情報保護學會論文誌
第13券 第5號, 2003. 10

능동적 공격에 안전한 WTLS Handshake 프로토콜

한 종 수*, 정 영 석*, 안 기 범*, 곽 진*, 원 동 호**

A WTLS Handshake protocol against Active Attack

Jong-Su Han*, Young-Seok Chung*, Gi-Bum An*, Jin Kwak*, Dongho Won**

요 약

WAP의 보안 프로토콜인 WTLS는 무선 인터넷 프로토콜에서 TCP의 보안에 사용하는 TLS를 무선 환경에 맞게 최적한 것으로, 안전하고 효율적인 서비스 제공을 목적으로 하고 있다. WTLS 프로토콜은 Handshake, ChangeCipherSpec, Alert, Application Data 등 4개의 프로토콜로 구성되어 있으며, 본 논문에서는 Handshake 프로토콜에 대해 master secret를 설정하는 과정과 특징을 분석하고, 이를 기반으로 하여 능동적 공격자 모델에 대한 안전성을 분석한다. 또한 안전성 분석 결과를 바탕으로 능동적 공격자 모델에 안전하고 다양한 보안 서비스를 제공할 수 있는 새로운 Handshake 프로토콜을 제안한다.

ABSTRACT

WTLS as secure protocol of WAP makes TLS that is used in wireless Internet protocol for TCP security be appropriate for wireless environments. And purpose of WTLS is to provide safe and efficient services. WTLS protocol consists of 4 protocols(Handshake, ChangeCipherSpec, Alert, Application Data etc.). In this papers we analyze properties of Handshake protocol and procedures of establishing master secret in detail. And then we analyze securities against several attacker models with them for a basis. Also we propose new Handshake protocol that is secure against active attacker model and can provide various security services.

keyword : *WTLS, Handshake protocol, Active attack, Elliptic curve cryptosystem*

Ⅰ. 서 론

최근 휴대폰이나 PDA와 같은 무선 단말기를 이용한 인터넷 서비스가 확대됨에 따라 무선 인터넷 프로토콜인 WAP(Wireless Application Protocol)에 대한 관심이 집중되고 있다. WAP은 WAP Forum에서 제정하고 전세계적으로 가장 많은 사용자가 이용하는 무선 인터넷 프로토콜 표준으로 전송(WDP), 보안(WTLS), 트랜잭션(WTP), 세션(WSP), 응용(WAE) 등 5개의 계층으로 구성되어 있다.[3,5] 이 중 보안 프로토콜인 WTLS(Wirless Transprot Layer Security)는 무선 인터넷 프로토콜에서 TCP의 보안에 사용하는 TLS (Transport Layer Security)를 무선 환경에 맞게 최적한 것이며, 안전하고 효율적인 서비스 제공을 위한 요구조건은 다음과 같다. 첫 번째, 전송 계층의 보안 프로토콜의 기본 요구 조건으로 기밀성(confidentiality), 무결성(integrity), 사용자 인증(user authentication) 등을 제공해야 하며, 다양한 응용 계층을 위한 상호 호환성을 제공하고, 새로운 알고리즘의 추가가 용이해야 한다. 두 번째, 무선 환경에서 동작하는 특성을 고려

* 성균관대학교 정보통신공학부 정보통신보호연구실({jshan, yschung, gbahn, jkwak}@dosan.skku.ac.kr)
** 성균관대학교 정보통신공학부 정교수(dhwon@dosan.skku.ac.kr)

하여, 패킷의 중복이나 손실 등을 처리할 수 있는 능력, 또한 무선 단말기의 낮은 CPU 파워와 연산 능력 등을 고려해야 한다.

WTLS는 Handshake, ChangeCipherSpec(CCS), Alert, Application Data 등 4개의 계층 구조를 가지며 이들 계층의 모든 메시지는 하위의 Record layer 프로토콜을 거치는 형태로 구성된다.

Handshake 프로토콜은 서버와 클라이언트 사이에서 사용할 알고리즘을 포함하는 보안 파라미터(security parameters) 등을 설정하고, 이 값을 통해 Record layer에서 사용할 키 값을 유도하게 된다. ChangeCipherSpec 프로토콜은 설정된 알고리즘과 키 값을 사용할 수 있도록 하며, Alert 프로토콜은 통신하는 과정에서 생기는 오류를 처리해 준다. Application Data는 실질적으로 전송되는 데이터 값이 된다.[5]

이 중 Handshake 프로토콜은 서버와 클라이언트가 서로 공유하게 되는 키 분배 과정을 수행하는데, 기존의 프로토콜에서는 많은 공격에 대한 위험 요소가 존재하고 있다. 무선 환경이라는 취약점과 안전하지 않은 채널을 사용하여 통신을 수행하는 등을 고려할 때 이러한 위험요소는 프로토콜에 치명적인 약점으로 작용할 수 있다. 그러므로 본 논문에서는 기존의 WTLS Handshake 프로토콜에 대해 여러 가지 능동적 공격 모델에 대한 안전성을 검토하고, 이러한 능동적 공격 모델에 안전한 새로운 WTLS Handshake 프로토콜을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 WTLS Handshake 프로토콜에 대해 설명하고, 3장에는 기존 프로토콜의 안전성을 수동적 공격자와 능동적 공격자로 분류하여 각 환경에서의 안전성을 분석한다. 다음으로 4장에는 안전성 분석 결과를 바탕으로 능동적 공격 모델에 안전한 새로운 WTLS Handshake 프로토콜을 제안하고 5장에는 제안하는 프로토콜의 안전성과 특징을 분석하며, 마지막으로 6장에서 결론을 맺는다.

Ⅱ. WTLS Handshake 프로토콜

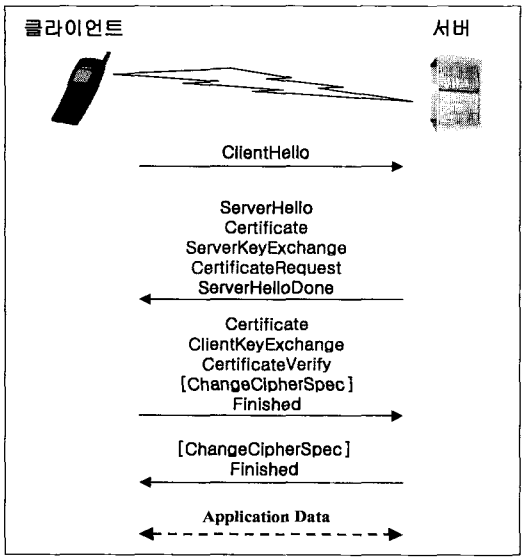
WTLS Handshake 프로토콜은 세션(session)과 커넥션(connection)으로 구성된다. 실제로 통신이 이루어지는 단위를 커넥션이라 하고 이를 묶은 단위가 세션이 된다. 각 세션은 예비 상태(pending state)와 현재 상태(current state)로 구분되며 예비 상태는 서버와 클라이언트가 통신을 수행하면서 설정된 알고리즘과 키를 임시로 저장하고 있는 상태이며, 현재 상태는 ChangeCipherSpec 프로토콜을 통해 Record layer에서 실제 데이터가 처리될 때의 상태를 의미한다. WTLS의 세션 상태 요소는 다음 [표 1]과 같으며, 하나의 세션이 종료될 때까지 유지된다.[3]

〔표 1〕 WTLS 세션 상태 요소

WTLS 세션 상태요소	설 명
Session identifier	세션을 구분하는 ID
Protocol Version	WTLS 버전
Peer Certificate	상대방의 인증서 : NULL도 가능
Compression Method	데이터를 압축하기 위해 사용되는 알고리즘
Cipher Spec	암호화, MAC 알고리즘
Master Secret	서버와 클라이언트가 공유하는 비밀 값
Sequence Number Mode	사용모드
Key Refresh	키 갱신 주기
Is Resumable	현재의 세션이 새로운 커넥션을 만들 수 있는지를 나타내는 값

