부가적인 키관리를 필요로 하지 않는 효율적인 분산 인증서버의 구축

(Construction of Efficient Distributed Authentication Server without Additional Key Management)

홍 성 민 † 윤 현 수 †† 이 승 원 ††† 박 용 수 ††† 조 유 근 ††††

(Seong-Min Hong) (Hyunsoo Yoon)(Seungwon Lee)(Yonsu Park)(Yookun Cho)

요 약 컴퓨터시스템과 네트웍이 발전하면서 인증과 전자서명의 필요성이 증가하고, 전자서명이나 인증을 반복적으로 많이 수행해야 하는 서버들의 필요성이 증대되고 있다. 대량으로 인증을 수행하는 서버를 구현하기 위해서, 한 대의 미니컴퓨터(minicomputer) 또는 메인프레임(mainframe)을 이용하는 것보다 저렴한 여러 대의 웍스테이션이나 PC를 이용하는 것이 경제적이다. 그러나, 여러 대의 플랫폼들로 서버를 구축하기 위해서는 모든 플랫폼들이 개인키(private key)를 가지고 있어야 하는데, 이는 많은 보안상 문제점을 야기시키고 키관리(key management) 비용을 증가시킨다. 본 논문에서는 이러한 문제점을 해결하기 위해, 한 개의 플랫폼에만 서명생성을 위한 개인키를 두며, 나머지 플랫폼들에는 개인키를 두지 않고 SASC (Server-Aided Secret Computation) 프로토콜을 이용하도록 하는 방법을 제시한다. 본 논문에서 제안하는 분산 인증서버는 사용하는 SASC 프로토콜에 따라 성능향상의 정도가 달라지므로, 이를 분석하고 실제적 적용가능성을 살펴보기 위해 구현하고 실험하였다. 본 논문에서 제안하는 방법을 이용하면, 키관리비용의 증가 없이 여러 대의 웍스테이션 혹은 PC들을 이용해서 강력한 인증서버를 구축할 수 있다.

Abstract Over the years, the extensive use of networks and distributed systems has increased the need for authentication and digital signatures. Inperforming authentication on a massive scale, authentication servers that use multiple workstations or PCs are more economical than servers that use one inicomputer or mainframe. However, the establishment of authentication servers with multiple platforms can cause some security problems and increases the cost of key management because all platforms within the server must have the private key of the authentication server.We propose a scheme which can solve these problems. The proposed scheme can establish a strong authentication server with no additional key management and improve the performance of the authentication server up to 13 times.

1. 서 론

네트웍과 컴퓨터의 발전은 인증과 전자서명의 필요성을 증가시켰고 최근의 환경변화는 대량의 인증과 전자서명이 수행되도록 요구하고 있다. 예를 들어, 전자 상거래를 통해 전세계의 상인들과 고객들은 하나의 상권에 속하게 되었고, 그 결과 상인들과 고객들 사이에는 많은 인증 작업이 필요하게 되었다. 또한, 전자화폐를 사용하므로 상인들은 수많은 사용자들의 전자서명 확인과 전자서명 생성 작업을 빠르게 수행해야 된다. 한편, 이동 통신은 대기중으로 (air interface) 데이타가 전달되어 도청과 가장 (masquerade)에 매우 취약하므로, 상대방에 대한 인증과 암호화가 필수적이다[1,2,3]. 더구나, 단말기의 이동성 때문에 도심지나 휴양지같이 사람들이 많이 모이는 지역에서는 여러 개의 단말기들이 동

† 비 회 원 : 한국과학기술원 전자전산학과
smhong@camars.kaist.ac.kr
†† 정 회 원 : 한국과학기술원 전자전산학과 교수
hyoon@camars.kaist.ac.kr
††† 비 회 원 : 서울대학교 컴퓨터공학부
leesw@ssrnet.snu.ac.kr
yspark@ssrnet.snu.ac.kr
†††† 종신회원 : 서울대학교 컴퓨터공학부 교수
cho@comp.snu.ac.kr
논문접수 : 1999년 3월 30일
심사완료 : 2000년 2월 13일

시에 하나의 기지국에 통신을 요구하게 되고 해당 기지국은 동시에 많은 인증작업을 처리해야 하는 부담을 가지게 된다.

네트웍 상에서 필요한 인증과 전자서명 서비스를 제공하는 시스템을 인증서버라 한다. 인증서버는 사용하는 암호 알고리즘의 종류에 따라 비밀키(secret key) 방식 인증서버와 공개키(public key) 방식 인증서버로 분류한다. 비밀키 암호 알고리즘에서는 인증을 하려는 두 객체는 특정 비밀정보를 공유해야 하며 이를 위한 안전한 채널이 요구된다[4]. (비밀키 시스템에서는 상대방에 대한 인증을 위해서, 상대방이 특정 비밀정보를 알고 있는지를 검사한다.) 따라서, 비밀키 방식 인증서버는 인터넷과 같이 사용자들이 광범위하게 존재하는 환경에는 적합하지 못하다. 반면, 공개키 방식 인증서버는 인증을 위해 공유해야 할 비밀정보가 없고 안전한 채널의 필요성도 없기 때문에 인터넷과 같은 공중망 환경에 더 적합하다.

그러나, 전자상거래나 이동통신과 같이 과중한 인증과 서명이 요구되는 환경에서, 공개키 방식의 인증서버는 전체 시스템 성능의 병목(bottleneck)이 우려된다[5,6]. 그 이유는 모든 시스템 서비스 제공에는 사용자 인증이 필요한데 각 인증요구는 인증서버와의 연결설정에 의한 통신부하(communication overhead)와 과중한 암호연산에 의한 계산부하(computation overhead)를 발생시키기 때문이다.

이러한 인증서버의 병목현상을 줄이기 위한 가장 자연스러운 방법은 인증서버를 여러 개의 플랫폼을 이용해서 구축하는 것이다. 왜냐하면, 플랫폼의 하드웨어를 향상시키는 방법은 상대적으로 비용이 많이 들뿐만 아니라 통신부하의 감소에 전혀 기여를 하지 못하기 때문에, 플랫폼의 개수를 늘리는 것이 계산부하와 통신부하 모두를 감소시키는 경제적인 해결책이다.

그러나, 인증서비스를 제공하는 시스템이 여러 개의 플랫폼에 복제될 경우 인증서버의 개인키(private key) 관리가 무척 어려워진다. 키관리(key management)는 보안 정책에 따른 키(key)의 생성, 저장, 분배, 삭제, 획득 등의 작업으로 정의되며, 키의 불법적인 노출, 변경, 치환, 위조 등의 방지를 목표로 한다[7]. 그런데, 키관리는 알고리즘이나 이론적 연구로 해결될 문제가 아니고 관리시스템의 구조가 가장 중요한 실제적(practical)인 문제이다[8]. 예를 들어, 인증서버의 개인키를 시스템에 저장할 경우 개인키를 어떤 암호알고리즘을 사용하여 암호화할 것인 가도 중요하지만, 암호화된 개인키를 어떻게 안전하게 시스템에 보관할 것인가 하는 문제가 더욱 중요하다. 만약 개인키가 여러 곳에 중복되어 있을 경우에는 중복된 키(key)들의 일관성 문제, 키의 안전한 분배 문제 등으로 키관리가 더욱 어려워진다.

본 논문에서는 추가적인 키관리 비용 없이 인증서버의 병목현상을 해결하는 방법을 제안한다. 제안하는 방법은 여러 대의 플랫폼들로 인증서버를 구축하면서, 인증서버의 개인키는 오직 하나의 플랫폼만이 갖도록 한다. (앞으로 개인키를 가진 플랫폼을 신뢰플랫폼(keyed platform)이라 하고, 가지지 못한 플랫폼을 공플랫폼(empty platform)이라 한다.) 본 논문에서 제안하는 방법은 신뢰플랫폼으로 하여금 서명 생성 시에 공플랫폼의 계산능력을 이용할 수 있도록 하기 위해 SASC (Server-Aided Secret Computation)[9] 프로토콜을 이용한다.

