분산 암호화를 이용한 웹 어플리케이션 보안

Web Application Security using Distributed Encipherment

허진경

호원대학교 사이버수사경찰학부

Jin-Kyoung Heo(heojk@howon.ac.kr)

요약

인터넷 사용자의 증가와 인터넷 쇼핑몰의 대중화로 인해 사용자가 요구하는 서비스의 양이 증가하고 있다. 이로 인해 웹 어플리케이션 시스템에는 서버의 과부하로 인한 성능 저하뿐만 아니라, 보안과 관련하여 많은 문제들이 발생한다. 또한 웹 어플리케이션 시스템에서는 클라이언트 요청에 의한 암호화가 필요하고 사용자를 인증하기 위한 공개키를 제공한다. 이때 발생하는 보안상 취약점을 해결하기 위한 방법으로 전달되는 메시지의 암호화와 공개키의 관리가 필요하다. 본 논문은 웹 어플리케이션 시스템에서 기밀성 및 인증을 제공함과 동시에, 다수의 클라이언트 접속 시 데이터의 암호화 처리로 인한 서버의 병목현상으로 인한 성능저하를 방지하고 서비스 질을 향상시키기 위한 방법으로 분산 암호화 시스템을 제안한다. 이러한 분산 암호화 시스템을 구현하기 위하여 자바의 객체 활성화 기술을 적용하였으며, 일부 분산서버의 다운타임에도 지속적인 서비스를 제공할 수 있도록 하였다.

■ 중심어 : | 암호화 | 웹 어플리케이션 보안 | 분산 | 공개키 | 알고리즘 |

Abstract

Quantity of encrypted data that transmitted through the network are increasing by development of encipherment technology. We have many problems; it is caused by technical development and service increase of user requests. It is necessary that create a many encryption key in one web application system. As a result, service quality comes to be low because of increased network traffic and system overload. There must be a system. That should be improved in secure service quality to process data. This paper describes a new approach for design and implementation of distributed encryption key processing for web application system. In this paper, it is based on distributed encipherment key, for the purpose of confidentially, integrity and authentication. It can prevent system degradation from server's data bottleneck and can improve service quality. For distributed encipherment system, we use java object activation technology. It can service while some distributed server are fail.

■ keyword : | Encipherment | Web Application Security | Distributed | Public Key | Algorithm |

I. 서 론

안전한 데이터 전송을 위해서는 데이터가 전달 시 노출되더라도 그 내용을 알지 못하도록 하는 방법이 필요하다. 이러한 방법으로 암호화 기법이 사용된다. 평문을 그대로 전송하는 것은 내용이 노출되기 때문에 아주

접수번호 : #080116-002
접수일자 : 2008년 01월 16일
심사완료일 : 2008년 03월 04일
교신저자 : 허진경, e-mail : heojk@howon.ac.kr

위험하다. 그러므로 암호화된 문서를 전송하면 보다 안전하다. 송신자는 평문을 암호화키로 암호문을 만들고, 이 암호문을 전달하면 수신자는 암호문을 복호화 하여 평문을 만들어 내용을 확인하게 된다. 네트워크를 통해 지나다니게 되는 것은 평문이 아닌 암호문이다. 해커가 암호문을 잡을 수 있지만 해커는 암호화키가 없으므로 암호문을 가로채도 복호화 시킬 수 없다[1][2]. 그러나 데이터들을 암호화시키기 위해서는 암호화하기 위한 처리 시간이 소요된다. 특히 웹 어플리케이션을 통해 클라이언트에 전달되는 데이터들은 정보에 태그가 포함되어 있다. 이로 인해 서버에서 암호화 처리에 필요한 시간이 더욱 늘어나게 되고, 이는 클라이언트의 응답시간과 직결된다[3]. 본 논문에서는 대량의 데이터를 처리해야 하는 웹 어플리케이션 시스템에서 암호화 처리를 위해 객체 활성화를 사용한 분산 시스템을 적용하였다. 객체 활성화는 프로세스를 메모리가 아닌 보조저장장치에 저장한 후 사용자의 요구 발생시 메모리에 로드하여 실행시켜 주는 기술이다. 객체 활성화 기술을 암호화 분산처리 시스템에 적용하여, 일부 서버의 다운타임 또는 병목현상으로 인해 서비스 응답시간이 지연될 경우, 다른 서버에 의해 서비스를 제공할 수 있도록 하였다. Ⅱ장에서는 기존의 암호와 알고리즘과 암호화 방법, 그리고 암호화 시스템에서의 병목 현상에 대하여 논하였고, Ⅲ 장에서는 객체 활성화를 이용한 분산 암호화 시스템에 대하여 언급하였다. Ⅳ장에서는 제안한 암호화 시스템을 구현하여 암호화 시스템의 안정성 및 효율성을 검증하고자 하였다.