2.1 WTLS Handshake 프로토콜의 구성

WTLS Handshake 프로토콜은 [그림 1]과 같이 구성된다. 클라이언트는 서버에게 ClientHello 메시지를 보내어 서버의 응답을 기다리고, 이 때 서버가 클라이언트에게 ServerHello 메시지를 보내지 않으면 에러가 발생하여 커넥션이 중단된다. 서버와 클라이언트는 Hello 메시지를 통해 WTLS 버전, 키 분배 알고리즘(본 WTLS Handshake 프로토콜에서는 master secret를 생성하여 공유하는 과정을 의미한다. : 이하 키 분배), 키 주기, 랜덤 값 등을 교환하고, 키 분배 알고리즘은 크게 DH(Diffie-Hellman), RSA, ECC 등을 사용할 수 있다. 서버는 ServerHello 메시지 다음으로 인증서(Certificate)를 보낼 수 있고, 인증서를 보내지 않거나 또는 인증서에 키 분배에 필요한 정보가 부족한 경우 ServerKeyExchange 메시지를 통해 필요한 정보를 보내게 된다. 또한 서버가 클라이언트의 인증이 필요한 경우 CertificateRequest 메시지를 통해 인증서를 요청한다. 그리고 서버는 Hello 메시지가 종료됨을 알리는 ServerHelloDone 메시지를 전송하고 클라이언트의 응답을 기다린다. 클라이언트 역시



〔그림 1〕 Handshake 프로토콜 구성

서버와 마찬가지로 인증서를 전송하거나 ClientKey-Exchange 메시지를 통해 필요한 정보를 서버에게 전달한다. 클라이언트가 서명을 수행하고 이를 서버가 검증할 수 있는 경우, 지금까지 전송한 Handshake 메시지에 대한 서명을 보냄으로써 클라이언트가 인증서에 대한 소유를 증명할 수 있는 CertificateVerify 메시지를 서버에게 보낸다. 클라이언트와 서버는 교환된 메시지를 이용하여 키를 생성하고 CCS 프로토콜을 수행한 뒤 Finished 메시지를 보내어 프로토콜을 종료한다.

2.2 WTLS Handshake 프로토콜의 동작

2.2.1 용어 정의

- V : WTLS 버전
- E : 최종 객체 (클라이언트, 서버)
- SID : 세션 ID
- $SecNeg_E$: key exchange suit, cipher suit, key fresh, compression method 등의 정보
- K_p : pre-master secret
- K_m : master secret
- $h()$: 일방향 해쉬 함수
- $\pi()$: 포인트의 x좌표를 출력해 주는 함수
- x_E : E의 비밀키
- P_E : E의 공개키
- $Cert_E$: E의 인증서
- R_E : E가 선택한 랜덤 값
- $E_K\{\}(D_K\{\})$: 키 K를 사용한 암호화(복호화)
- $x \| y$: x와 y 연접

2.2.2 프로토콜 수행 과정

Kwak[7] 등이 제시한 기존의 WTLS Handshake 프로토콜(W-H 프로토콜)의 수행과정은 다음과 같다.

① 클라이언트와 서버는 각각 ClientHello와 Server-Hello 메시지를 통해 키 값을 생성하기 위한 랜덤 값과 WTLS 버전 값과 세션 ID, 그리고 기타 알고리즘 정보 등을 서로에게 전송한다.

$$R_C, V, SID, SecNeg_C$$
$$R_S, V, SID, SecNeg_S$$

② 서버는 자신의 공개키가 포함되어 있는 인증서를 클라이언트에게 전달한다.

$$Cert_S$$

③ 서버는 클라이언트의 인증서를 요청하는 CertificateRequest 메시지와 Hello 메시지가 종료됨을 알리는 ServerHelloDone 메시지를 전송하고 클라이언트의 응답을 기다린다.

④ 클라이언트는 전송 받은 인증서를 검증하고, 서버의 공개키 P_S와 자신의 비밀키 x_C로 pre-master secret K_p를 계산한다.

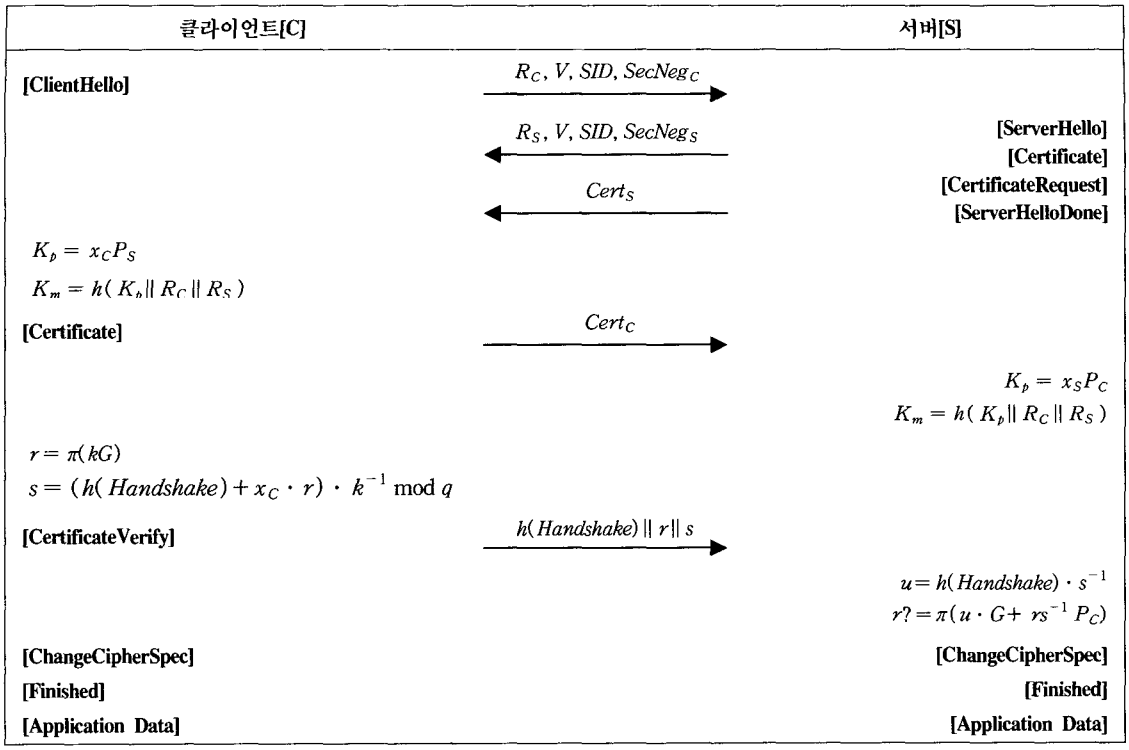
$$K_p = x_C P_S$$

⑤ 클라이언트는 pre-master secret와 서버에게 전송 받은 랜덤 값 R_S과 자신이 선택한 랜덤 값 R_C을 일방향 해쉬 함수의 입력으로 하여 master secret K_m를 계산한다.

$$K_m = h(K_p \| R_C \| R_S)$$

⑥ 클라이언트는 서버의 요청에 따라 자신의 인증서를 전송한다.

$$Cert_C$$



※ G : 타원 곡선의 generator, n : generator G의 위수 , q : 큰 소수 $|q| = |h(\cdot)|$
k : 클라이언트가 선택한 랜덤 값 *where* $k \in [1, n)$

〔그림 2〕 기존의 ECC 기반 WTLS Full Handshake 프로토콜 (W-H)

⑦ 서버는 전송 받은 인증서를 검증하고, 클라이언트의 공개키 P_C와 자신의 비밀키 x_S로 pre-master secret K_p를 계산한다.

