Plan

1. Context
2. RNS for Cryptographic Computations
3. New RNS Modular Multiplication
4. Specific Patterns for Exponentiations
One objective of our research group:
Design efficient hardware implementations of asymmetric cryptography using fast arithmetic techniques

Examples of targeted cryptosystems:
- RSA [RSA78]
- Discrete Logarithm Cryptosystems: Diffie-Hellman [DH76] (DH), ElGamal [Elg85]
- Elliptic Curve Cryptography (ECC) [Mil85] [Kob87]

The residue number system (RNS) is a representation which enables fast computations for cryptosystems requiring large integers (or $\mathbb{F}_p$ elements)
ECC Very Short Overview

$P$ large prime of 160–600 bits

Elliptic curve $E$ over $\mathbb{F}_P$:

$$y^2 = x^3 + ax + b$$

Curve level operations:
- Point addition (ADD): $Q + Q'$
- Point doubling (DBL): $Q + Q$
- Scalar multiplication:
  $$[k]Q = Q + Q + \ldots + Q$$
  \(k\) times

Security (ECDLP): knowing $Q$ and $[k]Q$, $k$ cannot be recovered

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

ECDLP : Elliptic Curve Discrete Logarithm Problem
Internal Operations of a Scalar Multiplication

One scalar multiplication requires...

Many curve level operations which require...

MANY $\mathbb{F}_p$ operations
One scalar multiplication requires...

Many curve level operations which require...

MANY $\mathbb{F}_p$ operations which can be performed in a parallel way using RNS
A large integer of $\ell$ bits ($\ell \approx 160$–4096) is represented by:

$$\overrightarrow{X} = (x_1, \ldots, x_n) = (X \mod m_1, \ldots, X \mod m_n)$$

RNS base $B = (m_1, \ldots, m_n)$, $n$ pairwise co-primes of $w$ bits, $n \times w \geq \ell$

The Chinese remainder theorem (CRT) is the base of RNS

Note: an **EMM** is a $w$-bit elementary modular multiplication (one channel)
RNS Properties

Pros:

- **Carry free** between channels
  - each channel is independent
- **Fast parallel** $+, -, \times$ and some exact divisions
  - computations over all channels can be performed in parallel
  - an RNS multiplication requires $n$ EMMs
- **Non-positional** number system
  - randomization of internal computations (SCA countermeasures)
- **Flexibility** for hardware implementations
  - the number of hardware channels and theoretical channels can be different
  - various area/time trade-offs and multi-size support

Cons:

- comparison, **modular reduction** and division are **much harder**
A Very Brief and Very Non-Exhaustive History of RNS for Cryptographic Implementations

- First motivation: parallel implementation of RSA
- Need for efficient modular reduction: [PP95, BDK98]
- Lead to RNS implementations of RSA [KKSS00, NMSK01]
- A protection based on randomization is proposed: the Leak Resistant Arithmetic (LRA) [BILT04]
- Ideas are adapted and reused for ECC and Pairings [Gui10, CDF+11, YFCV12]

Now:
- New algorithms, new selection of parameters for RNS arithmetic [GLP+12, BDE13, BT13, YFCV14, BT14]
- New protections based on RNS [Gui11, BEG13, PITM13, NP15]
- New architectures [BM14]
- New applications [BEMP14]
Base Extension [ST67]

Issue:
computing a reduction modulo a large number $P$ from the small residues

- Usual technique for modular reduction:
  Use conversions between 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 [ST67]) is defined by:

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

- 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 $n^2 + n$ EMMs
RNS Montgomery Reduction (MR) [PP95]

Input: \( \overrightarrow{X} \), \( \overrightarrow{X}' \) with \( X < \alpha P^2 < PM \) and \( 2P < M' \)

