

# Library for Prototyping the Computer Arithmetic Level in Crypto Applications

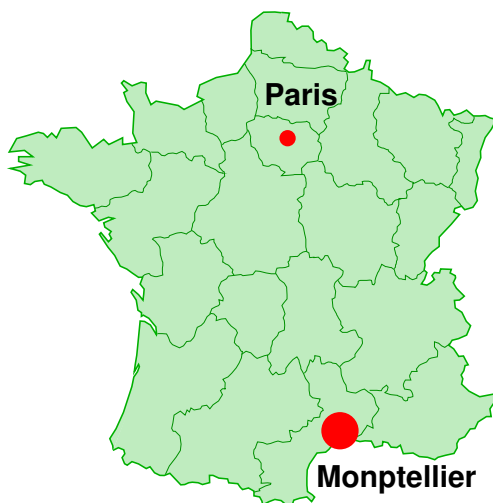
Arnaud Tisserand

LIRMM, CNRS–Univ. Montpellier 2  
Arith Group

Waterloo, August 2007



## Montpellier?



## Outline

### Introduction

- LIRMM Laboratory
- Arith Group
- Design of Computer Arithmetic Operators

### Some Research Activities

- Residue Number Systems
- Addition Chains
- Double-Base Number Systems

### PACE Library

- PACE Overview
- PACE Architecture
- Arithmetic Layer
- Monitoring Layer
- Application Layer (ECC)
- Validation
- Other Tools

### Future Prospects

A. Tisserand, Arith–LIRMM, CNRS, Univ. Montpellier

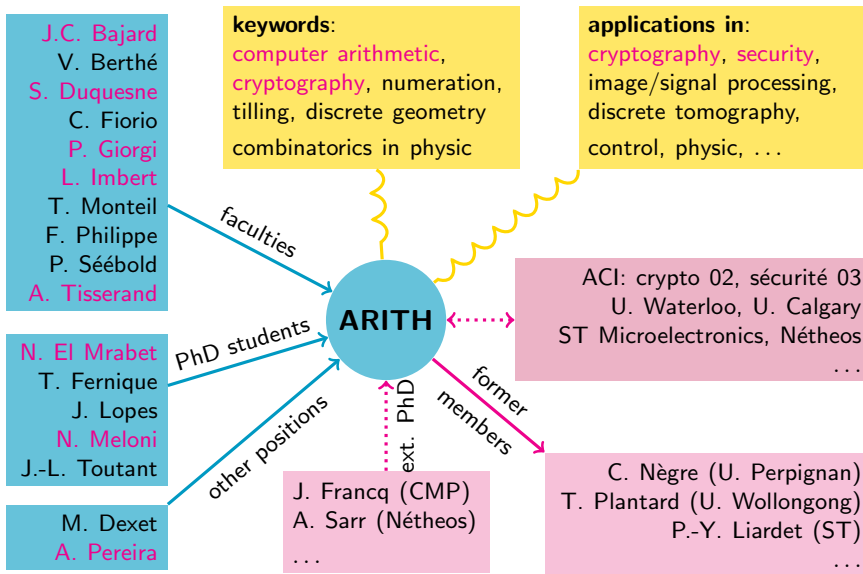
2/30

## LIRMM

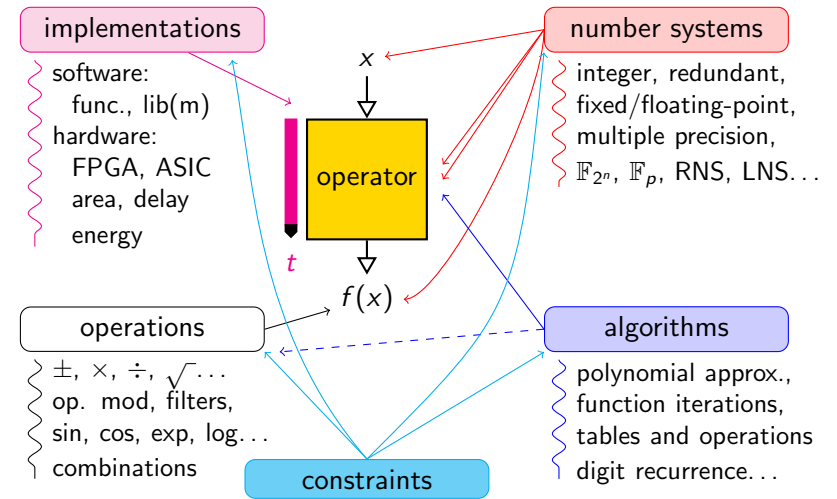
Montpellier Laboratory of Computer Science, Robotics, and Microelectronics