$$K_p = x_S P_C$$

⑧ 서버는 pre-master secret와 클라이언트에게 전송 받은 랜덤 값 R_C과 자신이 선택한 랜덤 값 R_S을 일방향 해쉬 함수의 입력으로 하여 master secret K_m를 계산한다.

$$K_m = h(K_p \| R_C \| R_S)$$

⑨ 클라이언트는 랜덤 값 k를 선택하여 kG를 구하고 π 함수를 이용하여 x좌표 r를 구한다.

$$r = \pi(kG)$$

⑩ 클라이언트는 지금까지 전송한 Handshake 메시지에 서명을 수행하고 서명한 값과 해쉬 값을 연접하여 CertificateVerify 메시지를 서버에게 전송한다.

$$s = (h(Handshake) + x_C \cdot r) \cdot k^{-1} \bmod q$$
$$h(Handshake) \| r \| s$$

⑪ 서버는 전송 받은 값을 통해 클라이언트의 서명을 검증한다.

$$u = h(Handshake) \cdot s^{-1}$$
$$r? = \pi(u \cdot G + rs^{-1} P_C)$$

⑫ 서버와 클라이언트는 ChangeCipherSpec 프로토콜을 수행하고, Finished 메시지를 전송하여 Handshake 프로토콜을 종료한다.

위의 [그림 2]는 기존의 ECC 기반 WTLS Full Handshake 프로토콜의 과정을 나타낸 것이다.

Ⅲ. WTLS Handshake 프로토콜 안전성 분석

3.1 공격자 모델

암호 프로토콜의 안전성을 증명하기 위해서는 먼저 대상이 되는 공격자 모델을 정의해야 한다. 암호 프로토콜에 대한 공격자는 크게 수동적 공격자(passive attacker)와 능동적 공격자(active attacker)로 나눌 수 있으며, 각각의 특징은 다음과 같다.[8]

- 수동적 공격자 : 프로토콜의 참가자와 실제로 통신에 참여하지 않고 두 참가자 사이의 통신 내용을 도청(eavesdropping)함으로써 공격을 수행하는 공격자
- 능동적 공격자 : 단순히 참가자들의 통신 내용을 도청하는 것뿐만 아니라 전송되는 메시지를 위·변조하거나 새로운 메시지를 삽입하거나 하는 등 실제 통신에 참여하는 보다 강력한 공격자

본 논문에서 고려할 능동적 공격자 모델은 active impersonation 공격자, forward secrecy에 대한 공격자, key compromise impersonation 공격자, known key security에 대한 공격자이며 각각에 대한 정의는 다음과 같다.

[정의 1] Active Impersonation(AI) attack
공격자가 자신을 임의의 다른 사용자로 위장하여 프로토콜에 참여하고, 정당한 사용자 U와 키 분배를 성공적으로 수행하는 경우에 AI가 가능하다고 한다.

[정의 2] Forward Secrecy(FS)
사용자 U와 V의 비밀키가 노출되더라도, 공격자가 두 사용자 사이에 설정된 과거의 세션키(본 WTLS Handshake 프로토콜에선 master secret를 의미 : 이하 master secret)를 계산할 수 없는 경우에 FS를 만족한다고 한다. FS는 노출되는 사용자의 키에 따라 다음과 같이 나눌 수 있다.

- Half Forward Secrecy : 한 사용자의 비밀키가 노출된 경우에만 master secret가 안전
- Full Forward Secrecy : 두 사용자의 비밀키가 노출된 경우에도 master secret가 안전

[정의 3] Key-Compromise Impersonation(KCI) attack
사용자 U의 비밀키가 노출되었을 때, 공격자 E가 누구에게나 사용자 U로 위장할 수 있고, 사용자 U에게 임의의 사용자 V로 위장할 수 있을 때, KCI가 가능하다고 한다. 그러나 공격자 E가 누구에게나 사용자 U로 위장할 수 있지만 사용자 U에게 임의의 다른 사용자로 위장할 수 없는 경우에는 키 분배 프로토콜이 key-compromise impersonation resilience 특성을 갖는다고 한다.

[정의 4] Known Key Security(KKS)
두 사용자 U, V 사이의 과거 master secret가 노출되더라도 현재 master secret의 안전성에는 아무런 영향을 미치지 않는 경우에 KKS를 만족한다고 한다. KKS에 대한 공격은 다음과 같이 두 가지로 나눌 수 있다.

- KKP(Known Key Passive) 공격 : 과거의 master secret 정보와 전송 정보 및 현재 세션의 전송 정보를 이용하여 현재의 master secret를 획득하려는 공격 방법
- KKI(Known Key Impersonation) 공격 : 세션에 직접 참여하여 과거의 master secret 정보와 전송 정보, 현재 세션의 전송 정보를 이용하여 사용자 U에게 사용자 V로 위장하여 master secret를 설정하려는 공격 방법

3.2 안전성 분석

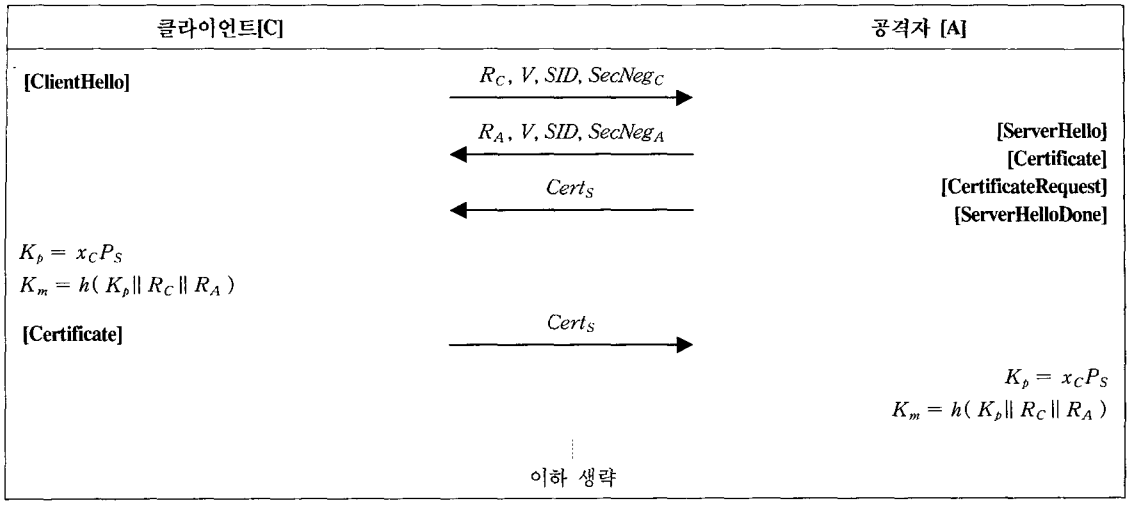
3.2.1 수동적 공격자에 대한 안전성

기존의 WTLS Handshake(W-H) 프로토콜은 기본적으로 그 안전성이 ECDH(Elliptic Curve Diffie-Hellman), ECDSA(Elliptic Curve Digital Signature Algorithm) 문제에 기반하므로 수동적 공격자가 공개 정보와 전송 정보를 이용하여 master secret를 구하는 어려움은 ECC 기반의 문제를 푸는 어려움과 동일하다.

3.2.2 능동적 공격자에 대한 안전성

• Active Impersonation에 대한 안전성

W-H 프로토콜은 상대방의 개체 인증을 위해 long term 키 쌍을 이용하고 또한 인증서를 통한 검증을 수행하므로 공격자가 세션에 직접 참여하여 다른 사용자로 위장하는 active impersonation 공격을 수행할 수 없다. 즉, 정당한 클라이언트의 비밀키를 알지 못하는 공격자가 프로토콜에 직접 참여하여 서버에게 클라이언트로 위장하기 위해서는 ECDH의 문제를 해



〔그림 3〕 W-H 프로토콜에 대한 KCI 공격 과정

결해야 한다. 또한 전송한 인증서에 저장된 공개키에 대응하는 정당한 서명을 생성할 수도 없기 때문에 AI 공격자의 어려움은 ECDSA 문제의 어려움과 동치이다.