Output: \((\overrightarrow{\omega}, \overrightarrow{\omega}')\) with \( \omega \equiv X \times M^{-1} \mod P \)

\[
\begin{align*}
\overrightarrow{Q} &\leftarrow \overrightarrow{X} \times (-\overrightarrow{P}^{-1}) \\
\overrightarrow{Q}' &\leftarrow \text{BE}(\overrightarrow{Q}, B, B') \\
\overrightarrow{S}' &\leftarrow \overrightarrow{X}' + \overrightarrow{Q}' \times \overrightarrow{P}' \\
\overrightarrow{\omega}' &\leftarrow \overrightarrow{S}' \times M^{-1} \\
\overrightarrow{\omega} &\leftarrow \text{BE}(\overrightarrow{\omega}', B', B)
\end{align*}
\]

(in base \( B \))

\[
\begin{align*}
\overrightarrow{Q} &\leftarrow \overrightarrow{X} \times (-\overrightarrow{P}^{-1}) \\
\overrightarrow{Q}' &\leftarrow \text{BE}(\overrightarrow{Q}, B, B') \\
\overrightarrow{S}' &\leftarrow \overrightarrow{X}' + \overrightarrow{Q}' \times \overrightarrow{P}' \\
\overrightarrow{\omega}' &\leftarrow \overrightarrow{S}' \times M^{-1} \\
\overrightarrow{\omega} &\leftarrow \text{BE}(\overrightarrow{\omega}', B', B)
\end{align*}
\]

(in base \( B' \))

\( \alpha \) is a parameter chosen to speed up some computations, \( M > \alpha P \) and \( M' > 2 \times P \)

MR cost: \( 2n^2 + O(n) \) EMMs
Typical RNS Computation Flow

\[ \pm \times \text{ over one channel over one RNS vector (i.e. } n \text{ channels)} \]

base extension modulo \( P \) in RNS

\[ \pm \times \text{ over one RNS vector} \]

\[ \pm \times \text{ modulo } P \text{ in RNS} \]
Cox-Rower RNS Architecture [KKSS00, Gui10]
Some Implementation Results of 256-bit ECC on FPGA

<table>
<thead>
<tr>
<th>ref.</th>
<th>[GP08]</th>
<th>[MLPJ13]</th>
<th>[Gui10] (RNS)</th>
<th>[BM14] (RNS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>prime</td>
<td>NIST</td>
<td>Any</td>
<td>Any</td>
<td>Any</td>
</tr>
<tr>
<td>FPGA</td>
<td>Virtex 4</td>
<td>Virtex 4</td>
<td>Virtex 5</td>
<td>Stratix II</td>
</tr>
<tr>
<td># Slices</td>
<td>1715</td>
<td>4655</td>
<td>1725</td>
<td>9177*</td>
</tr>
<tr>
<td># DSPs</td>
<td>32</td>
<td>37</td>
<td>37</td>
<td>96*</td>
</tr>
<tr>
<td>Freq. MHz</td>
<td>490/245**</td>
<td>250</td>
<td>291</td>
<td>157</td>
</tr>
<tr>
<td>time ms</td>
<td>0.62</td>
<td>0.44</td>
<td><strong>0.38</strong></td>
<td>0.68</td>
</tr>
<tr>
<td>([k]P) Algo.</td>
<td>DBL &amp; ADD</td>
<td>Möller [Mö01]</td>
<td>Mont. ladd. [JY02]</td>
<td></td>
</tr>
</tbody>
</table>

*: Stratix II FPGA is counted in ALM instead of Slices and 9 × 9 multiplier instead of Xilinx DSP (18 × 25)

**: [GP08] uses 2 clock domains: 490 MHz for arithmetic and 245 MHz for control
RNS as a SCA protection

Protections based on randomization

- [CNPQ03] proposes to randomly choose the 2 RNS bases in a large set of moduli (e.g. 2 bases of 9 moduli in a set of 69)

- [BILT04] introduces the **Leak Resistant Arithmetic (LRA)**:
  - at the beginning both bases are chosen randomly from $2n$ moduli (i.e once)
  - Very costly if used at each MR

- [Gui11] adapts LRA to the Kawamura et al. base extension

- [PITM13] implements LRA and an initial base permutation against EMA attacks

- [NP15] implements a trade-off in LRA usable for each MR

Fault detection using redundancy, e.g. [WH66, Man72, YL73, CNPQ03] and recently adapted to cryptographic implementations [Gui11, BEG13]
Two main ideas to reduce the impact of modular reductions:

