f49 40bf c488 b3aa
kG.x	0b2c b506 810d 46e4 63b7 0cec 24b7 9aac 4dbd 0230
kG.y	01cd 8c7d e8b6 9d2b d48f d7f2 e298 356b 2e7e 8266
P_I	c721 ad70 c7a5 c218 20 ac15 ac10 cc2c
*	49ff 2f0
A	8c5c 401b de7c bbbe 0276 9d09 0655 9b43 53b5 0a17 d078 36a5 9260 566e 2ea1 a059 eb60 aa1b c226 5bc0 e029 86aa 2863 2bff 6d9a 0c26 daf1 cca6 db5e 26f2 c495 2667 6e1d 6b0d aa99 093e fff2 ee9f b1b1 0dfc 6f54 f8bb ed4d 24ca fad8 9030 46e3 64c8 6556 908b 0b2e 1642 61d5 9a3a 0a08 10bd 058c 2b99 56c2 ee64 4b8a e71f b063 13e6

k = 051e 40c9 5de2 0dc0 7588 546c 112a 3ccc 511b 4156 (개인키)

라. 실험



[그림 1] 초기 접속

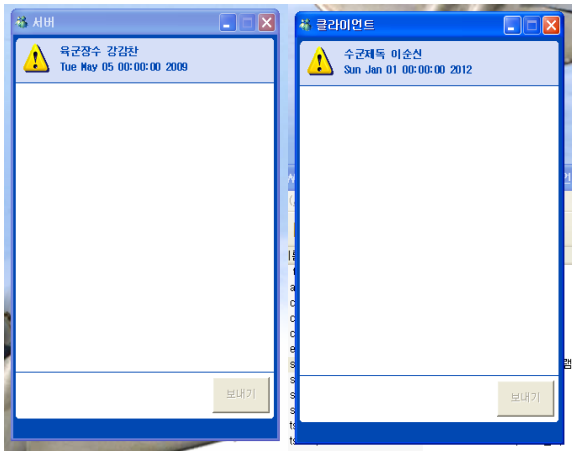
공개된 프로그램과 프로토콜을 구현한 보안 프로그램 사이에 외형적 차이는 없다. 그림 1은 초기접속으로 같은 외형을 보인다. 프로토콜이 적용이 안 된 공개 프로그램은 초기 접속 후 보안점검 없이 그림 2에서 개인정보와 유효기간이 표시되지 않은 대화창을 시작한다.

프로토콜이 적용된 프로그램의 외형상 차이는 그림 2~4에서 대화창 위에 상대편 개인정보와 인증서 유효기간이 표시된다. 하지만 TCP층을 통해 교환되는 정보는 다음과 같이 차이가 크다.

초기접속 그림 1처럼 선택에 따라서 서버/클라이언트로 결정된다. 클라이언트는 접속을 원하는 서버의 IP를 기입하여 연결한다. 이때 서버와 클라이언트는 프로토콜에서 ②~③단계를 실행함으로 상대방의 공개키, 인증서의 개인정보, 유효기간을 받아 인증서 유효여부를 결정한다.

상대편 인증서기간이 유효한 경우 그림 2와 같이 화면이 바뀌면서 대화창 윗부분에 상대편 개인정보와 인증서 유효기간이 표시된다. 만약 인증서 유효기간이 경과한 경우 더 이상 접속은 진행되지 않으며 그림 4와 같이 자동적으로 접속이 차단된다.

서버와 클라이언트는 프로토콜 ④~⑤단계를 실행하면서 수신키와 송신키를 각각 공유한다. 공유한 수신키와 송신키로 ARIA의 round와 비밀키 크기를 결정하여 자신의 인증키를 상대에게 보낸다.

