RSA 서명생성을 위한 새로운 SASC(Server-Aided Secret Computation) 프로토콜 *

신 준범*, 홍 성민**, 이 광형**, 윤 현수**, 한 상근*
* 한국과학기술원 수학과
** 한국과학기술원 전산학과

A New Server-Aided Secret Computation(SASC) Protocol for RSA Signature Generation

J.B.Shin*, S.M.Hong**, K.H.Lee**, H.Yoon**, S.G.Hahn*
* Department of Mathematics, KAIST
** Department of Computer Science, KAIST

요약 : SASC(Server-Aided Secret Computation) 프로토콜은 클라이언트(스마트 카드)의 비밀정보를 공개하지 않으면서 서버(untrusted auxiliary device)에게 서명을 생성하는데 도움을 받도록 하는 프로토콜이다. RSA 서명을 위한 최초의 SASC 프로토콜은 RSA-S1 으로서 그 실효성이 크기때문에, 그 후로 SASC 프로토콜에 대한 연구가 많이 이루어져 왔다. 기존의 SASC 프로토콜들의 공통된 접근방식은, 클라이언트의 비밀정보를 서버로부터 감추기 위해, 비밀정보를 여러 조각으로 나누어 일부분만을 서버에게 전달하는 방식이었다. 그러나, 이러한 접근방식은 클라이언트의 계산량은 줄어들지만 서버의 계산량과 통신량이 너무 많고, 능동적 공격(active attack)에 노출되기 쉽다는 단점을 가진다. 본 논문에서는 이러한 단점을 극복하기 위해 RSA서명 생성을 위한 새로운 방식의 SASC 프로토콜을 제안한다. 본 논문에서 제안하는 방식은 비밀정보를 서버로부터 감추기 위해 비밀정보에 난수들을 곱하거나 더하여 서버에게 전달하는 방식이다. 제안 프로토콜은 능동적 공격에 대해 안전하며, 안전성이 데이타의 개수에 의존하지 않으므로 서버의 계산량과 통신량이 매우 적다. 또한, 클라이언트가 수행해야 할 계산량도 기존 프로토콜들의 43%로 줄게 된다.

1 서론

암호 시스템을 구현하기 위해 해결해야 할 가장 중요한 문제들 중 하나는 비밀정보를 관리하는 방법이다. 특히, 공개키 암호 시스템을 사용할 경우 비밀정보의 크기가 너무 커서 사람이 기억하기 어렵기때문에, 사용자의 비밀정보를 기억할 수 있는 별도의 장치가 필수적이다.

스마트 카드는 프로세서가 부착되어 있는 플라스틱 카드이므로 지니고 다니기에 용이하며 하드웨어적으로 안전하기때문에 사용자의 비밀정보를 기억하는 장치로 많이 이용되고 있다. 또한, 계산 능력도 지니고 있어 전자서명 수행이나 사용자 인증과 같은 암호학적 연산을 수행하는 데에 많이 이용된다. 그러나 스마트 카드에 부착될 수 있는 프로세서는 하드웨어 기술의 한계로 인해 범용 프로세서에 비해 성능이 많이 떨어진다. RSA와 같은 공개키 방식의 전자서명은 많은 연산을 필요로 하므로 이러한 스마트 카드에서 효율적으로 수행하기는 힘들다 [1].

*본 연구는 암호이론연구회의 연구결과임

이러한 문제점을 해결하기 위해, 1988년 Matsumoto, Kato, 그리고 Imai에 의해 SASC(Server-Aided Secret Computation) 프로토콜이 제안되었다 [2]. 그들이 제안한 방식인 RSA-S1과 RSA-S2는 RSA 전자서명을 위한 SASC 프로토콜로서, 클라이언트(스마트 카드)가 자신의 비밀정보를 감추기 위해, 비밀정보를 여러 조각으로 나누어 서버에게 전달한다. 그리고나서, 클라이언트는 서버(믿을 수 없는 주변장치, untrusted auxiliary device)가 계산하여 돌려준 값들을 이용해서 서명을 생성한다. 이 방식은 RSA서명을 생성하는데 있어 클라이언트가 해야 할 연산의 수가 스마트카드 스스로 RSA서명을 생성하는데 필요한 연산에 비해 매우 적으므로 현실적인 효용성이 크다. 따라서, SASC 프로토콜에 대한 많은 연구들이 수행되어 왔다 [3, 4, 5, 6, 7, 8, 9, 10, 11].

