Revisiting Scalable Modular Multiplication over GF(2^m) for Elliptic Curve Cryptography

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

Abstract

A new unbalanced exponent modular reduction over GF(2^m) is proposed. The algorithm can achieve high efficiency when computing on a certain class of fields generated by $f(x) = x^m + T(x)$ where deg$[T(x)] \ll m$. The algorithm is applied in modular multiplication on the basis of scalable polynomial basis(SPB) to form scalable modular multiplication. Most of irreducible polynomials used in Elliptic Curve Cryptography(ECC) fulfill the characteristic mentioned above well. So the scalable algorithm is implemented in ECC computation with high flexibility and efficiency both in theoretic calculation and application.

1. Introduction

Elliptic Curve Cryptography(ECC) was first proposed by Koblitz[12] and Miller[15], and has been widely used in Elliptic Curve Digital Signature Algorithm (ECDSA), Elliptic Curve Integrated Encryption Scheme (ECIES), elliptic curve Diffie-Hellman scheme[4,5]. Elliptic curves over GF(2^m) are particularly attractive, because the finite field operations can be implemented efficiently both in hardware and software[1,2,14]. Thus in this paper, we concentrate on ECC over GF(2^m). In applications, we may need different security level with different key length and scalable method is a useful one to fulfill this demand. To support scalable computation in hardware and software, scalable algorithms are required first. In ECC, the complexity of computation bases on point multiplication when parameters of elliptic curves are defined. A fast Montgomery point multiplication in projective field was suggested by Lopez[7], which divides point multiplication into modular addition, modular multiplication and modular inverse, and reduces modular inverse into one time. With this method, most of computations of ECC are modular multiplications and modular additions.

Many fast modular multiplications in binary field have been proposed[9,13] and Montgomery method is one of the most widely used[2,3]. However, most of these methods suit for general condition and do not consider the special characteristics of elliptic curves over GF(2^m) [4,5,6] whose irreducible polynomials are in the form of $f(x) = x^m + x^k + 1$ wherein $k \ll m$ or $f(x) = x^m + x^a + x^b + x^c + 1$ wherein $a \ll m$. Making use of this characteristic, we propose an improved scalable modular multiplication over GF(2^m) which can achieve higher computation speed than other scalable algorithms.

The rest of this paper is organized as follows: Section 2 introduces details of scalable polynomial representation(SPB). Based on SPB, a new modular multiplication in GF(2^m) are proposed in Section 3 . Section 4 presents the comparison between scalable modular multiplication proposed here and the widely used Montgomery algorithm. Finally conclusions are drawn in Section 5.

2. Scalable Polynomial Representation

According to [5], there are two common families of basis representations: polynomial basis representations and normal basis representations. In polynomial basis representation, each element of GF (2^m) is represented by a different binary polynomial of degree less than m. More explicitly, the bit string $(a_{m-1}, a_{m-2}, \ldots, a_1, a_0)$ is taken to represent the binary polynomial

$$a(x) = \sum_{i=0}^{m-1} a_i x^i = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0 \qquad (1)$$

here polynomial basis is the set $\{x^{m-1}, x^{m-2}, \ldots, x, 1\}$.

The addition of bit strings, corresponds to addition of binary polynomials. Multiplication is defined in terms of an irreducible binary polynomial $f(x)$ of degree m, called the *field polynomial* for the representation. The product of two elements is simply the product of the corresponding polynomials, reduced modulo $f(x)$.

In reality, with the purpose to fulfill different security level, the key length should be scalable. In high security implementation, a long key is used while a short key is employed in low security occasion so we can

1-4244-0161-5/06/$20.00 ©2006 IEEE

achieve higher encryption speed. In this method, to represent an m-bits long number with r-bits long word, word counts we need are $s = \lceil m/r \rceil$, i.e., we can write $a(x)$ as an sr-bit number consisting of s words, where each word is of length r. Thus, a m-bit number a can be shown as $a = (A_{s-1}, A_{s-2}, \ldots, A_1, A_0)$, where $A_i = (a_{ir+r-1}, a_{ir+r-2}, \ldots, a_{ir+1}, a_{ir})$. So the scalable polynomial basis(SPB) representation is:

$$\begin{aligned} a(x) &= \sum_{i=0}^{s-1} A_i(x)x^{ir} \\ &= A_{s-1}(x)x^{(s-1)r} + \cdots + A_1(x)x^r + A_0(x) \end{aligned} \tag{2}$$

where $A_i(x)$'s polynomial basis form is:

$$A_i(x) = \sum_{j=0}^{r-1} a_{ir+j}x^j = a_{ir+r-1}x^{r-1} + \cdots + a_{ir+1}x + a_{ir}.$$

