RNS Base Extension through Operand Scaling on FPGA for Large Integers

Émilie DEBELLE Arnaud TISSERAND Karim BIGOU

ARITH 2026, Fulda

This work received funding from the France 2030 program, managed by the French National Research Agency under grant agreement No. ANR-22-PECY-0004 ARSENE

Context

Large-integers arithmetic for asymmetric cryptography ECC, RSA, FHE, isogenies: $500 \cdots 4\,000$ bits

Residue number system (RNS) offers natural parallel $\{\pm, \times\}$, but $\{\div, \mathrm{mod}, \gtrless, \mathtt{round}\}$ operations are more complex

RNS base extension (BE) is used inside those operations

We propose:

- Alternative efficient BE algorithm with interesting properties
- FPGA accelerators for performance evaluation and comparisons

ECC: elliptic curve cryptography, FHE: fully homomorphic encryption

Agenda

- Short introduction to RNS and BE
- Our new BE algorithm: OSBE
- Accelerator architecture and FPGA implementations
- Comparisons to SotA solution
- Conclusion

SotA: state of the art

Notations

- Upper case variables: large values ($500 \cdots 4\,000$ bits)
- Lower case variables: small values & machine words (< 60 bits)
- Parameter w: width of machine words
- Modular reduction: $(\texttt{expr}) \bmod \texttt{val} = |\texttt{expr}|_{\texttt{val}}$
- EMM: elementary modular multiplication $|x \cdot y|_m$, with x, y, m machine words (common metric for cost coarse estimation)
- $\langle X \rangle$: RNS representation of X
- CRT: Chinese remainder theorem
- Gray text: not presented here (see paper / ask us)

Residue Number System (RNS)

Represent X by its residues $\langle X \rangle_{\mathcal{A}} = (x_1, x_2, \ldots, x_k)$, with $x_i = |X|_{a_i}$

- Base $\mathcal{A}$ of coprime moduli: $(a_1, a_2, \ldots, a_k)$, $A = \prod_{i=1}^{k} a_i$
- All x_i and a_i are w bits integers
- Parameters: k, w such that kw large enough

Where RNS is great: *fully parallel* $\{\pm, \cdot\}$ operations!

$$\langle X \pm Y \rangle_{\mathcal{A}} = \left(|x_1 \pm y_1|_{a_1} \ , \ \cdots \ , \ |x_k \pm y_k|_{a_k} \right)$$
$$\langle X \cdot Y \rangle_{\mathcal{A}} = \left(|x_1 \cdot y_1|_{a_1} \ , \ \cdots \ , \ |x_k \cdot y_k|_{a_k} \right)$$

But RNS is a non-positional number system, $\{\div, \bmod, \gtrless, \texttt{round}\}$ operations are more complex

RNS Base Extension (BE)

Converts $\langle X \rangle_{\mathcal{A}}$ to $\langle X \rangle_{\mathcal{B}}$ where $\mathcal{B}$ is coprime to $\mathcal{A}$

Interest:

- In $\mathcal{A}$, some operations are not possible (e.g. $\div a_i$)
- So, convert to $\mathcal{B}$, perform operations, convert back to $\mathcal{A}$

$\mathcal{A}$ and $\mathcal{B}$ have k moduli of w bits (all coprime)

Operating modes:

- Exact mode: result is $\langle X \rangle_{\mathcal{B}}$
- Approximate mode: result is $\langle X \rangle_{\mathcal{B}}$ or $\langle X + A \rangle_{\mathcal{B}}$ (even $\langle X + 2A \rangle_{\mathcal{B}}, \cdots, \langle X + kA \rangle_{\mathcal{B}}$)

Notations: $\mathrm{BE}_{\mathrm{ex}}$ and $\mathrm{BE}_{\mathrm{ap}}$

Motivation for Efficient BE: RNS Modular Multiplication

RNS variant of Montgomery modular multiplication (RNSMM):

Input: $\langle X\rangle_{\mathcal{A}}, \langle X\rangle_{\mathcal{B}}, \langle Y\rangle_{\mathcal{A}}, \langle Y\rangle_{\mathcal{B}}$
Output: $\langle Z\rangle_{\mathcal{A}}, \langle Z\rangle_{\mathcal{B}}$ where $Z \equiv XYA^{-1} \bmod N$

