Improving Modular Inversion in RNS using the Plus-Minus Method

Karim Bigou and Arnaud Tisserand

IRISA-CAIRN

CHES 2013: August 20–23

Context and Objectives

Research group main objective:

Design hardware implementations of cryptoprocessor for ECC (elliptic curve cryptography) on FPGA and ASIC

Various aspects of arithmetic operators for ECC:

- algorithms
- representations of numbers
- hardware implementations

This work

Modular inversion operators in the residue number system (RNS)

My Ph. D. objectives:

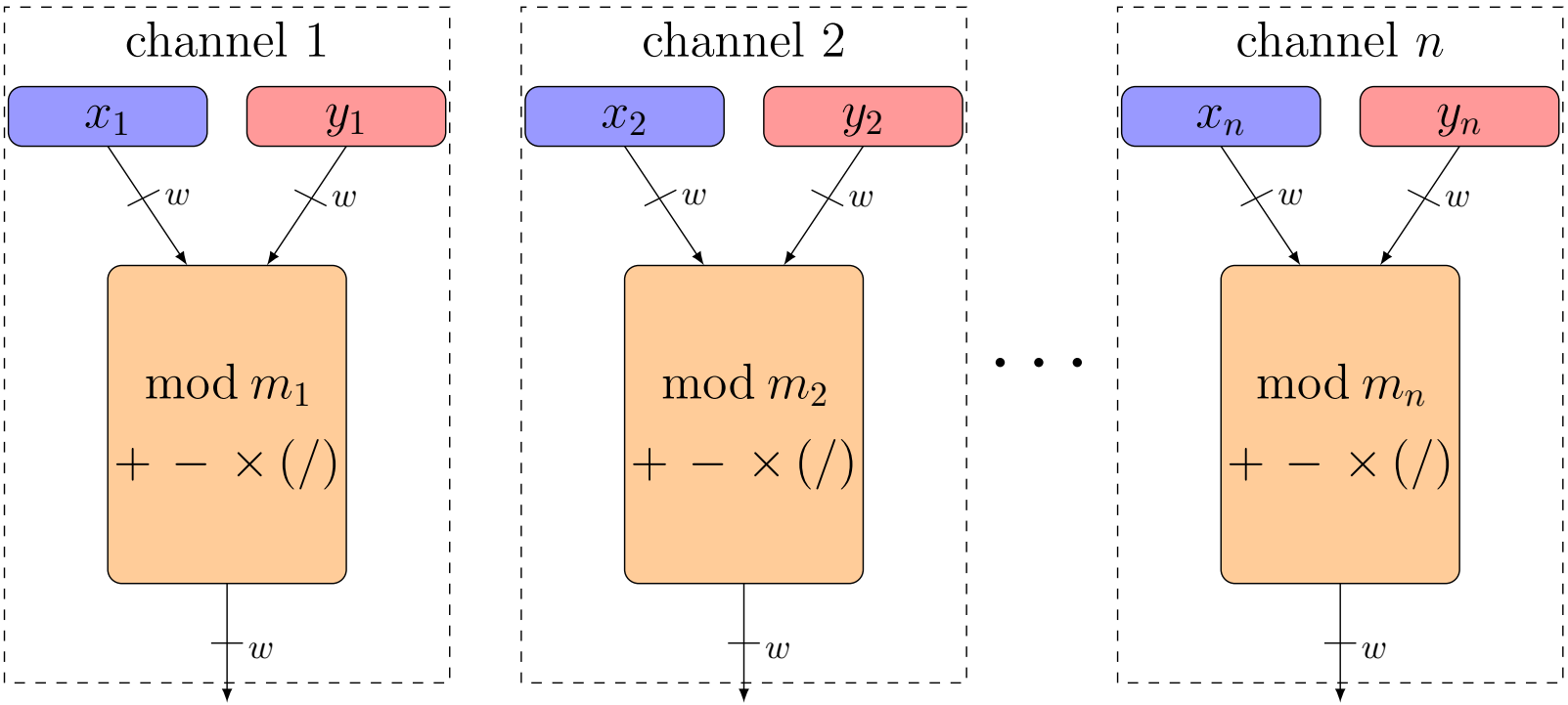
- natural parallelism $\rightarrow$ speed
- natural support for randomization $\rightarrow$ protection against some side-channel attacks (SCA)

Residue Number System (RNS) [8] [3]

