Modular Hardware Architecture for Somewhat Homomorphic Function Evaluation

CHES 2015

Sujoy Sinha Roy[1], Kimmo Järvinen[1], Frederik Vercauteren[1], Vassil Dimitrov[2], and Ingrid Verbauwhede[1]

[1]ESAT/COSIC and iMinds, KU Leuven

[2]The University of Calgary, Canada and Computer Modelling Group

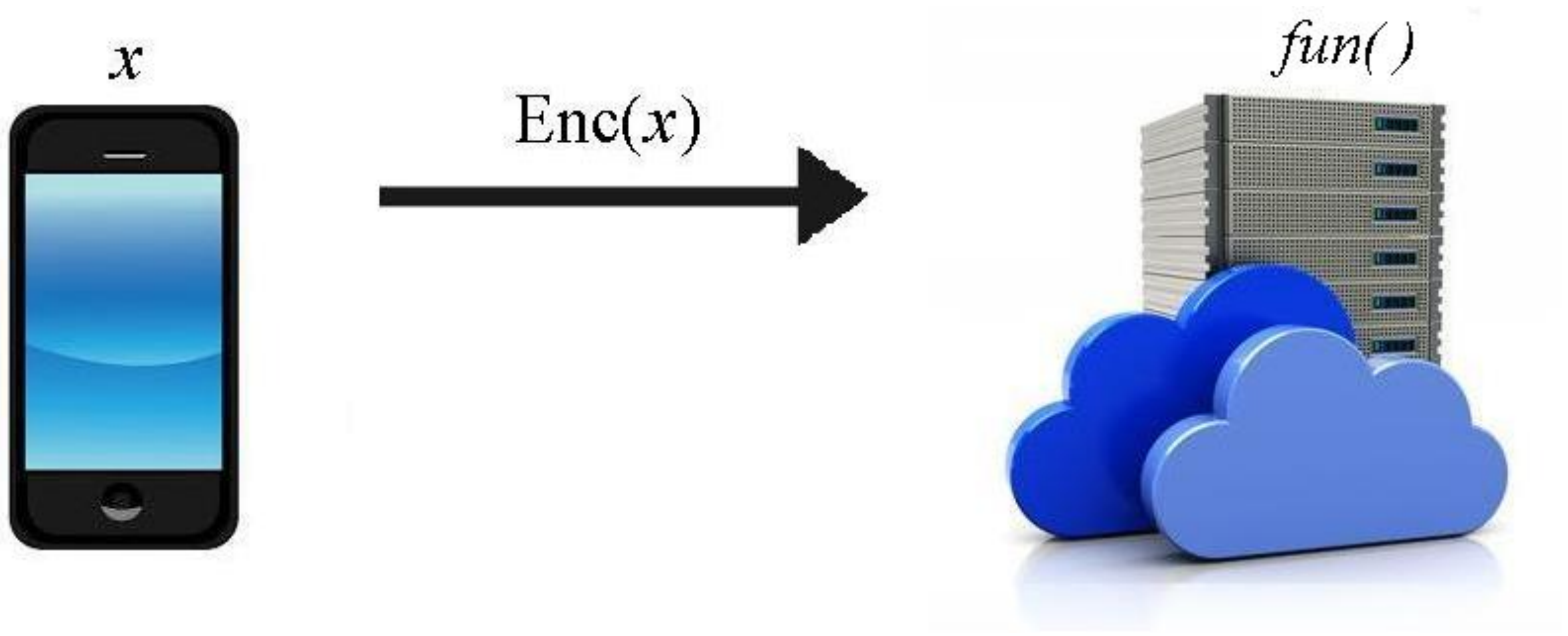
Outsourcing Computation

Outsourcing Computation

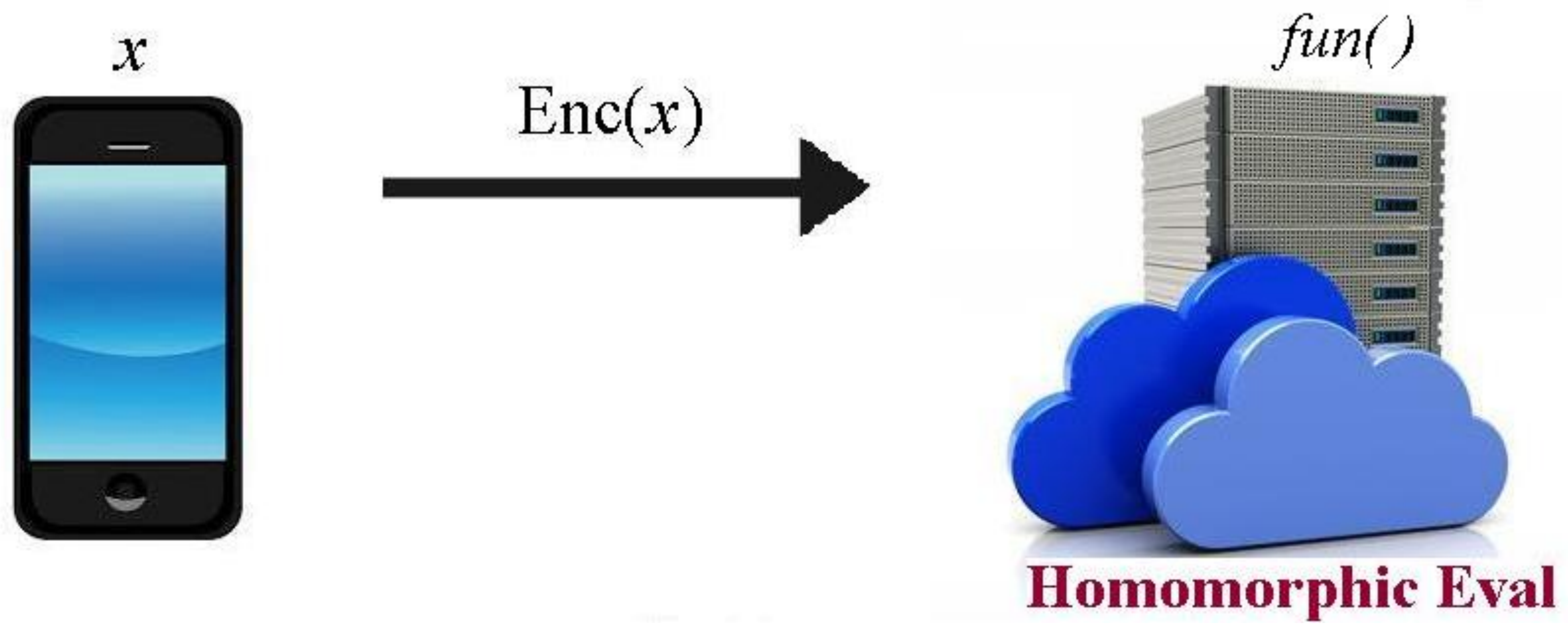
x $fun()$

Outsourcing Computation

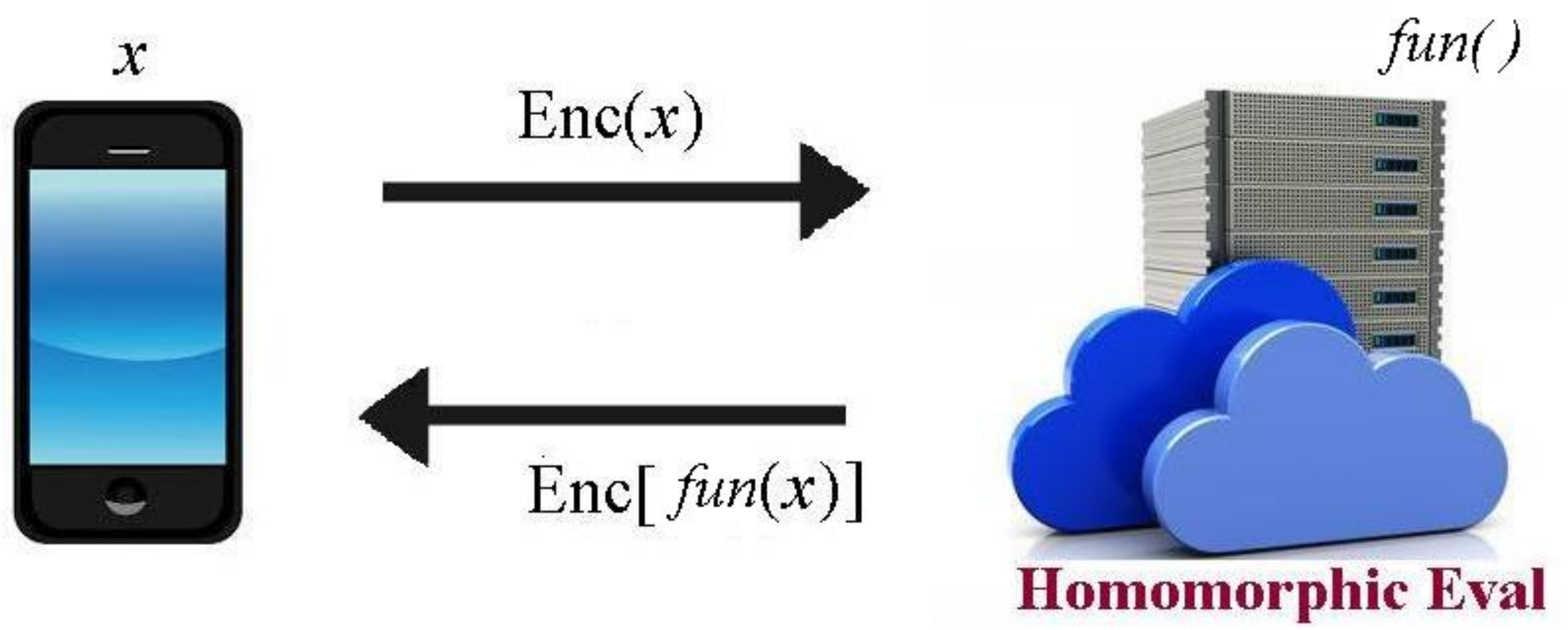
Outsourcing Computation



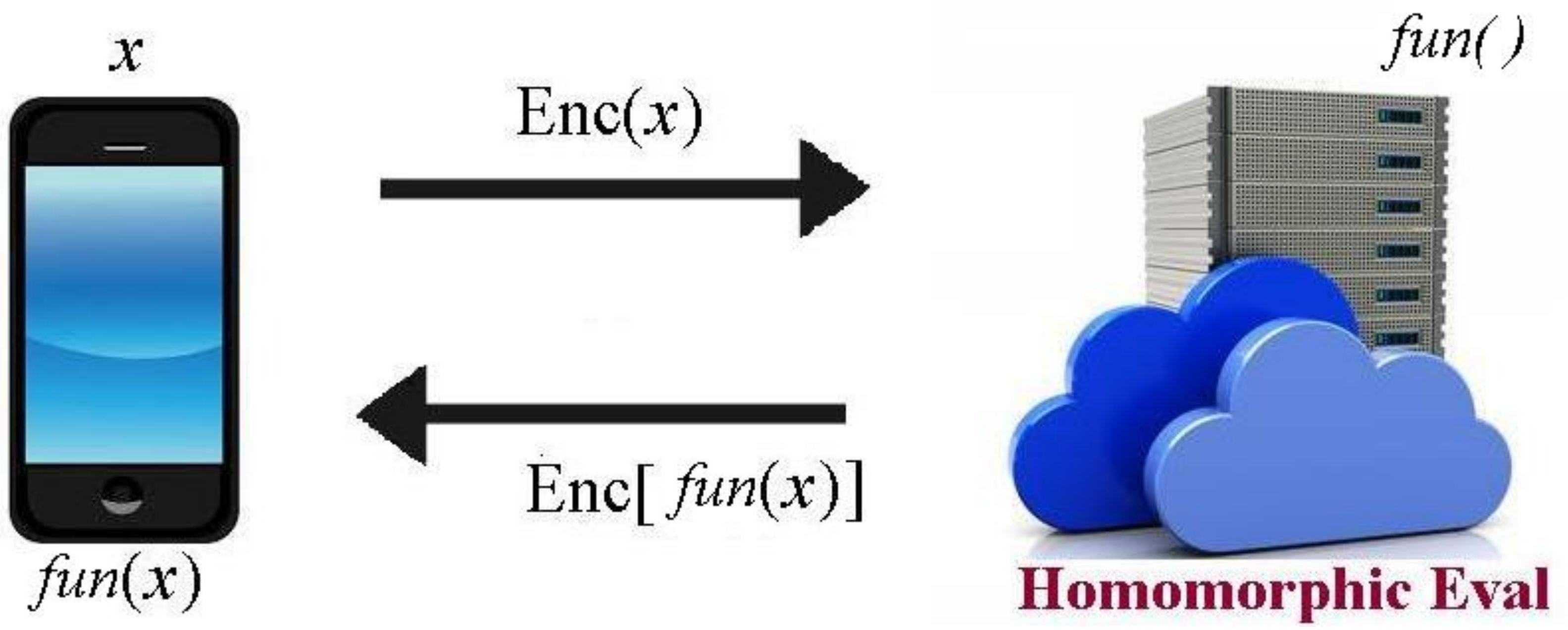
Outsourcing Computation



Outsourcing Computation



Outsourcing Computation

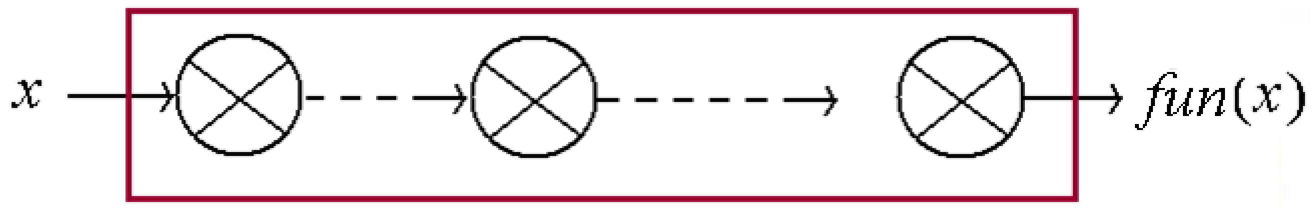


