The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 16, No. 6, pp.83-94, Dec. 31, 2016. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2016.16.6.83
JIIBC 2016-6-10

가상화 환경의 안전한 데이터 공유를 위한 다중 인스턴스간 상호인증 기법

Mutual Authentication Scheme between Multiple Instances for Secure Data Share of Virtualized Environment

최도현*, 김상근**

Dohyeon Choi*, Sangkun Kim**

요 약 최근 클라우드, 빅데이터, 인공지능 등 다양한 분야의 서버 플랫폼이 가상화 기술을 사용하고 있지만 지속적으로 발생하는 구조적인 보안 취약성이 이슈화 되고 있다. 또한 대부분 가상화 보안 기술은 종류가 제한적이고 플랫폼 제공자에 의존적인 것으로 알려져 있다. 본 논문은 가상화 환경의 안전한 데이터 공유를 위한 다중 인스턴스간 상호인증 기법에 대해 제안한다. 제안하는 기법은 다중 인스턴스 간에 독립적인 상호인증을 고려하여 보안 구조를 설계하고 키 체인 기법을 적용하여 데이터 공유에 보안 프로토콜의 안전성을 강화 시켰다. 성능분석 결과 기존 보안 구조와는 다른 방식의 안전한 가상화 인스턴스 세션을 생성하고, 상호인증 과정의 각 인스턴스에 대한 효율성이 우수 하다는 것을 확인하였다.

Abstract Recent cloud, big data, there is a problem for the architectural security vulnerability to the server platforms of various fields such as artificial intelligence occurs consistently, but using the virtualization technology. In addition, most secure virtualization technology is known to be dependent on the type is limited and the platform provider. This paper presents a method for mutual authentication for secure data between multiple instances of a shared virtualized environment. The proposed method was designing a security architecture in consideration of the mutual authentication between multiple independent instances, and enhance the safety of a security protocol for sharing data by applying a key chain techniques. Performance analysis results and the existing security architecture demonstrated that protect each virtualized instances of the session and the other way, a compliance effectiveness for each instance of the mutual authentication process.

Key Words : Virtualization, Mutual Authentication, Hypervisor, Key Management, Multiple Instance

Ⅰ. 서 론

최근 클라우드(Cloud) 서비스의 고도화는 대용량 데이터에 대한 운영 및 관리에 대한 다양한 장점을 제공하였고, 핵심 기술인 가상화(Virtualization) 기술의 발전에 따라 현재 구글, 아마존, IBM 등 선진 기업을 중심으로 차세대 플랫폼이 연구 및 개발되고 있는 실정이다[1][2]. 그러나 보안성을 중심으로 현재 활용되는 가상화 보안 구조들을 분류하고 분석했을 때 각 다른 장·단점을 제공하지만, 방식의 차이점에 한계가 있고 각 플랫폼 제공자의

*정회원, 숭실대학교 컴퓨터공학
**정회원, 성결대학교 컴퓨터공학부
접수일자 : 2016년 9월 28일, 수정완료 : 2016년 10월 28일
게재확정일자 : 2016년 12월 9일

Received: 28 September, 2016 / Revised: 28 October, 2016 / Accepted: 9 December, 2016
**Corresponding Author: sgkim@sungkyul.edu
Division of Computer Engineering, Sungkyul University, Korea

보안 구조에 의존적이라는 문제점이 존재한다[3]. 본 논문에서는 현재 활용되는 가상화 보안 구조의 장·단점을 분석하고 추출된 보안 요구사항에 따른 보안 구조를 설계한다. 보안 기능은 플랫폼 제공자 의존성 문제를 해결하기 위해 독립적인 모듈에서 수행하도록 한다. 또한 가상화 자원의 보호를 위해 세션마다 생성되는 다중 인스턴스 간에 키 체인 기법 기반 상호인증 기법을 적용하였다.

2장은 기존 가상화 보안 기술과 최근 새로 제안된 가상화 구조를 비교분석하고 보안 구조 설계에 앞서 보안 요구사항 추출한다. 3장은 각 보안 요구사항을 해결할 수 있는 제안하는 기법의 보안 구조, 상호인증 기법을 설명한다. 4장은 성능분석, 5장 결론으로 마친다.

II. 관련연구

1. 가상화 보안 기술

최근 국·내외 클라우드 컴퓨팅, 인공지능, 빅데이터 등 차세대 분야에서는 대부분의 플랫폼에서 자원 관리 및 비용의 효율성을 위해 가상화 기술을 채택하여 활용하는 추세이다[4]. 보안 관점에서 가상화 기술은 기존 안티바이러스나 S/W, H/W 방화벽 등 기존 보안 기술과는 다른 보안 계층을 구축한다[5]. 최근 서버 플랫폼은 이러한 구조적인 차이점에 따라 새로운 가상화 환경에 적합한 보안 기술의 연구[6][7]와 개발이 진행되고 있다.

가상화 보안 기술은 가상화 계층에 대한 취약성을 대응하기 위한 보안 기술을 의미한다. 그러나 현재 가상화 기술로 알려진 가상머신 내부정보 분석 기반 침입 탐지, 에이전트리스 가상 보안 어플라이언스[8][9] 등 관련 연구가 많이 부족하고 대부분의 가상화 환경이 하이브리드 클라우드 전환되면서 보안이슈가 복잡해져 다양한 문제가 지적되고 있다[10].

그림 1은 가상화 기술의 핵심인 하이퍼바이저(Hypervisor)구조를 나타낸다[11]. 타입 1은 하드웨어 위에 직접 설치되고, 타입 2는 호스트 운영체제 위에 설치되는 차이점이 있다. 하이퍼바이저의 위치는 보안 계층을 담당하는 VMM(Virtual Monitor Machine)과 직접적인 관계가 있다. 타입 1과 타입 2를 비교했을 때 성능적인 면에서는 타입 1인 가상머신 내부 정보 분석 기반 침입 탐지가 효율적이지만, 관리적인 효율성을 고려했을 때 호스트 운영체제 위에서 운영되는 타입 2가 효율적인 것을 알 수 있다.

