이동 에이전트 환경을 위한 안전한 연속 위임 구현 기법

이 현 석† · 엄 영 익††

요 약

이동 에이전트 환경에서는 에이전트의 이동성으로 인하여 에이전트의 이주(migration)가 연속적으로 발생할 수 있다. 이에 따라 에이전트를 실행할 권한을 위임(delegation)하기 위해 플레이스(place)간에 연속위임(cascaded delegation)이 발생할 수 있다. 기존의 연구는 에이전트 이주에 관련한 두 플레이스만을 위임의 대상으로 고려하기 때문에 안전한 연속 위임을 지원하지 않는다. 본 연구에서는 이동 에이전트 환경에서 연속 위임을 안전하게 수행하는 연속 위임 구현 기법을 제안한다. 제안 기법은 플레이스간의 신뢰관계에 따라 각 위임토큰(delegation token)을 다음에 생성되는 위임토큰에 내포시킨 후 서명하는 방법과 에이전트의 생성자에 의해 서명된 초기 토큰(initial token)만을 내포시킨 후 서명하는 방법을 나눠서 사용한다. 또한 본 제안 기법이 메시지 재연에 의한 공격과 위임토큰 치환 공격에 안전함을 증명한다.

Reliable Cascaded Delegation Scheme for Mobile Agent Environments

Hyun-suk Lee† · Young Ik Eom††

ABSTRACT

In mobile agent environments, migration of an agent occurs continuously due to the mobility of agents. So cascaded delegation can occur among places for delegating the privilege to execute the agent. Because the existing delegation scheme considers only the delegation between two places that participate in migration of an agent, it does not support secure cascaded delegation. In this paper, we propose a cascaded delegation scheme that provides agents with secure cascaded delegation in mobile agent environments. Depending on the trust-relationship among places, the proposed scheme achieves the goal by nesting each delegation token or by nesting only initial token signed by creator of the agent within the signed part of the next immediate delegation token. And we prove that the proposed scheme is secure against the attack of replaying a message and the attack of substituting a delegation token.

키워드 : 분산 컴퓨팅(Distributed Computing), 모바일 에이전트 시스템(Mobile Agent System), 보안(Security), 연속위임(Cascaded Delegation)

1. 서 론

이동 에이전트란 지능을 가지고 여러 호스트들을 자율적으로 이동하면서 호스트에서 제공되는 자원들을 이용하여 노드에 독립적으로 작업을 수행하거나 다른 에이전트들과 상호작용을 하면서 사용자의 작업을 수행할 수 있는 프로세스로 정의된다. 기존의 클라이언트/서버 구조에 비교해 볼 때 자율성과 이동성을 가진 이동 에이전트는 네트워크의 트래픽 감소, 클라이언트와 서버간의 비동기 연산 지원, 서비스의 분산 및 병렬 처리, 그리고 동적인 서버 인터페이스의 변경 지원과 같은 여러 장점들을 가지기 때문에 최근에는 이동 에이전트 이동성을 보다 안전하게 보장하기 위하여 에이전트 및 호스트의 인증 문제 또한 에이전트 및 호스트 보안 문제에 관한 연구들이 진행되고 있다[1-6].

본 논문에서는 에이전트 이주가 연속적으로 발생하는 경우 요구되는 연속 위임을 안전하게 수행할 수 있는 연속 위임 구현 기법을 제시한다. 위임(delegation)은 한 주체(principal)가 다른 주체에게 자신을 대신하여 에이전트를 실행시킬 수 있도록 권한을 부여하는 과정으로 정의되며, 연속 위임은 셋 이상의 주체들간에 위임이 연속적으로 수행되는 과정으로 정의된다[7, 8].

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로서 이동 에이전트 환경에서의 두 플레이스간의 위임과 분산 환경에서의 연속 위임에 대해 설명한다. 3장에서는 본 논문에서 제안하는 이동 에이전트 환경을 위한 내포된 토큰 기반의 연속 위임 구현 기법에 대해 설명하고, 동작 시나리오와 알고리즘을 통해 제안 기법의 구체적인 동작 과정에 대해 알아본다. 4장에서는 본 제안 기법의 안정성을 증명한다. 마지막으로, 5장에서는 결론 및 문제점들을 지적하고 향후 연구 과제에 대해서 기술한다.

2. 관련 연구

이동 에이전트 환경에서의 위임을 위한 대표적인 연구로

※ 이 논문은 2002년도 한국학술진흥재단의 지원에 의하여 연구되었음(KRF-2002-041-D00420).
† 준 회 원 : 성균관대학교 대학원 정보통신공학부
†† 종신회원 : 성균관대학교 정보통신공학부 교수
논문접수 : 2003년 5월 6일, 심사완료 : 2003년 12월 23일

Berkovits의 연구를 들 수 있다[2]. 이 연구는 이동 에이전트 환경에서의 가능한 4가지 시나리오를 제시하였으며, 각각의 시나리오에 따라 이동 에이전트의 이주 시 에이전트의 실행 주체가 변경되어야 함을 제시하고 있다. 그러나 이 연구는 이주와 관련된 두 플레이스간의 위임만을 고려하므로 셋 이상의 플레이스간에 요구되는 안전한 연속 위임에는 부적절하다. 또한 기존의 연속 위임 기법은 Varadharajan et al.에 의해 제시된 내포된 토큰(nested tokens) 방식, Low and Christianson에 의해 제시된 signatures to bind two tokens 방식, Neuman 에의해 제시된 delegation key to bind two tokens 방식, 그리고 Y. Ding and H. Petersen에 의해 제시된 hierarchical delegation tokens 방식이 있다[7]. 각각의 연속 위임 기법은 명확히 다르며, 본 논문은 그 기법들 중 Varadharajan et al.이 분산시스템 이론으로 만든 연속 위임 기법을 이동 에이전트 환경에 적용하여 안정성을 보장한다. 본 장에서는 이동 에이전트 환경에서의 두 플레이스간의 기존 위임 기법과 분산 환경에서의 연속 위임에 대해 설명한다.

2.1 이동 에이전트 환경에서의 두 플레이스간의 위임

Berkovits의 연구에서는 이주와 관련된 주체들간의 신뢰관계에 따라 에이전트의 실행 주체(executing principal)가 변경된다. 플레이스 I_1에서 주체 P_1으로 에이전트 A를 실행시키고 플레이스 I_2로 에이전트를 이주시켜 새로운 주체 P_2로 에이전트를 실행시키는 상황을 가정할 때, I_2가 I_1에 의해서 신뢰를 받는지 아니면 A에 의해 신뢰를 받는지에 따라서 4가지 다른 신뢰관계를 표현하는 P_2의 값이 달라진다. 즉, I_2가 A를 실행시키기 위한 권한이 변경되는 것이다. 4가지 신뢰관계는 다음과 같다.

