신뢰성 있는 키 교환을 위한 4 단계 핸드셰이크 연결 메커니즘

송태일*, 홍충선**
경희대학교 컴퓨터공학과
e-mail:tisong@networking.khu.ac.kr*, cshong@khu.ac.kr**

A four-way handshake for reliable key exchange mechanism

Tae ill Song*, Choong Seon Hong**
Department of Computer Engineering, Kyung Hee University

요 약

유·무선의 망을 함께 사용하는 무선 단말기에서는 기존의 통신방법으로는 안전하지 못한 네트워크 환경에서의 메시지 교환의 무결성을 입증하기 어렵다. 기존 논문에서 제시한 인증메시지를 이용한 신뢰성 있는 키 교환에서는 안전하지 못한 네트워크 환경에서의 신뢰성있는 키 교환이 가능하나, 상대적으로 많은 메시지 교환 절차와 암호화로 무선단말기와 서버 사이의 통신에 단점을 가지고 있다. 본 논문은 안전하지 못한 네트워크 환경에서의 무선 단말기와 유선 서버사이의 신뢰성 있는 키 교환을 위해 상호 연결을 맺는 과정을 제시한다. 제안된 논문은 연결 요청단계, 인증 메시지 교환단계, 홉 카운팅 단계, 신뢰성 확인단계로 구성되어 있다. 연결 요청단계에서는 서버와 클라이언트 사이의 연결 요청이 이루어지며, 인증 메시지 교환단계는 인증 메시지를 이용해 상호간의 신뢰성 확인 및 Seed 값의 분배를 실시한다. 홉 카운팅 단계에서는 패킷의 TTL(Time to live)를 이용하여 메시지의 변조여부를 확인하며, 신뢰성 확인 단계는 카운팅된 값으로 메시지의 신뢰성여부를 판단하는 단계로 구성된다.

1. 서론

유・무선의 복합적인 통신 환경을 가진 모바일 기기에서는 각각의 환경에서 적용되는 보안기법으로 종단까지의 신뢰성 있는 연결을 보장하기 힘들다. 신뢰성 있는 연결은 안전한 비밀 값을 공유하기 위해 필수적인 방법이며, 기존의 유선 및 무선에서 사용된 방법은 처음부터 유・무선의 복합적인 통신환경을 고려하여 설계하지 않았기 때문에 문제점이 제기 되고 있다. 또한 성능의 제약이 있는 모바일 단말기의 경우, 에너지 효율성이 중요하다. 그에 따라 모바일 단말기는 기존의 클라이언트보다 적은 계산을 요구하는 암호화 방법 및 연결 방식이 필요하며, 종단 간의 메시지 변조 및 도청에 대한 신뢰성을 보증할 방법이 필요하다. 이에 따라 본 논문에서는 무선 단말기와 유선 서버 사이의 상호 인증 메시지를 이용한 신뢰성 있는 키 교환 연결 메커니즘 방식을 제시하고자 한다.

2. 관련연구

IEEE 802.11i[1]의 무선랜 보안은 사용자 인증 방식, 키 교환 방식, 무선구간 암호화 알고리즘을 정의한다. 이 인증방식은 무선장치, AP, 인증 서버 상호간의 통신 단계로 구성된다. 이러한 방식의 인증 개념은 무선 단말기와 AP 사이의 통신에 대한 신뢰성만 제공할 뿐 목적지까지의 신뢰성을 제공하지 못한다. 소켓을 이용한 TCP와 UDP의 통신의 경우 웹브라우저 보안 프로토콜중 하나인 SSL(Secure Socket Layer)[2]과 비교하여 간단한 세션을 맺는 과정이 가능하나, 상대적으로 보안성이 취약하다는 단점이 존재한다. 제 3자를 이용한 키 교환[3]의 경우는 클라이언트 상호간의 신뢰성 있는 키 교환은 가능하나, 신뢰성 있는 서버 혹은 제 3자의 도움이 필요하다는 전제조건이 필요하다. H. Fereidooni가 제안한 안전하지 못한 네트워크상에서의 키 교환 프로토콜[4]에서는 인증 메시지, 쿠키, 메시지 해시함수를 이용하여 종단간의 신뢰성 있는 메시지 전송 및 키 분배를 가능하게 하였으나, 연결 과정 및 종단간의 메시지 교환 절차가 복잡하며 많은 연산과정을 거친다는 단점이 있다.

3. 제안사항

본 논문에서 제안하는 사항은 2가지의 가정을 가지고 있다.

1. 모바일 단말기(이하 클라이언트)와 서버(이하 서버)사이에 연결 경로는 최초에 통신한 경로를 사용한다.