Ⅱ. 암호화

정보 보안을 위협하는 요소에는 도청(盜聽), 변조(變造), 위조(僞造), 부인(否認) 4가지가 있을 수 있다. 이들 각각의 위협 요소들과 관련된 보안 요소는 내용이 노출되지 않는 기밀성(Confidentially), 내용이 변경되지 않는 무결성(Integrity), 정당한 사용자임을 확인 하는 인증(Authentication), 그리고 사용 증거를 확보하는 부인방지(Non-repudiation)가 있다. 암호화 알고리즘은 크게 대칭키 알고리즘, 해쉬 알고리즘, 공개키 알고리즘이 있는데, 대칭키 알고리즘은 기밀성에는 사용되고, 해쉬 알고리즘은 무결성에 사용된다. 그리고 공개키 알고리즘은 기밀성, 인증, 부인방지에 사용된다[4].

1. 대칭키 암호 방식

대칭키 암호(Symmetric Cryptographic Technique) 알고리즘은 암호화 할 때 사용하는 키를 복호화 할 때 반대로 사용하면 복호화 되는 알고리즘이다. 메시지를 전송하는 송신측에서 사용하는 암호화키와 수신측에서 사용하는 복호화 키가 대칭을 이룬다고 해서 대칭키 알고리즘이라고 하며 비밀키(Secret Key) 방식이라고도 한다. 대칭키 알고리즘에는 DES, Blowfish, 3DES, ARS, RC4, SEED등이 있다[4][5].

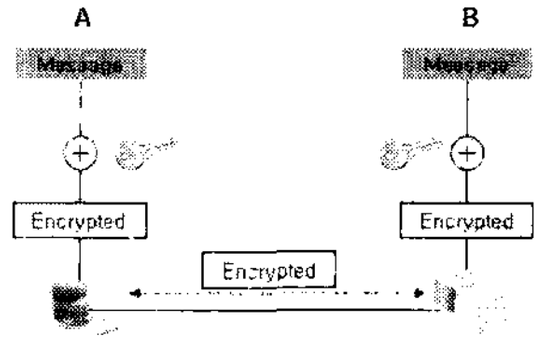


그림 1. 대칭키 암호 방식

[그림 1]은 대칭키 암호방식의 암호화 및 복호화를 보여주고 있다. [그림 1]에서 보여주고 있는 것처럼 대칭키 암호 방식은 메시지를 보내는 송신측과 받는 수신측이 같은 키를 사용하여 암호화/복호화를 수행 한다. 대칭키 알고리즘은 다양한 알고리즘 개발이 용이하며, 안전성 검장방법이 비교적 정형화 되어 있다. 그리고 암호화 및 복호화 속도가 매우 빠른 장점이 있다. 그러나 키 관리 및 키 분배의 어려움이 있다. 그 이유는 Entry 쌍의 개수만큼의 키가 필요한데, N명의 사용자가 대화하기 위해서는 n(n-1)/2 쌍의 비밀키가 필요하다. 예를 들면 1천명의 사용자가 대화하기 위해서는 50만 여개의 키가 필요하게 된다. 뿐만 아니라 디지털 서명, 부인방지 등의 기능이 불가능하게 되고, 어느 한쪽이 키를 분실하면 시스템에 위험을 초래하게 되는 키 공유의 문제가 발생하는 등의 단점이 존재하게 된다. 대칭키 알고리즘은 주로 대용량 데이터 암호화에 사용한다.

2. 공개키 암호 방식

공개키 암호(Public Key Cryptosystem) 알고리즘이 나타나기 전에는 대칭키 암호 알고리즘식이 사용되었다. 그러나 이러한 방식은 송신자의 비밀키를 사용하여 메시지를 암호화한 후 수신자에게 전달하는 방식으로, 수신자가 암호화된 메시지를 해독하기 위해서는 송신자의 복호화 키를 알고 있어야 한다. 따라서 이 방식은 복호화 키를 아는 사람은 누구라도 암호문을 복호화 할 수 있으므로 복호화 키를 수신자에게 전달할 때에는 각별한 주의가 필요하다.

이러한 단점을 극복한 방식이 바로 공개키 암호 알고리즘인데 이 방식은 복호화 키를 공유해야 하는 어려움을 해결하였다. 공개키 암호 알고리즘에서는 암호화키와 복호화와 키 중 암호화키를 외부에 공개한다. 그러나 이 공개된 암호화키로 복호화 키를 알아낼 수는 없다. 따라서 공개키 암호방식을 이용하여 특정인에게 비밀 메시지를 보내고자 하는 사람은 공개된 특정인의 암호화키를 이용하여 메시지를 암호화해서 보낸다. 그러면 그 특정인은 자신만이 아는 복호화 키로 메시지를 복호화 할 수 있다. 이런 이유 때문에 공개키 암호방식에서는 암호화키를 공개키라고 부른다. 다시 말해서 공개키 암호방식은 암호화키와 복호화 키가 서로 다른 암호 방식을 말한다.