X and Y two $\mathbb{F}_P$ elements (160–600 bits) are represented by:

$$\overrightarrow{X} = (x_1, \ldots, x_n) = (X \bmod m_1, \ldots, X \bmod m_n)$$
$$\overrightarrow{Y} = (y_1, \ldots, y_n) = (Y \bmod m_1, \ldots, Y \bmod m_n)$$

Modular operations over w-bit chunks, e.g. w is 16–64



RNS base $\mathcal{B} = (m_1, \ldots, m_n)$, n co-prime integers of w bits with $n \times w \geqslant \log_2 P$

RNS Properties

Pros:

- **Carry-free** between channels
  - each channel is independant
- **Fast parallel** $+$, $-$, $\times$ and some exact divisions
  - computations over all channels can be performed in parallel
  - a multiplication requires n modular multiplications of w-bit words
- **Non-positional** number system
  - randomization of computations (SCA countermeasures)

Cons:

- comparison, **modular reduction** and division are hard

Base Extension [9]

- Usual technique for modular reduction: add redundancy using 2 bases
- $\mathcal{B} = (m_1, \ldots, m_n)$ and $\mathcal{B}' = (m'_1, \ldots, m'_n)$ are coprime RNS bases
- X is $\overrightarrow{X}$ in $\mathcal{B}$ and $\overrightarrow{X}'$ in $\mathcal{B}'$
- The base extension (BE, introduced in [9]) is defined by:

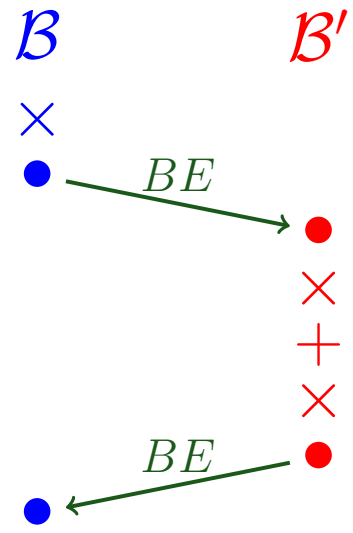
$$\overrightarrow{X}' = BE(\overrightarrow{X}, \mathcal{B}, \mathcal{B}')$$

- 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
- State-of-art BE algorithms cost $n^2 + n$ w-bit modular multiplications

RNS Montgomery Reduction (RNS-MR) [7]

Input: $\overrightarrow{X}$, $\overrightarrow{X}'$ with $X < \alpha P^2 < PM$ and $2P < M'$
Output: $(\overrightarrow{\omega}, \overrightarrow{\omega}')$ with $\omega \equiv X \times M^{-1} \bmod P$
$0 \leqslant \omega < 2P$

$\overrightarrow{Q} \longleftarrow \overrightarrow{X} \times (-\overrightarrow{P}^{-1})$ (in base $\mathcal{B}$)
$\overrightarrow{Q}' \longleftarrow BE(\overrightarrow{Q}, \mathcal{B}, \mathcal{B}')$
$\overrightarrow{S}' \longleftarrow \overrightarrow{X}' + \overrightarrow{Q}' \times \overrightarrow{P}'$ (in base $\mathcal{B}'$)
$\overrightarrow{\omega}' \longleftarrow \overrightarrow{S}' \times \overrightarrow{M}^{-1}$ (in base $\mathcal{B}'$)
$\overrightarrow{\omega} \longleftarrow BE(\overrightarrow{\omega}', \mathcal{B}', \mathcal{B})$



RNSMR cost: $2n^2 + O(n)$ w-bit modular multiplications

How to exploit RNS properties?

Maximizing the use of fully parallelizable operations, e.g. computing patterns in the form of $(AB + CD) \bmod P$

State-of-Art RNS Inversion Algorithm

State-of-art RNS modular inversion (FLT-RNS) [4, 2]:

- based on Fermat's Little Theorem (FLT): $X^{-1} = X^{P-2} \bmod P$
- requires a large exponentiation
- involves a lot of modular reductions
- parallelization is very limited due to data dependencies

FLT-RNS complexity: $O(\log_2 P \times n^2)$ multiplications of w-bit words

Binary Extended Euclidean from [6]§ 4.5.2

Input: $X, P \in \mathbb{N}, \quad P > 2$ with $\gcd(X, P) = 1$
Output: $|X^{-1}|_P$