SASC 프로토콜은 기존에 스마트카드의 부족한 계산능력을 보충하기 위해 제안된 것으로서, 많은 종류의 프로토콜들이 제안되어 왔다. 본 논문에서는 SASC 프로토콜을 분산환경에 적용할 수 있도록 모델링 하여인증서버의 구현에 적합하도록 변형하고, 그에 따른 성능향상을 분석한다. 또한, 제안 모델을 구현할 수 있는 구체적인 프로토콜을 설명하고, 적용하는 각 SASC 프로토콜에 따른 성능향상을 비교한다. 제안방법을 이용하면 별도의 추가비용과 키관리 위험 없이 최대 13배까지의 성능향상을 얻을 수 있다. 마지막으로, 본 논문에서 제안하는 방법을 실제로 구현하여 성능향상을 측정한다.

본 논문에서 제안하는 방법은 많은 응용이 가능하다. 제안 방법은 개인키를 알려주지 않으면서 안전하지 못한 컴퓨터들의 계산능력을 이용하기 때문에, 휴지(idle) 상태에 있는 다른 용도의 서버들을 이용할 수 있다. 따라서, 별도의 비용과 키관리 위험을 발생시키지 않으면서 인증서버의 성능을 향상시킬 수 있다. 또한, 현재의 추세에서 볼 때 퇴역된 중고 컴퓨터들(e.g., 386,486, sparc2 등)을 이용해서 저렴하고 안전하게 인증서버의 성능을 향상시킬 수 있다. 또한, 폭주(bursty) 특성을 지닌 네트웍에서 인증요구가 몰릴 때 추가비용 없이 동적으로 인증서버의 계산능력을 늘임으로써 안정적인 인증서비스를 제공할 수 있다.

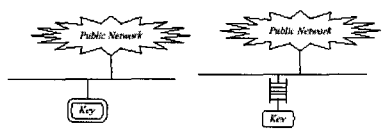
본 논문의 구성은 다음과 같다. 2 장에서는 기존 인증서버 구축 모델들을 성능향상과 키관리 비용 측면에서 장단점을 분석하고, 기존의 SASC기법에 대해 살펴본다. 3 장에서는 제안된 기법의 일반적인 모델을 제시하고, 그 성능을 분석한다. 4 장에서는 기존의 SASC프로토콜을 제안된 모델에 적용시키고 성능을 분석한다. 5 장에서는 4 장에서 설명한 방법들을 구현하여 그 성능을 측

정하고 성능분석 결과와 비교한다. 마지막으로 6 장에서는 제안된 기법의 응용방안을 제시하고 결론을 맺는다.

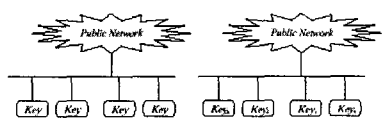
2. 관련 연구

2.1 여러 가지 인증서버 구축 모델들

그림 1은 인증서버의 부하(overhead)를 줄이기 위한 여러 가지 모델들을 나타낸다. 인증서버의 부하(overhead)를 줄이는 방식은 크게 두 가지로 나뉜다. 하나는 인증서버를 구성하는 플랫폼의 계산능력을 늘이는 방식으로 그림 1(a)와 (b)가 이에 해당한다. 다른 하나는 플랫폼의 숫자를 늘이는 방식으로 그림 1(c)와 (d)가 이에 해당한다.



(a) Hardware upgrade (b) Batch processing



(c) Multiple platforms (d) With different keys

그림 1 인증서버의 모델

물론, 그림 1에 나타난 모델들 이외에도, 인증서버의 안전성과 가용성을 동시에 추구하는 많은 다른 연구들이 있다. 예를 들면, 역치(threshold) 기법을 이용한 인증서비스 제공방식[5,10,11]과 인증서버에서 제공하는 인증확인서와 다른 서비스 간의 관련성을 약화시키는 방법이 있다[12]. 전자는 인증서버의 개인키를 여러 개로 나눠서 각 플랫폼들이 지니고 있다가, 일정한 수 이상의 플랫폼들이 모여야 인증서비스를 제공하는 기법이고, 후자는 인증서버가 발급하는 인증서의 유효기간을 길게 하는 방법이다. 그러나, 본 논문에서는 인증서버의 성능을 높이는 방식에 초점을 두고 있기 때문에, 이러한 방식들과는 목표가 다르다. 또한, 이러한 방식들은 본 논문에서 제안하는 모델과 병행해서 사용함으로써 본래의 목적을 추구할 수 있다.

그림 1(a)는 인증서버 기능을 수행하는 플랫폼의 하드웨어를 향상된 기종으로 바꾸는(upgrade) 방법을 나타낸다. 이 경우 인증서버의 계산부하는 감소하지만 통신부하는 전혀 감소하지 않으므로 효과가 적다. 게다가 하드웨어를 향상된 기종으로 바꾸는 비용은 일반적으로 비싸다.

그림 1(b)는 일괄(batch) 암호알고리즘들을 이용함으로써 인증서버의 계산부하를 줄이는 기법을 보여준다[13,14,15,16]. 이 방법 역시 앞의 방법과 같은 이유로 성능 향상의 한계가 있다. 그리고, 일괄처리의 특성 때문에 클라이언트가 기다려야 하는 대기시간이 길어지는 단점을 가진다.

그림 1(c)와 (d)는 여러 개의 플랫폼들을 이용해서 인증서버를 구축하는 방식을 보여준다. 그림 1(c)는 모든 플랫폼들이 동일한 인증서버의 개인키를 가지는 시스템을 나타내고, 그림 1(d)는 여러 개의 플랫폼들이 자신의 고유한 개인키를 가진 시스템을 나타내고 있다. 이처럼 인증서버를 구성하는 플랫폼의 숫자를 늘리는 방식은 인증서버의 계산부하 뿐만 아니라 통신부하도 감소시키고, 성능향상을 위해 투자되는 비용도 상대적으로 적다는 장점을 가지고 있기 때문에, 하나의 플랫폼으로 인증서버를 구성하는 방식보다 좋은 성능을 보인다. 그러나, 플랫폼의 증가는 심각한 키관리 문제를 발생시킨다.

인증서버를 여러 개의 플랫폼들에게 분산시키면, 기본적으로 안전하게 관리해야 할 물리적 대상이 늘어나기 때문에 많은 비용이 요구되며 안전성 또한 약해진다. 게다가, 중복된 키의 일관성 유지 문제, 키의 안전한 분배 문제 등과 같은 새로운 문제들이 나타난다.

키관리(key management)는 보안 정책에 따른 키(key)의 생성, 저장, 분배, 삭제, 획득 등의 작업으로 정의되며, 키의 불법적인 노출, 변경, 치환, 위조 등의 방지를 목표로 한다[7]. 이러한 맥락에서 볼 때, 늘어난 플랫폼들은 다음과 같은 문제들을 해결할 것을 요구한다: (1) 어떻게 안전하게 각 플랫폼을 지킬 것인가? (2) 각 개인키를 어떻게 저장할 것인가? (3) 어떤 방법으로 키들을 일관성 있게 바꿀 것인가? (4) 새로 더해지는 플랫폼에 어떻게 안전하게 키를 전달할 것인가? (5) 클라이언트에게 해당하는 공개키를 어떻게 정확히 전달할 것인가? 이외에도 수많은 문제들이 발생한다. 물론, 플랫폼이 하나일 경우에도 역시 해결해야 하는 문제들이 있지만, 플랫폼의 개수가 늘어날 수록 더욱 복잡해진다.

결국 키관리 측면에서 보면 여러 개의 플랫폼을 이용하는 접근 방법은 매우 비싼 해결책이다. 본 논문에서는 추가적인 키관리 비용을 발생시키지 않으면서 인증서버의 부하를 줄이기 위해 SASC 프로토콜을 이용한다.

2.2 Server-Aided Secret Computation (SASC)

SASC(Server-Aided Secret Computation) 프로토콜은 TM마트카드와 같이 계산능력이 현저히 부족한 장치로 하여금 서버(e.g., 현금지급기, 카드리더 등)의 도움

을 받아 비밀계산을 수행할 수 있도록 하는 프로토콜이다[9]. SASC 프로토콜의 큰 효용성으로 인해 SASC 프로토콜에 대한 많은 연구들이 이루어져 왔다[17,18,19,20,21,22,23,24,25,26]. SASC 프로토콜은 주로 RSA 서명생성을 빠르게 하기 위한 것들이 많으며, DSS (Digital Signature Standard) 서명을 위한 것과 Prover-Aided Verification과 같은 프로토콜들도 있다[27,23]. 본 논문에서 제안하는 모델과 방법론은 기존의 모든 SASC 프로토콜을 이용해서 모든 방식의 인증서버의 성능향상을 얻을 수 있는데, 특히 본 논문에서는 서버의 도움을 받아(server-aided) RSA 서명을 생성하는 프로토콜을 이용해서 제안 방법을 기술한다[1].

