TCP/IP를 이용하는 전산망의 해킹방지를 위한 경제적인 방화벽 토큰 설계 방안

A Novel Cost-Effective Firewall Token for Hacking Protection on TCP/IP Based Network

고 재 영
국방과학연구소

요 약

최근 전산망의 트래픽을 제어하여 해킹방지를 위해 방화벽을 구축한다. 방화벽의 보안 서비스는 인증, 접근통제, 기밀성, 무결성 그리고 감사기록 이다. 사용자는 방화벽에 인증을 위하여 토큰을 사용한다. 토큰은 작은 배터리를 내장하므로 전력 용량이 한정된다.

본 논문은 TCP/IP를 이용하는 전산망의 해킹방지를 위한 경제적인 방화벽 토큰 설계 방법을 제안한다. 공개키 암호 시스템의 주요 연산이며, 토큰 전력 소모의 대부분을 차지하는 지수연산에 Sparse 소수를 이용한 고속 처리 방법을 제안한다. 제안한 방법은 지수연산에서 모듈러 연산 량을 감소시킴으로 토큰의 배터리 용량 또는 CPU 가격을 낮출 수 있다.

Abstract

Recently a firewall is being employed to protect hacking by controlling the traffics. The security services in the firewall include authentication, access control, confidentiality, integrity, and audit trail. A token is adapted for authentication to the firewall. A token has a small battery within which has restricted power capacity.

This paper proposes a novel cost-effective firewall token for hacking protecting on transmission control protocol/internet protocol (TCP/IP) based network. This paper proposes a fast exponentiation method with a sparse prime that take a major operation for a public-key crypto-system and a major power consumption in the token. The proposed method uses much less amount of modular operations in exponentiation that is reduced of battery's capacity or CPU's price in the token.

Key Words : Firewall(방화벽), Token(토큰), Hacking(해킹), Authentication(인증), Public-key crypto-system(공개키 암호시스템), Exponentiation(지수연산), Modular reduction(모듈러 감소), Sparse prime(Sparse 素數)

1. 머릿말

현재는 인터넷(internet) 시대이다. 인터넷은TCP/IP[1]를 바탕으로 사용자가 급격하게 증가하여 전세계를 단일 문화권으로 묶는 거대한 망으로 성장하였다[2]. 정보통신기술의 급속한 발전으로 고도 정보사회로 진전하면서 정보누설, 전산망 해킹(hacking)[3] 등과 같은 역기능 현상이 국가나 사회적으로 심각한 문제점으로 등장하고 있는 실정이다.

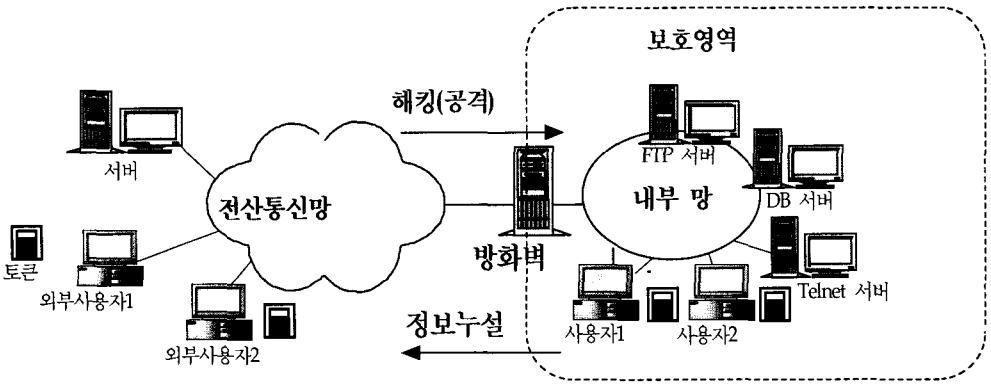
네트워크에 대한 보안을 위해서는 관리자는 네트워크에 관한 보안정책(security policy)을 수립하고, 보안정책의 목적에 적합한 보안 방어 수단의 범위를 확립할 필요가 있다[4]. 이에 따라 네트워크 관리자는 내부망을 보호하기 위하여 적법한 사용자와 불법 사용자를 구별할 수 있는 방안으로 방화벽(firewall)을 구축하였다[5]. 방화벽을 통과하려면 사용자는 방화벽에 인증을 해야하며, 이 때 사용자는 토큰을 이용한다. 토큰은 휴대용 전자 수첩 크기의 인증 알고리즘 처리 장치이다. 인증을 위해서는 공개키 암호 알고리즘을 이용하는데, 현재 대표적인 공개키 암호 알고리즘은 RSA(Rivest Shamir Adleman)[6], DSS(digital signature standard)[7], Diffie-Hellman[8] 등이 있다.

공개키 암호 알고리즘은 대부분 모듈러 지수 연산을 사용한다. 하지만, 암호 알고리즘은 안전성을 고려하여 큰 수의 지수 연산을 사용하기 때문에 이를 처리하기 위해서는 많은 계산 량과 많은 준비시간이 요구된다. 이를 토큰에서 실시간으로 처리하기 위해서는 고속 CPU와 사전 계산 값의 저장을 위한 많은 량의 메모리가 필요하다. 최근 대부분의 토큰은 PCMCIA(personal computer memory card international association) 형태로 개발되고 있으며, 이는 크기와 전력 용량의 제한으로 실시간 처리를 위한 토큰 설계에 많은 어려움을 준다.