• Forward Secrecy에 대한 안전성

W-H 프로토콜의 master secret는 pre-master secret와 최종 객체($E_{(1/2)}$) 서버와 클라이언트가 각각 선택한 랜덤 값을 통해 생성한다.

$$K_p = x_{E_1} P_{E_2}$$
$$K_m = h(K_p \| R_{E_1} \| R_{E_2})$$

따라서 두 객체 중 한 객체의 비밀키만 노출되더라도 두 객체 사이의 master secret를 쉽게 계산할 수 있다. 따라서 어떤 형태의 forward secrecy를 제공하지 않는다.

• Key Compromise Impersonation에 대한 안전성

W-H 프로토콜에서 클라이언트 C의 비밀키 x_C가 노출되는 경우에 공격자는 임의의 서버에게 클라이언트 C로 위장할 수 있을 뿐만 아니라 정당한 클라이언트 C에게 서버 S로 위장하는 것이 가능하다. 따라서 W-H 프로토콜은 key compromise resilience 특성을 갖지 못한다. 클라이언트 C의 고정된 비밀키 x_C를 획득한 공격자가 서버 S로 위장하여 master secret를 설정하는 공격 과정은 [그림 3]과 같다.

• Known Key attack에 대한 안전성

W-H 프로토콜은 과거에 생성한 master secret 정보 중 pre-master secret가 노출되면 KKP와 KKI의 모든 공격에 대해 안전하지 못하다.

KKP의 경우 과거의 pre-master secret가 노출되면, 각 객체의 long term 비밀키와 공개키로 계산하기 때문에 각 세션마다 같은 값이 되게 된다. 따라서 현재 세션의 전송 정보인 랜덤 값을 이용하면 쉽게 현재 세션의 master secret를 획득할 수 있다.

$$K_p = x_{E_1} P_{E_2}$$
$$K_m = h(K_p, R_{E_1}, R_{E_2})$$

KKI의 경우도 마찬가지로 모든 세션마다 고정되어 있는 pre-master secret 값을 획득한 공격자가 프로토콜에 직접 참여하는 경우에는 임의의 클라이언트로 위장하여 정당한 서버와 master secret를 설정하는 공격이 가능하다.

Ⅳ. 제안하는 WTLS Handshake 프로토콜

본 논문에서 제안하는 WTLS Handshake 프로토콜은 3장에서 정의한 능동적 공격 모델에 대해 안전하고 추가적인 보안 서비스를 제공하기 위해 Self-certified Signature(SCS)[9]의 개념을 도입한다.

4.1 용어 정의

• x_{CA} : 인증기관의 비밀키

- P_{CA} : 인증기관의 공개키
- $x_{E_{(C\cdot S)}}$: 최종 객체의 long term 비밀키
- $P_{E_{(C\cdot S)}}$: 최종 객체의 long term 공개키
- x_E : 프로토콜에 사용되는 임시 비밀키
- P_E : 프로토콜에 사용되는 임시 공개키
- u_E : 중간 해쉬 값
- CI_E : 최종 객체의 공개키 값을 포함하는 기타 정보
- *kdf* : ANSI X9.63[10]에 정의된 키 유도 함수
- 나머지 정의는 2.2.1과 동일

4.2 인증서 발급 및 검증과정

제안하는 ECC 기반 인증서 발급 및 검증과정은 다음과 같다.

① 최종 객체(서버 혹은 클라이언트)는 인증기관에 신원확인 절차를 수행하고, 자신의 공개키 P_{E_t} 값과 일련 번호, 서명자와 발급자의 ID, 유효기간 등의 정보를 포함한 CI_E의 내용을 확인한다.

② 인증기관은 랜덤 값 k_{CA}를 선택하여 $r_E(=k_{CA}G)$를 구하고 π 함수를 이용하여 x좌표 r_E' 구한다.

$$r_E' = \pi(r_E)$$

③ 인증기관은 자신의 비밀키 x_{CA}를 사용하여 서명을 생성한다.

$$s_E = x_{CA}\cdot h(CI_E \| r_E') + k_{CA} \mod n$$

④ 인증기관은 ②, ③에서 생성한 값을 연접하여 인증서를 생성한다.

$$Cert_E = (r_E' \| s_E)$$

⑤ 인증기관은 ANSI X9.63에서 정의하고 있는 Static unified model을 적용하여 세션키 K를 생성하여 $Cert_E$와 CI_E를 암호화하여 암호문 C를 최종 객체에게 전송한다.

$$K = kdf(x_{CA}\cdot P_{E_t})$$
$$C = E_K(Cert_E \| CI_E)$$

⑥ 최종 객체는 세션키 K를 생성하여 암호문 C를 복호화하여 $Cert_E$와 CI_E를 얻는다.

$$K = kdf(x_{E_t}\cdot P_{CA})$$
$$D_K\{C\} = Cert_E \| CI_E$$

⑦ 최종 객체는 복호화한 CI_E와 r_E'를 일방향 해쉬

최종 객체[E]		인증기관[CA]
long term 키 쌍 (x_{E_t}, P_{E_t})		(x_{CA}, P_{CA})
$P_{E_t} = x_{E_t}G$		$P_{CA} = x_{CA}G$
		CI_E
		$r_E = k_{CA}G = (x_t, y_t)$
		$r_E' = \pi(r_E)$
		$s_E = x_{CA}\cdot h(CI_E \| r_E') + k_{CA} \mod n$
$K = kdf(x_{E_t}\cdot P_{CA})$		
$D_K\{C\} = Cert_E \| CI_E$	$\leftarrow C$	$Cert_E = (r_E' \| s_E)$
$u_E = h(CI_E \| r_E')$		$K = kdf(x_{CA}\cdot P_{E_t})$
$s_E G - u_E P_{CA} = (x_t, y_t) = r_E$		$C = E_K(Cert_E \| CI_E)$
$r_E' \stackrel{?}{=} \pi(r_E)$		

※ G : 타원 곡선의 generator, n : generator G의 위수, $|n| = |h(\cdot)|$
k_{CA} : 인증기관이 선택한 랜덤 값 *where* $k_{CA} \in [1, n)$, K : 세션키
CI_E = [일련 번호, P_E, 서명자의 식별자, 발급자의 식별자, 유효기간 등의 정보]

〔그림 4〕 ECC 기반 인증서 발급 및 검증과정

함수를 사용하여 u_E를 계산하고, 인증기관 공개키 P_{CA}를 사용하여 서명이 정당한지 검증한다.

$$u_E = h(CI_E \| r_E')$$
$$s_E G - u_E P_{CA} = (x_t, y_t) = r_E$$
$$r_E' \stackrel{?}{=} \pi(r_E)$$

⑧ 최종 객체는 인증기관으로부터 전송 받은 인증서 값 중 s_E을 비밀리에 보관하고 인증서 값으로 CI_E, r_E을 사용한다.

위의 [그림 4]는 ECC 기반 인증서 발급 및 검증 과정을 나타낸 것이다.

4.3 제안하는 프로토콜 수행과정

제안하는 WTLS Handshake 프로토콜(P-W-H 프로토콜)은 다음과 같다.