1 $\langle U\rangle_{\mathcal{A}} \leftarrow \langle X\rangle_{\mathcal{A}} \cdot \langle Y\rangle_{\mathcal{A}}$
2 $\langle U\rangle_{\mathcal{B}} \leftarrow \langle X\rangle_{\mathcal{B}} \cdot \langle Y\rangle_{\mathcal{B}}$
3 $\langle V\rangle_{\mathcal{A}} \leftarrow \langle U\rangle_{\mathcal{A}} \cdot \langle -N^{-1}\rangle_{\mathcal{A}}$
4 $\langle V\rangle_{\mathcal{B}} \leftarrow \mathrm{BE}_{\mathrm{ap}}(\langle V\rangle_{\mathcal{A}})$
5 $\langle Z\rangle_{\mathcal{B}} \leftarrow \langle A^{-1}\rangle_{\mathcal{B}} \cdot (\langle U\rangle_{\mathcal{B}} + \langle V\rangle_{\mathcal{B}} \cdot \langle N\rangle_{\mathcal{B}})$
6 $\langle Z\rangle_{\mathcal{A}} \leftarrow \mathrm{BE}_{\mathrm{ex}}(\langle Z\rangle_{\mathcal{B}})$
7 **return** $\langle Z\rangle_{\mathcal{A}}, \langle Z\rangle_{\mathcal{B}}$

In target crypto applications, BEs dominate computation time

More details: [1]–[3]

SotA BE Algorithms

- MRS: Mixed Radix System [4], [5]
- FBE: Fast Base Extension [6]
- SK: Shenoy and Kumaresan [7]
- KBE: Kawamura et al. [3] SotA in hardware

BE algorithm	MRS	FBE	SK	KBE
$\mathrm{BE}_{\mathrm{ap}}$	✓	✓	×	✓
$\mathrm{BE}_{\mathrm{ex}}$	✓	×	✓	✓
constraints	$\emptyset$	$\emptyset$	additional modulus	moduli close to 2^w
chains of dependencies	long	short	short	short

KBE, FBE and SK are CRT based BEs

Operand Scaling Base Extension (OSBE): Principle

Notation: $A_k = \prod_{i=1}^{k-1} a_i$, *i.e.* A without a_k the last modulus

Idea: perform a scaling by A_k and add missing operations

$$X = q_X \cdot A_k + R_X$$

Remarks:

- q_X and R_X are computed in $\mathcal{B} \cup \{a_k\}$
- Scaling by A_k costs almost one BE
- $2k$ additional EMMs required to complete the full BE
  (lines 4, 6, 10 in algorithm next slide)

OSBE Algorithm

Input: $\langle X \rangle_{\mathcal{A}}$, v_0, $\langle Z_0 \rangle_{\mathcal{B}}$
Output: $\langle Z \rangle_{\mathcal{B}} = \langle X \rangle_{\mathcal{B}}$ or $\langle X + A \rangle_{\mathcal{B}}$

$\langle Z \rangle_{\mathcal{B}} \leftarrow \langle Z_0 \rangle_{\mathcal{B}}$
for $i = 1$ **to** $k - 1$ **do**
  $y_i \leftarrow |x_i \cdot \texttt{CST1}_i|_{a_i}$ // $\texttt{CST1}_i = |A_{i,k}^{-1}|_{a_i}$
$v_k \leftarrow |v_0 + x_k \cdot \texttt{CST1}_k|_{a_k}$ // $\texttt{CST1}_k = |A_k^{-1}|_{a_k}$
for $i = 1$ **to** $k - 1$ **do**
  $v_k \leftarrow |v_k + y_i \cdot \texttt{CST2}_i|_{a_k}$ // $\texttt{CST2}_i = |-a_i^{-1}|_{a_k}$
  for $j = 1$ **to** k **do**
    $z_j \leftarrow |z_j + y_i \cdot \texttt{CST3}_{i,j}|_{b_j}$ // $\texttt{CST3}_{i,j} = |A_{i,k}|_{b_j}$
for $j = 1$ **to** k **do**
  $z_j \leftarrow |z_j + v_k \cdot \texttt{CST4}_j|_{b_j}$ // $\texttt{CST4}_j = |A_k|_{b_j}$
return $\langle Z \rangle_{\mathcal{B}}$