공개키 암호 알고리즘은 암호화 할 때 사용하는 키와 복호화 할 때 사용하는 키가 대칭을 이루지 않는다. 따라서 비대칭 키 방식이라고도 부른다. 공개키 암호 알고리즘은 비대칭이므로 1쌍, 즉 두 개의 키가 필요하다. 두 개의 키는 각각 공개키(Public Key)와 개인키(Private Key)로 부른다. 공개키는 인증기관을 통해 공개하는 키 이며, 비공개로 본인만이 소유하게 된다.

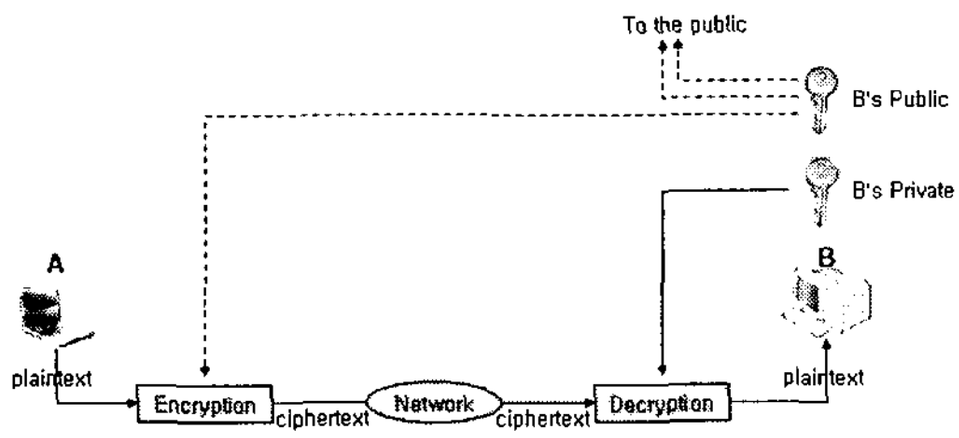


그림 2. 공개키 암호 방식

[그림 2]는 공개키 암호방식을 보여주고 있다. [그림 2]에서는 송신측에서 암호화 할 때의 키와 수신측에서 암호화된 데이터를 복호화 하는데 사용하는 키가 서로 다르게 되어 있다. 공개키 암호방식의 경우는 키의 크기가 크고, 또한 각 키는 특수한 성질을 요구하기 때문에 비밀키 암호화 방식의 키와 같이 사용자가 직접 원하는 키를 만들지는 못한다. 많이 사용되는 공개키 암호방식의 키 크기는 높은 안전성을 갖도록 하기 위하여 현재 사용되는 공개키 암호 방식 보안 제품은 중 RSA는 1,024bits의 키를 사용하고 있다

공개키 암호 알고리즘은 기밀성과 인증을 제공하며, 부인불가에 사용될 수 있다. 기밀성은 공개키는 인증기관을 통해 알 수 있으므로, 수신자의 공개키를 이용하여 메시지를 암호화 하면 수신자 외의 누구도 메시지를 복호화 할 수 없다. 즉, 사용자 A가 사용자 B에게 평문 메시지를 사용자 B의 공개키를 사용하여 암호화 하여 보내고, 사용자 B는 자신의 비밀키로 암호화된 메시지를 복호화 하여 평문 메시지를 얻는다. 사용자 B가 복호화에 사용되는 비밀키는 자신만이 가지고 있으므로 자신 외에는 아무도 자신의 비밀키로 암호화 된 메시지를 복호화 하여 볼 수 없다. 따라서 사용자 B만 메시지를 볼 수 있는 기밀성을 제공한다. 인증은 송신자가 자신의 개인키로 메시지를 암호화 하면 수신자가 송신자의 공개키로 복호화 하여 송신자에 대한 확인이 가능하다. 즉, 사용자 A가 사용자 B에게 평문 메시지를 자신의 비밀키를 사용 암호화 하여 보내고, 사용자 B는 사용자 A의 공개키로 암호화된 메시지를 복호화 하여 평문 메시지를 얻는다. 사용자 A가 암호화에 사용하는 비밀키는 자신만이 가지고 있으므로 자신 외에는 아무도 메시지를 암호화 할 수 없다. 따라서 암호화된 메시지는 사용자 A로부터 왔다는 인증을 제공한다. 이 경우에는 수신자를 제외한 제3자(해커)도 복호화 할 수 있다.