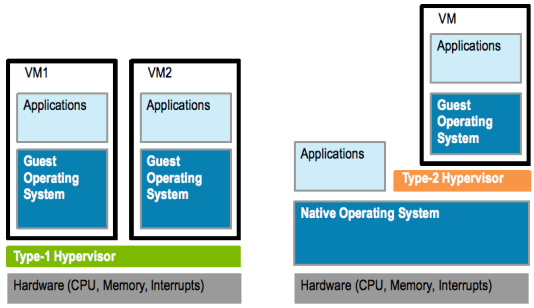

그림 1. 하이퍼바이저의 구조(Type1, Type2)
Fig. 1. Structure of Hypervisor(Type1, Type2)

타입 2인 에이전트리스 보안 어플라이언스 구조는 성능 효율성이 낮은 단점과 함께 하드웨어 위에 각 에이전트(보안 계층)를 구현해야 하는 어려움이 있다. 최근 빅데이터 등 대용량 데이터의 분산 실시간 처리를 목적으로 가상화 기술들이 구현되는 추세이다[12]. 보안 구조 설계에 앞서 가상화 보안 구조는 반드시 최신 플랫폼에 대한 데이터 처리의 효율성과 안전한 보안 구조를 제공해야 한다. 결론적으로 타입1과 타입2 하이퍼바이저는 모두 장·단점을 가지고 있지만 단점을 해결하기 위해서는 새로운 보안 구조가 설계가 요구되었다.

2. 차세대 플랫폼의 가상화 기술

차세대 플랫폼 서버 분야에서 구글, 레드햇, 오픈스택 등은 새로운 가상화 기술로 Docker를 채택하였다[13]. 컨테이너(Container)라는 개념을 도입하여 기존의 타입 2 하이퍼바이저와 같은 호스트 기반이지만 성능의 효율성이 높고 플랫폼과 하드웨어에 독립된 환경을 제공하는 특징이 있다. 컨테이너는 프로그램이 작동하기 위한 최소한의 요소들을 묶어 패키징(Packaging)한 것으로 운영체제 보다 매우 작으면서 독립적인 배포와 실행을 가능하게 하는 일종의 가상 머신이다[14].

그림 2는 기존 타입 2 하이퍼바이저와 Docker의 차이점을 나타낸다[15]. 분리된 게스트 운영체제에서 어플리케이션을 관리할 필요 없이 최소한의 바이너리와 라이브러리를 공유하고, Docker가 관리하는 형태로 효율성과 편의성을 모두 제공한다. Docker와 같이 기존 타입 2 하이퍼바이저의 가장 큰 문제인 성능적인 효율성 문제를 해결할 경우 보안 구조를 설계하는데 다양한 방향성을 제

공해 줄 수 있다.

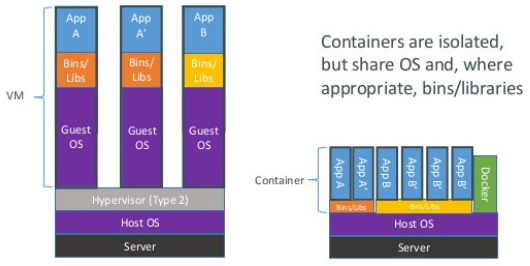

그림 2. 컨테이너와 기존 가상화 방식(VM)의 차이점
Fig. 2. Different between Container and VM

실시간 대용량 데이터 처리에 적합한 가상화 기술은 타입 1을 하이퍼바이저이다. 타입 1의 보안 구조를 고려하여 설계할 경우 구현의 어려움, 데이터 접근 문제, 관리 문제 등 단점을 해결해야 하는 문제가 있지만 Docker와 같은 새로운 가상화 기술의 등장은 이러한 한계점을 해결할 수 있을 것으로 예상된다. 결론적으로 Docker와 같은 새로운 가상화 기술 기반으로 새로운 보안 구조를 설계할 경우 타입 1의 성능 효율성과 타입 2의 관리 효율성 모두 충족할 수 있을 것으로 기대할 수 있다.

3. 가상화 기술의 보안 구조 문제점 분석

표 1은 타입 1, 2 하이퍼바이저의 보안 계층에 존재하는 VMM 기능을 기존 가상화 보안 구조 기술에 적용했을 때 설계상의 문제점을 나타낸다.

① 데이터 처리 효율성 : 내부정보 기반과 에이전트리스는 기존 알려진 기술로써 성능의 효율성이 알려져 있으며 구글 Docker와 같은 새로운 가상화 구조는 효율성이 높다고 알려져 있지만 구글에서 제공하는 플랫폼 서버 이외에 검증되지 않았기 때문에 실질적인 효율성 분석이 요구되었다.

② 데이터 분석 및 접근성 : 보안 구조상 내부정보 기반은 메타데이터의 분석 및 접근성이 어려운 것으로 알려져 있으며 비교적 어려움이 낮은 에이전트리스는 보통 수준으로 정의 되었으며, Docker의 경우 가상화 기술 구조상 컨테이너로 독립된 장점을 이용할 경우 데이터 분석 및 접근성이 높을 것으로 분석되었다.

표 1. 가상화 기술의 보안 구조의 문제점 분석
Table 1. Architecture Analysis of Security Problems of Virtualization

문제점	내부정보 분석 기반 침입 탐지	에이전트리스 가상 보안 어플라이언스	구글 Docker
데이터 처리 효율성	낮음	높음	검증되지 않음
데이터 분석 및 접근성	어려움	보통	쉬움
호환성 및 확장성	높음	높음	검증되지 않음
기존 보안기술 적용	어려움	어려움	어려움
접근 권한 분배	플랫폼 서버에 의존적	플랫폼 서버에 의존적	플랫폼 서버에 의존적
보안 계층 VMM의 독립성	제한적	제한적	제한적

③ 호환성 및 확장성 : 타입 1, 2에 적용되고 있는 기존 보안 기술의 경우 데스크톱 또는 서버로 분류되어 용도가 명확하다. 이는 오랜 기간 동안 연구 개선된 기술로써 플랫폼 서버의 용도를 고려했을 때 호환성 및 확장성을 준수한다고 할 수 있다. Docker의 경우 독립된 컨테이너 구조의 장점으로 확장성은 제공하지만 구글 플랫폼이 아닌 이외 플랫폼 서버의 호환성에 대한 테스트가 요구되는 것으로 분석되었다.

④ 기존 보안 기술 적용 : 가상화된 자원의 분산처리, 가상 스위칭으로 인한 데이터 포맷의 차이점 등 가상화 환경의 특징으로 인해 기존 보안 구조 실시간 데이터 처리에 대한 보안성을 제공하는 기술은 각 플랫폼 서버의 가상화 보안 구조에 의존적인 것으로 분석되었다.

