情報保護學會論文誌
第12券 第6號, 2002. 12

XTR을 가장 효율적으로 구성하는 확장체

한 동 국*, 장 상 운*, 윤 기 순*, 장 남 수*, 박 영 호**, 김 창 한***

The Most Efficient Extension Field For XTR

Dong-kuk Han, Sang-woon Jang, Ki-sun Yoon, Nam-su Jang,
Young-Ho Park, Chang-han Kim

요 약

XTR은 유한체 $GF(p^6)$의 곱셈군의 부분군의 원소를 새롭게 표현하는 방법이며, 유한체 $GF(p^{6m})$으로도 일반화가 가능하다.[6,9] 본 논문은 XTR이 적용 가능한 확장체 중에서 최적 확정체를 제안한다. 최적 확장체를 선택하기 위해 일반화된 최적 확장체(Generalized Optimal Extension Fields : GOEFs)를 정의하며, 소수 p의 조건, $GF(p)$위에서 $GF(p^{2m})$을 정의하는 다항식, $GF(p^{2m})$에서 빠른 유한체 연산을 실현하기 위해서 $GF(p^{2m})$에서 빠른 곱셈 방법을 제안한다. 본 논문의 구현 결과로부터, $GF(p^{36}) \rightarrow GF(p^{12})$이 XTR을 위한 가장 효과적인 확장체이며, $GF(p^{12})$에서 $Tr(g)$이 주어질 때 $Tr(g^n)$을 계산하는 것은 평균적으로 XTR 시스템의 결과보다 두 배 이상 빠르다.[6,10] (32 bits, Pentium Ⅲ/700MHz에서 구현한 결과)

ABSTRACT

XTR is a new method to represent elements of a subgroup of a multiplicative group of a finite field $GF(p^6)$ and it can be generalized to the field $GF(p^{6m})$.[6,9] This paper progress optimal extention fields for XTR among Galois fields $GF(p^{6m})$ which can be aplied to XTR. In order to select such fields, we introduce a new notion of Generalized Opitimal Extention Fields(GOEFs) and suggest a condition of prime p, a defining polynomial of $GF(p^{2m})$ and a fast method of multiplication in $GF(p^{2m})$ to achieve fast finite field arithmetic in $GF(p^{2m})$. From our implementation results, $GF(p^{36}) \rightarrow GF(p^{12})$ is the most efficient extension fields for XTR and computing $Tr(g^n)$ given $Tr(g)$ in $GF(p^{12})$ is on average more than twice faster than that of the XTR system on Pentium Ⅲ/700MHz which has 32-bit architecture.[6,10]

Keyword : *XTR 공개키 시스템, Pseudo-Mersenne 소수, GOEF, 카라슈바 곱셈 방법*

Ⅰ. 서 론

타원 곡선 암호시스템을(ECC)을 제외하고 대부분의 공개키 시스템은 사이즈가 큰 키를 갖는다. 이러한 성질은 스마트 카드와 무선 통신과 같이 전력과 밴드 대여폭이 제한된 응용부분에서 비실용적이다. 많은 암호학자들은 제한된 하드웨어 환경에 적용 가능한 가장 효율적인 공개키 시스템을 ECC라고 생각하였다.

XTR 공개키 시스템은 Crypto 2000에서 소개되었다.[6] 안전성의 관점에서 볼 때, XTR은 부분군에서의 이산 대수 문제에 기반을 두고 있다. 그러나

* 고려대학교 정보보호대학원({christa, jangsw, ksyoon, michael, arra}@cist.korea.ac.kr)
** 세종사이버대학교 정보보호시스템공학과(youngho@cybersejong.ac.kr)
*** 세명대학교 정보보호 수리정보학과(chkim235@chollian.net)

XTR은 부분군의 원소를 표현하고 계산하는데 표준적인 방법을 사용하지 않으며, 전통적인 방법보다 대폭적인 통신상/계산상의 이점을 갖는다. 또한 XTR은 $GF(p^2)$을 이용하여 $GF(p^6)$의 안전성을 획득한 최초의 방법이며, $GF(p^6)$을 명확하게 구성하여 사용하지 않는다. [6]에서 보여지는 것과 같이, 1024비트 RSA의 안전성과 동일한 XTR은 임의의 소체 Z_p) 위에서의 ECC에 기반을 둔 암호시스템과 속도와 안전성 면에서 비슷하다. XTR의 공개키는 ECC보다 두 배만큼 사이즈가 크지만, 파라미터를 초기화하는 데에는 RSA와 ECC의 파라미터 초기화 시간보다 무시할 만큼의 작은 시간이 소요된다. 따라서 XTR은 스마트 카드와 무선 통신과 같은 응용부분에서 RSA나 ECC에 대한 훌륭한 대안이 될 수 있다. 그러므로 XTR을 위한 최적화된 확장체를 찾는 것은 의미 있는 일이 된다. $GF(p^{6m})$형태의 확장체를 이용하여 XTR이 일반화될 있고, 일반화에 대한 디자인이 [9]에서 간단하게 언급되었다. 그러나 [9]에서는 최적화된 확장체를 제시하지 않았다.

본 논문에서는, 32비트 워드 시스템에서 여러 확장체 $GF(p^{6m})$중 최적화된 확장체를 제안한다. 소수 p의 크기에 따라 다음의 다섯 가지 XTR을 위한 확장체를 살펴보면 충분하다.

□ **XTR을 위한 일반화된 확장체 :**
$GF(p^{6m}) \rightarrow GF(p^{2m})$

- $GF(p^6) \rightarrow GF(p^2)$, 소수 p의 크기는 약 170비트이고, m=1.
- $GF(p^{12}) \rightarrow GF(p^4)$, 소수 p의 크기는 약 85비트이고, m=2.
- $GF(p^{18}) \rightarrow GF(p^6)$, 소수 p의 크기는 약 64비트이고, m=3.
- $GF(p^{36}) \rightarrow GF(p^{12})$, 소수 p의 크기는 약 32비트이고, m=6.
- $GF(p^{66}) \rightarrow GF(p^{22})$, 소수 p의 크기는 약 16비트이고, m=11.

본 논문에서는 위의 다섯 가지 확장체의 기본연산에 대한 복잡도를 비교하고 XTR을 위한 최적 확장체를 선택한다. 또한 $Tr(g)$, $n \in Z$이 주어질 때 $Tr(g^n)$을 계산하는 단순 지수승(single exponentiation)의 비용도 고려한다. XTR에서[6,9] 가장 자주 사용되는 연산은 다음 세 가지 형태이다:

$x, y, z \in GF(p^{2m})$일 때, x^2, xy, $xy - yz^{p^m}$

위의 세 가지 연산은 XTR 단순 지수승의 속도를 향상시키는데 중요한 역할을 한다. $GF(p^{2m})$에서 기본 연산을 최적화하기 위해서, 소수 p와 $GF(p)$위에서 $GF(p^{2m})$을 정의하는 다항식을 선택하는데 있어서 다음의 세 가지 성질을 조건으로 명시한다.

