이동 에이전트의 데이타 보호를 위한 일회용 에이전트 키 생성 시스템

(One-Time Key Generation System for Agent Data Protection in Mobile Agent Systems)

박 종 열 † 이 동 익 †† 이 형 효 ††† 박 중 길 ††††

(Jong-Youl Park) (Dong-Ik Lee) (Hyung-Hyo Lee) (Joong-Gil Park)

요 약 본 논문은 이동 에이전트 시스템에서 발생할 수 있는 보안 문제, 특히 악의를 가진 에이전트 서버로부터 에이전트의 데이타를 보호하기 위한 일회용 에이전트 키 생성시스템을 제안한다. 제안된 일회용 에이전트 키 시스템은 일방향(one-way) 해쉬함수와 연결고리(coupler) 개념을 병용한다. 먼저 일방향 함수는 에이전트 데이타의 비밀성과 무결성을 보장하기 위해서 중요한 역할을 하며, 연결고리는 에이전트의 데이타를 보호하기 위해서 연속되는 암호화키들 사이의 일정한 연관관계(key chain)를 설정하기 위해서 사용된다. 즉 모든 에이전트 키들은 한 방향의 연결고리를 형성하게 된다. 위와 같이 일회용 에이전트 키 생성시스템의 두 가지 특징은 에이전트 소유자(처음 에이전트를 생성한 사용자)만이 에이전트가 순회하면서 수집한 모든 데이타를 복호화 할 수 있도록 하며 악의를 가진 다른 사용자로부터 에이전트 데이타를 보호할 수 있다.

Abstract This paper deals with security issues in a mobile agent system, especially protecting agent data from malicious agent servers. For this purpose, one-time key generation system, OKGS in short, is proposed. In OKGS, we integrate notions of a one-way hash function and a coupler. One-way function plays a major role in ensuring confidentiality and integrity of agent data. And the notion of a coupler is used to establish inter-relationship among consecutive encryption keys for agent data, i.e. all agent keys form a unidirectional chain. With these two features of OKGS, therefore, only the agent owner, who creates the agent bearing data, can decrypt and protect all the agent data which are gathered in the itinerary.

1. 서 론

이동 에이전트 기술은 자율성이나 이동성 등의 특징 때문에, 기존 메시지 기반의 시스템이나 RPC (Remote Procedure Call)에서 극복하지 못했던 단점[1]들을 쉽게 극복할 수 있다. 하지만 이동 에이전트 기술의 이동성은 새로운 보안 문제를 야기 시켰다. 이 문제는 크게 "악의를 가진 이동 에이전트로부터 에이전트 서버를 보호하는 문제"와 "악의를 가진 서버로부터 이동 에이전트를 보호하는 문제"로 분류된다. 본 논문에서는 악의를 가진 에이전트 서버의 에이전트 공격, 특히 에이전트의 데이타를 보호하는데 중점을 두고 있다.

1.1 이동 에이전트 시스템

이동 에이전트란 그림 1-b에서 보는 바와 같이 이질적인 네트워크 상에서 스스로 하나의 서버로부터 다른 서버로 이동하는 프로그램이다. 반면 그림 1-a는 전통적인 클라이언트-서버 구조를 보이고 있다. 기존의 클라이언트-서버 구조는 단순하고 이미 설치되어 있는 정적인 서비스들을 제공하는 반면 이동 에이전트 모델은 그

· 이 논문은 정보통신부 대학정보통신 연구센터 지원사업의 지원 및 한국소프트웨어진흥원의 관리로 수행되었음.

† 비 회 원 : 광주과학기술원 정보통신공학과
jypark@kjist.ac.kr
†† 종신회원 광주과학기술원 정보통신공학과 교수
dilee@kjist.ac.kr
††† 비 회 원 : 원광대학교 정보,전자상거래학부 교수
hlee@wonkwang.ac.kr
†††† 비 회 원 : 충남대학교 컴퓨터과학과
jgpark@etri.re.kr

논문접수 : 2000년 12월 1일
심사완료 2001년 5월 17일

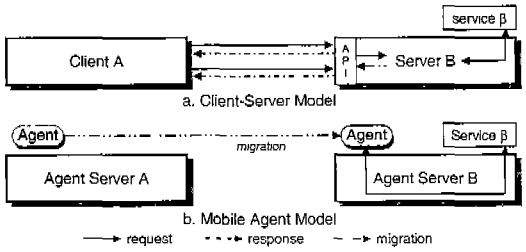

그림 1 이동 에이전트 개념

본래의 특징인 이동성과 자율성에 의해서 능동적이고 유연성 있는 서비스를 제공한다. 이동 에이전트 기술은

▶ 네트워크 트래픽을 줄이며,

▶ 작업의 병행성을 증가시키고,

▶ 보다 유연성 있는 서비스를 가능

하게 하는 등 그 자체가 근본적으로 많은 장점을 가지고 있다. 특히 JDK 1.1의 개발 이후 RMI (Remote Method Invocation)와 코드 직렬화(Serialization)를 이용한 에이전트 개발 연구가 매우 활발해 졌다. 표 1은 지금까지 개발된 이동에이전트 시스템을 보이고 있으며, 이들 중 많은 시스템들이 Java 언어를 기반으로 하고 있는 것이 그러한 이유이다.

표 1 이동 에이전트 시스템

이 름	언어	개발기관
AgentTcl	Tcl	Dartmouth College, USA
Ara	Java	University of Kaiserslautern, Germany
Agelets	Java	IBM, Japan
Concrdia	Java	Mitsubishi, USA
GrassHopper	Java	IVK++, Germany
JATLite	Java	Stanford University, USA
Mole	Java	University of Stuttgart, Germany
Odyssey	Java	General Magic, USA
Voyager	Java	ObjectSpace, Inc., USA
X-MAS	Java	K-JIST, Korea

반면 이동 에이전트 기술은 몇 가지 보안상의 심각한 문제들을 발생시켰고 이러한 문제들은 현재 이동 에이전트 기술을 실제로 사용하는데 가장 큰 장애요인으로 인식되고 있다. 이동 에이전트 시스템의 보안 문제는 다음과 같다.

- **악의를 가진 에이전트 서버로부터 이동 에이전트의 보호**: 이동 에이전트의 비밀성과 무결성은 에이전트 시스템에서 중요한 문제이다. 실제 에이전트 서버는 이동 에이전트의 코드와 데이타를 쉽게 수정하거나 삭제 또는 복사가 가능하다. 이것은 이동 에이전트가 에이전트 서버의 도움 없이 스스로 이동하거나 수행할 수 없는 특징을 가지고 있기 때문이다. "에이전트 수행 권한이 없는 서버가 에이전트를 수행하는 공격"과 "에이전트 데이타의 불법적인 수정"이 전형적인 예이다.
- **악의를 가진 이동 에이전트로부터 에이전트 서버의 보호**: 이동 에이전트는 에이전트 서버 위에서 수행되기 때문에, 악의를 가진 에이전트는 인터넷 웜 또는 컴퓨터 바이러스처럼 동작할 수 있다. 임의의 에이전트 A가 서버 B로 이동한 경우를 가정해 보자, A는 B의 CPU와 메모리를 이용해서 자신에게 주어진 작업을 수행한다. 구체적으로, A가 B의 자원에 대해서 무제한의 권한(파일 시스템, 메모리, CPU)을 인가한다면, A는 B의 자원을 고갈시켜서 시스템을 다운시킬 수 있을 것이다. 결국 에이전트 서버는 하나의 코드를 수행할 때마다 에이전트의 코드 상태를 평가해야만 한다[2, 3]. 컴퓨터 바이러스와 이동 코드의 가장 큰 차이점이 코드가 악의성을 가지고 있는가 하는 점이기 때문에 에이전트 서버는 각 에이전트 코드가 필요한 평가 과정(접근 권한 획득과 인증을 위해서)을 수행하게 된다.