Exact mode: $\langle Z_0 \rangle_{\mathcal{B}} = \langle -(k-2)A_k \rangle_{\mathcal{B}}$ and $v_0 = k - 2$
Approx. mode: $\langle Z_0 \rangle_{\mathcal{B}} = \langle 0 \rangle_{\mathcal{B}}$ and $v_0 = 0$

OSBE *vs* Other BE Algorithms

BE algorithm	MRS	FBE	SK	KBE	OSBE
$\mathrm{BE}_{\mathrm{ap}}$	✓	✓	×	✓	✓
$\mathrm{BE}_{\mathrm{ex}}$	✓	×	✓	✓	✓
constraints	$\emptyset$	$\emptyset$	additional modulus	moduli close to 2^w	$\emptyset$
chains of dependencies	long	short	short	short	short

Cost coarse estimation:

- KBE: $k^2 + k$ EMMs
- OSBE: $k^2 + 2k - 1$ EMMs

Remark: slightly higher cost but simpler architecture...

OSBE Interesting Property: Wider Operands

OSBE only requires k coprime moduli without any restriction or specific form

OSBE supports much wider operands for any given w

Example $w = 17$ bits (from our target FPGA):

- KBE limited to 1 118 bits operands [8]
- OSBE limited to 94 544 bits operands

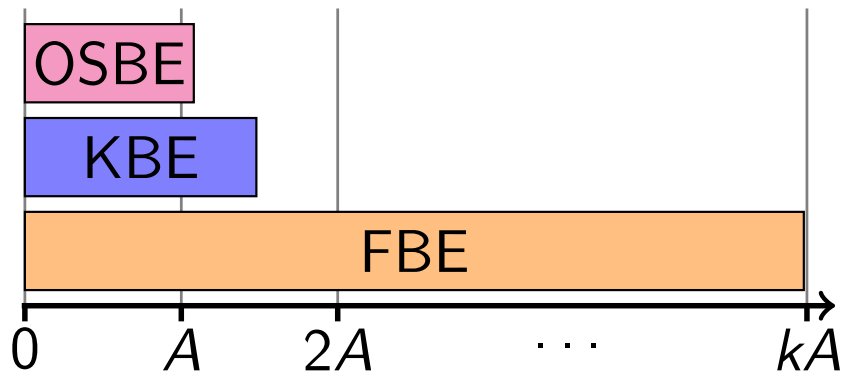
Interesting to fit FPGA or GPU computations units

OSBE Interesting Property: Sharper Approximation

Maximum BE result depends on the algorithm

In modular sums of products, how many terms can be accumulated ?

Algo.	BE result	512 bits, $w = 17$	2 048 bits, $w = 18$
OSBE	$< (1 + \frac{k-2}{a_k})A$	1 695	2
KBE	$< \frac{3}{2}A$	1 130	1
FBE	$< k\,A$	27	0