- **Reduce the cost** of modular reduction in specific contexts, for instance:
  - rearranging computations in an ECC context [Gui10]
  - rearranging computations in RSA exponentiation context [GLP+12]
  - our proposed modular multiplication algorithms [BT15, BT14] and new exponentiation algorithms for discrete logarithm and RSA

- **Reduce the number** of modular reductions, for instance:
  - computing pattern of the form $AB + CD \mod P$ in ECC formulas [BDE13]
  - our proposed modular inversion algorithm PM-MI in an ECC context [BT13]
New RNS Modular Multiplication
Improving Modular Multiplication

RNS modular multiplication **MM is the most costly operation** in RNS cryptographic applications (ECC, RSA, DL)

Two different multiplications:
- simple RNS multiplication: \( n \) EMMs
- **MM** = simple RNS multiplication + MR: \( 2n^2 + O(n) \) EMMs

Our idea: **modify RNS** to add some positional information

Let us assume \( \mathcal{B}_a \) with \( \frac{n}{2} \) moduli of \( w \) bits (\( \log_2 P \approx n \times w \), \( \mathcal{B}_a \) is a “half base”), then \( (\overrightarrow{K_x}, \overrightarrow{R_x}) \) represents:

\[
\overrightarrow{X} = \overrightarrow{K_x} \overrightarrow{M_a} + \overrightarrow{R_x}
\]

where \( \overrightarrow{M_a} = \prod_{i=1}^{n_a} m_{a,i} \)

Note: \( \overrightarrow{K_x} \) and \( \overrightarrow{R_x} \) are \( \frac{\log_2 P}{2} \) bits long
Decomposition with Split Algorithm

Input: $X_{a|b}$

Precomp.: $(M_a^{-1})_{b}$

Output: $(K_x)_{a|b}$, $(R_x)_{a|b}$ with $X_{a|b} = (K_x)_{a|b} \times (M_a)_{a|b} + (R_x)_{a|b}$

$(R_x)_b \leftarrow \text{BE} \left( (R_x)_a, B_a, B_b \right)$

$(K_x)_b \leftarrow \left( X_b - (R_x)_b \right) \times (M_a^{-1})_b$

if $(K_x)_b = -1$ then

$(K_x)_b \leftarrow 0$ /* Kawamura BE correction */

$(R_x)_b \leftarrow (R_x)_b - (M_a)_b$

$(K_x)_a \leftarrow \text{BE} \left( (K_x)_b, B_b, B_a \right)$

return $(K_x)_{a|b}$, $(R_x)_{a|b}$

Note: the cost of Split is dominated by the 2 BEs (on half bases):

$\frac{n^2}{2} + O(n)$ when $n_a = n_b = n/2$
SBMM (Single Base Modular Multiplication) idea:

- $X$ is represented by $(K_x, R_x)$
- $P = M_a^2 - 2$ with $P$ prime and $M_a$ odd

Some remarks

- $P$ is an equivalent for RNS to pseudo-Mersenne numbers for the radix 2 standard representation (for instance $P = 2^{521} - 1$)
- $P = M_a^2 - 1$ is never prime
- One can find a lot of $P$ for a given size (probabilistic primality tests using `isprime` from Maple, for instance generating 10 000 $P$ of 512 bits in 15 s.)
Classical RNS \( \text{MM} \) principle

\[ B_{a|b}, B_{c|d} : \text{full RNS bases} \]
\[ B_a, B_b, B_c, B_d : \text{half bases} \]

\[ Z = \left| XY \right|_P \]

\( 2n \) EMMs

\( 2n^2 + O(n) \) EMMs
SBMM Principle 1/2

\[ XY \equiv 2K_xK_y + (K_xR_y + K_yR_x)M_a + R_xR_y \equiv U + VM_a \mod P \]
\[ XY \equiv U + V M_a \equiv (K_u + R_v)M_a + (R_u + 2K_v) \equiv K_z M_a + R_z \mod P \]