⑤ 접근 권한 분배 : 가상화된 자원의 최상위 레벨의 접근 권한 분배가 내부 가상화 영역에서 처리되기 때문에 플랫폼 서버에 의존적인 것으로 분석되었다. 이는 가상화 영역에 보안 기술 적용에 앞서 특정 중요 데이터에 대하여 접근 권한을 분배할 수 있는 방안이 요구되는 것으로 분석되었다.

⑥ 보안 계층 VMM의 독립성 : 오픈소스 기반 플랫폼 서버의 경우 하이퍼바이저 보안 계층의 분리하여 개발할 경우 독립성을 제공할 방안이 존재하지만, 대부분의 서버 플랫폼의 가상화 기술이 오픈소스가 아니기 때문에 독립적인 모듈화 가능성이 제한적인 것으로 나타났다. 이는 타입 1, 2 하이퍼바이저 모두 공통적으로 활용할 수 있는 표준 VMM 기술이 존재하지 않기 때문인 것으로 분석되었다.

Ⅲ. 다중 인스턴스간 상호인증 기법

1. 제안하는 기법의 보안 구조와 보안 고려사항

표2는 제안하는 다중 인스턴스 기법의 구성요소 및 역할을 나타낸다.

표 2. 다중 인스턴스 기법의 구성요소 및 역할
Table 2. Multiple Instances of Components and Roles

구성요소	설명
User	사용자 : 인증 요청 및 정보 전송
SP	서비스 제공자 : 인스턴스 생성 요청 및 보안 파라미터 전송
PP	플랫폼 제공자 : 서비스 제공자를 경유하여 인스턴스 생성 및 상호인증 파라미터 검증
AGT	가상화 영역 내 생성된 인스턴스(자원) 별 보안 에이전트
MAGT	마스터 에이전트 : 어플리케이션의 보안 에이전트 관리
AS	서비스 제공자의 인증서버 : 기존에 구축된 상호인증 가능한 인증서버
AMT	서비스 제공자의 에이전트 관리자 : 플랫폼 제공자의 MAGT와 직접 통신
IMT	플랫폼 제공자의 인스턴스(APP) 관리자 : 인스턴스 관리자

그림 3은 서비스 제공자와 플랫폼 서버 내부의 가상화 보안 구조와 관계를 나타낸다. 호스트 기반 하이퍼바이저 타입 2 방식 구조를 기반으로 구글 Docker와 같은 바이너리와 라이브러리를 공유하면서, 가상 인스턴스를 생성하여 효율성의 장점을 유지하고, 인스턴스 보호를 위해 계층별로 동적인 보안 에이전트를 생성한다.

VMM 역할을 수행하는 호스트 운영체제 위치에서 하위 에이전트(AGT)를 관리하는 마스터 에이전트(MAGT)를 두고, 인스턴스 매니저(IMT)에서 각 보안 에이전트에 대한 권한 레벨 변경 기능을 수행하는 구조이다.

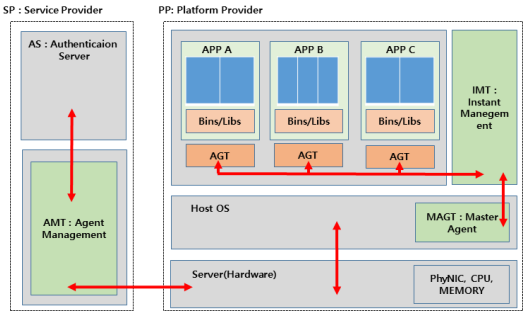

그림 3. 제안하는 가상화 보안 구조
Fig. 3. Proposed Virtualization Security Architecture

상호인증 과정에서 서비스 제공자의 AMT와 플랫폼 제공자의 MAGT 접근 권한의 위치를 분리한다. 제한된 구조는 플랫폼 제공자에 가상화 구조를 변경하지 않으면서 보안 기능을 적용할 수 있도록 설계되었다.

2. 초기 키 교환 및 보안 파라미터 생성

접근 권한 분리에 앞서 선행되어야 할 핵심 과정은 서비스 제공자의 AMT와 플랫폼 제공자의 MAGT의 상호인증이다. 표 3은 초기 상호인증을 위한 보안 파라미터의 요약을 나타낸다.

표 3. 초기 키 교환 및 보안 파라미터
Table 3. Initial Key Exchange and Security Parameter

파라미터	설명
Pri	User와 SP의 개인키(512비트 정수열)
Pub	User와 SP의 공개키(2048비트 정수열)
C	암호문(512비트 블록, 패딩 있음)
P	사용자 인증정보: 메시지
Sk	세션키(256비트, 해쉬함수 사용)

그림 4는 초기 키 교환 과정 및 가상화 내부의 인스턴스 보안 파라미터 생성과정을 나타낸다.

① 사용자 등록 요청 : 사용자는 반드시 서비스 제공자를 통해 정상적으로 가입된 사용자여야 한다. 가입된 이후 로그인 과정에서 아이디, 패스워드를 이용하여 사용자 인증 과정을 수행할 수 있다.

② 공개키 교환 : 사용자는 서비스 제공자와 개인키(Pri) 생성과 공개키(Pub) 교환 과정을 수행하고,

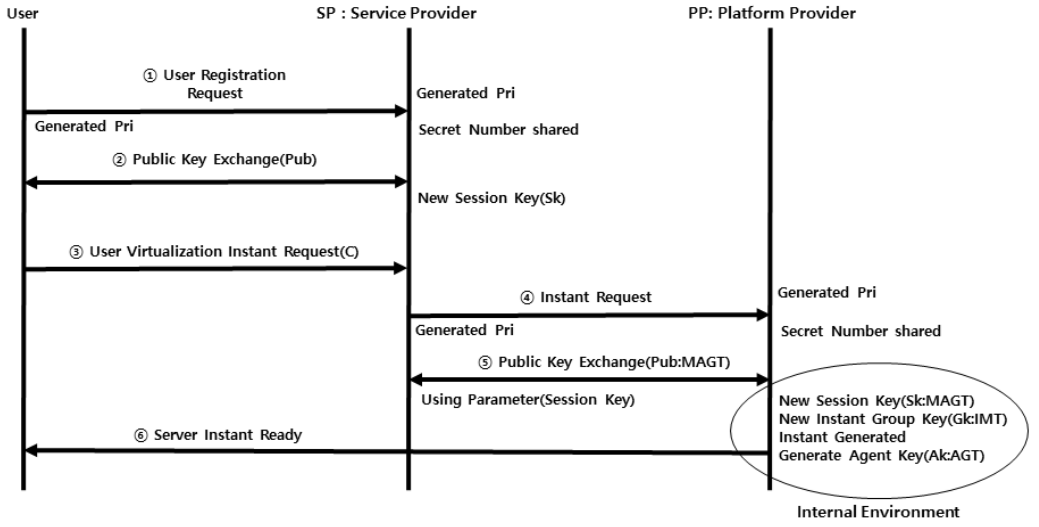

그림 4. 초기 키 교환 및 가상화 내부 보안 파라미터 생성과정
Fig. 4. Initial Key Exchange and Virtualized Internal Security Parameter Generation Process