3. Modular Multiplication

Montgomery algorithm is a widely used modular multiplication algorithm nowadays and it does not compute $a \cdot b \, mod \, f(x)$ straightforward. Instead, it leverages precomputation and compute $a \cdot b \cdot r^{-1} \, mod \, f(x)$[3]. This algorithm can achieve high efficiency when computing modular multiplication for general situations.

In the fields used by ECC over GF(2^m), however, polynomials are chosen with their special characteristics of a small number of terms[4, 5]. These irreducible polynomials modulo 2 that are minimal, in the sense that they have as few terms as possible and that those terms are of the smallest possible degree. More precisely, if an irreducible binary trinomial $x^m + x^k + 1$ exists, then the minimal possible value of k is listed; if no such trinomial exists, then a pentanomial $x^m + x^a + x^b + x^c + 1$ is listed. In the latter case, the value of a is minimal; the value of b is minimal for the given a; and c is minimal for the given a and b. Here, k and a, b, c fulfill inequality $k \ll m$ and $c < b < a \ll m$. So we partition $f(x)$ into x^m and $T(x)$, then $f(x) = x^m + T(x)$ where $T(x) = x^k + 1$ or $T(x) = x^a + x^b + x^c + 1$. Making use of this characteristic and reference Montgomery's method[3], we propose a new two stages modular multiplication without pre-computation.

Algorithm 1	Scalable Multiplication over $GF(2^m)$
Input	$A(x) = (A_{s-1}, A_{s-2}, \ldots, A_1, A_0)$ $B(x) = (B_{s-1}, B_{s-2}, \ldots, B_1, B_0)$
Output	$R(x) = A(x)B(x) = (R_{2s-2}, R_{2s-3}, \ldots, R_1, R_0)$
	for $i = 0$ *to* $s-1$\{ $z = 0$; *for* $j = 0$ *to* $s-1$\{; $S = R_{i+j} + A_i B_j + z$; $R_{i+j} = S$ *mod* x^r; $z = S/x^r$; \} $R_{i+s} = z$; \}

Stage 1: to compute $a \cdot b$. As the similar scalable multiplication in [8], we present scalable multiplication over GF(2^m) as Algorithm 1. In Algorithm 1, S is a temporary variable of $2r$-bit length and z means the high r bits of S.

Stage 2: to compute modular reduction. On bit-level consideration, assuming input $R(x)$ of $deg[R(x)] = 2m - 2$. Then $R(x)$ can be divided into two parts with high degree part $R_h(x)$ and low degree part $R_l(x)$. Note that $\stackrel{\triangle}{=}$ means dividing the left operator into higher and lower part.

$$R(x) \stackrel{\triangle}{=} R_h(x)x^m + R_l(x) \tag{3}$$

where

$$R_h(x) = r_{2m-2}x^{m-2} + r_{2m-3}x^{m-3} + \cdots + r_{m+1}x + r_m \tag{4}$$

$$R_l(x) = r_{m-1}x^{m-1} + r_{m-2}x^{m-2} + \cdots + r_1 x + r_0 \tag{5}$$

Calculate modular on $R(x)$ with $f(x)$, because modular subtraction is the same as modular addition over GF(2^m), we can obtain:

$$\begin{aligned} R(x) &\equiv R_h(x)x^m + R_l(x) mod \, f(x) \\ &\equiv R_h(x)(x^m + T(x)) + R_h(x)T(x) + R_l(x) mod \, f(x) \\ &\equiv R_h(x)T(x) + R_l(x) mod \, f(x) \end{aligned} \tag{6}$$

Repeat (6) until R_h is zero. Thus, we have $deg[R(x)] \leq m - 1$.

This new bit-level modular reduction algorithm is shown as Algorithm 2.