기존의 SASC 프로토콜들는 모두 RSA-S1과 RSA-S2와 같은 접근방식을 사용하는데, 이러한 접근방식에는 다음과 같은 문제점들이 있다: 첫째, 안전성이 클라이언트가 서버에게 보내는 데이타의 개수에 의존하므로, 통신량과 서버가 해야할 계산량이 많다. 둘째, 기존의 연구들이 밝히듯이 이러한 접근방식은 능동적 공격에 대해 취약하다 [3, 4, 5, 6, 7, 8].

본 논문에서는 이러한 단점들을 극복하기 위해 RSA서명 생성을 위한 새로운 방식의 SASC 프로토콜을 제안한다. 기존의 SASC 프로토콜들에서는 비밀정보를 감추기 위해 비밀정보를 여러 조각으로 나누는 방식을 사용한 반면, 제안 프로토콜에서는 비밀정보에 난수들을 곱하거나 더하는 방식을 사용한다. 제안 프로토콜은 능동적 공격에 대하여 안전하며, 또한 안전성이 데이타의 개수에 의존하지 않으므로 서버의 계산량과 통신량이 매우 적다. 또한, 클라이언트가 해야할 계산량이 기존 프로토콜들에 비해 43%이므로 매우 효율적이다.

본 논문의 구성은 다음과 같다. 2절에서는 RSA 서명방식과 기존의 연구에 대하여 간략히 소개하고, 3절에서는 본 논문에서 제안하는 프로토콜을 설명한다. 4절에서는 제안 프로토콜의 안전성에 대해 분석하고, 5절에서는 성능을 분석한다. 6절에서는 원하는 정도의 안전성을 얻으면서 계산량을 최소로 하는 보안인수를 선택하는 방법에 대해 설명하고, 기존의 프로토콜과 성능을 비교한다. 마지막으로 7절에서 결론을 맺는다.

2 관련 연구

본 논문에서 제안하는 프로토콜을 설명하는데 사용될 용어들의 일관성을 위해 RSA 서명 방식과 덧셈사슬(addition-chain)에 대해 간략히 설명한다.

RSA 서명 알고리즘 서명자는 매우 큰 소수 p, q를 구하고 $N = pq$를 계산한다. 그리고나서, $\phi(N)(= (p-1)(q-1))$과 서로 소인 임의의 정수 e를 고른 후에 $ed \equiv 1 \bmod \phi(N)$이 성립하는 d를 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.

덧셈사슬 RSA 서명 생성을 비롯한 많은 공개키 암호 시스템은 모듈라 멱승 연산을 주로 이용한다. 모듈라 멱승 연산은 모듈라 곱셈의 반복으로 이루어지는데, 이 때 모듈라 곱셈의 연산순서를 지수들의 덧셈으로 표현한 것이 덧셈사슬이며 다음과 같이 정의된다.

정의 1 임의의 양의 정수 d에 대한 **덧셈사슬**은 다음과 같은 성질을 지닌 일련의 수열 $a_0, a_1, \ldots, a_{l(d)}$이며, $l(d)$는 d에 대한 덧셈사슬의 길이다.

1. $a_0 = 1$, $a_{l(d)} = d$.
2. $a_i = a_j + a_k, 0 \le j \le k < i \le l(d)$.

위의 덧셈사슬의 정의에 의하면 $M^d \bmod N$을 구하고 나면, $M^{a_i} \bmod N$(, where $0 \le i \le l(d)$)들은 모두 구해져 있다.

기존의 SASC 프로토콜 Matumoto 등이 [2]에서 제안한 RSA-S1 프로토콜의 기본 구조는 다음과 같다. RSA 서명 생성을 위한 SASC 프로토콜의 목적은 클라이언트가 비밀키 d를 서버에 알려주지 않으면서 $M^d \bmod N$을 계산하는 것이다. 클라이언트는 우선 $d \equiv \sum_{i=1}^{M} d_i f_i \bmod \phi(N)$이 성립하는 벡터(vector) $D = (d_1, d_2, \cdots, d_M)$와 $F = (f_1, f_2, \cdots, f_M)$를 찾는다. 이들 중 D를 서버에게 전해주고, 서버는 $w_i \equiv M^{d_i} \bmod N$(, where $1 \le i \le M$)들을 계산하여 클라이언트에게 돌려준다. 클라이언트는 $M^d \equiv \Pi_{i=1}^{M}(w_i)^{f_i} \bmod N$를 계산함으로써 RSA 서명을 얻는다.