"본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT 연구센터 지원사업의 연구결과로 수행되었음" (NIPA-2010-(C1090-1031-0005))
Dr. CS Hong is corresponding author.

Client Certification Message

$CERT_{client}$= {ID_{client}, PK_{client}, $Seed_{client}$, {ID_{client}, PK_{client}, $Seed_{client}$ }PK^{-1}_{client}}

Server Certification Message

$CERT_{server}$= {ID_{server}, PK_{server}, $Seed_{server}$, {ID_{server}, PK_{server}, $Seed_{server}$}PK^{-1}_{server}}

(그림 2) 인증 메시지 형태

<표 1> 클라이언트 인증 메시지의 형태

Client
ID_{client} : 클라이언트의 ID
PK_{client} : 클라이언트의 공개키
$Seed_{client}$: 클라이언트가 생성하는 Seed 값

<표 2> 서버 인증 메시지의 형태

Server
ID_{server} : 서버의 ID
PK_{server} : 서버의 공개키 값
$Seed_{server}$: 서버가 생성하는 Seed 값

2. 클라이언트와 목적지 서버는 서로 같은 방식의 암호화 알고리즘을 사용한다.

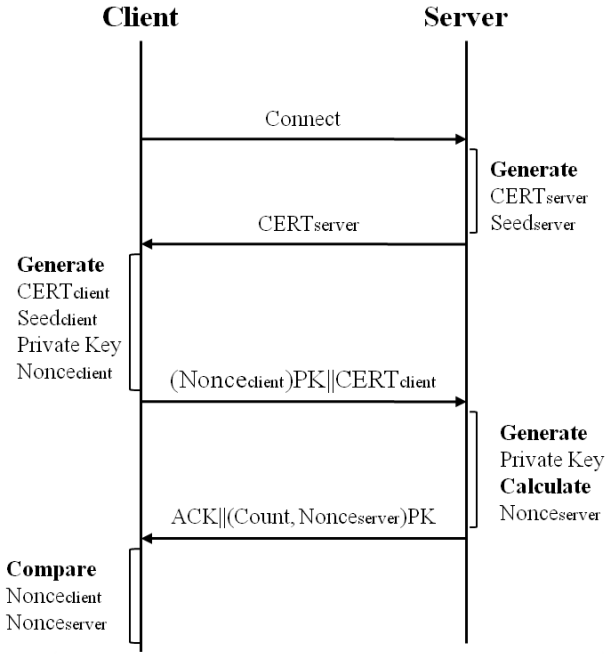
인증 메시지는 [4]의 연구내용을 바탕으로 그림1과 같이 제안하였다. 그리고 표 1과 표 2는 인증 메시지를 구성하는 요소들에 대한 설명을 나타낸다. 서버와 클라이언트 인증 메시지의 구성은 상호 인증을 위한 고유한 값의 ID, 메시지 암호화를 위한 공개키, 비밀키 생성을 위한 Seed 값과 인증 메시지의 같은 값을 공개키로 암호화 하는 암호화된 메시지로 구성된다. 암호화된 메시지는 해당 ID, 공개키, Seed 값을 공개키로 암호화하여 종단부분에서 값을 일치성을 판단하여 중간노드의 개입으로 메시지 변조 여부를 확인하여 인증 메시지의 무결성을 판단하게 된다.

3.1 연결 요청 단계

클라이언트는 서버에 접속하기 위해 서버와 상호 약속된 메시지로 연결 및 인증 요청을 한다. 이 과정에서 서버와 클라이언트는 상호간의 비밀키 값을 소유할 경우 해당 키 값으로 통신하게 되며, 만일 서버에서는 비밀키 값이 존재하지 않을 경우나, 이상여부가 판단되면 인증 메시지 생성 및 전송을 한다. 그리고 클라이언트는 비밀키 생성을 위한 $Seed_{server}$ 값을 신뢰성 있는 저장장소에 저장한다.

3.2 인증 메시지 교환단계

클라이언트는 서버로부터 전송받은 인증 메시지의 암호화된 메시지 값을 복호화 한 후 ID, 공개키, Seed 값과 비교하여 값이 일치 하지 않을 경우 서버에 재전송을 요청하며, 값이 일치할 경우 메시지 내의 Seed값을 저장한다. 그리고 클라이언트는 서버에게 신뢰성 인증을 위해 인증메시지 생성한다. 그리고 인증메시지로부터 얻은 $Seed_{server}$값과 클라이언트가 생성한 $Seed_{client}$ 값을 이용하여 비밀키 값을 도출한다. 비밀키 값 생성이 완료되면, 클라이언트는 $2^4 \sim 2^8$ 사이의 임의의 수 $Nonce_{client}$를 생성한다. 이 값은 패킷을 만들 때, 홉 카운팅을 위한 TTL 값으로 사용되며, 종단간의 메시지의 무결성을 입증하기 위해서 사용된다. 클라이언트는 생성된 $Nonce_{client}$를 패킷의 TTL의 값으로 지정 한 후, 비밀키를 이용하여 암호화 하여 인증 메시지와 함께 서버로 전송한다.