① 플레이스 핸드오프(place handoff) : 4가지의 신뢰관계 중 신뢰도가 가장 높은 관계이다. 이 경우 I_1은 에이전트를 I_2로 핸드오프 할 수 있다. 그 후 I_2는 P_1을 대신하여 에이전트를 실행하게 된다.

② 플레이스 위임(place delegation) : 플레이스 핸드오프보다 낮은 신뢰도를 가지는 관계이며 이 경우 I_1은 I_2로 에이전트를 위임할 수 있다. 그 후 I_2는 P_1의 권한에 자신의 권한을 추가하여 에이전트를 실행시킨다.

③ 에이전트 핸드오프(agent handoff) : 에이전트는 직접 I_2로 자신을 핸드오프 할 수 있다. 그 후 I_2는 P_1에 상관없이 에이전트 자체의 권한으로 에이전트를 실행하게 된다.

④ 에이전트 위임(agent delegation) : 에이전트는 I_2로 자신을 위임할 수 있다. 그 후 I_2는 에이전트 자체의 권한에 자신의 권한을 추가하여 P_1에 상관없이 에이전트를 실행하게 된다.

신뢰관계에 따른 에이전트 실행 주체의 변경은 <표 1>과 같다. 이는 Lampson의 형식 이론(formal theory)[9, 10]을 기반으로 하여 안정성이 증명된 인증 기법이다. 여기에서 A for S는 송신자에 의해 서명된 에이전트의 자체 권한을 의미한다.

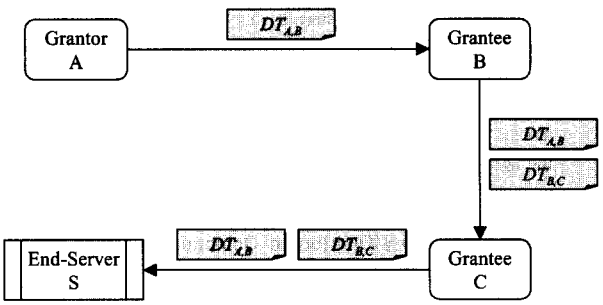
〈표 1〉 신뢰관계에 따른 실행 주체의 변경

에이전트 이주요청	신뢰 관계	플레이스 I_1의 실행주체	플레이스 I_2에서의 실행 주체
플레이스	플레이스 핸드오프	P_1	P_2 == P_1
플레이스	플레이스 위임	P_1	P_2 == I_2 for P_1
에이전트	에이전트 핸드오프	P_1	P_2 == A for S
에이전트	에이전트 위임	P_1	P_2 == I_2 for A for S

그러나 Berkovits의 연구는 신뢰관계의 정의에 있어서 이주와 관련된 두 플레이스만을 고려하기 때문에, 셋 이상의 플레이스들간의 안전한 연속 위임에는 부적절하다.

2.2 분산 환경에서의 연속 위임

연속 위임은 셋 이상의 주체들간에 위임이 연속적으로 수행되는 과정으로 정의된다. (그림 1)은 위임자(grantor) A에 의해 위임이 시작되어 피위임자(grantee) B, C로 연속 위임이 일어난 후, 종단 서버 S로 서비스를 요청하는 과정을 보인다.



(그림 1) 연속 위임의 동작 과정

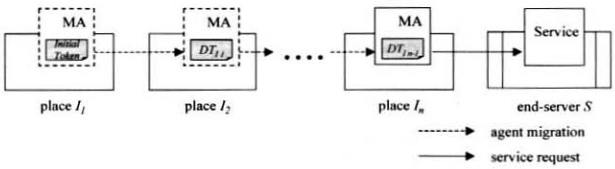
연속 위임을 위하여 위임토큰을 사용하며 위임토큰 $DT_{X,Y}$는 플레이스 X가 플레이스 Y로 위임할 권한과 위임기한을 나타내는 타임스탬프 등을 포함하여 연속 위임을 수행한다. 그러나 (그림 1)에서는 위임토큰들간에 안전한 관계를 맺지 않았으므로 위임토큰 치환 공격으로부터 안전하지 않을 수 있다.

3.안전한 연속 위임 기법

본 논문은 Varadharajan et al.이 분산시스템 이론으로 만든 내포된 토큰 방식을 이동 에이전트 환경으로 가져와서 적용시켰고, challenge-response 기반 인증 프로토콜을 추가하여 Berkovits 논문에서 제시된 이동 에이전트 환경에서의 가능한 4가지 시나리오에 맞게 적용시켜 이동 에이전트 환경에서 연속 위임을 안전하게 수행하는 연속 위임 구현 기법을 제안한다. 우선 본 제안 기법의 시스템 구조와 자료구조를 설명하고 알고리즘과 동작 시나리오를 통해 본 제안 기법의 동작을 설명한다.

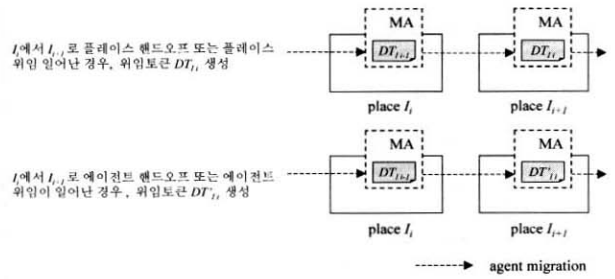
3.1 시스템 구조

(그림 2)는 에이전트 MA가 플레이스 I_1부터 I_n까지 이주하고 위임토큰 DT_{In-1}을 기반으로 종단 서버 S에게 서비스를 요청하는 과정을 보인다.



