Elliptic Curve Point Multiplication by Generalized Mersenne Numbers

Tao Wu and Li-Tian Liu

Abstract—**Montgomery modular multiplication in the residue number system (RNS) can be applied for elliptic curve cryptography. In this work, unified modular multipliers over generalized Mersenne numbers are proposed for RNS Montgomery modular multiplication, which enables efficient elliptic curve point multiplication (ECPM). Meanwhile, the elliptic curve arithmetic with ECPM is performed by mixed coordinates and adjusted for hardware implementation. In addition, the conversion between RNS and the binary number system is also discussed. Compared with the results in the literature, our hardware architecture for ECPM demonstrates high performance. A 256-bit ECPM in Xilinx XC2VP100 field programmable gate array device (FPGA) can be performed in 1.44 ms, costing 22147 slices, 45 dedicated multipliers, and 8.25K bits of random access memories (RAMs).**

Index Terms—**Elliptic curve cryptography, generalized Mersenne numbers, modular multiplier, residue number system.**

1. Introduction

Public key cryptography plays a significant role in information security that it provides convenient digital signatures and exchange of private keys[1]. Among kinds of public key cryptography schemes, RSA cryptography[2], Diffie-Hellman key exchange protocol[3], and elliptic curve cryptography (ECC)[4] are the most frequently used. Especially, ECC utilizes much shorter keys to reach the same security level as that of RSA cryptography. For example, the ECC with a 256-bit key provides the same security level as that of an RSA cryptosystem with a 3072-bit key[5].

In the above cryptosystems, the fundamental operation refers to multi-precision modular arithmetic. In RSA cryptography and the Diffie-Hellman key exchange protocol, modular exponentiation is the dominant operation in the whole computation. Within ECC the main computation is also modular multiplication in finite field. For instance, in the ECC digital signature protocol, there is 1 elliptic curve point multiplication (ECPM) to generate an ECC digital signature, and 2 ECPMs to authenticate a digital signature[5]. Meanwhile, 1 ECPM include about $10N$ to $30N$ modular multiplication, where N is the field bit length.

As long as ECPM over prime fields is concerned, several designs have been implemented by setting up a long-precision modular multiplier. With the large modular multiplier, all modular multiplication in ECC are then processed sequentially. In order to decrease the complexity in arithmetic and accelerate the computation, architectures with superior parallelism are effective and welcome. A large Montgomery modular multiplier has been developed with a quotient pipeline architecture[6], and has also been devised by a systolic array architecture[7].

Meanwhile, in a residue number system (RNS)[8],[9] multiprecision multiplication and addition are partitioned into a number of single-precision operations. Such architectures rely on the RNS Montgomery modular multiplication algorithm[10]–[17]. A Cox-Row architecture is built up[12] to perform modular exponentiation in RSA cryptography, and is applied for ECPM[13]. A very large scale integration circuit (VLSI) architecture for RNS Montgomery modular multiplication is also proposed[18], in which the conversion between the binary number system and RNS is emphasized and improved. Implementation over other platforms like microprocessors and graphic processing units (GPUs) have also been reported[19],[20].

In this paper, based on RNS Montgomery modular multiplication and generalized Mersenne numbers, an optimized ECPM architecture is proposed for hardware implementation. Firstly, a unified modular multiplier over generalized Mersenne numbers is devised, which performs modular multiplication with respect to two conjugated moduli. A group of such modular multipliers then cover two groups of RNS moduli, so it can be applied to RNS Montgomery modular multiplication. Besides, the binary-to-RNS and RNS-to-binary conversions can also be

Manuscript received June 25, 2012; revised July 27, 2012. This work was partially supported by the National Natural Science Foundation of China under Grant No. 61073173.

T. Wu and L.-T. Liu are with Department of Microelectronics and Nanoelectronics, Tsinghua University, Beijing (e-mail: t-wu03@whu.edu.cn; liulitian@tsinghua.edu.cn)

Digital Object Identifier: 10.3969/j.issn.1674-862X.2012.03.003

carried out by configuring the RNS Montgomery modular multiplier in different modes. Together with lots of precomputation for RNS Montgomery modular multiplication and mixed coordinates for ECPM, high-performance ECPM can be carried out by efficient modular arithmetic over generalized Mersenne numbers.

The remaining parts of this paper are organized as follows. Section 2 introduces the background for the RNS Montgomery modular multiplication algorithm and elliptic curve arithmetic with ECC. Then a unified modular multiplier is proposed for RNS Montgomery modular multiplication in Section 3, in which the residue arithmetic by generalized Mersenne numbers is developed. Section 4 discusses ECPM based on such RNS arithmetic, where the RNS-to-binary conversion and its inverse operation are also discussed. In Section 5, we give the results with hardware implementation of ECPM over prime field. The last Section concludes this paper.

2. Background for RNS and ECC

2.1 Residue Number System

A set of pairwise prime numbers define a residue number system, which is called RNS moduli. Suppose RNS Ω is defined by the moduli $\{m_1, m_2, \cdots, m_n\}$, then the dynamic range of Ω is $M=\prod m_i$ for $1\le i\le n$, where n is the number of moduli. If an integer lies between the range, i.e., $X\in[0, M-1]$, it can then be denoted in RNS by an n-tuple: $X=(x_1, x_2, \cdots, x_n)$, with $x_i = X \bmod m_i$ $(1\le i\le n)$. Set $Y=(y_1, y_2, \cdots, y_n)$, then addition and multiplication can be carried out as follows if $X+Y$ and XY are within the dynamic range $[0, M-1]$:

$$X+Y=\left(\left|x_1+y_1\right|_{m_1}, \left|x_2+y_2\right|_{m_2}, \cdots, \left|x_n+y_n\right|_{m_n}\right)$$

$$XY=\left(\left|x_1y_1\right|_{m_1}, \left|x_2y_2\right|_{m_2}, \cdots, \left|x_ny_n\right|_{m_n}\right).$$

It can be found out that the addition and multiplication in RNS are carry-free and enjoy high parallelism.

2.2 Improved Chinese Remainder Theorem

The Chinese remainder theorem is a mathematical connection between the binary expression and the RNS form of a number. Suppose $R=(r_1, r_2, \cdots, r_n)$ in Ω: $\{m_1, m_2, \cdots, m_n\}$, with $M=\prod m_i$, and $M_i=M/m_i$, for $i=1, 2, \cdots, n$. Then the Chinese remainder theorem declares that

$$R=\sum_{i=1}^{n} M_i\left|r_iM_i^{-1}\right|_{m_i} \bmod M. \tag{1}$$

The improved Chinese remainder theorem replaces the last modular reduction by subtraction[21], i.e.,

$$R=\sum_{i=1}^{n} M_i\left|r_iM_i^{-1}\right|_{m_i} -\alpha M \tag{2}$$

where α is an integer between $[0, n-1]$.