Some Facts about Homomorphic Encryption

- Any *fun*( ) can be represented as a sequence of {+, ×} over GF(2)
- + is *xor gate*
- × is *and* gate
- {*xor, and*} gates together give us *universal gate*

Homomorphic encryption scheme allows us to homomorphically compute GF(2) addition and multiplication on encrypted data.

Some Facts about Homomorphic Encryption



Multiplicative Depth

- Multiplicative depth of *fun* is number of *and gate* in critical path
- Fully Homomorphic Encryption (FHE) ≡ **unlimited** depth
  - ➢ Thus any *fun*
- Somewhat Homomorphic Encryption (SHE) ≡ **limited** depth
  - ➢ Less complicated *fun*

Performances of FHE and SHE

Performance of FHE

Batch Fully Homomorphic Encryption over Integers, by Coron, Lepoint, and Tibouchi. Eurocrypt 2013

- Encryption 61 seconds, Decryption 9.8 seconds
- Multiplication 0.72 seconds
- Recrypt 172 seconds

- AES evaluation takes 113 hours on Intel Core i7-2600 at 3.4 GHz
  - 5120 Multiplications and 2448 Recrypt

FHE is Very Slow

Performance of SHE

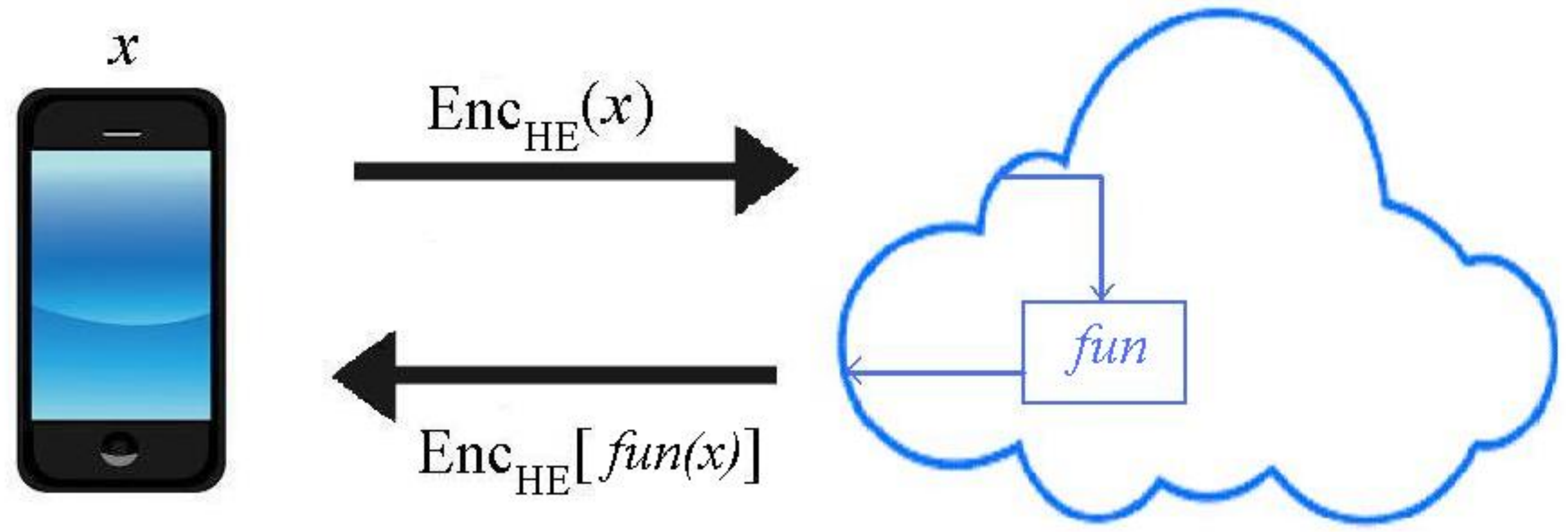
A Comparison of the Homomorphic Encryption Schemes FV and YASHE, by Lepoint, Naehrig. Africacrypt 2014

- Evaluate SIMON -64/128 using YASHE in 70 minutes
  - No recrypt
  - Using 4-cores of Intel Core i7-2600 at 3.4 GHz

SHE is > faster than FHE

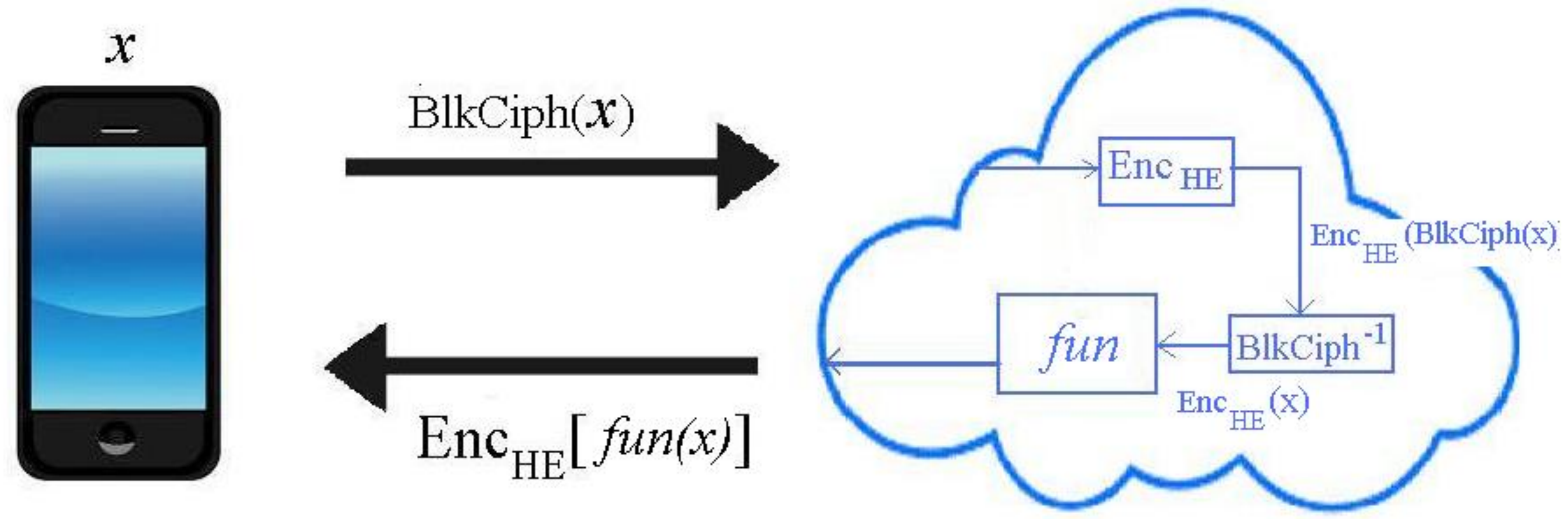
Motivation: Can we accelerate using FPGAs?