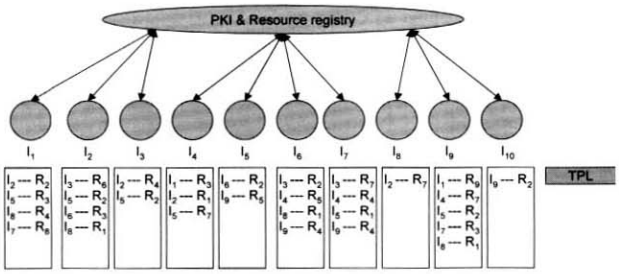
(그림 2) 이동 에이전트 환경에서의 연속 위임

(그림 2)에서 I_1은 에이전트 MA의 생성자에 의해 서명된 초기토큰을 가지고 MA를 실행한다고 가정한다. I_1에서 실행중인 MA는 I_2로 위임할 권한을 담은 DT_{I1}을 생성하여 MA에 포함시킨 후 MA를 I_2로 전송한다. 이 때 두 플레이스간의 인증은 공개키 기반(PKI : Public Key Infrastructure)의 인증방식을 사용한다. MA를 받은 I_2는 I_3로 위임할 권한을 담은 DT_{I2}를 생성하야 MA에 포함시킨 후 MA를 I_3로 전송한다. 이와 같은 연속 위임 과정을 거쳐 I_n에 도착한 MA는 DT_{In-1}을 기반으로 종단 서버 S에게 서비스를 요청한다. 임의의 위임토큰 $DT_{Ii}(1 \le i \le n-1)$는 신뢰관계에 따라 (그림 3)과 같이 DT_{Ii} 또는 DT'_{Ii}로 다르게 생성된다.



(그림 3) 신뢰관계에 따른 두가지의 토큰방식

이러한 위임과정을 수행하기 위해서는 이주에 관련된 플레이스들간의 신뢰관계를 알아내야 한다. 따라서 플레이스들은 자체적으로 신뢰할 수 있는 플레이스들의 목록(TPL : Trustable Place List)을 가지고 있어야 한다. TPL은 Berkovits의 연구에서 제시된 플레이스간의 4가지 신뢰관계에 따라 현재 이동 에이전트가 실행되고 있는 플레이스가 이주될 플레이스의 신뢰관계를 알 수 있는 정보와 그 플레이스들이 어떠한 자원을 제공하고 있는지에 대한 정보도 같이 가지고 있다. 이동 에이전트가 종단 서버에게 서비스를 제공받기 위하여 다른 플레이스로 이주할 때, 현재 이동 에이전트가 실행되고 있는 플레이스가 이주될 플레이스의 신뢰여부를 알기 위하여 PKI(Public Key Infrastructure)와 Resource registry에 접속하여 TPL을 생성하게 된다. 이를 위해 (그림 4)와 같은 추가적인 구조가 필요하다.



(그림 4) 신뢰관계를 알아내기 위한 추가 구조

본 제안 기법에서는 플레이스간의 인증을 위해서 공개키 기반의 인증방식을 사용하여 Berkovits et al.의 연구에서 제시된 4가지 시나리오와 그에 따른 인증방식을 수행한다. 따라서 플레이스들은 TPL을 주기적으로 갱신하기 위하여 PKI에 접속해야 한다. 또한 TPL은 다른 플레이스들이 어떤 자원을 제공하는가에 대한 정보를 가지고 있는 Resource registry에 주기적으로 접근하여 새로운 정보로 갱신하여야 한다. 따라서 주기적으로 갱신된 TPL을 가지고 플레이스는 다음 절에 나오는 통신 메시지와 위임토큰을 사용하여 Berkovits의 논문에서 제시된 이동 에이전트 환경에서 가능한 4가지 시나리오를 안전하게 수행한다.

3.2 자료 구조

본 절에서는 제안 기법에서 사용하게 될 플레이스들간의 통신 메시지 구조, 종단 서버에 서비스를 제공받기 위한 통신 메시지 그리고 위임토큰의 구조에 대해서 설명한다. (그림 5)는 제안 기법에서 사용하는 통신 메시지들과 위임토큰의 구조를 보인다.

DR:	Delegator_ID	Migration_Type	Timestamp	Nonce	
DA:	Delegatee_ID	$Sig_{Delegatee}$(DR's Nonce)	Timestamp	Nonce	
SRC:	Requestor_ID	Timestamp	Nonce		
SRR:	Requested_ID	$Sig_{Requested}$(SRC's Nonce)	Timestamp	Nonce	
SR:	Requester_ID	$Sig_{Requester}$(Server's Nonce)	Delegation_token	Timestamp	Service_ID
SA:	Server_ID	Service_ID	Result		
DT_{Ii}:	Delegator_ID	Delegatee_ID	Privilege	$Sig_{Delegator}$(DA's Nonce)	Timestamp	DT_{Ii-1}
DT'_{Ii}:	Delegator_ID	Delegatee_ID	Privilege	$Sig_{Delegator}$(DA's Nonce)	Timestamp	Initial Token

(그림 5) 통신 메시지와 위임토큰 구조

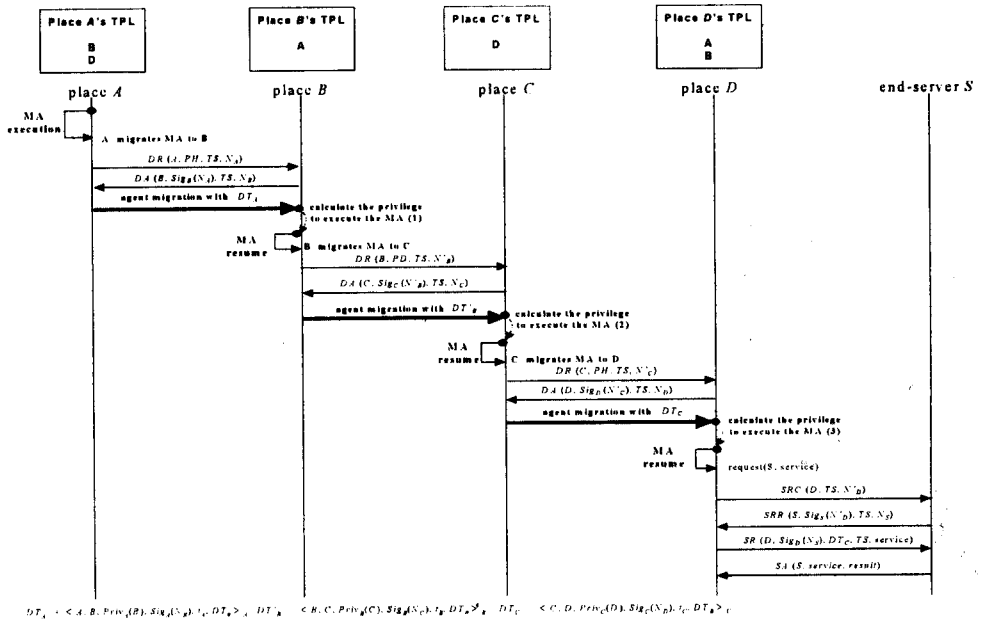
DR 메시지와 *DA* 메시지는 플레이스간에 인증을 하기 위해 사용된다. *SRC* 메시지와 *SRR* 메시지는 위임의 마지막 플레이스와 종단 서버간의 인증을 하기위해 사용된다. *SR* 메시지와 *SA* 메시지는 각각 종단 서버에 서비스를 요청하고 서비스의 결과를 받기위해 사용된다. 위임토큰 *DT*는 위임받는 플레이스를 인증하고 에이전트를 실행할 권한을 넘겨주기 위하여 생성된다. 본 제안 기법에는 두 가지의 위임토큰이 존재한다. 에이전트 이주 시 플레이스간의 신뢰관계가 플레이스 핸드오프 또는 플레이스 위임인 경우 I_i는 DT_{Ii-1}을 축적시켜 DT_{Ii}를 생성한다. 그러나 에이전트 핸드