Denote $\sigma_i=\left|r_iM_i^{-1}\right|_{m_i}$, then

$$R=\sum_{i=1}^{n} M_i\sigma_i-\alpha M \tag{3}$$

which can be rewritten as[11]

$$\sum_{i=1}^{n}\left(\sigma_i/m_i\right)=\alpha+R/M. \tag{4}$$

Because $0\le R=M<1$, there is

$$\alpha\leqslant\sum_{i=1}^{n}\left(\sigma_i/m_i\right)<\alpha+1$$

i.e.,

$$\alpha=\left\lfloor\sum_{i=1}^{n}\left(\sigma_i/m_i\right)\right\rfloor. \tag{5}$$

As long as the integer α is estimated from (5), R can be obtained from (2) within the range $[0, 2M]$. Kawamura's method replaces σ_i/m_i by $\sigma_i/2^L$ and truncates certain digits[11],[12].

For example, let $\{m_i\}=\{2, 3, 5\}$, then $M=2\times3\times5=30$, $<M_i>=(15, 10, 6)$, $<M_i^{-1} \bmod m_i>=(1, 1, 1)$. Suppose $x=7=(1, 1, 2)$, then $<\sigma_i>=(1, 1, 2)$, $\alpha=\left\lfloor1/2+1/3+2/5\right\rfloor=1$, thus from (3) we have $x=1\times15+1\times10+2\times6-1\times30\equiv7$, which is just the original number.

2.3 Modular Arithmetic in RNS

It is well-known that RNS can be used to accelerate classic arithmetic by parallel operations, in which short-precision modular arithmetic is employed to achieve long-precision addition, subtraction, and multiplication. In fact, long-precision modular addition, modular subtraction, and modular multiplication can also be performed in RNS.

Assume A and B are two integers represented in RNS Ω, i.e., $A=(a_1, a_2, \cdots, a_n)$ and $B=(b_1, b_2, \cdots, b_n)$, while the moduli is $\{m_1, m_2, \cdots, m_n\}$ and the RNS dynamic range is M. Suppose we have to compute the modular addition $C\equiv A+B \pmod P$ in Ω, then a pure addition $C'=A+B$ is performed instead of C as long as $A+B<M$.

However, the RNS modular subtraction and RNS modular multiplication need more considerations, which are respectively discussed below and in Algorithm 1.

Usually, we perform finite arithmetic in the domain of positive integers, and in order to avoid overflow a correction step should follow the RNS modular subtraction, which just adds the modulus. For instance, when we compute $(A-B) \bmod P$ in RNS with $0\le A<P$ and $0\le B<P$, there are two possibilities about the result:

1) $(a_i-b_i) \bmod m_i$, for $1\le i\le n$;

2) $(a_i-b_i+p_i) \bmod m_i$, for $1\le i\le n$.

The first case refers to $A\ge B$, while the second case is with $A\le B$. Since we have to ensure that the result of a RNS subtraction is still positive[10], RNS tuple with the modulus should be added to the RNS result[10]. In general, if $A\le dP$, and $B\ge eP$, then in RNS arithmetic one should replace

$(a_i - b_i) \bmod m_i$ by $(a_i - b_i + ep_i) \bmod m_i$.

2.3 RNS Montgomery Modular Multiplication

Algorithm 1. RNS Montgomery modular multiplication

Input: RNS Ω is defined by moduli $\{m_1, m_2, \cdots, m_n\}$, $M=\prod m_i$, $M_i=M/m_i$, for i=1, 2, ···, n. RNS Γ is defined by moduli $\{m_{n+1}, m_{n+2}, \cdots, m_{2n}\}$, $N=\prod m_{n+i}$, $N_i=N/m_{n+i}$, for i=1, 2, ···, n. Meanwhile, $X=(x_1, x_2, \cdots, x_n, \cdots, x_{2n})$, $Y=(y_1, y_2, \cdots, y_n, \cdots, y_{2n})$, $P=(p_1, p_2, \cdots, p_n, \cdots, p_{2n})$. $X \cdot Y < P \cdot M$, $X<M$, $M<N$.

Precomputation: for i=1, 2, ···, n; j=n+1, n+2, ···, 2n:

$< p_j >=< P \bmod m_j >$, $< \tau_j >=< M^{-1} \bmod m_j >$,

$< \eta_i >=< M_i^{-1} \bmod m_i >$, $< \omega_j >=< N_j \bmod m_j >$,

$< \zeta_j >=< N_j^{-1} \bmod m_j >$, $< \mu_j >=< M \bmod m_j >$,

$< \lambda_i >=< N \bmod m_i >$, $< M_j^{(i)} >=< M_i \bmod m_j >$,

$< N_i^{(i)} >=< N_j \bmod m_i >$, $< \theta_i >=< \eta_i > \cdot < -p_i^{-1} >$,

$< \chi_j^{(i)} >=< M_j^{(i)} p_j \tau_j \zeta_j >$, $< \psi_j >=< \tau_j \zeta_j >$,

$< \phi_j >=< \mu_j p_j \tau_j \zeta_j >=< p_j \zeta_j >$,

where the tickets <·> denotes a set of modular operations.

Output: $R=< r_i >:=(r_1, r_2, \cdots, r_n)=< r_j >:=(r_{n+1}, r_{n+2}, \cdots, r_{2n})=$

$$X \cdot Y \cdot M^{-1} \bmod P, \text{ with } 0 \leqslant R < 3P.$$

1: $< z_i >=< x_i > \cdot < y_i >$;

2: $< z_j >=< x_j > \cdot < y_j >$;

3: $< \sigma_i >=< z_i > \cdot < \theta_i >$;

4: $< u_j >=< z_j > \cdot < \psi_j >$;

5: $\alpha = \left\lfloor \sum_{i=1}^{n} \frac{\text{trunc}(\sigma_i)}{2^L} \right\rfloor$;

6: $< \xi_j >:=< u_j >$;

7: for i=1 to n

8: $< \xi_j >:=< \xi_j > + \sigma_i \cdot < \chi_j^{(i)} >$;

9: end for

10: $< v_j >= \alpha \cdot < \phi_j >$;

11: $< \xi_j >:=< \xi_j > - < v_j >$;

12: $< r_i >:=< 0 >$; 13: $\beta = \left\lfloor \sum_{i=n+1}^{2n} \frac{\text{trunc}(\xi_i)}{2^L} + 0.5 \right\rfloor$;

14: for i=1 to n

15: $< r_i >:=< r_i > + < \xi_j > \cdot < N_i^{(j)} >$;

16: end for

17: $< r_i >:=< r_i > - \beta \cdot < \lambda_i >$;

18: $< r_j >=< \xi_j > \cdot < \omega_j >$;

19: return $\{< r_i >, < r_j >\}$.

In Algorithm 1, the main computation is two base conversions between RNS Ω and Γ. The colon before "=" emphasizes that it is an assignment to the original variable, trunc(x) sets the lowest (L-l) bits of "x" as zeros and "·" means modular multiplication. The value of l can be determined by RNS moduli[11].

According to Fermat's little theorem, if p is a prime number and GCD(a, p)=1, then $a^{-1} \equiv a^{p-2} \pmod{p}$. Therefore, RNS modular inverse over prime fields can be calculated by a series of RNS modular multiplication.

2.4 ECPM

An elliptic curve over the prime field F_p can be defined by the elliptic curve equation[5]:

$$y^2 = x^3 + ax + b \tag{6}$$

where $4a^3+27b^2 \neq 0 (\bmod p)$, and p is a prime number with p>3. In the elliptic curve field, point addition, point doubling, and point multiplication are basic operations.

