情報保護學會論文誌
第19卷 第3號, 2009. 6

# 금융 보안 서버의 개인키 유출 사고에 안전한 키 교환 프로토콜

김 선 종,[1†] 권 정 옥[2‡]
[1]고려대학교 정보경영공학전문대학원, [2]삼성SDS 정보보안그룹

# Secure Key Exchange Protocols against Leakage of Long-term Private Keys for Financial Security Servers

Seon Jong Kim,[1†] Jeong Ok Kwon[2‡]
[1]Graduate School of Information Management and Security CIST, Korea University,
[2]Information Security Group, Samsung SDS

## 요 약

세계적으로 통용되고 있는 키 교환 프로토콜은 TLS/SSL 등의 공개된 암호 통신프로토콜인 반면에 국내 금융권에서는 공인인증과 더불어 금융권에 적합한 공개키 기반 구조(PKI: Public Key Infrastructure)를 이용한 키 교환 프로토콜을 민간 주도로 개발하여 사용하고 있다. 하지만 금융권에서 사용하고 있는 키 교환 프로토콜은 클라이언트 위장공격(client impersonation attack)과 기지 키 공격(known-key attack)에 취약하며, 전방향 안전성(forward secrecy)을 제공하지 않는다. 특히, 암호문과 서버 측 개인키(예: RSA 개인키)만 있으면 쉽게 과거의 세션키(session-key)를 알아내 암호화된 메시지를 복호화 할 수 있기 때문에, 만약 내부 관리 등의 문제로 인해 금융보안 서버의 개인키 유출 시 막대한 개인정보와 금융정보가 노출될 우려가 있다. 본 논문에서는 금융권에 사용 중인 암호 통신 프로토콜의 취약점을 분석하고, 국내 환경에 적합하도록 프로토콜 교체 비용을 최소화하면서 클라이언트 위장공격과 세션키 노출 및 개인키 유출 사고에도 안전한 두 개의 키 교환 프로토콜을 제안한다. 또한 제안하는 두 번째 프로토콜이 HDH(Hash Diffie-Hellman) 문제가 어렵다는 가정 하에 증명 가능한 전방향 안전성을 제공함을 보인다.

## ABSTRACT

The world's widely used key exchange protocols are open cryptographic communication protocols, such as TLS/SSL, whereas in the financial field in Korea, key exchange protocols developed by industrial classification group have been used that are based on PKI(Public Key Infrastructure) which is suitable for the financial environments of Korea. However, the key exchange protocols are not only vulnerable to client impersonation attacks and known-key attacks, but also do not provide forward secrecy. Especially, an attacker with the private keys of the financial security server can easily get an old session-key that can decrypt the encrypted messages between the clients and the server. The exposure of the server's private keys by internal management problems, etc, results in a huge problem, such as exposure of a lot of private information and financial information of clients. In this paper, we analyze the weaknesses of the cryptographic communication protocols in use in Korea. We then propose two key exchange protocols which reduce the replacement cost of protocols and are also secure against client impersonation attacks and session-key and private key reveal attacks. The forward secrecy of the second protocol is reduced to the HDH(Hash Diffie-Hellman) problem.

**Keywords:** Key exchange, Impersonation attack, Forward secrecy, Known-key secrecy, Key secrecy, HSM

접수일(2009년 1월 12일), 수정일(2009년 4월 24일),
게재확정일(2009년 4월 24일)

† 주저자, seonjong.kim@initech.com
‡ 교신저자, jeongok.kwon@samsung.com

## I. 서 론

국내에서는 1999년 전자서명법이 개정되면서 금융권을 시작으로 PKI 보안 솔루션들이 급속도로 보급되기 시작했다. 인증 분야는 공인인증서가 자리를 잡았으며, 사용자 정보 보호를 위한 클라이언트와 서버 간의 보안 통신 프로토콜은 금융감독원의 전자금융안전대책기준(2000.9)에 의거 SEED 암호 알고리즘의 사용이 권고 되어 TLS/SSL 암호 통신 프로토콜이 아닌 민간 보안 업체에서 직접 개발한 암호 프로토콜을 사용하게 되었다[1].

현재 금융권에서는 대고객 서비스 분야인 인터넷 뱅킹 분야에 사용되는 암호 통신 프로토콜을 가장 중요하게 다루고 있으며, 프로토콜의 형태는 PKI/TTP를 기반으로 구성되어 있고, 공개키 암호 알고리즘으로는 RSA만이 사용되고 있다. 민간 기업에서 개발한 암호 프로토콜은 혼합(hybrid) 구조로, 키 교환은 공개키 암호 시스템을 이용하고, 메시지에 대한 암호화는 교환된 세션키를 이용해 대칭키 기반으로 암호화하고 있다. 따라서 클라이언트와 서버 간의 보안 통신의 안전성은 키 교환 프로토콜의 안전성에 의존한다.

키 교환 프로토콜의 가장 기본적인 안전성 요구사항은 키 기밀성(key secrecy)이다. 공격자는 정직한 개체 간의 통신을 도청하거나, 메시지를 위조하거나 변조하여 전송 할 수 있다. 키 기밀성은 이러한 수동적 및 능동적 공격자가 세션키에 대한 어떠한 정보도 얻을 수 없어야 함을 의미한다. 키 기밀성 이외에 키 교환 프로토콜에 요구되는 안전성은 전방향 안전성(forward secrecy)과 기지 키 공격에 대한 안전성(known-key secrecy)이다. 전방향 안전성은 개체의 롱텀 개인키(long-term private key)를 알고 있는 어떠한 공격자라도 개인키 노출 이전에 정직한 두 개체 간에 성공적으로 교환된 세션키에 대한 어떠한 정보도 얻을 수 없어야 함을 의미한다. 기지 키 공격에 대한 안전성은 여러 세션의 세션키들이 노출되어도 노출되지 않은 세션의 키 비밀성에는 영향을 주지 않아야 함을 의미한다.

금융 암호 통신 프로토콜에서 문제가 되는 것은 키 교환 부분으로, 키 기밀성, 기지 키 공격에 대한 안전성, 전방향 안전성 모두 만족하지 않는다. 만약 금융 암호 통신 프로토콜이 위와 같은 공격에 대해 안전성을 갖도록 설계 된다면, 국내 금융권 보안 수준을 한 층 강화 할 수 있고, 금융권에서는 개인키 관리를 위해 도입되는 고가의 HSM(Hardware Security Module)[1] 장비 비용을 줄일 수 있을 뿐만 아니라, 보안 소프트웨어 공급 업계도 암호 소프트웨어의 업그레이드로 인한 시장 창출에 기여할 수 있다.

본 논문에서는 국내 환경에 적합하도록 프로토콜 교체 비용을 최소화 하면서 키 기밀성, 기지 키 공격에 대한 안전성, 전방향 안전성을 모두 제공하는 키 교환 프로토콜을 제안한다.

본 논문의 2장에서는 현재 국내 금융권 암호 통신 프로토콜의 사용 현황과 특징을 살펴보고, 금융권 암호 통신 프로토콜의 취약점 및 대책 현황을 설명한다. 3장에서는 국내 금융권에 적용 가능한 전방향 안전성을 제공하는 두 가지의 안전한 키 교환 프로토콜을 제시한다. 4장에서는 제안하는 키 교환 프로토콜의 기존 프로토콜 대비 효율성과 안전성에 대해서 설명하고, 5장에서 결론을 맺는다. 부록에서 제안 프로토콜의 안전성을 증명한다.

## II. 국내 금융권 암호 통신 프로토콜의 사용 현황 및 취약점

국내에서는 인증기관(CA: Certificate Authority)이 TTP(Trusted Third Party)의 역할을 하는 PKI가 공인인증이라는 이름으로 광범위하게 이용되고 있다. CA가 발행하는 공인인증서는 신원확인 이외에, 전자서명에 대한 부인 방지, 암호화 통신을 위한 공개키 전달 등 다양한 용도로 사용되고 있다. 그리고 세계적으로 통용되고 있는 웹 브라우저(IE, FireFox, Safari, Opera 등)에서도 SSL(Secure Socket Layer)을 지원하고 있기 때문에, 국내를 포함해 세계 145개국 9만 여개의 ISP(Internet Service Provider)에서 SSL 인증서를 발급받아 암호 통신 서비스를 제공하고 있다 (2008년 5월 Verisign 기준, TNS Research 조사).