$(U_1, U_3) \leftarrow (0, P), \quad (V_1, V_3) \leftarrow (1, X)$
while $V_3 \neq 1$ *and* $U_3 \neq 1$ **do**
  while $|V_3|_2 = 0$ **do**
    $V_3 \leftarrow \frac{V_3}{2}$
    if $|V_1|_2 = 0$ **then** $V_1 \leftarrow \frac{V_1}{2}$ **else** $V_1 \leftarrow \frac{V_1+P}{2}$
  while $|U_3|_2 = 0$ **do**
    $U_3 \leftarrow \frac{U_3}{2}$
    if $|U_1|_2 = 0$ **then** $U_1 \leftarrow \frac{U_1}{2}$ **else** $U_1 \leftarrow \frac{U_1+P}{2}$
  if $V_3 \geq U_3$ **then** $V_3 \leftarrow V_3 - U_3,\ V_1 \leftarrow V_1 - U_1$
  else $U_3 \leftarrow U_3 - V_3,\ U_1 \leftarrow U_1 - V_1$
if $V_3 = 1$ **then** **return** $|V_1|_P$ **else** **return** $|U_1|_P$

Extended Euclidean algorithms are not used due to comparison and division costs in RNS

Proposed Algorithm (PM-RNS) (1/4)

Our proposition is based on binary extended Euclidean algorithm, where comparisons are replaced by cheaper operations using Plus-Minus (PM) trick [1]:

- if X and Y are odd then $X + Y = 0 \bmod 4$ or $X - Y = 0 \bmod 4$

We only choose odd moduli:

- multiplication by $4^{-1} \leftrightarrow$ division by 4

To use Plus-Minus trick we must define a cheap $\mathrm{mod}4$

Our proposition must be efficient on the state-of-art architecture of ECC with RNS, reuses and adapts existing blocks

Proposed Algorithm (PM-RNS) (2/4)

Input: $\overrightarrow{X}$, $P > 2$ with $\gcd(X, P) = 1$

Output: $\overrightarrow{S} = \overrightarrow{|X^{-1}|_P}$, $S < 2P$

Initialisations

while $\widehat{V_3} \neq \widehat{\pm 1}$ *and* $\widehat{U_3} \neq \widehat{\pm 1}$ **do**

  while $|b_{V_3}|_2 = 0$ **do**

    if $b_{V_3} = 0$ **then** $r \leftarrow 2$ **else** $r \leftarrow 1$

    $\widehat{V_3} \leftarrow \mathrm{div2r}(\widehat{V_3}, r)$, $\widehat{V_1} \leftarrow \mathrm{div2r}(\widehat{V_1}, r)$

    $b_{V_3} \leftarrow \mathrm{mod4}(\widehat{V_3})$, $b_{V_1} \leftarrow \mathrm{mod4}(\widehat{V_1})$, $v \leftarrow v + r$

  $\widehat{V_3^*} \leftarrow \widehat{V_3}$, $\widehat{V_1^*} \leftarrow \widehat{V_1}$

  if $|b_{V_3} + b_{U_3}|_4 = 0$ **then**

    $\widehat{V_3} \leftarrow \mathrm{div2r}(\widehat{V_3} + \widehat{U_3}, 2)$, $\widehat{V_1} \leftarrow \mathrm{div2r}(\widehat{V_1} + \widehat{U_1}, 2)$

    $b_{V_3} \leftarrow \mathrm{mod4}(\widehat{V_3})$, $b_{V_1} \leftarrow \mathrm{mod4}(\widehat{V_1})$

  else

    $\widehat{V_3} \leftarrow \mathrm{div2r}(\widehat{V_3} - \widehat{U_3}, 2)$, $\widehat{V_1} \leftarrow \mathrm{div2r}(\widehat{V_1} - \widehat{U_1}, 2)$