Why do we need to Evaluate SIMON in Cloud?


x
$\mathrm{Enc}_{\mathrm{HE}}(x)$
$\mathrm{Enc}_{\mathrm{HE}}[fun(x)]$
fun

- User encrypts message bits using $Enc_{\mathrm{HE}}(\ )$
  - Ciphertext size is huge (can be in GBs)
  - Heavy load on the communication network

Why do we need to Evaluate SIMON in Cloud?



- Ciphertext size is message size
- SIMON has small multiplicative depth

The YASHE Scheme

The YASHE Scheme

- Defined over a ring $R_q = \mathbb{Z}[x]/(f(x))$
  - We use 1228 bit *q*
  - *f* ( ) is 65535-th cyclotomic polynomial, degree $n = 2^{15}$
- YASHE.KeyGen( )→ (*pk*, *sk*, *evk*), *pk*, *sk* $\in R_q$, *evk* $\in R_q^{22}$

The YASHE Scheme

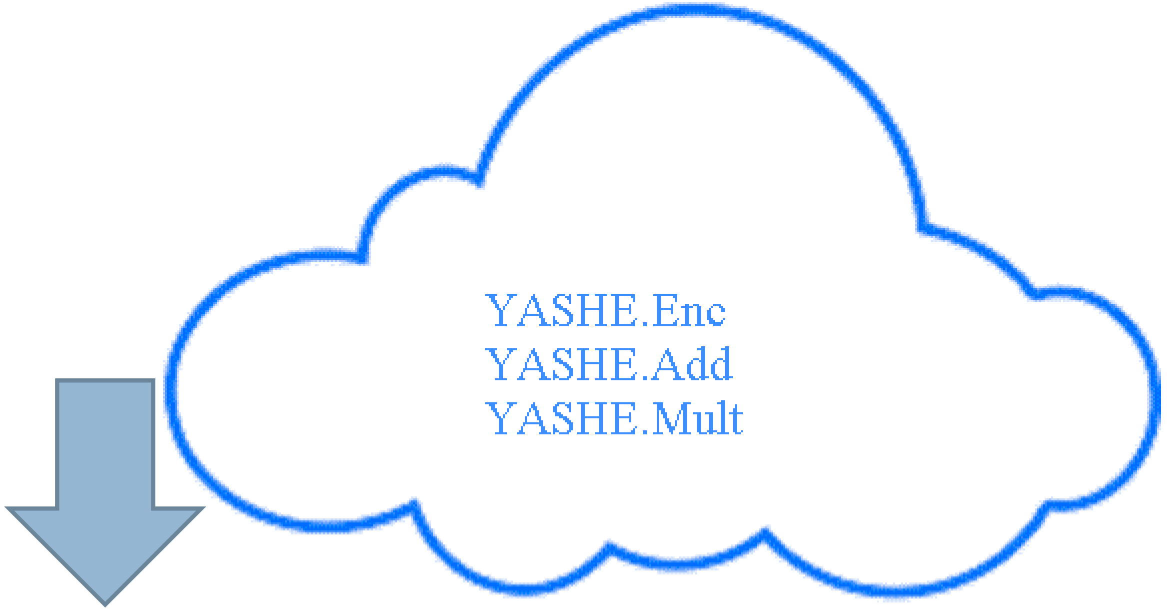
- YASHE.Enc (m, pk)→ c
  - ➢ Gaussian sampling from narrow distribution
  - ➢ One polynomial multiplication and two additions
- YASHE.Dec(c, sk)→ m
  - ➢ One polynomial multiplication and a decoding

The YASHE Scheme

- YASHE.Add $(c_1, c_2) \rightarrow c = c_1 + c_2 \in R_q$
- YASHE.Mult (c_1, c_2)
  - ➢ Compute polynomial multiplication $c_1 \cdot c_2$ in R_Q
  - ➢ $Q \sim n \cdot q^2$ [In our case $|Q|$ = 2,517 bits]
  - ➢ Division and rounding $c' = \lfloor \frac{2}{q} c_1 c_2 \rceil$
  - ➢ Return $c = \mathsf{YASHE.KeySwitch}(c', evk) \in R_q$
  - ➢ $\mathsf{YASHE.KeySwitch}(\ )$ performs 22 poly mult and 21 poly add

Implementation

Operations in the Cloud



- Discrete Gaussian sampling (from narrow distribution)
- Polynomial addition
- **Polynomial multiplication**
- **Division and rounding**

Costly Computation

Polynomial Multiplication

- FFT based multiplication has low complexity ***($n \log n$)***

- Number Theoretic Transform (**NTT**) is a generalization of FFT
  - n-th primitive root of 1 in $\mathbb{Z}_q$ (an integer)
  - Only integer arithmetic modulo q