공개키 방식의 특징은 서로 연관된 키 쌍(개인키/공개키)이 필요하고, 하나의 키로 암호화 한 경과는 반드시 쌍이 되는 키로만 복호화가 가능하다. 그리고 구조상 복잡한 수학연산이 필요하고, 안전성이 이러한 수학적 이론에 근거한다. 또한 N명의 사용자가 있으면 N개의 공개/개인키 쌍으로 충분하기 때문에 키 관리 문제

를 해결 할 수 있다. 또한 서명기능, 기밀성 등의 기능이 가능하다. 그러나 공개키 방식의 단점으로는 동일한 양의 데이터를 암호화 시 대칭키 암호화에 비해 매우 느린데, DES에 비하여 하드웨어 구현에서 1000배가량 더 시간이 소요된다.

공개키 방식 알고리즘으로는 RSA(Rivest Shamir Adleman)와 ECC (Elliptic Curve Cryptography: 타원곡선 암호)가 있다.

3. 공개키와 대칭키 조합

공개키 방식은 기밀성 및 인증이 가능하므로 공개키 방식만 사용해도 된다. 그러나 공개키 방식은 암호 알고리즘이 복잡하여 연산속도가 느려지므로, 대용량 데이터를 암호화 하는 데는 부적절하다. 반면, 대칭키 방식은 빠른 속도로 데이터를 암호화/복호화 수행한다. 따라서 일반적으로 메시지는 대칭키 방식으로 암호화 하고, 비밀키만 공개키 방식으로 암호화 하게 되면, 효율적인 방식으로 암호화/복호화를 할 수 있다[11][12]. 다음 [그림 3]은 A와 B사이에서 메시지와 비밀키의 암호화를 보여주고 있다.

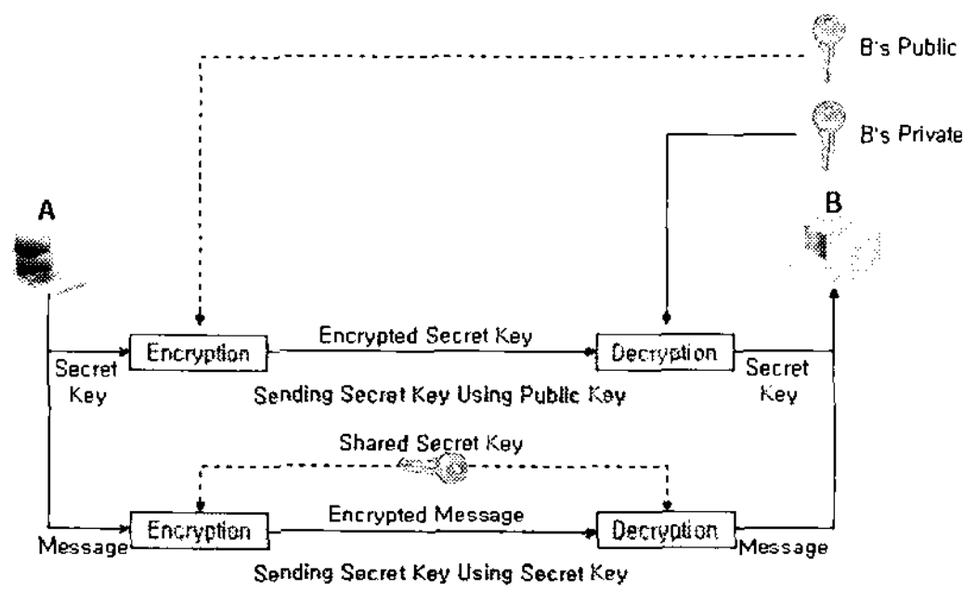


그림 3. 대칭키와 공개키 조합

[그림 3]에서 송신자는 수신자에게 보낼 메시지를 대칭키 암호 방식을 이용하여 암호화 한다. 이때 수신자가 메시지를 복호화 하기 위해 필요로 하는 키는 공개키 암호 방식으로 암호화 한 후 수신자에게 전송된다. 수신자는 자신의 개인키를 이용해 송신측에서 전송되어진 비밀키를 복호화 한다. 그리고 비밀키를 이용해 메시지를 복호화 하는 방법을 사용하게 된다. 이러한 방법을 사용하면 웹 어플리케이션에서 전송되어야 할 대량의 데이터는 대칭키 암호 방법을 통해 보다 빠르게 암호화 할 수 있고, 그 외에 비밀키 또는 기밀성과 인증이 요구되는 데이터는 공개키를 사용하여 암호화 할 수 있다.

이러한 암호화 시스템들의 단점은 암호화를 위한 데이터가 한 곳에 집중이 된다는 것이다. 이로 인해 데이터 병목현상이 발생하고, 이는 서비스 지연으로 이어진다.