오프 또는 에이전트 위임인 경우에는 이전의 내포된 위임 토큰 DT_{Ii-1}을 버리고 생성자에 의해서 서명된 초기 토큰만 내포시켜 DT'_{Ii}를 생성한다. 이는 I_i 이전의 플레이스들에게 위임받은 권한이 더 이상 필요지 않으며 또한 그 플레이스들에 의해서 내포된 위임토큰이 없이도 위임토큰 치환공격에 안전하기 때문이다.

3.3 동작 시나리오

본 절에서는 제안 기법의 동작 과정을 두가지의 시나리오를 통해서 구체적으로 알아본다. 아래의 시나리오에서 *X migrates MA to Y*는 플레이스 X가 플레이스 Y로 에이전트 MA를 이주시키려 하는 것이고, *migrate(Y)*는 에이전트가 현재 자신이 실행되고 있는 플레이스에게 플레이스 Y로 이주를 요청하는 것이다. 시나리오는 모두 플레이스 A, B, C, D를 이주하면서 위임을 받은 에이전트 MA가 위임토큰을 기반으로 종단 서버 S게 서비스를 요청하는 과정이다. 그러나 시나리오 1에서는 플레이스 간에 플레이스 핸드오프, 플레이스 위임, 플레이스 핸드오프 순서로 연속위임이 발생하고, 시나리오 2에서는 에이전트 위임, 플레이스 핸드오프, 에이전트 핸드오프 순서로 위임이 발생한다.

시나리오 1에서(그림 6) A는 MA를 실행한다. A가 MA를 B로 이주시키려 하면, 우선 A는 자신의 TPL 목록에 B가 있는지 확인한다. B가 자신의 TPL에 있으므로 A는 자신의 식별자, PH, 타임스탬프 그리고 난수 N_A로 구성된 DR 메시지를 B로 보내게 된다. DR 메시지의 타임스탬프 동안 DR 메시지를 수신한 B가 응답 메시지로 B의 식별자, N_A에 대한 B의 서명, 타임스탬프 그리고 난수 N_B로 구성된 DA메시지를 전송하면, A는 DA 메시지의 $Sig_B(N_A)$에 대한 인증과정을 거쳐 B에 대한 인증을 수행하고 인증이 성공적으로 이루어지면 위임토큰 DT_A를 생성하고 자신의 비밀키로 서명하여 MA에 포함시킨 후 MA를 B로 보낸다. DT_A는 A의 식별자, B의 식별자, A가 B로 위임하는 권한, N_B에 대한 A의 서명, DT_A의 타임스탬프 그리고 에이전트의 생성자에 의해 서명된 초기 토큰으로 구성된다. A로부터 MA를 받은 B는 DT_A의 $Sig_A(N_B)$에 대한 검증과정을 거치고 검증이 성공적으로 이루어지면 MA를 실행시키기 위한 권한을 계산한다. A로부터 전송된 DR메시지에 신뢰관계가 PH이므로 B는 DT_A에 있는 권한으로 MA의 실행을 재개하게 된다. 만약 B가 MA를 C로 이주시키려 하면, B는 자신의 TPL목록에 C가 있는지 확인한다. C가 자신의 TPL에 없으므로 B는 자신의 식별자, PD, 타임스탬프 그리고 난수 N'_B로 구성된 DR 메시지를 C로 보내게 된다. DR 메시지를 수신한 C는 B에서 A를 인증하는 과정과 동일한 과정을 거쳐 DA메시지를 B에게 보내게 된다. DA메시지를 받은 B는 DT'_B를 생성하고 자신의 비밀키로 서명하여 MA에 포함시킨 후 MA를 B로 보낸다. DT_B는 B의 식별자, C의 식별자, B가 C로 위임하는 권한, N_C에 대한 B의 서명, DT'_B의 타임스탬프 그리고 에이전트의 생성자에 의해 서명된 초기 토큰으로 구성된다. 이 경우 DT_A를 DT'_B에 내포시키지 않고 초기토큰만을 포함시키는 것은 B에서 C로 MA의 이주 시, 위임의 신뢰관계가 플레이스 위임이기 때문이다. B로부터 MA를 받은 C는 DT'_B의 $Sig_B(N_C)$에 대한 검증과정을 거치고 검증이 성공적으로 이루어지면 MA를 실행시키기 위한 권한을 계산한다. B로부터 전송된 DR 메시지에 신뢰관계가 PD이므로 C는 DT_B에 있는 권한에 자신의 권한을 추가하여 MA의 실행을 재개하게 된다. 만약 C가 D로 MA를 이주시키려 하면, A가 B로 MA를 전송한 과정과 동일한 과정을 거쳐 MA를 D로 전송한다. 이 과