현재 ISO, IEEE 등 국제 표준기구에서 표준으로 정해진 인증서를 사용하는 키 교환 스킴들은 대부분이 Diffie-Hellman 키 교환 방식[10]을 확장한 스킴들이다. 이는 Diffie-Hellman 키 교환 방식이 지니고 있는 구조의 유연성(flexibility)과 안전성 증명의 용이성에 기인하고 있다. 현재 학계에서 발표되는 키 교환 방식의 대부분도 Diffie-Hellman 방식을 이용하여 다양하고 융합된 환경과 강화된 안전성을 제공하도

1) HSM(Hardware Security Module): 하드웨어 형태의 보안 모듈로 키 관리와 암호 연산을 대행한다.

록 설계되고 있다[8,9,11-14]. 반면에 국내에서는 상용 서비스 중인 모든 인증기관이 암호 통신을 위해 키 암호화(key encipherment) 용도로 발급하는 인증서(X.509)들은 모두 RSA 알고리즘만 지원하고 있다. 이는 사회적으로 TTP(즉 인증기관)를 통한 RSA 기반의 암호/인증 서비스가 이미 일반화 되어, TTP와 RSA 기반이 아닌 키 교환 프로토콜은 국내에 사용하기 어렵기 때문이다. 따라서 금융권에서 사용하는 암호 통신 프로토콜도 모두 RSA 암호 스킴만을 사용하고 있다. 또한 PKI에서 TTP의 역할을 하는 인증기관은 보안 사고에 대해서 오프라인(off-line) 상에서 중재역할을 하는 보험 창구의 기능도 수행하기 때문에, TTP가 존재하지 않는 자력 집행형 인증(self-enforcement authentication) 방식의 프로토콜도 금융 분야와 같은 민감(critical)한 업무에는 사회적 분위기상 적용하기 어렵다.

## 2.1 금융권에서 사용 중인 암호 통신 프로토콜

국내 금융권에 사용 중인 암호 프로토콜은 공개키 암호 시스템과 대칭키 암호 시스템을 혼합(hybrid cipher)해서 사용 중이며, 그 형태에 따라 크게 키 교환 정보와 암호문이 일체화된 전자 봉투 방식과 키 교환을 별도로 수행 후 암호화된 메시지만 주고받는 세션키 교환 방식의 두 가지 형태로 구성된다[2].

[표 1]은 국내 인터넷 뱅킹 서비스를 제공하고 있는 시중 은행별 사용 중인 암호 프로토콜 형태에 대해 기술한 표이다.

세션키 교환 방식과 전자봉투 방식을 함께 쓰는 은행은 하나의 시스템에서 두 가지 방식을 모두 사용하는 것이 아니라, 각각의 업무에서 하나의 방식을 채택해 사용하고 있다. (예: 개인 인터넷 뱅킹-세션키 교환 방식, 기업 인터넷 뱅킹 - 전자 봉투 방식)

### 2.2.1 전자봉투 방식

전자봉투 방식은 하나의 암호문에 키 교환 정보와 메시지를 암호화한 내용이 함께 구성되어 있어 하나의 통신 전문으로 처리할 수 있는 편의성 때문에 많이 사용되고 있다. 하지만 세션키 교환 방식에 비해 재생공격(replay attack), 세션 하이젝킹 공격(session hijacking attack) 등에 취약해 점차 세션키 교환 방식으로 바뀌고 있는 추세이다. (대부분의 은행들이 2010년까지 세션키 방식으로 교체할 예정이다.)

[표 1] 은행별 인터넷 뱅킹 암호 프로토콜 방식
(2009년 4월 기준)

| No. | 은행 명 | 암호 프로토콜 |
|---|---|---|
| 1 | 국민은행 | 세션키 교환 방식 + 전자봉투 방식 |
| 2 | 우리은행 | 세션키 교환 방식 |
| 3 | 신한은행 | 세션키 교환 방식 |
| 4 | 외환은행 | 세션키 교환 방식 |
| 5 | 한국씨티은행 | 세션키 교환 방식 |
| 6 | 기업은행 | 세션키 교환 방식 |
| 7 | SC제일은행 | 세션키 교환 방식 + 전자봉투 방식 |
| 8 | 하나은행 | 세션키 교환 방식 |
| 9 | 경남은행 | 전자봉투 방식 |
| 10 | 광주은행 | 전자봉투 방식 |
| 11 | 대구은행 | 전자봉투 방식 |
| 12 | 부산은행 | 전자봉투 방식 |
| 13 | 전북은행 | 전자봉투 방식 |
| 14 | 제주은행 | 전자봉투 방식 |
| 15 | 농협 | 전자봉투 방식 |
| 16 | 새마을금고 | 전자봉투 방식 |
| 17 | 수협 | 전자봉투 방식 |

다음은 *Client*가 전자봉투 형태의 암호문을 만드는 과정이다.

*Step* 1. *Server* 인증서의 유효성을 검증 후 *Server* 인증서 $Cert_S$에서 *RSA* 공개키를 추출한다. (*Server* 인증서는 *X.509* 표준을 따르며, 보통 공인인증기관(*CA*)에서 발급 받는다.)

*Step* 2. 대칭키 암호 알고리즘을 선택한다. (보통 SEED를 사용한다.)

*Step* 3. 난수 발생기(secure-random)를 이용해, 선택된 대칭키 암호 시스템에서 사용 가능한 키 길이의 세션키를 생성한다.

*Step* 4. 세션키를 *Server*의 공개키로 암호화(PKCS#1 RSAES)한다.

*Step* 5. 세션키로 전송하고자 하는 평문을 암호화한다. (예: SEED-CBC)

*Step* 6. 암호화된 세션키(키 교환 정보)와 평문에 대한 암호문을 하나의 암호 전문으로 구성한다.

*Step* 7. 완성된 암호문을 *Server*에게 전송한다.

다음은 *Server*가 전자봉투 형태의 암호문을 복호화하는 과정이다.

*Step* 1. *Client*로부터 전송된 암호 전문을 키 교환 정보와 암호화된 평문으로 분리한다.

*Step* 2. *Server*의 *RSA* 개인키를 이용해 암호화된 세션키를 복호화한다.

*Step* 3. 복호화된 세션키로 암호화된 평문을 복호화한다. (예: SEED-CBC)

### 2.1.2 세션키 교환 방식

세션키 교환 방식은 통신을 개시하는 단계에서 보안 핸드쉐이킹 과정이 있어 키를 교환 후 암호화 통신을 수행한다. 보안 세션을 별도로 관리해야하는 부담이 따르지만, 모든 암호화 트렌젝션에 RSA 연산을 해야 하는 전자봉투 방식에 비해, 초기 키 교환 과정에서만 RSA 연산이 한번 수행되기 때문에, 서버 시스템 부하를 최소화 할 수 있어 많이 사용 되고 있다. 전체적인 세션키 교환과정은 [그림 1]과 같은 방식이나, 정확하게는 두 종류의 키 교환 방식이 존재한다.

다음은 금융권에서 사용 중인 키 교환 과정이다. 이 중 사전비밀정보 PS(premaster secret)를 통해 세션키를 생성하는 키 생성 알고리즘 $f$는 사용 중인 암호 소프트웨어 마다 조금씩 다르다.

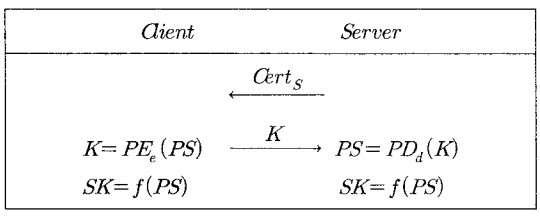
*Step* 1. *Client*는 *Server*에게 *Server* 인증서 $Cert_S$를 요청한다.

*Step* 2. *Server*는 *Client*로 *Server* 인증서 $Cert_S$를 전송한다.

*Step* 3. *Client*는 *Server*인증서의 유효성을 검증 후, 난수 발생기(secure-random)을 이용해, 임의의 사전비밀정보 *PS*(premaster secret)를 생성하고, *Server* 인증서 내의 *RSA* 공개키 $e$를 이용해 *PS*를 암호화(PKCS#1 RSAES)한 $K = PE_e(PS)$를 *Server*로 전송한다.

*Step* 4. *Server*는 $K$를 자신의 *RSA* 개인키 $d$를 이용해 복호화한다($PS = PD_d(K)$).

*Step* 5. *Client*와 *Server*는 세션키 생성함수 $f$를 이용해 *PS*로부터 세션키 $SK = f(PS)$를 계산한다.