본 논문에서는 전산망의 해킹 방지를 위한 토큰 설계에서 알고리즘 처리시간과 메모리 량을 최소화 할 수 있는 방법을 제안함으로써, 크기를 작게 하고, 동일 용량의 배터리로 기존 토큰보다 오랜 시간 사용할 수 있고, 동일시간 사용시 저용량의 배터리 또는 저가, 저속의 CPU를 사용할 수 있도록 한다.

2. 방화벽

방화벽의 의미는 네트워크의 보안 사고나 문제가 더 이상 확대되는 것을 막고 격리하는 것이다. 방화벽

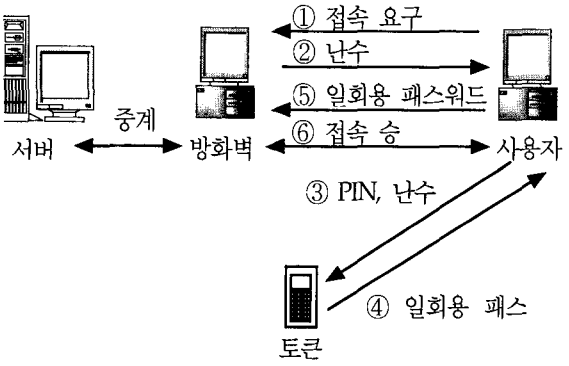


〔그림 1〕 해킹으로부터 내부 망 보호를 위한 방화벽
〔Fig 1〕 A firewall to protect the internal network from a hacking

은 내부의 전산망을 인터넷 등 외부 망과 연결하거나 사설 망을 구축할 경우 도청, 위장, 수정, 부인 등의 보안 위협 요소로부터 내부의 중요한 기밀과 정보를 보호하기 위한 장치이다. 즉, 방화벽은 그림 1과 같이 내부 망과 외부 망 사이에 설치되어 상호간에 미치는 영향을 차단시키기 위한 보안정책 및 이를 지원하는 소프트웨어 및 하드웨어를 통칭한다. 일반적으로 방화벽이 존재할 때 내부 망의 사용자가 외부 망의 호스트에 접속하기 위해서는 먼저 방화벽에 접속하여 인증을 통해야 하며, 반대로 외부 망의 사용자가 내부 망의 호스트에 접속하기 위해서도 먼저 방화벽에 인증을 통해야 한다. 따라서 내부 망과 외부 망의 접속은 방화벽을 통해서만 가능하기 때문에 해커가 들어올 수 있는 경로를 한정시키는 역할을 해 준다.

(1) 일회용 패스워드

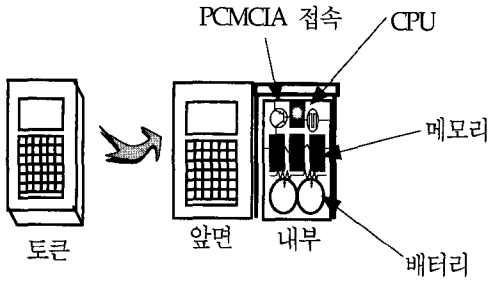
일회용 패스워드는 최근 방화벽의 사용자 인증 시스템으로 이용되며, 신청-응답(challenge-response) 방법으로 사용자가 서버에 접속을 시도할 때마다 새로운 패스워드를 사용한다. 일회용 패스워드의 개요는 그림 2와 같고, 인증 절차는 사용자가 서버에 접속을 요구하면 방화벽이 가상 서버(proxy server) 기능을 하여 난수를 생성하여 사용자에게 준다. 사용자는 토큰에 방화벽으로부터 받은 난수와 개인 식별 번호(PIN; personal identification number)를 입력하면 내장된 알고리즘을 통하여 일회용 패스워드를 얻는다. 사용자는 일회용 패스워드를 키보드를 이용하여 방화벽에 입력하면, 방화벽은 사용자로부터 받은 일회용 패스워드와 방화벽 자체에 사용자의 토큰과 동일하게 내장된 알고리즘을 통하여 생성된 값을 비교하여, 일치하면 사용자와 서버간의 패킷을 중계한다.



〔그림 2〕 일회용 패스워드의 개요
〔Fig 2〕 Overview of one time password

(2) 방화벽 토큰

방화벽을 통과하려면 사용자는 방화벽에 인증을 해야하며, 이 때 사용자는 토큰을 이용한다. 토큰은 휴대용 전자 수첩 크기의 인증 알고리즘 처리 장치로 그림 3과 같이 크게 CPU, 메모리, 배터리 등으로 구성된다. 최근 대부분의 토큰은 PCMCIA 형태로 개발되고 있으며, 설계 시 가장 중요한 요소는 토큰의 크기와 배터리 용량이다. 메모리 량과 배터리 용량이 많아지면 크기를 줄이는데 많은 어려움이 있다. 배터리 용량은 전력 소모량에 관계되며, 전력 소모량은 CPU 클록 속도와 메모리 량과 밀접한 관계가 있다. 공개키 암호 알고리즘의 실시간 처리는 많은 메모리와 고속 CPU가 요구되어 토큰 설계 및 제작에 많은 어려움을 준다.