우선 일반적인 RSA 서명생성 과정을 설명한다. RSA [28]에서 서명을 생성하기 위해서는, 매우 큰 소수 p,q를 구하고 n=pq를 계산한다. 그리고 나서, $\lambda(n)(=(p-1)(q-1))$과 서로 소인 임의의 정수 e를 고른 후에 $d \cdot e \equiv 1 \bmod \lambda(n)$이 성립하는 d 를 찾는다. 이러한 초기 설정이 끝나면, 서명자는 n,e를 공개하고, d는 자신만이 아는 개인키의 역할을 하게 된다. 임의의 메시지 m에 대한 서명은 $m^d \bmod n$ 이 된다. RSA 서명생성을 위한 SASC 프로토콜의 목적은 클라이언트가 개인키 d를 서버에 알려주지 않으면서 서버로 하여금 $m^d \bmod n$ 을 계산하도록 하는 것이다.

위에서 설명한 RSA 서명생성 과정 중에서, 시스템 계수들 p, q, d, e 등을 설정하는 과정은 클라이언트에서 수행하거나, 안전한 곳에서 비밀리에 수행한 후에 클라이언트에 저장된다고 가정한다. SASC 프로토콜은 클라이언트가 서버에게 일을 전달하는 방식에 따라서, 나눔기법과 감춤기법으로 나누어 볼 수 있다. 우선 나눔기법 프로토콜을 설명한다. 클라이언트는 우선 $d \equiv \sum_{i=0}^{m} a_i x_i \bmod \lambda(n)$이 성립하는 벡터(vector) $X=(x_1,x_2,\cdots,x_m)$와 $A=(a_1,a_2,\cdots,a_m)$를 찾는다. 이들 중 X를 서버에게 전해주고, 서버는 $z_i \equiv m^{x_i} \bmod n$ (where $1 \le i \le m$)들을 계산하여 클라이언트에게 돌려준다. 클라이언트는 $m^d \equiv \prod_{i=1}^{m} z_i^{a_i} \bmod n$을 계산함으로써 RSA 서명을 얻는다. 상기 방식은 Matsumoto, Kato, 그리고 Imai가 [9]에서 제안한 프로토콜로서, 이후에 많은 공격방법들과 해결방법들이 제안되어 왔다. 그들 중 Beguin과 Quisquater[29]의 방법(이후 BQ 기법이라고 한다.)은 가장 최근에 제안된 방법들 중 하나로서 매우 효율적이다. 그러나, 최근 Nguyen과 Stern은 lattice reduction을 이용한 공격방법을 발표하였다[26]. 본 논문에서는 대표적인 나눔기법 프로토콜로서 BQ 기법을 Nguyen과 Stern의 공격방법에 대해 안전하도록 수정하여 사용한다.

Hong, Shin, Lee, 그리고 Yoon은 [25]에서 통신량과 서버의 계산량을 줄이면서 클라이언트의 서명생성을 가속화 할 수 있는 새로운 접근방식(나눔기법)을 제안하였다. 나눔기법은 클라이언트의 비밀정보를 많은 수들로 나누는데 반해 (앞 문단에서 벡터 X의 원소 갯수만큼 나누는데, 이는 응용 시스템에서 필요로 하는 안전도에 따라 달라지며, [29]에서는 약 20~30 개를 추천하고 있다.), 감춤기법은 클라이언트의 비밀정보를 감추기 위해 일련의 수들을 클라이언트의 비밀정보에 곱하고 더한다. 이 방식은 Nguyen과 Stern의 공격방법을 포함한 기존의 모든 공격방법들에 대해 안전하다. 또한, 이 방법은 나눔기법의 방식에 비해 서버의 계산량과 서버와 클라이언트 사이의 통신량이 적어, 본 논문에서 가정하는 분산환경에 더 적합하다. 앞으로 본 논문에서는 Hong 등의 기법을 HSLY 기법이라 부른다.

3. 제안 모델

3.1 일반적 인증절차

그림 2는 일반적인 인증과정을 도식화 한 것이다. 인증서버가 사용자 u로부터인증요청 Σ_u를 받으면 인증서버는 알고리즘 V를 이용해서 사용자의 인증요청 Σ_u가 정당한지를 확인한다. 공개키 인증시스템에서 인증의 주된 작업은 Σ_u가 권한이 있는 사용자의 올바른 서명인지를 확인하는 것이다.

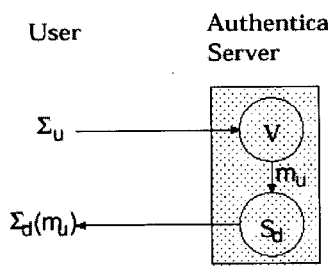


그림 2 일반적인 인증절차

1) RSA는 가장 널리 사용되는 암호 알고리즘이며, 기존의 SASC 프로토콜도 대부분 RSA 서명생성을 가속화하는 데 초점을 맞추어 왔다. 또한, 성능분석 결과에서 예측할 수 있는데, RSA에 대한 프로토콜 이외의 모든 SASC 프로토콜들은 많은 서버의 계산량을 필요로 하기 때문에 얻을 수 있는 성능향상이 매우 적다.

인증작업이 끝나면 인증서버는 사용자 u에 대한 인증확인서 $\sum_d(m_u)$를발급해 주는데, 일반적으로 인증확인서는 인증서버의 개인키에 의한 서명으로 이루어져 있다[30]. 공개키 암호기법(e.g. RSA)에 따른 서명알고리즘을 S_d라고 하면 인증확인서 메세지 m_u에 대한 전자서명이 인증확인서 $\sum_d(m_u)$가 된다.

인증서버의 계산상 병목이 되는 곳은 S_d알고리즘을 수행하는 부분으로서, 현재 널리 사용되는 개인용 컴퓨터나 웍스테이션으로도 1024 혹은 2048 비트의 S_d서명수행은 상당한 계산량을 필요로 한다. 따라서, 본 절에서는 S_d를 계산하기 위해 안전하지 못한 계산기기(computing device)들의 도움을 얻을 수 있는 방법을 제안한다.

3.2 SASC 프로토콜에 대한 개략화 및 분산환경에의 적용

본 절에서는 SASC의 일반적인 모델을 개략화 하여 표현하고, 이를 분산환경에 적용하기 위한 요구사항들을 분석한다. 다음은 일반적인 SASC 프로토콜을 설명한 것이다. SASC 프로토콜의 최종목표는 다음 세 가지 조건을 만족하는 알고리즘 I_d, M, 그리고 F_d를 이용해서, 임의의 메세지 m에 대한 서명 $\sum_d = m_d \bmod n$을 계산하는 것이다.

1. 세 개의 알고리즘 I_d, M, 그리고 F_d의 연속적인 수행은 서명알고리즘 S_d와 같은 출력을 내야 한다. 즉, $\sum_d(m) = F_d(M(I_d(m)))$.

2. 임의의 입력 메세지 m에 대해 알고리즘 I_d와 F_d는 비밀정보 d를 가지고 있어야만 수행할 수 있고, 알고리즘 M은 비밀정보를 몰라도 수행할 수 있어야 한다. 수행순서는 그림 3과 같다.

3. 프로토콜의 수행이 클라이언트의 비밀정보 d를 노출시키지 않아야 한다.

기존의 SASC 프로토콜들은 서버의 계산능력이 클라이언트의 계산능력에 비해 월등히 높은 환경을 가정하기 때문에, 클라이언트에서의 알고리즘 수행시간이 전체 시스템의 성능을 좌우한다. (예를 들면, 스마트카드와 카드리더 혹은 현금지급기 사이의 관계) 따라서, 클라이언트의 계산량을 최소화하는 것이 주된 목표로서 서버의 계산량은 상대적으로 많이 늘어나도록 설계되어 왔다.