공격에 대해 취약하다 [3, 4, 5, 6, 7, 8].

이러한 형태의 SASC 프로토콜들은 능동적 공격에 약해서 많은 공격방법들이 제안되어 왔으며, 이를 막기위한 프로토콜들도 많이 제안되었다 [3, 4, 5, 6, 7, 8, 9, 10, 11]. 1995년 Beguin과 Quisquater에 의해 기존의 모든 능동적 공격에 대해 안전한 프로토콜이 제안되었으며, 이 프로토콜도 RSA-S1의 기본 골격을 이용한다 [10]. 본 논문에서는 제안 프로토콜의 성능을 Beguin과 Quisquater에 의해 제안된 프로토콜과 비교한다.

3 제안 프로토콜

본 논문에서 제안하는 RSA 전자서명을 위한 SASC 프로토콜을 설명한다. 제안 프로토콜은 클라이언트가 프로토콜을 수행하기 전에 미리 알고있어야 하는 사전계산(precomputation) 부분과, 실제로 전자서명을 수행하는 서명생성 부분으로 나뉜다.

사전계산 클라이언트는 다음의 순서로 사전 계산을 행한 후 그 값을 스마트 카드 내부에 기억한다. 클라이언트는 임의의 수 R, R'(,where $\lfloor \log R \rfloor + 1 \le B_R, \lfloor \log R' \rfloor + 1 \le B_{R'}$)을 생성한 후, 다음의 식이 성립하도록 r_i, r'_i, s_j, s'_j(, where $1 \le i \le k, 1 \le j \le l$)들을 임의로 선택한다. $k, l, B_R, B_{R'}$은 보안인수(security parameter)들이다. r'_j, s'_j들은 이진수로 표현되어 있다고 가정하며, $||$은 접합(concatenation)을 의미한다.

$$
\begin{gathered}
R = R_1 || R_2 \\
r_i \in R_1 \text{에 대한 덧셈사슬} \\
s_j \in R_2 \text{에 대한 덧셈사슬} \\
R' = r'_1 || r'_2 || \cdots || r'_k || s'_1 || s'_2 || \cdots || s'_l
\end{gathered}
$$

다음 식을 만족하는 두 정수 t_1, t_2를 계산한다:

$$
\begin{aligned}
t_1 &\equiv \tfrac{1}{r'_k}(\cdots(\tfrac{1}{r'_1}(d + (p-1) - r_1) - r_2) - \cdots - r_k) - R_1 \bmod \phi(N), \text{ and} \\
t_2 &\equiv \tfrac{1}{s'_l}(\cdots(\tfrac{1}{s'_1}(d + (q-1) - s_1) - s_2) - \cdots - s_l) - R_2 \bmod \phi(N).
\end{aligned} \tag{1}
$$

위의 식에서 $c = \frac{1}{a}b$'는 '$a \times c = b$'가 되는 값을 말한다.
$w_p \equiv q(q^{-1} \bmod p) \bmod N$, $w_q \equiv p(p^{-1} \bmod q) \bmod N$를 구한다.

서명생성 실제로 매번 서명을 생성하는 과정은 다음과 같다.

Client　　Server

M, t_1, t_2, N
$\longrightarrow$

Compute

$$M_1 = M^{t_1} \bmod N \text{ and } M_2 = M^{t_2} \bmod N$$

$$M_1, M_2$$
$$\longleftarrow$$

If the received values are zero or plus/munus one, this protocol stops.
$Z_p = M_1 \bmod p$, $Z_q = M_2 \bmod q$, $M_p = M \bmod p$, $M_q = M \bmod q$
$A = (\cdots((Z_p \times {M_p}^{R_1})^{r'_k} \times {M_p}^{r_k})^{r'_{k-1}} \cdots)^{r'_1} \times {M_p}^{r_1} \bmod p$,
$B = (\cdots((Z_q \times {M_q}^{R_2})^{s'_l} \times {M_q}^{s_l})^{s'_{j-1}} \cdots)^{s'_1} \times {M_q}^{s_1} \bmod q$,
$signature = Aw_p + Bw_q \equiv M^d \bmod N$
If the *signature* is not correct, this protocol stops.