[그림 1] 세션키 교환 방식

## 2.2 금융권에서 사용 중인 암호 통신 프로토콜의 취약점 및 대책 현황

### 2.2.1 전자봉투 방식의 취약점

키 기밀성. 전자봉투 방식은 클라이언트 가장 공격(impersonation attack)에 취약하기 때문에 능동적 공격자(active attacker)에 대한 키 기밀성을 제공하지 않는다. 전자봉투 방식은 클라이언트가 세션키를 생성해 서버의 공개키로 암호화 후 서버로 전달하는 방식이다. 이 때 클라이언트가 전송하는 메시지에 대한 인증(authentication)이 없으므로 서버는 클라이언트로부터 받은 메시지가 실제 클라이언트가 보낸 메시지인지 확인할 수 없다. 따라서 공격자는 정직한 클라이언트를 가장하여 서버와 세션키를 교환할 수 있게 되고, 서버는 클라이언트와 세션키를 교환했다고 생각하고 세션키를 향후 보안 통신에 사용할 것이다.

기지 키 공격에 대한 안전성. 전자봉투 방식은 클라이언트가 세션키를 생성해 서버의 공개키로 암호화 후 서버로 전달할 때 난수 또는 타임스탬프(time stamp) 등이 사용되지 않으므로 재전송 공격(replay attack)이 가능하다. 만약 공격자가 특정 세션에서 클라이언트로부터 서버에 전송되었던 암호문을 새로운 세션에서 재전송한다면, 두 세션에서 계산되는 세션키는 항상 동일하다. 이 경우, 만약 한 세션의 세션키가 노출된다면 노출되지 않은 다른 세션키 역시 알아낼 수 있으므로 전자봉투 방식은 기지 키 공격에 취약하다.

전방향 안전성. 전자봉투 방식은 클라이언트가 세션키를 생성해 서버의 공개키로 암호화 후 서버로 전달하는 방식이기 때문에 서버의 개인키가 노출되었을 때, 이 키를 이용해 과거 보안 통신의 암호문을 복호하여 세션키를 알아낼 수 있기 때문에 전방향 안전성을 만족하지 않는다.

다음은 전자봉투 방식이 서버의 개인키 노출에 취약함을 실제 실험을 통해 알아본 것이다. 다음의 [그림 2]와 [그림 3]은 전자봉투 방식이 서버의 개인키 노출에 취약함을 실제 예를 통해 알아보기 위하여 전자봉투 형태의 암호 프로토콜을 사용하는 N은행의 통신 패킷을 캡쳐한 결과와 그 암호문의 구성을 분석한 것이다.

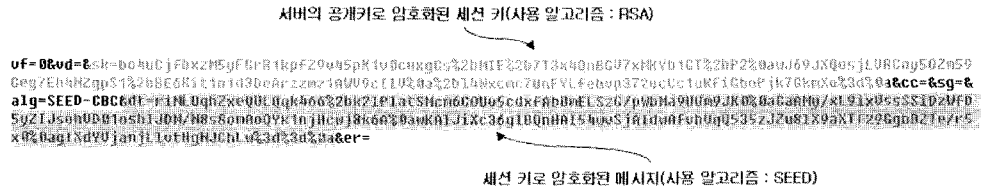
[그림 4]의 분석 결과를 보면 전자 봉투 형태의 프

```
POST /servlets/IGateWeb HTTP/1.1
Accept: */*
Accept-Language: ko
Content-Type: application/x-www-form-urlencoded
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.2; SV1; .NET CLR 1.1.4322; .NET CLR 2.0.50727)
Host: [redacted]
Content-Length: 620
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: JSESSIONID=40ef810TZ32j4QI18IavliOKgOVjGDfXja7EAabvmPRMgfjUu2in5K77R8cbm6UC.IB_WE83_servlet_ib02

INIpluginData=vf%3D0%26vd%3D%26sk%3Dbo4uCjfDxzM5yFGrR1kpfZ9u45pK1v0CuxgGs%252bMIE%252b7T3x4QnBGU7xNKYb1GT%25
2bP2%250awJ69JXQusjLVRCny5OZm59Geg7Eh4NZgpS1%252bBE6Kit1n1d3DeArzzmz1AWV9cIlV%250a%252bl4Wxcmc7UnFYLFebvp372
vcCc1uKFiGboPjk7GkmXo%253d%250a%26cc%3D%26sg%3D%26alg%3DSEED-CBC%26dt%3DriMLOqhZxeQULQqk466%252bk2lPlatSMcm6
COUv5cdxFAbBmELSzG%2FpVbMa9UUm9JK0%250aCaAMy%2FxL9ixUssSSiDzWfD5yZIJsohVD01oshIJDM%2FN8s8omAoQYk1njHcwj8k6A%
250awKAlJiXc36glBQnHAI54wvSjAIdwAfvhUqQ535zJZu81X9aXTf29GgbBZTe%2Fr5x0%250agIXdYUjanjLlvtHgNJChLw%253d%253d%
250a%26er%3D&com_cmd=&InqCbn=&RnmNbr=&Url=&Gbn_1=&InqGjaNbr=&GjaSctNbr=&RnmNbr2=
```

↑ 전자 봉투 형태로 암화된 메시지

〔그림 2〕 전자봉투 형태로 암호화된 HTTP Request(N은행)




〔그림 3〕 전자봉투 형태로 구성된 암호문의 분석

로토콜은 비교적 분석이 간단함을 알 수 있다. 전자봉투 형태의 암호문은 모든 암호문에 복호화에 필요한 모든 정보(예: 키 교환을 위한 정보)가 들어있기 때문에, 패킷 단위로 수집하기도 편리하고, 프로토콜 자체가 전방향 안전성을 제공하지 않기 때문에, 서버의 공개키에 쌍이 되는 개인키를 소유한 사람은 누구든 암호문을 쉽게 복호화 할 수 있다. 결과적으로 세션키를 복호할 수 있고, 알아낸 세션키로 암호화된 메시지를 복호할 수 있다.

### 2.2.2 세션키 교환 방식의 취약점

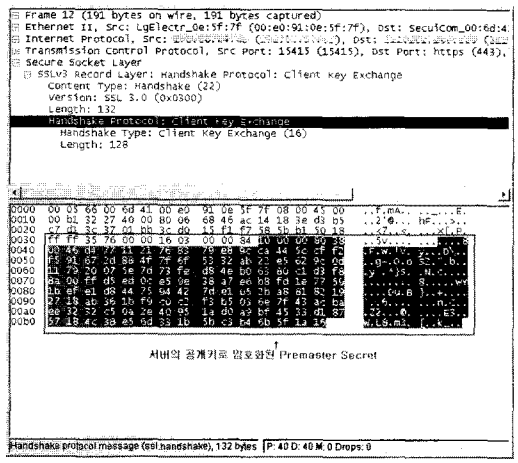
키 기밀성. 세션키 교환 방식도 전자봉투 방식과 마찬가지로 클라이언트가 세션키를 생성하는데 사용되는 사전비밀정보 PS를 생성해 서버의 공개키로 암호화 후 서버로 전달하는 방식으로 클라이언트가 전송하는 메시지에 대한 인증(authentication)이 없으므로 누구나 클라이언트를 위장하여 서버와 세션키 교환이 가능하다. 따라서 세션키에 대한 키 기밀성이 만족하지 않는다.

기지 키 공격에 대한 안전성. 세션키 교환 방식도 전자봉투 방식과 마찬가지로 재전송 공격(replay attack)으로 매 세션의 세션키를 동일하게 설정할 수 있다. 따라서 기지 키 공격에 안전하지 않다.

전방향 안전성. 세션키 교환 방식은 클라이언트가 사전비밀정보 PS를 생성해 서버의 공개키로 암호화 후 서버로 전달하는 방식이기 때문에 서버의 개인키가 노출되었을 때, 이 키를 이용해 과거 보안 통신의 암호문을 복호하여 세션키를 알아낼 수 있다. 따라서 전방향 안전성을 만족하지 않는다.

다음은 세션키 교환 방식이 서버의 개인키 노출에 취약함을 실제 실험을 통해 알아본 것이다. 다음의 〔그림 4〕와 〔그림 5〕는 세션키 교환 방식이 서버의 개인키 노출에 취약함을 실제 예를 통해 알아보기 위하여 세션키 교환 방식의 암호 프로토콜을 사용하는 K은행과 H은행의 키 교환 통신 부분을 캡쳐한 결과이다.