| Computer Science                  | Robotics                                       | Micro-electronics               |
|-----------------------------------|--|---------------------------------|
| 11 teams/projects                 | 5 teams/projects                               | 2 teams/projects                |
| 76 faculties<br>72 PhD students   | 24 faculties<br>25 PhD students                | 24 faculties<br>37 PhD students |
| working group on digital security |  |                                 |
| CNRS                              | <a href="http://www.lirmm.fr">www.lirmm.fr</a> | Univ. Montpellier 2             |

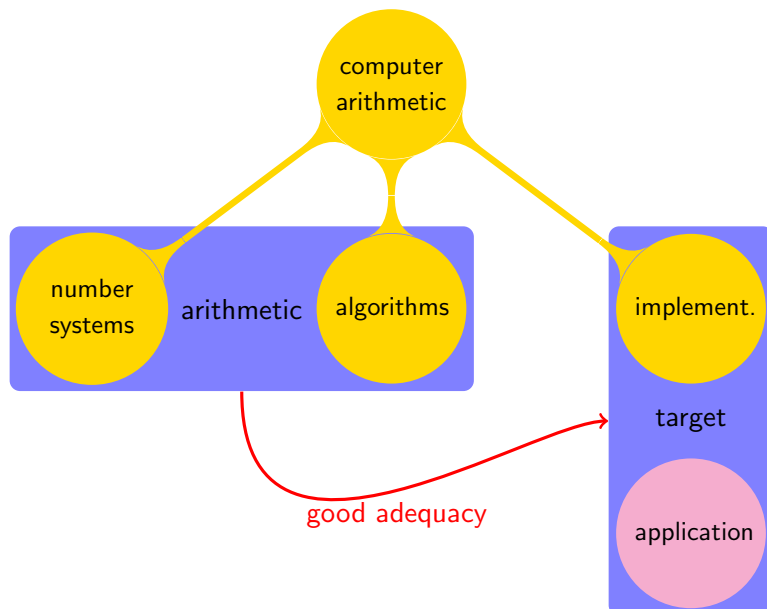
## Arith Group



## Design of Computer Arithmetic Operators



## Problems in Computer Arithmetic



## Practical Problems in Computer Arithmetic

- **limited support in design tools**
  - software: integer, floating-point, libraries
  - hardware: integer, fixed-point, a few IP blocs
- **validation**  
verification of the correctness of a program (function, library, hardware bloc, circuit) at design time
- **test**  
verification of the correctness of an implementation

Our solution: **number systems**, **algorithms** and **TOOLS**

- libraries for software and hardware implementations
- automatic generation of low-level descriptions (C and VHDL)
- include new arithmetic types and primitives in design tools (compilers, CAD tools)

## Some Research Activities

Computer arithmetic for cryptography applications:

- modular arithmetic
- hyperelliptic curves
- implementation of basic crypto primitives
- **residue number systems**
- **double-base number systems**
- **addition chains for ECC implementations**
- secured arithmetic operators design
  - ▶ power-consumption aspects
  - ▶ fault injection
- implementations of applications
- **PACE library**
- pairings

[www.lirmm.fr/arith/](http://www.lirmm.fr/arith/)

## Residue Number Systems(RNS)

Contact: J.C. Bajard

- Public Key Cryptography uses **large** integers
- 192 to 521 bits for ECC, more than 1024 bits for RSA
- Residue Number Systems distribute large integer operations over small residues (parallel computations for + and ×)
- Operation with large integers made independently on each residue (non-positional number system)

## Residue Number Systems

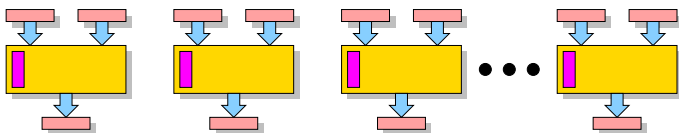
Theorem (Chinese Remainder Theorem)

