Use of Residue Number System for ECC

Karim Bigou

INRIA DGA IRISA CAIRN

Journées C2 2012: 8 au 12 octobre

Context and Objectives

- Elliptic Curve Cryptography (ECC) over $\mathbb{F}_p$
- Residue Number System (RNS) $\longrightarrow$ high performances
- Efficient implementation by N. Guillermin in [2] on FPGA
- **My Ph. D. objectives**
  - Speed (parallelism)
  - Protections against some side-channel attacks (randomization)

Elliptic Curve Cryptography

- Elliptic curve E over $\mathbb{F}_p$ (p prime, 160-500 bits) :

$$y^2 = x^3 + a\,x + b$$

with $-16(4a^3 + 27b^2) \neq 0 \bmod p$

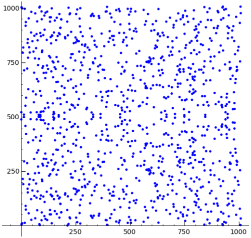


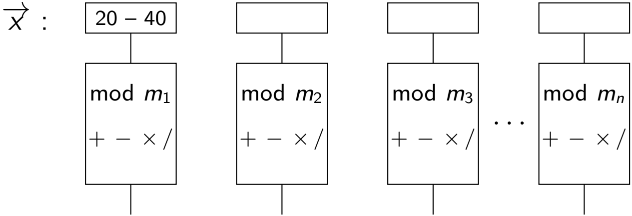
Figure: $y^2 = x^3 + 4x + 20$ over $\mathbb{F}_{1009}$

- A group law $+$ is defined over E: the chord-and-tangent rule
- Scalar multiplication: $[k]P = \underbrace{P + P + \ldots + P}_{k \text{ times}}$
- Knowing P and $[k]P$, k cannot be recovered (ECDLP)
- $[k]P$ must be fast and robust
- Required many operations at field level: $+, -, \times$ and inversions

Residue Number System

X : | 160 – 521 |

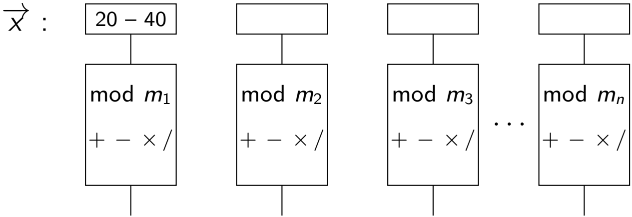
Residue Number System



- RNS Base: $(m_1, \ldots, m_n)$ co-primes
- If $0 \leqslant x < \prod_{i=1}^{n} m_i$ then $\overrightarrow{x}$ is determined by

$$\overrightarrow{x} = (x_1, \ldots, x_n) = (x \bmod m_1, \ldots, x \bmod m_n)$$

Residue Number System

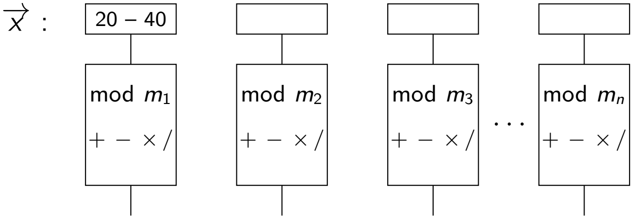


- RNS Base: $(m_1, \ldots, m_n)$ co-primes
- If $0 \leqslant x < \prod_{i=1}^{n} m_i$ then $\overrightarrow{x}$ is determined by

$$\overrightarrow{x} = (x_1, \ldots, x_n) = (x \bmod m_1, \ldots, x \bmod m_n)$$

- Carry-free between blocks

Residue Number System

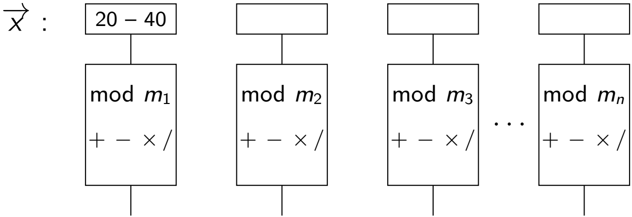


- RNS Base: $(m_1, \ldots, m_n)$ co-primes
- If $0 \leqslant x < \prod_{i=1}^{n} m_i$ then $\overrightarrow{x}$ is determined by

$$\overrightarrow{x} = (x_1, \ldots, x_n) = (x \bmod m_1, \ldots, x \bmod m_n)$$

- Carry-free between blocks
- Fast Parallel Addition, Substraction, Multiplication and Exact Division