K은행의 경우(〔그림 4〕) SSL Handshake를 통해 키 교환하는 것을 볼 수 있다. 이 경우에는 TLS/SSL 프로토콜이 공개이므로, SSL Handshake 과정에서의 패킷과 서버의 RSA 개인키만 있으면 세션키를 계산




〔그림 4〕 K은행 키 교환 과정의 패킷


〔그림 5〕 H은행 키 교환 과정의 패킷

할 수 있다.

H은행의 경우(〔그림 5〕)에도 Handshake 과정에서의 패킷과 서버의 RSA 개인키만 있으면 PS(premaster secret)까지는 복호화가 가능하나, PS를 통해 세션키(SK)를 계산하는 $f$ 알고리즘은 공개되어 있지 않다. 하지만 암호 소프트웨어를 담당하는 은행 직원과 개발 업체는 이 내용을 공유하고 있으므로, 최소한 이들이 내부공격자(inside attacker)가 되는 경우 세션키(SK)를 쉽게 계산할 수 있다. 즉, 현재 금융권에서 사용 중인 세션키 교환 방식도 전자봉투 방식과 마찬가지로 전방향 안전성을 제공하지 않으므로, 서버의 개인키와 $f$ 알고리즘을 아는 공격자인 경우 세션키를 쉽게 얻을 수 있다.

### 2.2.3 키 유출 사고에 대한 대책 현황

앞에서 언급한 것과 같이 모든 금융권에서 사용 중인 암호 통신 프로토콜은 암호문과 서버의 RSA 개인키만 있으면 모든 과거 암호 메시지를 복호화 할 수 있는 구조이기 때문에 내부에 악의적인 직원이 개인키를 유출하거나, 관리의 부실로 개인키를 도난당했을 경우 심각한 보안 문제가 발생하게 된다.

금융권에서도 이러한 문제 때문에 개인키의 중요성에 대해 인식하고 있어, 개인키 유출 방지를 위해 HSM(Hardware Security Module) 장비를 도입해 한번 주입된 개인키는 밖으로 나올 수 없게 구성해 놓았다. 하지만 결국 HSM 장비의 장애상황 발생을 우려해 개인키를 파일로 백업 받아 놓아, HSM 장비의 역할이 유명무실하다고 볼 수 있다. 그리고 HSM 장비를 도입한 은행도 30~40% 정도 밖에 되지 않기 때문에, 대부분의 은행들은 단순히 파일로 개인키를 관리하고 있어 그 관리의 실태가 심각하다. 따라서 암호 소프트웨어를 담당하는 금융권 내부 직원 또는 금융권에 서버 및 암호 소프트웨어 유지 보수를 담당하는 외부 직원이라면 대부분이 개인키를 쉽게 얻을 수 있다. 더욱 우려되는 상황으로, 은행 별로 HSM 장비를 도입했다가도 못쓰고 있는 경우도 있는데, 이는 HSM 장비 성능이 은행에 사용 중인 슈퍼컴퓨터 급 서버 성능에 미치지 못해 HSM 장비가 병목 구간이 되어 정상적인 서비스를 할 수 없었기 때문이다.

상기한 바와 같이 개인키 유출은 가능성이 희박한 학술적인 우려가 아니라, 국내의 특수한 환경에서 반드시 대처해야할 위협이다. 이러한 이유에서 개인키 유출에도 과거의 통신으로부터 개인정보와 금융정보를 보호할 수 있는 전방향 안전성을 제공하는 키 교환 프로토콜의 개발이 반드시 필요한 상황이다.

## III. 금융권에 적용 가능한 키 유출 사고에 안전한 키 교환 프로토콜

서론에서 언급한 바와 같이 국내에 상용 서비스 중인 모든 인증기관(CA)이 RSA 공개키 기반의 인증서만을 발급하고 있기 때문에, 프로토콜의 교체 비용을 최소화하기 위해서는 RSA 인증서와 RSA 알고리즘의 파라미터(parameter)만을 사용하여 키 교환 스킴을 설계해야하는 제약이 따른다. 본 장에서는 전방향 안전성을 위해 널리 사용되는 Diffie-Hellman 기법에 현재의 인증서를 그대로 사용하는 RSA 암호

와 전자서명을 혼합한 키 교환 프로토콜을 제안한다. 먼저 RSA 알고리즘과 Diffie-Hellman 기법을 혼합할 수 있는 직시적인(straightforward) 방법인 FKE1 (Financial Key Exchange1) 프로토콜을 제시하고, 좀 더 교체비용이 적은 FKE2 프로토콜을 제안한다. 제안 프로토콜은 모두 능동적 공격자에 대한 키 기밀성과 기지 키 공격에 대한 안전성 및 전방향 안전성을 제공하도록 설계되었다.

본 논문에서는 $a$부터 $b$ 사이의 정수의 집합을 $[a,b]$로 나타낸다. 집합 $S$로부터 임의로 선택된 $c$에 대한 표기로 $c \xleftarrow{R} S$을 사용한다. 제안 프로토콜에서 사용하는 기호는 [표 2]와 같다.

[표 2] 기호

| | |
|---|---|
| $\theta$ | 안전성 파라미터(security parameter) |
| $PKE=(PK, PE, PD)$ | 공개키 암호 알고리즘. $PK$는 키 생성 함수로 $\theta$를 입력으로 하여 개인키와 공개키 쌍인 $(e,d)$를 생성. $PE_e(m)$!는 암호화 알고리즘으로 $e$를 사용하여 평문 $m$에 대한 암호문 $c$를 출력. $PD_d(c)$는 복호화 알고리즘으로 만약 암호문이 올바르다면 $d$를 사용하여 암호문에 대한 평문을 출력하고, 그렇지 않다면 $\perp$를 출력 |
| $\Sigma=(Gen, Sign, Verify)$ | 서명 알고리즘. $Gen$은 키 생성 함수로 $\theta$를 입력으로 하여 서명용 키와 검증용 키 쌍인 $(d,e)$를 생성. $Sign_d(m)$은 서명 생성 알고리즘으로 $d$를 사용하여 평문 $m$에 대한 서명 $\sigma$를 생성. $Verify_e(m,\sigma)$는 서명 검증 알고리즘으로 $e$를 사용하여 만약 평문 $m$에 대한 서명 $\sigma$가 올바르다면 1을 출력하고, 그렇지 않다면 0을 출력 |
| $C$와 $S$ | 클라이언트 $C$와 서버 $S$의 아이디 |
| $Cert_X$ | $X$의 공개키 인증서 |
| $H$ | $\{0,1\}^* \rightarrow \{0,1\}^\theta$인 해쉬 함수 |

### 3.1 FKE1 프로토콜

**초기화 단계.** 클라이언트 $C$와 서버 $S$는 인증기관 $(CA)$으로부터 전자서명용 인증서 $Cert_C$와 $Cert_S$를 각각 발급 받는다. 공개 파라미터인 $(\Sigma, H)$는 모든 개체에게 접근 가능한 정보이다. 서명 알고리즘으로 RSA 서명 알고리즘을 사용한다.

**키 교환 단계.** 클라이언트 $C$와 서버 $S$는 세션키를 교환하기 위해 다음과 같이 프로토콜을 수행한다.

*Step* 1. 서버 $S$는 Diffie-Hellman 파라미터인 $P_{DH}=(G,p,g)$을 생성한다. $G$는 소수 $p$를 위수(order)로 갖는 순환군이고, $g$는 $G$의 생성자(generator)이다. 서버 $S$는 $[1,p]$ 내에서 랜덤하게 $b$를 선택하여 $Y_b = g^b \bmod p$를 계산한다. 그리고 RSA 개인키 $d_S$를 이용해 서명값 $\sigma_S = Sign_{d_S}(P_{DH}, Y_b)$를 계산 후, 클라이언트 인증을 받을지 여부를 결정해 $c \in \{0,1\}$를 선택 후, 클라이언트 $C$에게 $(Cert_S, (P_{DH}, Y_b), \sigma_S, c)$를 전송한다.