1. p를 pseudo-Mersenne 소수로 선택한다. 즉, $\log_2 c \le \frac{1}{2} n$을 만족하는 c에 대하여 $p = 2^n \pm c$ 형태이며, 이것은 효율적인 부분체에서의 모듈러 reduction을 가능하게 한다.
2. $GF(p)$위에서 $GF(p^{2m})$을 정의하는 다항식을 이항 다항식(binomial)이나 $2m$-th all-one-polynomial (AOP)로 선택하여 확장체에서 효율적인 부분체에서의 모듈러 reduction을 가능하게 한다.
3. p의 크기를 프로세서의 워드사이즈보다 작고 동시에 비슷하게 선택하여 모든 부분체 연산이 프로세서의 빠른 정수 연산의 장점을 갖도록 한다.

세 번째 조건을 만족하기 위해서 m을 충분히 크게 선택한다. 그러면 위에서 설명한 바와 같이 프로세서의 워드사이즈보다 작고 가깝도록 p를 선택할 수 있다. 반면 $GF(p^{2m})$에서 다항식의 곱셈을 하는데 필요한 $GF(p)$에서의 곱셈과 덧셈의 수가 빠르게 증가한다. 보통 $GF(p)$에서의 덧셈은 고려하지 않으나, p가 작고 m이 크게 되면 $GF(p)$에서의 덧셈은 무시할 수 없게 된다. 본 논문에서는 $GF(p^{2m})$에서 카라슈바 형태의 다항식 곱셈방법(Karastuba-like method at polynomial multiplication)을 사용하여 부분체에서의 곱셈 수를 줄이고, $GF(p)$에서 덧셈 수를 줄이기 위해서 카라슈바 방법을 수정한다. 위에서 언급한 소수 p와 확장체 $GF(p^{2m})$에서 빠른 다항식 곱셈에 대한 고려 때문에 본 논문에서 제안하는 시스템은 단순 지수승 연산을 하는데 XTR[6,10]보다 평균적으로 두 배 이상 빠르다.

본 논문의 구성은 다음과 같다. 1절에서는 XTR 공개키 시스템에[6,9,10] 대해 간단하게 소개할 것이며, 3, 4절에서는 각각 일반화된 최적 확장체(GOEF)와 GOEF에서 효율적인 연산을 다룬다. 5절에서 본 논문의 구현 결과를 제시하고 이것을 기초로 하여 XTR을 위한 최적화된 확장체를 제안하며 마지막 절에서 결론을 맺는다.

Ⅱ. XTR 공개키 시스템에 대한 제고

이 절에서는 [6,9,10]의 몇 가지 결과들을 다시 살펴본다.

[정의 1]

$h \in GF(p^{6m})$에 대한 $GF(p^{2m})$위에서의 $Tr(h)$는 h의 $GF(p^{2m})$위에서 conjugates 의 합이다. 즉, $Tr(h) = h + h^{p^{2m}} + h^{p^{4m}}$ 이다.

XTR을 구성하는데 있어서, 소수 p, q와 자연수 m은 다음의 조건을 만족해야 한다 :

- p와 $2m+1$은 소수이고, $p \pmod{2m+1}$은 Z_{2m+1}에서 원시근(primitive root)이어야 한다.
- $\Phi_{6m}(p)$은 160비트 이상의 소수 q를 인수로 포함하여야 한다.

자연수 n에 대하여 $\Phi_n(X)$은 n-th cyclotomic 다항식이다. 위의 첫 번째 조건은 $GF(p^{2m})$에서 타입 Ⅰ의 최적 정규 기저(Optimal Normal Basis of Type Ⅰ)의 존재성을 보장한다.[11, 정리 5.2] 두 번째 조건에 의하여 위수가 q인 부분군은 $GF(p^{6m})$의 진부분체에 속하지 않고 오직 $GF(p^{6m})$에만 포함된다.[7, 보조정리 2.4]

[정의 2]

$c \in GF(p^{2m})$에 대하여, 다항식 $F(c, X) = X^3 - cX^2 + c^{p^m}X - 1 \in GF(p^{2m})[X]$은 $GF(p^{6m})$에서 h_0, h_1, h_2을 근(이 근들은 서로 틀리지 않아도 된다)으로 갖으며, 정수 n에 대하여 $c_n = h_0^n + h_1^n + h_2^n$라고 정의하자.

(Note)

만약 $F(c, X)$이 $GF(p^{2m})$위에서 기약 다항식이면, $c_n = Tr(h_0^n)$이다.

보조정리 1 [9, 보조정리 2.1]

i. $c = c_1$.
ii. $h_0h_1 + h_1h_2 + h_2h_0 = c^{p^m}$.
iii. $h_0h_1h_2 = 1$.
iv. 자연수 n에 대하여, $c_n = c_{np^m} = c_n^{p^m}$.
v. 모든 h_i에 대하여, $o(h_i) \mid p^{2m} - p^m + 1$이고 $h_i > 3$ 이거나, $h_i \in GF(p^{2m})$이다.
vi. $(c_n)_t = c_{nt} = (c_t)_n$.
vii. 정수 n에 대하여, $c_n \in GF(p^{2m})$.

따름정리 1 [9, 보조정리 2.3].

c, c_{n-1}, c_n, c_{n+1}이 주어져 있다고 하자.
i. $c_{2n} = c_n^2 - 2c_n^{p^m}$.
ii. $c_{n+2} = c * c_{n+1} - c^{p^m} * c_n + c_{n-1}$.
iii. $c_{2n-1} = c_{n-1} * c_n - c^{p^m} * c_n^{p^m} + c_{n+1}^{p^m}$.
iv. $c_{2n+1} = c_n * c_{n+1} - c * c_n^{p^m} + c_{n-1}^{p^m}$.

XTR에서 주어진 $Tr(g)$로부터 $Tr(g^n)$을 계산하는 알고리즘은 이산 대수 문제에 기반을 둔 공개키 시스템에서 g^n을 계산하는 것과 유사하다.

[정의 3]

$S_n(c) = (c_{n-1}, c_n, c_{n+1}) \in GF(p^{2m})^3$이라고 하자.

주어진 $c \in GF(p^{2m})$, $m \in Z^+$으로부터 $S_n(c)$을 계산하기 위해서, [6]의 알고리즘 2.3.7($m=1$인 경우)과 [9]의 알고리즘 4.2에서 제안되었다. 또 다른 단순 지수승 계산 방법은[10] 평균적으로 [6]의 알고리즘 2.3.7보다 35% 빠르다. $S_n(c)$을 계산하는 알고리즘에서[6,9,10] 가장 빈번하게 사용되는 연산은 보조정리 1의 i, iii, iv 이다. 따라서 $S_n(c)$을 계산하는 알고리즘의 복잡도는 $x, y, z \in GF(p^{2m})$에 대해 x^2, xy, $xy - yz^{p^m}$을 계산하는 복잡도에 의존한다.

보조정리 2.[10, 보조정리 2.2]

$x, y, z \in GF(p^2)$, $p = 2 \bmod 3$ 이라고 하자.

i. x^p을 계산하는 것은 연산량이 없다.
ii. x^2을 계산하는 것은 $GF(p)$에서 2번의 곱셈이 소요된다.
iii. xy을 계산하는 것은 $GF(p)$에서 2.5번의 곱셈이 소요된다.
iv. $xy - yz^p$을 계산하는 것은 $GF(p)$에서 3번의 곱셈이 소요된다.