$(m_1, m_2, \dots, m_n)$  a set of coprime integers and  $M = \prod_{i=1}^n m_i$ . If  $(x_1, x_2, \dots, x_n)$  such that  $x_i < m_i$ , then there is a unique  $X$  such that:

$$0 \leq X < M \quad \text{and} \quad x_i = X \bmod m_i = |X|_{m_i} \quad \text{for } 1 \leq i \leq n$$

Definitions

- The set  $(m_1, m_2, \dots, m_n)$  of coprimes is called RNS basis.
- $X = (x_1, x_2, \dots, x_n)_{RNS}$ ,  $0 \leq X < M$ ,  $x_i = X \bmod m_i, \forall i$
- $A \pm B = (|a_1 \pm b_1|_{m_1}, \dots, |a_n \pm b_n|_{m_k})_{RNS}$
- $A \times B = (|a_1 \times b_1|_{m_1}, \dots, |a_n \times b_n|_{m_k})_{RNS}$



## Residue Number Systems

Example

Consider the RNS basis  $\mathcal{B} = (3, 7, 13, 19)$ .

All the numbers lower than  $M = 5187$  are represented


$$\begin{aligned} X &= 147 & Y &= 31 \\ \bar{X} &= (0, 0, 4, 14)_{RNS} & \bar{Y} &= (1, 3, 5, 12)_{RNS} \end{aligned}$$


The addition and the multiplication mod  $M$  become:

$$\begin{aligned} \bar{X} + \bar{Y} &= (|0 + 1|_3, |0 + 3|_7, |4 + 5|_{13}, |14 + 12|_{19})_{RNS} \\ &= (1, 3, 9, 7)_{RNS} \\ &= 178 \\ \bar{X} \times \bar{Y} &= (|0 \times 1|_3, |0 \times 3|_7, |4 \times 5|_{13}, |14 \times 12|_{19})_{RNS} \\ &= (0, 0, 7, 16)_{RNS} \\ &= 4557 \end{aligned}$$

## Residue Number Systems

- [1, 2] Prime finite fields  $\mathbb{F}_p$  and modular arithmetic: RNS Montgomery multiplication algorithm, Leak Resistant arithmetic, Applications to RSA and ECC...
- [3] Arithmetic in  $\mathbb{F}_{2^k}$  using trinomial residue arithmetic
- [4] Lagrange representations  $\mathbb{F}_{p^k}$  of medium prime characteristic  $p$

 J.C. Bajard et L. Imbert, *A Full RNS Implementation of RSA*, IEEE Transactions on Computers, juin 2004 (Vol. 53, No. 6) p. 769-774

 J.C. Bajard, L. Imbert, P.Y. Liardet, Y. Tiglia, *Leak Resistant Arithmetic*, Workshop on Cryptographic Hardware and Embedded Systems CHES 2004, in LNCS, pages 62-75, Cambridge (Boston), USA, août 11-13, 2004.

 J.C. Bajard, L. Imbert, and G. A. Jullien, *Parallel Montgomery Multiplication in  $GF(2^k)$  using Trinomial Residue Arithmetic*, in Proceedings of the 17th IEEE symposium on Computer Arithmetic (ARITH 17) juin 2005, Cape Cod, MA, USA.

 J.C. Bajard, L. Imbert et C. Nègre, *Arithmetic Operations in Finite Fields of Medium Prime Characteristic Using the Lagrange Representation*, IEEE Transactions on Computers, septembre 2006 (Vol. 55, No. 9) p. 1167-1177

