공개키 기반의 암호 시스템에 적합한 모듈러 연산기 알고리즘의 효율적인 설계

김정태* · 허창우 · 류광열
*목원대학교

Modular Multiplication Algorithm Design for Application of Cryptosystem based on Public Key Structure

Jungl-Tae Kim* · Chang-Woo Hur · Kwang-Ryul Ryu
*Mokwon University
E-mail : jtkim3050@mokwon.ac.kr

요 약

정보보호 기술이 필요로 하는 분야는 위성통신, CATV, 인터넷, 전자문서 교환(EDI ; Electronic Data Exchange)을 포함한 전자상거래(electronic commerce), smart IC card, EFT(Electronic Funds Transfer) 등 거의 모든 정보통신 산업 관련 분야를 망라하고 있다. 특히 이러한 정보시스템의 경우 암호의 누출 및 hacking 문제는 사회 및 국가 안보분야에도 큰 영향을 미치게 된다. 따라서 본 논문에서는 최근에 무선통신 환경에 적합한 타원형 곡선 알고리즘의 유한체에서의 polynomial과 normal 기저에 대한 연산 결과를 분석하였다.

ABSTRACT

The computational cost of encryption is a barrier to wider application of a variety of data security protocols. Virtually all research on Elliptic Curve Cryptography(ECC) provides evidence to suggest that ECC can provide a family of encryption algorithms that implementation than do current widely used methods. This efficiency is obtained since ECC allows much shorter key lengths for equivalent levels of security. This paper suggests how improvements in execution of ECC algorithms can be obtained by changing the representation of the elements of the finite field of the ECC algorithm. Specifically, this research compares the time complexity of ECC computation over a variety of finite fields with elements expressed in the polynomial basis(PB) and normal basis(NB)

Key Words
Elliptic Curve, Cryptography, Security

I. Introduction

There are three families of public key algorithms that have considerable significance in current data security practice. They are integer factorization, discrete logarithm and elliptic curve based schemes. Integer factorization based schemes such as RSA and Discrete Logarithm based schemes such as Diffie Hellman provides intuitive ways of implementation. However, both methods admit sub-exponential algorithms of cryptography. In this regard, elliptic curve crytographic method is available. The best current brute force algorithms for cryptoanalysis of ECC require O(n/2) steps where n is the order of the additive group. For example, using the best current brute force algorithms ECC with a key size of 173 bits provides the same level of cryptographic security as RSA with a key size of 1024 bits. This results in smaller system parameters, bandwidth savings, faster implementations and lower power consumption. In addition, elliptic curves over finite fields offer an inexhausible supply of finite abelian groups, thus allowing more flexible field selections than conventional discrete logarithm schemes. Because of these advantages, ECC has extracted extensive attention in recent years[1,2].

II. Elliptic Curves

Elliptic curves were first suggest in 1985 by Victor Miller and Neal Kolitz for implementing public key cryptosystems. The addition operation of this group is easy to implement. Moreover, the discrete logarithm problem in this group is believed to be very difficult, even much harder than the discrete logarithm problem in finite fields of the same size as K[1].

III. Elliptic Curve Cryptography

An ECC over GF(2^n) is defined to be the set of points (x,y) satisfying an equation of the form $y^2 + axy + by = x^3 + cx^2 + dx + e$, where a, b, c, d and e are real numbers satisfy some conditions which depends on the field it belongs to, such as real number or finite number field. There is a point O called the point at infinity or the zero point. The basic operation of elliptic curve is addition. To double for a point P, it is equivalent to do P+P. Similary, we can calculate 3P=2P+P and so on. One important property is that it is very difficult to find an integer n such that nP=Q.

In order to use elliptic curve to do cryptographic operation, some basic setup is needed.

- Find a curve $y^2=x^3+ax+b$ over finite field.
- Find a point $G=(x_0,y_0)$ such that Order (G) = p which is a larger prime number.
- The curve and point G is known to everyone and can be shared by multiple user.
- Private key is an integer.
- Public key is a point on the curve.