보조정리 3.[9, 보조정리 4.3]

p와 $2m+1$은 소수이고, $p \pmod{2m+1}$은 Z_{2m+1}

에서 원시근이라고 하자. 그러면 $x, y, z \in GF(p^{2m})$에 대하여 다음이 성립한다.

i. x^{p^m}을 계산하는 것은 연산량이 없다.
ii. x^2을 계산하는 것은 $GF(p)$에서 곱셈하는 복잡도의 80%가 소요된다.
iii. xy을 계산하는 것은 $GF(p)$에서 $4m^2$번의 곱셈 이 소요된다.
iv. $xy-yz^{p^m}$을 계산하는 것은 $GF(p)$에서 $4m^2$번 의 곱셈이 소요된다.

(주의)

$GF(p^2)$에서 $S_n(c)$은 $GF(p)$에서 $7\log_2 n$의 곱셈으로 계산된다.[10]

[정리 1]

$c \in GF(p^{2m})$과 자연수 n이 주어져 있다고 하자. 그러면 $S_n(c)$은 $GF(p)$에서 $(2a+b)\cdot$

$\log_2 n$의 곱셈으로 계산된다. 여기서 a는 $x \in GF(p^{2m})$에 대하여 $GF(p)$에서 x^2을 계산할 때 필요한 곱셈 수이며, b는 $x, y, z \in GF(p^{2m})$에 대하여 $GF(p)$에서 $xy-yz^{p^m}$을 계산할 때 필요한 곱셈 수를 의미한다.

Ⅲ. 일반화된 최적 확장체(GOEF)

$GF(p^m)$에서 유한체 연산의 성능은 주로 소수 p와 $GF(p^m)$을 정의하는 다항식과 같은 확장체에 대한 파라미터의 선택에 의존한다. 곱셈에서 reduction 단계는 가장 많은 시간 복잡도를 갖는다. 따라서 reduction단계에서 복잡도를 줄이기 위해 다음과 같은 많은 방법들이 제안되었다.

[정의 4]

c이 양의 정수일 때, pseudo-Mersenne 소수는 $2^n \pm c$, $\log_2 c \le \lfloor (1/2)n \rfloor$ 형태의 소수를 말한다.

[정의 5][3]

최적 확장체(Optimal Normal Extention : OEF)는 다음의 조건을 만족하는 유한체 $GF(p^m)$을 말한다.

i. p는 pseudo-Mersenne 소수이다.
ii. $GF(p)$위에서 기약 다항식 $P(x)=x^m-w$이 존재한다.

[정리 2][11]

$m \ge 2$인 정수이고, $w \in GF(p)^*$라고 하자. 그러면 이항 다항식 x^m-w이 $GF(p)[x]$에서 기약이기 위한 필요충분조건은 다음의 두 조건이다.

i. m의 소인수는 $GF(p)$위에서 w의 위수 e를 나누고, $(p-1)/e$는 나누지 않는다.
ii. 만약 $m \equiv 0 \bmod 4$이면 $p \equiv 1 \bmod 4$이다.

추가적인 연산상의 장점을 갖는 두 가지 형태의 특별한 OEF가 있다. 소수 $p=2^n \pm 1$ 인 타입 I OEF은 매우 작은 복잡도로 부분체의 모듈로 reduction이 가능하다. 기약 이항 다항식 x^m-2을 갖는 타입 II OEF의 경우 확장체 모듈로 축소의 복잡도로 reduction가 가능하다. 확장체 모듈로 축소의 복잡도를 줄이는 또 다른 방법이 있다.

[정의 6]

다음 형태의 다항식 f_m을 m-th-all-one-polynomial (AOP) 라고 부른다.

$$f(x)=\Phi_{m+1}(x)=\frac{x^{m+1}-1}{x-1}$$
$$=x^m+x^{m-1}+\cdots+x+1$$

다음의 정리는 AOP가 언제 기약이 되는지를 보여준다.

[정리 3]

p를 소수라고 할 때, $f_m(x)$이 $GF(p)$위에서 기약이기 위한 필요충분조건은 $m+1$은 소수이고, $p(\bmod\ m+1)$은 Z_{m+1}에서 원시근이다.

[정리 4]

$m+1$은 소수이고, $p(\bmod\ m+1)$은 Z_{m+1}에서 원시근이고, p는 소수이거나 소수의 멱승이고 α를 m-th AOP의 근이라고 하자. 그러면 $\{\alpha, \alpha^p, \cdots, \alpha^{p^{m-1}}\}$은 $GF(p)$위에서 $GF(p^m)$의 기저가 된다. 또한 $\{\alpha, \alpha^p, \cdots, \alpha^{p^{m-1}}\}=\{\alpha, \alpha^2, \cdots, \alpha^m\}$이 성립한다.

(증명)

정리의 첫째 사항은 [11]에 의하여 성립한다. 둘째 사항을 증명하기 위하여 Z_{m+1}에서 $\{1, p, p^2 \cdots, p^{m-1}\}=\{1, \cdots, m\}$을 증명하면 충분하다.

$0 \le j \le i \le m-1$과 $p^i \equiv p^j \bmod(m+1)$이라고 하자. 그러면 $p^j(p^{i-j}-1) \equiv 0 \bmod(m+1)$이다. 따라서 $p^{i-j} \equiv 1 \bmod(m+1)$이고 $o(p) \mid (i-j)$이 성립한다. 그러나 $o(p)=m$($\because$ p는 mod $m+1$)에 대한 원시근)과 $i-j < m$에 의하여 $i=j$이다. 따라서 $\{1, p, \cdots, p^{m-1}\}$은 모두 서로 다르며, $\{1, p, p^2, \cdots, p^{m-1}\} = \{1, 2, \cdots, m\}$이다.

[정의 7]

집합 $A=\{\alpha, \alpha^2 \cdots, \alpha^m\}$이 $GF(p)$위에서 $GF(p^m)$의 기저를 이루면 A를 $GF(p)$위에서 $GF(p^m)$의 non-conventional 기저라고 부른다.

본 논문에서는 기약 다항식을 AOP로 하는 $GF(p^m)$에 대한 non-conventional 기저의 표현법을 사용한다. $\alpha^{m+1}=1, 1=-\alpha-\cdots-\alpha^m$ 두 성질 때문에 non-conventional 기저에서 확장체 모듈로 reduction속도를 증가시킬 수 있다. 자세한 설명은 4.3절에서 이루어진다. 새로운 형태의 Galois 체를 소개한다.

[정의 8]

일반화된 최적 확장체(Generalized Optimal Extension Field : GOEF)은 다음의 조건을 만족하는 유한체 $GF(p^m)$이다.

i. p는 pseudo-Mersenne 소수이다.
ii. 이항 다항식 x^m-w과 m-th AOP 둘 중에 하나는 기약이다.