(그림 1) 신뢰성 있는 4단계 핸드셰이크 연결 과정

3.3 홉 카운팅 단계

서버는 클라이언트로부터 전송받은 메시지로부터 TTL 필드 값을 얻은 후 인증메시지와 비밀키로 암호화된 $Nonce_{client}$를 분리한다. 서버는 클라이언트와 마찬가지로 인증메시지의 무결성 여부를 판단하여 메시지의 재요청 여부를 결정한다. 인증 메시지의 무결성이 입증되면 클라이언트와 마찬가지로 서버가 가지고 있는 $Seed_{server}$값과 인증 메시지 내의 $Seed_{client}$ 값으로 비밀키를 도출한다. 그리고 이 비밀키를 이용하여 $Nonce_{client}$를 복호화 하여 수식 1과 같이 카운트 값과 $Nonce_{server}$를 생성한다.

$$Count_{server} = Nonce_{client} - TTL$$
$$Nonce_{server} = TTL$$

(수식 1) 서버의 Nonce, Count 값 계산

마지막으로 서버는 계산된 $Count_{server}$와 $Nonce_{server}$를 비밀키 값으로 암호화 한 후 ACK 메시지와 결합하고, TTL

값을 $Nonce_{client}$으로 설정하여 클라이언트로 전송한다.

3.4 신뢰성 확인단계

클라이언트는 서버로부터 전송받은 메시지로부터 TTL 값을 $Reply_Nonce_{client}$으로 저장한다. 그리고 ACK와 비밀키로 암호화된 $Count_{server}$와 $Nonce_{server}$를 분리한다. ACK의 이상여부를 확인 후, 비밀키로 $Count_{server}$와 $Nonce_{server}$를 복호화 하여 다음과 같은 비교과정을 실시한다.

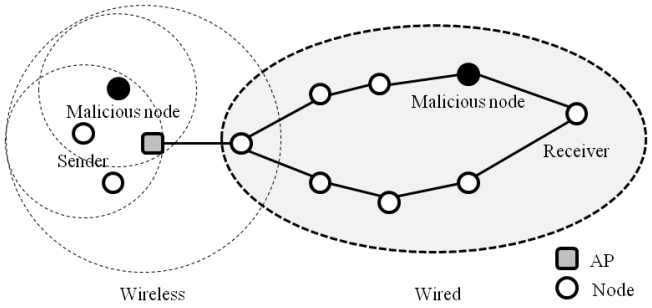
$$Compare(Nonce_{client},\ Count_{server}+Nonce_{server})$$

두 값이 일치할 경우 암호화된 $Nonce_{client}$ 값이 중간노드의 방해 없이 서버와 클라이언트 사이의 통신했다는 것을 확인 가능하다.

$$Compare(Reply\_Nonce_{client},\ Nonce_{server})$$

그리고 두 값들이 일치할 경우, 노드는 같은 경로로 메시지가 전송 되었다는 것이 추론 가능하다. 해당 결과들은 서버와 클라이언트가 중간노드의 개입 없이 정상적인 메시지의 교환이 성공하였으며, 같은 비밀키 값을 가졌다는 것을 확인 가능하다.