그러나, 본 논문에서 목표로 하는 분산서버의 경우에는 각 시스템 (서버와 클라이언트) 간의 계산능력에 큰 차이가 없다고 가정한다. 따라서, 전체 시스템 성능을 결정하는 척도가 달라진다. 즉, 기존에는 클라이언트가 알고리즘 I_d와 F_d를 수행하는 데 필요한 계산량이 주된 성능척도였으나, 분산서버에서는 I_d와 F_d그리고 M을 수행하는 데 필요한 계산량의 합 또한 중요한 요소가 된다. 구체적인 성능분석은 3.4절에서 수행한다.

3.3 다중 플랫폼을 이용한 인증 모델

본 절에서는 추가적인 위험 부담 없이 안전하게 인증서버의 계산부하와 통신부하를 줄일 수 있는 방법을 제안한다. 그림 5에 본 논문에서 제안하는 분산 인증서버 모델이 나타나 있다. 한 대의 신뢰플랫폼에만 서명생성을 위한 인증서버의 개인키가 들어가 있고, 나머지 공플랫폼들에는 들어있지 않다. 신뢰플랫폼과 공플랫폼 모두 사용자의 서명요구를 받아들일 수 있으며, 신뢰플랫폼의 계산부담을 덜기 위해 SASC 프로토콜을 사용한다. (기

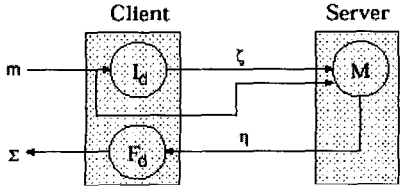
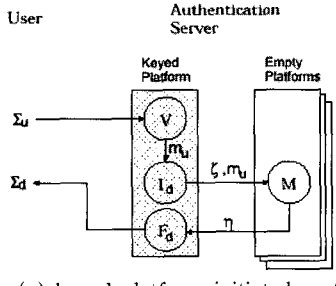
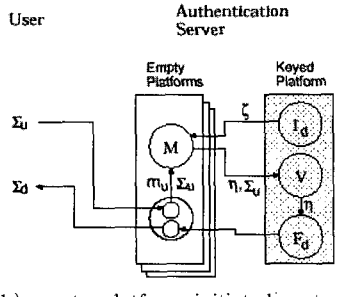


그림 3 SISC 프로토콜의 일반적 모델



(a) keyed platform initiated protocol



(b) empty platform initiated protocol

그림 4 인증서버의 동작방식

존의 SASC 프로토콜들은 나눔기법과 감춤기법 모두 적용 가능하다. 4 절에서는 각 기법을 적용했을 때의 성능향상 정도를 비교 분석한다.) 실제 응용시에는 여러개의 신뢰플랫폼을 두도록 일반화 할 수 있으나, 본 논문에서는 성능분석과 실험을 위해 한 개의 신뢰플랫폼만을 가정한다.

그림 5에 본 논문에서 제안하는 인증과정이 나타나 있다.

그림 5(a)는 신뢰플랫폼으로 들어오는 인증요청들에 대해, 신뢰플랫폼의 계산부담을 덜기 위해, 안전하지 못한 계산기기들(공플랫폼)을 이용하는 절차를 나타낸다. (1절에서 정의했듯이, 인증서버의 개인키를 가지고 있는 플랫폼을 신뢰플랫폼이라 하고, 그렇지 않은 플랫폼들을 공플랫폼이라 한다.) 신뢰플랫폼은 인증절차를 마친 후에, 인증확인서를 발급하기 위해 자신의 서명을 수행할 때 SASC 프로토콜을 이용한다. SASC 프로토콜 중에서 비밀정보를 알지 못해도 수행할 수 있는 알고리즘 M을 공플랫폼들로 하여금 수행하도록 한다.

그림 5(b)는 신뢰플랫폼으로 들어오는 인증요청들의 네트웍 부하를 줄이기 위해서, 공플랫폼들도 인증요청을 받을 수 있도록 할 경우의 인증절차를 보여준다. 공플랫폼들이 인증요청을 받아들이게 할 경우 고려해야 할 점들이 있다. 첫째, 공플랫폼들이 인증절차를 수행하게 되면, 신뢰플랫폼으로서는 공플랫폼의 인증확인 통보를 믿을 수 없다. 왜냐하면, 인증확인 통보가 공플랫폼으로부터 온 것인지 다른 제 3자로부터 온 것인지도 알 수 없고, 공플랫폼이 공격당해 오염(corrupt) 되었을 수도 있기 때문이다. 따라서, 오염되지 않은 공플랫폼으로부터 온 메세지를 다른 메세지들로부터 구분해야 하는 부하(overhead)가 발생하고, 이는 SASC 프로토콜의 사용 자체를 의심스럽게 할만큼 치명적일 수 있다. 따라서 인증 알고리즘 V는 여전히 신뢰플랫폼에서 수행해야 한다. 그러나, V는 간단히 수행될 수 있는 연산이므로 거의 계산부담이 없다[2]. 둘째, 기존의 SASC 프로토콜들은 모두 I_d를 수행하는 데 있어서, 서명하고자 하는 메세지의 값과 관계가 없다. 다만, 매 번 새로 수행하기만 하면 되므로, 공플랫폼-시작(empty platform initiated) 프로토콜을 수행할 때는, I_d 를 신뢰플랫폼이 미리 (아무 일도 하지 않을 때, 혹은 여력이 있을 때) 계산해서 줌으로써, 실제 인증확인서 발행 시에 성능을 높일 수 있다. 마지막으로 공플랫폼이 인증요청을 받아들이게 될 경우에도 공플랫폼과 신뢰플랫폼 사이의 통신은 피할 수가 없으므로 네트웍 부담이 줄지 않는다고 생각할 수 있으나, 그렇지 않다. 왜냐하면, 사용자로부터 인증요청을 받아들이고 인증확인서를 발급할 때의 네트웍 부담은, 항상 연결이 설정되어 있는 공플랫폼들 사이의 통신과는 비교할 수 없을 만큼 크기 때문이다.

3.4 성능분석

서버의 개인키 d에 의한 전자서명을 수행하는 알고리즘을 S_d라하고, 임의의 알고리즘 A를 수행하는 데 걸리는 시간을 T(A)로 표현한다. 계산기기의 계산능력을 μ로 표현하고, $\sum$, S_d, I_d, M, 그리고 F_d는 모두 3.2절에서 사용한 것과 같은 의미를 지닌다.

SASC 프로토콜을 사용하지 않는 단일 서버에 의한 성능은 다음과 같이 표현할 수 있다. 사용자들의 모든 인증요구는 정당하다고 가정하며, T(V)=0으로 놓는다.

$$P_1 = \frac{\mu_d}{T(S_d)}$$

본 논문에서 제안하는 방법을 이용할 경우의 성능향상을 계산한다. 신뢰플랫폼의 계산능력은 μ_d로 표현하고, 공플랫폼들 전체 계산능력의 총합은 μ_ϕ로 표현한다. 서명수행을 요구하는 여러 개의 서명요구들이 있을 때, 신뢰플랫폼은 공플랫폼들에게 사전계산한 결과를 전달해 주고, 자신은 남는 시간에 S_d를 수행할 수 있다. 이 때, 신뢰플랫폼이 SASC 프로토콜을 이용하지 않고 자신이 직접 서명을 수행하는 서명요구의 비율을 α라하고, SASC 프로토콜을 이용해서 공플랫폼의 도움을 얻는 서명요구의 비율을 β로 표현한다. 공플랫폼들과 신뢰플랫폼 모두에서 알고리즘 수행이 종료되어야 모든 서명요구들이 해결될 수 있으므로, 성능향상은 다음과 같다.

$$P' = \min\left\{\frac{\mu_d}{T_{KP}}, \frac{\mu_\phi}{T_{EP}}\right\},$$
$$where\ T_{KP} = \alpha T(S_d) + \beta(T(I_d) + (F_d)),$$
$$T_{EP} = \beta T(M),$$
$$\text{and}\ \alpha + \beta = 1.$$

위 식에서 T_{KP}는 신뢰플랫폼이 자신의 작업을 모두 마치기 위해 필요한 시간이며, T_{EP}는 공플랫폼들이 각자의 작업을 모두 마치기 위해 필요한 시간을 나타낸다. 이 때, 신뢰플랫폼과 공플랫폼들 모두가 최대한 활용

2) RSA의 경우에, 공개키는 작은 수(e.g., 3, 7, 또는 65535)로 정하는 것이 일반적이다. 이는 몇 번의 모듈라 곱셈으로 서명인증을 가능하게 한다. 심지어, MSR(Modulo Square Root) 알고리즘의 경우에는 모듈라 곱셈 한 번으로 서명인증이 가능하다. 서명생성 시에는 수백 번의 모듈라 곱셈이 필요하므로, 서명인증은 상대적으로 무시할 만큼의 적은 시간만을 필요로 한다.