Polynomial Multiplication using NTT

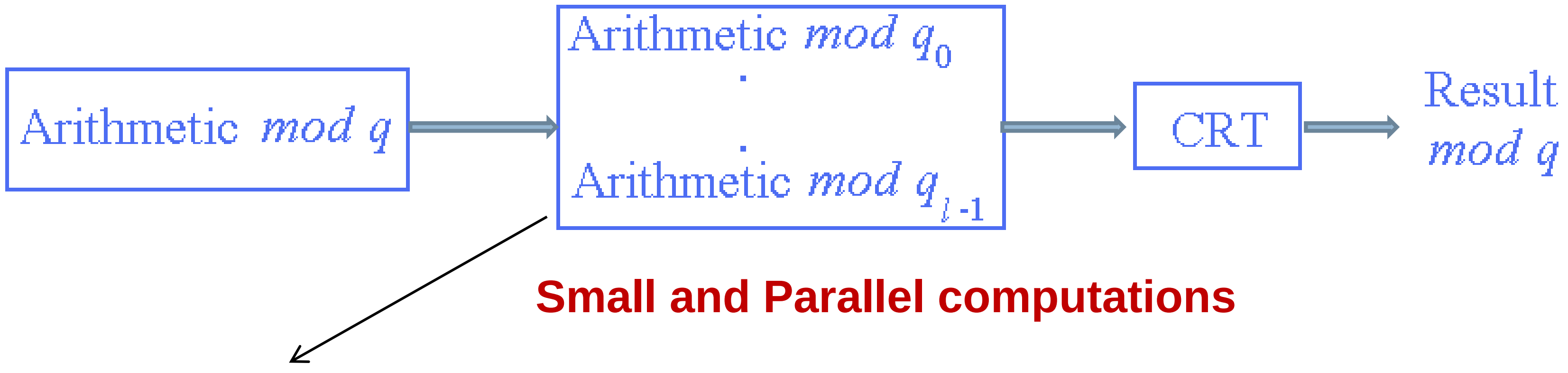
- Expand input polynomials from *n* coefficients to $N = 2^k > 2n - 2$
- Compute *N*-point NTTs
- Multiply them coefficient wise
- Compute INTT
- Finally reduce the result modulo $f(x)$ **[deg(*f*) = *n*]**
- Our $f(x)$ is 65535-th cyclotomic polynomial **[it supports SIMD]**
  - ➢ Not a sparse polynomial
  - ➢ We use polynomial Barrett reduction

Handling of Long Integer Arithmetic

- Coefficients are modulo *q* where |*q*| = 1,228 bits

  [and sometimes modulo *Q* where |*Q*| = 2,517 bits]