In fact, an ECPM can be separated into a series of elliptic curve point addition and point doubling. Set $Q_0=(x_0, y_0)$ as a starting point on the elliptic curve, and choose k as a binary number with $k=(k_{l-1} \cdots k_1 k_0)_2$. As it is shown in Algorithm 2, the ECPM kQ_0 can be performed by the binary left-to-right method[5].

Algorithm 2. ECPM by point addition and point doubling

Input: Q_0 is a point in an elliptic curve over F_p, and $k=(k_{l-1} \cdots k_1 k_0)_2$ is a binary number.

Output: $S_{l-1}=kQ_0$.

1: $S_0 := Q_0$;

2: for i=1 to l–1

3: $S_i := 2S_{i-1}$;

4: if k_{l-1-i}=1 then

5: $S_i := S_i + Q_0$;

6: end if

7: end for

8: return S_{l-1}.

In the above algorithm, the addition "+" denotes point addition, while the multiplication of 2 denotes point doubling.

2.5 Elliptic Curve Point Doubling and Point Addition

Assuming Q_0 is the starting point with Jacobian coordinate (X_0, Y_0, Z_0), then its affine coordinate reads[5] $x_0=X_0/Z_0^2$ and $y_0=Y_0/Z_0^3$. Suppose Q_1 is the doubling point of Q_0, and its Jacobian coordinate is (X_1, Y_1, Z_1). Substituting x_i, y_i by X_i, Y_i, Z_i, there are

$$\begin{cases} X_1 = (3X_0^2 + aZ_0^4)^2 - 8X_0Y_0^2 \\ Y_1 = (3X_0^2 + aZ_0^4)(4X_0Y_0^2 - X_1) - 8Y_0^4 \\ Z_1 = 2Y_0Z_0. \end{cases} \tag{7}$$

In the above equations, the parameter a is just the

coefficient within the elliptic curve equation: $y^2 = x^3 + ax + b$.

For point addition of two points P_0 and P_1, mixed coordinates can be applied[5]. One point P_0 keeps the affine coordinate (x_0, y_0), and the other point is denoted by the Jacobian coordinate (X_1, Y_1, Z_1). Suppose $P_2=P_0+P_1$, and it has the Jacobian coordinate (X_2, Y_2, Z_2), then there is[5]

$$\begin{cases} X_2 = (y_0 Z_1^3 - Y_1)^2 - \\ \qquad (x_0 Z_1^2 - X_1)^2 (X_1 + x_0 Z_1^2) \\ Y_2 = (y_0 Z_1^3 - Y_1)[X_1 (x_0 Z_1^2 - X_1)^2 - X_2] - \\ \qquad Y_1 (x_0 Z_1^2 - X_1)^3 \\ Z_2 = (x_0 Z_1^2 - X_1) Z_1. \end{cases} \tag{8}$$

3. Proposed Modular Multiplier

Generalized Mersenne numbers are widely known to facillitate modular arithemtic[22]–[24], and in this section unified modular arithmetic over conjugated generalized Mersenne numbers $2^L - 2^k \pm 1$ is developed. The main idea is to find out the most common parts between two modular multipliers respectively over 2^L–2^k–1 and 2^L–2^k+1, which results in an area-efficient modular multiplier architecture for hardware implementation.

3.1 New Modular Arithmetic over Generalized Mersenne Numbers

Suppose that A and B are two binary numbers, with $T=A\cdot B=2^L\cdot T_{h,f}+T_{l,f}$. Here $T_{h,f}$ and $T_{l,f}$ are the higher and lower halves of T. In this work, we will process the two parts of T in parallel, in order to avoid the final full addition before modular reduction. As a result, a carry out appears from the lower part, which is denoted as c_f and processed in together. With $T_l = T_{l,f}$, $T_h + c_f = T_{h,f}$, there is

$$T = T_h 2^L + T_l + c_f 2^L. \tag{9}$$

As it is shown in Fig. 1, a 54-bit multiplier can be achieved by 18-bit multipliers, adders, and carry save logic. Similary, other multi-precision multiplications can also be performed by single-precision multiplications and separated into high and low parts.

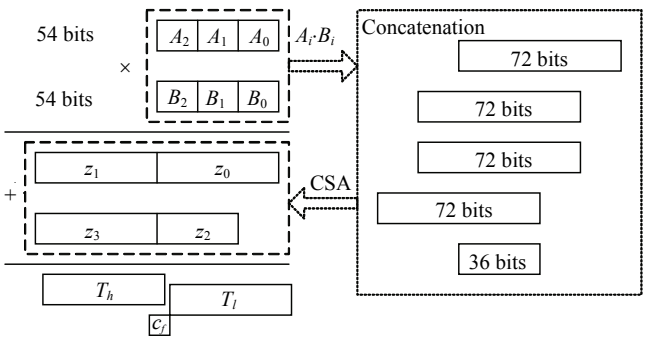

Fig. 1. $L\times L$-bit multiplication with middle carry out bit.

Apparently, the generalized Mersenne numbers can be divided into two kinds: $m=2^L-2^k-1$ and $m=2^L-2^k+1$. At first, we can separately simplify the modular arithmetic with each case.

A. $m=2^L-2^k-1$

Set $T_{h1} = (t_{L-1}\cdots t_{L-k+1}t_{L-k})_2 < 2^k$, $T_{h2} = (t_{L-k-1}\cdots t_1 t_0)_2 < 2^{L-k}$, then $T_h = 2^{L-k}T_{h1} + T_{h2}$, and

$$\begin{aligned} C = T \bmod m \equiv \\ (2^k+1)T_h + T_l + c_f(2^k+1) \equiv \\ (2^k+1)(2^{L-k}T_{h1} + T_{h2}) + T_l + c_f(2^k+1) \equiv \\ (2^L+2^{L-k})T_{h1} + (2^k+1)T_{h2} + T_l + c_f(2^k+1) \equiv \\ (2^L+1+2^{L-k})T_{h1} + (2^k+1)T_{h2} + T_l + c_f(2^k+1). \end{aligned} \tag{10}$$

Let

$$C' = (2^k + 2^{L-k} + 1)T_{h1} + (2^k+1)T_{h2} + T_l + c_f(2^k+1) \tag{11}$$

then $C' \equiv C \pmod m$.

In (11), $c_f(2^k+1)$ is a $(k+1)$-bit number. With $L-k > k+1$, it can be combined with $2^{L-k}T_{h1}$, i.e., $2^{L-k}T_{h1} + c_f(2^k+1) = (t_{L-1}\cdots t_{L-k} \underbrace{0\cdots 0}_{(L-2k-1)\text{ bits}} c_f \underbrace{0\cdots 0}_{(k-1)\text{ bits}} c_f)$. Therefore, only 6 partial products need to be accumulated, all of which are below 2^L. For example, $2^k T_{h1} < 2^{2k} < 2^{L-1}$. In the accumulation, before final carry ripple addition, we can employ three carry save addition to reduce the six parts to two $(L+2)$-bit unsigned numbers.