1.2 관련연구

악의를 가진 이동 에이전트로부터 에이전트 서버를 보호하는 것은 분산 시스템에서의 보안문제인 인증(authentication), 권한부여(authorization) 와 상당히 유사하다. 그래서 지금까지 제안된 많은 보안 모델들이 인증과 접근제어를 주로 사용하고 있다. 최근 접근 제어를 이용한 연구가 많이 진행되어서 Aglet 시스템의 권한부여언어(authorization language)[4], Ara 시스템의 여권모델(passport model)[5], Java의 Sandbox(그림 2) 보안 모델들이 구체적으로 연구되었다. 이들 중 권한부여언어가 가장 진보한 형태인데, 권한부여언어는 생성자, 소유자, 주인, 승인권한의 기본 기능을 이용해서 기본적인 보안정책을 정의하고, 정의된 승인들의 조합으로 합성승인을 생성하여 완전한 보안정책을 수립한다. 이러한 방법은 하나의 접근 권한에 대해 여러 가지 승인정보를 부여하여 구조적인 관리 구조를 지원한다. 반면, 이동 에이전트를 보호하는 문제는 아직까지 뚜렷한 해결책이 제시되지 않고 있다. 단지 썬(Sun Microsystems)에서

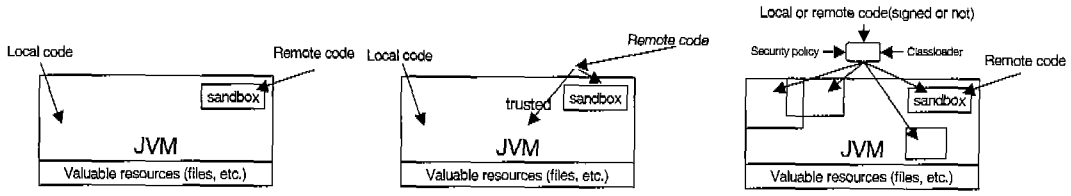

그림 2 자바 Sandbox 보안 모델

개발한 "Safe Tcl Security Model"[6, 7], UC 버클리에서 제안한 "Toward Mobile Cryptography"[8] 그리고 "Mole"에이전트 시스템에서 제안된 "Time Limited Blackbox Security"[9, 10] 정도가 연구 진행중이며 아직 기초적인 연구 단계에 있다. 이들 연구의 특징을 간략히 설명하면 다음과 같다.

- ***Safe Tcl Security Model***: 에이전트 Tcl은 Tcl 스크립트 언어와 Tcl 확장을 기반으로 하고 있으며, Safe-Tcl은 코드의 수행 중에 안전성 검사 기능을 추가로 제공한다[8]. 이 방법은 언어차원에서 보안 기능을 제공하므로 강력한 보안 기능을 제공한다. 단점은 보안 기능 강화를 위해서 Tcl언어 자체가 확장되었지만 다른 Tcl기반의 에이전트 시스템과 호환되지 못한다.
- ***Toward Mobile Cryptography***: "서버는 암호화된 함수를 복호화 하지 않고 수행할 수도 있다"는 생각에 기반한 이 방법은 준동형(homomorphic)의 암호화 방법에 기반하고 있다. 이동 에이전트의 코드는 서버로부터 비밀성을 보장받기 힘들기 때문에, 암호화된 함수로 만들어서 암호화된 상태로 에이전트 서버에서 수행하도록 하여, 에이전트를 보호하려는 방법이다. 제안된 방법이 구현된다면 훌륭한 보안 모델이 되겠지만, 암호화된 이동 함수(Mobile Encrypted Function)를 생성하는 일이 매우 어렵고, 구현된 함수들도 제한적인 기능만을 수행하는 단점을 가지고 있다.
- ***Time limited Blackbox Security***: 이동 에이전트는 에이전트를 생성한 서버를 제외한 모든 에이전트 서버에서 에이전트 코드가 생성자의 블랙박스(Black-box)를 통해 헝클어진(mess-up) 형태로 수행하도록 하여, 일정 시간동안 원래 코드를 생성할 수 없도록 하는 방법이다. 이 모델의 보안 성능은 에이전트를 헝클어뜨리는 알고리즘에 따라 달라지며, 헝클어진 코드를 읽을 수 있는 코드로 변환하는 과정에 시간이 필요하다는 점에 착안하고 있다. 이 방법은 무작위 공격에 매우 약하며, 컴퓨터의 성능 발달에 따라 제한 시간이 줄어드는 문제점을 가지고 있다.

1.3 논문의 목적 및 구성

본 논문에서는 이동 에이전트의 안전한 실행환경을 구축하기 위해 이동 에이전트 시스템의 보안 모델을 제안한다. 지금까지 제안된 에이전트 보안 모델들은 에이전트의 정확한 수행에 중점을 두고 있지만, 본 논문은 에이전트의 수행보다는 수행후의 결과를 보호하는 방법을 제시하고 있다. 에이전트의 특성상 에이전트 코드는 모든 서버에서 수행이 되어야 하기 때문에, 서버로부터 코드의 비밀성을 보장받을 수 없다. 이러한 특징은 기존의 시스템들이 해결하기 어려운 한계이며 이를 해결하기 위해서 우리는 각 서버들이 가지는 고유의 키를 부여하였고, 이들 사이의 일정한 연관관계(key chain)를 만들었다. 특히 일방향 해쉬함수는 각 서버에서 생성된 고유 키들의 일정한 연관관계를 부여한다. 이 연관관계는 각 서버에서 사용되는 키들 사이의 일정한 연결고리(coupler)를 제공하여 키들의 무결성을 제공한다. 이와 같이 일정한 연관관계를 가지도록 설계된 키들을 "일회용 에이전트 키"라고 하며, 에이전트 데이타를 직접 암호화하며 에이전트 데이타의 비밀성과 무결성을 보장한다. 논문의 구성은 다음과 같다: 2장은 전자서명, "일회용 에이전트 키"로 이루어진 전체 보안 모델에 대해서 설명하고, 3장은 "일회용 에이전트 키"의 알고리즘과 각 모듈의 기능적인 특징을 자세히 설명한다. 4장에서는 제안된 알고리즘의 안전성을 분석하고 5장에서 결론을 맺는다.

2. 일회용 에이전트 키 생성 시스템

앞에서 언급한 바와 같이 이동 에이전트 시스템에서 보안 모델은 이동 에이전트와 에이전트 서버로 구성된다. 에이전트 서버를 위한 보안 모델은 접근제어를 통해서 제공되며, 초기 시스템 설계 단계에 고려해야하는 사항으로 본 논문에서는 언급하지 않고 이동 에이전트 보호를 위해 제안된 "일회용 에이전트 키"와 전자서명만

을 언급한다.

2.1 일회용 에이전트 키

키 이동 에이전트가 여러 서버에서 수집한 데이타의 비밀성과 무결성에 관한 위험요소를 고려하기 위해 다음과 같은 항공권 예약 시스템을 가정한다. A항공사와 B항공사는 서로 경쟁 상대이고, 사용자는 에이전트에게 각 항공사를 방문하여 항공권 가격과 좌석현황 그리고 경유지에 대한 정보를 수집하게 한다. 기존의 공개키 기반 시스템에서는 에이전트가 각 서버를 방문해서 필요한 정보를 홈 서버의 공개키로 암호화하여 가져오게 된다. A항공사를 먼저 방문하고 B항공사를 방문한다면, B항공사는 A항공사의 정보를 모두 삭제할 수 있다. B항공사는 비록 A항공사의 정보를 알지 못하지만 A항공사의 정보를 삭제함으로써 마치 A항공사의 서버가 동작하지 않는 것처럼 보이게 할 수 있다. 또한 홈 서버의 공개키는 누구나 알고 있는 정보이기 때문에 B항공사는 A항공사의 정보를 거짓으로 쉽게 만들 수 있다. 그러므로 에이전트 데이타의 비밀성과 무결성을 보장하기 위해서 B항공사는 A항공사가 제공한 정보를 읽을 수 없어야 하며, 에이전트는 B항공사가 A항공사의 정보를 삭제할 수 없도록 데이타를 보호하는 기능을 수행해야 한다. 이러한 요구에 대해서 B항공사와 A항공사의 데이타 간에 어떤 연관성을 갖게 하는 것이 하나의 해결책이 될 수 있다. 본 논문에서는 에이전트 서버들이 생성한 데이타들을 각 서버만이 알 수 있는 키로 암호화하고, 암호키 생성에 필요한 정보는 에이전트 서버가 부인하거나 공격자들이 변조하지 못하도록 처리한다. 그리고, 각 에이전트 서버가 사용하는 키들간에 연관관계를 부여함으로써 홈 에이전트 서버만이 모든 암호키를 복원할 수 있도록 한다.