- Difficult to implement
- We use CRT and take $q = \prod_0^{l-1} q_i$

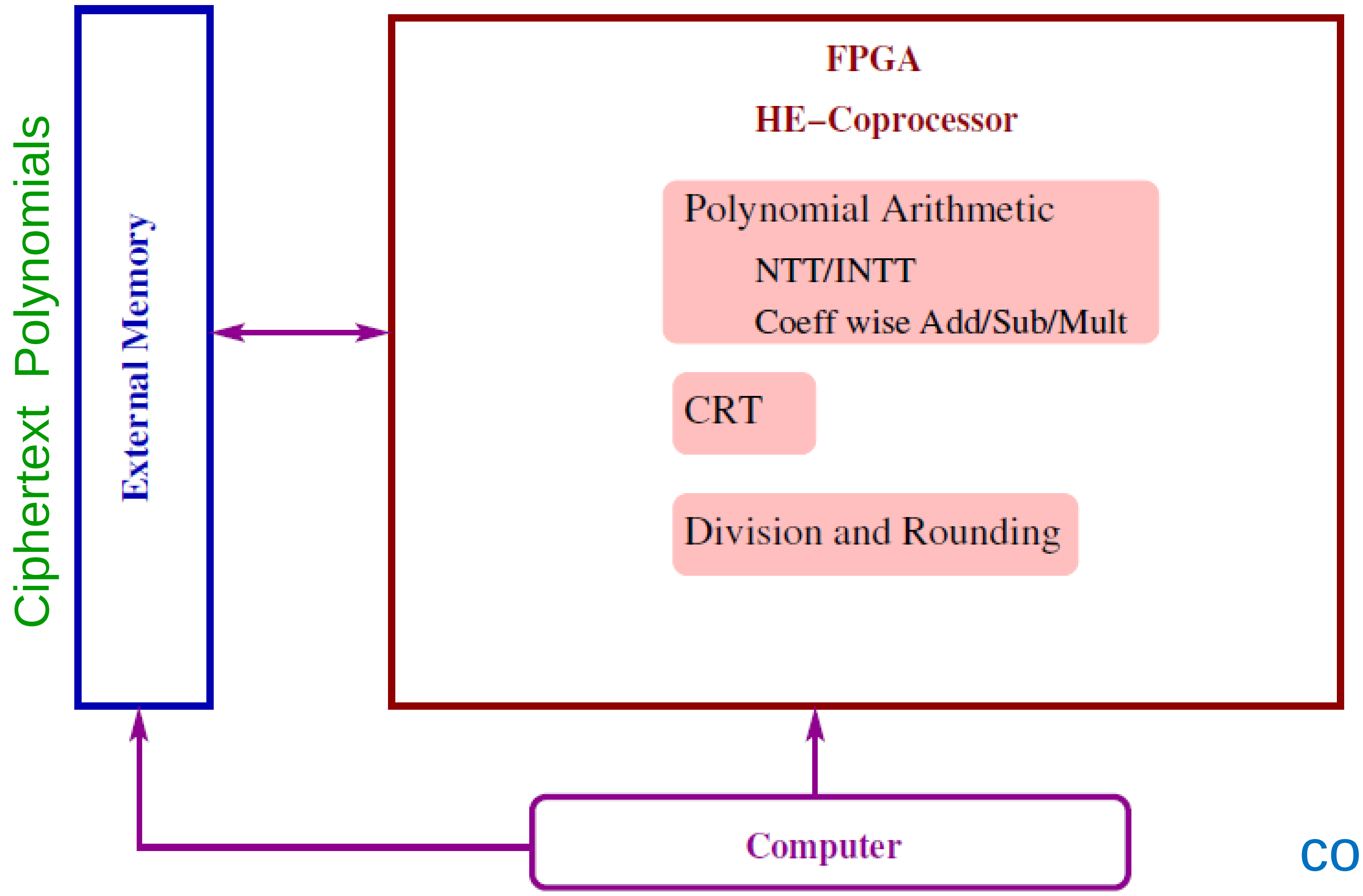


Small and Parallel computations

use DSP multipliers of the FPGA

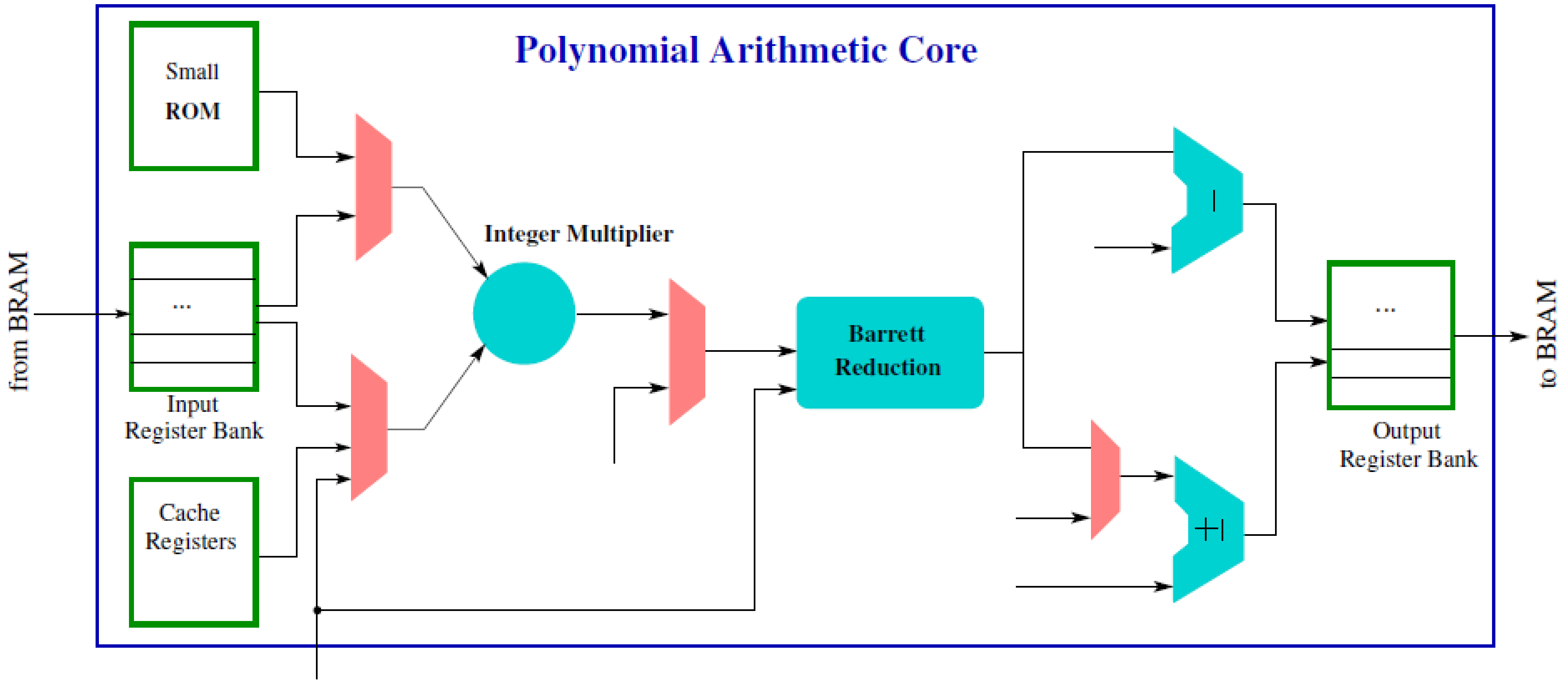
Architecture

Overview of the HE Architecture



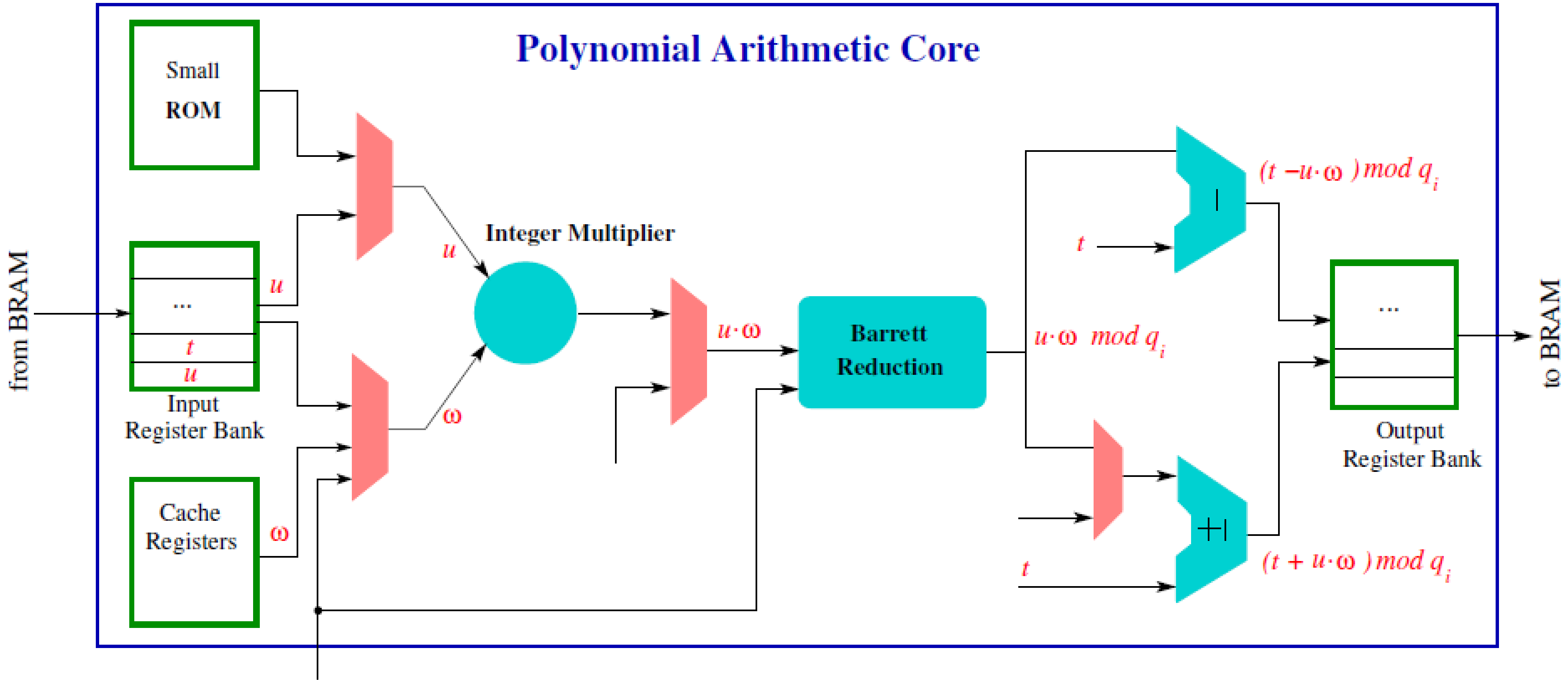
codesign

Polynomial Arithmetic Unit Core



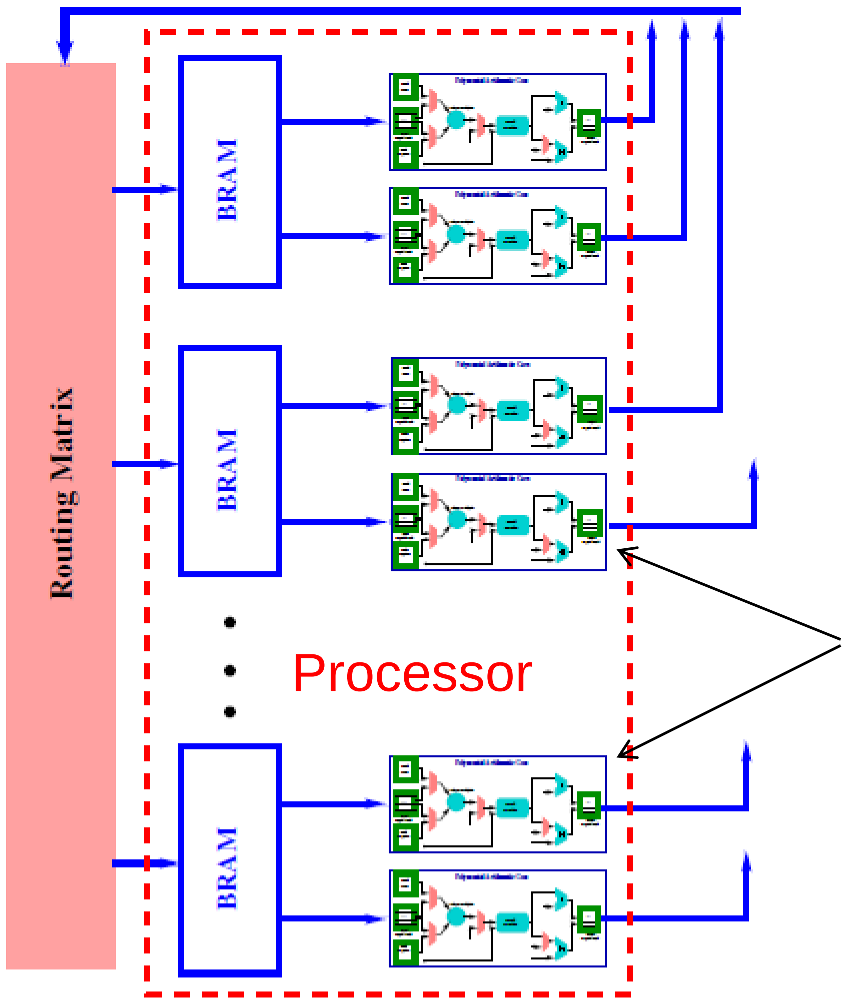
The core is based on our CHES2014 paper "Compact ring-LWE Cryptoprocessor"