〔그림 3〕 방화벽 토큰
〔Fig 3〕 A firewall token

3. 경제적인 방화벽 토큰 설계

현재 사용되는 대부분의 공개 키 암호시스템의 안전성은 소인수 분해 문제(integer factorization problem)와 이산대수 문제(discrete logarithm problem)에 기인한다. 대표적인 예로서 Diffile-Hellman의 키분배 과정, P가 소수일 때 Z^*_P상에서 ElGamal 프로토콜[9], P, Q가 소수일 때 모듈러 수 $n=PQ$ 를 사용하는 RSA 시스템 등이 있다. 이러한 시스템의 알고리즘 계산 시간은 CPU의 연산능력과 메모리 량에 비례하여 단축된다. 그러나 토큰은 배터리 용량의 한정 및 크기의 제한으로 고속의 CPU 및 많은 량의 메모리를 사용할 수 없다. 이를 극복하여 토큰에서 암호 알고리즘을 실시간 처리를 할 수 있는 방법은 메모리 량 및 계산 량을 단축시키는 고속 연산 알고리즘을 고안하는 것이다.

(1) 기존 계산 량 감소 방법

계산 량을 줄이는 방법은 여러 가지 관점에서 연구되었다. 첫 번째 방법은 곱셈과 모듈러 연산 과정에서의 계산 량을 감소시키는 방법이다. Yang[10]과 Montgomery[11]는 반복적인 나눗셈을 하지 않고 빠른 모듈러 감소를 하는 방법을 제안하였다. Chivers와 Kawamura[12]는 기수(base)단위 이내의 몫에 대한 표를 미리 만들고 덧셈을 사용하여 단계적으로 모듈러 감소를 하는 방법을 제안하였다.

두 번째 방법은 지수 승 연산과정에서 계산 량을 감소시키는 방법이다. 지수 승을 계산하기 위한 가장 효과적인 방법은 addition chain[13]을 사용하는 것이다. 일반적으로 512비트 모듈러 지수연산하기 위하여 크기가 605인 addition chain을 사용할 수 있다고 알려져 있다.

세 번째 방법은 RSA 시스템의 경우에 기수가 고정이라는 점에 착안을 하여 사전 계산으로 각 지수 승 위치에 대하여 사전 계산한 값을 이용하는 것이다. Brickell[14]과 P.J.Lee[15]이 제안한 이러한 방법은 기수가 고정되지 않은 시스템에서는 사용할 수가 없다.

마지막 방법은 큰 수를 사용하면서도 계산 량을 줄이는 방법이다. Vanstone과 Zuccherato[16]의 논문에서 이러한 주제에 대하여 처음으로 언급하였다.

(2) 제안한 계산 량 감소 방법

안전성을 해치지 않으면서 가중치가 매우 크거나 작은 sparse 소수를 이용하는 방법을 제안한다. Sparse 소수의 사용은 ElGamal이나 Diffile-Hellnam과 같이 이산대수 방식을 이용한 암호시스템에서는 쉽게 적용 가능하다. 이산대수 방식은 지수 연산의 복잡성과 관계된다. $y=x^d \bmod P$ 에서 비밀 키 d의 안전성은 $d=\log_x y \pmod P$ 의 계산에 따른 어려움에 기인한다. 따라서 이산대수를 사용하는 암호시스템에서 sparse 소수를 사용하여 계산 량을 줄이는 과정은 안전성에 어떠한 영향을 주지 않는다. 오히려 같은 계산 량에 대하여 sparse 소수를 사용하게 되면 P 의 크기를 상대적으로 증가시킬 수 있게 되므로 더욱 더 안전한 암호시스템을 구축할 수 있다.

1) 예비 사항

본 논문에서는 이후 사용하는 용어, 기호, 부울 함수의 여러 가지 암호학적 특성을 정의한다.

정의 1) 정수 P가 기수 b로 하는, 크기가 N 자리수일 때 다음과 같이 표현한다.

$$P=\sum_{i=0}^{N-1} p_i b^i, \quad \text{for } 0 \le p_i < b, \quad 0 \le i < N-1$$

또한, 표기의 편이성을 위해

$p[i:j] = p_i b^{i-j} + p_{i-1} b^{i-j-1} + \cdots + p_j$

$(i > j, p[i:i] = p_i)$ 로 표현한다.

정의 2) 소수 P가 기수 2로 하는, 크기가 N자리 수일 때 $(p_0, p_1, \cdots, p_{n-1})$에 대하여 다음과 같이 정의한다.

① 가중치(Weight) w_p는 P를 2진수로 표현할 때 1 의 개수이다.

② 1's Sparse P는 w_p가 N에 가까운 소수로서, 0 보다 1 이 많은 소수이다.

③ 0's Sparse P는 w_p가 1 에 가까운 소수로서, 1 보다 0 이 많은 소수이다.

④ 길이(Length) l_p는 P의 비트 수이다.

정의 3) 세 자연수 a, b, P 에 대하여 다음과 같이 정의한다.

① a와 b의 뺄셈 $a-b$가 P의 배수일 때 'a와 b는 모듈러 P에 관하여 합동 (a is congruent to b modulo P)'라고 하고 $a \equiv b(\mathrm{mod}\ P)$로 나타낸다.

② 'b mod P'라 함은 모듈러 P에 관하여 b와 합동이면서, P보다 작고 0 이상인 자연수를 의미한다. 즉, $a = b \bmod P$는 $a \equiv b(\mathrm{mod}\ P)$ 이고, $0 \le a < P$임을 의미한다.

③ 자연수 a에 대하여 $[a]$는 a보다 작지 않은 최대 자연수이다.

정의 4) 두 자연수 g, b^N에 대하여 다음과 같이 정의한다.

① $g \gg N$은 g를 $l_{b-1} \times N$번 오른쪽으로 이동시킨 값이다.