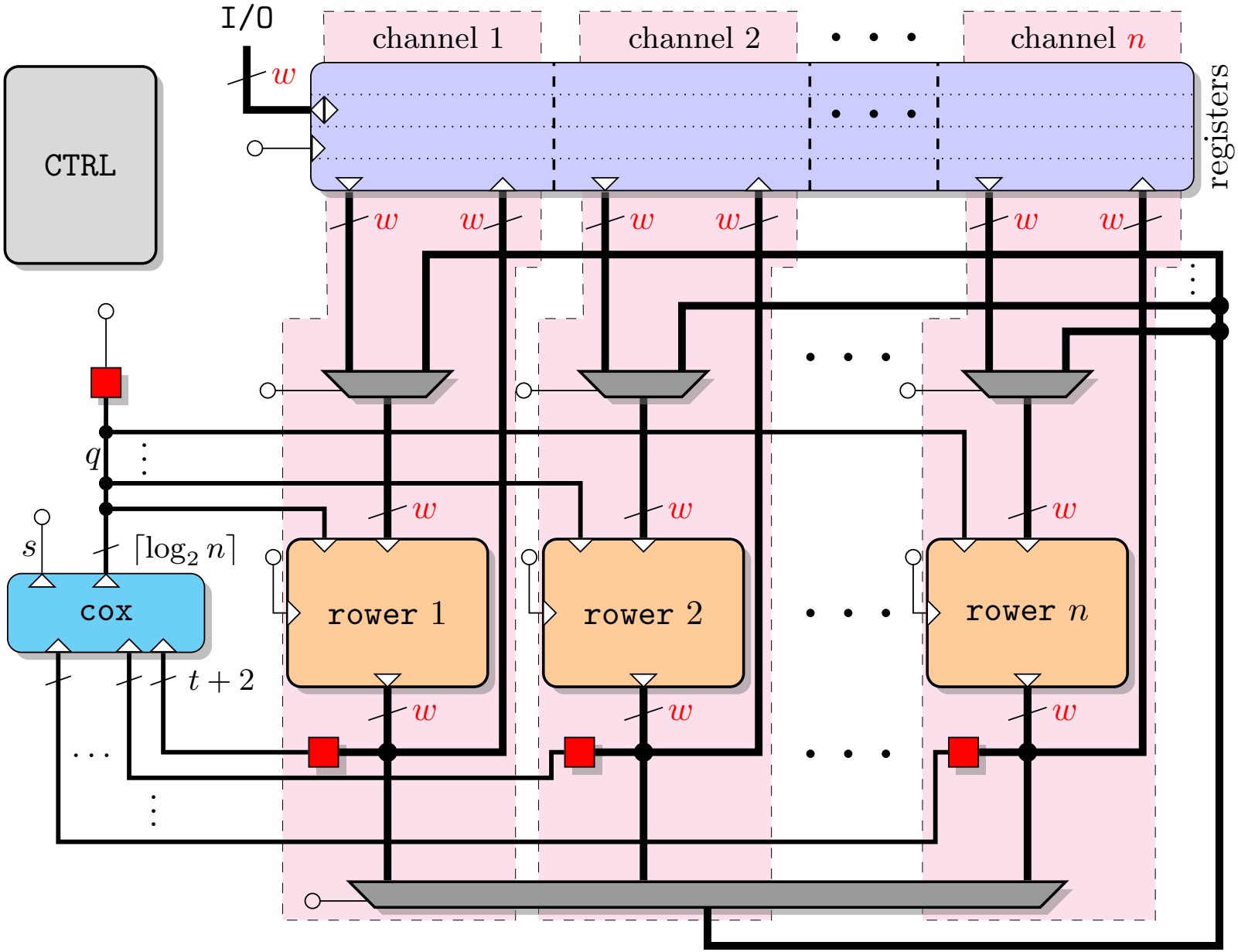
    $b_{V_3} \leftarrow \mathrm{mod4}(\widehat{V_3})$, $b_{V_1} \leftarrow \mathrm{mod4}(\widehat{V_1})$

  if $v > u$ **then** $\widehat{U_3} \leftarrow \widehat{V_3^*}$, $\widehat{U_1} \leftarrow \widehat{V_1^*}$, $u \leftrightarrow v$

  $v \leftarrow v + 1$

Final corrections

Global Architecture (adaptation of [4])



Proposed Algorithm (PM-RNS) (3/4)

To use Plus-Minus trick we must define a cheap $\mathrm{mod}4$ using CRT:

$$|X|_4 = \left|\left|\sum_{i=1}^{n} \left|x_i \cdot M_i^{-1}\right|_{m_i} \cdot M_i\right|_M\right|_4 = |s - q \cdot M|_4 = s - |q \cdot M|_4$$

with $M = \prod_{i=1}^{n} m_i$, $M_i = \frac{M}{m_i}$ and $q = \left\lfloor \frac{\sum_{i=1}^{n} \left|x_i \cdot M_i^{-1}\right|_{m_i} \cdot M_i}{M} \right\rfloor$