4. 평가

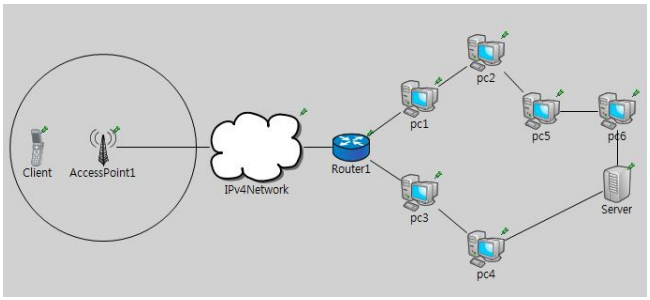


(그림 3) 클라이언트와 서버간의 가상 연결 시나리오

그림 3과 같이 클라이언트와 서버가 연결되었다고 가정을 하면, 유·무선의 악의적인 노드들은 리플레이 어택, 메시지 변조, 도청등이 가능하다. 각 유·무선 상의 메시지는 악의적인 노드가 존재하는 위쪽 경로로 이동한다고 가정한다. 먼저 상호간의 인증을 위한 인증 메시지의 변조 여부는 메시지 내의 공용키를 이용하여 암호화된 메시지와 암호화 되지 않은 메시지의 값의 일치여부에 따라 판단된다. 만약 이 값이 일치하지 않는다면 네트워크 중간 경로에서 공격자의 개입 혹은 간섭이 있다고 판단 가능하다. 패킷의 TTL을 이용하여 홉 카운팅을 이용할 경우 중계역할을 하는 악의적인 노드는 TTL 값을 변조하여 각 종단에 전송 가능하다. 그러나 TTL 값이 변조되었다 하더라도 각 종단은 자신이 전송하고 받은 TTL 값과 홉카운팅 값을 상호 비교하므로 중간 노드의 개입에 대한 의심과 신뢰성있는 메시지 전송이 되었다고 확인 가능하다. 리플레이 어택으로 인증정보를 도용한다 하더라도, 미리 분배된 키 도출 방식 및 암호화 방식과 다른 메시지가 나올 경우 중간 노드의 개입에 대한 의심이 가능하다.

5. 시뮬레이션

본 논문에서 제시하는 메커니즘을 시뮬레이션 위해서 OMNeT++[5] 상에서 아래의 그림 4, 그림 6 와 같이 모듈을 구현하였다. 무선 전송은 IEEE 802.11g의 3G망 표준 환경을 구현하였으며, 유선 전송은 100Mbps의 환경으로 표 3처럼 구성하였다. 본 논문에서는 [4]에서 제시한 인증 메커니즘과 제안한 메시지 인증 메커니즘을 비교하여 인증 메시지 속도 성능과 에너지 효율성을 평가하였다.

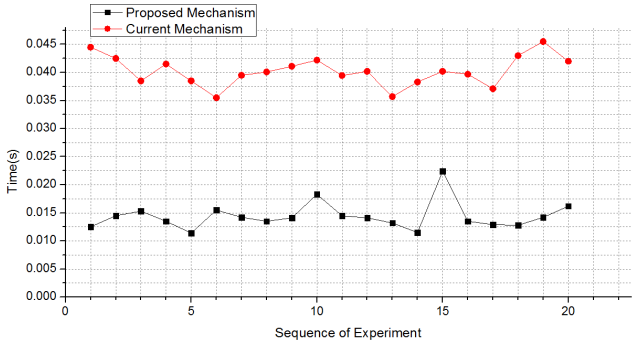


(그림 4) 인증 메시지 속도 측정 시나리오

<표 3> 유무선 구간의 시뮬레이션 환경

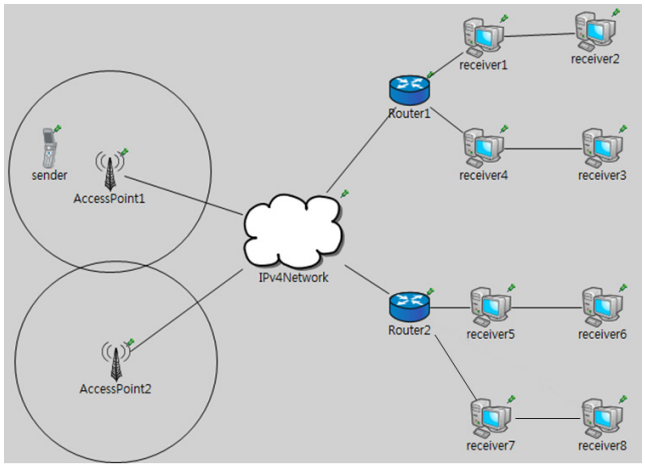
Parameters for Wireless	
Frequency	2.4GHz
Data Rate	4.5 ~ 11Mbps
Parameters for Wire	
Data Rate	100Mbps

그림 4는 서버와 클라이언트 간의 인증 메시지 속도를 비교한 시뮬레이션 환경이다. 무선 클라이언트는 AP를 이용하여 IPv4 네트워크망을 이용하여 서버까지 신뢰성 있는 메시지를 전송한다 가정 하였다. 클라이언트로부터 전송된 인증 메시지는 유·무선 네트워크를 통하여 서버로 전송되며, 시뮬레이션 횟수마다 서버로 이동하는 구간은 임의로 정하였다. 그리고 각 구간의 딜레이는 임의의 값으로 생성하였으며 동일한 환경에서의 각각의 성능을 비교하였다. 제안된 4단계 핸드셰이크 메커니즘과 [4]에서 제안한 키 교환 방식에서의 각각의 인증 메시지의 모든 생성속도는 같다고 가정하여 실험하였다.