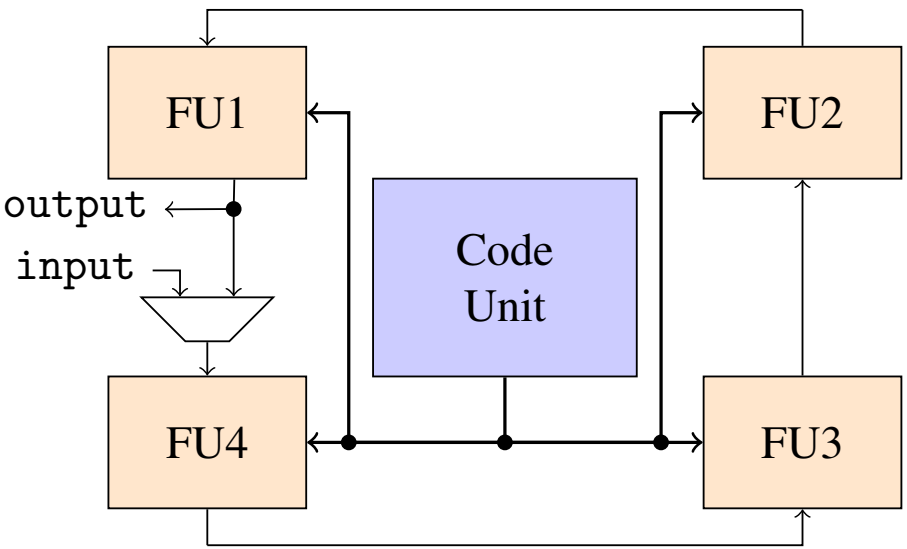


Reminder: $A = \prod_{i=1}^{k} a_i$ (*i.e.* kw bits), a_k last modulus (*i.e.* w bits)

Accelerator Architecture (Variants)

Synthesis-time parameters:

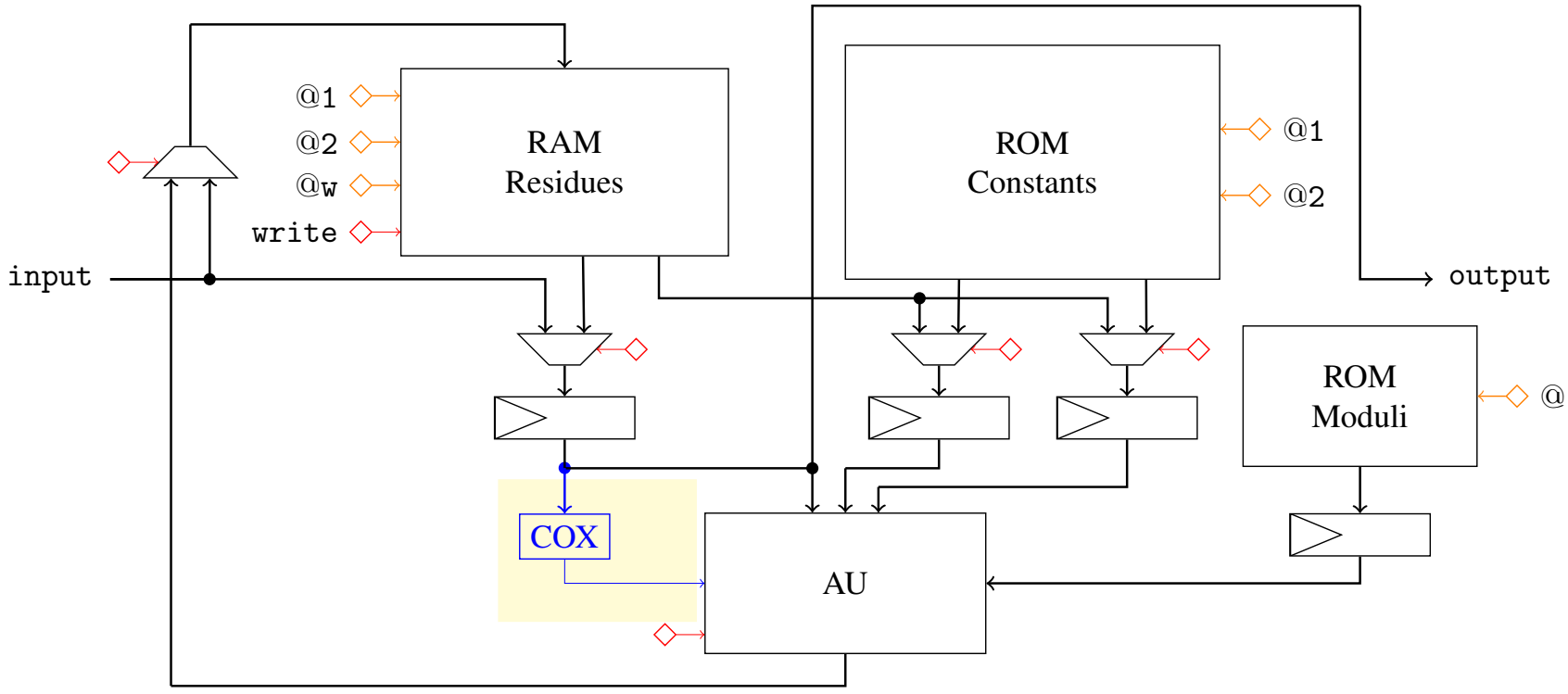
- Number of functional units (FUs): 4, 8, 16
- Operand sizes: 507, 1 024, 2 048, 4 096 bits (future: flexible)
- w width of machine words (optim. $w = 17$ on target FPGA)
- Moduli forms:
  - Generic Moduli (GM)
  - Pseudo-Mersenne (PM): $2^w - \epsilon_i$