<table>
<thead>
<tr>
<th>2K_xK_y</th>
<th>R_xK_y</th>
<th>R_xR_y</th>
<th>R_xK_y</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ + + + +</td>
<td>+ + + + +</td>
<td>+ + + + +</td>
<td>+ + + + +</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>U</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSplit</td>
<td>CSplit</td>
</tr>
</tbody>
</table>

\[ 2\left(2\left(\frac{n}{2}\right)^2 + O(n)\right) = n^2 + O(n) \text{ EMMs} \]

\[ = K_z \]

\[ = R_z \]
**SBMM Algorithm**

**Parameters:** $B_a$ such that $M_a^2 = P + 2$ and $B_b$ such that $M_b > 6M_a$

**Input:** $(K_x)_{a|b}, (R_x)_{a|b}, (K_y)_{a|b}, (R_y)_{a|b}$ with $K_x, R_x, K_y, R_y < M_a$

**Output:** $(K_z)_{a|b}, (R_z)_{a|b}$ with $K_z < 5M_a$ and $R_z < 6M_a$

\[
\begin{align*}
U_{a|b} &\leftarrow 2K_xK_y + R_xR_y \\
V_{a|b} &\leftarrow K_xR_y + R_xK_y
\end{align*}
\]

\[
\begin{align*}
(K_u)_{a|b}, (R_u)_{a|b} &\leftarrow \text{Split}(U_{a|b}) \\
(K_v)_{a|b}, (R_v)_{a|b} &\leftarrow \text{Split}(V_{a|b})
\end{align*}
\]

\[
\begin{align*}
(K_z)_{a|b}, (R_z)_{a|b} &\leftarrow (K_u + R_v)_{a|b}, (2 \cdot K_v + R_u)_{a|b}
\end{align*}
\]

return $(K_z)_{a|b}, (R_z)_{a|b}$

$M_b$ is a few bits larger than $M_a$ because outputs $K_z$ and $R_z$ are larger than inputs $K_x, K_y, R_x, R_y$
SBMM Algorithm

Using an extra modulo $m_\gamma$ in $\mathcal{B}_b$:

- one can have $M_b > 6M_a$
- it enables to compress output values from SBMM
- it can be chosen small (e.g. $m_\gamma = 2^6$)

<table>
<thead>
<tr>
<th>Algo.</th>
<th>MM [GLP+12]</th>
<th>SBMM</th>
<th>SBMM + Compress</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMM</td>
<td>$2n^2 + 4n$</td>
<td>$n^2 + 5n$</td>
<td>($n^2 + 7n$) EMM + $(n + 2)$ GMM</td>
</tr>
<tr>
<td>Precomp. EMW</td>
<td>$2n^2 + 10n$</td>
<td>$\frac{n^2}{2} + 3n$</td>
<td>$\frac{n^2}{2} + 4n + 2$</td>
</tr>
</tbody>
</table>

EMM is a $w$-bit modular multiplication
GMM is a one multiplication modulo $m_\gamma$ (6 bits in practice)
EMW is a $w$-bit word stored as a precomputation
FPGA implementations:

- **MM** and **SBMM** have been implemented
- **n Rowers** for MM and **n/2 Rowers** for SBMM
- **3 field lengths** implemented: 192, 384 and 512 bits
- \( w = 16 \) bits for 192 and 32 for 384 and 512
- on various FPGAs
  - high performance Virtex 5 (LX220)
  - low cost Spartan 6 (LX45/LX100)
  - recent mid-range Kintex 7 (70T)
- (parallel) compression not implemented yet
SBMM Architecture with $n/2$ Rowers

channel 1

$\begin{align*}
\text{CTRL} & \quad \text{cox} \\
& \quad \text{rower 1} \\
& \quad \text{rower 2} \\
& \quad \text{...} \\
& \quad \text{rower } \frac{n}{2} \\
& \quad \text{rower } \frac{n}{2} + 1 \\
& \quad \text{Output}
\end{align*}$

channel 2

$\begin{align*}
& \quad \text{rower 1} \\
& \quad \text{rower 2} \\
& \quad \text{...} \\
& \quad \text{rower } \frac{n}{2} \\
& \quad \text{rower } \frac{n}{2} + 1 \\
& \quad \text{Output}
\end{align*}$