② $g \ll N$은 g를 $l_{b-1} \times N$번 왼쪽으로 이동시킨 값이다.

③ $g \% b^N$은 g를 b^N에 대하여 모듈러 값이다.

2) Sparse 소수

모듈러 감소를 위하여 많은 알고리즘이 있다. 예를 들어, 고전적인 방법, Barret의 알고리즘, Yang의 알고리즘, Montgomery의 알고리즘 등이 있다. 단지 감소 과정만을 고려한다면, 가장 빠른 것은 Montgomery의 감소방법이다. 이 절에서는 특별한 형태의 모듈러를 $(0 < P' < b^{N'}, N' < N)$ 이용하여 모듈러 감소 과정을 단순화시킨다. 소수 P는 $P = b^N - p'$, 를 만족하면, 감소 지수 형태(diminished radix form)를 갖는다고 말한다. (간단히, DR 모듈러라고 표현하자.) 이런 형태는 모듈러 감소 과정을 매우 간단하게 만들어 준다. 예를 들어, x mod P(x는 $2N$-digit의 정수)를 모듈러 감소하는 경우를 고려해 보자. 합동 식 $b^N = P' \bmod P$로부터 다음의 간단한 알고리즘이 모듈러 감소 연산을 수행함을 알 수 있다.

for $(i = 2N-1; I \ge N; i = i-1)$ {

$x[i-1:i-N] \leftarrow x[i-1:i-N] + x_i P'$

if $(c_{i>0})$ $x[i-1:i-N] \leftarrow x[i-1:i-N] + x_i P'$}

알고리즘을 수행 후 $x[N-1:0] \ge P$이면, $x[N-1:0]$에서 P만큼 뺀 값이 최종 결과이다. 그러나, 모듈러 연산시 중간 모듈러 감소단계에서는 위의 감산과정을 할 필요가 없다. 이 알고리즘의 성능은 조건문에 매우 밀접한 관계를 갖는다. 상대차수 δ가 $\delta = N - N'$ 이라고 하면, 조건문을 수행할 확률은 평균적으로 $b^{-(\delta+1)}$ 보다 작게 된다. 즉, 캐리(carry)는 $x[i-1 ; i-N]$가 b^N에 가까울 때 발생하게 된다. 결론적으로 위의 알고리즘은 조건 문 내의 수행을 상당부분 생략할 수 있다. Bosselaers[17]은 세 가지 방법에 대한 계산 량을

표 1과 같이 비교 분석하였다.

〔표 1〕 $2N$ 크기를 갖는 x에 대하여 P의 모듈러 감소(기존의 방법)
〔Table 1〕 Modular reduction method of x mod P of $2N$ size (classical method)

알고리즘 / 항목	Classical	Barrett	Montgomery	DR
곱 셈	$N(N+2.5)$	$N(N+4)$	$N(N+1)$	$N(N-\delta)$
나 눗 셈	N	0	0	0
사 전 계 산	*Normalization*	$b^{2N} \div P$	$-p_0^{-1} \bmod b$	$b^N - P$
작업메모리	$2N$	$4N$	$4N$	$2N$

① 소수 P가 $b^N \pm 1$형태인 경우(case 1)

DR에서 제시한 방법의 특별한 경우로서 $P' = 1$ 이면 보다 효과적인 계산이 가능하며, $P' = -1$ 일 때도 같은 효과를 갖는다. 즉, P의 값이 $b^N \pm 1$의 형태이면, $P' = b^N - P = b^N - P = b^N - (b^N \pm 1) = \mp 1$ 이다. P'가 P보다 훨씬 작은 1의 차수를 가지므로, 상대차수 $\delta = N-1$ 이므로, 조건문의 수행을 할 확률은 $b^{-(N-2)}$보다 작게 된다.

for $(i = 2N-1;\ i \geq N;\ i = i-1)$ {
  $x[i-1:i-N] \leftarrow x[i-1:i-N] \mp x_i$
  if(carry or borrow)
  $x[i-1:i-N] \leftarrow x[i-1:i-N] \mp c_i$ }

여기서, c_i는 올림수(carry) 또는 내림수(borrow) 이다.

위의 수식에서 for 문이 생략 가능하다. x_i는 상위 자리수를 표현하므로 다음과 같이 표기가 가능하다.

$$x[N:0] \leftarrow x[2N-1:N] + x[N-1:0] \quad \cdots\cdots\cdots\cdots (1)$$
if $(x[N:0] \geq P)$ $x[N-1:0] \leftarrow x[N:0] - P$
if $(x[N:0] < 0)$ $x[N-1:0] \leftarrow x[N:0] + P$

$b^N \pm 1$의 형태를 갖는 모듈러 수를 가지고 모듈러 감소를 수행하게 되면 곱셈, 나눗셈, 사전계산이 모두 필요 없게 되면서 단 한 번의 덧셈과 한 번의 조건문으로 모듈러 감소를 수행하게 된다.

식 (1)을 이용하여 곱셈(또는 제곱)과정에서 직접 모듈러 감소 연산을 수행 할 수 있다. 기존의 방법은 기수 b에 대한 N의 크기를 갖는 X와 Y를 곱한 결과 값 Z는 $2N$의 크기를 가지므로 Z에 대하여 다시 모듈러 감소를 해야하지만, $b^N \pm 1$의 특성을 이용하게 되면 곱셈과정에서 직접 모듈러 감소를 수행 할 수 있는 특징을 갖는다.