① 클라이언트와 서버는 각각 ClientHello, ServerHello 메시지를 통해 키 값을 생성하기 위한 랜덤 값 $R_{(C/S)}$를 선택하여 계산한 랜덤 포인트 값과 WTLS 버전 값과 세션 ID, 기타 알고리즘 정보 등을 서로에게 전송한다.

$$R_C G,\ V,\ SID,\ SecNeg_C$$
$$R_S G,\ V,\ SID,\ SecNeg_S$$

② 서버는 자신의 long term 비밀키 x_{E_s}과 인증기관으로부터 발급 받은 인증서 값 중 비밀리에 보관하고 있는 s_S로 임시 비밀키 x_S을 생성한다.

$$x_S = x_{E_s} + s_S$$

③ 서버는 자신의 long term 공개키 P_{E_s}와 인증기관의 공개키 P_{CA}, 그리고 인증서 값 r_S, 중간 해쉬 값 u_S로 ②에서 생성한 비밀키에 대응하는 임시 공개키 P_S를 생성한다.

$$u_S = h(CI_S \| r_S')$$
$$P_S = P_{E_s} + P_{CA} \cdot u_S + r_S$$

④ 서버는 랜덤 값 k_S를 선택하여 $T_S\ (= k_S G)$를 구하고 π 함수를 이용하여 x좌표 T_S'를 구한다.

$$T_S = k_S G = (x_{t_s}, y_{t_s})$$
$$T_S' = \pi(T_S)$$

⑤ 서버는 자신의 임시 비밀키 x_S를 사용하여 서명문을 생성한다.

$$S_S = x_S \cdot h(CI_S \| r_S' \| T_S') + k_S \bmod n$$

⑥ 서버는 자신의 인증서 값과 ④, ⑤에서 생성한 서명문을 함께 클라이언트에게 전송하고 클라이언트의 인증서를 요청하는 Certificate-Request 메시지와 Hello 메시지가 종료됨을 알리는 ServerHello-Done 메시지를 전송하고 클라이언트의 응답을 기다린다.

$$(T_S' \| S_S) \| CI_S \| r_S$$

⑦ 클라이언트는 전송 받은 CI_S에서 서버의 long term 공개키 P_{E_s}을 확인하고 인증기관의 공개키 P_{CA}와 인증서 값 r_S, 중간 해쉬 값 u_S을 사용하여 서버의 임시 공개키 P_S를 계산한다.

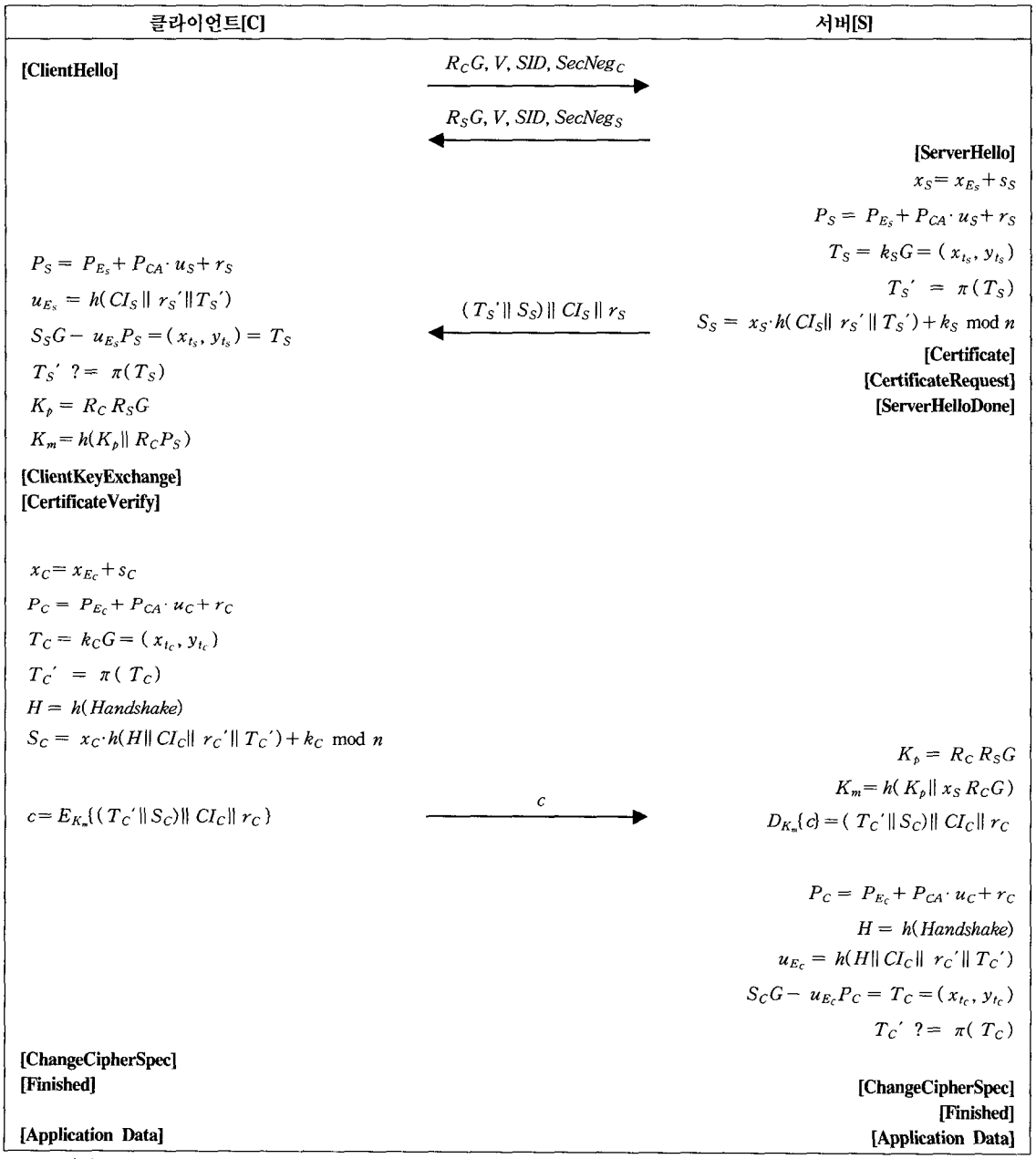
$$P_S = P_{E_s} + P_{CA} \cdot u_S + r_S$$

⑧ 클라이언트는 전송 받은 CI_S와 T_S', r_S'를 일방향 해쉬 함수를 사용하여 u_{E_s}를 계산하고, ⑦에서 계산한 서버의 공개키 P_S를 사용하여 서명이 정당한지 검증한다.

$$u_{E_s} = h(CI_S \| r_S' \| T_S')$$
$$S_S G - u_{E_s} P_S = (x_{t_s}, y_{t_s}) = T_S$$
$$T_S' \stackrel{?}{=} \pi(T_S)$$

⑨ 클라이언트는 서버의 서명을 검증함으로써 인증을 수행한 후 ①에서 전송 받은 $R_S G$와 자신이 선택한 R_C로 pre-master secret를 계산한다.

$$K_p = R_C R_S G$$



※ G : 타원 곡선의 generator, n : generator G의 위수, $|n| = |h(\cdot)|$
k_E : 최종 객체(서버, 클라이언트)가 선택한 랜덤 값 $where\ k \in [1, n)$
X' : 포인트 X의 x좌표

〔그림 5〕 제안하는 ECC 기반 WTLS Full Handshake 프로토콜 (P-W-H)

⑩ 클라이언트는 pre-master secret와 서버의 공개키 P_S, 자신이 선택한 랜덤 값 R_C로 master secret를 생성한다.

$$K_m = h(K_p \| R_C P_S)$$

⑪ 클라이언트는 자신의 long term 비밀키 x_{E_c}과 인증기관으로부터 발급 받은 인증서 값 중 비밀리에 보관하고 있는 s_C로 임시 비밀키 x_C을 생성한다.

$$x_C = x_{E_c} + s_C$$

⑫ 클라이언트는 자신의 long term 공개키 P_{E_C}와 인증기관의 공개키 P_{CA}, 그리고 인증서 값 r_C, 중간 해쉬 값 u_C로 ⑪에서 생성한 비밀키에 대응하는 임시 공개키 P_C를 생성한다.