한 쌍의 pseudo-Mersenne 소수와 기약 다항식을 가지고 GOEF를 구성할 수 있다. GOEF를 구성하기 위해서, 기약 다항식으로서 이항 다항식이나 AOP의 선택의 문제가 남아있다. 그러나 GOEF에서 기약 다항식의 결정은 pseudo-Mersenne 소수의 선택에 관련이 있다. 기약 다항식의 선택은 확장체 모듈로 reduction를 하는데 소요되는 복잡도를 결정한다. 그리고 pseudo-Mersenne 소수의 선택은 부분체 모듈로 reduction에 영향을 준다. 따라서 한 쌍의 pseudo-Mersenne 소수와 기약 다항식을 선택하는 것은 매우 중요한 일이다. 기약 다항식 f에 따라 $A \in GF(p^m)$를 다항식 기저나 정규 기저로 표현하는 것을 살펴보자.

i. $f(x)$이 이항 다항식일 때
$A(\alpha)=a_{m-1}\alpha^{m-1}+\cdots+a_1\alpha+a_0$,
ii. $f(x)$이 AOP 일 때
$A(\alpha)=a_{m-1}\alpha^m+\cdots+a_1\alpha^2+a_0\alpha$,
(단, $a_i \in GF(p)$, α는 $f(x)$의 근)

p를 워드 사이즈보다 작게 선택하기 때문에, $A(x)$를 m개의 레지스터로 표현할 수 있다.

Ⅳ. GOEF에서 효율적인 연산

본 절은 GOEF가 특별한 경우일 때, 유한체 $GF(p^m)$에서 연산을 위한 기본구성을 기술한다.

4.1 덧셈과 뺄셈

두 유한체의 원소를 더하고 빼는 것은 다항식 기저나 정규 기저 표현에 의한 계수를 더하고 뺌으로써 구현이 되며 필요시에는 계산된 결과로부터 p를 더하거나 빼어서 모듈로 reduction를 실행한다.

4.2 곱셈

유한체 곱셈은 두 단계로 구분할 수 있다. 첫째 단계는, 두 유한체 원소 $A(\alpha)$와 $B(\alpha)$의 곱을 이행한다. $f(x)$를 이항 다항식으로 설정하고 다항식 기저를 사용할 경우 중간단계의 곱 $C'(\alpha)$의 차수는 $2m-2$보다 작거나 같다.

$$C'(\alpha)=A(\alpha)B(\alpha)=c'_{2m-2}\alpha^{2m-2}+\cdots c'_1\alpha+c'_0$$
$$(c'_i \in GF(p),\ i=0,1,\cdots,2m-2)$$

AOP를 사용할 경우 중간단계의 곱 $C'(\alpha)$의 차수는 $2m$보다 작거나 같다.

$$C'(\alpha)=A(\alpha)B(\alpha)=c'_{2m-2}\alpha^{2m}+\cdots c'_1\alpha^3+c'_0\alpha^2$$
$$(c'_i \in GF(p),\ i=0,1,\cdots,2m-2)$$

계수 $c'_i, (i=0,1,\cdots,2m-2)$를 계산하는데 school-book method를 사용하면 $GF(p)$에서 m^2번의 곱셈과 $(m-1)^2$번의 덧셈으로 가능하다. 많은 공개키 알고리즘에서 유한체 원소를 곱하는 것은 매우 중요한 연산이므로 본 논문에서는 계수를 구하는데 Karastuba-

like Method(카라슈바 방법)를[5] 사용하는데 이 연산 방법은 복잡도가 $O(m^{1.59})$이고 부분체에서 더 많은 덧셈비용이 필요하다.[1] 이 방법을 사용하면 부분체에서 곱셈비용에 대한 상당한 장점을 갖는다. 일반적으로 $GF(p)$에서 덧셈과 뺄셈의 비용은 고려되지 않으나 소수 p의 크기가 보통의 프로세서의 워드사이즈보다 작게 되도록 m을 선택하면 덧셈과 뺄셈의 비용은 반드시 고려되어야 한다. 예를 들어서, $GF(p^6)$에서 곱셈을 계산하는데 $GF(p)$에서 62번의 덧셈/뺄셈과 18번의 곱셈이 필요하지만, $GF(p^{22})$의 두 원소를 곱하는 데에는 $GF(p)$에서 722번의 덧셈/뺄셈과 147번의 곱셈이 든다[부록 표 2 참조]. [표 6]의 결과로부터 $GF(p^6)$에서는 $T_{mul}/T_{add+sub}=5.76$, $GF(p^{22})$에서는 $T_{mul}/T_{add+sub}=1.7$이다. 여기서 T_{mul}과 $T_{add+sub}$은 부분체에서 곱셈과 덧셈/뺄셈을 하는데 요구되는 시간을 말한다. 카라슈바 방법의 자세한 기술은 지면이 부족한 관계로 본 논문에서 다루지 않는다. 대신에 부록에서 수정된 카라슈바 방법으로 $GF(p^{12})$에서 두 원소를 곱하는 예를 보인다.

또한, 유한체 원소의 곱셈 방법으로 Schnhage-Strassen FFT 방법이 있는데 이것은 $O(m(\log_2 m)(\log_2\log_2 m))$의 복잡도를 갖는다 그러나 이 방법은 대략 $m \geq 302$일 때 고전적인 곱셈 방법보다 좋은 성능을 발휘한다.

다음 소절에서는 $C(\alpha)\equiv C'(\alpha) \bmod f(\alpha)$, $GF(p^m) \in C(\alpha)$을 계산하는 효율적인 방법을 제시한다.

4.3 확장체 모듈로 Reduction

유한체 원소를 곱한 후 중간단계의 결과값 $C'(\alpha)$를 얻는다. 일반적으로 $f(x)$가 이항 다항식일 때 $C'(\alpha)$의 차수는 m보다 크고, AOP일 때에는 $m+1$보다 클 것이다. 이 경우 모듈로 reduction을 이행한다. 모듈로 reduction을 하는 자연스러운 방법은 기약 다항식으로 나머지를 가지고 long division을 하는 것이다. 그러나 기약 다항식이 특별한 형태일 경우 모듈로 reduction은 계산적인 효율성을 가능하게 한다.

기약 다항식이 이항 다항식일 때 : x^m-w

[정리 5][3]

차수가 $2m-2$보다 작거나 같은 $GF(p)$위에서 주어진 다항식 $C'(\alpha)$에 대하여, 많아야 $GF(p)$에서 w를 $m-1$번 곱하고, $m-1$번 더하는 연산으로 $C'(\alpha)$은 모듈로 $f(x)=x^m-w$에 의해 reduction이 가능하다.

축소된 후 다항식에 대한 일반적인 표현은 다음과 같다 :

$$C(\alpha)\equiv c'_{m-1}\alpha^{m-1}+(wc'_{2m-2}+c'_{m-2})\alpha^{m-1} +\cdots+(wc'_{m+1}+c'_1)\alpha+(wc'_m+c'_0) \bmod f(\alpha)$$

최적화를 고려하면, 기약 이항 다항식을 x^m-2으로 선택하면 곱셈은 계수의 자리 이동(shifting)으로서 구현이 가능하다.

기약 다항식이 AOP일 때 :

$$x^m+x^{m-1}+\cdots+x+1$$

[정리 6]

차수가 $2m$보다 작거나 같은 $GF(p)$위에서 주어진 다항식 $C'(\alpha)$에 대하여, 많아야 $GF(p)$에서 $m-2$번의 덧셈과, m번의 뺄셈으로 모듈로 $f(x)=x^m+\cdots+1$에 의해 $C'(\alpha)$은 reduction이 가능하다.