그림 3은 일회용 에이전트 키 생성 시스템의 구조를 나타내고 있다. 그림 3의 각 파라메터들에 관한 상세한 설명은 III장으로 미루고 개략에 대해서만 설명하면 다음과 같다. 그림 중간의 육면체가 "일회용 에이전트 키"의 키 생성 모듈이며 키 생성 모듈은 일방향 해쉬함수에 기반을 두고 있다. 에이전트 서버는 직전 서버로부터 에이전트를 받아서 먼저 코드에 대한 서명을 검사하고, 에이전트와 함께 받은 키 생성정보를 이용하여 새로운 키(new key)를 생성한다. 이 키 생성정보가 이전 서버로부터 전달된 연결고리(그림 3 왼쪽의 coupler C_{k-1})이다. 새롭게 생성된 키를 이용하여 이동 에이전트가 수행한 결과를 암호화하고 에이전트는 암호화된 데이타를 저장한다. 결국 에이전트 서버는 자신이 제공한 데이타를 자신이 암호화하고 다음 서버가 사용할 연결고리를 일방향 해쉬함수를 이용해서 생성한다. 이 연결고리는 에이전트가 다음 서버로 이동하여 새로운 에이전트 키를 생성하는데 사용한다. 연결고리의 개념은 연속적인 키들 사이의 상호연관관계를 가지는 것을 가능하게 한다.

그림 3에서 에이전트의 구조를 자세히 보면, 에이전트가 이동하면서 새롭게 추가되는 데이타가 있다. 연결고리 값 C_k는 항상 바뀌는 값이지만, $DES_k(data_k)$와 $AS_{0-pub}(R1_k, R2_k, R3_k, DSA_k(X, TimeStamp), TimeStamp)$가 추가된다. 결국 에이전트는 현재 Coupler인 C_k와 DES로 암호화한 암호문 $\sum_{i=0}^{k} DES_i(data_i)$, 그리고 홈 에이전트 서버($AS_0$)의 공개키($AS_{0-pub}$)로 암호화한 $\sum_{i=0}^{k} AS_{0-pub}(R1_i, R2_i, R3_i, DSA_i(X, TimeStamp), TimeStamp)$를 가지고 이동하게 된다.

에이전트의 작업공간에 암호화된 데이타들은 "일회용

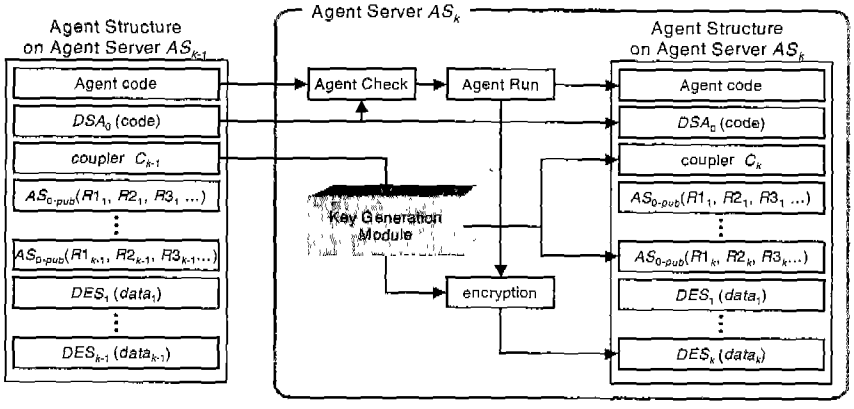

그림 3 일회용 에이전트 키 생성 시스템의 동작 구조도

에이전트 키"들을 이용해서 모두 복호화 할 수 있다. 이동 에이전트가 홈 에이전트 서버로 돌아 왔을 때, 각 서버의 에이전트 키는 일방향 해쉬함수와 각 서버의 비밀정보($R1$, $R2$, $R3$)로 생성되었기 때문에 홈 서버에서 재생 가능하다. 그래서 홈 서버는 자신이 생성한 초기 값을 이용하여 각 서버에서 사용된 키를 생성하고 각 데이타들을 복호화하게 된다.

2.2 전자서명과 타임스탬프

일회용 에이전트 키 생성 시스템에서 전자서명과 타임스탬프는 에이전트 코드나 데이타의 무결성을 보장하기 위해 사용한다. 이 시스템에서 전자 서명은 두 가지 목적으로 사용된다. 하나는 에이전트 코드의 무결성을 제공하기 위한 에이전트 코드의 서명이고, 다른 하나는 에이전트 데이타의 무결성에이전트 코드의 서명은 에이전트를 생성한 홈 에이전트 서버가 에이전트 코드를 서명한 것으로 그림 3의 $DSA_0(code)$가 여기에 해당되며, 각각의 에이전트 서버는 에이전트 코드가 변경되었는가를 서명을 검사하여 확인하게 된다. 또한 에이전트 데이타의 서명은 에이전트 키를 생성하기 위해 각 서버가 생성하는 난수 $R1$, $R2$, $R3$ 값이 자신이 생성한 것임을 증명하기 위해서 사용된다. 단 여기서 서명 내용이 $R1$, $R2$, $R3$가 아닌 암호문과 타임스탬프를 이용하게 되는데, 이는 현재 서버가 작업한 내용을 적절히 반영하기 위함이며, 홈 서버의 공개키로 암호화하기 때문에 다른 서버에 의해서 재사용 되지는 않는다.

3. 일회용 에이전트 키 생성 알고리즘 및 구현

일방향 해쉬함수란 충돌이 없는(collision free) 해쉬함수, 안전한(secure) 해쉬함수 등으로 부르기도 하는데 주어진 메시지 x에 대해서 $y=f(x)$를 구하기는 쉽지만 반대로 $f^{-1}(y)=x$인 f^{-1}함수 혹은 x값을 찾는 것이 어려운 특징을 가지는 함수를 말한다. 이러한 해쉬함수의 비대칭적인 연산 특징 때문에 많은 보안 알고리듬과 시스템에서 사용되고 있다. 일방향 해쉬함수를 정의하는 일방향성에 대한 정의는 다음과 같다.

정의 1. 주어진 메시지 다이저스트 z에 대해서 $h(x)=z$인 메시지 x를 찾는 것이 계산론적으로 불가능한(computationally infeasible) 해쉬함수 h는 일방향성(one-way property)을 가진다.

메시지 x에 대해서 일방향 해쉬함수를 n번 수행해서 얻는 값을 $h^n(x)$라 가정할 때, 다음 a단계의 계산 값인 $h^{n+a}(x)$는 구할 수 있다. 하지만 초기 메시지인 x나 $0<k<n$인 $y=h^k(x)$ 값은 정의 1의 일방향성에 의해서 구할 수 없다. 암호화에 사용되는 "일회용 에이전트 키"는 이러한 일방향 해쉬함수에 기반 하여 생성되고 연속되는 두 개의 키들 사이에는 상호연관관계를 가진다. 그림 4는 상호연관관계가 어떻게 생성되는지를 간략히 보여주고 있다. 난수 $R1_k$, $R2_k$, $R3_k$는 현재의 에이전트 서버 AS_k에 의해서 생성된 값이며, 홈 에이전트 서버 AS_0의 공개키 $AS_{0\text{-}pub}$로 암호화된다. 즉, 하나의 에이전트 키는 $R1_k$, $R2_k$ 뿐만 아니라 연결고리인 C_{k-1}를 이용해서 생성한다. 따라서 오직 홈 에이전트 서버인 AS_0만이 각 서버의 $R1_k$, $R2_k$, $R3_k$, C_{k-1}를 재생하고, 모든 에이전트 서버의 에이전트 키를 복원할 수 있다.