Next, the bound of C' should be determined by the modulus. It is self-evident that $C \geq 0$. Meanwhile,

$$\begin{aligned} C' \leqslant (2^k + 2^{L-k} + 1)(2^k - 1) + (2^k+1)(2^{L-k} - 1) + \\ 2^L - 1 + 2^k + 1 = \\ 3(2^L - 2^k - 1) + 2^{2k} + 2^{k+1} + 2^k + 1 = \\ 3m + 2^{2k} + 2^{k+1} + 2^k + 1 < \\ 3m + 2^{2k} + 2^{k+2}. \end{aligned} \tag{12}$$

With $m = 2^L - 1 - 2^k = \sum_{i=0}^{L-1} 2^i - 2^k = \sum_{i=0, i\neq k}^{L-1} 2^i$, $k < L/2-1$, i.e., $L > 2k+2$, we have

- If $k=1$, then $L>2+2=4$, i.e., $L\geq 5$. Then $2^{2k} + 2^{k+2} = 12 < 2^{5-1} \leqslant 2^{L-1} < m$.
- If $k=2$, then $L>2\times 2+2=6$. Also, $2^{2k} + 2^{k+2} = 2^5 < 2^{L-1} < m$.
- If $2<k<L-2$, then $m - (2^{2k} + 2^{k+2}) = \sum_{i\neq k, k+2, 2k}^{0\leqslant i\leqslant L-1} 2^i > 0$.

Thus there is still $m > 2^{2k} + 2^{k+1}$.

Finally, there is always $0 \leq C' < 4m$.

B. $m=2^L-2^k+1$

Set $T_{h1} = (t_{L-1}\cdots t_{L-k+1}t_{L-k})_2 < 2^k$ and $T_{h2} = (t_{L-k-1}\cdots$

$t_1t_0)_2 < 2^{L-k}$, then $T_h = 2^{L-k}T_{h1} + T_{h2}$. Thus,

$$\begin{aligned} C = T \bmod m \equiv\ & (2^k-1)T_H + T_L + c_f \cdot 2^L \equiv \\ & (2^k-1)(2^{L-k}T_{h1} + T_{h2}) + T_L + c_f(2^k-1) \equiv \\ & (2^L - 2^{L-k})T_{h1} + (2^k-1)T_{h2} + T_l + c_f(2^k-1) \equiv \\ & (2^k - 1 - 2^{L-k})T_{h1} + (2^k-1)T_{h2} + T_l + c_f(2^k-1) \ (\text{mod } m). \end{aligned} \quad (13)$$

Also, set

$$C'' = (2^k - 1 - 2^{L-k})T_{h1} + (2^k-1)T_{h2} + T_l + c_f(2^k-1) \quad (14)$$

then $C'' \equiv C \ (\text{mod } m)$.

In (14), the partial product $c_f(2^k-1)$ has only k bits, and can be incorporated into $2^k T_{h2}$ as

$$2^k T_{h2} + c_f(2^k-1) = (t_{L-k-1}\cdots t_1 t_0 \underbrace{c_f \cdots c_f}_{k \text{ bits}})_2 .$$

Thus, there are also 6 partial products in accumulation, and carry save addition can be employed to compress it to 2 parts. The upper bound of C'' can be determined as

$$\begin{aligned} C'' = {} & T_l + (2^k-1)T_{h2} + c_f(2^k-1) - (2^{L-k} - 2^k + 1)T_{h1} \le \\ & 2^L - 1 + (2^k-1)(2^{L-k}-1) + 2^k - 1 = \\ & (2^L - 2^k + 1) + (2^L - 2^{L-k} + 2^k - 2) = \\ & m + m - 2^k(2^{L-2k} - 1 - 3\cdot 2^{-k}) \le \\ & 2m - 2^k(2^2 - 1 - 3\cdot 2^{-k}) < \\ & 2m. \end{aligned} \quad (15)$$

As $0 \le T_{h1} \le 2^k - 1$, the lower bound yields

$$\begin{aligned} C'' \ge {} & -(2^{L-k} - 2^k + 1)T_{h1} \ge -(2^{L-k} - 2^k + 1)(2^k - 1) = \\ & (2^{2k} - 2^{k+1}) + 2^{L-k} + 1 - 2^L \ge 2^{k+1} + 2^{L-k} + 1 - 2^L > \\ & 2^k + 1 - 2^L > -m. \end{aligned} \quad (16)$$

Finally, we have $-m < C'' < 2m$. Therefore, the six parts with C'' can be accumulated as six (L+2)-bit signed numbers.

3.2 Unified Modular Multiplier over a Pair of Generalized Mersenne Numbers

The unified modular multiplier here refers to a modular multiplier that works for two conjugated moduli $m=2^L-2^k-1$ or $m=2^L-2^k+1$.

Let us define a sign function at first, i.e.,

$$\text{sign} = \begin{cases} +1, & \text{if} \quad m = 2^L - 2^k - 1 \\ -1, & \text{if} \quad \delta = 2^L - 2^k + 1. \end{cases} \quad (17)$$

Then, set $S_1 = 2^k T_{h1}$, $S_2 = \text{sign} 2^{L-k} T_{h1}$, $S_3 = \text{sign} T_{h1}$, $S_4 = 2^k T_{h2}$, $S_5 = \text{sign} T_{h2}$, $S_6 = T_l$, and $S = \sum_{i=1}^{6} S_i$. These parts are compressed into two (L+2)-bit numbers by carry save addition, e.g., (C_m, S_m), with $C_m + S_m = S$. According to the analysis, S is also an (L+2)-bit number. Assume that the redundant result has been obtained, then up to three times of addition/subtraction is enough to reduce it down to [0, m) by Algorithm 3.

Algorithm 3. Modular reduction in unified modular multiplier

Input: C_m and S_m are both (L+2) bit numbers, with $C_m + S_m \equiv T \ (\text{mod } m)$ and $m = 2^L - (2^k \pm 1)$.

Output: $S = T \bmod m$, $0 \le S < m$.

1: $X_0 := (C_m + S_m) \bmod 2^{L+2}$;
2: $X_1 := X_0 - m$;
3: if sign=+1 then $X_2 := C_m + S_m - 2m$;
4: else $X_2 := C_m + S_m + m$
5: end if
6: $X_3 := C_m + S_m - 3m$;
7: if sign=+1 then
8: if $X_{2,L+1} = 1$ then
9: if $X_{1,L+1} = 1$ then $S := X_{0,L-1\cdots 0}$;
10: else $S := X_{1,L-1\cdots 0}$;
11: end if
12: else
13: if $X_{3,L+1} = 1$ then $S := X_{2,L-1\cdots 0}$;
14: else $S := X_{3,L-1\cdots 0}$;
15: end if
16: end if
17: else //sign=−1
18: if $X_{0,L+1} = 1$ then $S := X_{2,L-1\cdots 0}$;
19: else if $X_{1,L+1} = 1$ then $S := X_{0,L-1\cdots 0}$;
20: else $S := X_{1,L-1\cdots 0}$;
21: end if
22: end if
23: return S.