---

4 안전성 분석

SASC 프로토콜의 안전성을 위협하는 공격방법에는 수동적 공격(passive attack)과 능동적 공격(active attack)이 있다. 수동적 공격은 SASC 프로토콜이 수행된 후에 클라이언트가 서버에게 제공한 정보들만으로 클라이언트의 비밀정보를 알아내려는 시도이며, 능동적 공격은 사용자의 비밀정보를 알아내기 위해 실제와는 다른 값을 주는 방식을 말한다.

4.1 수동적 공격

SASC 프로토콜의 안전성은 비밀정보 d를 알아내기 위해 공격자가 시도해야 할 경우의 수로 나타내어지며, 이것은 프로토콜 수행 중에 서버에게 전해지는 정보에 의해 좌우된다. 프로토콜을 수행한 후에 서버에게 주어지는 정보(혹은, 공격자에게 노출된 정보)는 $t_1, t_2, M^d \bmod N$이며, 악의의 서버(혹은, 공격자)는 이를 이용해서 공격을 수행할 수 있다.

4.1.1 수동적 공격 1

클라이언트가 보내는 정보 중 t_1, t_2를 바탕으로 이루어지는 공격이다. 만일 공격자가 r_i, r'_i, s_i, s'_i값들을 모두 추측하여 알아내게 되면 공격자는 d값을 알아낼 수 있게 된다. 공격자가 t_1, t_2로부터 r_i, r'_i, s_i, s'_i값들을 추측할 경우 필요한 경우의 수를 살펴본다.

우선 r'_i, s'_i들을 추측하여 맞추기 위해서는 R'의 비트열을 알아내야 하므로, $2^{B_{R'}}$번을 추측해야 한다. 그후에, r_i, r'_i들이 어떻게 나누어졌는지 알아내기 위해 $\begin{pmatrix} B_{R'} \\ k+l \end{pmatrix}$가지의 경우를 탐색해야 한다. 따라서, r_i, r'_i들의 탐색영역은 $2^{B_{R'}} \times \begin{pmatrix} B_{R'} \\ k+l \end{pmatrix}$이다.

r_i, s_i들의 경우에는 조금 다르다. R의 비트열을 알아내기 위해 2^{B_R}번을 추측해야 하는 것은 r'_i, s'_i의 경우와 같다. 그러나, r_i, s_i들은 나누는 방식은 r'_i, s'_i와는 다르게 중복이 허용된다. 따라서, r_i, s_i들의 분할 방식을 알아내기 위해서는 $(B_R+1)^{k+l}$번을 탐색해야 한다. 따라서, r_i, s_i들의 탐색영역은 $2^{B_R} \times (B_R)^{k+l}$이다. 따라서, 제안 프로토콜에서 **수동적 공격1**을 위해 필요한 탐색영역은 다음과 같다.

$$2^{B_R+B_{R'}} \times \begin{pmatrix} B_{R'} \\ k+l \end{pmatrix} \times (B_R+1)^{k+l} \tag{2}$$

4.1.2 수동적 공격 2

서명이 '$signature = Aw_p + Bw_q \bmod N$' 구조를 갖는다는 성질을 이용한다. 만일 서버가 서로 다른 $M, \overline{M}$에 대하여 서명을 받을 수 있다면 서버는 w_p, w_q값을 알 수 있다.

제안 프로토콜에서 서버가 클라이언트로부터 받는 값 $signature = Aw_p + Bw_q \bmod N$에서 A와 B의 구조는 다음과 같다:

$$\begin{aligned} &A = {M_1}^{\alpha} M^{\beta},\ B = {M_2}^{\alpha'} M^{\beta'},\\ &\text{where } \alpha = r_1' r_2' \cdots r_k',\ \beta = r_1 + r_1' r_2 + \cdots + (r_1' r_2' \cdots r_k') R_1,\\ &\alpha' = s_1' s_2' \cdots s_k',\ \beta' = s_1 + s_1' s_2 + \cdots + (s_1' s_2' \cdots s_l') R_2. \end{aligned}$$

따라서 서버가 $M_1^{\alpha} M^{\beta}$, $M_2^{\alpha'} M^{\beta'}$의 값을 알기 위해서는 서버는 $\alpha, \alpha', \beta, \beta'$값을 알아야 한다.