Polynomial Arithmetic Unit Core



Computing $t + u \cdot \omega$ $t - u \cdot \omega$ … butterfly during an NTT

Multi-Core Polynomial Arithmetic Unit



- NTT is parallelizable
- Speedup using many cores

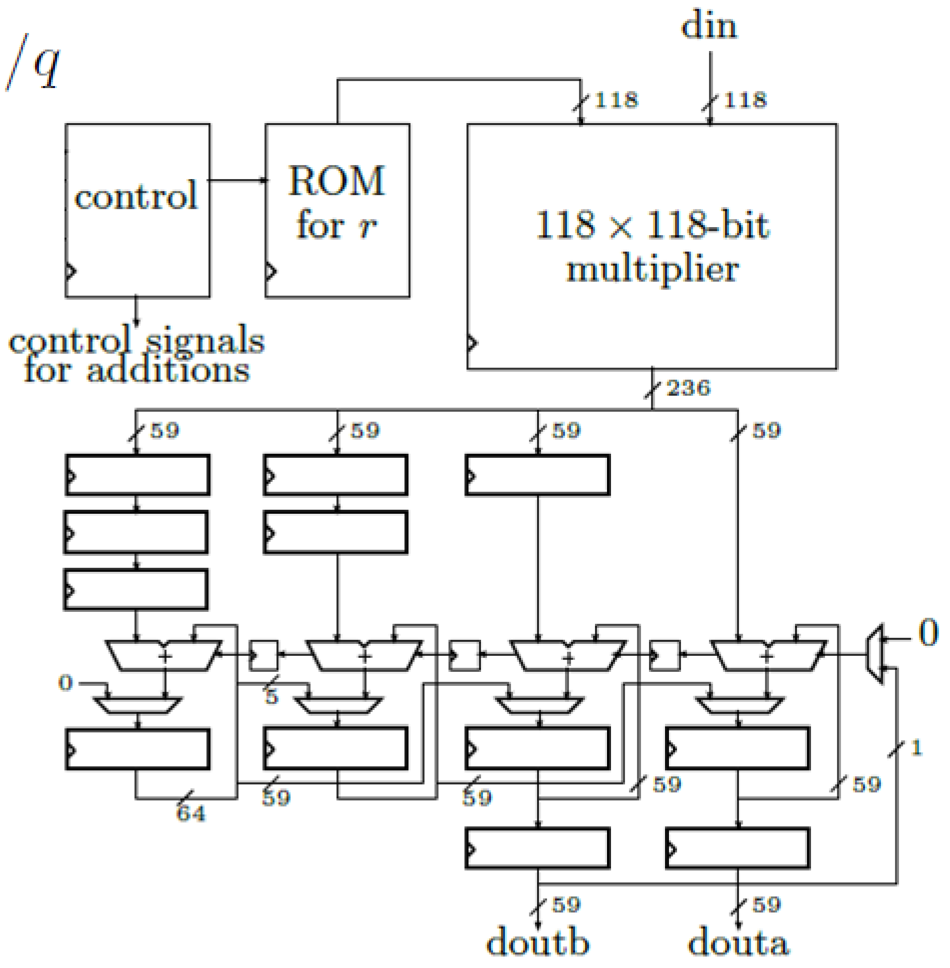
Our architecture has 16 cores

- Routing friendly NTT
  - Local data access

[details in the paper]

Division and Rounding Unit (DRU)

- Divides by q and then rounds to nearest integer (q is fixed)
- Precomputed reciprocal $r = 1/q$
- Multiplies input by r