Residue Number System

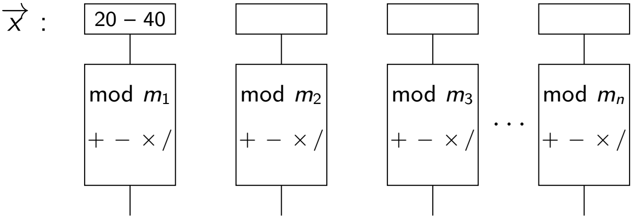


- RNS Base: $(m_1, \ldots, m_n)$ co-primes
- If $0 \leqslant x < \prod_{i=1}^{n} m_i$ then $\overrightarrow{x}$ is determined by

$$\overrightarrow{x} = (x_1, \ldots, x_n) = (x \bmod m_1, \ldots, x \bmod m_n)$$

- Carry-free between blocks
- Fast Parallel Addition, Substraction, Multiplication and Exact Division
- Non-positional System

Residue Number System



- RNS Base: $(m_1, \ldots, m_n)$ co-primes
- If $0 \leqslant x < \prod_{i=1}^{n} m_i$ then $\overrightarrow{x}$ is determined by

$$\overrightarrow{x} = (x_1, \ldots, x_n) = (x \bmod m_1, \ldots, x \bmod m_n)$$

- Carry-free between blocks
- Fast Parallel Addition, Substraction, Multiplication and Exact Division
- Non-positional System
- Comparison, Modular Reduction, Division are hard

Base Extension

- Idea for modular reduction: add redundancy using 2 bases
- $\mathcal{B} = (m_1, \dots, m_n)$ and $\mathcal{B}' = (m'_1, \dots, m'_n)$ are coprime RNS bases
- x is $\overrightarrow{x}$ in $\mathcal{B}$ and $\overrightarrow{x}'$ in $\mathcal{B}'$
- The base extension (BE) is defined by:

$$BE(\overrightarrow{x}, \mathcal{B}, \mathcal{B}') = \overrightarrow{x}'$$

- Some operations become possible after a base extension
  - $M = \prod_{i=1}^{n} m_i$ is invertible in $\mathcal{B}'$
  - Exact Division by M can be done easily

Kawamura's Base Extension [4]

It is based on the chinese remainder theorem (CRT) :

$$x = x \bmod M = \left| \sum_{i=1}^{n} |x_i \cdot M_i^{-1}|_{m_i} \cdot M_i \right|_M$$

$$|x|_{m_i'} = \left| \sum_{i=1}^{n} |x_i \cdot M_i^{-1}|_{m_i} \cdot M_i - k \cdot M \right|_{m_i'}$$

where $M = \prod_{i=1}^{n} m_i$, $M_i = \frac{M}{m_i}$ and $k < n$.

- Sum done over each channel of $\mathcal{B}'$
- Operation cost is n^2 word multiplications
- Precomputations are needed ($|M_i^{-1}|_{m_i}$, $|M_i|_{m_i'}$, $|M|_{m_i'}$)

Kawamura's base extension [4]

$$\frac{X}{M} + k = \sum_{i=1}^{n} \frac{|x_i \cdot M_i^{-1}|_{m_i}}{m_i}$$

$$k = \left\lfloor \sum_{i=1}^{n} \frac{|x_i \cdot M_i^{-1}|_{m_i}}{m_i} \right\rfloor = \alpha + \left\lfloor \sum_{i=1}^{n} \frac{trunc(|x_i \cdot M_i^{-1}|_{m_i})}{2^r} \right\rfloor$$

- m_i pseudo-Mersenne number $m_i = 2^r - \varepsilon_i$
- $trunc(y)$ sets the $r - t$ last bits of y to 0
- α is a correction term
- a simple accumulator can be used in parallel

Montgomery Modular Reduction

Algorithm 1: Montgomery Reduction [5]

Data: $x < 4\,p^2$ and $p < R$ with $gcd(p, R) = 1$

Result: $\mathrm{MM}(x,p) = \omega \equiv x \cdot R^{-1} \bmod p$, $0 \leqslant \omega < 2p$

begin

$q \longleftarrow (x \cdot (-p^{-1})) \bmod R$

$s \longleftarrow x + q \cdot p$

$\omega \longleftarrow s \cdot R^{-1}$

- Classical case: $R = 2^w$ to have easy division and modular reduction
- Introduce Montgomery representation
- RNS: easy reduction modulo M but division impossible in $\mathcal{B}$

Modular Reduction in RNS

Algorithm 2: RNS Montgomery Reduction [1]

Data: $\overrightarrow{x}$, $\overrightarrow{x}'$ with $x < \alpha p^2$

Data: $M > \alpha p$ and $M' > 2 \cdot p$

Result: $\text{RNSMM}(x, p, \mathcal{B}, \mathcal{B}') = \overrightarrow{\omega} \equiv x \cdot M^{-1} \bmod p$ in both bases, $0 \leqslant \omega < 2p$

begin

$\overrightarrow{q} \longleftarrow \overrightarrow{x} \cdot (-\overrightarrow{p}^{-1})$ in base $\mathcal{B}$ and $\mathcal{B}'$