X와 Y를 다음과 같이 놓자. 계산의 편의상 N을 짝수로 설정한다.

$$X = x_1 b^{N/2} + x_0,\ 0 \leq x_i < b^{N/2},\ i = 0, 1$$
$$Y = y_1 b^{N/2} + y_0,\ 0 \leq y_i < b^{N/2},\ i = 0, 1$$

$Z = X \times Y \bmod P$ 는 식 (1)의 특성에 따라 다음과 같이 표현한다.

$$\begin{aligned} & X \times Y \bmod P \\ &= (x_1 b^{N/2} + x_0) \times (y_1 b^{N/2} + y_0) \bmod P \\ &= x_1 y_1 b^N + (x_0 y_1 + x_1 y_0) b^{N/2} + x_0 y_0 \bmod P \end{aligned}$$

$$R_1 = ((x_0 y_1 + x_1 y_0) \ \%\ b^{N/2}) b^{N/2x} + x_0 y_0$$
$$R_2 = x_1 y_1 + ((x_0 y_1 + x_1 y_0) \gg N/2)$$
$$Z_1 = R_1 \ \%\ b^N$$
$$Z_2 = R_2 + (R_1 / b^N)$$
$$Z = Z_1 \mp Z_2$$
if $(Z \geq P)$ $Z = Z - P$
if $(Z < 0)$ $Z = Z + P$

실제 사용할 수 있는 $b^N \pm 1$의 성질을 갖는 소수는 많이 존재하지 않는다. 그러나, XY mod P연산 시 기존의 곱셈 방법에 대하여 작업 메모리의 크기 2을 N으로 줄일 수 있다. 또한, 모듈러 감소 연산을 따로 할 필요가 없어 획기적으로 계산 량을 줄일 수 있다.

② 소수 P에 특정한 상수 α를 곱했을 때 $b^{N+n} \pm 1$ 형태인 경우(case 2)

Case 1의 방법은 소수의 영역이 b^N의 주변으로 제한된다는 것이다. 따라서, 소수를 변경하여 사용하는 시스템에서는 조건에 맞는 소수를 구하는 일이 쉬운 일이 아니다. 이러한 문제점을 해결하기 위하여 소수에 상수인 특정한 수 α를 곱했을 때 $b^{N+n} \pm 1$의 형태를 만족하는 소수를 사용한다. 여기서, P가 $b^{N-1} \le P b^N$ 이므로, αP는 $b^{N+n} \pm 1$의 값을 갖기 위해서, α는 $b^n \le \alpha \langle b^{n+1}$를 만족하는 값을 가져야 한다. 이제 $b^{N+n} \pm 1$을 이용하여 모듈러 감소방법을 제시한다.

αP의 값이 $b^{N+n} \pm 1$ 이면, $\alpha P = b^{N+n} - \alpha P = \mp 1$ 이다.

$$x[N+n:0] \leftarrow \mp x[2N-1:N+n] + x[N+n-1:0] \quad (2)$$

$$x[N-1:0] \leftarrow x[N+n:0] \bmod P \quad \cdots\cdots\cdots\cdots (3)$$

곱셈과정에서 $b^{N+n} \pm 1$의 형태를 갖는 모듈러 수를 가지고 모듈러 감소를 수행하게 되면, 중간에 곱셈은 필요 없지만, 식 (3)과 같이 n 크기의 모듈러 감소 연산이 요구된다.

$$R_1 = (x_1 y_1 \% b^n) b^N + ((x_0 y_1 + x_1 y_0) \% b^{N/2+n}) b^{N/2} + x_0 y_0$$

$$R_2 = (x_1 y_1 \gg n) + ((x_0 y_1 + x_1 y_0) \gg N/2 + n)$$

$$Z_1 = R_1 \% b^{N+n}$$

$$Z_2 = (R_2 + (R_1 / b^{N+n}))$$

소수에 특정한 상수 α를 곱했을 때 $b^{N+n} \pm 1$형태인 조건을 만족하는 소수는 상당수 존재한다. 효과적인 상수 α의 크기는 n이 작은 값을 갖도록 하는 것이다. 이렇게 하면 적은 수의 나눗셈만으로 충분히 가능하다. 역시, 곱셈 연산 시 기존의 곱셈 방법에 대하여 사용 메모리의 크기 $2N$ 을 $N+n$으로 줄일 수 있다.

③ 소수 P에 특정한 상수 α를 곱했을 때 $b^{N+n} \pm \beta_1 b^{r1} \pm \cdots \beta_k b^{rk}$ 형태인 경우(case 3)

Sparse 소수의 일반적인 형태인 경우에 대하여 알아본다. 계산 량을 줄이기 위하여 β의 크기가 작으면 작을수록 효과가 크다. 즉, β가 크면 클수록 모듈러 감소의 효과가 상대적으로 줄어드는 단점이 발생한다. β의 크기가 크지만, β의 형태가 $\beta_1 b^{r1} \pm \cdots + \beta_k b^{rk}$ $(N \rangle r_1 \rangle \cdots \rangle r_k)$ 이면 모듈러 감소연산을 줄일 수 있다. 이때, b^{r1}의 크기가 작을수록 효과가 커진다. 이러한 값들은 난수처럼 보이므로 RSA 시스템에서도 사용 가능하다. 이러한 성질을 갖는 수들은 $GF(P)$상에 많이 분포하며 b^N상의 주변에서 멀리 떨어진 값을 사용할 수 있으므로 안전성에서도 강하다고 할 수 있다. 소수에 상수인 특정한 수 α를 곱했을 때 $b^{N+n} \pm \beta_1 b^{r1} \pm \cdots \beta_k b^{rk}$의 형태를 만족하는 소수를 사용한다. 여기서, P가 $b^{N-1} \le P \langle b^N$이므로, αP가 $b^{N+n} \pm \beta_1 b^{r1} \pm \cdots \beta_k b^{rk}$의 값을 갖기 위해서는 α는 $b^n \le \alpha \langle b^{n+1}$ 의 값을 가져야 한다. $b^{N+n} \pm \beta_1 b^{r1} \pm \cdots \beta_k b^{rk}$을 이용하여 $2N$의 크기를 갖는 수를 모듈러 감소를 하는 방법을 분석해 본다.