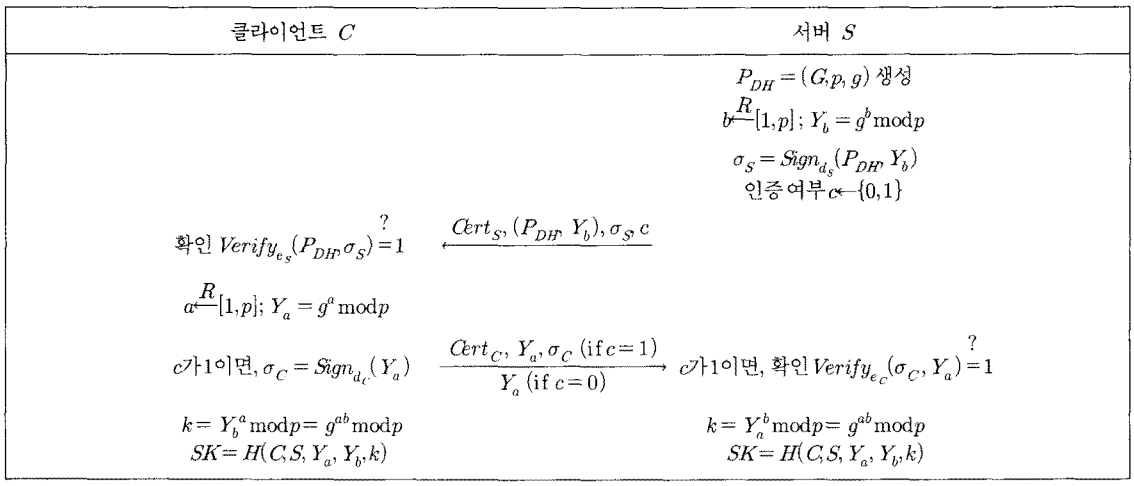
*Step* 2. 클라이언트 $C$는 $Cert_S$의 유효성을 검증 후, 유효하다면 $Cert_S$에서 공개키를 추출하여 서명 $\sigma_S$를 검증한다. 만약 $\sigma_S$가 유효하다면, $[1,p]$ 내에서 랜덤하게 $a$를 선택하여 $Y_a = g^a \bmod p$를 계산한다. 만약, *Step* 1에서 서버로부터 받은 $c$의 값이 0인 경우 서버 $S$에게 $Y_a$를 전송하고, $c$의 값이 1인 경우 클라이언트 인증을 요구하는 것이므로, 자신의 개인키 $d_C$를 이용하여 $Y_a$에 대한 서명 $\sigma_C = Sign_{d_C}(Y_a)$를 생성한다. 그리고 서버 $S$에게 $(Cert_C, Y_a, \sigma_C)$를 전송한다.

*Step* 3. 클라이언트 $C$는 $k=(g^b)^a \bmod p$를 계산한다. 서버 $S$는 $c$가 0인 경우 $k=(g^a)^b \bmod p$ 를 계산하고, $c$가 1인 경우 $Cert_C$의 유효성을 검증하고, 유효하다면 서명 값 $\sigma_C$를 검증한다. $\sigma_C$가 유효하다면 서버 $S$는 $k=(g^a)^b \bmod p$를 계산한다.

*Step* 4. 클라이언트 $C$와 서버 $S$는 대칭키 암호 시스템에서 사용 가능한 형태의 세션키 $SK=H(C,S,Y_a,Y_b,k)$를 계산한다.

실제로 금융권에서는 모든 암호 통신이 클라이언트 인증을 받지는 않기 때문에 클라이언트 인증 부분은 옵션으로 처리해 두었다. Step 4 이후 해쉬 함수나 MAC(message authentication code)을 이용한 키 확인(key conformation) 방법을 통해 정상적으로 키가 교환되었는지 확인하는 과정을 추가할 수 있다.

FKE1 프로토콜은 기존의 서명을 사용한 DH 키 교환 방식과 큰 차이가 없고, 단지 DH 키 교환을 위해 DH 파라미터를 서버가 생성 후, RSA 서명해서 클라이언트에 전달해 주는 부분에서 차이가 있다. 현재 국내 금융 보안 서버에는 키 암호화(key




[그림 6] FKE1 프로토콜

encipherment)용 인증서만 발급되어 사용되고 있고, 클라이언트에게는 서명용 인증서만 발급되어 사용되고 있다. 따라서 본 프로토콜의 도입 시에는 서버가 전자 서명용 인증서를 별도로 발급 받아야 하므로 추가 비용이 발생한다. 다음에서 클라이언트 측에서만 RSA 전자서명을 사용하도록 한 스킴을 제안한다.

### 3.2 FKE2 프로토콜

초기화 단계. 클라이언트 $C$와 서버 $S$는 신뢰된 기관($CA$)으로부터 전자서명용 인증서 $Cert_C$와 키 암호화용 인증서 $Cert_S$를 각각 발급 받는다. 공개 파라미터인 $(PKE, \Sigma, H)$는 모든 개체에게 접근 가능한 정보이다. 암호 알고리즘으로 RSA 암호 알고리즘을 사용하고, 서명 알고리즘으로 RSA 서명 알고리즘을 사용한다.

키 교환 단계. 클라이언트 $C$와 서버 $S$는 세션키를 교환하기 위해 다음과 같이 프로토콜을 수행한다.

*Step* 1. 서버 $S$는 서버 인증서 $Cert_S$에서 RSA 공개키 파라미터인 $P_{RSA}=(n,e)$를 추출한다. 서버 $S$는 $[1,n]$ 내에서 랜덤하게 $b$를 선택하여 $Y_b = e^b \bmod n$를 계산한다. $n$은 RSA의 공개된 법(modulus)이고, $e$는 공개된 지수(exponent) 이다. 서버 $S$는 클라이언트 인증을 받을지 여부를 결정해 $c \in \{0,1\}$를 선택 후, 클라이언트 $C$에게 $(Cert_S, Y_b, c)$를 전송한다.

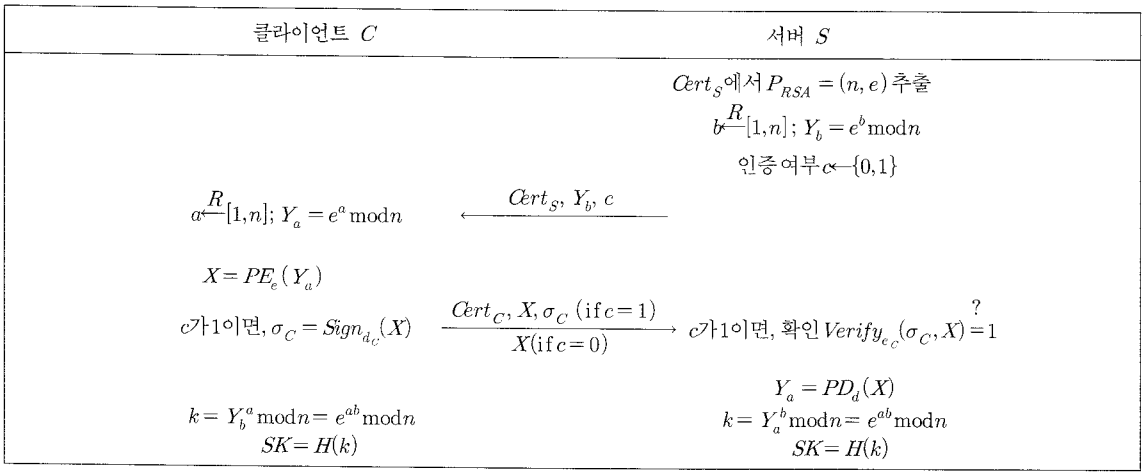
*Step* 2. 클라이언트 $C$는 $Cert_S$의 유효성을 검증 후, 유효하다면 $[1,n]$ 내에서 랜덤하게 $a$를 선택하여 $Y_a = e^a \bmod n$를 계산한다. 그리고 서버의 공개키 $e$를 이용해 $Y_a$를 암호화한 $X = PE_e(Y_a)$를 계산한다. (이때, 암호화 함수 $PE$의 설계는 PKCS#1 RSAES-OAEP를 따를 수 있다.) 만약, *Step* 1에서 서버로부터 받은 $c$의 값이 0인 경우 서버 $S$에게 $X$를 전송하고, $c$가 1인 경우 클라이언트 인증을 요구하는 것이므로, 자신의 개인키 $d_C$를 이용하여 $X$에 대한 서명 $\sigma_C = Sign_{d_C}(X)$을 생성한다. 그리고 서버 $S$에게 $(Cert_C, X, \sigma_C)$를 전송한다.