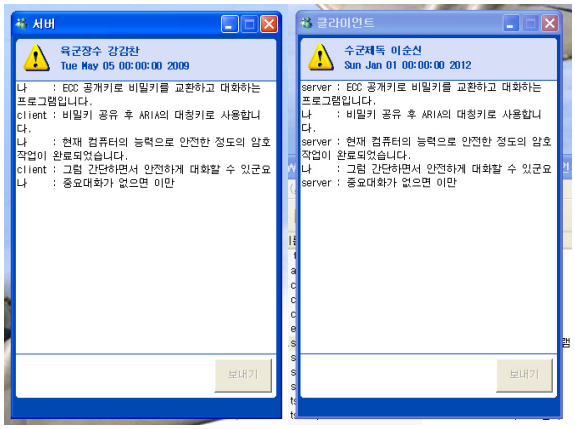


[그림 2] 접속자 정보표시

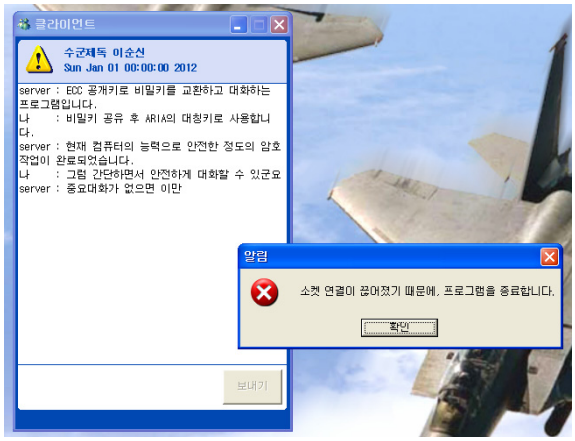
②~③단계에서 얻은 상대편 공유키, 개인정보, 유효기간을 문자열로 종합하여 해시함수 MD5로 처리하고 인증키를 인증기관 공개키로 확인하는 프로토콜 ⑥~⑧단계를 실행한다. 이때 인증키로 계산된 결과와 해시 값이 일치하지 않으면 그림 4와 같이 자동적으로 연결이 차단된다. 연결차단은 인증서를 부당한 방법으로 얻었지만 개인키 소유를 증명하지 못함에 공개키의 진정한 소유자가 아님을 의미한다. 프로토콜 ⑨단계는 서버와 클라이언트 사이에 송수신 비밀키를 공유하고 상대방의 인증키를 확인함으로 인증서의 내용이 인증기관이 서명한 것임을 확인한다. 그림 3은 인증키 확인 후 서버와 클라이언트가 교환된 키로 보안채널을 형성함과 실시간 실행을 보인다.

대화 종료 후 혹은 서버와 클라이언트 사이에 공개키 소유를 증명하지 못하거나, 인증서의 유효성을 증명하지 못한 경우 서버와 클라이언트는 접속이 종료되는 데 이 때 그림 4처럼 종료한다.

위와 같은 국방망 실험에서 서버와 클라이언트가 접속하여 그림 1에서 시작하여 프로토콜에 따른 안전한 보안 채널이 형성되는 그림 3까지 소요된 시간은 1초 이하이다.



[그림 3] 서버와 클라이언트 보안대화



[그림 4] 상대방 종료화면

마. 안정성 검토

1) 도청

도청이 가능한 경우는 블록암호의 송수신 양방향 비밀키 모두를 확보한 경우이다. C와 S가 접속할 때 공개키, 개인정보, 유효기간은 제3자에게 노출할 수 있지만 제3자는 개인키 ks, kc를 알 수 없으므로 송수신용 비밀키 모두를 확보할 수 없다. 수신용 비밀키가 없는 상태에서 상대방의 인증키를 받는 경우 해시 값과 인증키는 일치하지 않아 보안채널은 자동적으로 차단된다. 보안채널 형성은 별도의 메시지 전달 없이 자동적 상태변화에 의해 형성됨으로 Dropping the change cipher spec message attack 문제는 없다.

2) man in the middle

SSL 2.0은 HELLO 메시지 내에서 cipher suite의 리스트를 수정함으로써 공격자는 서버와 클라이언트 사이에 40비트 암호를 사용하도록 할 수 있는 Cipher suite rollback attack 문제점이 있으나[15] 제안된 프로토콜은 공개키 및 대칭키 암호의 종류와 크기가 고정되었으므로 공격의 여지가 없다. 또한 공개키를 160비트 ECC로 고정함으로 Key exchange algorithm rollback을 막을 수 있다.

3) 비밀키 재사용

접속 때마다 S와 C는 각각 새로운 난수 m_S와 m_C를 생성하여 상대방의 공개키에 곱한 후 상대에게 보낸다. 접속하는 상대가 다를 경우 상대방이 제공하는 공개키가 다르고, 난수가 다르므로 동일한 키를 사용할 가능성은 없다. 같은 상대에게 다시 접속할 경우에도 새로운 난수를 생성함으로 송수신 비밀키 재사용 가능성은 없다.

바. 효율성 검토

1) 인증

서버와 클라이언트 각자는 보안채널 형성을 위한 초기접속 시 인증서가 제공하는 개인정보와 유효기간을 간단히 확인한 후 ECCDH로 송수신 비밀키를 교환한다. 비밀키 교환 후 블록 암호화된 인증키 접속을 통해 개인키 소유를 확인한다. 이 절차로 서버는 자신과 접속을 원하는 모든 잠재적 클라이언트들의 id와 password를 별도로 저장할 필요가 없으며, 인증서 저장을 위한 DS(Directory Server)가 필요 없게 된다.