In the above algorithm, in Line 1 if sign=+1, then $X_0 \in$ [0, $4m$); else $X_0 \in [-m, 2m)$. Line 2 to Line 6 can be computed in parallel with carry save addition. In detail, C_m and S_m produce a "generate" vector and a "propagate" vector, which then take part in the carry save addition at the same time. Also, $X_{i,L-1\cdots 0}$ (i=1, 2, 3) means the lowest L bits of X_i, and $X_{i,L+1}$ means the (L+1)th bit of X_i.

The hardware architecture of a unified modular multiplier is shown in Fig. 2, consisting of two stages. The first stage performs multiplication or concatenation of A and B, while the second stage performs modular reduction of Algorithm 3. In Fig. 2 the REG denotes registers for buffering, Comb means the combination of six parts according to different inputs, CSA is carry save addition, and MUX is the final multiplexor.

Although the architecture supports modular arithmetic over two conjugated and generalized Mersenne numbers, it only deals with one modulus at one time. Meanwhile, as shown in Algorithm 1, the computational steps in an RNS Montgomery modular multiplication take place in sequence from one RNS to another RNS, for which this modular multiplier can be applied.

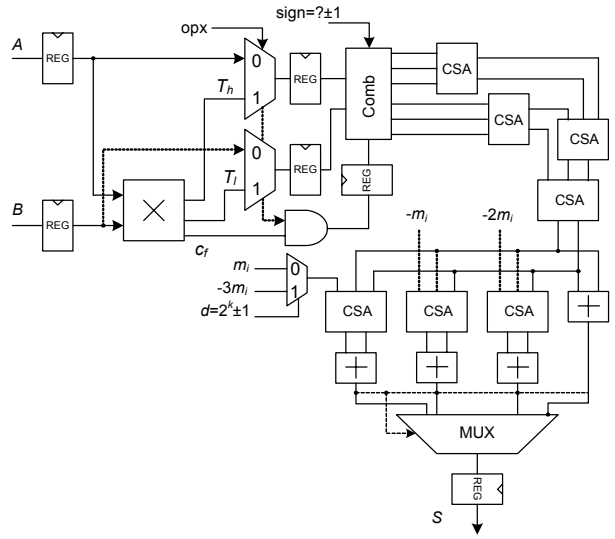

Fig. 2. Unified modular multiplier.

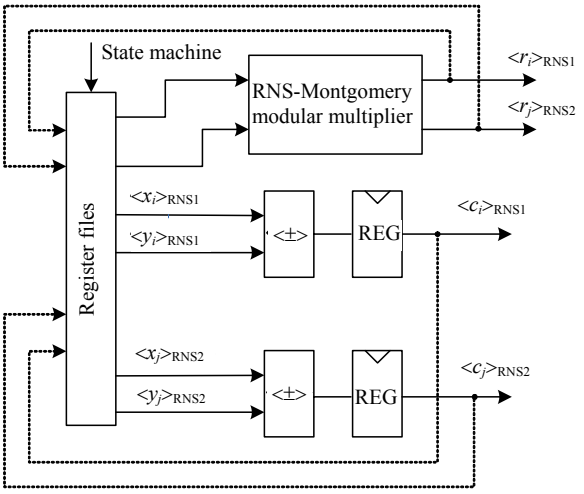

Fig. 3. ECPM based on RNS Montgomery modular multiplier.

4. ECPM by RNS Arithmetic

By RNS Montgomery modular multiplication, ECC can be performed through RNS arithmetic. Obviously, there are two levels of operations: 1) the modulo p operation at a high level and 2) the modulo m_i operation at a low level. The integer p is the characteristic of the elliptic curve field, while $\{m_1, m_2, \cdots, m_n\}$ are RNS moduli. In fact, the implementation of ECC in RNS is just to replace modular arithmetic over p by modular arithmetic over m_i.

It may be doubted that there is a conflict between the RNS Montgomery algorithm and conversion of $\bar{X} = X \cdot M \cdot M^{-1} \bmod p$, because M is represented as zero in RNS. Such a problem can be avoided by setting $\bar{X} = X \cdot (M \bmod p) \cdot M^{-1} \bmod p$.

An ECPM based on RNS Montgomery modular multiplication is illustrated in Fig. 3. In addition to the RNS Montgomery modular multiplier, two groups of modular adders/subtractors are employed to deal with other operations in ECPM. The conversion from the binary number system to RNS and the inverse operation are considered in Section 4.1.

In particular, we have applied the left-to-right signed-digit coding technique[25] in Algorithm 2.

4.1 Conversion between Binary Number System and RNS

In this work, to perform binary-to-RNS conversion, e.g., A mod m_i for i=1, 2, ···, n, a serial modular reduction method is developed, as shown in Algorithm 4. It requires (n−1) sequential steps to reduce an n-word integer.

Algorithm 4. Serial modular reduction

Input: $A = (A^{(n-1)} A^{(n-2)} \cdots A^{(0)})_{2^L}$, $0 \le A^{(i)} < 2^L$, $0 \le m_i < 2^L$.

Output: $S_{n-2} = A \bmod m_i$.

1: $S_0 := (A^{(n-1)} A^{(n-2)})_{2^L} \bmod m_i$;

2: for i=1 to n−2

3: $S_i = [2^L \cdot S_{i-1} + A^{(n-2-i)}] \bmod m_i$;

4: end for

5: return S_{n-2}.

By concatenating S_i to $A^{(n-2-i)}$ in Fig. 2, the above algorithm can be easily implemented, no extra multipliers or memories are required.

The RNS-to-binary conversion is completed by the improved Chinese remainder theorem and *N*. Guillermin's method[13]. Suppose that the final result R is represented by an n-tuple (r_1, r_2, ···, r_n), and it reads ($D_{n-1} \cdots D_1 D_0$) in the binary number system, where $0 \le D_i \le 2^L$. Also, set $R_0 = R$ and precompute $< \gamma_i > = 2^{-L} \bmod < m_i >$.

It is obvious that $D_0 = r_1 = R \bmod 2^L$, and the following process can be done by similar steps. Assume D_{i-1} to D_0 have been obtained, and R_0 has become $R_{i-1} = (r_1^{(i-1)}, r_2^{(i-1)}, \cdots, r_n^{(i-1)})$, then

$$R_i := < r_j^{(i)} > = (< r_j^{(i-1)} - D_{i-1} > \cdot < \gamma_j >) \bmod < m_j > \quad (18)$$

where j=1, 2, ···, n. Obviously, $R_i / M < 2^{-L} \ll 1$. Then, by the improved Chinese remainder theorem we have

$$R_i = \sum_{j=1}^{n} M_i \left| r_j^{(i)} M_j^{-1} \right|_{m_j} - \alpha^{(i)} M \quad (19)$$

where $\alpha^{(i)} = \left\lfloor \sum_{j=1}^{n} \frac{1}{2^L} \mathrm{trunc}\left(\left| r_j^{(i)} M_j^{-1} \right|_{m_j} \right) + 0.5 \right\rfloor$. Thus, with the precomputation of $M_j^{-1} \bmod m_j$, $M_i \bmod 2^L$, and $(-M) \bmod 2^L$, (19) can be used to obtain $D_i = R_i \bmod 2^L$. The accumulation in the equation can be performed by n times of full L-bit addition. The above processes can be carried out until R_{n-1} and D_{n-1} are obtained.