되도록 α와 β를 결정해야 인증서버는 최대의 성능을 발휘한다. 따라서 $\frac{\mu_d}{T_{KP}}$와 $\frac{\mu_\phi}{T_{EP}}$가 같아질 때, 최대성능을 얻을 수 있다. 그 때의 α_{opt}와 β_{opt}는 다음과 같다.

$$\alpha_{opt} = 1 - \beta_{opt},$$

$$\beta_{opt} = \frac{\mu_\phi T(S_d)}{\mu_\phi T(S_d) + \mu_d T(M) - \mu_\phi (T(I_d) + T(F_d))}$$

신뢰플랫폼과 공플랫폼의 성능비 μ_ϕ / μ_d를 μ_δ로 표현하면, α와 β를 α_{opt}와 β_{opt}로 설정했을 때의 성능은 다음과 같다.

$$P'_{opt} = (1 + \frac{T(S_d) - T(I_d) - T(F_d)}{T(M)} \mu_\delta) \times P_1.$$

공플랫폼들의 전체 성능합인 μ_ϕ가 크면 클수록 얻을 수 있는 성능향상도 커진다. 그러나, SASC 프로토콜을 이용하더라도 신뢰플랫폼이 수행해야 할 계산량이 재하기 때문에, μ_ϕ가 무한히 커지는 것은 의미가 없다. 왜냐하면, μ_ϕ가 어느 이상 커지게 되면, 공플랫폼들이 계산능력이 아무리 커도, 신뢰플랫폼에서 병목(bottleneck)이 되어 공플랫폼들은 아무 일도 하지 않고 신뢰플랫폼만을 기다리고 있게 되기 때문이다. 따라서, 공플랫폼들에서 M을 계산하는데 걸리는 시간이 신뢰플랫폼에서 사전계산과 사후계산을 하는 시간보다 오래 걸릴 경우에만 공플랫폼들의 계산능력 증가가 의미가 있다. 즉, $\frac{T(M)}{\mu_\phi} \geq \frac{T(I_d) + T(F_d)}{\mu_d}$인 상황에서만이 공플랫폼들의 계산능력 향상이 의미가 있다. 따라서, 의미 있는 공플랫폼들의 최대 계산능력은 다음과 같다.

$$\mu\delta_{\max} = \frac{T(M)}{T(I_d) + T(F_d)}$$

공플랫폼들의 계산능력이 충분할 때의 성능향상을 계산한다. 공플랫폼의 계산능력이 $\mu\phi_{\phi\max}$일 때는 β_{opt}는 1이 되고, α_{opt}는 0이 된다. 즉, 공플랫폼의 계산능력이 충분할 때는, 신뢰플랫폼이 I_d와 F_d만을 수행하고, S_d를 수행하지 않는 것이 가장 효율적이 된다. 요약하면, 충분한 계산능력의 공플랫폼들이 의존하면, 안전성에 전혀 영향을 미치지 않으면서, 다음과 같은 성능향상을 얻을 수 있다.

$$P'_{\max} = \frac{T(S_d)}{T(I_d) + T(F_d)} \times P_1$$

4. Specific Scheme

본 절에서는 3절에서 제안한 모델을 이용해서 인증서버를 구현하기 위한 프로토콜들에 대해 설명한다. 기존의 SASC 프로토콜들 중 다음 세 가지 프로토콜을 이용해서 설명한다: (1) HSLY 기법, (2) 변형된 HSLY 기법, (3) Nguyen과 Stern의 공격[26]에 이겨낼 수 있도록 수정된 BQ 기법. 모두 RSA 서명생성을 위한 알고리즘이고, 피연산자는 1024비트로 가정한다.

4.1 HSLY 기법

HSLY 기법을 사용하기 위해서는 사전계산이 필요하다. 이는 인증서버의 개인키 d를 생성할 때 단 한번만 수행하면 된다. 신뢰플랫폼은 d를 t'으로 변형하고, u를 계산해 둔다:

$$t' \equiv \frac{1}{r'_k}(\cdots(\frac{1}{r'_1}(d - r_1) - r_2) - \cdots - r_k) - R \bmod \lambda(n),$$

그리고 $u \equiv \prod_{i=1}^{k} \frac{1}{r'_i} \bmod \lambda(n)$. t'과 u를 계산하는데 필요한 난수들 r_i, r'_i, R, 그리고 R'을 r_i, r'_i, R, 그리고 R'을 선택하는 방법은 [25]에 상세히 설명되어 있다. 신뢰플랫폼은 중국인의 나머지 정리(Chinese Remainder Theorem:CRT) 적용을 위해 $w_p \equiv q(q^{-1} \bmod p) \bmod n$과 $w_q \equiv p(p^{-1} \bmod q) \bmod n$을 미리 계산해 저장해 둔다. 그림 5에 각 모듈의 알고리즘이 나타나 있다.

Algorithm I_d (no input, output ζ : $\{t, \sigma_p, \sigma_q\}$)
  randomly choose d_1.
  compute t :
    $t = t' - u \times d_2 \bmod \lambda(N)$,
      where $d_2 = d - d_1$
  compute σ_p and σ_q :
    $\sigma_p = d_2 \bmod (p-1) + \varrho_p(p-1)$,
      where $\varrho_p \in_R \{0, \ldots, q-2\}$, and
    $\sigma_q = d_2 \bmod (q-1) + \varrho_q(q-1)$,
      where $\varrho_q \in_R \{0, \ldots, p-2\}$.

Algorithm M (input ζ, output η : $\{Z, y_p, y_q\}$)
  compute Z :
    $Z = M^t \bmod n$
  compute y_p and y_q :
    $y_p = M^{\sigma_p} \bmod N$
    $y_q = M^{\sigma_q} \bmod N$

Algorithm F_d (input η, output $\sum_d$: S)
  compute z_p and z_q :
    $z_p = (\cdots((Z \bmod p \times M_p^R)^{r_k} \times M_p^{r_{k-1}} \cdots)^{r_1} \times M_p^{r_1} \bmod p$
    $z_q = (\cdots((Z \bmod q \times M_q^R)^{r_k} \times M_q^{r_{k-1}} \cdots)^{r_1} \times M_q^{r_1} \bmod q$
  compute $S = w_p S_p + w_q S_q \bmod N$, where
    $S_p = y_p z_p \bmod p$ and $S_q = y_q z_q \bmod q$.
  if $S^v \bmod N = M$, then transmit S.

그림 5 HSLY 기법을 이용한 서명생성 프로토콜

HSLY 기법을 사용했을 때의 성능을 살펴본다. 피연산자들의 크기를 1024비트로 가정했으므로, 1024비트 모듈라 곱셈을 한 번 수행하는 데 걸리는 시간을 τ로 놓으면, $T(S_d)$는400τ가 된다[3]. T(I_d)는 1.5τ이고,T(M)은 1703τ가 되며, $T(F_d)$는 보안인수 설정에 따라서 그 값이 달라지는데, $\langle b_{R=11}, b_R=26, k=3\rangle$으로 설정하면 35τ가 된다. 표 1에 HSLY 기법을 이용함으로써 얻을 수 있는 성능향상이 나타나 있다.