## Addition Chains

Contact: N. Meloni

Elliptic curve point scalar multiplication

**Input:**  $P$  a point of a curve  $E$ , an integer  $k = \sum_{i=0}^{n-1} k_i 2^i$

**Output:** the point  $Q = [k]P = \underbrace{P + P + P + \dots + P}_{k \text{ times}}$

Standard implementation: double-and-add

1:  $Q \leftarrow P$

2: **for**  $i$  **from**  $n-2$  **to**  $0$  **do**

3:      $Q \leftarrow 2P$

4:     **if**  $k_i = 1$  **then**  $Q \leftarrow Q + P$

Problem: **not resistant to SPA!**

## Addition Chains

Some algorithms are “more” resistant to SPA:  $w$ -NAF recoding

In

$$k = \sum_{i=0}^{n-1} k_i 2^i, \quad k_i \in \{0, 1\}$$

process  $w$  digit at a time

$$|k_i| < 2^{w-1}$$

Example:

$$k = 267 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & \bar{1} & 0 & \bar{1} \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 3 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \bar{5} \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 11 \end{pmatrix} \begin{matrix} )_2 \\ )_{2-NAF} \\ )_{3-NAF} \\ )_{4-NAF} \\ )_{5-NAF} \end{matrix}$$

Leads to  $n - 1$  DBL and  $\frac{n}{w+1}$  ADD

## Addition Chains

Addition chain (AC): finite sequence of integers  $v_1, \dots, v_n$  satisfying  $\forall l < n, v_l = v_i + v_j$  for some  $i, j < n$

Euclidean addition chain: AC where  $v_1 = 1, v_2 = 2, v_3 = v_1 + v_2$  and  $\forall 3 \leq i \leq n - 1$  if  $v_i = v_{i-1} + v_j$  for some  $j < i - 1$  then  $v_{i+1} = v_i + v_{i-1}$  or  $v_{i+1} = v_i + v_j$

**advantage:** resistant to SPA attacks

**drawback:** problem to find a short chain

Find a chain for the integer  $k$ : choose  $k'$  coprime with  $k$  and apply the subtractive form of Euclid's algorithm.

New formulae for the point addition:

**input:**  $P_1$  and  $P_2$  have the same  $Z$

**output:**  $ADD(P_1, P_2) = (P_1 + P_2, P_1)$  where  $P_1 + P_2$  and  $P_1$  have the same  $Z$

## Addition Chains

Example:  $k = 34$ , choose  $k' = 19$

|    |   |    |   |    |
|----|---|----|---|----|
| 34 | - | 19 | = | 15 |
| 19 | - | 15 | = | 4  |
| 15 | - | 4  | = | 11 |
| 11 | - | 4  | = | 7  |
| 7  | - | 4  | = | 3  |
| 4  | - | 3  | = | 1  |
| 3  | - | 1  | = | 2  |
| 2  | - | 1  | = | 1  |
| 1  | - | 1  | = | 0  |

For  $k = 34$ ,  $k' = 21$  is better (Fibonacci)

Strategy: test several  $k'$  starting with  $k' = k/\phi$

Current research: improvements, FPGA implementation for measurements (collab. UCC)

## Double-Base Number Systems(DBNS)

Contact: L. Imbert

Redundant representation based on a sum of mixed powers of 2 and 3:

$$x = \sum_{i=1}^n x_i 2^{a_i} 3^{b_i}, \text{ with } x_i \in \{-1, 1\}, a_i, b_i \geq 0$$

Example:  $127 = 108 + 16 + 3 = 72 + 54 + 1 = \dots$

|    |   |   |   |   |    |
|----|---|---|---|---|----|
|    | 1 | 2 | 4 | 8 | 16 |
| 1  |   |   |   |   | 1  |
| 3  | 1 |   |   |   |    |
| 9  |   |   |   |   |    |
| 27 |   |   | 1 |   |    |

|    |   |   |   |   |
|----|---|---|---|---|
|    | 1 | 2 | 4 | 8 |
| 1  | 1 |   |   |   |
| 3  |   |   |   |   |
| 9  |   |   |   | 1 |
| 27 |   | 1 |   |   |

## Double-Base Number Systems

Smallest  $x > 0$  requiring  $n$  terms in DBNS:

| $n$ | unsigned      | signed |
|-----|---------------|--------|
| 2   | 5             | 5      |
| 3   | 23            | 105    |
| 4   | 431           | (4985) |
| 5   | 18,431        | ?      |
| 6   | 3,448,733     |        |
| 7   | 1,441,896,119 |        |
| 8   | ?             |        |

Theorem: Every positive integer  $x$  can be represented as a sum or difference of at most  $O(\log x / \log \log x)$  terms