$$u_C = h(CI_C \| r_C')$$
$$P_C = P_{E_C} + P_{CA} \cdot u_C + r_C$$

⑬ 클라이언트는 랜덤 값 k_C를 선택하여 $T_C\ (= k_C G)$를 구하고 π 함수를 이용하여 x좌표 T_C'를 구한다.

$$T_C = k_C G = (x_{t_C}, y_{t_C})$$
$$T_C' = \pi(T_C)$$

⑭ 클라이언트는 지금까지 전송한 Handshake 메시지의 해쉬 값과 기타 정보를 자신의 임시 비밀키 x_C를 사용하여 서명을 수행한다.

$$H = h(Handshake)$$
$$S_C = x_C \cdot h(H \| CI_C \| r_C' \| T_C') + k_C \bmod n$$

⑮ 클라이언트는 자신의 서명이 공개되어 자신의 신원이 노출되는 것을 방지하고, 즉 익명성(anonymity)을 제공하고, 서버와 클라이언트의 간에 설정한 master secret 확인(key confirmation)을 제공하기 위해 ⑩에서 생성한 K_m으로 인증서 값과 서명문을 암호화하여 암호문 c를 서버에게 전송한다.

$$c = E_{K_m}\{(T_C' \| S_C) \| CI_C \| r_C\}$$

⑯ 서버는 ①에서 전송 받은 $R_C G$와 자신이 선택한 R_S로 pre-master secret를 계산한다.

$$K_p = R_S R_C G$$

⑰ 서버는 pre-master secret와 자신의 비밀키 x_S, $R_C G$로 master secret를 생성한다.

$$K_m = h(K_p \| R_C P_S)$$

⑱ 서버는 ⑰에서 생성한 K_m으로 c를 복호화하여 클라이언트의 인증서 값과 서명문을 얻는다.

$$D_{K_m}\{c\} = (T_C' \| S_C) \| CI_C \| r_C$$

⑲ 서버는 CI_C에서 클라이언트의 long term 공개키 P_{E_C}를 확인하고 인증기관의 공개키 P_{CA}와 인증서 값 r_C, 중간 해쉬 값 u_C을 사용하여 서버의 임시 공개키 P_C를 계산한다.

$$P_C = P_{E_C} + P_{CA} \cdot u_C + r_C$$

⑳ 서버는 CI_C와 지금까지 전송한 Handshake 메시지의 해쉬 값 H, T_C', r_C'을 일방향 해쉬 함수에 돌려 u_{E_C}를 계산하고, ⑲에서 계산한 서버의 공개키 P_C를 사용하여 서명이 정당한지 검증한다.

$$H = h(Handshake)$$
$$u_{E_C} = h(H \| CI_C \| r_C' \| T_C')$$
$$S_C G - u_{E_C} P_C = T_C = (x_{t_C}, y_{t_C})$$
$$T_C' \stackrel{?}{=} \pi(T_C)$$

서버와 클라이언트는 ChangeCipherSpec 프로토콜을 수행하고, Finished 메시지를 전송하여 Handshake 프로토콜을 종료한다. [그림 5]는 제안하는 ECC 기반 Full Handshake 프로토콜을 나타낸 것이다.

V. 제안하는 프로토콜의 안전성 분석 및 특징

본 장에서는 3장에서 정의한 공격자 모델에 따라 제안하는 프로토콜에 대한 안전성을 분석하고 기존의 WTLS Handshake 프로토콜과의 비교·분석을 통해 프로토콜의 특징에 대하여 설명한다.

5.1 안전성 분석

5.1.1 수동적 공격자에 대한 안전성

제안하는 WTLS Handshake 프로토콜은 기본적으로 그 안전성이 ECDH, ECDSA 문제에 기반하므로 수동적 공격자가 공개정보와 전송 정보를 이용하여 master secret를 구하는 어려움은 ECC 기반의 문제를 푸는 어려움과 동일하다.

5.1.2 능동적 공격자에 대한 안전성

▪ Active Impersonation에 대한 안전성

P-W-H 프로토콜은 서버와 클라이언트가 객체 인증을 자신의 비밀키로 생성한 서명문과 인증기관이 발행한 인증서를 가지고 수행하므로 각 객체의 비밀키를 모르는 공격자가 자신을 임의의 다른 사용자로 위장하여 프로토콜에 참여하고, 정당한 객체와 master secret를 성공적으로 공유하는 active impersonation 공격이 불가능하다. P-W-H 프로토콜은 임시 공개키와 비밀키를 사용하지만 공격자가 정당한 키 쌍을 생성하기 위해서는 인증기관의 비밀키와 랜덤하게 생성한 k_{CA}를 알아야 하므로 공격이 불가능하다.

▪ Forward Secrecy에 대한 안전성

P-W-H 프로토콜의 master secret는 클라이언트의 경우 pre-master secret와 자신이 랜덤하게 선택한 R_C 값 그리고 서버의 공개키 P_S를 가지고 계산한다.

$$K_p = R_C R_S G$$
$$K_m = h(K_p \| R_C P_S)$$

만약 클라이언트의 비밀키가 노출되어 공격자가 이를 얻었다하더라도 공격자는 과거의 master secret를 계산할 수 없다. master secret를 계산하기 위해서는 클라이언트가 랜덤하게 선택한 R_C를 구해야 하고 이는 ECC를 푸는 어려움과 동일하므로 공격이 불가능하다.

서버의 경우에는 pre-master secret와 클라이언트로부터 전송 받은 랜덤 포인트 값 $R_C G$ 그리고 자신의 비밀키 x_S를 가지고 계산한다.

$$K_m = h(K_p \| x_S R_C G)$$

서버의 비밀키 x_S가 노출되는 경우 공격자는 master secret의 일부분은 계산할 수 있지만 pre-master secret를 계산할 수 없으므로, 정당한 master secret를 계산할 수 없다. 공격자가 master secret를 계산하기 위해서는 전송되는 랜덤 포인트 값으로부터 R_C 혹은 R_S을 구해야 하고 이는 ECC를 푸른 어려움과 동일하므로 공격이 불가능하다.

따라서 P-W-H 프로토콜은 두 객체의 비밀키가 노출된 경우에도 master secret가 안전하므로 Full Forward Secrecy를 만족한다.

▪ Key Compromise Impersonation에 대한 안전성

P-W-H 프로토콜은 클라이언트 C의 long term 비밀키가 노출된 경우에도 공격자가 임의의 서버에게 C로 위장하거나 정당한 C에게 서버 S로 위장하는 것이 불가능하다. P-W-H 프로토콜은 이러한 공격을 방지하기 위해 long-term 키를 그대로 사용하지 않고, 인증기관으로부터 발급 받은 인증서 값 중 비밀리에 보관하는 s_C를 가지고 임시의 키 쌍을 생성하여 사용한다. 따라서 공격자가 클라이언트 C의 long term 비밀키 x_{E_C}를 얻었다 하더라도 s_C를 알지 못하면 임시 비밀키 x_C를 구할 수 없다.

또한 임시 비밀키 x_C가 노출된 경우에도 공격자는 인증기관으로부터 전송 받은 r_C의 값을 알 수 없으므로 정당한 서명문 S_C와 암호문 c를 생성할 수 없다. 따라서 공격자는 long-term 키 x_{E_C} 또는 임시 비밀키 x_C를 얻었다 하더라도 정당한 클라이언트로 위장하는 것은 불가능하다.

서버 S도 클라이언트와 마찬가지로 임시 키 쌍을 생성하여 세션을 구성하므로 두 객체의 long term 비밀키가 노출되더라도 공격자는 그 누구로도 위장할 수 없다.

따라서 P-W-H 프로토콜은 KCI에 대한 안전성을 보장할 수 있다.

▪ Known Key attack에 대한 안전성

P-W-H 프로토콜은 master secret를 계산하기 위해 매 세션마다 랜덤한 pre-master secret를 사용하므로 이전의 전송 정보와 master secret가 노출되더라도 현재 세션의 master secret를 구하는데 아무런 도움을 주지 못한다. 따라서 KKP 공격자의 어려움은 이전 세션에 대한 아무런 정보가 없는 수동적 공격자의 어려움과 동일하다.