αP의 값이 $b^{N+n} \pm \beta_1 b^{r1} \pm \cdots \beta_k b^{rk}$이면, $\alpha P' = b^{N+n} - \alpha P = b^{N+n} - (b^{N+n} \pm \beta_1 b^{r1} \pm \cdots \pm \beta_k b^{rk}) = \mp \beta_1 b^{r1} \mp \beta_k b^{rk}$이다.

이 방법은 두 가지의 경우가 있는 데 $n \ge 2r_1$일 때와 $n \langle 2r_1$일 때로 구분한다.

첫 번째는 $n \ge 2r_1$일 때,

$x[N+n:0] \leftarrow x[2N-1:N+n] \times (\mp\beta_1 b^{r_1} \mp \cdots \mp \beta_k b^{r_k}) + x[N+n-1:0]$

이고, 다시

$x[N+n:0] \leftarrow \mp x[2N-1:N+n]\beta_1 b^{r_1} \mp \cdots \mp x[2N-1:N+n]\beta_k b^{r_k} + x[N+n-1:0]$

이므로, 다음과 같은 알고리즘으로 나타낼 수 있다.

$x[N+n:0] \leftarrow (\mp x[2N-1:N+n] \ll r_1)\beta_1 \mp \cdots \mp (x[2N-1:N+n] \ll r_k)\beta_{k+} x[N+n-1:0]$ ..... (4)

$x[N-1:0] \leftarrow x[N+n:0] \bmod P$ ..... (5)

다음은 $n < 2r_1$ 일 때, β_k의 값이 b보다 작은 경우라고 하자.

$x[N+r_1-n:0] \leftarrow x[2N-1:N+n] \times (\pm\beta_1 b^{r_1} \pm \cdots \pm \beta_k b^{r_k}) + x[N+n-1:0]$ 이고, 다시

$x[N+r_1-n:0] \leftarrow \mp x[2N-1:N+n]\beta_1 b^{r_1} \mp \cdots \mp x[2N-1:N+n]\beta_k b^{r_k} + x[N+n-1:0]$

이므로, 다음과 같은 알고리즘으로 나타낼 수 있다.

$x[N+r_1-n:0] \leftarrow (\mp x[2N-1:N+n] \ll r_1)\beta_1 \mp \cdots \mp (x[2N-1:N+n] \ll r_k)\beta_k + x[N+n-1:0]$. (4')

$x[N-1:0] \leftarrow x[N+r_1-n:0] \bmod P$ ..... (5')

곱셈연산에서 $b^{N+n} \pm \beta_1 b^{r1} \pm \cdots \beta_k b^k$의 형태를 갖는 모듈러 수를 가지고 모듈러 감소를 수행하게 되면 식 (4) 또는 (4')에서는 곱셈이 필요 없어지면서, 비트 shift가 일어난다. 다만 β_k와의 곱셈이 필요하다. 또한, 식 (5) 또는 (5')에서 모듈러 감소 연산이 요구된다.

$R_3 = (R_2 + (R_1 \ / \ b^{N+n})) \quad +$

$Z_2 = \mp(R_3 \ll r_1) \times \beta_1 \mp \cdots \mp (R_3 \ll r_k) \times \beta_k + R_3$

소수에 특정한 상수 α를 곱했을 때 $b^{N+n} \pm \beta_1 b^{r1} \pm \cdots \pm \beta_k b^k$형태인 값을 사용할 때는, 비록 n크기 또는 $r_1 - n$ 크기의 모듈러 감소 연산을 사용해야 하지만 소수가 널리 분포할 수 있으므로 안전성에 매우 효과적이다. 효과적인 상수 α의 크기는 작은 값을 사용하는 것이 좋으며, r_1은 N에 비하여 작은 값이 유리하며 k도 적을수록 좋다. 이렇게 하면 모듈러 감소 연산 량을 줄일 수가 있다. 만일 r_1의 크기가 커지면 커질수록 계산 량은 그만큼 커지므로 실제 제안한 방법의 효과가 줄어든다. 역시, 곱셈 연산 시 기존의 곱셈 방법에 대하여 사용 메모리의 크기 $2N$을 $N+n$ 또는 $N+r_1-n$으로 줄일 수 있다.

④ 응용 가능성 및 비교 분석

암호시스템에서 사용하는 계수들은 시스템의 안전성을 보장하기 위하여 가능한 큰 정수들을 사용한다. 이러한 수들은 컴퓨터의 워드 길이보다 훨씬 더 큰 다정도워드(multiple - precision words)이며, 일반적으로 기수 b를 2의 멱승 값을 사용하지 않고 다른 기수를 사용하여 sparse 수의 성질이 노출되지 않도록 할 수 있다. Case 3에서 제안한 방법으로 다음과 같은 예를 들 수 있다. $P = 7677_{(10)}$일 때, $XY \bmod P$를 계산해 보자. 16진수로 표현하면 $P = 1DFD_{(16)}$이다. 2진수로 표현하면 $P = 1110111111101_{(2)}$이고, $l_p = 13$, $w_p = 11$이다.