표 1 HSLY 기법을 이용한 인증서버에서 사용 가능한 공플랫폼의 계산능력에 따른 성능향상

μδ	0	1	2	5	10	53
P_{opt}	1	1.22	1.43	2.08	3.16	12.31

예를 들어, 1초에 1024비트 모듈라 곱셈을 200회 수행할 수 있는 신뢰플랫폼과 같은 성능을 지니는 공플랫폼을 생각한다. μ_d와μ_ϕ는 모두 200 (회/sec.) 으로서, μ_ϕ/μ_d는 1이다. 이 때, α_{opt}는 0.8627이 되고, β_{opt} 는 0.1373이 된다. 그 때의 성능은 0.571 sign/sec.이 된다. 즉, 1개의 서명을 생성하는데 평균 1.75초가 걸림을 의미한다. 신뢰플랫폼 혼자서는 2 초가 걸릴 일이므로, 같은 성능의 공플랫폼 한대를 이용함으로써 속도가 15% 빨라졌다. 이는 주변의 아무 플랫폼이나 사용 가능한 계산기기를 이용해서 안전성이 침해받지 않고, 키관리 문제에 부하를 발생시키지 않으면서 15%만큼 인증서버의 성능향상을 얻을 수 있음을 의미한다. 공플랫폼들의 총 계산능력이 충분한 상황에서는 최고 13배 이상의 성능향상을 얻을 수 있다. 즉, 같은 신뢰플랫폼을 이용해서 안전성의 침해를 받지 않고, 1개의 서명을 생성하는데 걸리는 시간을 평균 2초에서 0.15초로 줄일 수 있다.

4.2 변형된 HSLY 기법

지금까지 기술한 프로토콜은 신뢰플랫폼이 수행해야 하는 계산량이 상당히 적기 때문에, 공플랫폼들의 계산능력이 충분히 클 때 ($\mu_\phi \gg \mu_d$), 높은 성능향상을 얻을 수 있다. 그러나, 공플랫폼의 계산능력이 충분하지 못한 환경에서는, 공플랫폼이 수행해야 하는 계산량도 성능향상의 중요한 요소가 된다. 따라서, 기존의 HSLY 기법을 변형하여 신뢰플랫폼의 계산량을 늘림으로써 공플랫폼의 계산량을 줄일 수 있도록 하는 방법을 제안한다.

프로토콜에 필요한 사전계산은 동일하다. 다만, u를 만들어 준비할 필요가 없다. 또한, 알고리즘 I_d 또한 별도로 수행할 필요가 없이 사전계산을 통해 계산된 t를 공플랫폼에게 전해주기만 하면 된다. 그림 6에 변형된 HSLY 기법을 이용한 알고리즘들이 나타나 있다.

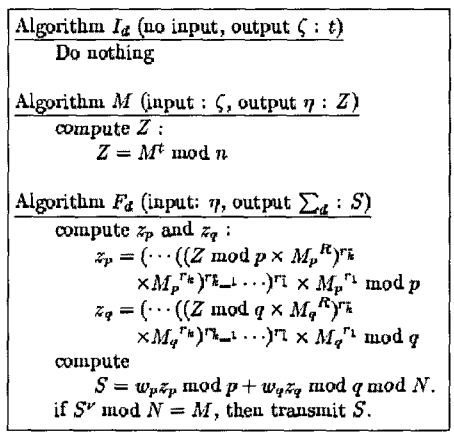


그림 6 HSLY 기법을 변형한 서명생성 프로토콜

이러한 변형을 가하더라도 Nguyen과 Stern의 공격을 포함한 대부분의 공격에 대해서는 여전히 안전하다. 그러나, continued fraction과 Pfitzmann과 Waidner의 공격[20]이 가능해지기 때문에, 보안인수들을 선택할 때 이러한 공격들에 이겨낼 수 있도록 선택해야 한다. 알려진 모든 공격들에 대해 안전하도록 하면서 좋은 성능을 내는 보안인수들 중 $\langle b_{R=50}, b_R=78, k=3\rangle$을 선택하면 $T(F_d)$는 88τ가 된다. 표 2에 변형된 HSLY 기법을 이용함으로써 얻을 수 있는 성능향상이 나타나 있다.

표 2 변형된 HSLY 기법을 이용한 인증서버에서 사용 가능한 공플랫폼의 계산능력에 따른 성능향상

μ_δ	0	1	2	5	10	14
P_{opt}	1	1.26	1.52	2.30	3.60	4.54

원래의 HSLY 기법을 이용했을 경우와 비교하면 같은 성능의 공플랫폼들을 이용할 때 얻을 수 있는 성능향상이 커졌다. 예를 들어, 신뢰플랫폼과 같은 능력의 공플랫폼 1대를 덧붙였을 경우, 원래의 HSLY 기법을

3) CRT를 사용한다고 가정한다. p와 q를 알고 있으므로, $m^{d \bmod (p-1)} \bmod p$와 $m^{d \bmod (q-1)} \bmod q$를 계산하여 CRT를 적용하면 되므로 총 1536번의 512비트 모듈라 곱셈을 수행하면 된다[31]. 모듈라 곱셈은 $O(n^2)$알고리즘이므로 약 400번의 모듈라 곱셈을 수행하는 만큼의 시간이 걸린다.

이용하면 21.5%의 성능향상을 얻을 수 있는데 반해, 변형된 HSLY 기법을 이용하면 26.0%의 성능향상을 얻을 수 있다. 그러나, 변형된 HSLY 기법은 확장성에 있어서 떨어지기 때문에 최대한 4.5배의 성능향상밖에 얻을 수 없다. 따라서, 사용 가능한 공플랫폼의 계산능력이 적은 경우에 유용한 프로토콜이다.

4.3 BQ 기법

BQ 기법도 쉽게 적용할 수 있다. 다만, 앞에서 언급했듯이 Nguyen과 Stern의 공격을 피하기 위해서는 프로토콜의 사전계산 알고리즘 I_d가 바뀌어야 한다. 따라서, 공플랫폼이 수행해야 할 계산량이 원래의 BQ 기법과 다르게 계산되어야 한다.

Nguyen과 Stern의 공격을 피하는 방법과 그 때의 계산량 변화를 설명한다. 사용하는 기호들은 2.2절에서 설명한 것을 이용한다. 원래의 프로토콜에서는 클라이언트가 임의의 x_i를 선택할 때, x_i들의 비트수 $l(x_i)$들을 $0 \le l(x_i) \le t - \log_2(mh) - 2$의 범위에 있도록 제약을 가한다. 이는 x_i 를 p - 1과 q - 1보다 작게 만들어 결과적으로 512비트 이하의 숫자들이 되도록 만들었다. 그러나 Nguyen과 Stern의 공격을 피하게 하기 위해서는 x_i들의 범위를 $0 \le x_i \le \lambda(n)$으로 바꾸어야 한다. 따라서, $l(x_i)$의 크기는 1024비트로 늘어나고, 이는 서버의 계산량과 클라이언트와 서버 사이의 통신량을 늘어나도록 한다. 따라서, 같은 보안인수 설정을 사용할 경우 T(M)은 7165 τ가 된다. 그러나, Nguyen과 Stern의 공격을 고려하더라도 $T(F_d)$는 변하지 않으므로 35τ가 된다.

BQ 기법의 경우에는 HSLY 기법과 다르게 보안인수를 바꿈으로써, 같은 안전성을 유지하면서 서버와 클라이언트의 계산량의 비율을 비교적 자유롭게 바꿀 수 있다. 그러나, 그로 인해 얻을 수 있는 성능향상 차이는 미미하다. 본 논문에서는 BQ 기법을 이용한 구체적인 프로토콜 기술은 생략한다.

표 3 각 방법들의 성능향상 비교

	BQ	HSLY	modified HSLY
$T(S_d)/\tau$	400		
$T(I_d)/\tau$	0	1.5	0
$T(M)/\tau$	7165	1703	1198
$T(F_d)/\tau$	35	31	88
ΔP	0.051 μ_δ	0.216 μ_δ	0.260 μ_δ
$\mu_{\phi_{max}}$	204.7 μ_d	52.4 μ_d	13.6 μ_d
P'_{max}	11.43	12.31	4.54

4.4 비교평가

본 절에서는 지금까지 기술한 3개의 프로토콜에 대해 성능을 비교한다. 표 4에 HSLY 기법, BQ 기법, 그리고, 변형된 HSLY 기법의 성능을 비교한 결과가 정리되어 있다. 표 4에서 ΔP는 활용 가능한 공플랫폼들의 계산능력에 따른 성능향상을 나타내며 다음과 같이 계산된다: $\Delta P = (P'_{opt} - P_1)/P_1 \times \mu_{\phi_{max}}$는 활용 가능한 공플랫폼들의 최대능력으로서 $\mu_{\phi_{max}} = \mu_{\delta_{max}}/\mu_d$로 표현된다.