Algorithm 2	New Bit-level Modular Reduction over $GF(2^m)$
Input	$R(x) = (r_{2m-2}, r_{2m-3}, \ldots, r_1, r_0)$ $f(x) = x^m + T(x)$ where *deg*$[T(x)] = k$
Output	$C(x) = R(x) mod \, f(x)$
	$R(x) \stackrel{\triangle}{=} R_h(x) \cdot x^m + R_l(x)$; *while* $(R_h(x) \neq 0)$\{ $R(x) = R_h(x) \cdot T(x) + R_l(x)$; $R(x) \stackrel{\triangle}{=} R_h(x) \cdot x^m + R_l(x)$; \} $C(x) = R_l(x)$; *return* $C(x)$;

Now we will give the expression of iteration number L in Algorithm 2. After one iteration, the most significant bit of $R(x)$ changes from $(2m-2)$ to $(m-2+k)$, it means that reduced degree of highest item is $\Delta p = m - k$ per-iteration. Obviously after L times iterations $R_h = 0$, so L should fulfill the inequalities:

$$(2m-2) - L\Delta p \leq m - 1 \tag{7}$$

$$(2m-2) - (L-1)\Delta p \geq m - 1 \tag{8}$$

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

$$(m-1)/(m-k) \leq L \leq (2m-2-k)/(m-k) \tag{9}$$

It is easy to prove the scope $[(m-1)/(m-k), (2m-2-k)/(m-k)]$ only contains one integer - the right result we need. Therefore, we have

$$L = \lfloor \frac{m-2}{m-k} \rfloor + 1 \tag{10}$$

We convert PB basis into SPB basis in Algorithm 2 to get the scalable modular reduction as showed in stage 2 of Algorithm 3. Note that $s = \lceil m/r \rceil$, $t = \lceil k/r \rceil$ as mentioned in Section 2.

Algorithm 3 Scalable Modular Multiplication over $GF(2^m)$

Input
$A(x) = (A_{s-1}, A_{s-2}, \ldots, A_1, A_0)$
$B(x) = (B_{s-1}, B_{s-2}, \ldots, B_1, B_0)$
$f(x) = x^m + T(x)$
where $T(x) = (T_{t-1}, T_{t-2}, \ldots, T_1, T_0)$

Output $C(x) = A(x)B(x) mod\ f(x)$

Stage 1:
$for\ i = 0\ to\ s-1\{$
$z = 0;$
$for\ j = 0\ to\ s-1\{;$
$S = R_{i+j} + A_i B_j + z;$
$R_{i+j} = S\ mod x^r;$
$z = S/x^r;$
$\}$
$R_{i+s} = z;$
$\}$

Stage 2:
$R(x) \triangleq R_h(x) \cdot x^{s \cdot r} + R_l(x)$
$while(R_h(x) \neq 0)\{$
$for\ i = 0\ to\ t-1\{$
$z = 0;$
$for\ j = 0\ to\ s-1\{;$
$S = R_{l_{i+j}}(x) + T_i(x) R_{h_j}(x) + z;$
$R_{l_{i+j}}(x) = S\ mod x^r;$
$z = S/x^r;$
$\}$
$R_{i+s} = z;$
$\}$
$R(x) = R_l(x);$
$R(x) \triangleq R_h(x) \cdot x^{s \cdot r} + R_l(x);$
$\}$
$C(x) = R_l(x);$
$return\ C(x);$

Algorithm 3 gives the full modular multiplication based on scalable polynomial basis.

4. Analysis of Scalable Algorithm

In this section, we analyze our scalable modular multiplication rigorously and compare this algorithm with Montgomery algorithm[3] of its improvement version in GF(2^m)[2, 10].

4.1. Complexity of Our Scalable Algorithm

In order to fulfill scalability, we assume the length of operation unit is a fixed number of r-bit—the same situation when we implement these algorithms in hardware. We define word multiplication r-bit$\times r$-bit as MULr, word addition operation r-bit$\oplus r$-bit as XORr in GF(2^m). And computations of complexity base on these two basic operations.