(그림 6) 시나리오 1

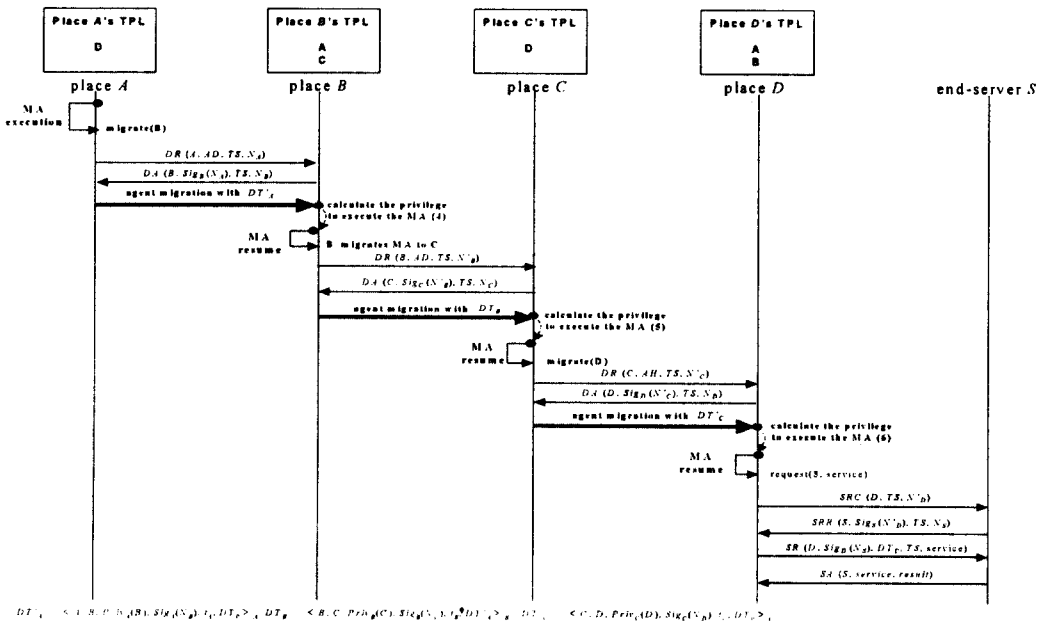
정에서 생성되는 DT_C는 C의 식별자, D의 식별자, C가 D로 위임하는 권한, N_D에 대한 C의 서명, DT_C의 타임스탬프 그리고 위임토큰 DT'_B로 구성된다. DT_C에 이전의 플레이스로부터 전송된 위임토큰 DT'_B를 내포시키는 것은 위임의 신뢰관계가 플레이스 핸드오프이기 때문이다. MA를 수신한 D는 DT_C에 있는 권한으로 MA를 실행을 재개하게 된다. 이 때 실행중인 MA가 S로 서비스를 요청하면 D는 자신의 식별자, 타임스탬프 그리고 난수 N_D로 구성된 SRC 메시지를 S로 전송한다. SRC 메시지를 받은 S는 자신의 식별자, N_D에 대한 S의 서명, 타임스탬프 그리고 난수 N_S로 구성된 SRR 메시지를 D로 전송하게 된다. SRR 메시지를 받은 D는 $Sig_S(N_D)$에 대한 인증과정을 수행하고, 인증이 성공적으로 이루어지면 자신의 식별자, N_S에 대한 D의 서명, 타임스탬프 그리고 요청할 서비스의 식별자로 구성된 SR 메시지를 생성하여 S로 전송한다. SR 메시지를 받은 S는 $Sig_D(N_S)$에 대한 검증과정을 거치고 검증이 성공적으로 이루어지면, SR 메시지에 포함된 DT_C에 MA의 생성자에 의해 서명된 초기토큰이 있는지 확인한다. 초기토큰이 존재하면 S는 MA가 요청한 서비스를 수행하고 그 수행결과를 SA 메시지에 포함하여 SA 메시지를 D로 보내게 된다.

시나리오 2는(그림 7) 시나리오 1과 유사한 과정을 거친다. 그러나 플레이스간의 신뢰관계에 따라 위임토큰의 생성 방식이 다르며, MA를 위임받은 플레이스가 MA의 실행을 재개할 때 가지는 권한이 서로 다르다는 점에서 시나리오 1과 차이점이 있다.이러한 차이점은 다음과 같다. A로부터 B로 MA의 이주가 발생될 경우, 시나리오 2에서는 MA가 자신이 실행되고 있는 A에게 B로의 이주를 요청한다. 이 경우 A의 TPL에 B가 없으므로 에이전트 위임의 신뢰관계 AD로 위임이 발생한다. 따라서 앞서 <표 1>에서 보여진 것과 같이 송신자에 의해 서명된 MA의 자체권한이 DT'_A의 권한부분에 포함된다. DT'_A가 포함된 MA를 받은 B는 송신자에 의해 서명된 MA의 자체권한에 자신의 권한을 추가하여 MA의 실행을 재개하게 된다. 만약 MA를 실행하고 있는 B가 C로 MA를 이주시키려 하면, B의 TPL에 C가 있으므로 플레이스 핸드오프의 신뢰관계로 연속위임이 발생하고 이 경우 시나리오 1에서 A에서 B로 또는 C에서 D로 연속위임이 발생되는 경우와 동일한 과정을 거쳐 연속위임이 발생한다. 이 경우에는 C는 DT_B에 있는 권한을 가지고 MA의 실행을 재개한다. C에서 실행되고 있는 MA가 C에게 D로의 이주를 요청하는 경우는 A에서 B로 위임이 발생되는 경우와 동일하며 D에서 MA가 다시 실행되는 도중 S에게 서비스를 요청하는 과정은 시나리오 1에서의 과정과 동일하다.

3.4 알고리즘

본 제안에서 전체 위임경로는 $I_1I_2I_3\cdots I_{n-1}I_nS$라고 가정한다. I_1에서 P_1의 권한을 가지고 실행되고 있는 MA는 S에게 서비스를 제공받기 위하여 $I_2I_3\cdots I_{n-1}I_n$을 순서대로 이주한다. 이때 각 플레이스 $I_i(2 \le i \le n)$에서는 MA를 실행하기 위한 변경된 권한을 위임받는다. MA가 위임경로 상 마지막 플레이스인 I_n에 도착한 후, 내포된 토큰을 기반으로 S에게 서비스를 요청한다. 본 제안 기법은 다음의 4가지로 나뉜다 : ① 위임할 플레이스 I_1은 I_2에게 연속 위임을 시작한다. ② 중간에 있는 플레이스 I_i $(2 \le i \le n-1)$는 연속 위임을 중재한다. ③ 연속 위임의 마지막 플레이스 I_n은 종단 서버 S에게 서비스를 요청한다. ④ 종단 서버 S는 I_n이 요청한 서비스를 제공한다.

플레이스 I_1이 MA를 I_2로 이주시키려 하거나 I_1에서 P_1



(그림 7) 시나리오 2

의 권한으로 실행되고 있는 MA가 I_2로 이주를 요청한 경우, I_1은 (알고리즘 1)을 사용하여 연속 위임을 초기화한다. 동시에 I_2는 I_1을 인증하고 자신을 I_1에게 인증받기 위하여 (알고리즘 2)의 Segment A을 사용한다. 아래의 알고리즘에서, TS는 각 메시지의 유효한 기간을 의미한다. *DR* 메시지의 MT는 Migration Type을 의미하며 PH(플레이스 핸드오프), PD(플레이스 위임), AH(에이전트 핸드오프) 그리고 AD(에이전트 위임) 중 하나의 값이 들어간다. N_X 는 플레이스 X에 의하여 생성된 난수를 의미하며, $Sig_Y(N_X)$는 플레이스 X에 의해 생성된 난수에 플레이스 Y의 공개키 기반 디지털 서명을 하는 것을 의미한다. 또한, $Priv_X(Y)$는 Y가 에이전트를 실행할 때 가지는 권한을 플레이스 X가 위임하는 것을 의미하며 Y는 *DR* 메시지의 Migration Type에 따라 위임받은 권한을 그대로 사용하거나 자신의 권한을 추가하여 MA를 실행하게 된다. 그리고 t_X는 플레이스 X에 의해 생성된 위임토큰의 유효한 기간을 의미한다.