*Step* 3. 클라이언트 $C$는 $k=(Y_b)^a \bmod n$를 계산한다. 서버 $S$는 $c$가 0인 경우 $X$를 복호화 해 $Y_a = PD_d(X)$를 생성 후, $k=(Y_a)^b \bmod n$를 계산하고, $c$가 1인 경우 $Cert_C$의 유효성을 검증한 후, 유효하다면 서명 값 $\sigma_C$를 검증한다. $\sigma_C$가 유효하다면 서버 $S$는 $X$를 복호화 해 $Y_a = PD_{d_S}(X)$를 생성 후, $k=(e^a)^b \bmod n$를 계산한다.

*Step* 4. 클라이언트 $C$와 서버 $S$는 대칭키 암호 시스템에서 사용 가능한 형태의 세션키 $SK=H(k)$를 계산한다.

Step 4 이후 해쉬 함수나 MAC을 이용하여 정상적으로 키가 교환되었는지를 확인하는 키 확인(key conformation) 과정을 추가할 수 있다.

완전성(Completeness). 프로토콜이 모두 정상




〔그림 7〕 FKE2 프로토콜

적으로 수행되면 클라이언트 $C$와 서버 $S$는 동일한 임시키 $k=(e^a)^b \bmod n=(e^b)^a \bmod n=e^{ab} \bmod n$를 계산하게 되고, 동일한 세션키 $SK=H(e^{ab})$를 공유하게 된다.

FKE2 프로토콜은 기존에 사용 중인 키 암호화용 서버 인증서와 클라이언트의 서명용 인증서를 교체하지 않고 사용할 수 있는 스킴으로, RSA 그룹 $Z_n^*$에서 Diffie-Hellman 키 교환 기법을 사용한 것으로 밑(base)을 서버의 RSA 공개키 $e$로 사용한다. 현재 금융권에서는 $e$를 고정된 소수인 $2^{16}+1$을 사용하고 있다〔7〕.

FKE2 프로토콜의 전방향 안전성은 HDH(Hash Diffie-Hellman) 문제의 어려움에 기반한다. HDH 문제는 해쉬된 Diffie-Hellman 튜플(tuple)값과 랜덤값을 구분하는 문제로 부록에서 자세히 설명한다. 합성수를 위수(composite order)로 가지는 순환군 $Z_n^*$에서 DDH 문제는 어렵다고 알려져 있다〔5〕. 여기서 $n=pq$이고, $p, q, \frac{p-1}{2}, \frac{q-1}{2}$는 소수이고, $Z_n^*$의 위수는 $(p-1)(q-1)$이다. 따라서 $Z_n^*$에서 HDH 문제 역시 어려운 문제라는 것은 자명하다.

만약 서버 인증서 발급 과정에서 RSA 키 생성 시 $e$의 위수가 $\phi(n)$이 되고, 위의 조건을 만족하도록 $p, q$를 선택한다면 기존의 $e$를 교체하지 않아도 되므로 프로토콜 교체 비용도 최소화할 수 있다.

〔그림 8〕은 FKE2 프로토콜을 실제로 구현했을 때 클라이언트와 서버가 정상적으로 같은 세션키를 공유하는지 확인할 수 있는 예제 프로그램 코드와 실행 결과이다.

## IV. 기존 프로토콜 대비 효율성 및 안전성 분석

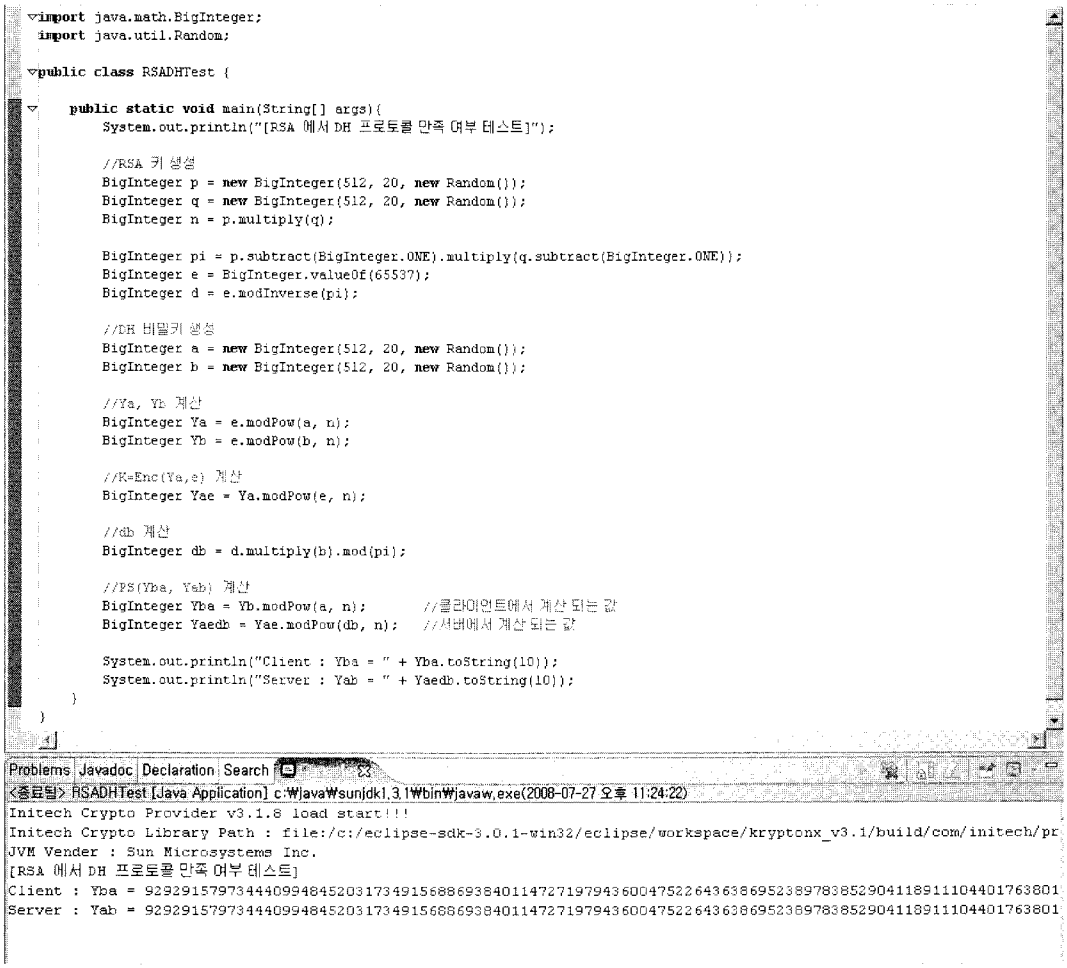
### 4.1 기존 프로토콜 대비 효율성

금융권 암호 통신 프로토콜에서 가장 많은 비용이 드는 연산은 지수승 연산이다. 그 중 1024bit에 가까운 RSA 개인키(private exponent) '$d$' 에 대한 지수승 연산 수행 시 가장 많은 비용이 필요하다. 그러므로 금융권에서는 서버를 기준으로 지수승 연산을 최소화 하는 프로토콜 설계를 요구하고 있다. (반면 클라이언트에서 지수승 연산이 많이 일어나는 것은 수용하고 있다.)

현재 금융권에 적용된 키 교환 시스템은 모두 한 번의 키 교환 과정에서 서버 기준으로 한 번의 지수승 연산이 이루어진다. 기존 프로토콜의 경우 키 교환 요청 시 서버 인증서만 클라이언트로 전달되지만, 제안된 프로토콜은 유형에 따라 서버 인증서와 몇 가지 안전성 파라미터(security parameter)가 추가로 전달된다. 이 안전성 파라미터에 따라 FKE1 프로토콜과 FKE2 프로토콜의 성능은 많은 차이를 나타낸다.