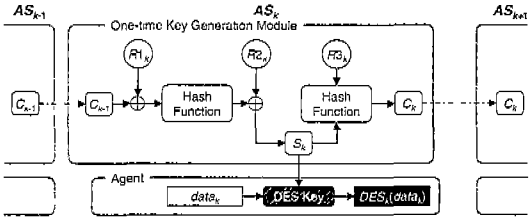

그림 4 일방향 해쉬함수를 이용한 키 생성 모듈

본 논문에서 일회용 에이전트 키 생성 시스템의 구축을 위해서 몇 가지 가정을 하고 있다. 우선 기본 통신 구조는 *PKI(Public Key Infrastructure)*를 가정하고, 일방향 해쉬함수로 160-비트 길이의 출력을 가지는 $SHA-1$[1]을 사용했다. 전자서명 알고리즘은 *DSA*(Digital Signature Algorithm)를 사용하고 암호화 알고리즘은 구현상의 편의를 위해서 *DES*를 사용했으며, *AES* 혹은 *SEED*로 대치가 가능하다. 여기서 해쉬함수 $SHA-1$이 안전하지 않다고 판단되면, 이 논문에서 제안한 알고리즘은 사용된 해쉬함수에 의존하지 않으므로 $SHA-1$은 *RIPEMD-160*이나 다른 해쉬함수로 대치시킬 수 있다.

3.1 알고리즘

자세한 알고리즘에 앞서 사용된 수식을 정리하면 다음과 같다.

K	: 에이전트가 방문한 서버의 일련번호
$DSA_0(code)$	: 홈 에이전트 서버 AS_0의 코드에 대한 서명
$SHA\ 1(m)$	: 메시지 m의 해쉬결과
AS_k	: k번째 방문한 에이전트 서버
$data_k$	: AS_k에서 에이전트가 수집한 데이타
$DES_k(data_k)$	: $data_k$를 DES키로 암호화한 암호문
$AS_{0\text{-}pub}(m)$	: 메시지 m을 홈 에이전트 서버 AS_0의 공개키로 암호화한 암호문

1) 본 논문에서 $SHA-1$은 일방향 해쉬함수로 가정한다.

RANDOM NUMBER: $R1_k$(80-bit), $R2_k$(160-bit), $R3_k$(80-bit)

RUN: 코드 서명 검사 - $DSA_0(code)$

RUN: S_k SHA-1(($R1_k$. $R1_k$) ⊕ C_{k-1}) ⊕ $R2_k$

RUN: $data_k$ = AS_k 에 의해 생성된 에이전트 데이터

- **DES$_k$** : $H(S_k) \oplus L(S_k) \oplus M(S_k)$

  $H(S_k) = S_{k1}:S_{k2}$, $L(S_k)$ = $S_{k4}:S_{k5}$, $M(S_k) = S_{k3}$이며, 각각의 S_{ki} $(i=1,2,\ldots,5)$ [illegible] 값을 S_k의 5 등분 해($S_k = S_{k1}:S_{k2}:S_{k3}:S_{k4}:S_{k5}$) 값이다.
- **Encryption:** $DES_k(data_k)$
- **X** = $DES_1(data_1) + DES_2(data_2) + \ldots DES_k(data_k)$
- **Encryption:** $AS_{0\,pub}(R1_k, R2_k, R3_k, DSA_k(\lambda, Timestamp), Timestamp)$

Replace: $C_k = SHA\text{-}1(S_k \oplus R3_k)$

Append: $DES_k(data_k)$

Append: $AS_{0\,pub}(R1_k, R2_k, R3_k, DSA_k(X, Timestamp), Timestamp)$

알고리즘 1 일회용 에이전트 키 생성 알고리즘

알고리즘 1은 일회용 에이전트 키 생성 알고리즘의 전체 과정을 보여주고 있다. 임의의 에이전트 서버 AS_k가 이전 에이전트 서버인 AS_{k-1}으로부터 연결고리 C_{k-1}을 받고, 자신의 비밀정보인 난수 $R1_k$, $R2_k$, $R3_k$를 생성한다. (만약 AS_{k-1}이 홈 서버라면(k=1), AS_0는 에이전트와 초기 연결고리 C_0를 생성하고 AS_1에게 보낸다). AS_k은 AS_{k-1}로부터 전달된 C_{k-1}와 자신이 생성한 $R1_k$, $R2_k$ 값을 $SHA-1$ 함수를 이용해서 키 생성 정보인 S_k을 생성한다. 그리고 S_k 값을 이용해서 AS_k의 64-비트 DES 키를 생성한 다음 AS_k은 다음 에이전트 서버 AS_{K+1}이 사용할 연결고리 C_k는 키 생성정보 S_k와 난수 $R3_k$를 해쉬함수를 통해 생성하여 에이전트 코드와 함께 전송하게 된다.

다음은 알고리즘 1에 기술된 일회용 에이전트 키 생성 알고리즘의 세부 동작을 단계별로 설명한다.

RANDOM NUMBER : $R1_k$(80-bit), $R2_k$(160-bit), $R3_k$(80-bit)

"일회용 에이전트 키"를 생성하기 위해서 3개의 난수 $R1_k$, $R2_k$, $R3_k$를 생성한다. 이 값들은 에이전트 수행 후 홈 에이전트 서버 AS_0의 공개키 AS_{0-pub}로 암호화하기 때문에 홈 에이전트 서버 AS_0만이 이 값을 복호화 하여 읽을 수 있다.

RUN : 코드 서명 검사 - $DSA_0(code)$

처음 에이전트를 수신한 후, 에이전트 서버는 에이전트 코드의 무결성을 검사하기 위해서 코드의 전자서명을 검사한다. 이 논문에서 PKI를 가정하고 있기 때문에, 전자 서명은 별도의 키를 사용하지 않고 공개키 기반의 DSA 알고리즘을 사용한다.

RUN: S_k=$SHA-1((R1_k : R1_k) \oplus C_{k-1}) \oplus R2_k$

S_k는 에이전트 서버가 사용할 DES 키 생성에 관한 정보로 에이전트가 저장하고 있다. 연산과정은 먼저 C_{k-1}과 $R1_k : R1_k$를 XOR 연산한다. 여기서 $R1_k : R1_k$는 80-비트의 난수 $R1_k$를 2개 연접(concatenation)한 것이며, C_{k-1}은 이전 에이전트 서버 AS_{k-1}으로부터 전송된 160-비트의 연결고리(coupler)이다. 만약 S_k를 생성하기 위해서 자신이 생성한 난수들을 이용하는 $SHA-1(R1_k : R1_k) \oplus C_{k-1}) \oplus R2_k$대신 연결고리 C_{k-1}만을 이용하는 $SHA-1(C_{k-1})$을 사용한다면, C_{k-1}을 알고 있는 이전 에이전트 서버가 공개 알고리즘인 $SHA-1$을 이용하여 S_k를 계산할 수 있으므로 에이전트 데이타의 무결성이 침해될 수 있다.

위의 알고리즘에서 S_k값은 이전 에이전트 서버 AS_{k-1}이 전송한 C_{k-1}를 이용해서 생성된다는 점에서 연결고리를 이용하여 "일회용 에이전트 키"들 사이에 상호연관관계를 설정할 수 있음을 알 수 있다.

RUN : $data_k$ = AS_k에 의해 생성된 에이전트 데이타 (단, AS_k입장에서 $data_1 \sim data_{k-1}$는 AS_k 이전에 에이전트가 방문했던 서버들의 데이타)

$data_k$는 에이전트 서버 AS_k에서의 수행 결과이며, 아래의 과정으로 생성된 DES 키 DES_k로 암호화된다.