Table 1. Complexity of our scalable modular multiplication

	MULr	XORr
$for\ i = 0\ to\ s-1\{$	-	-
$z(x) = 0;$	-	-
$for\ j = 0\ to\ s-1\{;$	-	-
$S = R_{i+j} + A_i B_j + z;$	s^2	$2s^2$
$R_{i+j} = S\ mod x^r;$	-	-
$z = S/x^r;$	-	-
$\}$	-	-
$R_{i+s} = z;$	-	-
$\}$	-	-
$R(x) \triangleq R_h(x) \cdot x^{s \cdot r} + R_l(x)$	-	-
$while(R_h(x) \neq 0)\{$	-	-
$for\ i = 0\ to\ t-1\{$	-	-
$z = 0;$	-	-
$for\ j = 0\ to\ s-1\{;$	-	-
$S = R_{l_{i+j}}(x) + T_i(x) R_{h_j}(x) + z;$	$s \cdot L \cdot t$	$2 \cdot s \cdot L \cdot t$
$R_{l_{i+j}}(x) = S\ mod x^r;$	-	-
$z = S/x^r;$	-	-
$\}$	-	-
$R_{i+s} = z;$	-	-
$\}$	-	-
$R(x) = R_l(x);$	-	-
$R(x) \triangleq R_h(x) \cdot x^{s \cdot r} + R_l(x);$	-	-
$\}$	-	-
$C(x) = R_l(x);$	-	-
$return\ C(x);$	-	-
Total:	$s^2 + s \cdot L \cdot t$	$2 \cdot s^2 + 2 \cdot s \cdot L \cdot t$

Table 2. Complexity of scalable Montgomery algorithm

	MULr	XORr
$C(x) = 0;$	-	-
$for\ i = 0\ to\ s-1\{$	-	-
$z(x) = 0;$	-	-
$t_i(x) = [C_0(x) + A_i(x) B_0(x)] q(x) mod\ x^r;$	$2 \cdot s$	s
$for\ j = 0\ to\ s-1\{$	-	-
$S(x) = C_j(x) + A_i(x) B_j(x) + t_i(x) P_j(x) + z(x);$	$2 \cdot s^2$	$3 \cdot s^2$
$if(j \neq 0)\ then\ C_{j-1}(x) = S(x) mod\ x^r$	-	-
$z(x) = S(x)/x^r;\}$	-	-
$C_{s-1} = z(x);\}$	-	-
Total:	$2 \cdot s^2 + 2 \cdot s$	$3 \cdot s^2 + s$

We calculate operation unit counts of each step for our scalable modular multiplication, and list them in Table 1. Note that in Table 1, L means the iteration number defined in (10), s means word counts of polynomial whose degree is m and t means word counts of $T(x)$ whose degree is k, i.e., $s = \lceil m/r \rceil$ and $t = \lceil k/r \rceil$.

4.2. Complexity of Scalable Montgomery Algorithm

The details of scalable Montgomery modular multiplication is showed in [10]. As the same manner of Section 4.1, operation unit counts of each step of scalable Montgomery algorithm are listed in Table 2.

4.3. Comparisons of Complexity

The comparison is of two levels: 1. from the complexity in terms of word multiplication, word addition opera-

Table 3. Comparing our algorithm with Montgomery algorithm in the fields generated by $f(x)$ SEC1[4] recommended of 32-bit word length

	Montgomery	Our Proposal	Speedup
$F_{2^{163}}$ $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$	$2s^2 + 2s$	$\frac{4}{3}s^2$	1.5
$F_{2^{233}}$ $f(x) = x^{233} + x^{74} + 1$	$2s^2 + 2s$	$\frac{7}{4}s^2$	1.14
$F_{2^{233}}$ $f(x) = x^{233} + x^{36} + 1$	$2s^2 + 2s$	$\frac{3}{2}s^2$	1.33
$F_{2^{239}}$ $f(x) = x^{239} + x^{158} + 1$	$2s^2 + 2s$	$\frac{23}{8}s^2$	0.7
$F_{2^{283}}$ $f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$	$2s^2 + 2s$	$\frac{13}{9}s^2$	1.38
$F_{2^{409}}$ $f(x) = x^{409} + x^{87} + 1$	$2s^2 + 2s$	$\frac{19}{13}s^2$	1.46
$F_{2^{571}}$ $f(x) = x^{571} + x^{10} + x^5 + x^2 + 1$	$2s^2 + 2s$	$\frac{10}{9}s^2$	1.8

Table 4. Experimental speedup values with $f(x) = x^{233} + x^{74} + 1$

word length r=	8bits	16bits	32bits	64bits	128bits
Speedup	1.5727	1.7627	1.8605	2.1429	2.375

tion counts; 2. implementing these two algorithm in C programs, and obtain timings on a 3GHz Intel Pentium 4 processor running the Linux operation system.