- Communications between FUs: w bits 1 way ring
- SIMD[1] control: 30 bits instruction sent to all FUs every cycle
- Same overall architecture for both OSBE and SotA (KBE [3])
- Code for results: RNSMM

[1] single instruction, multiple data

Functional Units & Arithmetic Units (AU)



OSBE:

- No COX unit
- AU operations:

$\texttt{acc} \leftarrow |\text{op}_1 \cdot \text{op}_2 + \texttt{acc}|_m$

$\texttt{acc} \leftarrow |\text{op}_1 \cdot \text{op}_2 + \text{op}_3|_m$

SotA:

- Requires a COX unit
- AU operations:

$\texttt{acc} \leftarrow |\text{op}_1 \cdot \text{op}_2 + \text{op}_3 + \texttt{acc}|_m$

$\texttt{acc} \leftarrow |\text{op}_1 \cdot \text{op}_2 + \text{op}_3|_m$

$\texttt{acc} \leftarrow |\text{op}_1 \cdot \text{op}_2 + \texttt{acc}|_m$

$\texttt{acc} \leftarrow |\text{op}_1 \cdot \text{op}_2|_m$

FPGA Implementations: Methodology

- Same efforts, tool and target FPGA for OSBE and SotA
- Design: SystemVerilog
- Simulations: Icarus Verilog for debug and intensive tests
- Synthesis and implementation: Vivado 2025.2 from AMD
- FPGA: Artix-7 (xc7a200tfbg676-2) from AMD
- $w = 17$ bits (18 bits 2'sC operand in DSP slices)
- Evaluated code: RNSMM

FPGA Implementations: Results

FU	Size bits	Archi.	Cycles /MM		Cycles total		Freq. MHz		Time μs		LUT		FF		BRAM	DSP
4	507	SotA	544		741		204		2.7		826		638		6	12
		OSBE	552	+1.4%	749	+1.1%	208	+1.9%	2.7		692	-19.4%	553	-15.4%	6	12
	1024	SotA	2 112		2 501		204		10.4		736		625		12	12
		OSBE	2 120	+0.4%	2 509	+0.3%	208	+1.9%	10.2	-2.0%	599	-22.9%	614	-1.8%	12	12
	2048	OSBE	7 820		8 569		208		37.6		616		587		43	12
	4096	OSBE	30 020		31 489		208		144.3		611		797		118	12
8	1024	SotA	1 056		1 441		200		5.3		1 283		1 096		14	24
		OSBE	1 072	+1.5%	1 457	+1.1%	204	+2.0%	5.3		1 114	-15.2%	1 003	-9.3%	14	24
	2048	OSBE	4 176		4 945		204		20.5		1 181		1 155		32	24
	4096	OSBE	15 516		17 005		204		76.1		1 178		1 279		112	24
16	2048	OSBE	2 112		2 873		208		10.2		2 229		2 101		35	48
	4096	OSBE	8 288		9 817		208		39.8		2 270		2 535		104	48

Comparison between OSBE and SotA:

- Similar speed: higher frequency mitigates additional EMMs
- Similar area for DSPs/BRAMs, small reduction for LUTs/FFs

SotA limit: $w = 17 \Longrightarrow$ operands $< 1\,118$ bits [8]

Conclusion

OSBE is an alternative BE algorithm leading to simple architectures

- As fast as SotA
- Slightly smaller than SotA
- Supports (very) wide operands
- Sharper approximation (future prospect)

Personal dedication: to Alberto...

Questions?