4.2 Moduli Selection

In the RNS Montgomery modular multiplication algorithm, $2n$ RNS moduli are required by Algorithm 1. Suppose the moduli are denoted as $m_k = f_1(k) = 2^L - 2^k - 1$ and $m_{n+k} = f_2(k) = 2^L - 2^k + 1$, with $k \in [1, L/2)$ being a set of numbers $\{k_1, k_2, \cdots, k_n\}$. It is necessary that k_i (i=1, 2, …, n) make all $\{m_{k_i}\}$ and $\{m_{n+k_i}\}$ pairwise prime. Except the generalized Mersenne numbers, two special moduli 2^L and 2^L-1 are included as RNS moduli.

For FPGA implementation, the binary length L should be sufficiently large for coprime moduli and the RNS dynamic range. And it is also set to be around multiples of 18, so as to use the dedicated multipliers in FPGA devices. The seeking of moduli can be done with Wolfram Mathematica. In this work, the RNS moduli for 192-bit, 256-bit, and 384-bit ECC over prime fields are demonstrated in Table 1.

In Table 1, RNS 1 is the original RNS, and RNS 2 is the auxiliary RNS. Modular multiplication over the two moduli m_i and m_{n+i} (n=5, 6, 8) are incorporated into one unified modular multiplier.

Table 1: RNS moduli for prime ECC

Design	RNS	Moduli	
192-bit ECC	RNS 1	$m_1 = 2^{36} - 1$	$m_2 = 2^{36} - 2^2 - 1$
		$m_3 = 2^{36} - 2^6 - 1$	$m_4 = 2^{36} - 2^8 - 1$
		$m_5 = 2^{36} - 2^{12} - 1$	$m_6 = 2^{36} - 2^{14} - 1$
	RNS 2	$m_7 = 2^{36}$	$m_8 = 2^{36} - 2^2 + 1$
		$m_9 = 2^{36} - 2^6 + 1$	$m_{10} = 2^{36} - 2^8 + 1$
		$m_{11} = 2^{36} - 2^{12} + 1$	$m_{12} = 2^{36} - 2^{14} + 1$
256-bit ECC	RNS 1	$m_1 = 2^{54} - 1$	$m_2 = 2^{54} - 2^2 - 1$
		$m_3 = 2^{54} - 2^6 - 1$	$m_4 = 2^{54} - 2^8 - 1$
		$m_5 = 2^{54} - 2^{14} - 1$	/
	RNS2	$m_6 = 2^{54}$	$m_7 = 2^{54} - 2^2 + 1$
		$m_8 = 2^{54} - 2^6 + 1$	$m_9 = 2^{54} - 2^8 + 1$
		$m_{10} = 2^{54} - 2^{14} + 1$	/
384-bit ECC	RNS1	$m_1 = 2^{54} - 1$	$m_2 = 2^{54} - 2^2 - 1$
		$m_3 = 2^{54} - 2^4 - 1$	$m_4 = 2^{54} - 2^8 - 1$
		$m_5 = 2^{54} - 2^{10} - 1$	$m_6 = 2^{54} - 2^{14} - 1$
		$m_7 = 2^{54} - 2^{18} - 1$	$m_8 = 2^{54} - 2^{20} - 1$
	RNS2	$m_9 = 2^{54}$	$m_{10} = 2^{54} - 2^2 + 1$
		$m_{11} = 2^{54} - 2^4 + 1$	$m_{12} = 2^{54} - 2^8 + 1$
		$m_{13} = 2^{54} - 2^{10} + 1$	$m_{14} = 2^{54} - 2^{14} + 1$
		$m_{15} = 2^{54} - 2^{18} + 1$	$m_{16} = 2^{54} - 2^{20} + 1$

4.3 Optimized Elliptic Curve Arithmetic

The point addition and doubling in Section 2 have been reorganized to facilitate hardware implementation[26], which are shown in Algorithm 5 and Algorithm 6. The basic idea is to reduce both the computational steps and buffering registers. The time complexity of one ECPM in this work is $T_{\text{total}}=h\cdot c\cdot N\cdot T_{\text{gmult}}$, where h is a factor determined by ECC arithmetic, c is the number of sequential steps to finish one RNS Montgomery modular multiplication, N is the binary length of ECC field bits, and T_{gmult} is the time of one modular multiplication over generalized Mersenne numbers. In this work, h=15 is for the average case, and h=11 is for the best case. T_{gmult} is just the critical path of the second stage in Fig. 2.

In fact, the lazy RNS modular reduction[27]–[29] can be applied to accelerate the ECPM to some extent[13],[29]. On average, to process one ECC bit, the average number of RNS modular reductions in this work is $11+12\times1/3=15$, where the factor 1/3 is due to signed-digit coding[25]. By contrast, only 13 sequential RNS modular reductions are needed by the lazy RNS reduction[13],[29]. However, this selection of ECC arithmetic steps does not affect the efficiency of unified modular multiplier.

Algorithm 5. Optimized point doubling

Input: Point Q_0 with Jacobian coordinate (X_0, Y_0, Z_0) lies at an elliptic curve, which is defined by $y^2 = x^3 + ax + b$.

Output: $Q_1=2Q_0$ with Q_1= (X_1, Y_1, Z_1).

1: $V_1 = X_0^2$, $F = Y_0 + Y_0$;

2: $V_2 = Z_0^2$, $G_1 = 3\cdot V_1 = V_1 + V_1 + V_1$;

3: $V_3 = V_2^2$;

4: $V_4 = aV_3$;

5: $E = G_1 + V_4$, $G_2 = F^2$;

6: $G_3 = G_2 \cdot X_0$;

7: $V_5 = G_3 + G_3$, $V_6 = E^2$;

8: $X_1 = V_6 - V_5$, $V_7 = F \cdot G_2$;

9: $V_8 = G_3 - X_1$, $G_4 = V_7 \cdot Y_0$;

10: $V_9 = E \cdot V_8$;

11: $Y_1 = V_9 - G_4$, $Z_1 = Z_0 \cdot F$;

12: return (X_1, Y_1, Z_1).

At Line 4 of Algorithm 5, the constant a has been represented in RNS as ($a\cdot M$) mod p by precomputation, where M is the RNS dynamic range.

Algorithm 6. Optimized point addition

Input: Point P_0 is denoted by the affine coordinate (x_0, y_0), and the other point P_1 is denoted as a Jacobian coordinate (X_1, Y_1, Z_1).

Output: $P_2=P_0+P_1$ with a Jacobian coordinate (X_2, Y_2, Z_2).