위 식에서 $\alpha \| \alpha'$는 $B_{R'}$자릿수이고, $\beta \| \beta'$은 $B_R + B_{R'} - 1$자릿수이다. 공격자가 r_i, r_i', s_i, s_i'들을 각각 찾지 않고, $\alpha, \alpha', \beta, \beta'$을 알기 위해서는 각각의 크기를 알아야 한다. $\lfloor \log \alpha \rfloor + \lfloor \log \alpha' \rfloor + 2 \le B_R'$이므로, α와 α'을 찾는데 필요한 탐색 영역은 $\sum_{i=1}^{B_{R'}} (2^i(i+1))$이다. 마찬가지로, $\lfloor \log \beta \rfloor + \lfloor \log \beta' \rfloor + 2 \le B_R + B_R' - 1$이므로, β와 β'을 찾는데 필요한 탐색 영역은 $\sum_{i=1}^{B_R + B_{R'} - 1} (2^i(i+1))$이다. 따라서, 제안 프로토콜에서 **수동적 공격2**를 이용해 클라이언트의 비밀정보를 알기 위해 필요한 탐색 영역은 다음과 같다:

$$\sum_{i=1}^{B_{R'}} (2^i(i+1)) \times \sum_{i=1}^{B_R + B_{R'} - 1} (2^i(i+1)) \tag{3}$$

4.2 능동적 공격

능동적 공격은 서버가 사용자의 비밀 정보를 알아내기 위하여 원래의 값과 다른 조작된 값을 보내어 주는 공격 방법을 말한다. 제안된 프로토콜에서는 서버가 M_1, M_2의 값을 조작할 수 있다.

서버가 능동적 공격을 하기 위해 보내온 값을 $\overline{M_1}, \overline{M_2}$라 하면($M_1 = M^{t_1}, M_2 = M^{t_2}$), 서버가 보내온 값 $\overline{M_1}, \overline{M_2}$가 프로토콜을 통과하기 위해서는 다음의 식을 만족시켜야 한다:

$$(\overline{M_1}^{\alpha} M^{\beta} \bmod p) \times w_p + (\overline{M_2}^{\alpha'} M^{\beta'} \bmod q) \times w_q \equiv M^d \bmod N. \tag{4}$$

따라서, 위의 식 (4)을 만족시키는 $\overline{M_1}$은 다음의 식을 만족한다:

$$\begin{aligned} &\overline{M_1}^{\alpha} M^{\beta} \equiv M^d \equiv {M_1}^{\alpha} M^{\beta} \bmod p\\ &\Longleftrightarrow \overline{M_1}^{\alpha} = {M_1}^{\alpha} \bmod p. \end{aligned}$$

따라서 서버가 식 (4)을 만족하는 $\overline{M_1}$을 알기 위해서는 서버는 α를 알아야 한다. 마찬가지로, 서버는 α'를 알아야만 식 (4)을 만족하는 $\overline{M_2}$을 알 수 있다. 따라서 서버가 능동적 공격을 하기 위해서는 α, α'을 알아야만 한다.

서버가 α, α'을 안다고 가정하자. 서버는 p, q를 알 수 없기 때문에 서버는 $\overline{M_1}^{\alpha} = {M_1}^{\alpha} \bmod N$, $\overline{M_2}^{\alpha'} = {M_2}^{\alpha'} \bmod N$을 만족하는 $\overline{M_1}, \overline{M_2}$를 먼저 찾아야만 한다. 그리고 이러한 $\overline{M_1}, \overline{M_2}$가 식 (4)를 만족해야만 능동적 공격을 할 수 있다.

$$(M^{\beta} \bmod p) {M_1}^{\alpha} \times w_p + (M^{\beta'} \bmod q) {M_2}^{\alpha'} \times w_q = M^d \bmod N \tag{5}$$

악의의 서버가 능동적 공격을 수행 할 경우 서버가 얻을 수 있는 정보는 식 (4)와 식 (5)가 있다. 그러나 식 (4)로부터 $\overline{M_1}^{\alpha} M^{\beta} \bmod p$ 혹은 $\overline{M_2}^{\alpha'} M^{\beta'} \bmod q$ 값을 알아내는 것은 모듈

러스 N에 대한 인수분해 문제와 동치이므로 서버는 식 (4)으로부터 아무런 정보도 얻을 수 없다. 따라서, 서버는 식 (5)를 이용하여야만 한다. 그러나, 식 (5)는 서버가 능동적 공격 없이 얻을 수 있는 정보와 같다. 따라서 서버는 능동적 공격으로부터 아무런 추가정보를 얻을 수 없다.