KKI의 경우에도 마찬가지로 공격자가 세션에 직접 참여하여 과거의 master secret 정보와 전송정보, 현재 세션의 전송 정보를 이용하여 임의의 서버 혹은 클라이언트로 위장하여 master secret를 설정하는 것이 불가능하다.

5.2 기존 프로토콜과의 비교 분석

본 단락에서는 3장에서 살펴본 기존 ECC 기반의 WTLS Handshake 프로토콜(W-H 프로토콜)의 안전성과 앞 단락에서 살펴본 제안하는 WTLS Handshake 프로토콜(P-W-H 프로토콜)의 안전성을 비교한다.

〔표 2〕 W-H과 P-W-H 프로토콜의 안전성 비교·분석

공격 모델	W-H 프로토콜	P-W-H 프로토콜
수동적 공격자	안 전	안 전
AI	안 전	안 전
FS	제공하지 않음	Full
KCI	불안전	안 전
KKP	불안전	안 전
KKI	불안전	안 전

W-H 프로토콜은 수동적 공격자 모델과 능동적 공격자 모델 중 AI에 대한 안전성만을 제공한다. FS은 고정되어 있는 long term 키 쌍과 전송정보인 랜덤 값을 통해 master secret를 계산하므로 두 객체 중 한 개체의 비밀키만 노출되면 공격자는 쉽게 정당한 master secret를 설정할 수 있으므로, 어떠한 FS도 제공하지 않는다. KCI 공격 모델의 경우에도 비밀키가 노출되면 쉽게 위장이 가능하고 KKP, KKI의 공격 모델에 대한 안전성도 고정되어 있는 pre-master secret가 노출되면 공격자가 쉽게 현재 세션의 master secret를 설정할 수 있는 문제점이 발생하였다.

그러나 제안하는 P-W-H 프로토콜은 앞서 살펴본 바와 같이 수동적 공격자 모델뿐만 아니라 능동적 공격자 모델에 대한 안전성을 보장할 수 있다. 다음 [표 2]는 W-H 프로토콜과 P-W-H 프로토콜의 안전성을 비교·분석한 것이다.

또한 Handshake 프로토콜의 특징을 분석하기 위해 master secret 설정에 필요한 통신 횟수와 개체 인증(entity authentication), 키 확인(key confirmation), 키 인증(key authentication), key freshness, 사용자 익명성(user anonymity) 등의 제공 여부를 분석하고 기존 프로토콜과 비교하였다. 각각에 대한 정의는 다음과 같다.

- n-pass 프로토콜 : 한 사용자 입장에서 master secret를 설정하기 위해 상대방과 n번의 통신이 필요한 프로토콜
- 개체 인증(Entity authentication) : 프로토콜에 참여하고 있는 상대방의 신원을 확인하는 것으로 명시적 키 인증(explicit key authentication)에 의해 제공될 수 있음.
- 키 확인(key confirmation) : 프로토콜에 참여한 합법적인 사용자가 자신이 의도한 상대방과 실제로 비밀 세션키(본 논문에서는 master secret를 의미)를 공유하였음을 확인
- 묵시적 키 인증(implicit key authentication) : 키의 소유 여부는 알려져 있지 않더라도 프로토콜에 참여한 상대방만 이 세션키(본 논문에서는 master secret를 의미)를 계산할 수 있음을 보장
- 명시적 키 인증(explicit key authentication) : 사용자는 자신이 통신을 하고자 하는 상대방만이 비밀 세션키를 계산할 수 있고 상대방이 실제로 그 키를 가지고 있음을 확인할 수 있음. 즉 묵시적 키 인증과 키 확인을 모두를 만족할 때 제공할 수 있음.
- key freshness : 세션마다 설정된 키(본 논문에서는 master secret를 의미)가 바뀜

W-H 프로토콜은 [그림 2]와 같이 5번의 통신 횟수를 가진다. 클라이언트는 서명문을 생성하고 서버는 이를 검증함으로써 서버는 클라이언트의 일방향 개체 인증이 가능하다. 또한 master secret를 생성할 때 각각의 long term 키가 사용되므로 참여한 서버와 클라이언트만이 계산할 수 있음을 보장하므로 묵시적 키 인증을 제공한다. 또한 각 개체가 선택한 랜덤 값이 master secret에 포함되므로 양방향 key freshness를 제공한다. 하지만 키 확인 과정이 생략되어 있고, 따라서 명시적 키 인증을 제공할 수 없다. 또한 사용자의 인증서가 평문 형태로 전송되므로 사용자의 신원 노출 및 접근 여부 등이 나타나므로 사용자의 프라이버시 침해에 영향을 줄 수 있다.

P-W-H 프로토콜은 [그림 5]와 같이 4번의 통신 횟수를 가진다. 그리고 서버와 클라이언트 모두 서명문을 생성하여 검증하므로, 양방향 개체 인증이 가능하다. 또한 master secret를 생성할 때 클라이언트는 서버의 공개키를 사용하여 계산하므로, 자신이 사용한 공개키에 대응되는 정당한 비밀키를 가진 서버만이 master secret를 계산할 수 있으므로 서버에 대한 묵시적 키 인증을 제공할 수 있다. 서버 역시 클라이언트의 비밀키와 랜덤 값 R_C를 아는 상대방만이 정당한 master secret와 서명문을 생성할 수 있으므로 클라이언트에 대한 묵시적 키 인증을 제공할 수 있다. 또한 클라이언트는 생성한 master secret로 클라이언트의 인증서와 서명문 등을 암호화하여 전송하고 서버는 이를 복호화함으로써 키 확인 과정을 제공한다. 따라서 클라이언트는 서버에게 명시적 키 인증을 제공한다. W-H 프

〔표 3〕 W-H과 P-W-H 프로토콜의 특징 비교·분석

특 징	W-H 프로토콜	P-W-H 프로토콜
n-pass	5	4
개체 인증	일방향	양방향
키 확인	제공하지 않음	일방향
묵시적 키 인증	양방향	양방향
명시적 키 인증	제공하지 않음	일방향
key freshness	양방향	양방향
사용자 익명성	제공하지 않음	제 공

로토콜과 마찬가지로 master secret를 계산할 때 각 개체가 선택한 랜덤 값이 포함됨으로 양방향 key freshness를 제공한다. 추가적으로 앞서 키 확인 과정에서 클라이언트의 인증서를 암호화하여 전송함으로써, 채널을 도청하는 수동적 공격자에 대한 안전성을 보장하고 클라이언트의 신원 노출 방지 등의 사용자 익명성을 제공한다.

[표 3]은 위에서 비교한 내용을 정리한 것이다.

Ⅵ. 결 론

본 논문에서는 기존의 WTLS Handshake 프로토콜에 대한 안전성을 분석하였다. 그 결과 여러 가지 공격 모델에 대한 취약점을 발견하였고, 이에 SCS의 개념을 도입하여 능동적 공격 모델에 안전하고, 키 분배 프로토콜의 특징을 개선한 새로운 프로토콜을 제안하였다.

기존의 Handshake 프로토콜은 크게 4부분으로 구성된다. 첫 번째는 클라이언트와 서버가 각각 선정한 랜덤 값을 전송하는 부분, 두 번째 인증서를 전송하고 검증하는 부분, 세 번째 master secret를 생성하는 부분, 네 번째 서명을 수행하는 부분으로 구성된다. 기존의 프로토콜과 달리 제안하는 프로토콜에는 임시 키 쌍을 생성하는 단계와 검증하는 단계, 그리고 서버의 서명 생성단계가 존재한다. 임시 키 쌍을 생성하는 과정은 적은 계산량을 요구하며, 서버는 무선 환경에서의 단말기와는 달리 충분한 계산 능력과 메모리가 있기 때문에 서명을 생성하는 데에 있어서 큰 부담을 가지지 않는다. 따라서 제안하는 프로토콜은 고수준의 안전성을 요구하는 무선 통신 환경에 적용하기 적합하다.