III. 분산 암호화 시스템

웹 어플리케이션은 대량의 데이터 처리해야 한다. 이로 인해, 암호화 처리 할 때에 더 많은 데이터 처리가 요구된다. 이로 인해 시스템에서는 처리 속도가 느려지는 단점이 존재하게 되고 이는 곧바로 서비스 질이 떨어지는 결과를 초래한다[3]. 이를 해결하기 위한 기본 개념으로 공개키 방식과 대칭키 방식을 함께 사용한다. 또한 암호화 처리를 위한 시스템을 분산 서버에 위임하여 처리하는 방법을 사용한다. 본 논문에서는 분산 처리에 객체 활성화 기술을 적용하여 일부 분산서버의 다운타임 시에도 지속적인 서비스를 제공해 주도록 하였다.

1. 분산 서버 제어

암호화 하기위한 분산 서버를 제어하기 위해서 레지스트리 풀(Registry Pool)을 사용하였다. 레지스트리 풀은 주 서버에 존재하여 각각의 분산 서버들의 상태와 분산 서버들의 레지스트리 레퍼런스를 갖고 있다. 레지스트리 풀은 싱글톤 디자인 패턴(Singleton Design Pattern)을 적용하여 주 서버에는 어떠한 경우라도 하나의 풀(Pool) 인스턴스가 존재하게 된다. 다음 [그림 4]에서는 미들 서버에서 분산 서버들을 제어하기 위해 주 서버에서 레지스트리 풀을 사용하는 것을 나타내고 있다.

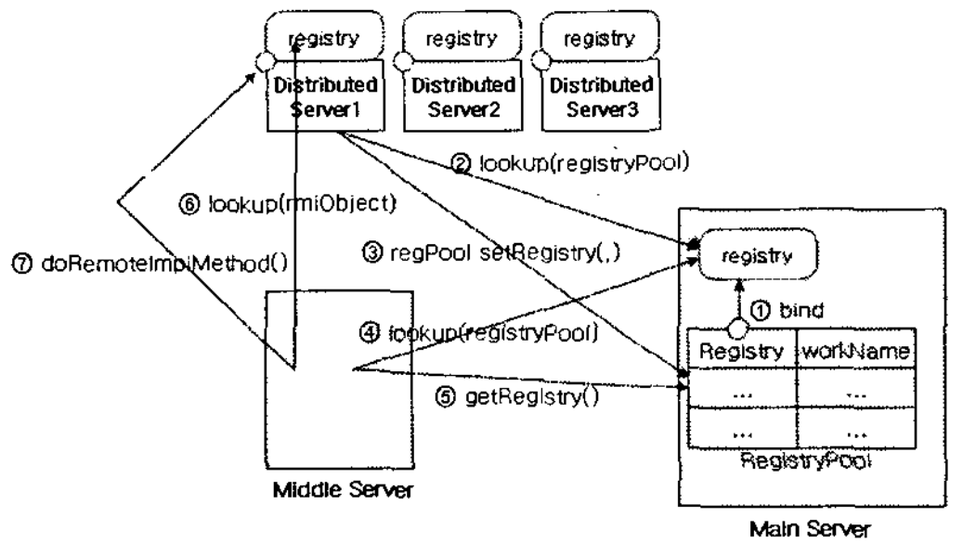


그림 4. 분산 서버 제어

주 서버에 레지스트리 풀을 등록하고, 미들 서버 또는 분산 서버에서 이를 참조하는 과정을 보면 다음과 같다.

a. 주 서버에서는 레지스트리 풀의 인스턴스를 생성하여, 주 서버의 레지스트리에 풀 객체를 등록한다. 주 서버의 레지스트리 풀에는 분산 서버의 레지스트리와 분산 서버가 전담하게 되는 작업 내용이 등록된다.

```
Registry reg = LocateRegistry.createRegistry(port);
RegistryPool regPool = RegistryPool.getInstance();
Naming.rebind("registryPool", (Remote)regPool);
```

그림 5. 주 서버의 레지스트리 풀 사용

b. 분산 서버들은 각각 자신의 서비스 객체들을 자신의 레지스트리에 등록한 후, 서버의 레지스트리 풀에는 분산 서버의 레지스트리 객체와 전담하는 작업 내용을 등록한다. 이때 분산 서버에서 활성화 시스템이 작동 중 인지의 여부를 함께 등록한다.

```
Registry reg2 = LocateRegistry.createRegistry(port);
Naming.rebind("processName", processObjectRef);

Naming.lookup("rmi://server/registryPool");
RegistryPool regPool2 = RegistryPool.getInstance();
regPool2.setRegistry(reg, "workName|true");
```