상호인증에 필요한 주요 비밀 값을 공유한다. 초기 통신 노드(사용자와 서비스 제공자)는 기존에 범용 적으로 사용되고 있는 SSL/TLS와 같은 표준 보안기술을 활용할 수 있다. 이 과정에서 서비스 제공자는 공유된 비밀 값으로 생성한 세션키(Sk)를 유지한다.

③ 사용자 가상 자원 요청 : 사용자의 가상 자원 요청이 발생할 경우 서비스 제공자를 경유하여 암호화된 인스턴스 생성 요청(C)을 플랫폼 제공자에 전송한다. 이는 로그인이 정상적으로 완료된 시점을 가정한다.

④⑤ 서비스 제공자 자원 요청과 공개키 교환 : 가상 자원 생성 이전에 서비스 제공자와 플랫폼 제공자는 개인키(Pri) 생성과 공개키(Pub) 교환을 수행한다.

요청된 어플리케이션의 인스턴스 메타 데이터 구조가 확정된 이전에 생성된 세션키(Sk)를 이용하여 각 인스턴스에서 사용할 그룹키(Gk)를 생성한다. 수식 1은 초기 키 교환 과정에서의 보안 파라미터 생성 방법을 나타낸다. SSL/TLS에서 일반적으로 사용되는 RSA 기반 암·복호화, 개인키를 이용한 메시지 인증코드, 랜덤 값과 공개키를 보안 파라미터로 이용한다.

세션 키 생성 방법은 수식 (1)을 사용한다.

$$
\begin{aligned}
&Pri = Select\,2048bit\ Secret\ Key \times Pub \\
&Pub = e, n(p \times q) \\
&C = (P^e \bmod n) \| MAC \\
&P = C^d \bmod n \\
&MAC = (H(P))^{Pri} \\
&Sk = (H(Random))^{Pub}
\end{aligned}
\tag{1}
$$

표 4는 플랫폼 제공자 내부의 가상화 영역의 상호인증을 위한 보안 파라미터 요약을 나타낸다.

표 4. 가상화 영역의 보안 파라미터
Table 4. Security Parameters of Virtualized Area

파라미터	설명
Pri	SP와 PP의 개인키(160비트 정수열)
Pub	SP와 PP의 공개키(163비트 정수열)
C	암호문(128비트 블록, 패딩 있음)
P	내부 인스턴스 요청 보안 파라미터 : 메시지
Sk	세션키(256비트, 해쉬함수 사용)
MAC	무결성 검증용 메시지 인증 코드

가상화 내부 영역의 상호인증은 기존 공개키 기반 암호 알고리즘을 적용했을 때 효율성이 떨어지기 때문에 RSA와 비교하여 연산 효율성이 높은 것으로 알려진 타원곡선(ECC) 기반 암·복호화를 적용한다. 수식 (2)는 플

랫폼 제공자 내부의 가상화 영역의 보안 파라미터 생성 방법을 나타낸다.

$$
\begin{aligned}
&Pri = Select\,160bit\,Secret\,Key \\
&Pub = (G, p, Y = xG) \\
&C = (C_1, C_2 = x'G, x'Y + P) \| MAC \\
&P = (C_2 - xC_1) \\
&MAC = (H(P))^{Pri} \\
&Sk = (H(Random))^{Pub}
\end{aligned}
\tag{2}
$$

그룹키에 포함되는 보안 파라미터 Gk와 Ak는 다음 절에서 설명한다.

⑥ 가상 자원 준비 : 플랫폼 제공자는 사용자에게 가상 자원이 생성되었고 준비가 되었다는 메시지를 전송한다.

4. 인스턴스 생명주기와 보안 파라미터 생성

가상화 내부 영역은 사용자의 자원 요청 또는 내부 자원 관리를 위해 동적으로 변경 관리되는 특징이 있다. 그림 5는 가상화 내부 영역에서 상태에 따른 인스턴스의 상호작용을 나타낸다.

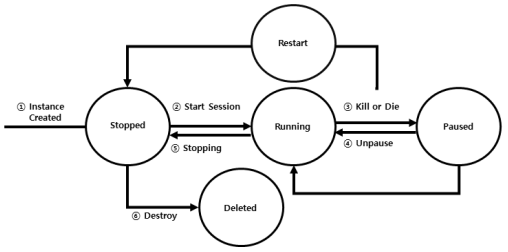

그림 5. 가상화 내부 인스턴스의 상호작용
Fig. 5. Interaction of Internal Virtualized Instances

① 인스턴스 생성 : AMT의 요청으로부터 초기 인스턴스가 생성된 상태로 MAGT와의 상호인증이 선행되어 세션키 Sk를 전달받는다.

② 보안세션 생성 : MAGT는 IMT로부터 현재 생성되어 있는 분산된 인스턴스 영역에 대한 정보를 얻고 IMT는 각 AGT에 할당할 그룹키를 생성한다. 가상화 내부 영역의 공유 자원은 대용량의 데이터를 처리하기 위해 하나의 서버가 아닌 다중 서버에 분산된 자원으로 공유되는 것이 일반적이다. 본 논문에서는 가상화 영역의 대용량 데이터 할당을 고려하여 대규모 네트워크에 적합한 클러스터 트리 토폴로지 기반의 그룹키 체인 방법을 적용한다. 그림 6은 루트 클러스터 MAGT 로부터 파생된 AGT의 트리 토폴로지 구조를 나타낸다.