Algorithm 1 (Initiation Algorithm)
dispatch and execute *MA* with DT_{I0} on I_1 ;
// DT_{P0} is the token of *MA*'s creator
when (I_1 migrates *MA* to I_2) {
  if (I_2 is in TPL)
    send $DR(I_1, PH, TS, N_{I1})$ message to I_2 ;
  else
    send $DR(I_1, PD, TS, N_{I2})$ message to I_2 ;
}
when (*migrate*(I_2) is executed in *MA*) {
  if (I_2 is in TPL)
    send $DR(I_1, AH, TS, N_{I1})$ message to I_2 ;
  else
    send $DR(I_1, AD, TS, N_{I1})$ message to I_2 ;
}
wait for $DA(I_2, Sig_{I2}(N_{I1}), TS, N_{I2})$ message from I_2 during *TS* in *DR* ;
if (*DA* message arrives from I_2 during *TS*) {
  auth_result ← *verify* ($Sig_{I2}(N_{I1})$) ; // $Sig_{I2}(N_{I1})$ is in *DA*
  if (auth_result is FALSE) {
    discard *MA* ;
    return FALSE ; // terminate cascaded delegation
  }
  if (MT == *PH* or *PD*) { // MT is in *DR*
    DT_{I1} ← *createDT*(I_1, I_2, $Priv_{I1}(I_2)$, $Sig_{I1}(N_{I2})$, t_{I1}, DT_{I0}) ;
    send *MA* with DT_{I1} to I_2 ;
  }
  else {
    DT'_{I1} ← *createDT*(I_1, I_2, $Priv_{I1}(I_2)$, $Sig_{I1}(N_{I2})$, t_{I1}, DT_{I0}) ;
    send *MA* with DT'_{I1} to I_2 ;
  }
  return TRUE ;
}
return FALSE ;

(알고리즘 1) 연속위임의 초기 알고리즘

우선 MA의 이주가 일어나는 상황은 플레이스가 에이전트를 이주시키는 경우와 에이전트가 플레이스에게 이주를 요청하는 두 가지로 나뉜다. (알고리즘 1)에서 I_1이 MA를 이주시키려 하는 경우, I_2가 I_1의 TPL에 있으면 플레이스 핸드오프로 에이전트를 이주시키고①, 없으면 플레이스 위임으로 에이전트를 이주시킨다②. 그러나 I_1에서 실행중인 MA가 I_1에게 이주를 요청하는 경우, I_2가 I_1의 TPL에 있으면 에이전트 핸드오프로 에이전트를 이주시키고③, 없으면 에이전트 위임으로 에이전트를 이주시킨다④. 만약 (알고리즘 1)이 TRUE를 반환하면(이것은 연속 위임이 성공적으로 초기화됐음을 의미한다), DT_{I1}을 포함한 MA는 I_2에 도착한다. 그러면 I_2는 *DR*메시지안의 Migration Type에 따라 MA를 재실행한다. 즉, Lampson의 형식 이론을 기반으로 ①과 ③의 경우에 각각의 주체는 I_1, A for S로서 DT_{I1}안의 privilege만으로 MA를 재실행하고, ②와 ④의 경우에 각각의 주체는I_2 for P_1, I2 for A for S로서 DT_{I1}안의 privilege에 자신 I_2의 권한을 추가하여 MA를 재실행한다. 계속해서 I_2에서 I_3으로 에이전트의 이주가 일어나면 I_3은 (알고리즘 2)의 Segment B를 사용한다. 동시에 I_3는 I_2를 인증하고I_2에게 자신을 인증시키기 위하여 (알고리즘 2)의 Segment A를 사용한다. 만약 (알고리즘 2)에서 TRUE를 반환하면 DT_{I2}를 포함한 MA는 I_3에 도착한다. 이 과정은 I_3, I_4 … I_{n-1}까지 반복된다. 만약I_{n-1}까지의 연속 위임이 성공적으로 끝나면 I_n에 DT_{In-1}을 포함한 MA가 도착한다.

Algorithm 2 (Mediation Algorithm)
/* I_i receives $DR(I_{i-1}, MT, TS, N_{Ii-1})$ message from I_{i-1} */
/* **Segment A** : Contacting with I_{i-1} */
send $DA(I_i, Sig_{Ii}(N_{Ii-1}), TS, N_{Ii})$ message to I_{i-1} ;
when (*MA* with DT_{Ii-1} arrives from I_{i-1} during *TS* in *DA*) {
  auth_result ← *verify* $Sig_{Ii-1}(N_{Ii})$; // $Sig_{Ii-1}(N_{Ii})$ is in DT_{Ii-1}
  if (auth_result is FALSE) {
    discard *MA* ;
    return FALSE ; // terminate cascaded delegation
  }
} /***End of segment A***/
if (MT == *PH* or *AH*) { // MT is in *DR*
  resume execution of *MA* with the privilege in DT_{Ii-1} ;
else { // MT == *PD* or *AD*
  new privilege ← I_i's own privilege *for* the privilege in DT_{Ii-1} ;
    // the *for* operator is defined in Lampson's Theory
    resume execution of *MA* with the new privilege ;
}
/* **Segment B** : Contacting with I_{i+1} */
when (I_i migrates *MA* to place I_{i+1}) {
  if (I_{i+1} is in TPL)
    send $DR(I_i, PH, TS, N'_{Ii})$ message to I_{i+1} ;
  else
    send $DR(I_i, PD, TS, N'_{Ii})$ message to I_{i+1} ;
}
when (*migrate*(I_{i+1}) is executed in *MA*) {
  if (I_{i+1} is in TPL)
    send $DR(I_i, AH, TS, N'_{Ii})$ message to I_{i+1} ;
  else
    send $DR(I_i, AD, TS, N'_{Ii})$ message to I_{i+1} ;
}
wait for $DA(I_{i+1}, Sig_{Ii+1}(N'_{Ii}), TS, N_{Ii+1})$ message from I_{i+1} during *TS* in *DR* ;
if (*DA* message arrives from I_{i+1} during *TS*) {
  auth_result ← *verify* ($Sig_{Ii+1}(N'_{Ii})$) ; // $Sig_{Ii+1}(N'_{Ii})$ is in *DA*
  if (auth_result is FALSE) {
    discard *MA* ;
    return FALSE ; // terminate cascaded delegation
  }
  if (MT == *PH* or *PD*) { // MT is in *DR*

```
    DT_Ii ← createDT(I_i, I_i+1, Priv_Ii(I_i+1), Sig_Ii(N_Ii+1), t_Ii, DT_Ii-1) ;
    send MA with DT_Ii to I_i+1 ;
  }
  else { // MT == AH or AD
    DT'_Ii ← createDT(I_i, I_i+1, Priv_Ii(I_i+1), Sig_Ii(N_Ii+1), t_Ii, DT_I0) ;
    send MA with DT'_Ii to I_i+1 ;
  } /*End of segment B*/
  return TRUE ;
}
return FALSE ;
```