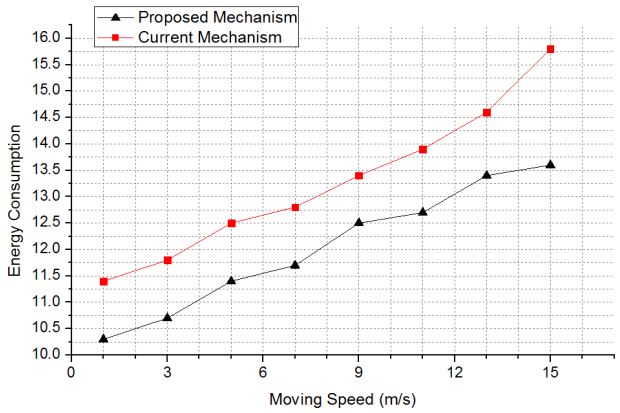
(그림 5) 메시지 인증 속도 측정

그림 5에서는 클라이언트와 서버간의 인증메시지 속도의 차이를 보여주고 있다. 제안된 메커니즘의 평균 인증 시간은 0.014초로 측정되었으며, 기존의 메커니즘은 0.039초로 제안된 메커니즘은 0.025초의 성능 개선을 보였다. 제안된 메커니즘의 장점은 기존 메커니즘에 비교하여 생성되는 메시지의 개수와 클라이언트와 서버 사이의 핸드셰이크 단계가 적다는 특징이 있다.



(그림 6) AP 이동 간 에너지 소모량 측정 시나리오

그림 6의 환경은 무선 클라이언트(이하 센더)가 지속적으로 이동하여 메시지를 생성해 서버(이하 리시버)와 세션을 맺는 환경을 구축했다. 센더는 각각의 AP 포인트로 지속적으로 이동을 하며, AP1에서는 센더가 IPv4 네트워크를 이용하여 Router1으로 이동하고, 임의 리시버에 인증 메시지를 전송하여 세션을 맺으며, AP2에서는 IPv4 네트워크를 이용하여 Router2로 이동하고, 임의의 리시버에 인증 메시지를 이용하여 세션을 맺는 방식이다. 센더가 이동하는 속도는 표 4 처럼 구성하였다.



(그림 7) 에너지 소모량 비교 측정

<표 4> 이동성 환경변수

Parameters for Mobile Device	
Mobility Type	Linear Mobility
Moving Speed	1 ~ 15 m/s

그림 7 은 센더의 이동속도에 따라 리시버까지의 신뢰성 있는 연결을 맺을 때 소모되는 에너지의 양을 측정 하였다. 제안한 메커니즘은 기존의 메커니즘과 동일하게 이동속도가 증가함에 따라 에너지 소모도 증가하는 특성을 보였다. 하지만 현재 키 교환 메커니즘 대비 에너지 소모측면에서 약 20%의 성능향상을 보였다.

6. 결론

본 논문에서 제안하는 메커니즘은 인증 메시지와 패킷의 TTL 옵션을 이용한 핸드셰이크 방법을 사용 할 경우 클라이언트는 서버와의 통신에서 경량화와 안전한 인증이 가능하며, 상호간의 신뢰성 있는 Seed값 교환을 이용하여 키 값을 얻을 수 있다. 이 기법은 모바일 스마트폰과 유선 서버사이의 비 신뢰성의 네트워크 환경을 고려하였다. 기존의 메커니즘 대비 에너지 효율성 높으며 서버와 클라이언트의 상호 인증 속도가 향상되었다. 향후 연구 사항은 제한사항의 구현을 통하여 실질적인 검증을 할 것이다.

참고문헌

[1] IEEE Std. 802.11i-2004. “Medium Access Control (MAC) Security Enhancements, Amendment 6 to IEEE Standard for Information technology Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer(PHY) specifications” [S]. June 2004.

[2] SSL and TLS: designing and building secure systems, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 2001

[3] R. Lu and Z. Cao, “Simple three-party key exchange protocol,” Computers & Security, vol. 26, no. 1, pp. 94 - 97, FEB 2007.

[4] H. Fereidooni, H. Taheri, M. Mahramian, new Authentication and Key Exchange Protocol for Insecure Networks, Wireless Communications, Networking and Mobile Computing, 2009.

[5] OMNeT++ 4.0 documentation, OMNeT++ Community - www.omnetpp.org