추후 클라이언트 측에 요구되는 추가적인 계산량을 고려하여 보다 효율적인 프로토콜에 관한 연구를 진행할 것이다.

참 고 문 헌

[1] A. Freier, P. Karlton, P. Kocher. "The SSL Protocol version 3.0", IETF Internet Draft, Nov. 1996.

[2] T. Dierks and C. Allen, "The TLS Protocol version 1.0", IETF RFC 2246, Jan. 1999.

[3] WAP Forum, "Wireless Application Protocol Wireless Transport Layer Security Specification version 18-FEB-2000", Feb. 2000.

[4] 이동훈, 임채훈, "SSL 3.0과 TLS 1.0의 비교 분석", (주)퓨처시스템 암호체계센터 Technical Report, Sep. 2000.

[5] 이동훈, 임채훈, "TLS와 WTLS의 비교 분석", (주)퓨처시스템 암호체계센터 Technical Report, Oct. 2000.

[6] 최승복, 임채훈, "SSL 가속 기술의 분류와 제품 비교", (주)퓨처시스템 암호체계센터 Technical Report, Sep 2001

[7] Dong Jin Kwak, Jae Cheol Ha, Hoon Jae Lee, Hwan Koo Kim, and Sang Jae Moon, "A WTLS Handshake Protocol with User Anonymity and Forward Secrecy", CIC 2002 LNCS 2524, pp.219~230, 2003.

[8] 오수현, 이승우, 심경아, 양형규, 원동호, "타원 곡선에 기반한 표준 키 분배 프로토콜의 안전성 분석 및 응용 분야에 관한 연구", 정보보호학회 논문지, pp.103~118, 2002. 6.

[9] Byoungcheon Lee, Kwangjo Kim, "Self- certified Signatures", INDOCRYPT 20-02 LNCS 2551, pp. 199~214, 2002.

[10] ANSI, "Public Key Cryptography for the financial services industry : key agreement and key transport using elliptic curve cryptography", ANSI X9. 63, 2001.

[11] Jong-Phil Yang, Weon Shin, Kyung -Hynue Rhee, "An End-to-End Auth-entication Protocol in Wireless Application Protocol", ACISP 2001.

[12] Y. Zheng, "An Authentication and Security Protocol for Mobile Computing", Proceeding of IFIP, pp.249~257, Sep. 1996.

[13] Jaeseung Go, Kwangjo Kim, "Wireless Authentication Protocols Preserving User Anonymity", SCIS-2001, pp.159~164, Jan 2001.

[14] 이동훈, 황효선, 임채훈, "타원 곡선 암호의 기

초와 응용", (주)퓨처시스템 암호체계센터 Technical Report, Aug. 2001.

[15] N. Koblitz, "Elliptic curve crypto-systems", Mathematics of Com putation, vol.48, no.177, pp.203~209, 1987.

부 록

■ DH 기반

최종 객체[E]		인증기관[CA]
long term 키 쌍 (x_{E_E}, y_{E_E}) $y_E = g^{x_E} \pmod p$		(x_{CA}, y_{CA}) $y_{CA} = g^{x_{CA}} \pmod p$ CI_E $k_{CA} \in_R Z_q^*$ $r_E = g^{k_{CA}} \pmod p$ $s_E = x_{CA} \cdot h(CI_E \| r_E) + k_{CA}$ $Cert_E = (r_E \| s_E)$ $K = y_E^{x_{CA}}$ $C = E_K\{Cert_E \| CI_E\}$
$K = y_{CA}^{x_E}$ $D_K\{C\} = Cert_E \| CI_E$ $g^{s_E} = y_{CA}^{h(CI_E \| r_E)} \cdot r_E \pmod p$	C ←	

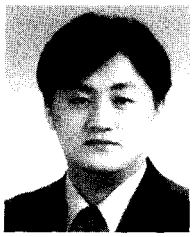
클라이언트[C]		서버[S]
[ClientHello]	g^{R_C}, V, SID, $SecNeg_C$ →	
	← g^{R_S}, V, SID, $SecNeg_S$	**[ServerHello]** $x_S = x_{E_S} + s_S$ $y_S = y_{E_S} \cdot y_{CA}^{h(CI_S \| r_S)} \cdot r_S$ $T_S = g^{k_S} \pmod p$ $S_S = x_S \cdot h(CI_S \| r_S \| T_S) + k_S$ **[Certificate]** **[CertificateRequest]** **[ServerHelloDone]**
$y_S = y_{E_S} \cdot y_{CA}^{h(CI_S \| r_S)} \cdot r_S$ $g^{S_S} = y_S^{h(CI_S \| r_S \| T_S)} \cdot T_S$ $K_p = g^{R_C R_S}$ $K_m = h(K_p \| y_S^{R_C})$ **[ClientKeyExchange]** **[CertificateVerify]** $x_C = x_{E_C} + s_C$ $y_C = y_{E_C} \cdot y_{CA}^{h(CI_C \| r_C)} \cdot r_C$ $T_C = g^{k_C} \pmod p$ $H = h(Handshake)$ $S_C = x_C \cdot h(H \| CI_C \| r_C \| T_C) + k_C$ $c = E_{K_m}\{(T_C \| S_C) \| CI_C \| r_C\}$	← $(T_S \| S_S) \| CI_S \| r_S$ c →	$K_p = g^{R_C R_S}$ $K_m = h(K_p \| g^{R_C x_S})$ $D_{K_m}\{c\} = (T_C \| S_C) \| CI_C \| r_C$ $y_C = y_{E_C} \cdot y_{CA}^{h(CI_C, r_C)} \cdot r_C$ $H = h(Handshake)$ $g^{S_C} = y_C^{h(H \| CI_C \| r_C \| T_C)} \cdot T_C$
[ChangeCipherSpec] **[Finished]** **[Application Data]**		**[ChangeCipherSpec]** **[Finished]** **[Application Data]**

〈著者紹介〉

한 종 수 (Jong-Su Han) 학생회원
2002년 2월 : 성균관대학교 정보공학과 졸업(공학사)
2002년 3월~현재 : 성균관대학교 정보통신공학부 석사 과정

정 영 석 (Young-Seok Chung) 학생회원
2002년 2월 : 성균관대학교 정보공학과 졸업(공학사)
2002년 3월~현재 : 성균관대학교 정보통신공학부 석사 과정

안 기 범 (Gi-bum Ahn) 학생회원
2002년 2월 : 성균관대학교 시스템경영공학부 졸업(공학사)
2002년 3월~현재 : 성균관대학교 정보통신공학부 석사 과정



곽 진 (Jin Kwak) 학생회원
2000년 8월 : 성균관대학교 바이오메카트로닉스공학과 졸업(공학사)
2003년 3월 : 성균관대학교 대학원 전기전자 및 컴퓨터 공학부 졸업(공학석사)
2003년 3월~현재 : 성균관대학교 정보통신공학부 박사 과정

원 동 호 (Dongho Won) 정회원
성균관대학교 전자공학과 졸업 (학사, 석사, 박사)
1978년~1980년 : 한국전자통신연구원 전임연구원
1985년~1986년 : 일본 동경공업대 객원연구원
1988년~1999년 : 성균관대학교 교학처장, 전기전자 및 컴퓨터공학부장, 정보통신대학원장, 정보통신기술연구소장
1996년~1998년 : 국무총리실 정보화추진위원회 자문위원
2002년~2003년 : 한국정보보호학회장
현재 : 성균관대학교 정보통신공학부 교수, 정통부 지정 정보보호 인증기술 연구센터장, 성균관대학교 연구처장