#### 4.1.1 FKE1의 효율성

키 교환 과정에서 사용 되는 DH Parameter 중 $p, g$는 한번 만 생성 후 계속 사용해도 되지만, Step 1 과정에서 $b, Y_b, \sigma_S$는 매번 생성해야 하므로, 2번의 지수승 연산이 필요하고, Step 3 과정에서 $k(=g^{ab} \bmod p)$계산을 위해 1번의 지수승 연산을 수행하므로 키 교환 과정에서 총 3번의 지수승 연산이 필요하다.



```
import java.math.BigInteger;
import java.util.Random;

public class RSADHTest {

    public static void main(String[] args){
        System.out.println("[RSA 에서 DH 프로토콜 만족 여부 테스트]");

        //RSA 키 생성
        BigInteger p = new BigInteger(512, 20, new Random());
        BigInteger q = new BigInteger(512, 20, new Random());
        BigInteger n = p.multiply(q);

        BigInteger pi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
        BigInteger e = BigInteger.valueOf(65537);
        BigInteger d = e.modInverse(pi);

        //DH 비밀키 생성
        BigInteger a = new BigInteger(512, 20, new Random());
        BigInteger b = new BigInteger(512, 20, new Random());

        //Ya, Yb 계산
        BigInteger Ya = e.modPow(a, n);
        BigInteger Yb = e.modPow(b, n);

        //K=Enc(Ya,e) 계산
        BigInteger Yae = Ya.modPow(e, n);

        //db 계산
        BigInteger db = d.multiply(b).mod(pi);

        //PS(Yba, Yab) 계산
        BigInteger Yba = Yb.modPow(a, n);       //클라이언트에서 계산 되는 값
        BigInteger Yaedb = Yae.modPow(db, n);   //서버에서 계산 되는 값

        System.out.println("Client : Yba = " + Yba.toString(10));
        System.out.println("Server : Yab = " + Yaedb.toString(10));
    }
}
```

Problems | Javadoc | Declaration | Search

<종료됨> RSADHTest [Java Application] c:₩Java₩sunjdk1.3.1₩bin₩javaw.exe(2008-07-27 오후 11:24:22)

```
Initech Crypto Provider v3.1.8 load start!!!
Initech Crypto Library Path : file:/c:/eclipse-sdk-3.0.1-win32/eclipse/workspace/kryptonx_v3.1/build/com/initech/pr
JVM Vender : Sun Microsystems Inc.
[RSA 에서 DH 프로토콜 만족 여부 테스트]
Client : Yba = 92929157973444099484520317349155886938401147271979436004752264363869523897883852904118911104401763801
Server : Yab = 92929157973444099484520317349155886938401147271979436004752264363869523897883852904118911104401763801
```

〔그림 8〕 RSA 그룹(group) 상에서 DH 구현 예제 (개발 언어: Java)

#### 4.1.2 FKE2의 효율성

Step 1 과정에서 $Y_b(=e^b \bmod n)$를 계산해야 하므로 1번의 지수승 연산이 이루어지진다. Step 2,3 과정에서 $Enc, Dec$ 함수를 PKCS#1 RSAES-OAEP 스킴으로 설계할 경우, $l$를 RSA의 공개키 $n$의 바이트 길이, $hLen$을 OAEP에서 사용할 해쉬 알고리즘 축약 값의 바이트 길이, $mLen$을 $Y^a$의 바이트 길이라고 했을 때, $Y^a$는 PKCS#1 RSAES-OAEP 스킴에 정의된 한 블록에서 암호화 가능한 메시지의 길이 $(mLen \le l-2hLen-2)$를 벗어나기 때문에, 2개의 블록으로 나누어 암호화를 해야 한다. 그러므로 $X$의 길이는 $2l$이 되어 Step 3 과정에서 $Dec$ 함수를 2번 사용하게 되므로, 2번의 지수승 연산이 필요하다. Step 4 에서는 $k(=e^{ab} \bmod n)$를 계산하기 위해 1번의 지수승 연산이 사용된다. 그러므로 총 4번의 지수승 연산이 필요하다.

하지만 $Y_a$를 암/복호화 하는 함수를 PKCS#1 RSAES-OAEP 스킴을 사용하지 않고, $X = PE_e(Y_a) = Y_a^e \bmod n$, $PD_d(X) = X^d \bmod n$으로 설계하면, Step 1 과정에서 $F(= db \bmod \phi(n))$를 미리 계산해 둠으로써, Step 3 과정에서 단 1번의 지수승 연산으로 $k(= X^F \bmod n = (e^{ae})^{db} \bmod n = e^{ab} \bmod n)$를 계산할 수 있다. 이 경우 Step 1 과정의 지수승 연산을 포함, 총 2번의 지수승 연산이 필요하다.

#### 4.1.3 기존 프로토콜과의 성능 비교

기존에 사용 중인 키 교환 프로토콜은 서버 측 기준으로 1번의 지수승 연산이 일어나는 반면 본 논문에서

제안하는 프로토콜은 2~4번 지수승 연산이 이루어진다. 하지만 두 프로토콜 모두 서버인증서 $Cert_S$와 함께 클라이언트로 전송되는 안전성 파라미터(security parameter)를 CPU 유휴시간 대에 많이 생성해 두거나, 중복 사용 하되, 교체 주기를 하루에 한번 정도로 조절 하면, 세션키($SK$)를 계산하는 과정에서만 지수승 연산이 필요하므로, 기존 프로토콜과 성능 면에서 동일해진다. 단, FKE2는 RSAES-OAEP 사용 시 3번의 지수승 연산이 사용된다. [표 3]은 RSA 키 크기 별 지수승 연산 부하의 평균치를 조사한 것이다.

[표 3] RSA 키 길이 별 지수 연산 1회에 걸리는 평균 시간 (Intel CPU 1.83Ghz, Sun Java 환경 기준)

| 키 길이 | 시간(ms) |
|---|---|
| 512 Bit | 7 |
| 1024 Bit | 47 |
| 2048 Bit | 330 |

[표 4]와 [표 5]는 FKE 1,2와 기존 키 교환(전자봉투, 세션키 교환) 프로토콜의 지수승 연산 수와 시간(1024Bit 기준)을 비교한 것이다.

[표 4] 클라이언트 인증이 없는 경우

| 스킴 | 클라이언트 | 서버 |
|---|---|---|
| 기존 키 교환 | 1 회<br>47 ms | 1 회<br>47 ms |
| FKE 1 | 3 회<br>141 ms | 3 회<br>141 ms |
| FKE 2(DH) | 2 회<br>94 ms | 2 회<br>94 ms |
| FKE 2(OAEP) | 3 회<br>141 ms | 3 회<br>141 ms |

[표 5] 클라이언트 인증이 있는 경우

| 스킴 | 클라이언트 | 서버 |
|---|---|---|
| 기존 키 교환 | 2 회<br>94 ms | 2 회<br>94 ms |
| FKE 1 | 4 회<br>188 ms | 4 회<br>188 ms |
| FKE 2(DH) | 3 회<br>141 ms | 3 회<br>141 ms |
| FKE 2(OAEP) | | 4 회<br>188 ms |

### 4.2 제안 프로토콜의 안전성 분석

FKE1은 서명을 사용하는 인증된(authenticated) Diffie-Hellman 키 교환 방식으로 이미 [14]에서 안전성 증명이 되었으므로, 본 논문에서는 FKE2에 대한 안전성만을 증명한다. 제안 프로토콜의 안전성 증명에 필요한 암호 프리미티브(primitive)와 키 교환 프로토콜의 안전성 모델(security model)을 부록에서 정의한다.

다음 정리에서 FKE2 프로토콜이 클라이언트 인증을 포함하고(즉, $c=1$), 공개키 암호 스킴으로 IND-CCA(indistinguishability under chosen ciphertext attack)에 안전한 RSA-OAEP를 사용할 경우에 대한 안전성을 증명한다.

정리 1. $G$가 HDH(Hash Diffie-Hellman) 가정이 만족하는 그룹이고, 프로토콜에서 사용하는 $PKE$가 IND-CCA에 안전한 공개키 암호 스킴이고, $\Sigma$가 강한 위조불가능성(strong unforgeability, SUF)을 지닌 서명 스킴이라면, FKE2는 능동적 공격자에 대한 키 기밀성과 전방향 안전성, 기지 키 공격에 대한 안전성을 제공한다. 구체적인 안전성은 다음과 같다.

$$Adv_{FKE2}(\theta,t) \le \frac{2(q_{ex}+q_{se})^2}{\phi(n)} + 2q_{ex}N \cdot Adv^{HDH}_{H,GG}(\theta,t) + 2Nq \cdot Adv^{IND-CCA}_{PKE}(\theta,t) + N \cdot Adv^{SUF}_{\Sigma,F}(\theta,t),$$

$t$는 공격자의 수행 시간을 포함한 최대 총 게임의 수행 시간이다. 공격자는 $q_{ex}$개의 $Execute$ 쿼리와 $q_{se}$개의 $Send$ 쿼리를 만든다. $N$은 전체 클라이언트의 수이고, $q=q_{ex}+q_{se}$이다.