Example: 127 has exactly 783 DBNS representations, among which 6 are canonic:  $127 = (108 + 18 + 1) = (108 + 16 + 3) = (96 + 27 + 4) = (72 + 54 + 1) = (64 + 54 + 9) = (64 + 36 + 27)$

## Double-Base Number Systems

Application:

$$314159 = 2^4 3^9 + 2^8 3^1 - 1$$

$$[314159]P = [2^4 3^9]P + [2^8 3^1]P - P$$

cost: 12 DBL + 10 TPL + 2 ADD

$$314159 = 2^4 3^9 - 2^0 3^6 - 3^3 - 3^2 - 3 - 1$$

$$[314159]P = 3(3(3(3^3([2^4 3^3]P - P) - P) - P) - P) - P$$

cost: 4 DBL + 9 TPL + 5 ADD

Goal: expansions with  $a_1 \geq a_2 \geq \dots \geq a_n \geq 0$ ,  $b_1 \geq b_2 \geq \dots \geq b_n \geq 0$ ,

$$x = \sum_{i=1}^n x_i 2^{a_i} 3^{b_i}, \text{ with } x_i \in \{-1, 1\}$$

We compute  $[x]P$  in a Horner-like fashion (reuse partial results)

## PACE Overview

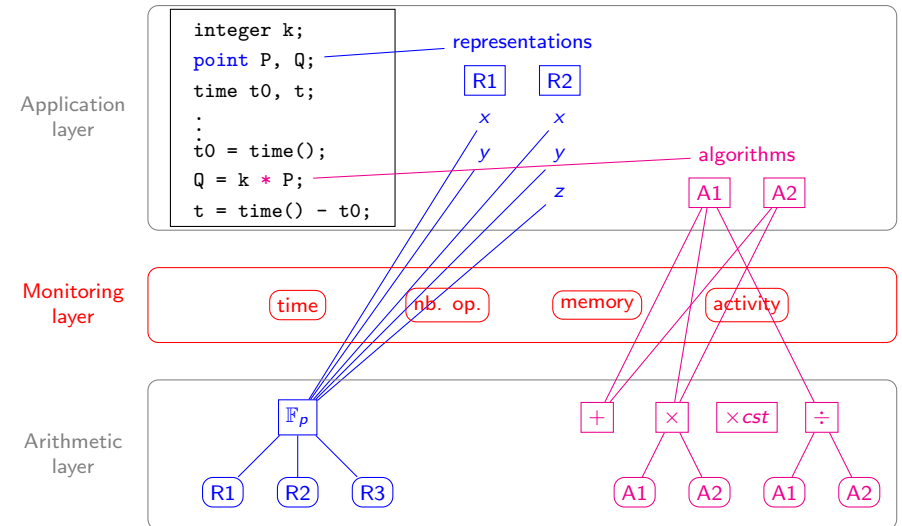
### Motivations:

- very limited mathematical support in languages and processors (“small” integers and floating-point approximation of reals)
- comparison of several solutions is difficult
  - ▶ hard to write all the tested solutions
  - ▶ needs a uniform test system for comparison accuracy
- fast validation of new algorithms/representations

### Solution: PACE library for “Prototyping Arithmetic for Crypto Easily”

- C++
- templates
  - ▶ generic software
  - ▶ specialization for performance (traits)
- GPL license
- very first version (ECC, prime fields, basic algorithms)
- authors: L. Imbert, A. Pereira, A. Tisserand

## PACE Architecture



## Arithmetic Layer

```
integer<100> p = 29;
typedef gfp<100, p> fp_29;
fp_29 x = 17, y = 20, z;
z = x + y;  assert(z == 8);
cout << x << " + " << y << " = " << z << endl;
z = x - y;  assert(z == 26);
cout << x << " - " << y << " = " << z << endl;
z = x * y;  assert(z == 21);
cout << x << " * " << y << " = " << z << endl;
z = inv(x); assert(z == 12);
cout << x << " ^(-1) = " << z << endl;
```