$\alpha^{m+1}=1$, $1=-\alpha-\alpha^2-\cdots-\alpha^m$ 두 성질을 이용하면, reduction된 후 일반적인 표현은 다음과 같이 주어진다 :

$$\begin{aligned}C(\alpha)\equiv&(c'_{m-2}-c'_{m-1})\alpha^m\\&+(c'_{m-3}-c_{m-1}+c'_{2m-2})\alpha^{m-1}\\&+\cdots+(c'_0-c'_{m-1}+c'_{m+1})\alpha^2\\&+(c'_m-c_{m-1})\alpha \bmod f(\alpha)\end{aligned}$$

(비교)

일반적으로 곱셈은 뺄셈보다 비용이 더 많이 든다. 정리 5에서 모듈로 reductioon은 많아야 $GF(p)$에서 w를 $m-1$번의 곱셈이 필요한데, $w=2$이면 위의 두 방법에 의한 모듈로 reduction의 복잡도는 거의 비슷하다. 그러나 $w>2$이면 AOP를 사용할 때의 확장체 모듈로 reduction이 더욱 효율적이다.

Reduction 단계를 결합하거나 또는 지연하는 방법 :

$a, b \in GF(p)$을 보통의 곱셈을 할 때, 정수 곱셈과

정수 reduction 단계가 필요하다. 일반적으로 school-book 방법을 이용하면 m^2의 곱셈이 필요하다. 즉 m^2번의 정수 reduction 단계가 이행되어야 한다. m이 크다면 reduction 단계의 비용은 매우 클 것이다. 따라서 reduction 단계의 수를 줄이는 방법에 의해 전체 작업량에 기여를 할 수 있게 된다. 가능한 많은 수의 개개의 곱 항을 덧셈과 뺄셈에 의하여 결합함으로써 reduction 단계의 수를 줄일 수 있게 되는데, 그러면 모듈로 p로 축적된 합을 단 한 번의 reduction으로 이행할 수 있다. 따라서 단지 m번의 reduction 단계가 필요하게 된다. 중간 결과 값이 절대값으로 p^2보다 크기 때문에 p가 워드 사이즈와 가까운 소수로서 선택되면, 결과가 되는 최종 reduction 비용은 원래의 reduction비용 보다 크다. 본 논문의 구현 결과에 의하면, p가 워드 사이즈이고 m이 클 경우는 성능이 좋지 못하다. 최신 워크스테이션 CPU는 레지스터 폭에 맞는 피연산자에 대한 정수 연산을 최적화되도록 되어 있어서 결합 단계에서 생성되는 워드 크기의 두 배 혹은 세 배가 되는 정수 연산은 하나의 워드로 표현되는 정수 연산보다 훨씬 비효율적이다. Reduction 단계를 결합하거나 지연하는 것이 전혀 새로운 것은 아니다. 초기의 응용으로서 [4]에 예가 있다.

4.4 모듈로 Reduction에 대한 빠른 부분체 곱셈

일반적으로 빠른 부분체 곱셈은 $GF(p^m)$에서 빠른 곱셈을 하는데 필수적이다. $GF(p)$에서 부분체 연산은 표준적인 모듈로 정수 기법에 의하여 구현된다. 효율적인 GOEF 연산을 위해, 부분체 연산의 최적화는 성능에 매우 중대한 영향을 끼친다. 최신 워크스테이션 CPU는 레지스터 폭에 맞는 피연산자에 대한 정수 연산을 최적화되어 있다. 하나의 레지스터에서 정수로 표현되는 원소를 갖는 부분체를 구성함으로써 GOEF는 이러한 장점을 갖는다.

두 개의 단순 워드(single-word) 정수를 곱셈을 실행하고 일반적으로 결과로서 두 배의 워드크기의 정수를 얻는다. 계산을 완료하기 위해서 모듈로 reduction을 해야 한다. 빠른 모듈로 reduction은 모듈로가 $2^n \pm c$(c는 작은 정수)일 때 가능하다. 이러한 형태의 정수는 나눗셈 없이 모듈로 reduction을 가능하게 한다. 연산자 $\langle\langle$과 $\rangle\rangle$는 각각 왼쪽으로 계수 자리 이동(left shift)과 오른쪽으로 계수 자리 이동(right shift)을 의미한다.

알고리즘 : Reduction modulo

$\log_2 c \le \frac{1}{2} n$에 대하여, $p = 2^n - c$.

입력 : 이진법으로 표현한 수 $x < p^2$와 modulus p.
출력 : $r \equiv x \bmod p$.

1. $q_0 \leftarrow (x \rangle\rangle n)$, $r_0 \leftarrow x - (q_0 \langle\langle n)$, $r \leftarrow r_0$, $i \leftarrow 0$
2. $q_i > 0$인 동안 다음을 수행한다 :
   2.1 $q_{i+1} \leftarrow q_i c \rangle\rangle n$, $r_{i+1} \leftarrow q_i c - (q_{i+1} \langle\langle n)$.
   2.2 $i \leftarrow i+1$, $r \leftarrow r + r_i$
3. $r \ge p$인 동안 다음을 수행한다 : $r \leftarrow r - p$.
4. 반환 (r).

(주의)

$p = 2^n + c$(c는 $\log_2 c \le (1/2)n$을 만족하는 양의 정수)이면 위의 알고리즘은 다음과 같이 수정될 수 있다 : 단계 2.2에서 $r \leftarrow r + r_i$을 $r \leftarrow r + (-1)^i r_i$으로 대체한다.

알고리즘 1은 while 반복문을 최대 두 번의 반복으로 종료되므로, 두 번의 c에 의한 곱셈이 요구된다. 실제로 이것은 명시적인 나눗셈을 수행하는 데 성능을 매우 향상시킨다. $p = 2^n + c$을 사용하면 $p = 2^n - c$ 을 사용할 때보다 두 번째 while 반복문(단계 3)의 반복 횟수를 줄일 수 있다. 위의 알고리즘은 계수의 자리 이동, 덧셈, 뺄셈, c에 의한 곱셈을 연산으로 가지고 있고 c가 잘 선택되면 c에 의한 곱셈은 계수의 자리 이동으로 대체될 수 있기 때문에 뺄셈 수를 줄이는 것은 의미 있는 일이 된다.

V. 구현 결과

다양한 유한체에서 XTR을 구현하였으며, 전 절에서 소개된 기법을 사용하여 전형적인 마이크로 프로세서(Pentium Ⅲ/700MHz, 332-bit μP : Window 2000, MSVC)를 가지고 XTR 연산을 실행하였다.

5.1 XTR에 대한 응용

본 절에서는 먼저 [표 1]에서 보여지는 것과 같이 소수 p와 기약 다항식으로서 XTR을 위한 최적화된 파라미터를 제안한다. [표 1]에서 소수 q는 160비트 이상을 선택하여야 하나 pseudo-Mressen 소수 p에 의존하여 q을 선택해야 하는 구현상의 편의때문에 300 비트 정도의 소수를 일괄적으로 선택하였다.