5. 실 험

본 논문에서는 제안 방식들 중 두 가지를 구현하였다. 하나는 HSLY 방식이고, 다른 하나는 변형된 HSLY 방식이다. C언어를 사용하여 웍스테이션 상에서 구현하였고, 사용한 컴파일러는 gcc version 2.8.1 이다. 사용한 웍스테이션들 중 하나는 신뢰플랫폼으로 동작하고 나머지 웍스테이션들은 공플랫폼으로 동작하도록 프로그래밍 되었다. 모듈라 멱승의 빠른 수행을 위해서, 신뢰플랫폼은 small-window 알고리즘[32]을 사용하고 공플랫폼은 BGMW 방법[33]을 사용하였다. 사용한 피연산자들의 크기는 1024비트이다. 공플랫폼들이 여러 개의 요구들을 받아들여 동시에 처리할 수 있도록 하고, 신뢰플랫폼 또한 사용자의 요구들을 동시에 처리할 수 있도록 하기 위해 다중 쓰레드 프로그래밍을 사용하였다.

두 개의 각각 다른 환경에서 실험하였다. 표 4는 첫 번째 실험환경에서의 실험결과를 보인다. 첫 번째 실험환경은 Sun SparcStation4 (110 MHz, 32 Mbytes) 웍스테이션 21대를 이용한다. 이 때 모든 웍스테이션들은 같은 LAN(Local Area Network)상에서 이더넷으로 연결되어 있다. 표 5는 두 번째 실험환경에서의 결과를 나타낸다. 두 번째 실험에서는, 같은 LAN 상에 있지 않은 웍스테이션들을 사용하였다.

그림 7(a)와 (b)는 각각 표 4와 5를 가시화하고 있다. 그림 7(a)는 제안 프로토콜을 사용할 때의 성능향상 정도 (P'_{opt})를 이론적인 값과 실험 값에 대해 비교하여 나타내고 있다. 이 때의 성능향상 정도는 공플랫폼의 신뢰플랫폼에 대한 성능비 (μ_δ)에 따라 달라진다. 실험결과를 나타내는 곡선은 이론적인 최적 곡선과 매우 근접하며, 그 차이는 네트워과 운영체제에서 발생하는 부하에 기인한다. 그림 7(a)는, 공플랫폼과 신뢰플랫폼 사이의 성능비가 15 이하일 경우에는, 변형된 HSLY 방법이 원래의 HSLY 방법에 비해 나음을 보여준다. 그림 7(b)는 공플랫폼들이 신뢰플랫폼과 같은 LAN에 있지 않더라도 전체 인증서버의 성능향상에 영향을 거의 미치지

않음을 나타낸다.

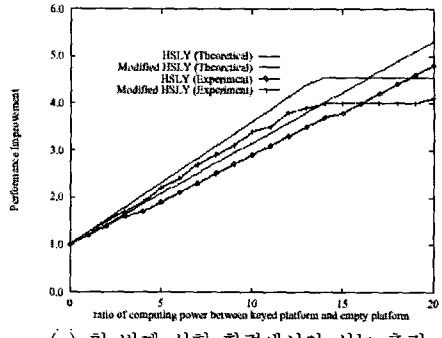
표 4 첫 번째 실험환경에서의 각 방법들의 구현결과. '# sign.' 행은 200초동안 수행한 서명의 개수를 나타낸다. 'F.A.'는 'factor of acceleration'의 약어로 P'opt/P1의 실험치를 나타내고 'opt'는 이론치를 나타낸다.

μδ	HSLY 기법			modified HSLY		
μδ	# sign.	F.A.	opt.	# sign.	F.A.	opt.
0	109.46	1.0	1.0	109.22	1.0	1.0
1	128.13	1.2	1.2	136.22	1.2	1.3
2	148.01	1.4	1.4	161.19	1.5	1.5
3	170.53	1.6	1.6	189.23	1.7	1.8
4	186.01	1.7	1.9	209.58	1.9	2.0
5	211.98	1.9	2.1	241.45	2.2	2.3
6	233.01	2.1	2.3	266.45	2.4	2.6
7	253.00	2.3	2.5	293.23	2.7	2.8
8	272.66	2.5	2.7	319.50	2.9	3.1
9	294.13	2.7	2.9	343.41	3.1	3.3
10	317.17	2.9	3.2	367.59	3.4	3.6
11	339.10	3.1	3.4	386.88	3.5	3.9
12	360.31	3.3	3.6	416.42	3.8	4.1
13	381.30	3.5	3.8	427.92	3.9	4.4
14	403.10	3.7	4.0	431.54	4.0	4.5
15	416.18	3.8	4.2	436.24	4.0	4.5
16	437.10	4.0	4.5	433.54	4.0	4.5
17	463.02	4.2	4.7	438.75	4.0	4.5
18	484.07	4.4	4.9	441.66	4.0	4.5
19	506.20	4.6	5.1	441.26	4.0	4.5
20	528.11	4.8	5.3	444.47	4.1	4.5

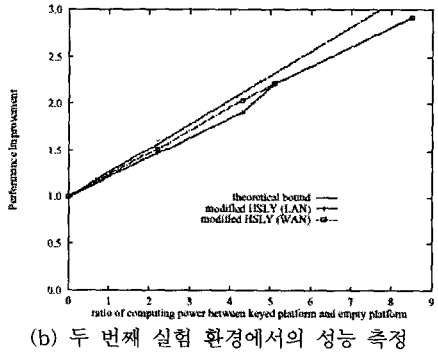
표 4에서 보면 신뢰플랫폼과 같은 능력의 공플랫폼 1대를 이용할 경우의 성능향상이 사용하는 프로토콜에 따라 5%-26%정도가 된다. 성능향상이 100%에 크게 못 미치는 이유는, 플랫폼이 개인키를 가지고 있으면 CRT를 이용하여 빠르게 계산할 수 있는데 반해 개인키가 없으면 CRT를 이용할 수 없기 때문이다. 네트웍에 연결된 안전하지 못한 플랫폼들을 최대로 활용하면 13배까지의 성능향상을 기대할 수 있다.

6. 결 론

본 논문에서는 부가적인 키관리 문제를 야기시키지 않으면서 여러 대의 플랫폼을 이용해 강력한 인증서버를 구축할 수 있는 모델을 제안하였다. 그리고, 우리는 우리의 모델을 구현하는 구체적인 방법을 제시하였고 그 성능향상을 분석하였다. 구현과 실험은 우리의 분석 결과를 확인시켜 주었다. 또한, 우리는 네트웍과 운영체제에 의한 과부하는 본 논문에서 제안하는 모델의 성능향상에 큰 영향을 미치지 않음을 알 수 있었다. (심지어, 공플랫폼들이 같은 LAN에 있지 않더라도) 기존의 분산 인증서버들은 매우 큰 키관리 비용 때문에 구현이 힘들었으나, 우리의 모델은 별도의 키관리 비용 없이 인



(a) 첫 번째 실험 환경에서의 성능 측정



(b) 두 번째 실험 환경에서의 성능 측정

그림 7 실험결과

표 5 변형된 HSLY 기법에 대한 두 실험환경에서의 성능향상 비교. 'LAN'은 모든 공플랫폼들이 같은 LAN에 물려 있는 경우를 나타내고, 'WAN'은 모든 공플랫폼들이 각각 다른 LAN에 속해 있는 경우를 의미한다. 각 항목들은 200초 동안 생성한 서명 개수를 나타낸다. 또한, 괄호 안의 내용은 표 4에서의 'F.A.'와 같은 의미를 표현한다.

	modified HSLY		
μ_δ	LAN	WAN	opt.
1.0	109.2(1.00)	109.2(1.00)	109.2(1.00)
2.2	161.2(1.47)	166.3(1.52)	173.0(1.57)
4.3	209.6(1.91)	223.4(2.04)	233.2(2.21)
5.1	241.4(2.21)	242.4(2.21)	256.1(2.33)
8.5	319.5(2.92)	319.4(2.92)	353.5(3.21)

증서버로 하여금 믿을 수 없는 플랫폼들을 사용할 수 있도록 한다. 따라서, 제안 모델과 방법들은, 공중망에서 강력한 분산 인증서버를 구축하고자 할 때, 다음과 같은 중요한 문제들에 대한 해답을 제시할 수 있다: (1) 부가적인 키관리 비용 없이 어떻게 인증서버의 계산능력은 안전하게 증가시킬 수 있을 것인가? (2) 안정적인 인증서비스 제공을 위해 어떻게 놀고있는 믿을 수 없는 플랫폼들을 이용할 수 있을 것인가? (3) 여러 개의 플랫폼들을 이용해 인증서버를 구축할 때, 안전성과 성능의 절충점을 어떻게 찾을 것인가? (4) 느리고, 오래된 컴퓨터들을 이용해서 강력한 인증서버를 구현할 수 있을 것인가?