그림 6. 미들서버의 레지스트리 풀 사용

c. 미들 서버는 클라이언트의 요구에 따라서 주 서버의 원격 레지스트리를 통해서 분산 서버에서 접근하여 사용하게 된다. 예외가 발생하여 분산 서버의 고유의 기능을 수행할 수 없으면 미들 서버는 다시 활성화 가능한 분산 서버를 찾게 된다. 활성화 가능한 분산 서버는 분산 서버가 구동된 후 서버의 레지스트리 풀에 등록된 레지스트리 객체들 중에서 활성화 가능한지의 여부를 확인하여 찾게 된다.

```
Naming.lookup("rmi://server/registryPool");          //[그림 4]의 ④
RegistryPool regPool = RegistryPool.getInstance();

try{
    Registry registry = regPool.getRegistry(workname);  //[그림 4]의 ⑤
    Remote remote = (Remote) registry.lookup(rmiObject);//[그림 4]의 ⑥
    remote.doRemoteImplMethod(data);                    //[그림 4]의 ⑦
}catch(Exception e1) {
    //클라이언트의 요구를 처리할 수 있는 분산 서버가 존재하지 않음
    //객체 활성화 시스템 구동
    try{
        Registry activationRegistry = regPool.getAvailableActivationRegistry();
        Remote remote = (Remote) activationRegistry.lookup(rmiObject);
        remote.doActivationImplMethod(data);
    }catch(Exception e2) {
        //클라이언트의 요구를 처리할 수 있는 활성화서버가 존재하지 않음
    }
}
```

그림 7. 분산 서버 제어 구현 알고리즘

2. 분산서버에 의한 암호화

분산 서버에 의한 메시지의 암호화 처리는 대칭형 알고리즘 사용하여 빠른 속도로 데이터를 암호화/복호화를 수행시킬 수 있다. 분산 서버는 신뢰성 있는 미들웨어에 암호화 된 데이터와 메시지를 복호화 하기 위한 비밀키를 공개키 방식으로 암호화 하여 전송한다.

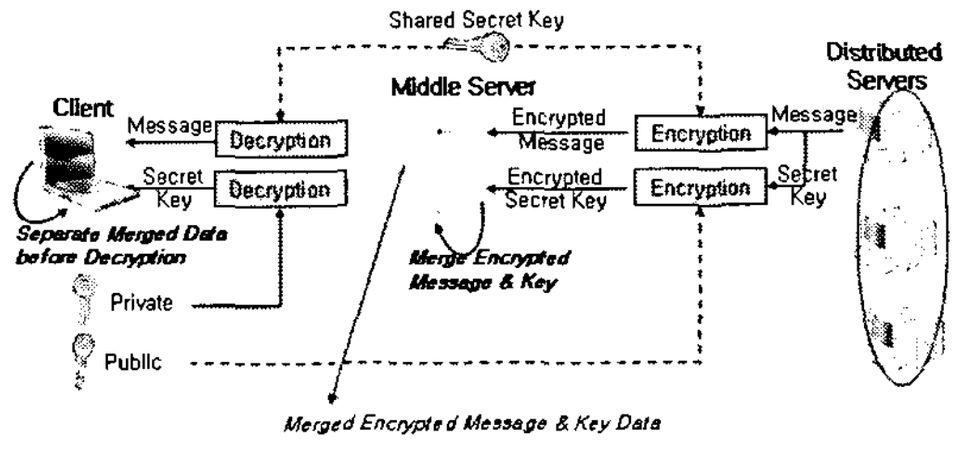


그림 8. 암호화 분산처리

미들서버는 암호화 된 데이터를 조합한다. 미들서버에서 암호화된 데이터를 조합할 때 다시 조합 코드를 공개키 방식으로 암호화한다. 암호화된 후 조합된 메시지에는 분산서버에 의하여 처리된 메시지들과 이 메시지들을 복호화 하기 위한 비밀키가 포함되어 있다. 미들서버는 조합된 메시지에 추가로 메시지들을 풀기 위한 해제키를 공개키 방식으로 암호화 하여 전송한다. 클라이언트는 메시지를 이해하기 위해서는 1차로 공개키 방식에 의하여 조합코드를 이해한다. 조합 코드에

의해 미들서버에 의해 조합된 메시지를 처리하여 분산 서버에서 생성된 암호화 코드와 암호화 코드를 해석하기 위한 비밀키를 얻는다. 이때 암호화 코드를 해석하기 위한 비밀키는 공개키 방식으로 암호화 되었으므로, 사용자 인증 및 기밀성 등을 제공 할 수 있다.

