Unbalanced Exponent Modular Reduction over Binary Field and Its Implementation

Haibin Shen
Zhejiang University
Institute of VLSI Design
Hangzhou, China
shb@vlsi.zju.edu.cn

Yier Jin
Zhejiang University
Institute of VLSI Design
Hangzhou, China
jinye@vlsi.zju.edu.cn

Rongquan You
Zhejiang University
Institute of VLSI Design
Hangzhou, China
yourq@vlsi.zju.edu.cn

Abstract

Modular reduction is the basic operation of cryptographic systems. The Barrett's Algorithm and Montgomery's Algorithm are widely used nowadays and they are both based on pre-computation. In the field of Elliptic Curve Cryptosystem (ECC) over $GF(2^m)$, the reduction polynomials recommended by SEC have few items and the degree of second item is much less than that of the first one. Making use of this characteristic, the paper presents a new method to accelerate modular reduction without pre-computation which speeds up modular reduction by 10–30 times over $GF(2^m)$ and speeds up ECC point multiplication by 40%–50%. This algorithm has been implemented in a high-speed public-key cipher accelerator.

1. Introduction

Elliptic Curve Cryptosystem(ECC) offers equivalent security with smaller key sizes in less computation time and with less memory than other public-key cryptography, so it has been widely used in Digital Signature and Integrated Encryption fields[10]. The main computation of ECC is point multiplication[6,7], where modular inverse is considered to be the slowest computation. Montgomery's method in projective coordinates[8] is an efficient method in computing point multiplications by reducing the number of modular inverse into one time. A binary algorithm based on Extended Euclidean to compute modular inverse was proposed in [1], and optimized in [2]. This method, however, costs large area when it is implemented by hardware. Some others are based on Fermat's Little Theorem[3], which converts modular inverse into modular multiplication and modular square in order to simplify hardware implementation.

Because the finite field operations can be implemented very efficiently in hardware, Elliptic Curves over $GF(2^m)$ are particularly attractive. A typical modular inverse over $GF(2^m)$ named Almost Inverse is provided in [4]. All these methods replace the modular inverse by modular multiplications, modular square and modular addition.

Modular reduction is the foundation of modular multiplication and some fast modular reduction algorithms have been proposed [5,9]. The well known Barrett reduction method and Montgomery reduction method leverage some pre-computations to avoid divisions. This paper presents a new reduction algorithm named Unbalanced Exponent Modular Reduction in $GF(2^m)$, and we embed this algorithm in public-key cipher accelerator.

The rest of paper is organized as follows: Section 2 introduces details of our algorithm as well as performance comparison with other reduction methods. Expanding this algorithm to modular multiplication is described in Section 3. Section 4 presents the implementation of this method in public-key cipher accelerator. Finally the conclusions are drawn in Section 5.

2. Algorithms

According to Standards for Efficient Cryptography(SEC), the reduction polynomials of ECC cryptography system in $GF(2^m)$ is expressed as $f(x) = x^m + T(x)$ with degree m, where the degree of $T(x)$ is far much smaller than m and with 2 or 4 items[10]. A polynomial $C(x)$ whose degree n fulfill the inequality of $2m > n > m$, can be divided into two parts as the following form:

$$C(x) = C_h(x)x^m + C_l(x) \tag{1}$$

Because modular subtraction is the same as modular addition over $GF(2^m)$, we can obtain:

$$C(x) \equiv C_h(x)T(x) + C_l(x) mod\ f(x) \tag{2}$$

Repeat (2), until C_h is zero. This algorithm can be described in Algorithm 1.

Proceedings of the First International Conference on Innovative Computing, Information and Control (ICICIC'06)
0-7695-2616-0/06 $20.00 © 2006 IEEE

Algorithm 1	Unbalanced Exponent Modular Reduction
Input	$C(x), f(x)$
Output	$C(x) mod\ f(x)$
	$Do\ \{$ $\quad Partition\ C(x)\ into\ C_h\ and\ C_l\ with$ $\quad Degree \geq\ m;$ $\quad C(x) = C_h T(x) + C_l\ \}$ $While\ (C_h \neq 0)$ $Return\ C(x)$

Details of unbalanced exponent modular reduction are showed here:

Assuming $C(x) = x^n + a_{n-1}x^{n-1} + \ldots + a_m x^m + a_{m-1}x^{m-1} + \ldots, T(x) = x^k + \ldots + 1, f(x) = x^m + T(x)$

Then $C(x) \equiv C_h(x)T(x) + C_l(x) mod\ f(x)$

$\equiv (x^{n-m} + a_{n-1}x^{n-m-1} + \ldots)(x^k + \ldots + 1) + (a_{m-1}x^{m-1} + \ldots) mod\ f(x)$

Here we will find that the degree of $C(x)$ changes from n to $n - m + k$, which means that reduced degree of $C(x)$ is $\Delta p = m - k$ per-iteration. Assume after L iterations $C_h = 0$, L should fulfill the inequalities:

$$n - L\Delta p \leq m - 1 \tag{3}$$

$$n - (L-1)\Delta p \geq m - 1 \tag{4}$$

According to (3) and (4), we acquire the integer L's range:

$$(n - m + 1)/(m - k) \leq L \leq (n - k)/(m - k) \tag{5}$$

It is easy to prove the scope $[(n - m + 1)/(m - k), (n - k)/(m - k)]$ only contain one integer—the right result we need.

As $k \ll m$(in most cases), there are only 2 iterations needed. When $f(x) = x^{239} + x^{158} + 1$ where the gap between m and k is the smallest among all these $f(x)$ recommended by SEC, only 3 iterations is needed. Furthermore, all these computations are in $GF(2^m)$, so they are easy to be implemented by hardware and can obtain high computing speed.

We extend the two typical modular reduction algorithms—Barrett Modular Reduction(BMR)[5] and Montgomery Modular Reduction(MMR)[9], to binary field. These two algorithms over $GF(2^m)$ are showed below.

Montgomery Modular Reduction	
Input	$C(x) = x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$ $f(x) = x^m + b_{m-1}x^{m-1} + \ldots + b_1 x + b_0$ $b'_0 = -b_0^{-1} mod\ x$
Output	$C(x)x^{-m} mod\ f(x)$
	$For\ i = 0\ to\ m-1\{$ $\quad p_i = a_i b'_0 mod\ x;$ $\quad C(x) = C(x) \oplus p_i f(x) x^i; \}$ $C(x) = C(x)/x^m$ $Return\ C(x)$

Table 1. Time cost ratios between Barrett Modular Reduction(BMR), Montgomery Modular Reduction(MMR) and Unbalanced Exponent Modular Reduction(UEMR)

Irreducible polynomials $f(x)$ SEC recommended	Time cost ratio BMR:UEMR	Time cost ratio MMR:UEMR
$x^{163} + x^7 + x^6 + x^3 + 1$	31.6	16.3
$x^{233} + x^{74} + 1$	30.4	17.5
$x^{233} + x^{36} + 1$	31.8	19.8
$x^{239} + x^{158} + 1$	17.2	10.7
$x^{283} + x^{12} + x^7 + x^5 + 1$	49.5	28.4
$x^{409} + x^{87} + 1$	58.4	31.5
$x^{571} + x^{10} + x^5 + x^2 + 1$	106.4	57.1

Barrett Modular Reduction	
Input	$C(x) = x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$ $f(x) = x^m + b_{m-1}x^{m-1} + \ldots + b_1 x + b_0$ $\mu = x^{2m}/f(x)$
Output	$C(x) mod\ f(x)$
	$q = [(C(x)/x^{m-1}) * \mu]/x^{m+1};$ $C(x) = C(x) mod\ x^{m+1} \oplus [q * f(x)] mod\ x^{m+1};$ $Return\ C(x)$

We exploit C program to simulate these three algorithms for all reduction polynomials in SEC[10] on Pentium IV processor running at 3.0GHz with Linux Operation System. The ratios of time cost by these three algorithms are listed in Table 1.

In Table 1, when computing in $GF(2^m)$, the operating speed of unbalanced exponent modular reduction is 20–50 times faster than that of Montgomery reductions in most cases. Compared to Barrett reduction, unbalanced exponent modular reduction is 30–100 times faster in most cases.

3. Extension

Modular multiplication, modular square and modular addition are common computations in ECC algorithm. We extend our unbalanced exponent modular reduction to modular multiplication described in Algorithm 2.