〔표 1〕 XTR을 위한 확장체 구성

확장체	표수 p	$f(x)$	q의 사이즈
$GF(p^6) \rightarrow GF(p^2)$	$2^{174}+7$	AOP	323 비트
$GF(p^{12}) \rightarrow GF(p^4)$	$2^{88}+7$	AOP	234 비트
$GF(p^{18}) \rightarrow GF(p^6)$	$2^{61}+15$	AOP	354 비트
$GF(p^{36}) \rightarrow GF(p^{12})$	$2^{30}+3$	AOP	302 비트
$GF(p^{66}) \rightarrow GF(p^{22})$	$2^{16}-17$	AOP	320 비트

〔표 2〕 카라슈바 방법을 이용할 때 $GF(p^{2m})$에서 곱셈수

확장체	$GF(p)$에서 곱셈	$GF(p)$에서 덧셈	$GF(p)$에서 뺄셈
$GF(p^6) \rightarrow GF(p^2)$	3	5	2
$GF(p^{12}) \rightarrow GF(p^4)$	9(16)	13(11)	8(4)
$GF(p^{18}) \rightarrow GF(p^6)$	18(36)	39(29)	23(6)
$GF(p^{36}) \rightarrow GF(p^{12})$	54(144)	130(131)	118(12)
$GF(p^{66}) \rightarrow GF(p^{22})$	147(484)	345(461)	377(22)

〔표 3〕 XTR을 위한 확장체 구성

확장체	소수 p	$S_n(c)$(msec)
$GF(p^6) \rightarrow GF(p^2)$	$2^{174}+7$	10.174
$GF(p^{12}) \rightarrow GF(p^4)$	$2^{88}+7$	28.393
$GF(p^{18}) \rightarrow GF(p^6)$	$2^{61}+15$	50.276
$GF(p^{36}) \rightarrow GF(p^{12})$	$2^{30}+3$	4.932
$GF(p^{66}) \rightarrow GF(p^{22})$	$2^{16}-17$	6.789

본 논문에서는 $GF(p^{2m})$에서 곱셈의 효율성을 높 이기 위해 카라슈바 방법을 사용하였다. 〔표 2〕는 $GF(p^{2m})$의 두 원소를 곱하는 데 필요한 $GF(p)$에서 곱셈, 덧셈, 뺄셈의 수를 보여주고 있다. 여기서 기약 다항식은 AOP가 사용되었다. 괄호 안의 수치는 school-book 방법에 의한 수를 말한다.

〔표 3〕에서는, 〔표 1, 2, 6〕의 값으로부터 선형 회귀식 $S_n(c)$을 계산하는 결과를 제시한다.

$GF((2^{30}+3)^{12})$에서 XTR 단순 지수승의 속도는 원래의 XTR 단순 지수승보다 두 배 이상 빠름을 알 수 있다.[10]

〔표 4〕의 결과로부터, XTR 단순 지수승의 속도를 향상시키기 위해 pseudo-Mersenne 소수와 카라슈바 방법이 필요하다는 것을 알게 된다.

마지막으로, 〔표 5〕에서 XTR을 위한 GOEF의 파라미터를 추천한다.

Ⅵ. 결 론

본 논문에서는, $GF(p^m)$에서 유한체 연산의 속도를 빠르게 하는 다양한 기법을 제시하였으며, 유한체의 클래스 GOEF를 도입하였으며 GOEF는 마이크로프로세서에서 유한체 연산에 대해 잘 알려진 최적화를 이용한다. 본 논문에서 제시된 주요 개선점은 유한체의 원소에 대한 곱셈과 유한체 연산을 빠르게

〔표 4〕 $GF(p^2)$에서 XTR[6,10]과 $GF(p^{12})$에서 XTR의 비교

확장체	소수 p	곱셈 방법	$f(x)$	$S_n(c)$(msec)
$GF(p^6) \rightarrow GF(p^2)$	general	카라슈바 방법[10]	AOP	11.910
	$2^{174}+7$	카라슈바 방법[10]	AOP	10.174
$GF(p^{36}) \rightarrow GF(p^{12})$	general	Schoolbook 방법	AOP	129.338
	general	카라슈바 방법(부록)	AOP	67.822
	$2^{30}+3$	Schoolbook 방법	AOP	8.218
	$2^{30}+3$	카라슈바 방법(부록)	AOP	4.932

〔표 5〕 $GF(p^{12})$에서 추천되는 소수와 $f(x)$

확장체	소수 p	$f(x)$	q의 사이즈
$GF(p^{36}) \rightarrow GF(p^{12})$	$2^{30}+3$	$x^{12}-2$	302 비트
	$2^{30}+7$, $(7=2^3-1)$	AOP	355 비트
	$2^{30}+129$, $(129=2^7+1)$	AOP	351 비트
	$2^{30}-257$, $(257=2^8+1)$	AOP	269 비트
	$2^{30}-513$, $(513=2^9+1)$	AOP	341 비트
	$2^{30}-513$	$x^{12}-3$	341 비트

〔표 6〕 GOEF에서 연산시간(700MHz)

확장체	소수 p	곱셈 (μsec)	덧셈 (μsec)	뺄셈 (μsec)
$GF(p^6) \rightarrow GF(p^2)$	$2^{174}+7$	7.784	0.761	0.431
$GF(p^{12}) \rightarrow GF(p^4)$	$2^{88}+7$	4.095	0.53	0.67
$GF(p^{18}) \rightarrow GF(p^6)$	$2^{61}+15$	3.365	0.5	0.31
$GF(p^{36}) \rightarrow GF(p^{12})$	$2^{30}+3$	0.131	0.006	0.006
$GF(p^{66}) \rightarrow GF(p^{22})$	$2^{16}-17$	0.0507	0.006	0.006

하기 위한 파라미터의 주의 깊은 선택이다. 위의 결과를 가지고, XTR을 위한 최적 확장체 즉, $GF(p^{36}) \rightarrow GF(p^{12})$를 제시하였다. $GF(p^{12})$을 정의하는 기약다항식은 12-th AOP이며 좋은 소수의 후보는 $2^{30}+3$이다. 구현의 결과로부터 제안한 유한체는 $Tr(g^n)$을 계산하는데 XTR보다[6] 두 배 이상 빠르다. 키 사이즈는 본래 XTR과 같다. 따라서 제안한 XTR을 위한 최적 확장체는 SSL/TLS(Secure Socket Layers, Transport Layer Security)와 공개키 기반 스마트카드, WAP/WTLS(Wireless Application Protocol, Wireless Transport Layer Security)와 같은 응용분야에서 RSA나 ECC에 대한 대안으로서 XTR보다[6] 더욱 우수하다.

참 고 문 헌

〔1〕 Aho, A., Hopcroft, J., Ullman,J., *The Design and Analysis of Computer Algorithms.*, Addison-Weseley, Reading Mass, 1974.

〔2〕 Bach, E., Shallit, J., *Algorithmic Number Theory., Vol. 1*, The MIT Press, Mass, 1996.

〔3〕 Bailey. D. V. and Paar C, Optimal extension fields for fast arithmetic in public-key algorithms., *Crypto '98, Springer-Verlag,* pp.472~485, 1998.

〔4〕 H. Cohen, A. K. Lenstra, Implementation of a new primality test., *Math. Comp.48 (1987)* 103-121.

〔5〕 D. E. Knuth, The art of computer programming., *Volume 2, Seminumerical Algorithm, second edition*, Addison-Wesley, 1981.

〔6〕 A. K. Lenstra, E. R. Verheul, The XTR public key system., *Proceedings of Crypto 2000, LNCS 1880, Springer-Verlag*, 2000, 1-19; available from www.ecstr.com.