Kawamura's approximation [5] of q:

$$q = \left\lfloor \sum_{i=1}^{n} \frac{|x_i \cdot M_i^{-1}|_{m_i}}{m_i} \right\rfloor \approx \left\lfloor \sum_{i=1}^{n} \frac{trunc(|x_i \cdot M_i^{-1}|_{m_i})}{2^w} \right\rfloor$$

The Cox module

- computes q: sum of n values of $t = 6$ bits (MSBs)
- computes s: sum of n 2-bit values modulo 4

The main differences with [4] comes from the Cox:

- In [4], the CRT sum was performed in n cycles, here in 1
- In [4], the Cox only computes q (not s)

Proposed Algorithm (PM-RNS) (4/4)

Kawamura's approximation of q requires $0 \leqslant X < (1 - err_{max})M$, with $err_{max} \in [0, 1[$ but PM requires subtractions. For instance $X = |U - V|_M = |-1|_M = M - 1$ so $X > (1 - err_{max})M$

$\widehat{X}$ is an affine function of $\overrightarrow{X}$ defined as

$$\widehat{X} = \left((x_1 + c_1) \cdot M_1^{-1}, \ldots, (x_n + c_n) \cdot M_n^{-1}\right)_{\mathcal{B}}$$

- $\overrightarrow{C} = (c_1, \ldots, c_n)$ is 0 mod 4 : $\mathrm{mod4}(\widehat{X}) = |X|_4$
- $-P < X < P$ in PM-RNS, so if $P \leqslant C < (1 - err_{max})M - P$ then $0 \leqslant \widehat{X} < (1 - err_{max})M$
- the factor $(M_1^{-1}, \ldots, M_n^{-1})$ is included and done once for the computation of q and s

Remark: $\mathrm{div2r}$ is designed to handle properly $\widehat{X}$, i.e. it returns the hat representation of the expected output (for instance $\widehat{\left(\frac{X+P}{4}\right)}$)

Cost of the Proposed Algorithm

Our proposition:

- On average, $0.71 \log_2 P$ main loop iterations
- PM-RNS works without base extension
- Total average complexity: $O(\log_2 P \times n)$

Example: for 192 bits (number of w-bit modular multiplications):

$n \times w$	FLT-RNS	PM-RNS	Gain Factor
12×17	103140	5474	18
9×22	61884	4106	15
7×29	40110	3193	12

Remark: state-of-art FLT-RNS complexity is $O(\log_2 P \times n^2)$ multiplications of w-bit words

Hardware Implementations and Comparisons

PM-RNS and FLT-RNS have been fully implemented:

- using the same CAD tools (ISE 12.4)
- on Virtex 5 FPGAs
- with the same synthesis options and efforts
- and have an optimized implementation (for fair comparison)