5 성능분석

SASC 프로토콜의 성능을 평가하기 위해서는 다음과 같은 3가지를 알아야 한다. 첫째는 서버에서의 계산량이고, 둘째는 클라이언트에서의 계산량, 마지막으로 서버와 클라이언트 사이의 통신량이다.

서버와 클라이언트에서 수행해야 하는 계산량을 모듈라 곱셈의 횟수로 표현하도록 한다. 모듈라 곱셈의 수행속도는 피연산자의 크기와, 피연산자들 중 변하지 않는 부분이 있는가에 따라서 크게 다르다. 따라서, 표준 모듈라 곱셈(SMM:Standard Modular Multiplication:SMM)을 정의하고 SMM 수행 시간을 기준으로 다른 종류의 모듈라 곱셈을 표현하도록 한다.

정의 2 $A \times B \bmod C$를 계산할 때, A,B,그리고 C에 대한 정보가 미리 주어져 있지 않고, C의 비트수가 *RSA* 연산에서의 젯수(*modulus*) N의 비트수와 같을때, 이러한 모듈라 곱셈을 **표준 모듈라 곱셈**으로 정의하고 표준 모듈라 곱셈을 수행하는 데에 소요되는 수행시간을 t_s로 표기한다.

본 논문에서 제안하는 프로토콜에서 주로 사용되는 모듈라 곱셈은 젯수 (modulus)가 RSA 연산에서의 큰 소수 p,q이므로 표준 모듈라 곱셈에서 사용되는 피연산자들의 길이의 절반이다. 이러한 모듈라 곱셈들을 **절반 모듈라 곱셈**(Half Modular Multiplication:HMM)이라고 정의하고, 절반 모듈라 곱셈을 수행하는 데에 소요되는 수행시간을 t_h로 표기한다. t_h는 $\frac{t_s}{4}$와 같다. 모듈라 멱승 연산을 위해서 이진방법(binary method)을 사용한다고 가정한다.

5.1 서버에서의 계산량

서버는 $M^{t_1} \bmod N$,$M^{t_2} \bmod N$을 계산하면 되므로, 두 번의 표준 모듈라 멱승 연산이 필요하다.

5.2 클라이언트에서의 계산량

제안 프로토콜을 수행하는 데 필요한 모듈라 곱셈 연산의 갯수를 계산한다. t_1, t_2의 계산은 사전에 미리 해 놓으며, 한 번 계산해 놓으면 그 값을 저장해 놓을 수 있으므로 프로토콜의 계산량에서 제외될 수 있다. 프로토콜의 수행 중에 클라이언트의 계산량은 대부분 서버로부터 $M^{t_1} \bmod N$과 $M^{t_2} \bmod N$을 받은 후에 이로부터 $M^d \bmod p$ $M^d \bmod q$를 얻어내는 데에 사용된다. 우선 $M^{r_i} \bmod p, M^{s_i} \bmod q$들을 계산하는데 $3/2(B_R)$번의 절반 모듈라 곱셈이 필요하다. 또한, r_i', s_i'들을 지수로 하는 모듈라 멱승 연산을 수행하는데, $3/2(B_{R'})$번의 절반 모듈라 곱셈이 필요하다. 또한, 이 값들을 모두 곱해야 하므로 $k+l$번의 절반 모듈라 곱셈을 수행해야 한다. $M^d \bmod p, M^d \bmod q$값을 구한 후에, 이 값들로부터 $M^{R_1^{-1}d} \bmod N$을 얻어내기 위한 계산(CRT를 이용한다)에 표준 모듈라 곱셈 2번이 사용된다. 마지막으로, 클라이언트의 전자서명 결과가 옳은지 확인하는 데에 두 번의 모듈라 곱셈이 더 필요하다(공개키 e는 3이 사용된다고 가정한다). 지금까지 설명한 계산과정을 t_s로 정리해 보면 다음

과 같다.