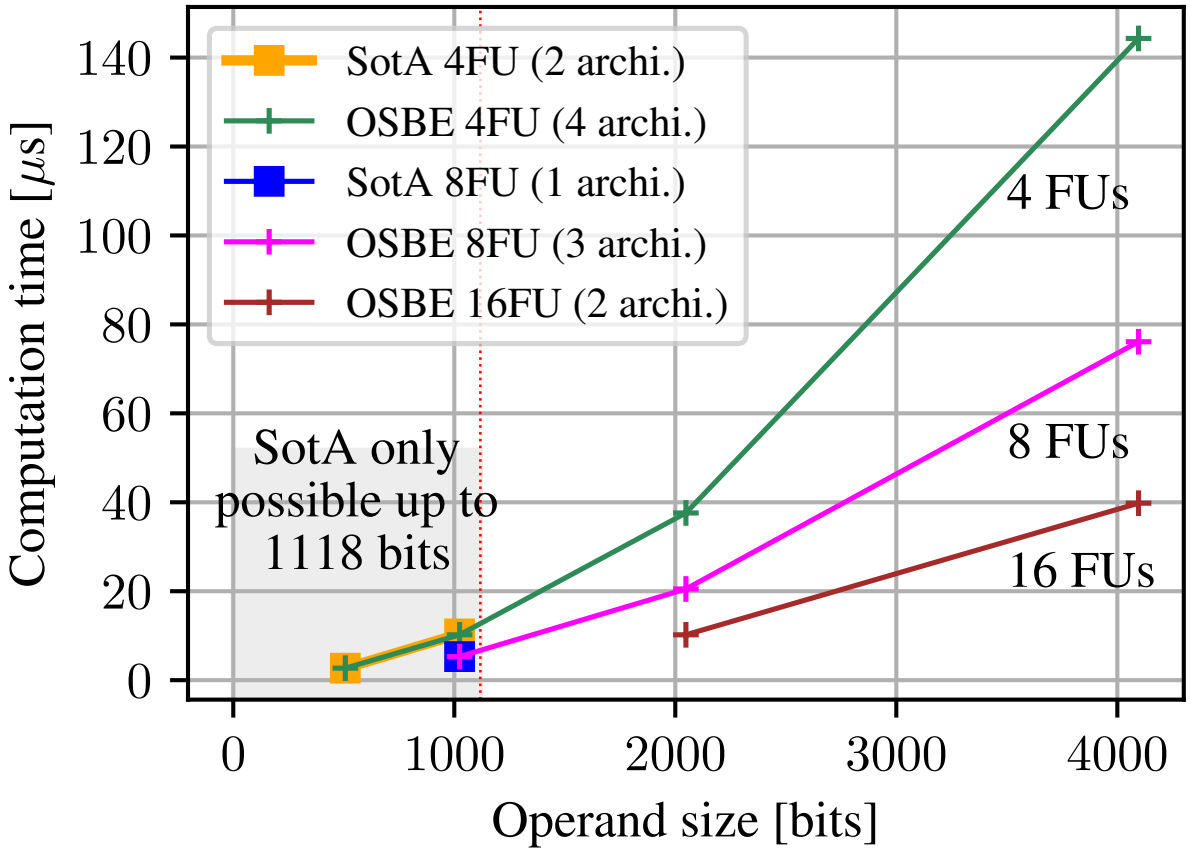
Residue Number System (RNS)

- All integers in $[0, A[$ can be represented in base $\mathcal{A}$
- All computations are performed modulo A
- Conversion from $\langle X \rangle$ to X uses CRT

Number of FUs & Pipeline Stages

- Number of FUs $\mathtt{f}$ chosen at synthesis time
- Each FU handles $k/\mathtt{f}$ residues
- Control designed for pipeline stages without bubbles
- Constraints:
  - $k > 6\mathtt{f}$ for GM
  - $k > 7\mathtt{f}$ for PM
- For "small" k, large architectures with high $\mathtt{f}$ may not be possible

FPGA Implementations: Timing Summary



Remarks:

- $w = 17 \Longrightarrow$ SotA only possible up to 1 118 bits operands
- $w = 17 \Longrightarrow$ OSBE possible up to 94 544 bits operands
- OSBE is as fast as SotA, but supports (much) wider operands