$\overrightarrow{q}' \longleftarrow BE(\overrightarrow{q}, \mathcal{B}, \mathcal{B}')$

$\overrightarrow{s}' \longleftarrow \overrightarrow{x}' + \overrightarrow{q}'\overrightarrow{p}'$ in base $\mathcal{B}'$

$\overrightarrow{\omega}' \longleftarrow \overrightarrow{s}' \times \overrightarrow{M}^{-1}$ in base $\mathcal{B}'$

$\overrightarrow{\omega}' \longleftarrow BE(\overrightarrow{\omega}, \mathcal{B}', \mathcal{B})$

Costs for RNS Modular Operations

a, b, p : n words of w bits

Operation	RNS	Standard
ab	$2\,n$ mult.	n^2 mult.
$a \bmod p$	$2\,n^2 + 3\,n$ mult.	$n^2 + n$ mult.
$ab \bmod p$	$2\,n^2 + 5n$ mult.	$2\,n^2 + n$ mult.
$(ab + cd) \bmod p$	$2\,n^2 + 7\,n$ mult.	$3\,n^2 + n$ mult.
$\left(\sum_{i=1}^{k} a_i b_i\right) \bmod p$	$2\,n^2 + (3 + 2k)n$ mult.	$(1 + k)\,n^2 + n$ mult.

Factor 2 in multiplication is due to the use of 2 bases for BE.

This was implemented on FPGA by Guillermin [2] and is called *lazy reduction*.

Ph. D. Objectives

- Improve ECC with RNS performances
  - Modular Reduction
  - Modular Inversion

- Introduce new protections
  - Use natural RNS properties against SCA
    - Randomization
    - Use redundancy with 2 bases

- Hardware implementation
  - Implementation of RNS arithmetic units
  - FPGA implementation of the new inversion
  - Implementation of randomization
  - Study costs
    - Speed
    - Area

Modular Inversion in RNS

- Comparison and Division hard $\longrightarrow$ Euclidean Algorithm hard
- Binary Euclidean algorithm: needs for modulo reduction and division by 2
- Implemented algorithm: Fermat's Little Theorem
  - very costly
  - a lot of wait cycles
- Proposed solution: Adaptation of Plus-Minus algorithm

Algorithm 3: Plus Minus

Data: $a, p \in \mathbb{N}$ with $gcd(a, p) = 1,\ k = \lceil \log_2 p \rceil$

Result: $\text{PM}(a,p,k) = a^{-1} \bmod p$

begin

$(U_1, U_3) \longleftarrow (0, p), (V_1, V_3) \longleftarrow (1, a), \alpha \longleftarrow k\ ,\ \beta \longleftarrow k$

while $\beta > 0$ **do**

  if $V_3 \equiv 0\ mod\ 4$ **then**

    $V_3 \longleftarrow V_3/4,\ V_1 := divideBy4(V_1, p),\ \beta = \beta - 2$

  else if $V_3 \equiv 0\ mod\ 2$ **then**

    $V_3 \longleftarrow V_3/2,\ V_1 := divideBy2(V_1, p),\ \beta = \beta - 1$

  else

    $V_3' \longleftarrow V_3,\ V_1' \longleftarrow V_1,\ \alpha' \longleftarrow \alpha,\ \beta' \longleftarrow \beta$

    if $U_3 + V_3 \equiv 0\ mod\ 4$ **then**

      $V_3 \longleftarrow (V_3 + U_3)/4,\ V_1 \longleftarrow divideBy4(V_1 + U_1, p)$

    else

      $V_3 \longleftarrow (V_3 - U_3)/4,\ V_1 \longleftarrow divideBy4(V_1 - U_1, p)$

    if $\beta < \alpha$ **then**

      $U_3 \longleftarrow V_3',\ U_1 \longleftarrow V_1'\ ,\ \alpha \longleftarrow \beta'\ ,\ \beta \longleftarrow \alpha' - 1$