Z = XY mod P (P = 7677, α = 13, $b^{r_1} = 100$, β_1 = 2, b = 10, N = 4, n = 1, αP = 99801)

= 0x0EA1 × 0x253E mod 0x1DFD

= 3745 × 9534 mod 7677

= 37×95×104 + (37×34 + 45×95)×102 + 45×34 mod 7677

= 3515×104 + 5533×102 + 1530 mod 7677

= (1530 + 53300 + 50000) + (35100+500) × 2 - (351+5) mod 7677

= 4830 + 35700 × 2 - 357 mod 7677
= 4830 + 71400 - 357 mod 7677
= 75873 mod 7677
= 75873 - 9 * 7677
= 6780
= 0x1A7C

위의 예제를 통하여 적절한 sparse 소수의 선택으로 지수연산시 효과적으로 계산 량을 줄일 수 있음을 보였다. 이상으로 제안된 3가지의 sparse 소수를 사용한 경우 항목별 비교한 결과는 표 2와 같으며, 표 1과 비교 결과, 본 논문에서 제안한 방식을 사용하게 되면 계산 량 및 작업 메모리를 효과적으로 줄게된다.

〔표 2〕 $2N$크기를 갖는 x 에 대하여 P 의 모듈러 감소(제안한 방법)
〔Table 2〕 Modular reduction method of x mod P of $2N$ size (proposal method)

항목 \ 소수형태	case1	case2	case3
곱셈	0	0	$N \times r_1$
모듈러감소연산	0	n	n or r_1+n_1-n
사전계산	none	none	*none*
작업메모리	N	$N \times n$	$N \times n$ or $N + r_1 + n_1-n$

〔주〕 n_1은 β_1의 크기

표 3은 512비트에 대한 지수 승의 비교표이다. 전체적으로 제안한 방법이 가장 효과적이며, 항목별로는 제안한 방법은 모듈러 연산에서 가장 획기적인 감소를 보이며, 곱셈연산은 Addition chain 방법이 효과적이다. 따라서, 지수가 가변인 공개 키 암호시스템의 연산은 모듈러 부분은 제안한 방법을 사용하고, 곱셈 부분은 Addition chain 방법을 사용하여 병행 처리함이 가장 효과적임을 알 수 있다.

표 3. 512비트 지수 승 연산 (평균 값)
Table 3. 512 bit exponential arithmetic(average)

항목 \ 알고리즘	Classical	Addition chain	Brickell	P.J.Lee	제안한 방법
제곱(A2)	511	511	0	0	511
곱셈(AB)	256	97	130	90	256
모듈러 (mod P)	767	608	130	90	0[*]
사전계산	필요 없음	필요 없음	필요	필요	필요 없음
테이블	필요 없음	필요 없음	필요	필요	필요 없음
단점	계산량이 많음	윈도우 설계	지수가 고정 다량 메모리	지수가 고정 다량 메모리	sparse 소수

[*] 모듈러 감소 연산은 제곱과 곱셈 연산 내에 포함되어 있다.

4. 실험 및 고찰

A 와 B의 공통키 $K_{AB} = g^{S_A} \bmod P$ 를 구하기 위해서 평균적으로 768번의 곱셈과 모듈러 감소 연산이 요구된다. 이를 TMS320C25-40MHz의 프로세서를 이용하여 계산하였다. 제곱과 곱셈은 모두 동일한 함수를 사용하였으며, 모듈러 감소 연산은 Barrett 의 방법을 사용하였다. Barrett 의 모듈러 감소 연산에는 곱셈이 두 번 들어간다. 따라서, $AB \bmod P$ 를 계산하기 위해서는 3번의 곱셈연산이 요구된다.

512비트의 지수연산에 적용하며, 16비트와 16비트의 곱셈을 조합하여 사용한다. 따라서 512비트(32 word)와 512비트(32 word)의 곱셈은 1024회의 16비트와 16비트의 곱셈이 요구된다. 또한 992회의 덧셈이

요구되며 결과는 표 4와 같다[100].

〔표 4〕 512비트 크기를 갖는 $A \times B$ 곱셈연산 (평균값)
〔Table 4〕 512-bit $A \times B$(average)

가중치 ω	소요시간 (Machine cycles)	소요시간 (40MHz)
$\omega_A = 512,\ \omega_B = 512$	28706	2.87msec
$\omega_A = 2,\ \omega_B = 2$	20057	2.00msec
$\omega_A = 256,\ \omega_B = 256$	21987	2.19msec

위의 표 4를 통하여 512비트(32 word) 크기 A와 16비트(1 word) 크기의 B의 곱셈연산은 약 68usec가 소요된다. 기존의 방법을 이용하여 공통키를 사용하여 만드는 데 걸리는 시간은 2.19msec에 총 곱셈횟수 768×3을 하면 5.04초가 걸린다.

본 논문에서 제안한 방법을 사용하면, Barrett 의 모듈러 감소 연산에 사용되는 두 번의 곱셈이 불필요하여 그만큼 계산 량이 줄어든다. 따라서 평균적으로 case 1의 방법을 사용하게 되면, 공통키를 만들 때 1.68초가 걸리고, case 2와 case 3의 방법은 표 5와 같다.