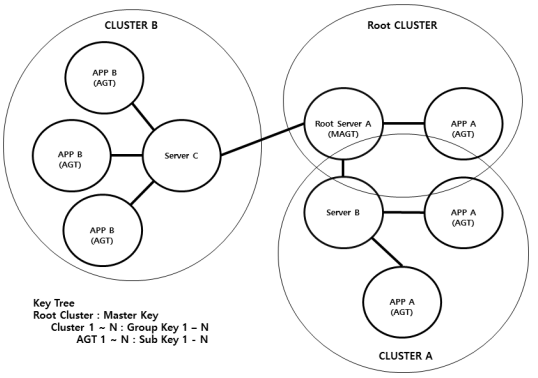

그림 6. 클러스터 트리 토폴로지 구조
Fig. 6. Cluster Tree Topology Structure

루트 클러스터(서버 A)의 MAGT는 그룹키를 위한 마스터키를 생성한다. 1-N의 서브 클러스터(서버 A, 서버 B)는 IMT를 통해 각 클러스터 그룹키를 생성한다. 이후 각 클러스터 내부에서 AGT 아래 각 자원(APP)의 접근을 위한 서브키를 부여한다. 표 5는 키 체인 기법 적용을 위한 보안 파라미터 요약을 나타낸다.

표 5. 키 체인 기법 적용을 위한 보안 파라미터
Table 5. Security parameters for applying keychain

파라미터	설명
Gk	MAGT의 마스터키(256비트, 세션키 이용하여 생성)
GS	IMT의 클러스터 키 그룹(256비트, 마스터키 이용하여 생성)
Ak	AGT의 서브 키 그룹(128비트, 그룹키 이용하여 생성)
GC, AC	IMT와 AGT에서 보유하는 암호문, MAC 포함
GP, SF	IMT와 AGT의 고유 파라미터, 모두 타임스탬프 포함
GMAC, AMAC	무결성 검증 위한 메시지 인증 코드

수식 (3)은 키 체인 기법 적용을 위한 마스터키, 그룹키, 서브키 생성방법을 나타낸다.

$$
\begin{aligned}
&GK(Master\ Key) = 256bit\ Random\ Select^{Sk} \\
&GS(Group^{1-n}\ Key) = (Cluster^{ID})^{GK} \\
&GC = ((GP) \| GMAC)^{GS} \\
&GP = MAGT^{ID}, MAGT^{TS} \\
&GMAC = H(GP) \\
&AK(AGT^{1-n}\ Key) = (APP^{ID})^{GS} \\
&AC = ((SF) \| AMAC)^{AK} \\
&SF = APP^{ID}, APP^{TS^{(n,c,d,m,l)}} \\
&AMAC = H(SF)
\end{aligned} \tag{3}
$$

루트 서버인 A로부터 부여된 마스터키는 기존에 생성된 세션키를 기반으로 생성한다. 이후 IMT, AGT에서 보유하는 암호화 데이터(GC, AC)는 고유 아이디(ID)와 타임스탬프(TS)를 포함한다. 최하위 수준의 APP의 경우 잦은 자원할당에 대한 변경정보를 지속적으로 확인해야 한다. 이를 위해 타임스탬프에는 생성(n), (변경(c), 삭제(d), 이동(m), 최근 변경 사항(l))에 대한 정보와 무결성 검증을 위한 GMAC과 AMAC을 포함한다.

③ 프로세스 정지 : 프로세스 정지된 경우 그룹키 또는 서브키는 접근 권한이 제한되어야 한다. APP의 상태가 정지 또는 일시정지 상태인 인스턴스에 대한 키 체인 식별 정보는 서비스 제공자의 AMT가 지속적으로 권한 테이블로 저장한다. 표 6은 서비스 제공자와 플랫폼 제공자의 권한 분배를 위한 권한 테이블 정보를 나타낸다.

표 6. 서비스 제공자의 권한 테이블 정보
Table 6. Authorization Information Table of Service Provider

파라미터	설명
AMT Request ID	AMT ID, MAGT ID(Response)
Session Number	H(Sk)
APP State	Stopped, Running, Paused, Deleted, Restart
Authority Level	1(MAGT), 2(IMT), 3(AGT)
Cluster Group	A, B, C, D...
AGT Number	1, 2, 3, 4...
Restriction	True, False(Key Level)
FileName, TimeStamp	Test.img, Date(n, c, d, m, l) : Cumulative

서비스 제공자는 AMT에 대한 MAGT ID를 인식하고 해당 APP의 상태에 변화가 발생할 경우 해당 클러스터 그룹의 AGT번호를 인식하여 권한수준을 확인하는 과정을 수행한다. 권한수준(1, 2, 3) 마다 권한제한이 True 상태인지 검증하기 위해서 각 인증 상태에 대한 마지막 상태 정보를 확인한다. 또한 타임스탬프 확인을 통해 파일 변경 내역을 추적할 수 있도록 한다. 권한 테이블은 서비스 제공자에서 제공하는 가장 중요한 정보이기 때문에 서비스 제공자의 개인키로 암호화하여 저장한다.

④ 프로세스 재시작 : AMT의 권한 테이블에서 APP의 상태 정보를 확인하고 APP를 재시작 한다. 보안 세션이 유지되어 있지만 예외상황으로 인해 프로세스를 재시작 할 수 없는 경우 프로세스를 정지하여 접근 권한을 제한한다.

⑤⑥ 프로세스 삭제 : 최상위 MAGT에서 부여된 보안 세션이 종료된 경우 IMT, AGT와 같은 하위 트리에 부여된 모든 키를 폐지한다. 최하위 트리에 존재하는 공유 APP 자원이 해제된 경우 권한 테이블 정보를 확인하여 클러스터 그룹에서 해당 AGT에 대한 접근 권한을 제한하고 해당 공유 APP을 삭제한다.

5. 내부 가상화 영역을 위한 권한분배

플랫폼 제공자의 가상 자원에 대한 접근 권한 정보를 서비스 제공자가 보유하여 접근 권한을 분리하였다. 그림 7은 내부 가상화 영역의 권한분배 과정을 나타낸다.

① 사용자 요청 : 1차적으로 상호인증으로 인한 보안 세션이 생성된 상태에서 접근 권한에 대한 사용자 요청 메시지를 전송을 시작한다.

② 권한 테이블 전송 : 서비스 제공자를 경유하여 초기 권한 레벨이 부여되지 않은 테이블 상태정보를 플랫폼 제공자에 암호화하여 전송한다.

③ 권한 테이블 요청 : 플랫폼 제공자는 내부 자원 변동사항을 확인하고 이에 대한 권한 테이블 정보를 업데이트하여 서비스 제공자에 전송한다.

④ 권한인가 : 서비스 제공자는 정상적인 그룹(마스터키)키를 확인하고 권한 테이블을 업데이트하여 플랫폼 제공자에 전송한다.