Table 2: Hardware implementation of F_p ECPM

References	Prime Field	Technology	Max. frequency (MHz)	Area (Slices)	ROM+RAM (bits)	Multipliers/DSP	ECPM time (ms)
This work	192-bit F_p	Xilinx XC2VP100	79.3	17747	32×192	24	1.13
This work	256-bit F_p	Xilinx XC2VP100	73.5	22147	(32+2) ×256	45	1.44
This work	256-bit F_p	Xilinx XC4VLX80	100	21572	(32+2) ×256	45	1.06
This work	384-bit F_p	Xilinx XC4VLX160	99.0	35109	(32+2) ×384	72	2.03
This work	256-bit F_p	TSMC CMOS 0.18 um process	123.46	221K gates	(32+2) ×256	0	0.857
This work	384-bit F_p	TSMC CMOS 0.18 um process	123.46	324K gates	(32+2) ×384	0	1.63
Ref. [30]	192-bit F_p	Xilinx XCV1000E	52.9	25012 LUTs	0	0	2.97
Ref. [30]	256-bit F_p	Xilinx XCV1000E	39.7	32716 LUTs	0	0	3.95
Ref. [13]	192-bit F_p	Altera EP1S30F780C5	89.6	12480 LE	Unspecified	80 DSP	0.72
Ref. [13]	256-bit F_p	Altera EP1S60F780C5	90.7	16200 LE	Unspecified	125 DSP	1.17
Ref. [13]	256-bit F_p	Altera EP2S30F484C3	157.2	9177 ALM	Unspecified	96 DSP	0.68
Ref. [13]	384-bit F_p	Altera EP2S30F484C3	150.9	12958 ALM	Unspecified	177 DSP	1.35
Ref. [6]	192-bit F_p	Xilinx XCV1000E	40	11416 LUTs	35 BRAMs	0	3
Ref. [31]	NIST-256 F_p	Xilinx XC4VSX55	375	24574	176 BRAMs	512 DSP	0.0405
Ref. [31]	NIST-256 F_p	Xilinx XC4VSX55	490	1715	176 BRAMs	32 DSP	0.495
Ref. [32]	256-bit F_p	Xilinx XC2VP125	39.46	15755	0	256	3.86

1: $H_1 = Z_1^2$;

2: $H_2 = x_0 \cdot H_1$;

3: $S = H_2 - X_1$, $U_1 = H_1 \cdot Z_1$;

4: $W = H_2 + X_1$, $U_2 = y_0 \cdot U_1$;

5: $T = U_2 - Y_1$, $H_3 = S^2$;

6: $H_4 = H_3 \cdot W$;

7: $U_3 = T^2$;

8: $X_2 = U_3 - H_4$, $U_4 = X_1 \cdot H_3$;

9: $U_5 = U_4 - X_2$, $H_5 = H_3 \cdot S$;

10: $H_6 = T \cdot U_5$;

11: $U_6 = Y_1 \cdot H_5$;

12: $Y_2 = H_6 - U_6$, $Z_2 = S \cdot Z_1$;

13: return (X_2, Y_2, Z_2).

In Algorithm 5 and Algorithm 6, U_i (i=1, 2, ···, 6) and V_i (i=1, 2, ···, 9) can be buffered by output registers of RNS processing units, while the other variables need extra registers for buffering.

5. Hardware Implementation

In this work, ECPM has been described by verilog hardware description language (VHDL) and then synthesized in Xilinx ISE foundation by Synplify 9.6.2. The palce and route process is completed in Xilinx ISE 10.1. The target devices are Xilinx XC2VP100-6 FF1696 FPGA (130 nm CMOS node) and XC4VLX80-12 FF1148 FPGA (90 nm CMOS node), which can be compared with Altera Stratix and Stratix II devices. This work is aiming at ECPM over general prime fields. The hardware implementation results are shown in Table 2. In order to measure the total area complexity, we have also synthesized the 256-bit and 384-bit F_p ECPM by TSMC 0.18 μm CMOS process, which respectively costs an area of 221K gates (NAND2) and 324 K gates at 123.46 MHz.

The number of sequential steps c in one RNS Montgomery modular multiplication is interfered with hardware implementation. In this work, c=26 for 192-bit ECC, c=24 for 256-bit ECC, and c=30 for 384-bit ECC. Correspondingly, the number of clock cylces for each ECPM is respectively 89984, 105785, and 201143.

Reference [30] is a pioneer for implementing F_p ECPM by RNS, in which the RNS parallelism of addition and multiplication were exploited. However, it deals with modular arithmetic out of RNS and therefore conversions into and out of the binary system become a bottleneck. In order to improve the performance, a pipelined conversion logic unit is designed, and the multiplication in their architecture appears in port conversions between RNS and the binary number system. Compared with their work, the proposed work obtains higher performance by more hardware resources.

In Ref. [13], an efficient RNS architecture with a deep pipeline was proposed for ECPM. It also depends on the RNS Montgomery modular multiplication algorithm, and the moduli are just generalized Mersenne numbers of 2^L–δ. Its critical path is reduced owing to deep pipeline stages. In Table 2, the Altera logic elements (LE) are equivalent to the look-up table (LUT) of the Xilinx Virtex II-pro. Meanwhile, one Altera DSP (digital signal processing)

block accounts for a 9×9-bit multiplier, while one DSP block in Xilinx FPGA includes an 18×18-bit multiplier. In general, this work is inferior to [13] in performance due to only 2 pipeline stages and the relative deficiency of elliptic curve arithmetic with ECPM.

Reference [6] is the initial work for implementing ECPM in FPGA, in which a high-radix Montgomery modular multiplier was used to perform ECPM, which is very area-efficient but relatively slow compared with other designs.

In [31], ECPM architectures for NIST (American National Institute of Standards and Technology) prime fields were implemented in FPGA. It makes best use of FPGA DSP resources to achieve a very high frequency, which is quite fast compared with both hardware and software implementation in the literature. However, the high frequency may be unavailable in many applications. Also it depends heavily on the special DSP units and is only applicable to NIST primes.

The work in [32] focused on the improvement of modular inverse in the elliptic curve field, which brings out good performance compared with related designs. In detail, in Xilinx XC2VP125-9 FPGA it requires 15755 slices and 256 18×18-bit multipliers for a 256-bit F_p ECPM, and computes one ECPM by 3.86 ms. By contrast, our work in Xilinx XC2VP100-8 FPGA requires 22147 slices and 45 18×18-bit multipliers, and completes one ECPM in 1.44 ms. Apparently, the proposed design enjoys better performance than [32].

6. Conclusions

In this work, we have improved the modular arithmetic over generalized Mersenne numbers $m=2^L-2^k\pm1$, by which a unified modular multiplier working over this pair of numbers can be obtained. Modular reductions over such generalized Mersenne numbers can be performed by a little addition and a few multiplexors.

This modular multiplier is then applied for ECPM by the RNS Montgomery modular multiplication algorithm. Together with the partially optimized point addition and doubling for elliptic curve arithmetic, the proposed ECPM demonstrates good performance for hardware implementation in FPGA and application specific integrated circuits (ASIC).

Acknowledgment

The authors would like to thank the comments from anonymous reviewers. Discussion with Prof. Shuguo Li is also acknowledged.

References

[1] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, Boca Raton: CRC Press, 1997.

[2] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[3] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. on Information Theory*, vol. IT-22, pp. 644–654, Nov. 1976.

[4] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, pp. 203–209, 1987, doi: http://dx.doi.org/10.1090/S0025-5718-1987-0866109-5

[5] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*, New York: Springer-Verlag, 2004.

