Hierarchical Approach in RNS Base Extension for Asymmetric Cryptography

Libey Djath\textsuperscript{1}, Karim Bigou\textsuperscript{1}, Arnaud Tisserand\textsuperscript{2}

\textsuperscript{1} Université de Bretagne Occidentale / Lab-STICC, UMR CNRS 6285
\textsuperscript{2} CNRS / Lab-STICC, UMR 6285

ARITH-26, 10-12 June 2019, Kyoto, Japan
1. Context

2. Hierarchical RNS Base Extension

3. Hardware Implementation

4. Conclusion
Asymmetric cryptography serves in:

- digital signature
- authentication
- secret key exchange

An example of asymmetric cryptosystem:

**Elliptic Curve Cryptography (ECC)** [Mil85, Kob87]

For ECC, computations are performed in $GF(P)$ with $P$ a 200 – 500 bits prime

1 ECC primitive requires a thousand of additions, subtractions and **multiplications modulo** $P$
Residue Number System (RNS)

RNS
- non-positional representation system
- Chinese Remainder Theorem (CRT)
- $X$ is represented by its residues over a base
- representation with internal parallelism

RNS base
An RNS base $\mathcal{A}$ is a tuple $(a_1, a_2, ..., a_n)$ of coprime integers named moduli

Representing the number $X$
$$\overrightarrow{X} = (X \mod a_1, \ X \mod a_2, \ldots, \ X \mod a_n)$$
$$\overrightarrow{X} = (x_{a_1}, \ x_{a_2}, \ldots, \ x_{a_n})$$

Converting back to positional representation
Compute the CRT over all the $x_{a_i}$s in base $\mathcal{A}$
In hardware implementations of asymmetric cryptosystems:

- **large integers** are splitted in **small residues** (typically 16-64 bits integers)
- computations on **large integers** are replaced by parallel computations on **small residues**

\[
\begin{align*}
\text{channel 1} & \quad \pm \times \mod a_1 \\ 
\text{channel 2} & \quad \pm \times \mod a_2 \\ 
\text{channel n} & \quad \pm \times \mod a_n \\
\end{align*}
\]

\[a_i\] are pseudo Mersenne for efficiency purpose
Main advantages of RNS architectures:

- carry free operations among the channels
- fast parallel $+, -, \times$
- random order internal computations

Drawback:

- Comparison, division and mod $P$ reduction are difficult
RNS Montgomery mod $P$ Reduction \([PP95]\)

**Input:** $X_A$, $X_B$

**Precomp.:** $P_A$, $P_B$, $(-P^{-1})_A$, $(A^{-1})_B$

**Output:** $S_A$ and $S_B$ with $S = (XA^{-1}) \mod P + \delta P$ and $\delta \in \{0, 1, 2\}$

1. $Q_A \leftarrow X_A \times (-P^{-1})_A$
2. $Q_B \leftarrow BE(Q_A, A, B)$
3. $R_B \leftarrow X_B + Q_B \times P_B$
4. $S_B \leftarrow R_B \times (A^{-1})_B$
5. $S_A \leftarrow BE(S_B, B, A)$
6. return $(S_A, S_B)$

---

**Algorithm 2:**

**Chinese Remainder Theorem (CRT) formula**

$$X = \left\lfloor \sum_{i=1}^{n} x_{a_i} \times \left( \frac{A}{a_i} \right)^{-1} \left| a_i \times \frac{A}{a_i} \right|_A \right\rfloor = \left( \sum_{i=1}^{n} x_{a_i} \times \left( \frac{A}{a_i} \right)^{-1} \left| a_i \times \frac{A}{a_i} \right) - hA \right.$$  

with $A = a_1 \times \ldots \times a_n$
Base Extension (BE) [KKSS00]

BE converts $X$ in base $\mathcal{A}$ into $X$ in base $\mathcal{B}$
BE algorithm from [KKSS00]

Input: $X_A$, $\sigma = 0$ or $0.5$
Precomp.: $T_{a_i} \forall i \in [1, n]$
Output: $X_B$

for $i$ from 1 to $n$ parallel do
  $\hat{x}_{a_i} \leftarrow |x_{a_i} \times T_{a_i}|_{a_i}$
for $i$ from 1 to $n$ do
  $\sigma \leftarrow \sigma + \frac{\text{trunc}(\hat{x}_{a_i})}{2^w}$
  $h_i \leftarrow \lfloor \sigma \rfloor$
  $\sigma \leftarrow \sigma - h_i$
for $k$ from 1 to $n$ parallel do
  $x_{b_k} \leftarrow x_{b_k} + \hat{x}_{a_i} \times \lfloor \frac{A_{a_i}}{b_k} \rfloor + h_i A_{b_k}$