DES$_k$ = $H(S_k) \oplus L(S_k) \oplus M(S_k)$

$H(S_k)=S_{k1}:S_{k2}$ $L(S_k)=S_{k4}:S_{k5}$, $M(S_k)=S_{k3}$: S_{k3}이며, 각각의 S_{ki}(i=1, 2, ..., 5)는 32-비트 값으로 S_k를 5 등분한 ($S_k=S_{k1}:S_{k2}:S_{k3}:S_{k4}:S_{k5}$)

값이다.

Encryption: $DES_k(data_k)$

여기서, 에이전트 데이타는 항상 새로운 키로 암호화되기 때문에 키 생성과 암호화 속도가 빠른 대칭키 암호 알고리즘인 *DES*를 사용했다. 키 생성을 위해 사용되는 S_k는 160-비트이고 *DES* 키는 64-비트이기 때문에 그림 5에서처럼 상위 64-비트와 하위 64-비트 그리고 중간의 32-비트를 XOR연산하여 새로운 *DES* 키를 생성한다. 여기서 생성된 키가 이미 알려진 취약한 64개의 키 중 하나라면 난수인 $R1_k$, $R2_k$부터 다시 생성해야 한다. 결국 에이전트의 수행결과는 새로운 *DES* 키로 암호화 $DES_k(data_k)$되고, 에이전트가 이동하면서 수집한 암호화된 데이타들을 X로 표기하고, 이를 서명 후 암호화한다.

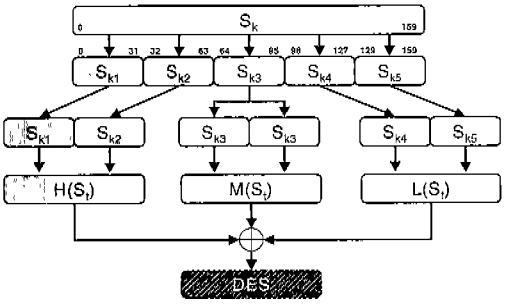

그림 5 DES 키 생성

$$X = DES_1(data_1)+DES_2(data_2)+...DES_k(data_k)$$

Encryption: $AS_{0\text{-}pub}(R1_k, R2_k, R3_k, DSA_k(X, TimeStamp), TimeStamp)$

수집한 데이타들의 무결성을 보장하기 위해서 수집된 전체 데이타를 전자서명 $DSA_k(X, TimeStamp)$하고 이 정보와 시간정보 *TimeStamp*, 난수$R1_k$, $R2_k$, $R3_k$를 홈 서버의 공개키인 $AS_{0\text{-}pub}$로 암호화하게 된다. 만약 악의를 가진 서버에 의해서 위 내용 중 일부 혹은 전부가 삭제되거나 수정되어도 해당 서버의 키를 알지 못하는 한 일회용 키를 복원하는 과정에서$DES_k(data_k)$의 변경 여부를 알 수 있다. $R1_k$, $R2_k$, $R3_k$, $DSA_k(X, TimeStamp)$ 값이 재사용되는 경우는 시간정보 *TimeStamp*와 *X*의 서명을 검사하여 확인할 수 있다.

Replace: $C_k=SHA-1(S_k \oplus R3_k)$

Append: $DES_k(data_k)$

Append: $AS_{0\ pub}(R1_k, R2_k, R3_k, DSA_k(X, TimeStamp), TimeStamp)$

다음 에이전트 서버인 AS_{k+1}이 S_k를 유추하지 못하게 하기 위해 S_k와 $R3_k$를 XOR연산을 수행한 후 해쉬 연산을 수행하여 C_k값을 생성한다. 프로그램 상에서 C_k값은 기존에 사용하던 C_{k-1}를 덮어쓴다. 이동 에이전트 입장에서 이 값을 변경하지 않지만, 서버를 이동할 때마다 각 서버들이 적절한 값으로 변경하기 때문에 키 생성에 매번 사용하여도 그 값이 항상 다르게 된다. C_k값을 변경한 후에 암호화된 데이타인 $DES_k(data_k)$와 $AS_{0\text{-}pub}(R1_k, R2_k, R3_k, DSA_k(X, TimeStamp), TimeStamp)$를 서버 AS_k가 저장하고 에이전트를 다음 에이전트 서버 AS_{k+1}에게 전송한다.

일회용 에이전트 키 생성 시스템에서 에이전트 데이타를 보호하기 위해서 필요한 추가적인 데이타의 크기는 데이타 크기에 무관하게 서버마다 항상 600-비트[2]이다. 이는 암호화된 $DES_k(data_k)$의 크기가 $data_k$의 크기와 같기 때문에 에이전트가 추가적으로 가져야하는 데이타는 $AS_{0\text{-}pub}(R1_k, R2_k, R3_k, DSA_k(X, TimeStamp), TimeStamp)$가 되기 때문이다. 또한 본 시스템에서 사용된 알고리즘인 *DES*, *SHA*-1는 수행속도가 빠르기 때문에 전체 에이전트 시스템의 성능은 크게 떨어지지 않는다.

홈 에이전트 서버 AS_0는 자신이 생성한 에이전트가 임무를 완성하고 돌아 왔을 경우, 자신의 비밀키 $AS_{0\text{-}pri}$를 이용하여 $AS_{0\text{-}pub}$로 암호화된 내용을 복호화하여 각 에이전트 서버 $AS_k(k=1, ..., n)$의 비밀정보인 난수 $R1_k$, $R2_k$, $R3_k$값을 얻게 된다. 따라서 홈 에이전트 서버 AS_0는 자신이 생성한 C_0와 난수들 $R1_k$, $R2_k$, $R3_k$를 이용해서 아래와 같이 순차적으로 각 서버의 C_k와 S_k를 생성한다.

$$S_1=SHA-1(R1_1:R1_1 \oplus C_0) \oplus R2_1,$$
$$C_1=SHA-1(S_1 \oplus R3_1)$$
$$S_2=SHA-1(R1_2:R1_2 \oplus C_1) \oplus R2_2,$$
$$C_2=SHA-1(S_2 \oplus R3_2)$$
$$\vdots$$
$$S_k=SHA-1(R1_k:R1_k \oplus C_{k-1}) \oplus R2_k,$$
$$C_k=SHA-1(S_k \oplus R3_k)$$
$$\vdots$$
$$S_n=SHA-1(R1_n:R1_n \oplus C_{n-1}) \oplus R2_n$$

결론적으로 홈 에이전트 서버 AS_0는 S_0부터 S_n까지 모든 S_k값을 가지게 되고 $H(S_k) \oplus L(S_k) \oplus M(S_k)$ 연산을 이용하여 모든 *DES*키를 구하게 된다.

2) 600-비트=80-비트($R1$)+160-비트($R2$)+160-비트($R3$)+160-비트(*DSA* 전자서명)+40-비트(타임스탬프)

3.2 구현

이동 에이전트 시스템은 에이전트와 에이전트 서버의 상호 연동으로 동작한다. 따라서 프로그램의 제어가 에이전트와 서버 사이에 상호 교환되는데, 이러한 구조를 지원하기 위해서 많은 시스템들이 에이전트 라이프사이클 모델을 지원한다. 본 논문에서도 born, init, running, stop, dead 의 5 단계의 라이프사이클을 지원하며, 에이전트 서버, 에이전트 템플릿, 사용자 에이전트의 3 객체가 연동한다.

3.2.1 에이전트 라이프사이클과 일회용 에이전트 키

이동 에이전트는 에이전트 서버로부터 최고 5번의 컨트롤을 부여받는다 그림 6은 에이전트의 5단계의 상태를 나타내며, 에이전트의 생성에서 소멸까지의 라이프사이클이라고도 한다. 또한 에이전트는 에이전트 템플릿과 사용자 에이전트로 구성되며, 에이전트 템플릿은 시스템 수준에서의 동작과 라이프사이클에 필요한 기반구조를 지원한다. 반면 사용자 에이전트는 에이전트가 각 서버에서 하는 동작을 기술하도록 되어 있다. 따라서 일회용 에이전트 키의 모든 부분이 에이전트 템플릿에서 구현된다. 사용자 에이전트는 비밀정보를 저장하는 경우 에이전트 템플릿에서 제공하는 Secure_Save()함수를 호출하기만 하면 된다.