[6] G. Orlando and C. Paar, "A scalable$GF(p)$ elliptic curve processor architecture for programmable hardware," in *Proc. of the 3rd Int. Workshop on Cryptographic Hardware and Embedded Systems*, Paris, 2001, pp. 348–363.

[7] S. Örs, L. Batina, and B. Preneel, "Hardware implementation of elliptic curve processor over $GF(p)$," in *Proc. of IEEE Int. Conf. on Application-Specific Systems, Architectures, and Processors*, Hague, 2003, pp. 433–443.

[8] P. Mohan, *Residue Number Systems: Algorithms and Architectures*, Boston: Kluwer Academic Publishers, 2002.

[9] M. Soderstrand, W. Jenkins, G. Jullien, and F. Taylor, *Residue Number System Arithmetic*: *Modern Applications in Signal Processing*, New York: IEEE Press, 1986.

[10] K. Posch and R. Posch, "Modulo reduction in residue number system," *IEEE Trans. on Parrallel and Distributed Systems*, vol. 6, no. 5, pp. 449–454, 1995.

[11] S. Kawamura, M. Koike, F. Sano, and A. Shimbo, "Cox-rower architecture for fast parallel montgomery multiplication," in *Proc. of Advances in Cryptology-EUROCRYPT* 2000, Bruges, 2000, pp. 523–538.

[12] H. Nozaki, M. Motoyama, A. Shimbo, and S. Kawamura, "Implementation of rsa algorithm based on rns montgomery modular multiplication," in *Proc. of the 3rd Int. Workshop on Cryptographic Hardware and Embedded Systems*, Paris, 2001, pp. 364–376.

[13] N. Guillermin, "A high speed coprocessor for elliptic curve scalar multiplications over F_p," in *Proc. of Cryptographic Hardware and Embedded Systems 2010*, Santa Barbara, 2010, pp. 48–64.

[14] Z. Lim and B. Phillips, "An rns-enhanced microprocessor implementation of public key cryptography," in *Proc. of the 41st Asilomar Conf. on Signals, Systems and Computers*, Pacific Grove, 2007, pp. 1430–1434.

[15] J. Bajard, L. Didier, and P. Kornerup, "An RNS montgomery modular multiplication algorithm," *IEEE Trans. on Computers*, vol. 47, no. 7, pp. 766–776, 1998.

[16] J. Bajard, L. Didier, and P. Kornerup, "Modular multiplication and base extensions in residue number systems," in *Proc. of the 15th IEEE Symposium on Computer Arithmetic*, Vail, 2001, pp. 59–65.

[17] J. Bajard and L. Imbert, "A full RNS implementation of RSA," *IEEE Tran. on Computers*, vol. 53, no. 6, pp. 769–774, 2004.

[18] D. Schinianakis and T. Stouraitis, "A rns montgomery multiplication architecture," in *Proc. of IEEE Symposium on Circuits and Systems*, Rio de Janeiro, 2011, pp. 1167–1171.

[19] Z. Lim, B. Phillips, and M. Liebelt, "Elliptic curve digital signature algorithm over $g\,f\,(p)$ on a residue number system enabled microprocessor," in *Proc. of IEEE Region 10 Conf., TENCON*, Singapore, 2009, pp. 1–6.

[20] S. Antão, J. Bajard, and L. Sousa, "Elliptic curve point multiplication on gpus," in *Proc. of the 21st IEEE Int. Conf. on Application-Specific Systems, Architectures and Processors*, Rennes, 2010, pp. 192–199.

[21] A. Shenoy and R. Kumaresan, "Fast base extension using a redundant modulus in rns," *IEEE Trans. on Computers*, vol. 38, no. 2, pp. 292–297, 1989.

[22] A. Hiasat, "New efficient structure for a modular multiplier for RNS," *IEEE Trans. on Computers*, vol. 49, no. 2, pp. 170–174, 2000.

[23] M. Ciet, M. Neve, E. Peeters, and J.-J. Quisquater, "Parallel FPGA implementation of RSA with residue number systems —can sidechannel threats be avoided?" in *Porc. of the 46th IEEE Int. Midwest Symposium on Circuits and Systems*, Cairo, 2003, pp. 806–810.

[24] J. Bajard, M. Kaihara, and T. Plantard, "Selected rns bases for modular multiplication," in *Porc. of the 19th IEEE Symposium on Computer Arithmetic*, 2009, pp. 25–32.

[25] M. Joye and S. Yen, "Optimal left-to-right binary signed-digit recoding," *IEEE Trans. on Computers*, vol. 49, no. 7, pp. 740–748, 2000.

[26] M. Brown, D. Hankerson, J. López, and A. Menezes, "Software implementation of the NIST elliptic curves over prime fields," in *Proc. of Topics in Cryptology-CT-RSA 2001*, San Francisco, 2001, pp. 250–265.

[27] R.-C. Cheung, S. Duquesne, J. Fan, N. Guillermin, I. Verbauwhede, and G.-X. Yao, "FPGA implementation of pairings using residue number system and lazy reduction," in *Proc. of Cryptographic Hardware and Embedded Systems*, Nara, 2011, pp. 421–441.

[28] J. Bajard, L. Imbert, P. Liardet, and T. Yannick, "Leak resistant arithmetic," in *Proc. of Cryptographic Hardware and Embedded Systems*, Cambridge, 2004, pp. 62–75.

[29] J. Bajard, S. Duquesne, and M. Ercegovac, Combining Leakresistant Arithmetic for Elliptic Curves Defined over F_p and RNS Representation. [Online]. Available: http://eprint.iacr.org

[30] D. Schinianakis, A. Fournaris, H. M. A. Kakarountas, and T. Stouraitis, "An rns implementation of an *Fp* elliptic curve point multiplier," *IEEE Trans. on Circuits and Systems, I: Regular Papers*, vol. 56, no. 6, pp. 1202–1213, 2009.

[31] T. Güneysu and C. Paar, "Ultra high performance ecc over NIST primes on commercial FPGAs," in *Proc. of Int. Workshop on Cryptographic Hardware and Embedded Software*, Washington, D.C., 2008, pp. 62–78.

[32] C. McIvor, M. McLoone, and J.V.McCanny, "Hardware elliptic curve cryptographic processor over GF(p)," *IEEE Trans. on Circuits and Systems, I: Regular Papers*, vol. 53, no. 9, pp. 1946–1957, 2006.

Tao Wu was born in 1981 in Hubei Province. He received the B.S. degree in electronic science and technology from Wuhan University, Wuhan in 2003 and the M.S. degree from Tsinghua University, Beijing in 2006. From September 2006 to April 2007, he served as a temporary assistant with the Device and System Laboratory, the Institute of Micro-electronics, Tsinghua University. Then he worked with the Department of Physics and Electronic Engineering, Guangxi Normal University from July 2007 to July 2008 . Since September 2008 he has been pursuing the Ph.D. degree with Tsinghua University. His current research refers to circuits and computer arithmetic about public-key cryptography.

Li-Tian Liu was born in 1947 in Jiangxi Province. He received the B.S. degree in electronic engineering from Tsinghua University, Beijing in 1970. He is currently a full professor with the Institute of Microelectronics, Tsinghua University. His research interests include the development of semiconductor devices and integrated circuits.