State of the art solution is usually called KBE

Cox-rower architecture from [Gui10]
Contents

1. Context

2. Hierarchical RNS Base Extension

3. Hardware Implementation

4. Conclusion
Idea of Hierarchical Base Extension (HBE)

Changing the notation

\[ \mathcal{A} = (a_1, \ldots, a_n) \]

with \( n = r \times c \)

Main Idea

- **gather** residues by row \((c \text{ residues per row})\) into super-residues in base \(\mathcal{A}\) by computing their partial CRTs
- compute the CRT of the super-residues of base \(\mathcal{A}\) in base \(\mathcal{B}\)
Rewriting the KBE Algorithm

1D KBE

Input: \( X_A, \sigma = 0 \text{ or } 0.5 \)
Precomp.: \( T_{a_i} \forall i \in [1, n] \)
Output: \( X_B \)

1 for \( i \) from 1 to \( n \) parallel do
2 \( \tilde{x}_{a_i} \leftarrow |x_{a_i} \times T_{a_i}|_{a_i} \)
3 for \( i \) from 1 to \( n \) do
4 \( \sigma \leftarrow \sigma + \frac{\text{trunc}(\tilde{x}_{a_i})}{2^w} \)
5 \( h_i \leftarrow \lfloor \sigma \rfloor \)
6 \( \sigma \leftarrow \sigma - h_i \)
7 for \( k \) from 1 to \( n \) parallel do
8 \( x_{b_k} \leftarrow x_{b_k} + \tilde{x}_{a_i} \times \left| \frac{A}{a_i} \right|_{b_k} + \left| -h_i A \right|_{b_k} \)

Main cost: \( n^2 \) executions of line 8

With \( n = r \times c \), main cost:
\( r^2c^2 \) executions of line 11
HBE \((c = 2)\)
Comparison between KBE and HBE

KBE

| Input: $X_A$, $\sigma = 0$ or 0.5 |
| Precomp.: $T_{a_i,j}$ $\forall i \in [1,r]$ and $\forall j \in [1,c]$ |
| Output: $X_B$ |

for $i$ from 1 to $r$ parallel do

for $j$ from 1 to $c$ parallel do

$\widehat{x}_{a_i,j} \leftarrow x_{a_i,j} \times T_{a_i,j}|_{a_i,j}$

for $i$ from 1 to $r$ do

for $j$ from 1 to $c$ do

$\sigma \leftarrow \sigma + \frac{\text{trunc}(\widehat{x}_{a_i,j})}{2^w}$

$h_{i,j} \leftarrow \lceil \sigma \rceil$

$\sigma \leftarrow \sigma - h_{i,j}$

for $k$ from 1 to $r$ parallel do

for $l$ from 1 to $c$ parallel do

$x_{b_{k,l}} \leftarrow x_{b_{k,l}} + \widehat{x}_{a_{i,j}} \times \left\lfloor \frac{A_{a_{i,j}}}{b_{k,l}} \right\rfloor + \lceil -h_{i,j} A_{b_{k,l}} \rceil_{b_{k,l}}$

Main cost: $r^2c^2$ executions of line 11

HBE

| Input: $X_A$, $\sigma = 0$ or 0.5 |
| Precomp.: $T_{a_i,j}$ $\forall i \in [1,r]$ and $\forall j \in [1,c]$ |
| Output: $X_B$ |

for $i$ from 1 to $r$ parallel do

for $j$ from 1 to $c$ parallel do

$\widehat{x}_{a_{i,j}} \leftarrow x_{a_{i,j}} \times T_{a_{i,j}}|_{a_{i,j}}$

for $i$ from 1 to $r$ do

for $j$ from 1 to $c$ do

$\sigma \leftarrow \sigma + \frac{\text{trunc}(\widehat{X}_{A_i})}{2^w \times c}$