〔7〕 A. K. Lenstra, Using Cyclotomic Polynomial to Construct Efficient Discrete Logarithm Cryptosystems over Finite Fields., *Proceedings of ACISP 1997, LNCS 1270, Springer-Verlag*, 1997, 127-138.

〔8〕 A. K. Lenstra, Lip 1.1, available at www.ecstr.com.

〔9〕 Seongan Lim, Seungjoo Kim, Ikkwon Yie, Jaemoon Kim, Hongsub Lee, XTR Extended to $GF(p^{6m})$. *Proceedings of SAC 2001, 317-328, LNCS 2259, Springer-Verlag, 2001, 125-143.*

〔10〕 Martijn Stam, A. K. Lenstra, Speeding Up XTR, *Proceedings of Asiacrypt 2001, LNCS 2248, Springer-Verlag*, 2001, 125-143; available at www.ecstr.com.

〔11〕 A. J. Menezes, Applications of Finite Fields., Waterloo, 1993.

〔12〕 S. B. Mohan and A. S. Adiga, Fast Algorithms for Implementating RSA Public Key Cryptosystem., *Electronics Letters,(21917)* :761,1995.

〔13〕 S. Oh, S. Hong, D. Choen, C. Kim, J. Lim and M. Sung, *An Extension Field of Characterstic Greater than Two and its Application. Technical Report 99-2, CIST, 1999.* Availiable from http://cist.korea.ac.kr/.

부 록

1. $GF(p)$에서 연산 성능의 비교

처음 세 개의 결과는 freelip version 1.1으로 구성되어[8] 있고, 마지막 두 개는 본 논문의 최적화 결과이다.

2. $GF(p^{12})$에서 카라슈바 방법에 의한 두 원소의 곱셈

$GF(p^{12})$의 두 원소 A, B에 대하여,

$A = a_1x + a_2x^2 + \cdots + a_{12}x^{12}$,

$B = b_1x + b_2x^2 + \cdots + b_{12}x^{12}$ $(a_i, b_j \in GF(p))$라고 하자. 기약 다항식으로 AOP를 사용하였다.

$C = c_1x + c_2x^2 + \cdots + c_{12}x^{12} (\mathrm{mod}\ 1 + x + \cdots + x^{12})$을 계산한다. 다음과 같은 사전 계산량이 필요하다.

[표 7]은 A, B를 곱하고 mod $1 + \cdots + x^{12}$으로 reduction 후 계수를 기술하고 있다. [표 8]은 곱셈과 reduction단계를 결합하여 $GF(p)$에서 총 연산량을 설명하고 있다.

STEP 1

$G_1 = a_1b_1$ $G_2 = a_2b_2$ $G_3 = G_{17}G_{18}$
$G_4 = G_3 - G_1 - G_2$ $G_5 = a_3b_3$ $G_6 = a_4b_4$
$G_7 = G_{19}G_{20}$ $G_8 = G_7 - G_5 - G_6$ $G_9 = (a_1 + a_3)(b_1 + b_3)$
$G_{10} = (a_2 + a_4)(b_2 + b_4)$ $G_{11} = (G_{17} + G_{19})(G_{18} + G_{20})$ $G_{12} = G_9 - G_1 - G_5$
$G_{13} = G_{11} - G_4 - G_8$ $G_{14} = G_{10} - G_2 - G_6$ $G_{15} = G_{12} + G_2$
$G_{16} = G_{14} + G_5$ $G_{17} = a_1 + a_2$ $G_{18} = b_1 + b_2$
$G_{19} = a_3 + a_4$ $G_{20} = b_3 + b_4$

STEP 2

$H_1 = a_5b_5$ $H_2 = a_6b_6$ $H_3 = H_{17}H_{18}$
$H_4 = H_3 - H_1 - H_2$ $H_5 = a_7b_7$ $H_6 = a_8b_8$
$H_7 = H_{19}H_{20}$ $H_8 = H_7 - H_5 - H_6$ $H_9 = (a_5 + a_7)(b_5 + b_7)$
$H_{10} = (a_6 + a_8)(b_6 + b_8)$ $H_{11} = (H_{17} + H_{19})(H_{18} + H_{20})$ $H_{12} = H_9 - H_1 - H_5$
$H_{13} = H_{11} - H_4 - H_8$ $H_{14} = H_{10} - H_2 - H_6$ $H_{15} = H_{12} + H_2$
$H_{16} = H_{14} + H_5$ $H_{17} = a_5 + a_6$ $H_{18} = b_5 + b_6$
$H_{19} = a_7 + a_8$ $H_{20} = b_7 + b_8$

STEP 3

$I_1 = I_{17}I_{18}$ $I_2 = I_{21}I_{22}$ $I_3 = (G_{17} + I_{17})(G_{18} + I_{18})$
$I_4 = I_3 - I_1 - I_2$ $I_5 = I_{19}I_{20}$ $I_6 = I_{23}I_{24}$
$I_7 = I_{25}I_{26}$ $I_8 = I_7 - I_5 - I_6$ $I_9 = I_{27}I_{28}$
$I_{10} = I_{29}I_{30}$ $I_{11} = (I_{27} + I_{29})(I_{28} + I_{30})$ $I_{12} = I_9 - I_1 - I_5$
$I_{13} = I_{11} - I_4 - I_8$ $I_{14} = I_{10} - I_2 - I_6$ $I_{15} = I_{12} + I_2$
$I_{16} = I_{14} + I_5$ $I_{17} = a_1 + a_5$ $I_{18} = b_1 + b_5$
$I_{19} = a_3 + a_7$ $I_{20} = b_3 + b_7$ $I_{21} = a_2 + a_6$
$I_{22} = b_2 + b_6$ $I_{23} = a_4 + a_8$ $I_{24} = b_4 + b_8$
$I_{25} = G_{19} + I_{19}$ $I_{26} = G_{20} + I_{20}$ $I_{27} = I_{17} + I_{19}$
$I_{28} = I_{18} + I_{20}$ $I_{29} = I_{21} + I_{23}$ $I_{30} = I_{22} + I_{24}$

STEP 4

$J_1 = a_9b_9$ $J_2 = a_{10}b_{10}$ $J_3 = J_{17}J_{18}$
$J_4 = J_3 - J_1 - J_2$ $J_5 = a_{11}b_{11}$ $J_6 = a_{12}b_{12}$
$J_7 = J_{19}J_{20}$ $J_8 = J_7 - J_5 - J_6$ $J_9 = J_{21}J_{22}$
$J_{10} = J_{23}J_{24}$ $J_{11} = (J_{17} + J_{19})(J_{18} + J_{20})$ $J_{12} = J_9 - J_1 - J_5$
$J_{13} = J_{11} - J_4 - J_8$ $J_{14} = J_{10} - J_2 - J_6$ $J_{15} = J_{12} + J_2$
$J_{16} = J_{14} + J_5$ $J_{17} = a_9 + a_{10}$ $J_{18} = b_9 + b_{10}$
$J_{19} = a_{11} + a_{12}$ $J_{20} = b_{11} + b_{12}$ $J_{21} = a_9 + a_{11}$
$J_{22} = b_9 + b_{11}$ $J_{23} = a_{10} + a_{12}$ $J_{24} = b_{10} + b_{12}$