2) 키 공유 신속성

1024비트 RSA(공개키 512비트) 인증키 서명확인을 10000번 실시하는 데 57분 9초가 소요되었다. 인증시간은 0.3429sec/1회 이다. 인증시간을 고려하여 Diffie-Hellman 암호화 알고리즘으로 키를 교환 할 경우 512비트 규모의 난수를 발생하여 primitive와 개인키로 암호화된 primitive를 암호화하고 복호화에 인증시간의 3배(1.0287sec/1회)가 소요된다. 반면에 같은 컴퓨터에서 160비트 ECC 키 교환시간은 10000

번 실시하는 데 1시간 39분 29초가 소요되었다. ECC에서 서버와 클라이언트의 키 교환시간은 0.5969sec/1회 이다. 별도의 하드웨어가 없어도 160비트 ECC 공개키를 사용하고, 1024비트 RSA 로 인증할 경우 비밀키를 교환하고 인증하는데 1초 이하의 실시간으로 채널형성이 가능하다. 구현된 실험도 같은 결과를 보인다.

5. 맺음말

본 연구는 PKI 기반 암호화 통신의 대표적인 TLS/SSL을 적용하여 보안채널을 형성하는 과정에서 발생하는 취약점을 개선하는 프로토콜과 알고리즘을 제안하고 구현하였다. 공개키 암호의 종류과 규격을 다양하게 선택하는 대신 단위 비트 당 암호화 강도가 높은 160비트 ECC로 고정하고, 보안채널 형성과정의 state의 변화는 서버와 클라이언트 내부의 mode 변수를 자동적으로 전환함으로 TLS/SSL의 Hello message 전송과정에서 발생하는 취약점을 개선하였다.

제안된 알고리즘과 프로토콜은 공개 소프트웨어 위에 구현하고 국방망에서 실험함으로 별도의 하드웨어 구현 없이 보급된 국방망 컴퓨터를 이용하여도 실시간에 보안채널 형성이 가능함을 보여주었다.

보안채널 끝단에 적용되는 응용프로그램에 따라 국방망의 여러 분야에 활용될 수 있다. 또한 국방망을 유선망과 무선망으로 혼합하여 구성할 경우 별도의 단말기 인증, 데이터 암호화 등에 필요한 추가적인 장치 없이 AP에서의 도청을 방지하는 해결안이 될 것이다.

후 기

본 연구 논문은 07년도 공군사관학교 국고연구비(KAFA 07-19) 예산지원으로 수행된 결과입니다.

참 고 문 헌

[1] 윤재호 외 3인, “IPsec VPNs vs. SSL VPNs”, 정보보호학회지, 제13권 제5호, 2003년 10월.
[2] 남의석, 민병조, 김학배, “리눅스 커널 쓰레드에 기반한 SSL 웹 암호화 가속기 개발”, 정보처리학회논문지 C, 제10-C권 제6호, 2003년 10월.
[3] 김소진, 신성한, 박지환, “SSL/TLS와 WTLS의 프로토콜 취약성 분석”, 한국멀티미디어학회지, 제5권 제3호, 2001년 9월.
[4] 이준, “VoIP 비화기 설계”, 공사논문집 제57집 제2권, 2006년 9월.
[5] 구자홍, “타원곡선 암호체계 구성방법”, 공개특허2000-0061570, 대한민국 특허청, 2000.
[6] WAP Forum, Wireless Transport Layer security, version 06-April-2001, 2001.
[7] 김정훈, “TCP/IP 소켓프로그래밍”, 교학사, pp. 330～453, 2003.
[8] 이준, ECC 공개키 암호의 실용적 구현에 관한 연구, 연구보고서 KAFA 05-23, 공군사관학교, 2005.
[9] 이준, RSA 공개키 암호의 실용적 구현에 관한 연구, 연구보고서 KAFA 02-1-3-9, 공군사관학교, 2002.
[10] http://www.nsri.re.kr/ARIA/
[11] http://www.ietf.org/rfc/rfc1321.txt
[12] 이준, 국방망에서 안전한 실시간 대화에 관한 연구, 연구보고서 KAFA 07-19, 공군사관학교, 2007.
[13] 이준, 김인택, “ECC를 이용한 키분배 프로토콜”, 한국군사과학기술학회지 제10권 제2호, 2007.
[14] 이준, ECC 공개키 암호의 실용적 구현에 관한 연구, 연구보고서 KAFA 05-23, 공군사관학교, 2005.
[15] 한국정보보호학회, “차세대 네트워크 보안기술”, 생능출판사, p. 190, 2002.