    else $\beta \longleftarrow \beta - 1$

if $U_1 < 0$ **then** $U_1 \longleftarrow U_1 + p$

if $U_3 = 1$ **then return** U_1 **else return** $p - U_1$

Algorithm 4: RNS Plus Minus

Data: $\vec{a}$, $\vec{p}$, $p > 2$ with $gcd(a, p) = 1$

begin

- $\vec{u_1} \longleftarrow \vec{c}$, $\vec{u_3} \longleftarrow \vec{p} \cdot \vec{\mu} + \vec{c}$, $\vec{v_1} \longleftarrow \vec{\mu} + \vec{c}$, $\vec{v_3} \longleftarrow \vec{a} \cdot \vec{\mu} + \vec{c}$, $\alpha = \beta = 0$
- $b_{v_1} \longleftarrow 1$, $b_{u_1} \longleftarrow 0$, $b_{u_3} \longleftarrow \vec{p} \bmod 4$, $b_{v_3} \longleftarrow ComputeMod4(\vec{v_3})$
- **while** $\vec{v_3} \neq \vec{\mu} + \vec{c}$ *and* $\vec{u_3} \neq \vec{\mu} + \vec{c}$ *and* $\vec{v_3} \neq -\vec{\mu} + \vec{c}$ *and* $\vec{u_3} \neq -\vec{\mu} + \vec{c}$ **do**
  - **while** b_{v_3} *mod* $2 = 0$ **do**
    - **if** $b_{v_3} = 0$ **then** $r \longleftarrow 2$ **else** $r \longleftarrow 1$
    - $\vec{v_3} \longleftarrow DivideBy2r(\vec{v_3}, r, b_{v_3})$
    - $\vec{v_1} \longleftarrow DivideBy2r(\vec{v_1}, r, b_{v_1})$
    - $b_{v_3} \longleftarrow ComputeMod4(\vec{v_3})$
    - $b_{v_1} \longleftarrow ComputeMod4(\vec{v_1})$
    - $\beta \longleftarrow \beta + r$
  - $\vec{t_3} \longleftarrow \vec{v_3}$, $\vec{t_1} \longleftarrow \vec{v_1}$
  - **if** $b_{v_3} + b_{u_3}$ *mod* $4 = 0$ **then**
    - $\vec{v_3} \longleftarrow DivideBy2r(\vec{v_3} + \vec{u_3} - \vec{c}, 2, 0)$
    - $\vec{v_1} \longleftarrow DivideBy2r(\vec{v_1} + \vec{u_1} - \vec{c}, 2, (b_{v_1} + b_{u_1}) \bmod 4)$
    - $b_{v_3} \longleftarrow ComputeMod4(\vec{v_3})$
    - $b_{v_1} \longleftarrow ComputeMod4(\vec{v_1})$
  - **else**
    - $\vec{v_3} \longleftarrow DivideBy2r(\vec{v_3} - \vec{u_3} + \vec{c}, 2, 0)$
    - $\vec{v_1} \longleftarrow DivideBy2r(\vec{v_1} - \vec{u_1} + \vec{c}, 2, (b_{v_1} - b_{u_1}) \bmod 4)$
    - $b_{v_3} \longleftarrow ComputeMod4(\vec{v_3})$
    - $b_{v_1} \longleftarrow ComputeMod4(\vec{v_1})$
  - **if** $\beta > \alpha$ **then**
    - $\vec{u_3} \longleftarrow \vec{t_3}$, $\vec{u_1} \longleftarrow \vec{t_1}$
    - $\alpha \longleftrightarrow \beta$
  - $\beta \longleftarrow \beta + 1$
- **if** $\vec{v_3} = \vec{\mu} + \vec{c}$ **then return** $(\vec{v_1} - \vec{c}) \cdot (\vec{\mu})^{-1} + \vec{p}$ **else if** $\vec{u_3} = \vec{\mu} + \vec{c}$ **then return** $(\vec{u_1} - \vec{c}) \cdot (\vec{\mu})^{-1} + \vec{p}$
  else if $\vec{v_3} = -\vec{\mu} + \vec{c}$ **then return** $-(\vec{v_1} - \vec{c}) \cdot (\vec{\mu})^{-1} + \vec{p}$ **else return** $-(\vec{u_1} - \vec{c}) \cdot (\vec{\mu})^{-1} + \vec{p}$

	Fermat	RNS Plus-Minus	HW RNS Plus-Minus
multiplications	31104	2738	3011
additions	22464	4410	6046
Cox additions	3456	2738	3693

Comparison of the number of operations with $\lceil \log_2 p \rceil = 192$ and $n = 6$

References I

[1] J.-C Bajard, L.-S. Didier, and P. Kornerup.
An RNS montgomery modular multiplication algorithm.
IEEE Transactions on Computers, 47:234–239, July 1998.

[2] N. Guillermin.
A high speed coprocessor for elliptic curve scalar multiplications over $\mathbb{F}_p$.
In *Proc. Cryptographic Hardware and Embedded Systems (CHES)*, volume 6225 of *LNCS*, pages 48–64. Springer, August 2010.

[3] D. R. Hankerson, S. A. Vanstone, and A. J. Menezes.
Guide to Elliptic Curve Cryptography.
Springer, 2004.

[4] S. Kawamura, M. Koike, F. Sano, and A. Shimbo.
Cox-rower architecture for fast parallel montgomery multiplication.
In *Advances in Cryptology-EUROCRYPT*, volume 1807, pages 523–538. Springer, 2000.

[5] P. L. Montgomery.
Modular multiplication without trial division.
Mathematics of Computation, 44(170):519–521, 1985.