⑤ 내부 가상화 영역의 키(그룹키, 서브키) 검증 : 내부 영역에 부여된 각 그룹키와 서브키를 검증하고

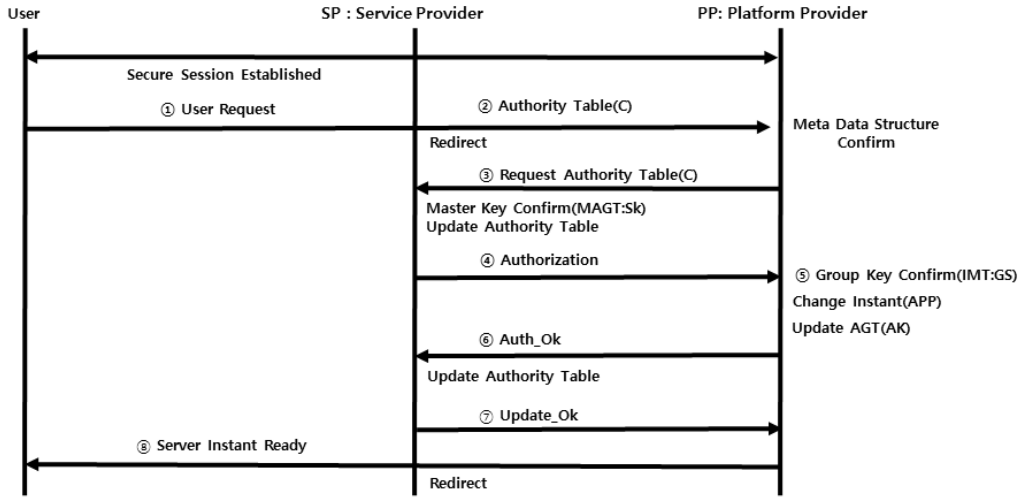

그림 7. 내부 가상화 영역의 권한분배 과정
Fig. 7. Authorized Distribution Process of Internal Virtualization Area

업데이트 한 후 권한 테이블 정보를 전송한다.

⑥⑦ 권한인가 확인 : MAGT부터 AGT 하위 수준까지 테이블 정보의 권한이 승인되었다는 것을 확인하고 메시지를 전송한다.

⑧ 가상자원 준비 : 플랫폼 제공자는 사용자에게 가상자원 변경에 따른 검증 과정이 완료되었다는 메시지를 전송한다.

Ⅳ. 성능 분석

1. 테스트 환경 구축

제안하는 다중 인스턴스 상호인증 기법의 환경 구성은 표 7과 같다. 컨테이너는 우분투 14.04에 Docker를 올려 이미지를 구동시키는 방식으로 구성하였다. 가상 자원 설정을 위해 아파치 웹 서버를 설치하고 JSP 로그인 페이지와 상호인증에 필요한 웹 어플리케이션을 Jmeter를 이용하여 테스트한다.

KVM과 달리 Docker의 경우 어플리케이션이 운영체제 내에서 프로세스로 관리되기 때문에 서버 별 IP 주소 설정의 웹 포트 포워딩 설정 이외에 특별한 네트워크 설정 없이 컨테이너 이미지만 구동하여 실행한다. 또한 NAT 설정으로 인한 네트워크 지연 문제를 고려하여 서버는 내부 IP가 아닌 실제 IP 대역을 할당하여 테스트 하였다.

표 7. 테스트 하드웨어 사양
Table 7. Test Hardware Specifications

구성요소	운영체제	어플리케이션
User	Windows 7 64bit Intel® Core2Duo 2.66 Ghz 6G Memory	Google Chrome (203.253.25.6)
SP(AS)	Linux Kernel 2.6:32(Ubuntu 14.04 64bit) Intel® Core2Duo 3.16 Ghz 12G Memory	Apache Tomcat 7.0, Jmeter Login and Authentication (203.253.25.7)
PP(HV)	Linux Kernel 3.16(Ubuntu 14.04 64bit) Intel® Core2Quad 2.66 Ghz 8G Memory	Apache 2, Docker - KVM (Server A, B, C : 203.253.25.8 - 10)

내부 하이퍼바이저는 루트 서버 A로부터 공유된 이미지를 서버 B와 C에서 공유하는 형태로 구성되었다. Docker 인스턴스에서 사용자 상호인증 기능을 수행하는 이미지를 할당하고, 비교대상은 기본적으로 가상머신을 제공하는 KVM과 동일하게 로그인 및 인증 성능을 비교 분석하였다.

2. 효율성 비교분석

테스트 항목은 Jmeter를 이용한 사용자 인증 구간의

지연속도와 가상화 내부 인스턴스 생명주기에 따른 성능을 비교분석한다. 표 8, 9은 지연(평균)에 대한 성능분석 결과(단위:초)를 나타낸다.

표 8. 성능분석 결과(Jmeter – KVM)
Table 8. Performance Analysis Result(Jmeter – KVM)

	외부	내부	전체
로그인 요청	0.512	0.974	1.486
로그인 수행(상호 인증)	1.764		1.764
컨테이너 접근권한 분배	-		
컨테이너 검증	-		
평균 합계			3.250

외부는 SSL/TLS가 적용된 노드, 내부는 가상화 내부에 대한 노드를 나타낸다. KVM과 제안 환경 동일하게 루트 서버 A에서 서버 C를 경유하여 내부 자원에 접근하였다.

표 9. 성능분석 결과(Jmeter – Proposal)
Table 9. Performance Analysis Result(Jmeter – Proposal)

	외부	내부	전체
로그인 요청	0.048	0.085	0.133
로그인 수행(상호 인증)	0.081		0.081
컨테이너 접근권한 분배(전체 과정)	0.082	0.021	0.103
컨테이너 검증(전체 검증)	0.093	0.182	0.265
합계			0.583

KVM은 로그인과 상호인증 구간에서 전체 평균 약 3.25초, 제안 프로토콜은 KVM과 동일한 로그인과 상호인증 구간에서 평균 약 0.21초로 Docker 기반 환경이 기본적인 네트워크 성능이 훨씬 높은 것으로 나타났다. 제안 프로토콜은 접근권한 분배와 컨테이너 검증과정을 포함하고도 전체 평균 약 0.58초로 KVM과 비교 하여 평균 지연 효율성이 % 높은 것으로 나타났다.