3.1 Arithmetic Operations Over GF(2^n)

This section describes the performance of field arithmetic operations over GF(2^n) with field size ranging from 100 to 1279 bits for PB and from 100 to 1019 bits for NB representations. The variation in field size results from using existing well defined finite fields. Theoretically, once the field size n is selected, the content of the input message should not affect the performance. A Message is a byte-string of a given size. Since we can generate a random input message easily, a different random input message was used for each run. Reported execution times are the average of 20 independent runs with 20 different input message of the same size. For the performance comparison between PB and NB representations. 5-10% difference in timings for the same type of operation is recognized as a significant difference in performance[3].

3.2 Addition and Subtraction

Addition and subtraction can be implemented efficiently by exclusive-OR of two field elements. The performance of these operations is illustrated in Figure 1. It can be seen that the time complexities of Addition/Subtraction are proportional to the field size and vary from 0.13 to 1.15 us over the field size range from 100 to 1279 bits for both NB and PB multiplication.

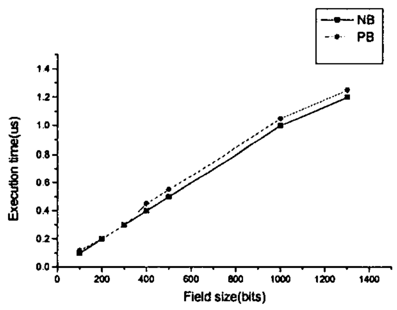

Fig. 1 Execution time for Addition and Subtraction

Polynomial basis: Multiplication in PB contains two step: partial multiplication and modular reduction. For partial multiplication, we employ the shift and add strategy: This algorithm runs in O(n*n/ws) time where n is the field size in bits, ws is machine word size. Figure 2 shows that the performance of multiplication in PB varies from 87 to 8960 us over field sizes ranging from 100 to 1279 bits.

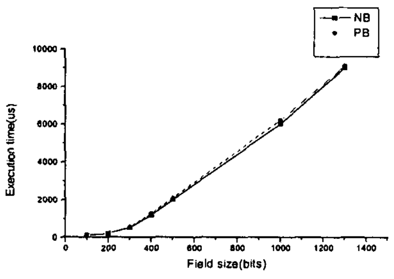

Fig. 2 Execution time for multiplication

Normal basis: The implementation computes multiplication through shift, XOR, and AND. First, shift one multiplier consecutively and store the results for later lookup. Since the lamda matrix can be precomputed, simply rotate the other multiplier, lookup the lamda matrix table twice, and performance XOR and AND. This algorithm takes O(n*n/ws) time where n is field size in bits and ws is the machine word size. The execution time for this implementation is shown in Figure 3.

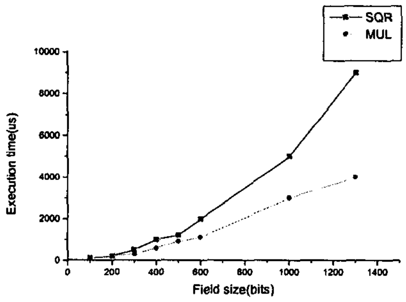

Fig. 3 Comparison of Execution time for Squaring with Multiplication in PB.

Figure 3. Execution times vary from 108 to 6066 us over field sizes ranging from 100 to 1019 bits. Multiplication in PB is about 17% faster than of NB. Both algorithms have similar time complexities. The PB implementation uses a trinomial to enhance the performance. The NB implementation uses table-lookup to simplify the computation.

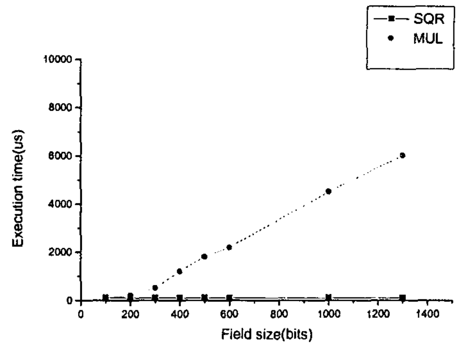

Fig. 4 Comparison of execution time for squaring with multiplication in NB

3.3 Squaring

Squaring is just a special case of multiplication. Both PB and NB implementations can simplified for the special case. This algorithm also runs in O(n). The NB implementation provides 40% performance improvement on squaring over general multiplication. Figure 4 shows that the time for squaring is negligible when compared to the time for general multiplication. This 100% performance improvement is equivalent to 83% improvement in squaring over multiplication in NB. This is more than twice the performance improvement that PB achieves.