IV. 실험 및 결과

본 논문에서는 암호화 및 암호화키 생성을 분산 시스템에 적용하였다. 메시지의 암호화를 위해서 대칭키 암호 알고리즘으로는 DES 알고리즘을 사용하여 구현하였으며, 암호화 된 메시지들을 복호화 하기 위한 비밀키의 암호화는 공개키 암호와 알고리즘인 RSA방법을 사용하여 구현하였다. 실험에 사용된 언어는 JAVA JDK5.0을 이용하였으며, 분산 처리를 위해서는 자바의 RMI기술을 사용하였다.

암호화에 사용된 데이터의 크기는 100Kbyte 임의의 문자열을 사용하였으며, 접속하는 클라이언트의 수를 증가시키는 방법으로 단일 시스템에서의 암호화 처리와 분산 시스템에서의 암호화 처리를 비교 하였다. 클라이언트와 서버가 존재하는 네트워크 환경은 100Mbps를 지원하고 있다.

1. 활성화 분배

[표 1]은 3대의 분산 서버와 각각의 서버에서 전담할 3개의 보안처리 작업이 있을 경우에 최대 활성화시켜야 할 내용을 나타낸 것이다. 서버는 서버 1부터 서버 3까지 번호로 나타내었고, 각각 작업은 분산 서버 A부터 C까지 알파벳으로 나타내었다. 처리작업 A는 암호화 핸들링, B는 인증처리, C는 디지털 서명 핸들링 작업을 진행한다고 가정한다. 각각의 서버는 활성화가 요청되기 전까지는 각각의 처리를 위한 기능만 메모리에 적재되어 있고, 어느 한 서버의 중단 시에도 전체적인 시스템의 중단 없이 지속적인 서비스를 제공해 주기 위해서 다른 서버들의 작업들이 활성화되어 있다.

표 1. 암호화 분산처리 배분

처리 서버	RMI	Activation
서버 1	Cipherment	B, C, D
서버 2	Authentication	A, C, D
서버 3	Digital Signature	A, B, D

활성화를 사용하지 않고 분산 처리 시스템에 있어서 분단 서버의 오류에 대비하기 위한 자원을 할당할 경우에는 N^2(N: 처리 수)의 자원을 요구하게 되지만, 활성화 시스템을 사용할 경우에는 σN^2(N: 처리 수, $\sigma < 1$: 활성화 요구비)의 자원을 요구하게 된다. 이는 분산 처리 시스템에서 서비스 실패에 대비하기 위한 미러링 시스템을 구축할 경우에도 보다 적은 자원 자원도 가능함 보인다[3].

2. 암호화 처리 소요시간

암호화 처리에 사용되는 소요시간 계산을 단일 시스템에서 처리할 때와 분산 시스템에서 처리할 때를 비교 실험하였다. 실험은 각 클라이언트 수에서 10회 응답 속도를 체크하여 평균값으로 처리하였다.

단일 시스템에서 암호화 처리 실험을 위한 시스템 사양으로 펜티엄 D 3.40Ghz CPU, 2G RAM PC 1대를 사용하였다.

분산 시스템에서 암호화 처리 실험을 위한 시스템 사양으로 미들서버 1대(펜티엄 D 3.40Ghz CPU, 2G Ram PC)와 분산서버 3대(펜티엄 D 3.40Ghz CPU, 2G Ram PC)를 이용하였다.

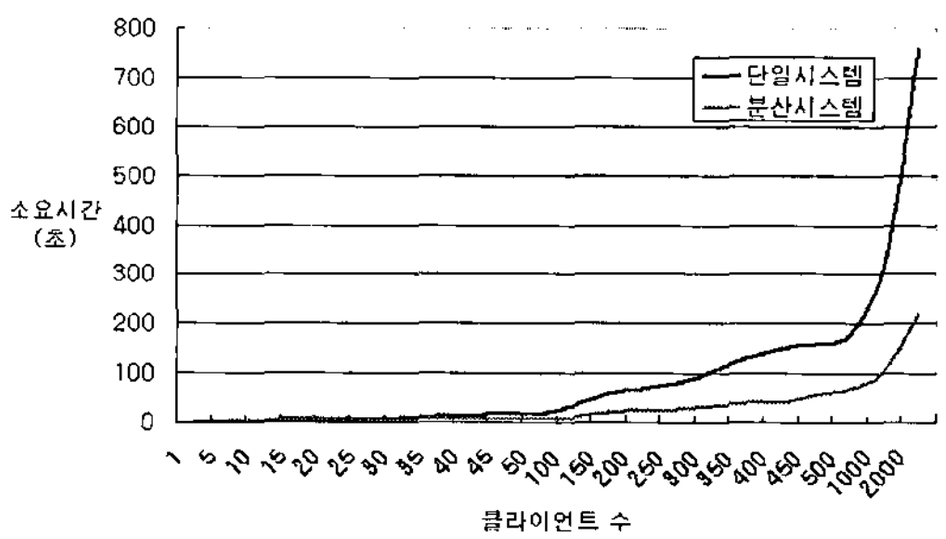


그림 9.클라이언트 수에 따른 응답시간

[그림 9]에서는 단일 시스템에서 암호화 처리는 암호화를 요구하는 클라이언트 수가 많아질수록 서버의 응답속도는 현저하게 떨어지는 것을 볼 수 있다. 그러나 분산 시스템에서는 스트레스테스트를 하기 위해 클라이언트 수를 증가시켰을 때 일정 수준까지 서버의 응답 속도를 유지하는 것을 볼 수 있다.