다음은 에이전트 라이프사이클의 각 단계에 대해서 설명하고, 단계별로 에이전트 템플릿에서 일회용 에이전트 키를 위해서 해야하는 일들을 서술한다.

- ▶ Born - 홈 에이전트 서버에서 수행되며, 에이전트의 생성과 초기 연결고리 값인 C_0, 코드의 서명인 $DSA_0(code)$를 생성한다.
- ▶ Init - 초기화 단계로 에이전트가 이동한 후 처음 호출된다. 코드의 서명 값을 검사하고, 연결고리 C_{k-1}를 이용하여 일회용 키를 생성한다.
- ▶ Running - 정상적인 에이전트를 수행하며, 비밀정보를 저장하는 경우 에이전트 템플릿의 Secure_Save함수를 호출한다. 함수 Secure_Save는 주어진 데이타를 일회용 에이전트 키로 암호화하는 작업을 수행하며, 암호화된 데이타는 따로 저장된다.
- ▶ Stop - $R3_k$를 이용하여 C_k값을 해쉬한다.

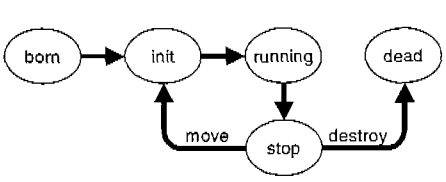

그림 6 에이전트 라이프사이클

- ▶ Dead - 일회용 에이전크키를 모두 복원하고 에이전트가 수집한 데이타를 복호화한다.

3.2.2 구현 결과

그림 7은 본 논문에서 제안한 일회용 에이전트 키 생성 시스템의 구현 결과를 보이고 있다. 실험을 위해서 두 대의 서버 TONGKI(홈 에이전트 서버 AS_0)와 DSL을 사용하였다. 그림에서 왼쪽 사각형이 TONGKI의 동작화면이고 오른쪽 사각형이 DSL의 동작 화면이다. 이동 에이전트는 TONGKI(AS_0, AS_1) → DSL(AS_2) → TONGKI(AS_0) 순으로 순회한다.

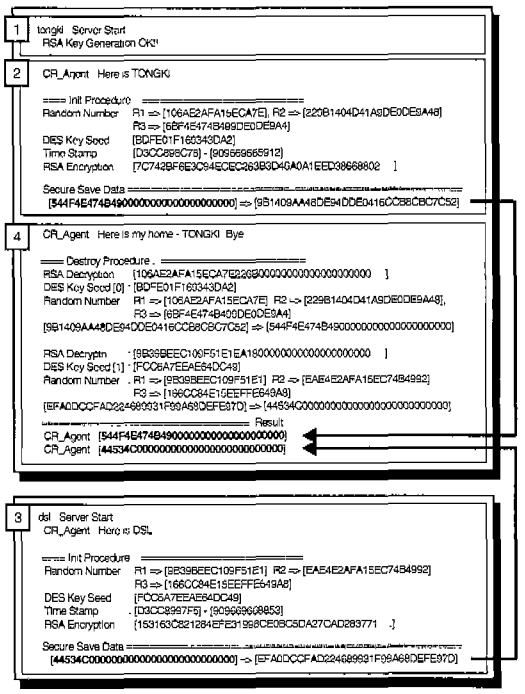

그림 7 일회용 에이전트 키 생성 알고리즘의 구동화면

먼저 AS_0인 TONGKI는 C_0를 생성하고, 아래와 같이 TONGKI와 DSL에서 각각의 작업을 수행한다.

- *TONGKI*(AS_0 : 그림 7의 1)
  - -에이전트 생성
  - -C_0 생성
- *TONGKI*(AS_1 : 그림 7의 2)
  - -난수 생성
  - (R1:106AE..., R2:22..., R3:6B...).
  - -에이전트 키 생성

S_1(BDFE01F169).
-에이전트 키를 이용하여 암호화
(544F4E47B... → 9B1409AA48...).

- *DSL*(AS_2 : 그림 7의 ③)
-*TONGKI*와 마찬가지로 난수들을 생성하고 *DES* 키 생성한 후 에이전트 데이타를 암호화한다.
(4453400000... → EFA0DCCFA...).
- *TONGKI*(AS_0 : 그림 7의 ④)
-$AS_{0\text{-}prt}$를 이용하여 각 서버의 비밀정보인 난수 복호화 및 서명 $DSA_k(X, TimeStamp)$검사
-각 에이전트 서버의 에이전트 키 생성. 그림 7에서 화살표를 따라 확인하면 처음 제공된 데이타가 정상적으로 복호화됨을 볼 수 있다.

4. 분 석

이 논문에서 제안한 "일회용 에이전트 키"는 무결성을 해치는 에이전트 서버로부터 에이전트 데이타의 수정을 방지할 수 없다. 그러나 에이전트 데이타의 비밀성을 완벽하게 제공하고 무결성 침해 또한 감지할 수 있다. 이를 검증하기 위해서 "일회용 에이전트 키"가 위험한 가능성을 열거하고 각각에 대한 안전성을 검토한다. 이를 위해서 먼저 시스템에서 사용되는 중요 정보와 성질을 나열하면 아래와 같다. 단, 여기서 $R1$, $R2$, $R3$,는 홈 에이전트 서버의 공개키인 $AS_{0\text{-}pub}$로 암호화하기 때문에 다른 에이전트 서버는 알 수 없다.

▶ 일방향성
▶ *DES* 키
▶ 에이전트 데이타

다음의 각 절에서는 위의 각각의 정보에 대한 위험요소를 분석하고 안전성을 분석한다.

4.1 일방향성 위협

해쉬함수의 일방향성은 본 시스템이 안전하기 위한 기본 가정이기 때문에, 일방향성이 깨지거나 일방향 함수에서 키 역할을 하는 난수를 추정할 수 있다면 시스템은 위험해진다.

일방향성의 위험 요소는 두 가지이며, 그중 하나는 일방향 함수의 일방향성을 깨는 것이고, 다른 하나는 일방향 함수의 충돌 쌍을 찾는 것이다.

4.1.1 일방향 함수 깨기

그림 8은 키 생성 과정을 기술하고 있다. 그림에서 DES_k의 생성정보인 S_k를 기준으로 생성 이전과 생성 이후를 ①, ②로 구분하였다. 여기서 일방향 함수 깨기는 ②에 해당된다. 즉, 이 공격은 다음 에이전트 서버가 자신에게 전달된 C_k로부터 S_k를 계산하지 못하도록 $R3_k$와 S_k를 해쉬함수로 뒤섞는 과정을 역계산하는 것이다. 이 공격을 정리하면 아래와 같다.

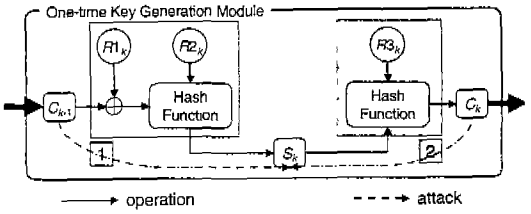

그림 8 키 생성과정 공격

◈ AS_k의 C_k를 알고 S_k를 찾는 공격

이 공격은 C_k로부터 S_k를 추론하는 공격이다. 즉 해쉬함수의 역함수 찾을 수 있다면, *DES*키가 S_k로부터 계산되기 때문에 에이전트 데이타의 비밀성이 침해될 수 있다. 이 공격이 가능하기 위해서는 S_k로부터 C_k를 구하는 함수가 일방향성을 가지지 않아야 하지만, *SHA*−1이 일방향 해쉬함수라고 가정하고 있으므로, 공격의 성공 가능성은 계산학적으로 불가능하다[11].