증명. 그룹 $G$에서 HDH 문제를 풀기 어렵다면, FKE2는 전방향 안전성을 제공한다. 클라이언트의 개인키 $d_C$와 서버의 개인키 $d$를 가지고 FKE2의 전방향 안전성을 깨려는 공격자 $A$는 과거 통신으로부터 모든 암호문 $X=PE_e(e^a \bmod n)$를 얻을 수 있고, 서버의 개인키를 이용해 암호문 $X$를 복호할 수 있다. 이로부터 공격자 $A$는 $(Y_a=e^a, Y_b=e^b)$를 알 수 있다. $Y_a$와 $Y_b$로부터 세션키 $SK=H(e^{ab})$에 대한 정보를 알아내기 위해서 공격자는 HDH 가정을 깨야한다. 무시할 수 없는(non-negligible) 확률로 FKE2의 전방향 안전성을 깨는 공격자 $A$를 이용하여 HDH 문제를 푸는

알고리즘을 만들 수 있다. (이에 대한 자세한 증명은 부록에서 설명한다.) 이것은 HDH 문제가 어렵다는 사실에 모순되므로 무시할 수 없는 확률로 다항 시간 안에 세션키에 대한 전방향 안전성을 깨는 공격자가 존재 할 수 없음을 의미한다. □

*PKE*가 IND-CCA에 안전한 공개키 암호 스킴이고, $\Sigma$가 강한 위조불가능성을 지닌 서명 스킴이라면, 제안 프로토콜은 능동적 공격자 $A$에 대한 키 기밀성을 제공한다. 만약 $A$가 서버를 가장하여 $b'$을 선택하고, 프로토콜의 첫 번째 라운드에서 서버 $S$의 인증서 $Cert_S$와 $Y_b' = e^{b'}$을 보낸다하더라도 실제 서버 $S$는 두 번째 라운드에서 클라이언트가 전송하는 $X = PE_e(Y_a)$로부터 $Y_a = e^a$에 대한 정보를 알아내기 위해서 공격자는 공개키 암호 스킴 *PKE*(RSA-OAEP)의 IND-CCA 안전성을 깨야한다. 또한 만약 $A$가 클라이언트를 가장하여 $a'$을 선택하고 $Y_a' = e^a$를 계산하여, 두 번째 라운드에서 $X' = PE_e(Y_a')$을 서버 $S$에게 전송하기 위해서는 $X'$에 대한 서명값을 위조하여야 한다. 무시할 수 없는(non-negligible) 확률로 FKE2의 키 기밀성을 깨는 공격자 $A$를 이용하여 *PKE*의 IND-CCA 안전성을 깨는 알고리즘이나 전자서명 스킴 $\Sigma$의 강한 위조불가능성을 깨는 알고리즘을 만들 수 있다. (이에 대한 자세한 증명은 본 논문의 full 논문인 [16]에서 설명한다.)

만약 두 세션에서 $Y_b = e^b$(또는 $Y_a = e^a$)가 서버(또는 클라이언트)에 의해 반복 사용된다면 공격자 $A$는 하나의 세션키를 이용해 다른 세션의 세션키를 알아낼 수 있다. 즉, $A$는 FKE2의 기지 키 공격에 대한 안전성을 깰 수 있다. 하지만 $Y_b$ 또는 $Y_a$가 반복되어 사용될 확률은 무시할만한(negligible) 하므로 공격자 $A$가 FKE2의 기지 키 공격에 대한 안전성을 깰 확률 역시 무시할만한 값이다. (이에 대한 자세한 증명은 본 논문의 full 논문인 [16]에서 설명한다.)

## V. 결 론

본 논문에서는 현재 국내 금융권 암호 통신 프로토콜이 서버의 개인키가 노출되었을 경우 쉽게 중요한 금융정보가 노출될 수 있음을 살펴보았으며, 클라이언트 위장공격과 기지 키 공격에 취약함을 보였다. 대부분의 보안사고가 내부 공격자에 의하여 이루어지고 있는 현실을 고려할 때, 서버의 개인키 노출에 대한 대비가 절실히 필요하다. 본 논문에서는 TTP와 RSA를 반드시 사용해야하는 국내 상황을 고려하여 국내 금융권에 사용 가능한 개인키 유출사고에 안전하며 키 기밀성과 전방향 안전성을 제공하는 증명 가능한 안전성을 가지는 키 교환 프로토콜을 제시하였다. 국내의 특수한 상황을 고려할 때 제안된 프로토콜은 국내 연구자에 의하여 수행되어야 되는 당위성과 시급성을 모두 만족하는 중요한 결과이다.

## 참 고 문 헌


[1] 금융감독원, "전자금융안전대책기준," p.12, 2000년 9월.

[2] 금융보안연구원, "종단간 암호화 적용 가이드," p.17, 2007년 10월.

[3] M. Abdalla, M. Bellare, and P. Rogaway, "DHAES: an encryption scheme based on the Diffie-Hellman problem," Submission to IEEE P1363a, p.29, Aug. 1998.

[4] M. Abdalla, M. Bellare, and P. Rogaway, "The oracle Diffie-Hellman assumption and an analysis of DHIES," CT-RSA'01, LNCS 2020, pp. 143-158, 2001.

[5] D. Boneh, "The Decision Diffie-Hellman Problem," In Proceedings of the Third Algorithmic Number Theory Symposium, LNCS 1423, pp. 48-63, 1998.

[6] M. Bellare, D. Pointcheval, and P. Rogaway, "Authenticated key exchange secure against dictionary attack," Eurocrypt'00, LNCS 1807, pp. 139-155, 2000.

[7] RSA Laboratories, "PKCS #1 v2.1: RSA Cryptography Standard," June 2002.

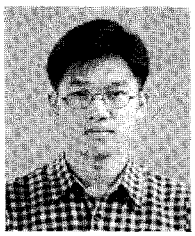
[8] R. Canetti and H. Krawczyk, "Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels," EUROCRYPT 2001, LNCS 2045, pp. 453-474, 2001.

[9] R. Canetti and H. Krawczyk, "Security Analysis of IKE's Signature-Based Key-Exchange Protocol," CRYPTO'02, LNCS 2442, pp. 143-161, 2002.

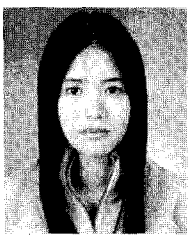
[10] W. Diffie and M.E. Hellman, "New directions in cryptography," IEEE Transactions on Information Theory, vol.

22, no. 6, pp. 644-654, Nov. 1976.
[11] W. Diffie, P. Oorschot, and M. Wiener, "Authentication and Authenticated Key Exchanges," Designs, Codes and Cryptography, vol. 2, no. 2, pp. 107-125, June 1992.
[12] I.R. Jeong, J. Katz, and D.H. Lee, "One-Round Protocols for Two-Party Authenticated Key Exchange," ACNS'04, LNCS 3089, pp. 220-232, 2004.
[13] I.R. Jeong, J.O. Kwon, and D.H. Lee, "A Diffie-Hellman Key Exchange Protocol Without Random Oracles," CANS 2006, LNCS 4301, pp. 37-54, 2006.
[14] V. Shoup, "On Formal Models for Secure Key Exchange," IBM Research Report RZ 3120, p.60, 1999.
[15] 김선종, 권정옥, "금융 보안 서버의 개인키 유출 사고에 안전한 키 교환 프로토콜," 제3회 금융부문 정보보호 우수논문 공모전, pp. 21-31, 2008년 9월, http://cist.korea.ac.kr/~pitapat/FKE 200809.pdf

## 〈著者紹介〉



김 선 종 (Seon Jong Kim) 정회원
2007년 2월: 울산대학교 공과대학 컴퓨터정보통신공학부 학사 졸업
2008년 9월~현재: 고려대학교 정보경영공학전문대학원 정보보호 전공 석사 과정
2004년 6월~현재: 이니텍 미래기술연구소 과장
〈관심분야〉 암호프로토콜, 금융정보보안



권 정 옥 (Jeong Ok Kwon) 정회원
2000년 8월: 동덕여자대학교 전자계산학과 학사 졸업
2003년 2월: 고려대학교 정보보호기술협동과정 석사 졸업
2007년 2월: 고려대학교 정보보호대학원 박사 졸업
2007년 3월~2007년 8월: 고려대학교 정보보호기술연구원 박사후연구원
2007년 9월~2009년 2월: 고려대학교 BK21 유비쿼터스 정보보호 사업단 연구교수
2009년 3월~현재: 삼성SDS 정보보안그룹 책임
〈관심분야〉 암호프로토콜, 암호이론