토큰을 그림 3과 같이 CPU(TMS320C25-40MHz), 메모리(AT29LV512) 1개, 기타 주변회로(PCMCIA 접속회로)로 구성하고, 전원은 3V이며 각각의 소비전력을 측정한 결과 90mW, 60mW, 30mW 이며, 리듐 배터리(CR 2430)의 2개 용량은 560mAh(3V) 이다. 총 토큰 사용시간은 9.3시간이며, case 2의 키 생성이 평균 2초라하고, 1일 최대 25개 분량의 키를 생성하면 최소 670일 사용 가능하다. 기존 방법으로 구현 시에는 265일 사용 가능하다.

〔표 5〕 공통키 생성 시간 (평균)
〔Table 5〕 Generation time of session key (average)

소수형태	case 1	case 2	case 3
공통키 생성	A	A+B×n	A+Bxn or A+B×(r1+n1-n)

[주] A = 512비트와 512비트의 768회 평균 곱셈연산(1.68sec)
B = 512비트와 16비트의 768회 평균 곱셈연산(54msec)
n_1은 β_1의 크기

5. 맺음말

본 논문에서는 전산망에 해킹방지를 위하여 설치한 방화벽에 사용자 인증을 위한 경제적인 방화벽 토큰 설계 방안을 제안하였다. 제안한 방안은 sparse 소수를 이용한 고속처리 방안으로, 성능 평가를 수행한 결과, 곱셈 횟수 면에서는 기존 방법은 대부분 N^2에 비하여, 제안한 방법은 최대 $N \times n$에서 최소 0이며, 메모리 사용 면에서도 기존은 4N(또는 2N)에 비하여, 제안한 방법은 대략 $N+n$ 정도 소요되었다. 또한 공개키 암호 시스템의 알고리즘 처리 속도 면에서도 기존보다 3배 정도의 경이적인 성능 향상을 보였다.

제안한 알고리즘은 처리시간과 메모리 량을 최소화할 수 있는 방법으로 이를 이용하여 토큰을 설계하면, 크기가 한정적인 PCMCIA 형태의 토큰제작이 가능하고, 동일 용량의 배터리로 기존 토큰보다 처리속도 및 메모리 감소를 고려하면 3배 정도의 오랜 시간 사용할 수 있고, 동일시간 사용 시 1/3 용량의 배터리 또는 저가, 저속의 CPU를 사용할 수 있다.

향후 소인수 분해방식의 시스템에서도 사용 가능한 sparse 모듈러 감소 연산 방법, β 의 크기가 큰 경우의 적용, 안전성, N 의 크기와 속도와의 관계 등이 연구돼야 할 과제이다.

참 고 문 헌

1. W. Richard Stevens, TCP/IP Illustrated, vol. 1, Addison Wesley, 1994.
2. 고재영, 공창국, 전산망 보안용 패킷 감시 및 암호화에 관한 연구, 국방과학연구소, 1996년 10월.
3. 김상정, 이현우, 신윤, 정용종, 박정현, 임휘성, 임채호, "정보시스템 해킹 방지 기술 현황 및 시스템 설계", WISC '97, pp. 3-25, 1997년 9월.
4. 홍순좌, "컴퓨터 통신 보안기법 및 메커니즘", 국방과학연구소, pp. 1-23, 1996년 5월.
5. R. William, Cheswick, and M. Steven Bellovin, Firewall and Internet Security, Addision Wesley, 1994.
6. R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Sigatures and Public Key Cryptosystems," Communications of the ACM, Feb. 1978.
7. S. Akl, "Digital Signatures: A Tutoroal Survey," Computer, Feb. 1983.
8. W. Diffe and M. E. Hellman, "Privacy and Authentication: An Introduction to Cryptography," Proceedings of IEEE, Mar. 1979.
9. T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," IEEE Transaction on Information Theory, vol. IT-31. no. 4, pp. 469-472, July 1985.
10. H. Morita, C. H. Yang, "A Modular-Multiplication Algorithm using lookahead Determination," IEICE Trans. Fundamentals, vol. E76-A, no. 1, Jan. 1993.
11. P. Montgomary, "Modular Multiplication without Trial Division," Maths. of Computation, vol. 44, no. 170, pp. 519-521, Apr. 1985.
12. S. Kawamura and K. Hirano, "A Fast Modular-Arithmetic Algorithm Using a Residue Table," Eurocrypt 88, pp. 245-260. 1988.
13. J. Bos, M. Coster, "Addition Chain Heuristics," Proceedings CRYPTO '89, Springer Verlag Lecture Notes in Computer Science, pp. 400-407, 1989.
14. E. F. Brickell, D. M. Gordon, K. S. McCurley, and D. B. Wilson, "Fast Exponation with Precomputation(Extended Abstract)", In Proc. Eurocrypt '92, Balatonfured, Hungary, NCS 839. Springer Verlag, pp. 200-207, 1992.
15. C. H. Lim and P. J. Lee "More Flexible Exponentiation with Precomputation," In Advances in Cryptology-Crypto '94, LNCS 839, Springer Verlag, pp. 95-107, 1994.
16. S. A. Vanstone and R. J. Zuccherato, "Short RSA Keys and Third Generation," J. Crypto- graphy, pp. 101-114, 1995.
17. A. Bosselaers, R. Govaerts, J. Vandewalle, "Comparison of three modular reduction function," In Advances in Crypto'93, Springer Verlag, pp. 176-185, 1993.