Algorithm 2 Unbalanced Exponent Modular Multiplication	
Input	$A(x), B(x), f(x)$
Output	$A(x)B(x) mod\ f(x)$
	$C(x) = A(x)B(x)$; Do { $Partition\ C(x)\ into\ C_h\ and\ C_l\ with$ $Degree \geq\ m$; $C(x) = C_h T(x) + C_l$ } $While\ (C_h \neq 0)$ $Return\ C(x)$

We can get modular square in the same way. Making use of optimized modular multiplication based on unbalanced exponent modular reduction, the performance is improved significantly for ECC computation.

The other two modular multiplications widely used nowadays are Montgomery Algorithm [9,12] and Barrett Algorithm [5,11]. These two algorithms can be implemented according to pseudocode over $GF(2^m)$ as below.

Montgomery Modular Multiplication	
Input	$A(x), B(x), f(x)$
Output	$A(x)B(x)x^{-m} mod\ f(x)$
	$C(x) = 0$ $For\ i = 0\ to\ m-1$ { $C(x) = C(x) \oplus a_i B(x)$; $C(x) = C(x) \oplus c_0 f(x)$; $C(x) = C(x)/x$; } $Return\ C(x)$

Barrett Modular Multiplication	
Input	$A(x), B(x), f(x)$ $\mu = x^{2m}/f(x)$
Output	$A(x)B(x) mod\ f(x)$
	$C(x) = A(x)B(x)$ $q = [(C(x)/x^{m-1} * \mu]/x^{m+1}$; $C(x) = C(x) mod\ x^{m+1} \oplus [q * f(x)] mod\ x^{m+1}$; $Return\ C(x)$

We use Montgomery's method in projective coordinates[8] to finish point multiplication. Based on these three modular multiplications, with all reduction polynomials recommended in SEC[10], performances of point multiplication are listed in Table 2 in the form of time cost ratio.

From table 2, we find that the point multiplication using our Unbalanced Exponent Modular Multiplication is about 1.45 times faster than using Montgomery's Algorithm, and 1.9 times faster than using Barrett's Algorithm over $GF(2^m)$ in ECC.

4. Implementation

Unbalanced exponent modular reduction has been embedded in our high-speed public-key cipher accelerator as

Table 2. Time cost ratios of computing point multiplications over $GF(2^m)$ in ECC using Barrett Modular Multiplication(BMM), Montgomery Modular Multiplication(MMM) and Unbalanced Exponent Modular Multiplication(UEMM)

Irreducible polynomials $f(x)$ SEC recommended	Time cost ratio BMM:UEMM	Time cost ratio MMM:UEMM
$x^{163} + x^7 + x^6 + x^3 + 1$	1.92	1.45
$x^{233} + x^{74} + 1$	1.92	1.41
$x^{233} + x^{36} + 1$	1.77	1.47
$x^{239} + x^{158} + 1$	1.73	1.44
$x^{283} + x^{12} + x^7 + x^5 + 1$	1.86	1.47
$x^{409} + x^{87} + 1$	1.90	1.48
$x^{571} + x^{10} + x^5 + x^2 + 1$	1.92	1.49

fundamental algorithm of ECC point multiplication over $GF(2^m)$. Figure 1 shows the architecture of this accelerator.

The public-key cipher accelerator can be divided into three main parts:

(1)Sequential Code Generator (SCG).

This is mainly a Finite State Machine (FSM) embedded with public-key accelerating algorithm. According to commands from main controller outside (CPU), SCG starts certain algorithm state machine and generates code for Crypto-Processor. The FSM controls algorithms' processes, and the code it outputs are totally sequential in order to im-

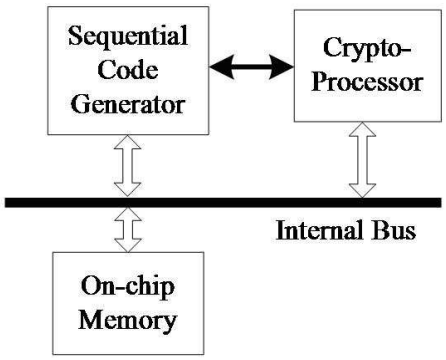

Figure 1. Architecture of high-speed public-key cipher accelerator

prove the efficiency of Crypto-Processor. ASM has embedded all nonsingular fields of ECC accelerating algorithms in $GF(2^m)$ where m=113–571. In order to enhance the flexibility of this accelerator, SCG can be bypassed, i.e., the main controller can also send code to the Crypto-Processor directly.