FPGA Implementations: 507 bits PM & GM

Moduli type	Archi.	Cycles /MM		Cycles total		Freq. MHz		Time μs		LUT		FF		BRAM	DSP
GM	SotA	544		741		204		2.7		826		638		6	12
	OSBE	552	+1.4%	749	+1.1%	208	+1.9%	2.7		692	-19.4%	553	-15.4%	6	12
PM	SotA	544		741		167		3.3		1 651		1 009		6	16
	OSBE	552	+1.4%	749	+1.1%	204	+18.1%	2.7	-22.2%	1 574	-4.9%	1 088	+7.3%	6	16

Maximum operand width for PM and $w = 17$ bits:

- [8] reports at most 506 bits operands for SotA
- We have at most 507 bits operands for SotA and OSBE

KBE Algorithm [3]

Input: $\langle X \rangle_{\mathcal{A}}$, $\alpha = 0$ or 0.5
Output: $\langle Z \rangle_{\mathcal{B}} = \langle X \rangle_{\mathcal{B}}$ or $\langle X + A \rangle_{\mathcal{B}}$

1 **for** $i = 1$ **to** k **do**
2 $\quad y_i \leftarrow |x_i \cdot \mathtt{CST1}_i|_{a_i}$ // $\mathtt{CST1}_i = \left|A_i^{-1}\right|_{a_i}$
3 **for** $j = 1$ **to** k **do**
4 $\quad z_j \leftarrow 0$
5 $c \leftarrow \alpha$
6 **for** $i = 1$ **to** k **do**
7 $\quad c \leftarrow c + \mathrm{trunc}(y_i)/2^w$
8 $\quad \varepsilon_i \leftarrow \lfloor c \rceil$
9 $\quad c \leftarrow c - \varepsilon_i$
10 $\quad$ **for** $j = 1$ **to** k **do**
11 $\quad\quad z_j \leftarrow |z_j + y_i \cdot \mathtt{CST2}_{i,j} + \varepsilon_i \cdot \mathtt{CST3}_j|_{b_j}$
$\quad\quad$ // $\mathtt{CST2}_{i,j} \leftarrow |A_i|_{b_j}$ and $\mathtt{CST3}_j = |-A|_{b_j}$
12 **return** $\langle Z \rangle_{\mathcal{B}}$

Chinese Remainder Theorem (CRT)

For $X \in [0..A-1]$ one has:

$$X = \left(\sum_{i=1}^{k} \left| x_i A_i^{-1} \right|_{a_i} A_i \right) \bmod A \; = \; S - qA \qquad (1)$$

with:

- $A_i = A/a_i$
- S the sum of products inside parentheses
- $q = \left\lfloor \frac{S}{A} \right\rfloor$

References I

[1] K. C. Posch and R. Posch, "Modulo reduction in residue number systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 5, pp. 449–454, May 1995, doi: 10.1109/71.382314.

[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*, vol. 61, no. 8, pp. 1071–1083, Aug. 2012, doi: 10.1109/TC.2012.84.

[3] S. Kawamura, M. Koike, F. Sano, and A. Shimbo, "Cox-Rower architecture for fast parallel Montgomery multiplication," in *Proc. Annual international conference on theory and applications of cryptographic techniques (EUROCRYPT)*, May 2000, vol. 1807, pp. 523–538. doi: 10.1007/3-540-45539-6_37.

[4] H. L. Garner, "The residue number system," *IRE Transactions on Electronic Computers*, vol. EC–8, no. 2, pp. 140–147, June 1959, doi: 10.1109/TEC.1959.5219515.

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

References II

[6] J.-C. Bajard, L.-S. Didier, and P. Kornerup, "Modular multiplication and base extensions in residue number systems," in *Proc. International symposium on computer arithmetic (ARITH)*, Apr. 2001, pp. 59–65. doi: 10.1109/ARITH.2001.930104.

[7] A. P. Shenoy and R. Kumaresan, "Fast base extension using a redundant modulus in RNS," *IEEE Transactions on Computers*, vol. 38, no. 2, pp. 292–297, Feb. 1989, doi: 10.1109/12.16508.

[8] B. Gérard, J.-G. Kammerer, and N. Merkiche, "Contributions to the design of residue number system architectures," in *Proc. International symposium on computer arithmetic (ARITH)*, June 2015, pp. 105–112. doi: 10.1109/ARITH.2015.25.