### produces

```
17 + 20 = 8
17 - 20 = 26
17 * 20 = 21
17 ^(-1) = 12
```

## Arithmetic Layer

- long integers
  - ▶ representations: GMP, basic, with guard bits (under dev.)
  - ▶ operations:  $\pm, \times, ^2, \div, \sqrt{\phantom{x}}, ^{-1}, mod, invmod, powmod, cmp, bit, popcount...$
- prime fields elements,  $\mathbb{F}_P$  with general  $P$ 
  - ▶ representations: GMP, basic, Montgomery (under dev.), with guard bits (under dev.)
  - ▶ operations:  $\pm, \times, ^2, ^{-1}, =, bit, popcount...$

To be added:  $\mathbb{F}_P$  with specific  $P$  values,  $\mathbb{F}_2$  and extensions, polynomials

## Monitoring Layer

- number of operations (at each level)
- time:
  - ▶ system time
  - ▶ performance counters of the processor (PAPI and Perfctr)
- memory (#words, max. size)
- number of objects (max or current count)
- other values:
  - ▶ cache misses
  - ▶ Hamming weight
- etc

## Application Layer (ECC)

```
integer<100> p = 29; typedef gfp<100, p> fp_29;
curve<fp_29> E(4, 20);
typedef point_aff<fp_29, E> point;
E.info();
point P1(5,22); point P2(16,27);
cout << "P1 = " << P1 << " P2 = " << P2 << endl;
point P3 = P1 + P2;
cout << "P1 + P2 = " << P3 << endl;
point P4 = 2 * P1;
cout << "[2] P1 = " << P4 << endl;
```

produces

```
Elliptic curve defined by  $y^2 = x^3 + 4x + 20$ 
P1 = (5 , 22) P2 = (16 , 27)
P1 + P2 = (13 , 6)
[2] P1 = (14 , 6)
```

## Validation

```
integer<100> p = 2003;
typedef gfp<100, p> fp_2003;
curve<fp_2003> E(1132,278);
typedef point_jac<fp_2003, E> point;
E.info();
point P1(1213, 408, 601);
point P2(1623, 504, 1559);
cout << "P1 = " << P1 << endl;
cout << "P2 = " << P2 << endl;
point P3 = P1 + P2;
assert(P3 == point(1683,1388));
cout << "P1 + P2 = " << P3 << endl;
point P4 = 2 * P1;
assert(P4 == point(1467,143));
cout << "[2] P1 = " << P4 << endl;
point P5 = 763 * P1;
assert(P5 == point(1455,882));
cout << "[763] P1 = " << P5 << endl;
```

```
Elliptic curve defined by  $y^2 = x^3 + 1132x + 278$ 
P1 = (1213 : 408 : 601)
P2 = (1623 : 504 : 1559)
P1 + P2 = (763 : 440 : 1934)
[2] P1 = (1800 : 1083 : 1684)
[763] P1 = (752 : 1146 : 543)
```

## Other Tools

- GMP, arbitrary precision arithmetic on signed integers, rationals and floating point numbers
- NTL, data structures and algorithms for arbitrary length integers, and for vectors, matrices, and polynomials over the integers and over finite fields
- CLN, computations with all kind of numbers, including complex numbers, and univariate polynomials in arbitrary precision
- Miracl, crypto primitives (RSA, DH, ECC, AES, SHA2, etc)
- $\text{mp}\mathbb{F}_q$ , finite fields
- ZEN, arbitrary finite field arithmetic
- CAO, "an experimental cryptography-aware domain-specific language and associated compiler system"
- Magma and Pari/GP for validation

## Future Prospects

- improvements on basic arithmetic operators
  - ▶ speed
  - ▶ area
  - ▶ robustness against side-channel attacks
- pairings
- LLL for (NTRU signature, cryptanalysis)
- arithmetic operators robust to fault injection
- FPGA implementations of demonstrators
- library for prototyping the arithmetic level of crypto applications
- add new arithmetic types and operators into design tools
- arithmetic support for light crypto

## The end, some questions ?

### Contact:

- <mailto:arnaud.tisserand@lirmm.fr>
- <http://www.lirmm.fr/~tisseran>
- Arith group
- LIRMM Laboratory, CNRS–Univ. Montpellier 2  
161 rue Ada. F-34392 Montpellier cedex 5. France

Thank you