$$(\frac{3}{2}(B_R + B_{R'}) + k + l) \times t_h + 2t_s + 2t_s \quad (6)$$

$$= \frac{1}{4}(\frac{3}{2}(B_R + B_{R'}) + k + l + 16) \times t_s \quad (7)$$

5.3 통신량

SASC 프로토콜에서는 계산량 이외에 통신량도 수행속도에 영향을 미친다. 본 논문에서 제안하는 프로토콜을 수행하는 동안 통신되는 수들은, 처음에 클라이언트가 서버에 전해주는 4개의 정수 M, N, t_1, t_2와 나중에 서버가 클라이언트에 돌려주는 2개의 정수 $M^{t_1} \bmod N$, $M^{t_2} \bmod N$이 있으므로 총 통신량은 6개의 정수이다. 512-bit의 모듈러스 N을 사용한다면, 총 통신량은 3072-bit가 된다.

5.4 전체성능

전체 수행시간은 클라이언트에서의 계산시간과 서버에서의 계산시간 그리고 통신량을 모두 더한 값이다. 클라이언트의 계산량 중 일부는 서버가 계산을 수행하는 동안에 동시에 수행할 수 있다. 클라이언트가 표준 모듈라곱셈을 1번 수행하는 데 걸리는 시간을 T_c, 서버가 표준 모듈라곱셈을 1번 수행하는 데 걸리는 시간을 T_s, 그리고 서버-클라이언트간에 1비트를 전송하는 데 걸리는 시간을 T_t라고 하면, RSA 서명을 수행하는 데 걸리는 시간은 다음과 같다.

$$\frac{1}{4}(\frac{3}{2}(B_R + B_{R'}) + k + l + 16) \times T_c + 2 \times \log N \times T_s + 6 \times \log N \times T_t \quad (8)$$

6 토의

6.1 보안인수 선택

능동적 공격이 적용되지 않으므로, 수동적 공격의 탐색영역을 원하는 만큼 이상으로 유지하면서 계산량을 최소가 되게 하는 보안인수를 찾는다. 수동적 공격의 탐색영역은 식 (2)와 식 (3)의 값들 중 더 작은 값이다. 탐색영역을 2^{64} 이상으로 놓았을 때, 17번 이하의 계산으로 서명을 수행하는 보안인수들을 구할 수 있다. 예를들면 $< B_R = 7, B_{R'} = 23, k = 3, l = 3 >$으로 놓으면 식 (2)의 값은 2^{65}이고 식 (3)의 값은 2^{64}이면서 식 (7)의 값은 17이다.

6.2 성능비교

지금까지 알려진 모든 능동적 공격과 수동적 공격에 대해 안전한 SASC 프로토콜은 [10]에서 제안된 방법이다. 본 논문에서 제안하는 프로토콜을 [10]에서 제안된 프로토콜과 성능을 비교한 결과가 표 1에 나와있다. 제안 프로토콜의 보안인수들은 $< B_R = 30, B_{R'} = 23, k = 3, l = 3 >$으로 놓았고, Beguin과 Quisquater의 프로토콜의 보안 인수들은 참고문헌 [10]에서 제시한 보안인수 설정들 중 메모리 필요량이 가장 적은 $< h = 11, m = 29 >$를 선택하였다. 그 이유는 참고문헌 [10]에서 제시한 보안인수 설정들 중 가장 메모리 필요량이 적은 것도 제안 프로토콜에서의 메모리 필요량에 비해 크기 때문이다.

1절에서 언급한 바와 같이, 상황에 따라 서버와 클라이언트간의 성능비는 다양하게 변한다. 이러한 상황에서의 프로토콜의 성능변화를 비교한 그래프가 그림 1에 나타나 있다. x축은 서버와 클라이언트의 성능비인 T_s/T_c이며, y축은 1회 전자서명에 걸리는 전체시간을 나타낸다. T_t는 0이라고 가정한다. 512비트 피연산자들을 사용하고, 제안 프로토콜의 보안

표 1: 512-bit RSA서명 생성에 필요한 계산량 및 통신량 비교.