(알고리즘 2) 연속위임의 중재 알고리즘

만약 I_n에서 실행중인 MA가 종단 서버 S에 서비스를 요청하면 I_n은 (알고리즘 3)을 사용하여 SR 메시지를 S에게 보낸다. 그러면 종단 서버 S는 (알고리즘 4)를 사용하여 SR 메시지 안에 있는 I_n의 디지털 서명 $Sig_{In}(N_S)$와 위임토큰 DT_{In-1}을 확인하는 과정을 거치고 이 과정이 성공적으로 끝나면 S는 I_n으로부터 요청된 서비스를 수행하여 그 결과를 SA 메시지에 담아서 I_n에 보내게 된다.

```
Algorithm 3 (Service Request Algorithm)
/* contacting with S */
when request (S, service) is executed in MA {
  send SRC(I_n, TS, N'_In) message to S ;
  wait for SRR (S, Sig_S(N'_In), TS, N_S) message from S during
    TS in SRC ;
  if (SRR message arrives from I_i+1 during TS in SRC) {
    auth_result ← verify (Sig_S(N'_In)) ; // Sig_S(N'_In) is in SRR
    if (auth_result is FALSE) {
      discard MA ;
      return FALSE ;
    }
  }
  send SR (I_n, Sig_In(N_S), DT_In-1, TS, service) message to S ;
  wait for SA (S, service, result) message from S during TS
  in SR ;
}
return TRUE ;
```

(알고리즘 3) 서비스 요청 알고리즘

```
Algorithm 4 (Service Response Algorithm)
/* S receives SRC(I_n, TS, N'_In) message from I_n */
send SRR (S, Sig_S(N'_In), TS, N_S) message to I_n ;
when (SR message arrives from I_n during TS in SRR) {
  auth_result verify (Sig_In(N'_S)) ; // Sig_In(N'_S) is in SR
  if (auth_result is FALSE) {
    discard MA ;
    return FALSE ;
  }
  else {
    if (DT_0 exists) { // DT_0 exists in DT_In within SR message
    execute the service requested from I_n ;
    send SA (S, service, result) message to I_n ;
   }
   else
    return FALSE ;
  }
}
return TRUE ;
```

(알고리즘 4) 서비스 응답 알고리즘

4. 안정성 증명

본 제안 기법은 공개키 기반 디지털 서명 방식을 취하고 있으므로 인증된 플레이스만이 연속 위임에 참여할 수 있다는 사실을 전제로 한다. 본 장에서는 제안 기법이 메시지 재연 공격(replay attack)과 위임토큰 치환 공격(substituting attack)으로부터 안전함을 증명한다.

[정리 1] 본 제안 기법은 메시지 재연 공격에 안전하다.
증명 : 전체 위임경로 $I_1I_2I_3\cdots I_{n-1}I_nS$로 가정한 경우, 악의있는 플레이스 I'_i $(1 \le i \le n)$가 메시지 재연 공격을 시도한다고 가정한다. 만약 I'_i가 DT_{Ii} 메시지 또는 SR 메시지의 복사본을 만들 수 있다고 할 때, I'_i는 메시지를 재사용하려고 할 수 있다. 그러나 DT_{Ii} 메시지 안의 $Sig_{Ii}(N_{I+1})$ 또는 SR 메시지의 안의 $Sig_{In}(N_S)$에서 각각의 난수 N_{I+1}와 N_S는 무작위로 생성되어 한 번 사용된 후 버려지므로 위와 같은 메시지 재연 공격은 성공할 수 없다. 그러므로 본 제안 기법은 메시지 재연 공격에 안전하다. □

[정리 2] 본 제안 기법은 위임토큰 치환 공격에 안전하다.
증명 : 두가지 다른 위임경로를 가지는 연속위임의 경우, (A) MA가 플레이스 I_1, I_i, I_j, I_k 그리고 I_n을 순서대로 이주하다가 종단 서버 S에게 서비스를 요청하는 경우와 (B) 이전에 MA가 플레이스 I_1, I'_i, I_j, I_h $(h \neq k)$ 그리고 I_n을 순서대로 이주하다가 종단 서버 S에게 서비스를 요청했을 경우를 고려한다. (A)의 경우 위임경로 $Dpath1$: $I_1I_iI_jI_kI_nS$가 생성된다. 이 때 플레이스간의 신뢰관계는 모두 플레이스 핸드오프 또는 플레이스 위임이라고 가정하면 위임경로 $Dpath1$에서 사용되는 위임토큰들은 다음과 같이 생성될 수 있다.

〈표 2〉 위임경로 $Dpath1$에서의 위임토큰

$I_1 \rightarrow I_i$	$DT_{I1} = \langle I_1, I_i, Priv_{I1}(I_i), Sig_{I1}(N_{Ii}), t_{I1}, DT_{I0} \rangle_{I1}$
$I_i \rightarrow I_j$	$DT_{Ii} = \langle I_i, I_j, Priv_{Ii}(I_j), Sig_{Ii}(N_{Ij}), t_{Ii}, DT_{I1} \rangle_{Ii}$
$I_j \rightarrow I_k$	$DT_{Ij} = \langle I_j, I_k, Priv_{Ij}(I_k), Sig_{Ij}(N_{Ik}), t_{Ij}, DT_{Ii} \rangle_{Ij}$
$I_k \rightarrow I_n$	$DT_{Ik} = \langle I_k, I_n, Priv_{Ik}(I_n), Sig_{Ik}(N_{In}), t_{Ik}, DT_{Ij} \rangle_{Ik}$

(B)의 경우 위임경로 $Dpath2$: $I_1I'_iI_jI_hI_nS$가 생성된다. 이 때 플레이스간의 신뢰관계는 모두 플레이스 핸드오프 또는 플레이스 위임이라고 가정하면 위임경로 $Dpath2$에서 사용되는 위임토큰들은 다음과 같이 생성될 수 있다.