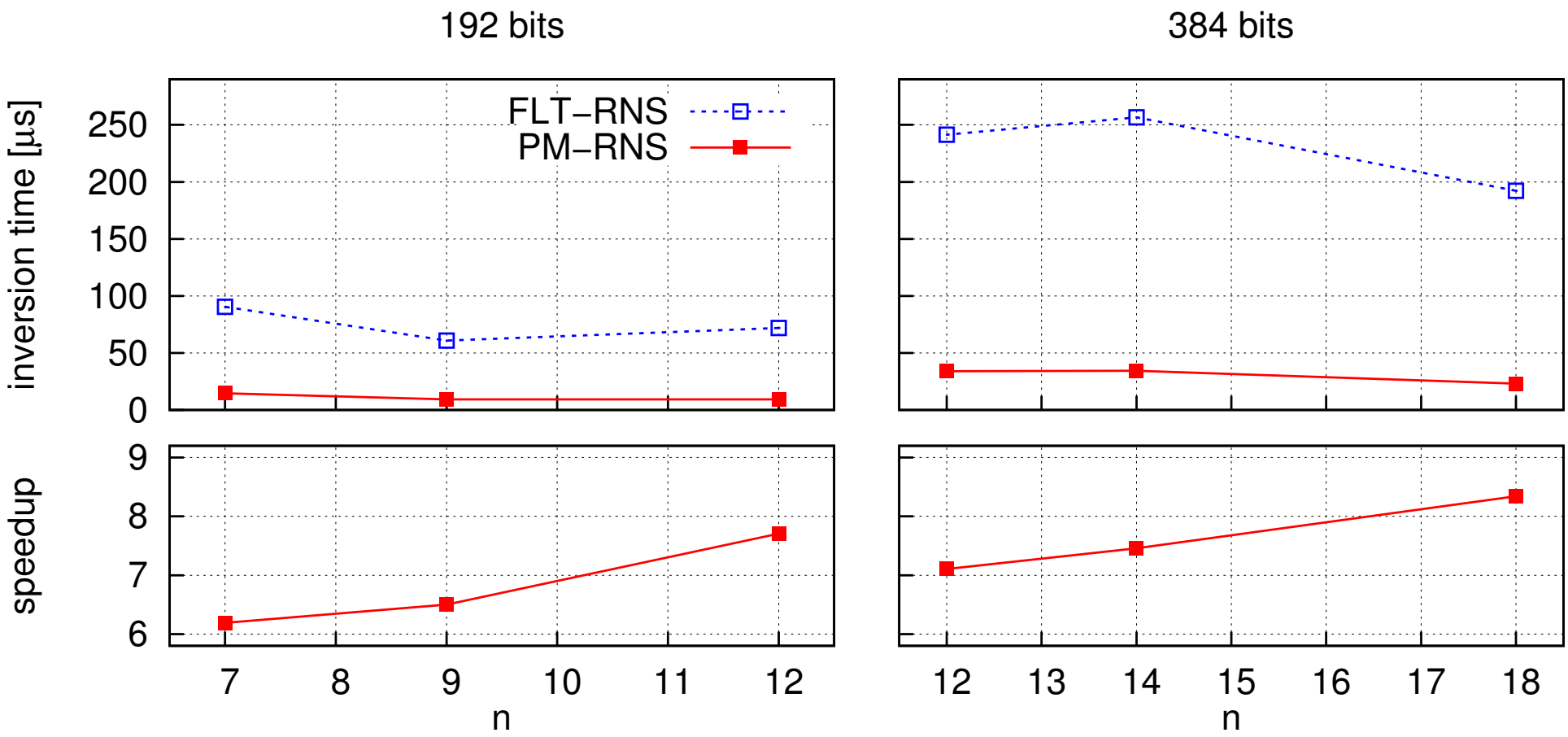
For both algorithms, 2 field sizes have been implemented:

- 192 bits
- 384 bits

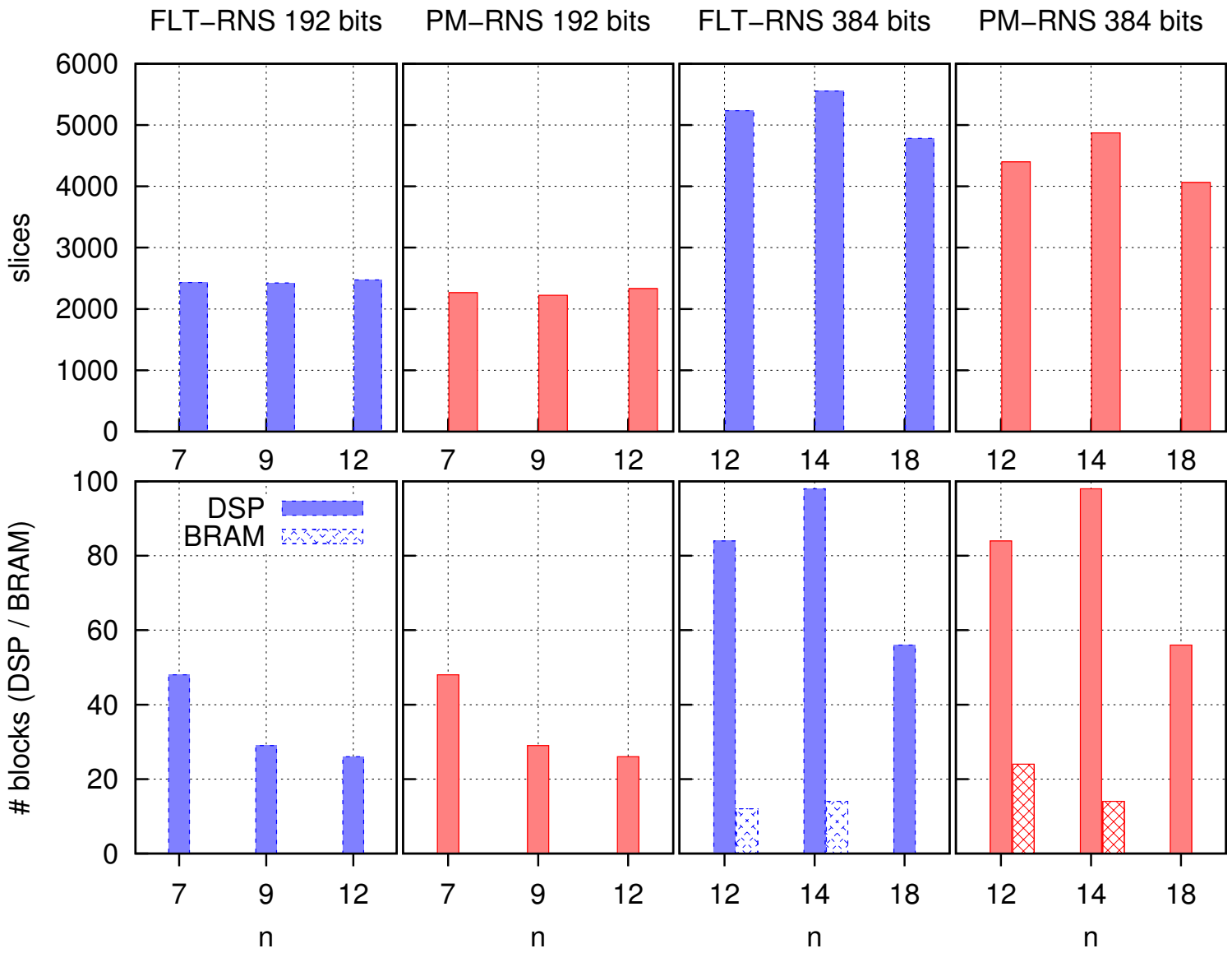
For each size, 3 couples (n,w) have been implemented