channel $\frac{n}{2}$

$\begin{align*}
& \quad \text{rower 1} \\
& \quad \text{rower 2} \\
& \quad \text{...} \\
& \quad \text{rower } \frac{n}{2} \\
& \quad \text{rower } \frac{n}{2} + 1 \\
& \quad \text{Output}
\end{align*}$

channel $\frac{n}{2} + 1$

$\begin{align*}
& \quad \text{rower 1} \\
& \quad \text{rower 2} \\
& \quad \text{...} \\
& \quad \text{rower } \frac{n}{2} \\
& \quad \text{rower } \frac{n}{2} + 1 \\
& \quad \text{Output}
\end{align*}$
FPGA Implementation Results

Reduction in Slices
(e.g. 0.4 is -40%)

Reduction in DSP blocks
Timing results for a single modular multiplication with (top) and without (bottom) DSP blocks

Spartan 6

Virtex 5

Kintex 7

Karim Bigou
RNS for Asymmetric Cryptography
May 29, 2015 28 / 40
Conclusion on SBMM

Theoretical conclusions:

- # EMMs / 2
- # precomputations / 4
- # moduli / 2
- the architecture is still flexible

First implementations conclusions:

- the area is almost divided by 2 for a small time overhead (< 10 %)

Further implementation works:

- n Rowers for SBMM (full parallel implementation)
- integration in a full scalar multiplication

This work will be presented at CHES 2015 (September in Saint-Malo)
Specific Patterns for Exponentiations
Idea

Goal: accelerate some specific, but usual, computation patterns which uses RNS modular multiplications

Examples:
- modular squares
- modular multiplication by constants
- more complex patterns with operands reuse

In state-of-the-art, RNS does not support accelerations for these patterns (except accelerations inside channels)
A Specific Fast Pattern

The cost of some patterns can be reduced without constraint on the field characteristic, for instance in the following algorithm [Gor98]:

**Input:** \( k = (k_{\ell-1}, \ldots, k_1, k_0)_2, G \in \mathbb{Z}/P\mathbb{Z} \)

**Output:** \( G^k \mod P \)

\[ S \leftarrow 1 \]

for \( i \) from \( \ell - 1 \) to 0 do

\[ S \leftarrow S^2 \mod P \]

if \( k_i = 1 \) then \( S \leftarrow S \cdot G \mod P \)

return \( S \)

One can observe:

\[ S^2 G \equiv (K_s^2 M_a^2 + 2K_s R_s M_a + R_s^2) G \mod P \]

\[ \equiv K_s^2 |M_a^2 G|_P + K_s R_s |2M_a G|_P + R_s^2 |G|_P \mod P \]

\[ \equiv K_s (K_s |M_a^2 G|_P + R_s |2M_a G|_P) + R_s^2 |G|_P \mod P \]
A Specific Fast Pattern

Values $|M^2_a G|_P$, $|2M_a G|_P$ and $|G|_P$ can be precomputed.

We choose $B_a$ with $n/2$ moduli of $w$ bits then $K_s$ and $R_s$ are $\ell/2$-bit values (i.e. the same size as $\sqrt{P}$).

If $U_2 = K_s (K_s |M^2_a G|_P + R_s |2M_a G|_P) + R_s^2 |G|_P$ then $\log_2 U_2 \approx 2\ell$ i.e. $U_2$ is a partially reduced value.

Finally, we use the state-of-the-art MR to finish the modular reduction.