Docker의 자체 지연속도만을 고려했을 때 제안 기법 내부 영역 0.36초는 전체 평균 지연 약 0.58초와 비교하여 % 지연이 추가적으로 발생함을 알 수 있다. 제안 기법의 보안 기능 수행으로 인한 지연 약 0.36초는 KVM의 로그인 요청 약 0.51초보다 오히려 빠른 것으로 나타나 실제 사용자 인증 전 과정에 큰 영향을 끼치는 않는 지연속도 수준임을 확인할 수 있다. 표 10은 루트 서버 A의 CPU 및 메모리 사용량(평균) 성능분석 결과를 나타낸다.

표 10. 성능분석 결과(CPU and Memory Usage)
Table 10. Performance Analysis Result(CPU and Memory Usage)

	KVM	Proposal
인스턴스 생성 및 실행	66.10%, 67MB	72.46%, 38MB
인스턴스 검증	78.21%, 77MB	83.34%, 42MB
인스턴스 변경	77.13%, 118MB	79.67%, 50MB
평균 합계	73.81%, 87.3MB	78.49%, 43.3MB

CPU 사용량은 운영체제의 기본 패키지인 mpstat 명령어를 이용하여 추출된 CPU idle 10회 평균값을 확인하였다. KVM과 제안프로토콜 모두 웹 서버를 구동한 상태에서 CPU 할당량이 4.68% 차이로 비교적 동일한 수준의 성능으로 나타났다. 그러나 KVM의 경우 가상머신 관리자에 대한 CPU 할당량을 포함되었기 때문에 기본적인 내부 인스턴스에 대한 CPU 할당의 효율성은 KVM이 다소 효율적인 것으로 나타났다.

메모리 사용량은 free 명령어를 이용하여 추출된 10회 평균값을 확인하였다. KVM의 경우 제안 환경과 비교하여 44MB 증가하여 많은 메모리를 사용하는 것으로 나타났다. 이는 Docker 기반환경의 컨테이너 자원 공유방식의 장점으로 제안 환경의 경우 보안 기능을 추가적으로 수행하면서도 훨씬 메모리 효율성이 높은 것으로 분석되었다.

3. 시스템 및 보안성 비교분석

분석 항목 대상은 세션키의 안전성, 가상 자원의 접근권한 분배 등 전체적인 시스템에 대한 안전성을 분석한다. 표 11은 기존 환경과 제안 프로토콜의 시스템 및 보안성 비교분석을 나타낸다.

표 11. 시스템 및 보안성 비교분석
Table 11. Comparison Analysis of System and Security

	기존 보안 구조 방식	Proposal
내부 자원 접근성	가상화 환경의 특징으로 인해 기존 보안 프로토콜 적용 어려움	Docker 환경을 기반으로 에이전트 기반 보안 구조로 설계하여 접근성이 용이
키에 대한 안전성	표준 보안 프로토콜이 없음, 키 교환에 제 3자 가장 공격에 취약	서비스 제공자에서 키 분배 역할 수행, 외부(SSL/TLS), 내부(RSA, ECC)에서 기존 표준 프로토콜 응용
접근권한 분배	KVM 이외에 대부분 플랫폼 제공자 보안 구조에 의존적	서비스 제공자와 플랫폼 제공자를 분리하여 접근권한 분리
호환성 및 확장성	각 다른 하이퍼바이저 구조에 따른 보안 구조 설계가 요구됨	컨테이너 방식의 특징을 이용하여 보안 기능을 웹 컨테이너에 이미지 형태로 할당
VMM의 독립성	플랫폼 제공자의 서버가 해킹될 경우 VMM의 전체 권한 획득 가능	상호인증 과정에서 내부 인스턴스의 보안구조 계층을 실시간으로 검증

기본적으로 가장 중요한 항목은 키에 대한 안전성이다. 제안 프로토콜은 기존 SSL/TLS에서 제 3자 가장 공격을 통한 키 교환 취약성을 해결하기 위해 키를 생성하는 역할에 사용자, 서비스 제공자, 플랫폼 제공자로 분리하고, 키 분배 역할을 서비스 제공자가 수행하여 양방향 안전성을 제공한다. 또한 Docker 환경 기반으로 접근성, 호환성을 준수하고, 접근권한 테이블을 적용하여 VMM의 독립성을 제공한다. 표 12는 제안 프로토콜에서 생성하는 키 체인 구조의 안전성 분석을 나타낸다.

표 12. 키 체인 구조 안전성 분석
Table 12. Security Analysis of Key Chain Structure

체인 방향		설명
상위	세션키	SSL/TLS 2048비트
	마스터키	세션키(256비트 랜덤)
	그룹키	클러스터$^{(2^{1\sim N})}$
하위	서브키	공유된 다중 인스턴스$^{(2^{1\sim N})}$
복잡도 : 공개키2048 × 세션키$^{(GK256)}$ × GS128 × AK128		

초기 상호인증 과정에서의 세션키로부터 마스터키, 그룹키, 서브키 구조로 분리된 키 체인 구조를 적용하였다. 공격자는 키 분석을 위해 키체인에 대한 복잡도 뿐만 아니라 다중 클러스터($2^{1\sim N}$)의 하위에 다중 인스턴스($2^{1\sim N}$)에 대한 키 관계도를 추가적으로 분석해야 하는 어려움을 가진다. 표 13은 제안환경 내부의 다중 인스턴스에 대한 안전성 분석을 나타낸다.

표 13. 다중 인스턴스 안전성 분석
Table 13. Security Analysis of Multiple Instance

	설명
암호화된 인스턴스 안전성	표준 알고리즘으로부터 생성된 RSA 2048비트에 준하는 세션키로부터 생성된 다중 키(마스터키 - 그룹키 - 서브키) 체인 구조의 암호화 안전성 제공
단일, 다중 인스턴스 취득	인스턴스 세션 및 암호문에서 키를 유추하기 위해서는 키 체인관계 분석(각 인스턴스가 존재하는 서버 데이터 흐름)이 요구됨
초기 세션 취득	초기 세션으로부터 키를 유추하기 위해서는 서비스 제공자와 플랫폼 제공자간의 상호인증 파라미터를 추가적으로 요구, 기존 구조는 플랫폼 제공자가 해킹될 경우 서버를 가장한 키 생성 및 교환 가능
에이전트 보안구조 계층 별 공격	보안구조 설계에서 접근 권한인가를 서비스 제공자에서 데이터 테이블로 관리하여 VMM의 독립성을 유지

내부에서 전송되는 각 인스턴스들은 내부 암호화 데이터에 타임스탬프 정보를 포함하여 데이터 변조와 재전송 공격을 방지하고, 키 체인 구조의 적용으로 인한 다중 인스턴스 관계분석의 어려움(각 보안 계층의 위치와 인스턴스의 위치를 파악해야함)을 제공한다. 또한 지속적인 데이터 변경에 대한 서브키 갱신 및 검증은 오래된 세션키의 사용으로 인한 키 노출 문제를 해결한다.