Timing Implementation Comparison

Virtex 5 results with DSP blocks and BRAMs, with $w \in \{17, 22, 29\}$ for 192 bits and $w \in \{22, 29, 33\}$ for 384 bits:



Area Implementation Results Comparison



Conclusion

Our proposition:

- is about 6–8 times faster than the best state-of-art solution
- with a small area overhead on RNS operator for ECC
- full FPGA optimization and fair comparison

Future works on hardware implementation:

- improve FPGA implementation of the new inversion
- integration in a complete ECC processor in RNS
- implementation of randomization for a scalar multiplication
- study speed vs. area trade-offs

Thank you for your attention

We thank the anonymous reviewers, Thomas Chabrier, Jérémy Métairie and Nicolas Guillermin for their valuable comments. This work has been supported in part by a PhD grant from *DGA–INRIA* and by the PAVOIS project (ANR 12 BS02 002 01).

References I

[1] R. P. Brent and H. T. Kung.
Systolic VLSI arrays for polynomial GCD computation.
IEEE Transactions on Computers, C-33(8):731–736, August 1984.

[2] F. Gandino, F. Lamberti, G. Paravati, J.-C. Bajard, and P. Montuschi.
An algorithmic and architectural study on montgomery exponentiation in RNS.
IEEE Transactions on Computers, 61(8):1071–1083, August 2012.

[3] H. L. Garner.
The residue number system.
IRE Transactions on Electronic Computers, EC-8(2):140–147, June 1959.

[4] 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, Santa Barbara, CA, USA, August 2010. Springer.

[5] S. Kawamura, M. Koike, F. Sano, and A. Shimbo.
Cox-rower architecture for fast parallel montgomery multiplication.
In *Proc. 19th International Conference on the Theory and Application of Cryptographic (EUROCRYPT)*, volume 1807 of *LNCS*, pages 523–538, Bruges, Belgium, May 2000. Springer.

References II

[6] D. E. Knuth.
Seminumerical Algorithms, volume 2 of *The Art of Computer Programming*.
Addison-Wesley, 3rd edition, 1997.

[7] K. C. Posch and R. Posch.
Modulo reduction in residue number systems.
IEEE Transactions on Parallel and Distributed Systems, 6(5):449–454, May 1995.

[8] A. Svoboda and M. Valach.
Operátorové obvody (operator circuits in czech).
Stroje na Zpracování Informací (Information Processing Machines), 3:247–296, 1955.

[9] N. S. Szabo and R. I. Tanaka.
Residue arithmetic and its applications to computer technology.
McGraw-Hill, 1967.