4.1.2 난수 대치

일방향성의 다른 위협 요소는 일방향 성질의 충돌 쌍을 찾는 것이다. 여기서 충돌회피 함수란 $h(x')=h(x)$이고 $x' \neq x$인 메시지 쌍 x와 x'를 찾는 것이 불가능한 함수를 의미한다. 일방향 해쉬함수는 충돌회피 함수이다[11]. 충돌쌍을 찾는 시도를 정리하면 아래와 같다.

◈ 에이전트 서버 AS_k에 의해 생성된 데이타의 비밀성을 공격하기 위해 AS_k의 입력 C_{k-1}과 출력 C_k를 생성하는 난수 $R1_k$, $R2_k$, $R3_k$에 대해서 동일한 C_{k-1}과 C_k를 가지는 난수 $R1_{k'}$, $R2_{k'}$, $R3_{k'}$을 생성

◈ $S_k = SHA-1(R1_k \oplus C_{k-1}) \oplus R2_k$의 $R1_k$, $R2_k$를 난수 $R1_{k'}$, $R2_{k'}$로 대치

이는 주어진 x에 대해서 충돌 쌍을 찾는 것으로 "충돌회피 해쉬함수" 정의에 의해서 *SHA*−1과 "일회용 키 생성 시스템"이 일방향성을 가지므로 난수 대치에 의한 공격으로부터 안전하다.

4.2 DES 키 위협

*DES*키의 안전성을 고려할 경우, *DES*키는 에이전트 데이타처럼 자유롭게 생성되는 값이 아니라 난수와 해쉬함수에 의해서 생성되는 값이기 때문에 키 자체에 대한 안전성뿐만 아니라 키 생성과정의 안전성도 고려하여야 한다.

*DES*키에 대한 공격에는 해당 에이전트 서버가 사용한 *DES*키를 찾는 것과 수정하는 것이 있다.

4.2.1 DES 키 찾기

*DES*키 찾기는 암호화된 데이타인 $DES_k(data_k)$에 대한 무작위 공격이나 키 생성의 모든 과정에서 *DES*키를 직접 찾는 것을 말한다. 그림 8에서 ①과 ②가 모두 여기에 해당되며, 이중 ②는 일방향 함수 깨기와 동일한 공격이다. 이러한 공격을 정리하면 아래와 같다.

◈ $DES_k(data_k)$ 이용하여 무작위 *DES*키 찾기
◈ $AS_{0\text{-}pub}(R1_k,\ R2_k,\ R3_k,\ DSA_k(X,\ TimeStamp),\ TimeStamp)$를 복호화
◈ 주어진 C_{k-1}를 이용해서 $R1_k$, $R2_k$를 유추하여 S_k를 찾는 공격(그림 8의 ①)
◈ AS_k의 C_k를 알고 S_k를 찾는 공격 (그림 8의 ②)

무작위 *DES* 키 찾기는 *DES*의 보안 강도에 좌우된다. 그리고 *DES*키의 안전성은 키의 사용 횟수에 좌우되는데 많을수록 안전하지 못하다. 그러나 본 시스템에서 *DES*키는 생성후 단 한번만 사용하므로 상대적으로 안전하다. $AS_{0\text{-}pub}$로 암호화 데이타를 복호화하는 것도 공개키 암호알고리즘인 *RSA*의 보안 강도에 좌우된다. 그리고 그림 8의 ①을 정상적으로 수행하면 $2^{80+160} \rightarrow 1.76\times10^{72}$의 시도가 있어야 하는데 이건 *DES*의 키 길이인 64비트 보다 큰 수로 *DES*키 자체보다 더 안전하다. 그림 8의 ②는 4.1.1에서 이미 언급하였다. 즉 일회용 에이전트 키 생성 시스템의 안전성은 위 3가지 알고리즘 중에서 가장 취약한 것에 의해서 결정된다. 그러므로 본 시스템에서 제안된 일회용 에이전트 키는 내부에서 사용된 암호학적 알고리즘보다 안전하기 때문에 충분히 안전하다.

4.2.2 DES 키 수정

*DES*키는 단독으로 생성되지 않기 때문에 임의적인 *DES*키의 생성 및 삭제가 불가능하다. 하지만 앞 절의 난수 대치와 같이 *DES*키를 수정할 수 있다. 즉 키들 사이의 상호연관관계가 유지된다면, 가상의 난수 $R1'$, $R2'$, $R3'$를 이용해서 적법한 *DES*키로 대치할 수 있다. 이 위협은 일방향성을 해치는 난수 대치를 이용하기 때문에 4.1.2의 난수 대치와 동일한 보안 위협으로 귀착된다.

4.3 에이전트 데이타 위협

에이전트 데이타는 훔쳐보기, 삽입, 변경 등의 모든 위협요소를 가지고 있다.

4.3.1 에이전트 데이타 훔쳐보기

에이전트의 데이타는 일회용 에이전트 키인 *DES*로 암호화되기 때문에 에이전트 데이타를 훔쳐보기 위해서는 *DES*키를 찾아야만 한다. 즉 에이전트 데이타 훔쳐보기는 4.2.1의 "*DES*키 찾기"의 보안 위협과 동일하다.

4.3.2 조작된 에이전트 경로 및 데이타 삽입

악의를 가진 에이전트 서버가 조작된 에이전트 데이타를 삽입하는 것은 가능하다. 우선 악의를 가진 서버가 그림 9에서처럼 이동경로 상에 존재하지 않는 가상 서버를 생성하고 자신이 가상 서버의 일을 처리하고 조작된 데이타를 삽입하는 방법이다. 이 경우 가상 서버의 동작이 일반 서버의 동작과 동일하게 처리되어 정당하지 않은 데이타의 삽입은 가능하지만, 전자서명 알고리즘 *DSA*에 의한 부인방지 기능에 따라 사후 악의에 의해 추가된 데이타의 삽입을 감지할 수 있다.

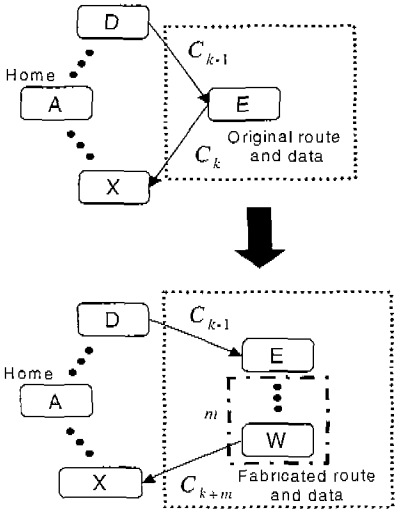

그림 9 조작된 에이전트 경로 및 데이타 삽입

4.3.3 에이전트 데이타 변경

에이전트 데이타의 변경은 두 가지 방법이 있다. 정당하게 데이타를 변경하는 공격과 임의로 변경하는 공격이 있다. 여기서 정당하다는 의미는 정확한 *DES*키를 이용하여 공격하는 방법을 의미한다.

◈ 정당한 *DES*키를 찾거나 생성하여 에이전트 데이타를 변경
◈ 에이전트 데이타의 임의 변경

정당한 *DES*키를 아는 것은 4.2.1의 *DES*키 찾기의 문제로 귀착이 된다. 그리고 키들 사이의 상호연관관계를 만족하는 유효한 *DES*키(난수 대치를 통한 공격)를 찾을 수 있을지라도 해당 서버의 비밀키를 알 수 없기 때문에 정당한 변경이 아닌 임의의 변경만이 가능하다. 그러나 에이전트 데이타의 임의 변경은 홈 에이전트 서버에 의한 키 생성과정으로부터 얻어진 *DES*키에 의해 검증된다.