The total cost of $|S^2 G|_P$:

- the Split mainly costs $n^2$ EMMs . . .
- and the final MR mainly costs $2n^2$ EMMs . . .
- leading to $3n^2$ EMMs

The same pattern is computed with $4n^2$ EMMs in state-of-the-art of RNS.
Exponentiation Algorithm

Average values for 2 bits of key (one 1 and one 0):

<table>
<thead>
<tr>
<th>Algo.</th>
<th>EMM</th>
<th>EMW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our algorithm</td>
<td>$5n^2 + 17n$</td>
<td>$3n^2 + 20n$</td>
</tr>
<tr>
<td>RNS-ME [GLP$^+$12]</td>
<td>$6n^2 + 12n$</td>
<td>$2n^2 + 10n$</td>
</tr>
<tr>
<td>Our algorithm (regular)</td>
<td>$6n^2 + 26n$</td>
<td>$3n^2 + 26n$</td>
</tr>
<tr>
<td>Regular RNS-ME [GLP$^+$12]</td>
<td>$8n^2 + 16n$</td>
<td>$2n^2 + 10n$</td>
</tr>
</tbody>
</table>

![Graph showing comparison between different algorithms](attachment:graph.png)
Conclusion on Fast Patterns

Our proposed modular exponentiation:

- reduces the number of EMMs up to 15% for the non regular algorithm
- reduces the number of EMMs up to 22% for the regular version
- can be easily adapted into a windowed version

Future works:

- implementations of the propositions in full cryptosystems
- time×area trade-off explorations
- analysis of other patterns
- analysis of the use of this pattern in other cryptosystems (e.g. ECC)
Proposition SPRR (presented at ASAP 2014):
Combines Split and MR on reduced bases
- gain in EMMs depends on the reuse of operands in operation sequences (up to 10% less EMMs)
- gain in precomputations of 25%
- works for discrete logarithm and ECC

Proposition PM–MI (presented at CHES 2013):
Adapts the binary extended Euclidean algorithm for RNS using the plus-minus trick
- it does not require BE
- it significantly reduces the number of EMM: # EMMs divided by 10–20
- PM–MI and state-of-art algorithm have been implemented on FPGA
  - PM–MI is 5–12 times faster
  - with a small area overhead on RNS operator for ECC
Conclusion
Conclusion of our contributions

Objective for a full RNS ECC implementation:

Several aspects of our propositions still have to be studied:

- a complete ECC cryptoprocessor in RNS implementation
- flexibility of the Cox-Rower architecture
- compatibility with the countermeasures based on RNS
RNS is interesting thanks to several natural properties (e.g. parallelism, randomization)

the relative costs between the different operations are not the same in RNS compared to the usual binary system
  - We have to count differently: e.g. in [BDE13] one has point ADD faster than DBL!

there is a lot of lines of research to improve the use of RNS for cryptographic applications
  - choice of parameters (e.g. moduli, curve parameters . . . )
  - new algorithms
  - new architectures
Thank you for your attention
Combining leak-resistant arithmetic for elliptic curves defined over \( \mathbb{F}_p \) and RNS representation.

An RNS Montgomery modular multiplication algorithm.

Fault detection in RNS Montgomery modular multiplication.

Babaï round-off CVP method in RNS: Application to lattice based cryptographic protocols.
Leak resistant arithmetic.

Double level Montgomery Cox-Rower architecture, new bounds.

Improving modular inversion in RNS using the plus-minus method.

RNS modular multiplication through reduced base extensions.
Single base modular multiplication for efficient hardware rns implementations of ecc. 
In Proc. 17th Cryptographic Hardware and Embedded Systems (CHES), LNCS. 

FPGA implementation of pairings using residue number system and lazy reduction. 

Parallel FPGA implementation of RSA with residue number systems - can side-channel threats be avoided? 

New directions in cryptography. 

A public key cryptosystem and a signature scheme based on discrete logarithms. 
The residue number system.  

An algorithmic and architectural study on Montgomery exponentiation in RNS.  

A survey of fast exponentiation methods.  

Ultra high performance ECC over NIST primes on commercial FPGAs.  

A high speed coprocessor for elliptic curve scalar multiplications over $\mathbb{F}_p$.  


References VI


Electromagnetic analysis on RSA algorithm based on RNS.

Modulo reduction in residue number systems.

A method for obtaining digital signatures and public-key cryptosystems.

Residue arithmetic and its applications to computer technology.

Operátorové obvody (operator circuits in czech).
Self-checked computation using residue arithmetic.  

Faster pairing coprocessor architecture.  

Novel RNS parameter selection for fast modular multiplication.  

Error correction in redundant residue number systems.  