	Beguin and Quisquarter's [10]	Proposed
# of mod. mul.(client)	40	17
# of mod. mul.(server)	2885	1024
# of bits transferred	25,016	3,072
memory (bytes)	1,153	452

인수들을 $< B_R = 7, B_{R'} = 23, k = 3, l = 3 >$으로 설정 했을 때의 수행시간이다. 그림 1에서

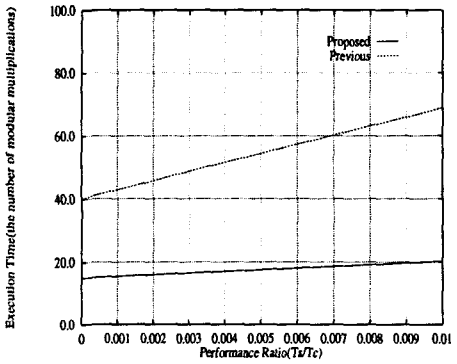

그림 1: 서버-클라이언트 성능비 변화에 따른 프로토콜들의 수행시간 비교

볼 수 있듯이, 제안 프로토콜의 수행속도는 클라이언트-서버간의 성능비가 달라져도 거의 변화없이 적음을 알 수 있다. 따라서, 제안 프로토콜은 서버의 성능이 클라이언트보다 압도적으로 우세하지 않은 상황에서도 좋은 성능을 보인다.

기존의 SASC 프로토콜들은 상당히 많은 양의 통신량을 필요로 하여, 프로토콜의 전체 성능을 떨어트린다. 통신속도의 변화에 따른 프로토콜의 성능변화를 비교한 그래프가 그림 2에 나타나 있다. x축은 통신속도와 클라이언트의 성능의 비율인 T_t/T_c이며, y축은 1회 전자서명에 걸리는 전체시간을 나타낸다. T_s는 0이라고 가정한다. 512비트 피연산자들을 사용하고, 제안 프로토콜의 보안인수들을 $< B_R = 7, B_{R'} = 23, k = 3, l = 3 >$으로 설정 했을 때의 수행시간이다. 그림 2에서 볼 수 있듯이, 제안 프로토콜의 수행속도는 클라이언트-서버간의 통신속도가 달라져도 거의 변화없이 적음을 알 수 있다. 따라서, 제안 프로토콜은 서버-클라이언트간의 통신속도가 상대적으로 열악한 환경에서도 좋은 성능을 보인다.

7 결론

본 논문에서는 RSA서명 생성을 위한 새로운 방식의 SASC 프로토콜을 제안하였다. 기존 프로토콜들의 접근방식은 비밀정보를 서버로부터 감추기 위해 비밀정보를 여러 조각으로 나누는 것인데 반해, 제안 프로토콜의 접근방식은 비밀정보에 난수들을 곱하거나 더하는 것이다. 본 논문에서 우리는 제안 프로토콜이 능동적 공격에 대해 안전함을 보였다. 클라이언트에서 수행해야 할 계산량을 비교하면, 제안 프로토콜은 기존 프로토콜의 43%의 계산량만으로 서명생성이 가능하다. 또한, 제안 프로토콜에서 필요로 하는 서버의 계산량과 통신량

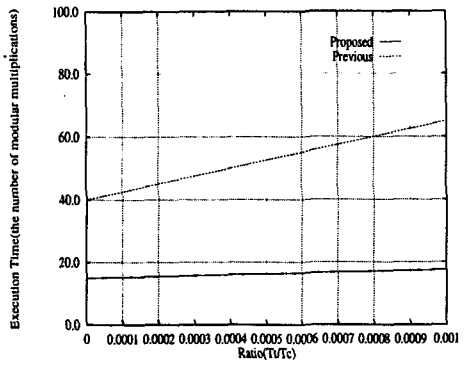

그림 2: 통신속도의 변화에 따른 프로토콜들의 수행시간 비교

도 기존의 프로토콜들에 비해 매우 적으므로, 상대적으로 취약한 성능을 가지는 서버나 통신 환경에서도 좋은 성능을 보인다.

참고 문헌

[1] 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.

[2] H. T.Matsumoto, K.Kato, "Speeding up secret computations with insecure auxiliary devices," in *Crypto'88*, pp. 497–506, 1988.

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

[4] 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.

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

[6] B.Pfitzmann and M.Waidner, "Attacks on protocols for server-aided rsa computation," in *Eurocrypt'92*, pp. 153–162, 1992.

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

[8] 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.

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

[10] P.Beguin and J.J.Quisquater, "Fast server-aided rsa signatures secure against active attacks," in *Crypto'95*, pp. 57–69, 1995.

[11] S.Kawamura, "Fast server-aided secret computation protocols for modular exponentiation," *IEEE JSAC*, vol. 11, no. 5, pp. 778–784, 1993.