STEP 5

$K_1 = K_{17}K_{18}$ $K_2 = K_{21}K_{22}$ $K_3 = K_{29}K_{30}$
$K_4 = K_3 - K_1 - K_2$ $K_5 = K_{19}K_{20}$ $K_6 = K_{23}K_{24}$
$K_7 = K_{31}K_{32}$ $K_8 = K_7 - K_5 - K_6$ $K_9 = K_{25}K_{26}$
$K_{10} = K_{27}K_{28}$ $K_{11} = (K_{25} + K_{27})(K_{26} + K_{28})$ $K_{12} = K_9 - K_1 - K_5$
$K_{13} = K_{11} - K_4 - K_8$ $K_{14} = K_{10} - K_2 - K_6$ $K_{15} = K_{12} + K_2$
$K_{16} = K_{14} + K_5$ $K_{17} = a_1 + a_9$ $K_{18} = b_1 + b_9$
$K_{19} = a_3 + a_{11}$ $K_{20} = b_3 + b_{11}$ $K_{21} = a_2 + a_{10}$
$K_{22} = b_2 + b_{10}$ $K_{23} = a_4 + a_{12}$ $K_{24} = b_4 + b_{12}$
$K_{25} = K_{17} + K_{19}$ $K_{26} = K_{18} + K_{20}$ $K_{27} = K_{21} + K_{23}$
$K_{28} = K_{22} + K_{24}$ $K_{29} = K_{17} + K_{21}$ $K_{30} = K_{18} + K_{22}$
$K_{31} = K_{19} + K_{23}$ $K_{32} = K_{20} + K_{24}$

STEP 6

$L_1 = (K_{17} + a_5)(K_{18} + b_5)$ $L_2 = (K_{21} + a_6)(K_{22} + b_6)$ $L_3 = (K_{29} + H_{17})(K_{30} + H_{18})$
$L_4 = L_3 - L_1 - L_2$ $L_5 = (K_{19} + a_7)(K_{20} + b_7)$ $L_6 = (K_{23} + a_8)(K_{24} + b_8)$
$L_7 = (K_{31} + H_{19})(K_{32} + H_{20})$ $L_8 = L_7 - L_5 - L_6$ $L_9 = L_{17}L_{18}$
$L_{10} = L_{19}L_{20}$ $L_{11} = (K_{17} + L_{19})(L_{18} + L_{20})$ $L_{12} = L_9 - L_1 - L_5$
$L_{13} = L_{11} - L_4 - L_8$ $L_{14} = L_{10} - L_2 - L_6$ $L_{15} = L_{12} + L_2$
$L_{16} = L_{14} + L_5$ $L_{17} = I_{27} + J_{21}$ $L_{18} = I_{28} + J_{22}$
$L_{19} = I_{29} + J_{23}$ $L_{20} = I_{30} + J_{24}$

STEP 7

$M_1 = I_1 - G_1 - H_1 + G_{16}$ $M_2 = I_4 - G_4 - H_4 + G_8$
$M_3 = I_{15} - G_{15} - H_{15} + G_6$ $M_4 = I_{16} - G_{16} - H_{16} + H_1$
$M_5 = I_8 - G_8 - H_8 + H_4$ $M_6 = I_6 - G_6 - H_6 + H_{15}$
$Q = H_{13} + K_{13} - G_{13} - J_{13}$

〔표 7〕 Reduction 단계에서 연산량

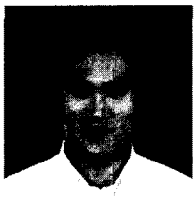
차수	계 수	덧셈	뺄셈
1	$H_{16} + K_{16} + L_1 - K_1 - H_1 - M_1 - J_{16} - Q$	2	5
2	$G_1 + H_8 + K_8 + L_4 - K_4 - H_4 - M_2 - J_8 - Q$	3	5
3	$G_4 + H_6 + K_6 + L_{15} - K_{15} - H_{15} - M_3 - J_6 - Q$	3	5
4	$G_{15} + L_{13} - K_{13} - H_{13} - I_{13} + G_{13} + H_{13} - Q$	3	4
5	$G_{13} + H_1 + L_{16} - K_{16} - H_{16} - M_4 + J_1 - Q$	3	4
6	$M_1 + H_4 + L_8 - K_8 - H_8 - M_5 + J_4 - Q$	3	4
7	$M_2 + H_{15} + L_6 - K_6 - H_6 - M_6 + J_{15} - Q$	3	4
8	$M_3 + J_{13} - Q$	1	1
9	$I_{13} - G_{13} - H_{13} + J_{16} - Q$	1	3
10	$K_1 - G_1 - J_1 + M_4 + J_8 - Q$	2	3
11	$K_4 - G_4 - J_4 + M_5 + J_6 - Q$	2	3
12	$K_{15} - G_{15} - J_{15} + M_6 - Q$	1	3
합 계		27	44

〔표 8〕 총 연산량

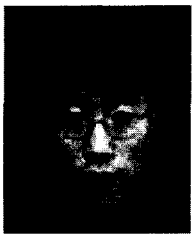
단 계	1	2	3	4	5	6	7	8	총합계
곱 셈	9	9	9	9	9	9	0	0	54
덧 셈	12	12	20	12	20	20	7	27	130
뺄 셈	10	10	10	10	10	10	14	44	118

〈著者紹介〉

한 동 국 (Dong-Guk Han) 정회원
1999년 2월 : 고려대학교 수학과 이학사
2002년 2월 : 고려대학교 수학과 이학석사
2002년 3월~현재 : 고려대학교 정보보호대학원 박사과정
〈관심분야〉 수론, 공개키 암호 ,CMVP, 부채널 공격

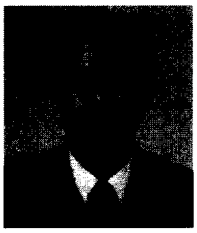


장 상 운 (Sang-Woon Jang)
2002년 2월 : 고려대학교 수학과 이학사
2002년 3월~현재 : 고려대학교 정보보호대학원 석사과정
〈관심분야〉 공개키 암호, 부채널 공경



윤 기 순 (Ki-Soon Yoon)
1998년 8월 : 경희대학교 수학과 이학사
1999년~2002년 : KSIGN 근무
2001년 3월~현재 : 고려대학교 정보보호대학원 석사과정
〈관심분야〉 공개키 암호, 암호 프로토콜

장 남 수 (Nam-Soo Jang)
2002년 2월 : 서울시립대학교 수학과 이학사
2002년 3월~현재 : 고려대학교 정보보호대학원 석사과정
〈관심분야〉 공개키 암호, 무선랜



박 영 호 (Young-Ho Park) 정회원
1990년 2월 : 고려대학교 수학과 학사
1993년 2월 : 고려대학교 수학과 석사
1997년 2월 : 고려대학교 수학과 박사
2001년~현재 : 고려대 정보보호기술연구센터 객원조교수
〈관심분야〉 정수론, 공개키 암호, 암호 프로토콜

김 창 한 (Chang-Han Kim) 정회원
1985년 2월 : 고려대학교 수학과 학사
1987년 2월 : 고려대학교 수학과 석사
1992년 2월 : 고려대학교 수학과 박사
2000년 2월~현재 : 세명대학교 컴퓨터수리정보학과 부교수
〈관심분야〉 정수론, 공개키 암호, 암호 프로토콜