〈표 3〉 위임경로 $Dpath2$에서의 위임토큰

$I_1 \rightarrow I'_1$	$DT_{I1} = \langle I_1, I'_i, Priv_{I1}(I'_i), Sig_{I1}(N_{I'i}), t_{I1}, DT_{I0} \rangle_{I1}$
$I'_i \rightarrow I_j$	$DT_{I'I} = \langle I'_i, I_j, Priv_{I'i}(I_j), Sig_{I'i}(N_{Ij}), t_{I'i}, DT_{I1} \rangle_{I'I}$
$I_j \rightarrow I_h$	$DT_{Ij} = \langle I_j, I_h, Priv_{Ij}(I_h), Sig_{Ij}(N_{Ih}), t_{Ij}, DT_{I'i} \rangle_{Ij}$
$I_h \rightarrow I_n$	$DT_{Ih} = \langle I_h, I_n, Priv_{Ih}(I_n), Sig_{Ih}(N_{In}), t_{Ih}, DT_{Ij} \rangle_{Ih}$

이 때 공격자가 $Dpath1$과 $Dpath2$에서 사용된 위임토큰

들을 불법적으로 획득한 후 유효하지 않은 위임경로 *Dpath3* : $I_1I_iI_jI_hI_nS$를 생성하기 위하여 위임경로 *Dpath2*의 위임토큰 $DT_{I'i}$를 *Dpath1*의 DT_{Ii}로 치환하려고 할 수 있다. 그러나 위임경로 *Dpath2*의 I_j에서 위임토큰을 생성할 때, $DT_{I'i}$를 내포시킨 후, I_j의 비밀키로 서명하게 된다. 따라서 공격자는 I_j의 비밀키를 알 수 없기 때문에 위임경로 *Dpath2*에서 사용된 DT_{Ij}안의 $DT_{I'i}$를 *Dpath2*에서 사용된 DT_{Ii}로 치환하려는 공격은 실패하게 된다. 따라서 본 제안기법은 위임토큰 치환공격에 안전하다. 이는 본 제안기법이 내포된 토큰방식을 사용하기 때문에 위임토큰 치환 공격을 탐지할 수 있는 특성에 기인한 것이며 플레이스간의 신뢰관계와 관계없이 공격을 탐지할 수 있다. □

5. 결 론

이동 에이전트 환경에서의 위임을 위한 기존의 연구는 이주와 관련된 두 플레이스만을 위임의 대상으로 고려하기 때문에, 안전한 연속 위임을 지원하지 않는다. 본 논문에서는 Berkovits 논문에서 제시된 이동 에이전트 환경에서의 가능한 4가지 시나리오에서 플레이스들간의 연속 위임을 안전하게 수행할 수 있는 방법을 제시하였다. 제안 기법으로 연속 위임을 수행하는 경우, 각각의 플레이스가 다른 플레이스로 권한을 위임할 때 플레이스간의 신뢰관계에 따라 다른 위임토큰의 생성과 권한변경 방식을 사용하여 Berkovits논문에서 제시되었던 위임방식을 적용시켰으며 에이전트의 이주가 발생함에 따라 생성되는 위임토큰들을 내포시키는 방식을 Varadharajan et al.이 분산시스템 이론으로 만든 내포된 토큰 방식을 이동 에이전트 환경으로 가져와서 적용시켜 사용하였고 challenge-response 기반 인증 프로토콜을 추가하여 플레이스들간의 연속 위임을 안전하게 수행할 수 있도록 하였다. 또한 본 제안기법은 메시지 재연 공격과 위임토큰 치환 공격으로부터 안전함을 증명하였다. 향후 연구 과제는 본 논문에서 제안한 TPL을 갱신하는 것에 있어서 신뢰할 수 있는 플레이스의 목록을 추가 또는 삭제하는 기준을 마련하는 것이다.

참 고 문 헌

[1] C. Harrison, D. Chess and A. Kershenbaum, "Mobile Agents : Are they a good idea?," Research Report 1987, IBM Research Division, 1994.

[2] S. Berkovits, J. Guttman and V. Swarup, "Authentication for mobile agents," Lecture Notes in Computer Science #1419 : Mobile Agents and Security, Springer-Verlag, pp. 114-136, 1998.

[3] W. Farmer, J. Guttman and V. Swarup, "Security for mobile agents : issues and requirements, Computer Communications : Special Issue on Advances in Research and Application of Network Security, 1996.

[4] W. Jansen, "Countermeasures for mobile agent security," Computer Communications : Special Issue on Advances in Research and Application of Network Security, 2000.

[5] A. Corradi, R. Montanari and C. Stefanelli, "Mobile agents protection in the internet environment," Proc. 23th Annual International Computer Software and Applications Conference, pp.80-85, 1999.

[6] U. Wilhelm, S. Stamann and L. Buttyan, "A pessimistic approach to trust in mobile agent platforms," IEEE Internet Computing, Vol.4, No.5, pp.40-48, 2000.

[7] Y. Ding and H. Petersen, "A new approach for delegation using hierarchical delegation tokens," Proc. 2nd Conference on Computer and Communications Security, pp.128-143, 1996.

[8] G. Vogt, "Delegation of tasks and rights," Proc. 12th Annual IFIP/IEEE International Workshop on Distributed systems : Operations & Management, pp.327-337, 2001.

[9] M. Abadi, M. Burrows, B. Lampson and G. Plotkin, "A calculus for access control in distributed systems," ACM Transactions on Programming Language and Systems, Vol.15, No.4, pp.706-734, 1993.

[10] B. Lampson, M. Abadi, M. Burrows and E. Wobber, "Authentication in distributed systems : theory and practice," Proc. 13th ACM Symposium on Operating Systems Principles, pp.165-182, 1991.

이 현 석

e-mail : ciga2000@ece.skku.ac.kr

2002년 성균관대학교 전기전자 및 컴퓨터공학부 학사

2002년~현재 성균관대학교 대학원 정보통신공학부 석사과정

관심분야 : 모바일 에이전트, 이동 컴퓨팅 시스템 등

엄 영 익

e-mail : yieom@ece.skku.ac.kr

1983년 서울대학교 계산통계학과 학사

1985년 서울대학교 대학원 전산과학과 석사

1991년 서울대학교 대학원 전산과학과 박사

2000년~2001년 Dept. of Info. and Comm. Science at UCI 방문교수

현재 성균관대학교 정보통신공학부 교수

관심분야 : 분산 시스템, 이동 컴퓨팅, 이동 에이전트, 시스템 소프트웨어, 시스템 보안 등