4.4 공모에 의한 가상 데이타 생성 위협

지금까지의 보안위협 분석은 각 에이전트 서버에 의한 단독 보안공격에 대해 분석했다. 본 절에서는 하나가 아닌 여러 에이전트 서버들이 공모해서 만들 수 있는 위협에 대해서 분석한다. 단, 각 에이전트는 자신이 생성한 데이타를 보호하기 위해서 노력한다고 가정한다. 이때 해당 에이전트 서버에서의 작업을 함수 F로 표현하면

$$C_k = F(C_{k-1}),$$

단 $F(X) = SHA-1(SHA-1(R1_K : R1_K \oplus X) \oplus R2_K \oplus R3_K$

이며 m개의 서버를 경유하는 데이타는

$$C_k = F^m(C_{k-m})$$

로 표현된다. 에이전트의 실제 이동 경로를 정확히 알지 못한다면 그림 9과 같이 $C_k = F(C_{k-1})$ 대신에 $C_k = F^m(C_{k-m})$을 삽입하여 가상의 결과를 생성할 수 있다. 그리고 그림 10과 같이 에이전트 서버 AS_{k-m}와 AS_k의 공모로 중간 경유 데이타를 가상으로 생성할 수 있다. 전자의 경우는 조작된 에이전트 데이타의 삽입에 해당되는 내용이며, 후자의 경우는 해당 서버의 과거 서명 내용을 이용하여 재사용(replay) 공격을 하는 것이다. 그림 9에서 E', …, Q'은 공격그룹인 D, R에 의해서 재사용 공격으로 만들어진 가상의 데이타이다. 이는 정당하지 않은 데이타를 완전히 정당하게 만들 위험이 있기 때문에 매우 위험하다. 그러나 어떤 서버에서 생성된 데이타의 암호문 X와 생성된 시각의 *TimeStamp*를 함께 전자서명을 실행하므로, 재사용 공격에 의한 공격 그룹이 원래 경로 E, …, Q의 전자 서명을 알고 있다고 하더라도 홈 에이전트 서버에서 부정한 데이타의 생성 및 삽입을 감지할 수 있다.

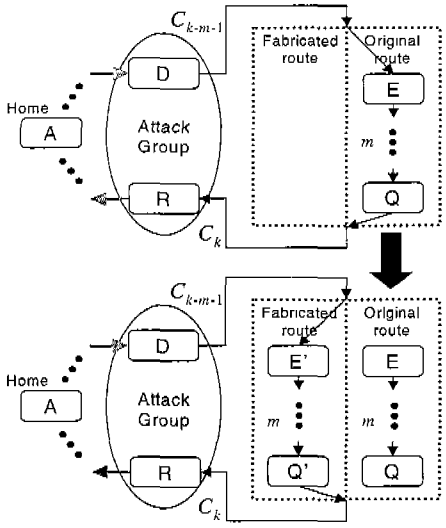

그림 10 공모에 의한 가상데이타 생성

5. 결 론

본 논문에서 악의를 가진 서버들로부터 에이전트 데이타를 보호하기 위해서 "일회용 에이전트 키"를 제안하였다. 그리고 어떠한 종류의 공격에 대해서도 안전함을 보였다. 제안된 시스템이 안전하기 위해서 $SHA-1$이 일방향 해쉬함수라고 가정하였다. 하지만 $SHA-1$이 일방향 성질을 만족하지 않는다 하여도 키 생성 시스템은 사용되는 해쉬함수에 독립적이므로 충분히 안전한 해쉬함수로 변경할 수 있다.

일회용 에이전트 키 생성 시스템의 기본 아이디어는 연결고리(coupler)를 통해서 연속적인 키들 사이에 일정한 상호연관관계를 제공하는 것이다. 결론적으로 에이전트가 이동하면서 수집한 데이타들의 수정이나 공격은 홈 에이전트 서버가 연속되는 키들 사이에 설정된 상호연관관계를 이용해서 최종 검증을 하게 되고 이때, 이 특징을 이용해서 무결성 공격을 감지하게 된다.

참 고 문 헌

[1] B.H. Tay and A.L. Ananda, "A Survey of Remote Procedure calls," *Operating Systems Review*, vol 24, No.3, pp.68-79, July 1990.

[2] W. Farmer, J. Guttman, and V. Swarup, "security for mobile agents: Authentication and state appraisal," *the European Symposium on Research in Computer Security(ESORICS)*, Lecture Notes in Computer Science, September 1996.

[3] W. Farmer, J. Guttman, and V. Swarup, "Security for mobile agents: Issues and requirements." *National Information Systems Security Conference*, National Institute of Standards and Technology, October 1996.

[4] G. Karjoth, D. B. Lange, and M. Oshima, "A Security Model for Aglets," *IEEE Internet Computing*, Vol. 1, No. 4, pp.68-77, July - August 1997.

[5] H. Peine, "Security Concepts and Implementation in the Ara Mobile Agent System," *7th IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Stanford University, USA, June 1998.

[6] R. Gray, "Agent Tcl: A flexible and secure mobile agent system," *In Proceedings of the Fourth Annual Tcl/Tk Workshop*, Monterey, Cal., pp.9-23, July 1996.

[7] John K. Ousterhout, Jacob Y. Levy, and Brent B. Welch, "The Safe-Tcl Security Model," TR-97-60,

March 1997.
[8] T. Sander and Chr. Tschudin, "Towards Mobile Cryptography," *the IEEE Symposium on Security and Privacy*, 1998.
[9] J. Baumann, F. Hohl, K. Rothermel, and M. Strasser, "Mole-Concepts of a mobile Agent System," *The World Wide Web Journal*, special issue on Software Agents, 1998.
[10] F. Hohl, "Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts," *Mobile Agents and Security*, Springer-Verlag, pp.99-113, 1998.
[11] Douglas R. Stinson, "Cryptography theory and practice," pp233, CRC press,1995.
[12] Jusung Baek, "A design of a protocol for detecting a mobile agent clone and its correctness proof using Coloured Petri Nets". technical report TR-DIC-CSL-1998-002, Information and Communications, K-JIST, 1998.
[13] Jusung Baek, R. S. Ramakrishna and Dong-Ik Lee, "A design of a protocol for detecting an Agent clone in Mobile Agent Systems and its Correctness Proof". *ACM Symposium on Principles of Distributed Computing*, ACM press, pp. 269, May 1999.
[14] Neil M. Haller, "The S/KEY One-Time Password System," *Proceedings of the ISOC Symposium on Network and Distributed System Security*, San Diego, CA, February 1994.
[15] L.R. Knudsen, X. Lai, and B. Preneel, "Attacks on fast double block length hash functions," *Journal of Cryptology*, Vol 11, No. 1, pp. 59-72, Winter 1998.

박 종 열
1996년 충남대학교 컴퓨터공학과 졸업. 1999년 광주과학기술원 정보통신공학과 석사. 1999년 ~ 현재 광주과학기술원 정보통신공학과 박사과정. 관심분야는 보안 시스템, 이동 에이전트, 이동 코드, 보안 프로토콜, 분산 시스템, Petri net 이론 등

이 동 익
1985년 영남대학교 전기공학과 졸업. 1989년 Osaka Univ. 전자공학과 석사. 1993년 Osaka Univ. 전자공학과 박사. 1990년 ~ 1995년 Osaka Univ. 전자공학과 research associate. 1993년 ~ 1994년 Univ. of Illiois at Urbana- Champaign, visiting assistant professor. 1995년 ~ 현재 광주과학기술원 정보통신공학과 부교수. 관심분야는 에이전트 시스템 및 보안, 비동기회로 설계/CAD, Petri net 이론 등

이 형 효
1987년 전남대학교 계산통계학과 졸업. 1989년 한국과학기술원 전산학과 석사. 2000년 전남대학교 계산통계학과 박과. 2000년 ~ 2001년 광주과학기술원 박사 후과정. 2001년 ~ 현재 원광대학교 정보, 전자상거래학부 전임강사. 관심분야는 보안모델, 운영체제 보안, 전자상거래 보안 등

박 중 길
1986년 동국대학교 전자계산학과 졸업. 1988년 서강대학교 전자계산학과 석사. 1988년 ~ 2000년 국방과학연구소 선임연구원. 2000년 ~ 현재 국가보안기술연구소 선임연구원. 1998년 ~ 현재 충남대학교 컴퓨터과학과 박사과정. 관심분야는 컴퓨터통신보안, 접근통제, 암호이론 등