$h_i \leftarrow \lceil \sigma \rceil$

$\sigma \leftarrow \sigma - h_i$

for $k$ from 1 to $r$ parallel do

for $l$ from 1 to $c$ parallel do

$\widehat{x}_{b_{k,l,i}} \leftarrow \widehat{X}_{A_i}|_{b_{k,l}}$

$x_{b_{k,l}} \leftarrow x_{b_{k,l}} + \widehat{x}_{b_{k,l,i}} \times \overline{A_i} + \lceil -h_i A_{b_{k,l}} \rceil_{b_{k,l}}$

Main cost: $r^2c$ executions of line 15
Theoretical Cost Comparison for \( c = 2 \)

Notation:
- \( CMM(w, w) \) for a \((w \times w \mod w)\)-bit modular multiplication
- \( CMR(w', w) \) for a \((w' \mod w)\)-bit modular reduction

KBE cost: \( n^2 CMM(w, w) + n CMM(w, w) \)
HBE cost: \( \frac{n^2}{2} CMM(w, w) + \frac{n^2}{2} CMR(2w + 1, w) + 2n CMM(w, w) \)

Theoretical cost ratio for one BE for various base sizes \((n)\)
Cox-rower architecture for KBE [Gui10]

Proposed architecture for HBE ($c = 2$)
Hardware Implementation

Target FPGA
ZYNQ-7 ZC702 from Xilinx (ZedBoard xc7z020clg484-1)

Tool
Vivado HLS (version 2017.4) from Xilinx

Implementation
- \( P \) size = 256, 384 bits
- \( w = 17, 20, 24, 28 \) bits

Optimization
Both algorithms, KBE and HBE (\( c = 2 \)) are implemented:
- same manner
- same optimization effort
Hardware Implementation Results

256-bit $P$:

- most of the time, we have a faster AND smaller solution
- no impact on BRAMs and periods

384-bit $P$:
Conclusion

The proposed hierarchical approach BE:

- improves the main cost of the BE algorithm from $r^2 c^2$ to $r^2 c$
- preserves quite well the internal parallelism (for $c = 2$)
- on a XC7Z020 FPGA, it shows an improvement up to 18% in total time and up to 31% in DSPs

Future Work

- study the architecture for the cases $c = 3, 4$
- implement a full ECC crypto-processor
A high speed coprocessor for elliptic curve scalar multiplications over $\mathbb{F}_p$.

Cox-Rower architecture for fast parallel Montgomery multiplication.

Elliptic curve cryptosystems.

[Mil85] V. S. Miller.
Use of elliptic curve in cryptography.

Modulo reduction in residue number systems.
## Hardware Implementation Results

<table>
<thead>
<tr>
<th>$\mathbb{F}_p$ width (bits)</th>
<th>BE algo.</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>KBE</td>
<td>HBE</td>
<td>KBE</td>
<td>HBE</td>
<td>KBE</td>
<td>HBE</td>
<td>KBE</td>
<td>HBE</td>
</tr>
<tr>
<td>$w$ (bits)</td>
<td>17</td>
<td>20</td>
<td>24</td>
<td>28</td>
<td>256</td>
<td>758</td>
<td>1073</td>
<td>784</td>
</tr>
<tr>
<td>nb. slices</td>
<td>445</td>
<td>758</td>
<td>1073</td>
<td>784</td>
<td>785</td>
<td>769</td>
<td>753</td>
<td>843</td>
</tr>
<tr>
<td>nb. DSP</td>
<td>51</td>
<td>35</td>
<td>45</td>
<td>39</td>
<td>52</td>
<td>42</td>
<td>76</td>
<td>60</td>
</tr>
<tr>
<td>nb. BRAM</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>period (ns)</td>
<td>9.8</td>
<td>10.3</td>
<td>9.6</td>
<td>8.9</td>
<td>9.6</td>
<td>9.5</td>
<td>9.7</td>
<td>9.6</td>
</tr>
<tr>
<td>nb. cycles</td>
<td>98</td>
<td>91</td>
<td>88</td>
<td>83</td>
<td>89</td>
<td>81</td>
<td>77</td>
<td>71</td>
</tr>
<tr>
<td>time (ns)</td>
<td>960.4</td>
<td>937.3</td>
<td>844.8</td>
<td>738.7</td>
<td>854.4</td>
<td>769.5</td>
<td>746.9</td>
<td>681.6</td>
</tr>
</tbody>
</table>

HLS implementation results on a XC7Z020 FPGA for our HBE and the KBE (from [KKSS00]) algorithms for two widths of prime field elements and four RNS channel widths $w$. 