In the first level comparison, we select all polynomials SEC1[4] recommended to compute word multiplication counts in Table 3. Note that we ignore counts of XOR adders as the complexity of adder is neglectable compared to multiplier. And we consider the occasion of word length r=32bits.

From Table 3, we will find that in most of the cases the algorithm presented here is about 1.5 times faster than Montgomery scalable modular multiplication.

In the second level comparison, taking polynomial $f(x) = x^{233} + x^{74} + 1$ for example, the timings are listed in Table 4 when we change word length r. With the increase of word length, the speedup of the proposed algorithm is increasing significantly. This merit can be rather useful for nowadays ASIC and MCU use, because the data path is becoming wider as well as the width of the operation units.

5. Conclusion

In this paper, a scalable modular multiplication is proposed to improve encryption efficiency for different security situations. This algorithm leverages an efficient modular reduction and can achieve higher enciphering speed compared to other methods in elliptic curve fields over GF(2^m). Furthermore, our method can be implemented in hardware easily for its finite field computation. By using a high-efficient r-bit $\times$ r-bit multiplier and r-bit $\oplus$ r-bit adder, high speed and flexibility for operator size can be achieved. Further work will be concentrated on the design of efficient word multiplier and word adder.

References

[1] G. B. Agnew, R. C. Mullin, and S. A. Vanstone, "An implementation of elliptic curve cryptosystems over F_2^{155}," *IEEE Journal on Selected Areas in Communications*, Vol. 11, Issue 5, pp. 804-813, 1993.

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

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

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

[5] *IEEE Standard Specifications for Public-Key Cryptography*, IEEE Std 1363-2000, 2000.

[6] *IEEE Standard Specifications for Public-Key Cryptography-Amendment 1: Additional Techniques*, IEEE Std 1363a-2004 (Amendment to IEEE Std 1363-2000), 2004.

[7] 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.

[8] Ç. K. Koç, T. Acar, and B. S. Jr. Kaliski, "Analyzing and comparing Montgomery multiplication algorithms," *Micro, IEEE*, Vol. 16, Issue 3, pp. 26-33, 1996.

[9] S. V. Bharathwaj, and K. L. Narasimhan, "An alternate approach to modular multiplication for finite fields GF(2^m) using Itoh Tsujii algorithm," *IEEE-NEWCAS Conference*, pp. 103-105, 2005.

[10] A. Satoh, and K. Takano, "A scalable dual-field elliptic curve cryptographic processor," *IEEE Transactions on Computers*, Vol. 52, Issue 4, pp. 449 - 460, 2003.

[11] H. Eberle, N. Gura, and S. Chang-Shantz, "A cryptographic processor for arbitrary elliptic curves over GF(2^m)," *Application-Specific Systems, Architectures, and Processors*, pp. 444 - 454, 2003.

[12] N. Koblitz, "Elliptic Curve Cryptosystems," *Mathematics of Computation*, 48, pp. 203-209, 1987.

[13] 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.

[14] E. De Win, S. Mister, B. Preneel, and M. Wiener, "On the performance of signature schemes based on elliptic curves," *Algorithmic Number Theory, Proceedings Third Intern. Symp., ANTS-III*, LNCS 1423, pp. 252-266, 1998.

[15] V. Miller, "Uses of elliptic curves in cryptography," *Advances in Cryptology: proceedings of Crypto'85, LNCS*, Vol. 218, pp. 417-426, 1986.

[16] A. F. Tenca, and Ç. K. Koç, "A Scalable Architecture for Modular Multiplication Based on Montgomery's Algorithm," *IEEE Transactions on Computers*, Vol. 52, No. 9, pp. 1215-1221, 2003.