V. 결 론

가상화 보안 기술은 최근 차세대 플랫폼 트렌드를 반영했을 때 다소 기술의 발전이 미비한 실정이다. 제안 프로토콜은 이를 고려하여 기존 가상화 보안 구조의 문제점을 해결하기 위해 컨테이너 방식을 사용하는 Docker 환경을 기반으로 다양한 장점을 극대화하고 가상화 환경의 다중 인스턴스 특징을 이용하여 키 체인 구조적용과

각 계층별 접근권한 분배를 독립적으로 분리함으로써 보안기능을 강화시켰다.

가상 자원 사용자는 일반적으로 보안 기능에 대한 인식이 낮은 경우가 대부분이지만 서비스 제공자는 기본적으로 보안 기능을 포함하는 이미지를 어플리케이션에 할당하는 컨테이너 선택형 서비스 형태로 제공할 수 있다. 개발자의 경우 추가적인 보안 기술에 전문적인 지식이 없어도 Docker 환경에 보안 기능을 수행하는 이미지를 웹 컨테이너에 쉽게 적용할 수 있다. 제안 프로토콜은 기존의 에이전트리스 기반 보안구조와 키 체인 기법을 활용하였지만 기존 가상화 보안 구조를 변경하지 않기 때문에 컨테이너를 기반으로 하는 다양한 가상화 기술에 적용할 수 있어 가상화 보안 기술 활성화에 기여할 수 있을 것으로 기대된다.

향후 연구로는 빅데이터 환경에서의 다중 인스턴스에 대한 효율성 분석에 따른 보안 구조 설계이다. 제안 보안 구조에서는 대규모 네트워크 환경을 고려하여 클러스터트리 토폴로지 구조를 적용하였지만 비교적 작은 규모의 서버를 통해 테스트되었기 때문에 정확한 성능 결과 데이터를 산출하기 위해서는 대용량의 실시간성 데이터 스트림을 대상으로 성능이 테스트 되어야 할 필요가 있다.

References

[1] Lee Jung, “Google I / O 2016 Showed the Future Strategy On-device with a Variety of Public Services based on Artificial Intelligence Platform”, Eugene Investiment & Securities, Global IT Monthy No. 25, 2016.

[2] Kim Deulpeul, “2015 World ICT Industry Main Issues and Forecast”, Telecommunications Union, iNSIGHT Global Trend, 2015.

[3] Choi Dohyeon, “A Multi Session User Authentication Methods for Secure Virtualization Layer in the Big Data Environments”, Soongsil University, 2016.

[4] Jo Yujin, Lee Jaeduk, Lee Minwoo, “Domestic and Foreign Policy, and Cloud Industry Trends”, Telecommunications Technology Association, Special Report Special Theme Cloud Computing, 2016.

[5] Kim Hwanguk, Cho Hwa, Sin Youngsang, “Secure Cloud Environment for Virtualization Security Issues and Technology Trends”, Korea Information and Communications Magzine Vol. 32, No. 10, pp. 49-57, 2015.

[6] Kang Jangmook, Song Youjin, “A Study on Structural Holes of Privacy Protection for Life Logging Service as analyzing/processing of Big-Data”, The Journal of Institute of Internet, Broadcasting and Communication(JIIBC), Vol. 14 No. 1, pp. 189-193, 2014.

[7] Im Seokjin, Hwang heejoung, “Design and Development of Framework for Health Data Relay based on OAuth2 in Cloud Environment”, The Journal of Institute of Internet, Broadcasting and Communication, Vol. 15, No. 4, pp. 153-159, 2015.

[8] GARFINKEL, M Rosenblum, “A Virtual Machine Introspection Based Architecture for Intrusion Detection”, In: NDSS, Vol. 3, pp. 191-206, 2003.

[9] TrendMicro, “Agentless Security for VMware Virtual Data Centers and Cloud”, Trend Micro Security, 2012.

[10] Jung Hyunjun, “Trends and Major Issues of Virtualization Technology(Ⅱ)”, Korea Information Society Development Insitute - Insitute of Convergence Science Vol. 25, No. 5, Serial No. 55, 2013.

[11] Sin Youngsang, “Hypervisor-based Security for Cloud Computing Enviroments”, Korea Internet & Security Agency - Korea Internet Conference 2012, 2012.

[12] Um Jungho, Kim Taehong, Lee Seungwoo, Jung Changwoo, Jung Hanmin, “Next-Generation Distributed System in Real Time Big Data Trends - Focusing on Spark and Storm”, Insitute for Information & Communications Tecnology Promotion Weekly Tecnology Trends, 2014.

[13] Sin Junhee, “Introduction to Open Source Technology Infrastructure for the Cloud Environment”, RockPLACE OpenSourcePlace,

2014.

[14] Chung Haejin, Nah Yunmook, “Effects of Hypervisor on Distributed Big Data Processing in Virtualizated Cluster Environment”, KIISE Transactions on Computing Practices, Vol. 22, No. 2, pp. 89-94, 2016.

[15] Thorsten von Eicken, “Docker vs. VMs? Combining Both for Cloud Portability Nirvana”, RightScale, 2014.

저자 소개

최 도 현(정회원)

- 2008년 2월 : 동서울대학교 컴퓨터소프트웨어학과 졸업
- 2010년 8월 : 숭실대학교 컴퓨터학과 석사
- 2016년 3월 : 숭실대학교 컴퓨터학과 박사

<주관심분야 : Mobile, Network Security, PKI, Virtualization>

- E-mail : cdhgod0@ssu.ac.kr

김 상 근(정회원)

- 1996년 : 중앙대학교 컴퓨터공학과 (공학박사)
- 2003년 ~ 2004년 : Sydney University, 방문교수
- 1996년 ~ 현재 : 성결대학교 컴퓨터공학부, 교수

<주관심분야 : PKI, 정보보안, 소프트웨어공학>

- E-mail : sgkim@sungkyul.ac.kr