V. 결론

웹 어플리케이션 시스템에서 암호화된 데이터를 전송하면서 복호화 하기 위한 복호화 키를 공개키 방식으로 암호화하여 전송한다. 이렇게 함으로써 데이터의 기밀성 및 인증작업을 수행할 수 있다. 그러나 암호화 처리에 필요한 데이터가 증가할수록 그와 비례하여 서버의 서비스 질은 떨어지게 된다. 본 논문에서는 암호화 데이터가 한곳에 집중되는 웹 어플리케이션 시스템에서 보다 효율적이고 안정적인 서비스를 제공하기 위하여 객체 활성화를 이용한 분산 시스템을 제안하였다. 암호화 처리를 수행하는 분산서버에 객체 활성화 기능을 추가함으로써 각각의 분산서버가 담당하는 프로세스가 아는 다른 서버의 프로세스를 비활성화 시킨 후, 특정 분산서버의 다운타임 또는 서비스 지연 시 해당 분산서버의 프로세스를 활성화 시켜 서비스를 제공해 줄 수 있게 하였다. 또한 미들서버에서 분산서버의 결과를 조합하는 작업과 조합된 메시지들을 해제하기 위한 비밀키를 암호화 전송함으로써 데이터 무결성 처리를 하였다. 차후 SSL통신 및 PKI를 적용하여 메시지를 보다 안전하게 전송하기 위한 시스템을 구현하고자 한다.

참 고 문 헌

[1] 한국전산원, "웹 환경 구축 및 운영을 위한 보안 기술 연구", NCA III-RER-97052, 1997(12).

[2] 서영규, *웹 프락시를 이용한 웹 입력 오류 공격 탐지 및 차단 연구*, 동국대학교 국제정보대학원 석사학위논문, 2004.

[3] 허진경, *멀티미디어 객체 활성화를 이용한 분산 처리 시스템*, 조선대학교 박사학위논문, 2004.

[4] N. D. Cho, E. S. L, and H. G. Park, "Security Intelligence: Web Contents Security System for Semantic Web," KES, LNCS, ISBN 978-3-540-46537-9, Vol.4252, pp.819-828, 2006.

[5] 한국썬교육센터, "웹 어플리케이션 보안", 2007.

[6] 황순일, 김광진, "웹 해킹 패턴과 대응", 사이텍미디어, 2005.

[7] F. Roger, 유해영, "웹 어플리케이션 개발 방법론", 이한출판사, 2002.

[8] S. Mike, McGraw-Hill Hacknotes, "Web Security Portable Reference," Companies Inc, 2003.

[9] 정보통신부 한국정보보호진흥원, "웹 어플리케이션 보안 템플릿", 2006.

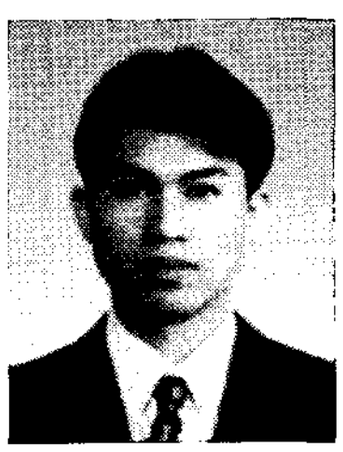
[10] 정보통신부 한국정보보호진흥원, "홈페이지 개발 보안 가이드", 2005.

[11] 과학기술부, "새로운 방식의 공개열쇠 암호의 제작과 기존 방식의 공개열쇠 암호의 연구", 2002.

[12] 정보통신부 한국 정보보호센터, "공개키 암호알고리즘 개발에 관한 연구", 최종연구개발결과보고서, 1999.

저 자 소 개

허 진 경(Jin-Kyoung Heo) 정회원



- 1998년 2월 : 호원대학교 전자계산학과(이학사)
- 2000년 2월 : 조선대학교 전산통계학과(이학석사)
- 2004년 2월 : 조선대학교 전산통계학과(이학박사)
- 2006년 4월 ~ 현재 : 호원대학교 사이버수사경찰학부 연구교수

<관심분야> : 분산처리, 웹어플리케이션보안