(2) Crypto-Processor

This is an embedded processor supporting code over both $GF(2^m)$ and $GF(p)$. Also it contains 64-bit width calculating unit.

(3) On-chip Memory

Storage for parameters in public-key cipher algorithm including reduction polynomials,parameters of Elliptic Curve, computation results, and so on.

We use Verilog hardware description language [16] to finish RTL level design of public-key cipher accelerator and tape out under SMIC $0.18\mu m$ standard cell library. Take the advantage of the algorithm presented in this paper, the speed of point multiplication over $GF(2^m)$ is about 2700 times per second.

5. Conclusion

This paper presented a fast modular reduction over $GF(2^m)$ without pre-computation. This algorithm speeds up computations of ECC over $GF(2^m)$, and has been implemented in pubic-key accelerator successfully. Further work will concentrate on improving performance in other aspects of ECC with the method this paper proposed.

References

[1] D.E. Knuth, *The Art of Computer Programming — Seminumerical Algorithm*, vol. 2, Addison-Wesley, 3rd ed., 1998.

[2] L.A. Tawalbeh, A.F. Tenca, S. Park, and Ç.K. Koç, "A dual-field modular division algorithm and architecture for application specific hardware," *Signals, Systems and Computers*, vol. 1, pp. 483-487, 2004.

[3] C.C. Wang, T.K. Truong, H.M. Shao, L.J. Deutch, and J.K. Omura, "VLSI architecture for computing multiplications and inverses in $GF(2^m)$," *IEEE Transactions on Computers*, vol. C-34, no. 8, pp. 709-717, 1985.

[4] R. Schroeppel, H. Orman, S. O'Malley, and O. Spatscheck, "Fast Key Exchange with Elliptic Curve Systems," *Advances in Cryptology-CRYPTO 95*, pp. 43-56, 1995.

[5] P. Barrett, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on astandard digital signal processor," *Advances in Cryptology-CRYPTO 86*, pp. 311-323, 1987.

[6] E. Savas, and Ç.K. Koç, "The Montgomery modular inverse-revisited," *IEEE Transactions on Computers*, Vol. 49, Issue 7, pp.763-766, 2000.

[7] T. Kobayashi, and H. Morita, "Fast Modular Inversion Algorithm to Match Any Operation Unit," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E82-A, no. 5, pp. 733-740, 1999.

[8] J. López, and R. Dahab, "Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation," *Cryptographic Hardware and Embedded Systems - CHES '99*, LNCS 1717, pp. 316-327, 1999.

[9] P.L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no.170, pp.519-521, 1985.

[10] *Standards for Efficient Cryptography 1: Elliptic Curve Cryptography*, Version 1.5, *draft*, 2005. http://www.secg.org/

[11] D.S. Phatak, and T. Goff, "Fast modular reduction for large wordlengths via one linear and one cyclic convolution," *Computer Arithmetic*, pp. 179-186, 2005.

[12] Ç.K. Koç, and T. Acar, "Montgomery Multiplication in $GF(2^k)$," *Design, Codes and Cryptography*, vol. 14, no. 1, pp. 57-69, 1998.

[13] V. Müller, "Fast Multiplication on Elliptic Curves over Small Fields of Characteristic Two," *Journal of Cryptology*, pp. 219-234, 11, 1998.

[14] A.E. Cohen, and K.K. Parhi, "Implementation of scalable elliptic curve cryptosystem crypto-accelerators for $GF(2^m)$," *Signals, Systems and Computers*, vol. 1, pp. 471-477, 2004.

[15] A. Satoh, and K. Takano, "A scalable dual-field elliptic curve cryptographic processor," *IEEE Transactions on Computers*, vol. 52, pp. 449-460, 2003.

[16] IEEE Standard Verilog Hardware Description Language, *IEEE Computer Society*, IEEE Std 1364-2001, 2001. http://www.ieee.org/.