Costs of RNS Modular Operations

X, Y, P : n words of w bits

Elementary operations: mult. = w-bit multiplication

Operation	RNS (mult.)	Standard (mult.)
XY	$2\,n$	n^2
$X \bmod P$	$2\,n^2 + 4\,n$	$n^2 + n$
$XY \bmod P$	$2\,n^2 + 6n$	$2\,n^2 + n$
$(X_1 Y_1 + X_2 Y_2) \bmod P$	$2\,n^2 + 8\,n$	$3\,n^2 + n$
$\left(\sum_{i=1}^{k} X_i Y_i\right) \bmod P$	$2\,n^2 + (4 + 2k)n$	$(1 + k)\,n^2 + n$

Factor 2 in RNS XY is due to the use of 2 bases and BEs

Cost of Inner Loop Operations

For each main loop iteration:

- the inner loop has on average $\frac{2}{3}$ iterations
- RNS $\times$ and $\mathrm{div2r}$ cost n w-bit modular multiplications
- RNS $+, -$ cost n modular w-bit additions
- $\mathrm{mod4}$ requires n additions of t-bit values and $n+1$ modulo 4 additions

Note: t is the number of truncated bits in q approximation

Implementation of FLT-RNS

Our FLT-RNS implementation:

- is based on the state-of-art one [4] (with the original Cox)
- uses state-of-art Gandino's algorithm [2] to perform RNS exponentiation
- has a better pipeline filling than state-of-art one [4] (control dedicated to the inversion here)

Operations Count

Algo.	ℓ	$n \times w$	w-bit EMM	w-bit EMA	cox-add	mod4-add
FLT-RNS	192	12×17	103140	85950	6876	0
		9×22	61884	48991	5157	0
		7×29	40110	30083	4011	0
	384	18×22	434322	382617	20682	0
		14×29	273462	233247	16086	0
		12×33	206820	172350	13788	0
FLT-RNS NIST	192	12×17	137520	114600	9168	0
		9×22	85512	65322	6876	0
		7×29	53480	40110	5348	0
	384	18×22	579096	510156	27576	0
		14×29	364616	310996	21448	0
		12×33	275760	229800	18 384	0
PM-RNS	192	12×17	5474	8750	5474	5930
		9×22	4106	6562	4106	4562
		7×29	3193	5104	3193	3650
	384	18×22	16487	26376	16487	17402
		14×29	12823	20514	12823	13738
		12×33	10991	17584	10991	11907

FPGA Implementation Results with Dedicated Hard Blocks

Algo.	ℓ	$n \times w$	Area			Freq.	Number	Duration
			slices (FF/LUT)	DSP	BRAM	MHz	of cycles	μs
FLT-RNS	192	12×17	2473 (2995/7393)	26	0	186	13416	72.1
		9×22	2426 (3001/7150)	29	0	185	11272	60.9
		7×29	2430 (3182/6829)	48	0	107	9676	90.4
	384	18×22	4782 (5920/14043)	56	0	178	34359	193.0
		14×29	5554 (5910/16493)	98	14	110	28416	258.3
		12×33	5236 (5710/15418)	84	12	107	25911	242.1
PM-RNS	192	12×17	2332 (3371/6979)	26	0	187	1753	9.3
		9×22	2223 (3217/6706)	29	0	187	1753	9.3
		7×29	2265 (3336/6457)	48	0	120	1753	14.6
	384	18×22	4064 (5932/13600)	56	0	152	3518	23.1
		14×29	4873 (6134/14347)	98	14	102	3518	34.4
		12×33	4400 (5694/12764)	84	24	103	3518	34.1

FPGA Implementation Results without Dedicated Hard Blocks

Algo.	ℓ	$n \times w$	Area			Freq.	Number	Duration
			slices (FF/LUT)	DSP	BRAM	MHz	of cycles	μs
FLT-RNS	192	12×17	4071 (4043/12864)	4	0	128	13416	104.8
		9×22	4155 (3816/13313)	4	0	122	11272	92.3
		7×29	4575 (3952/15264)	0	0	126	9676	76.7
	384	18×22	7559 (7831/27457)	0	0	163	34359	210.7
		14×29	9393 (7818/30536)	0	0	126	28416	225.5
		12×33	9888 (7640/31599)	0	0	107	25911	242.1
PM-RNS	192	12×17	3899 (4212/12519)	4	0	150	1753	11.6
		9×22	3809 (3986/12782)	4	0	146	1753	12.0
		7×29	4341 (4107/14981)	0	0	141	1753	12.4
	384	18×22	7677 (8053/128306)	0	0	168	3518	20.9
		14×29	9119(8113/30619)	0	0	127	3518	27.7
		12×33	9780 (7908/31902)	0	0	108	3518	32.5