IV. Elliptic Group Operations

Elliptic group operations include point negation, point addition, point subtraction, point doubling and scalar multiplication. Even though high-level implementations of these operations are the same for PB and NB, the performance of them may be differ since subroutines called in major functions may performance differently between PB and NB. Both the fast and slow algorithms of the underlying field operations were used as subroutines for these Group operations, Polynomial Basis.

Figure 5 shows the execution time for point addition and point subtraction are very close. This holds since point subtraction consists of point addition and point negation and the time for negation is negligible. Point doubling is 18.7% slower than point addition. This is probably because point doubling contains one extra field squaring operation when compared to point addition. Point addition consists of 1 squaring, 2 multiplications and 1 conversion, which are equivalent to 17% point addition. The Expected time for point doubling is basically consistent with this analysis. The 1.7% variation is probably attributed to the constant factors in implementation and random errors. Scalar multiplications are the most time consuming group operation. The times for this operation vary from 0.04 to 50s over field sizes from100 to 1279 bits. The times for this operation vary from 0.04 to 50s over field sizes from 100 to 1279 bits. This is more than two orders of magnitude slower than point addition.

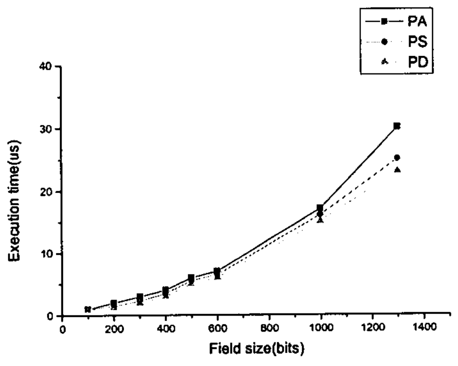

Figure 5 Execution time for Elliptic Group Operations in PB

The algorithms takes (f_s-1) PD and (fs/2-1)PA where f_s is the field size in bits. Since PD=118.7% PA based on these measurement.

The execution time for point negation is very similar to point negations in PB and negligible compared to timings of other group operations . This is why point addition and point subtraction have nearly identical executions times. Normal Basis point doubling appears to take about the same time as point addition. Theoretically, point doubling takes an extra squaring relative to point addition. However, in NB, the time for squaring is negligible. This results in the roughly equal timings between point doubling and point addition.

V. Conclusions

Finite field arithmetic operations include addition, subtraction, multiplication, squaring and inversion. These results show that both addition and subtraction can be implemented very efficiently and the differences between PB and NB are small. Multiplication in PB using PB and NB are small. Multiplication in PB using trinomial as the irreducible polynomial is 17% faster than multiplication in NB. Squaring, a special case of multiplication can be implemented 40% faster than multiplication in NB. Elliptic group operations include point negation, point addition, point subtraction, point doubling and scalar multiplication. These results show that point negation can be implemented very efficiently in both PB and NB and the time is small compared to other group operations. Point addition and subtraction runs in similar time in both PB and NB.

References

[1] G.B.Agnew, et all, "An implementation of Elliptic Curve Cryptosystems over GF(2n), IEEE J. on Selected Areas in Commu. v.11, no.5, pp.804-813, 1993

[2] T.ElGamal, "A Public key cryptosystem and a signature scheme based on discrete logarithms", IEEE Tran. on Information Theory, v.31. pp.473-481, 1985

[3] N.Koblitz, "Elliptic curve cryptosystems", Mathematics of Computation, v.48, no.177, pp.203-209, 1987