참 고 문 헌

[1] C. Boyd and A. Mathuria, "Key establishment protocols for secure mobile communications: A selective survey," in ACISP'98, Lecture Notes in Computer Science, vol. 1438, pp. 344~355, 1998.

[2] K.Vedder, "Security aspects of mobile communications," in Computer Security and Industrial Cryptography, LNCS 741, pp. 193~210, Springer Verlag, 1993.

[3] S. Lee, S.-M. Hong, H. Yoon, and Y. Cho, "Accelerating key establisment protocol in mobile communication," in to be appeared in LNCS series - Information Security and Privacy, 1999.

[4] W.Diffie and M.E.Hellman, "New direction in cryptography," IEEE Trans. Computers, vol. IT-22, pp. 644~654, June 1976.

[5] L. Gong, "Increasing availability and security of an authentication service," IEEE Journal on Selected Areas in Communications, vol. 11, no. 5, pp. 657~662, 1993.

[6] U. Blumenthal, N.C.Hien, and J.H.Rooney, "Low-cost secure server connection with limited-privilege clients," in ACISP'98, Lecture Notes in Computer Science No.1438 (C. Boyd and E. Dawson, eds.), pp. 90~98, 1998.

[7] ISO, Open systems interconnection reference model-part 2: Security architecture," ISO International Standard 7498-2, 1988.

[8] W. Fumy and P. Landrock, "Principles of key management," IEEE journal on Selected Areas in Communications, vol. 11, no. 5, pp. 785~793, 1993.

[9] T.Matsumoto, K.Kato, and H.Imai, "Speeding up secret computations with insecure auxiliary devices," in Crypto'88, pp. 497~506, 1988.

[10] A. Shamir, "How to share a secret," Communications of the ACM, vol. 22, no. 11, pp. 612~613, 1979.

[11] G.R.Blakley, "Safeguarding cryptographic keys," in Proceedings of National Computer Conference (AFIPS'79), pp. 313~317, 1979.

[12] M.P.Herlihy and J.D.Tygar, "How to make replicated data secure," in Advances in Cryp-tology - Proceedings of Crypto'87, LNCS 293, pp. 379~391, 1987.

[13] M. Bellare, J. A. Garay, and T. Rabin, "Fast batch veri cation for modular exponentiatio and digital signature," in Advances in Cryptology - EUROCRYPT'98, LNCS1403, pp. 236~250, Springer Verlag, 1998.

[14] A. Fiat, "Batch RSA," Journal of Cryptology, vol. 10, no. 2, pp. 75~88, 1997.

[15] Y. Yacobi and M. J. Beller, "Batch diffe-hellman key agreement systems," Journal of Cryptology, vol. 10, no. 2, pp. 89~96, 1997.

[16] D. M'Raihi and D. Naccache, "Batch exponentiation - a fast dlp based signature generation strategy," in 3rd ACM Conference on Computer and Communications Security, pp. 58~61, ACM, 1994.

[17] S.-M.Yen, "Cryptanalysis of secure addition chain for sasc applications," Electronics Letters, vol. 31, no. 3, pp. 175~176, 1995.

[18] S.-M.Yen and C.-S.Laih, "More about the active attak on the server-aided secret computation protocol," Electronics Letters, vol. 28, no. 24, p. 2250, 1992.

[19] R.J.Anderson, "Attack on server assisted authentication protocols," Electronics Letters, vol. 28, no. 15, p. 1473, 1992.

[20] B.P tzmann and M.Waidner, "Attacks on protocols for server-aided RSA computation," in Eurocrypt'92, pp. 153~162, 1992.

[21] C.H.Lim and P.J.Lee, "Security and performance of server-aided RSA computation protocols," in Crypto'95, pp. 70~83, 1995.

[22] J.Burns and C.J.Mitchell, "Parameter selection for server-aided RSA computation schemes," IEEE Trans. on Computers, vol. 43, no. 2, pp. 163~174, 1994.

[23] C.H.Lim and P.J.Lee, "Server(prover/signer)-aided verification of identity proofs and signature," in Eurocrypt'95, pp. 64~78, 1995.

[24] S.Kawamura and A.Shimbo, "Fast server-aided secret computation protocols for modular exponentiation," IEEE JSAC, vol. 11, no. 5, pp. 778~784, 1993.

[25] S.-M. Hong, J.-B. Shin, H.Lee-Kwnag, and H. Yoon, "A new approach to server-aided secret

computation," in International Conference on Information Secuirty and Cryptology (ICISC'98), pp. 33~45, 1998.
[26] P. Nguyen and J. Stern, "The beguin-quisquater server-aided RSA protocol from crypto'95 is not secure," in Advances in Cryptology - Asiacrypt'98, LNCS 1514, pp. 372~379, Springer Verlag, 1998.
[27] P. Beguin and J.-J. Quisquater, "Secure acceleration of DSS signatures using insecure server," in Asiacrypt'94, pp. 249~259, 1994.
[28] R.L.Rivest, A.Shamir, and L.Adleman, "A method for obtaining digital signatures and public key cryptosystems," CACM, vol. 21, pp. 120~126, 1978.
[29] P.Beguin and J.J.Quisquater, "Fast server-aided RSA signatures secure against active attacks," in Crypto'95, pp. 57~69, 1995.
[30] CCITT (Consultative Committee on International Telegraphy andTelephony), Recommen-dation X:509: The Directory|Authentication Framework, 1988.
[31] J.-J.Quisquater and C.Couvreur, "Fast decipherment algorithm for RSA public-key cryptosystem," Electronics Letters, vol. 18, no. 21, pp. 905~907, 1982.
[32] D.E.Knuth, The art of computer programming Vol.2. Addition-Wesley,Inc., 1981.
[33] E. F.Brickell, D. M.Gordon, K. S.McCurley, and D. B.Wilson, "Fast exponentiation with precomputation," in Advances in Cryptology - Eurocrypt'92, LNCS 658, pp. 200~207, 1993.

홍 성 민
1994년 한국과학기술원 전산학과 학사. 1996년 한국과학기술원 전산학과 석사. 1996년 ~ 현재 한국과학기술원 전자전산학과 박사과정. 관심분야는 암호학, 암호시스템의 고속구현, 네트웍 보안

윤 현 수
1979년 서울대학교 전자공학과 학사. 1981년 한국과학기술원 전산학과 석사. 1981년 ~ 1984년 삼성전자 연구원. 1988년 오하이오 주립대학 전산학 박사. 1988년 ~ 1989년 AT&T Bell Labs. 연구원. 1989년 ~ 현재 한국과학기술원 전자전산학과 교수. 관심분야는 상호연결 네트워크, ATM switch, 병렬 컴퓨터 구조임

이 승 원
1995년 한국과학기술원 전산학과 학사. 1997년 서울대학교 컴퓨터공학과 석사. 1997년 ~ 현재 서울대학교 컴퓨터공학부 박사과정. 관심분야는 네트워크 보안, 암호 응용; 암호프로토콜등임

박 용 수
1996년 한국과학기술원 전산학과 학사. 1998년 서울대학교 컴퓨터공학과 석사. 현재 서울대학교 컴퓨터공학부 박사과정. 관심분야는 네트워크 보완, 보완 알고리즘

조 유 근
1971년 서울대학교 공대 졸업. 1978년 미네소타대학교 전산학과 박사. 1979년 ~ 현재 서울대학교 컴퓨터공학부 교수. 관심분야는 알고리즘, 운영체제, 데이타구조 등