Implementation of CRT

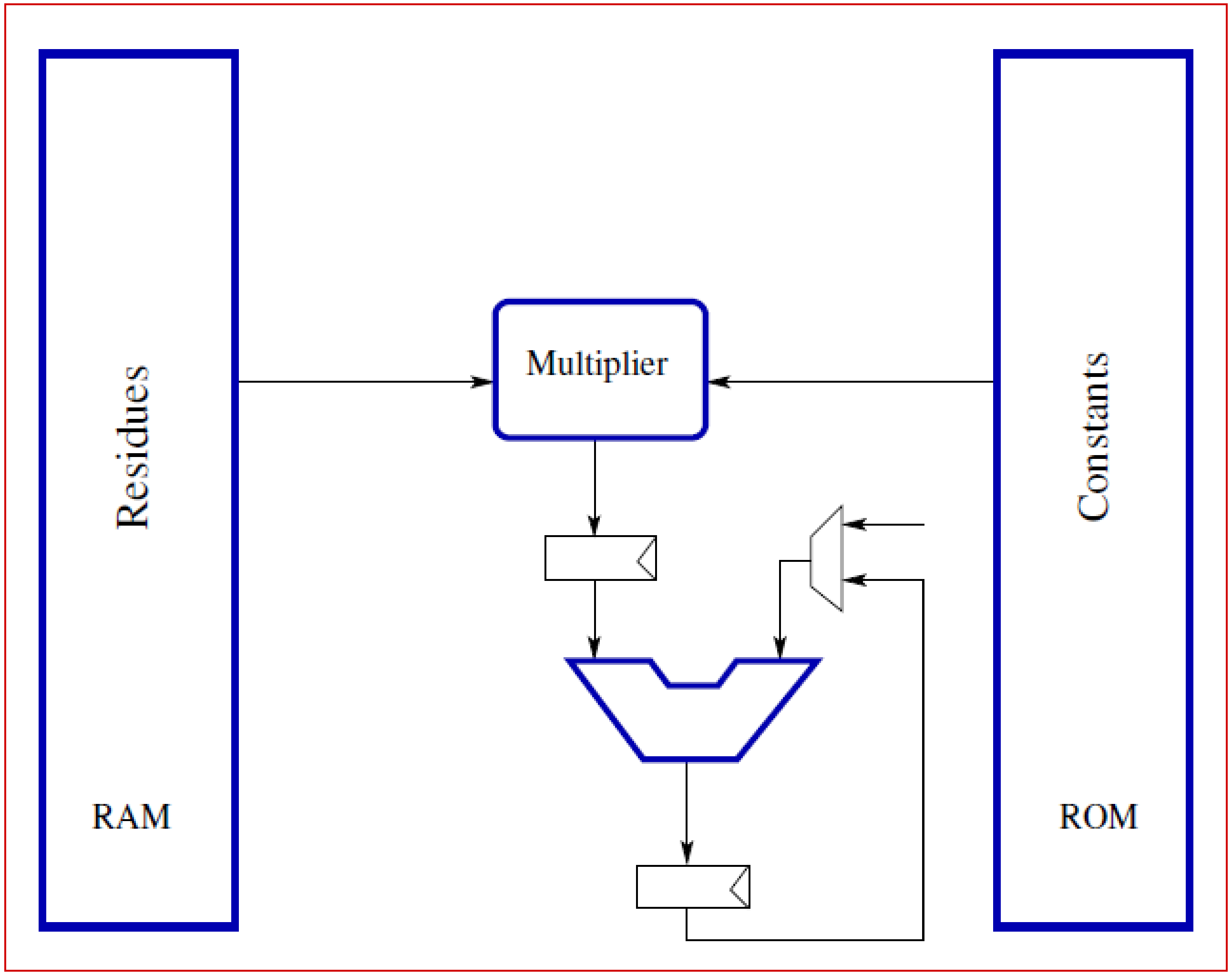
Small-CRT

Large-CRT

CRT Computation

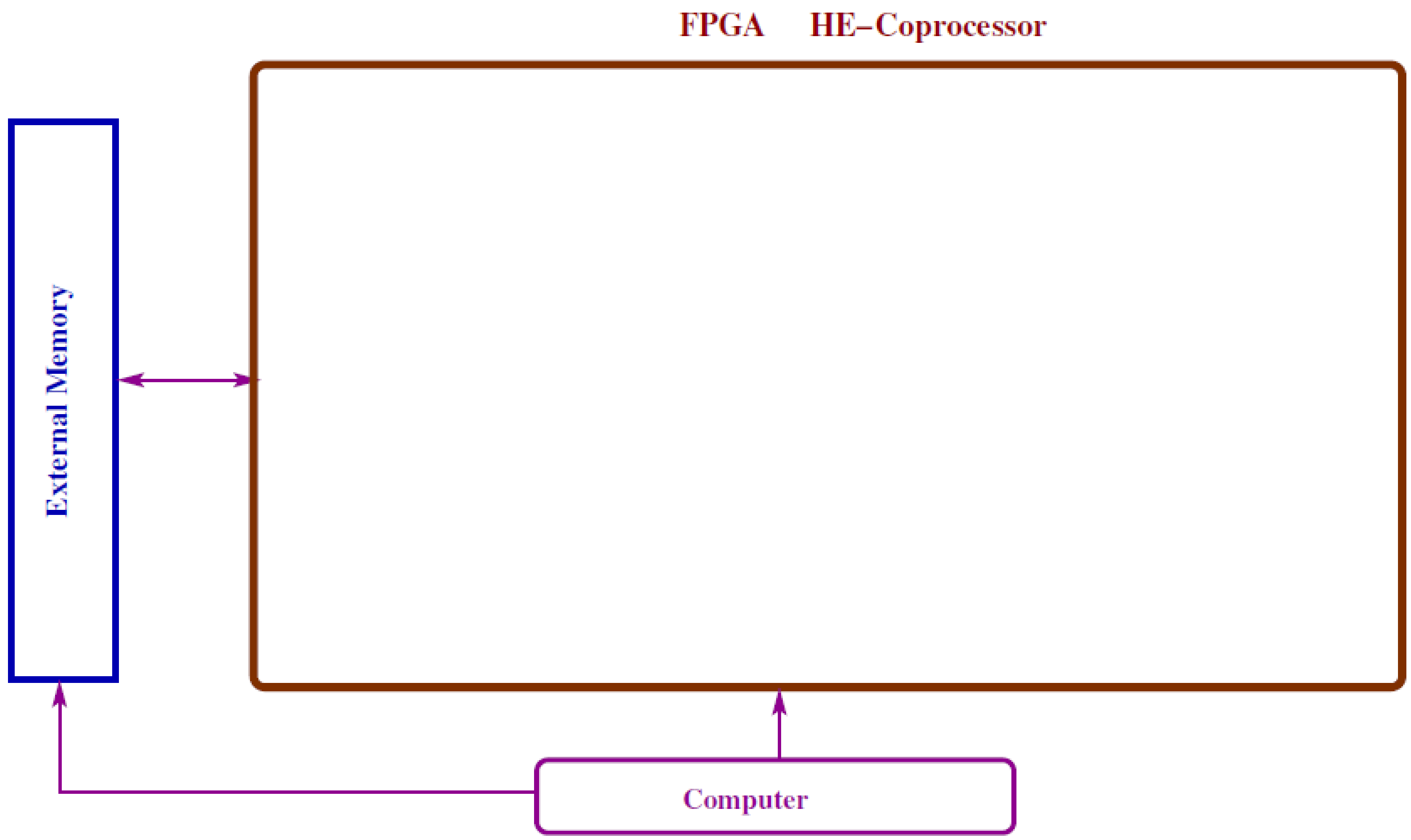
- Small CRT is required to map coefficients *c* from R_q to R_Q
- Computation involves
  - Sum of long and short products
  - Division in parallel

Sum of Product during CRT

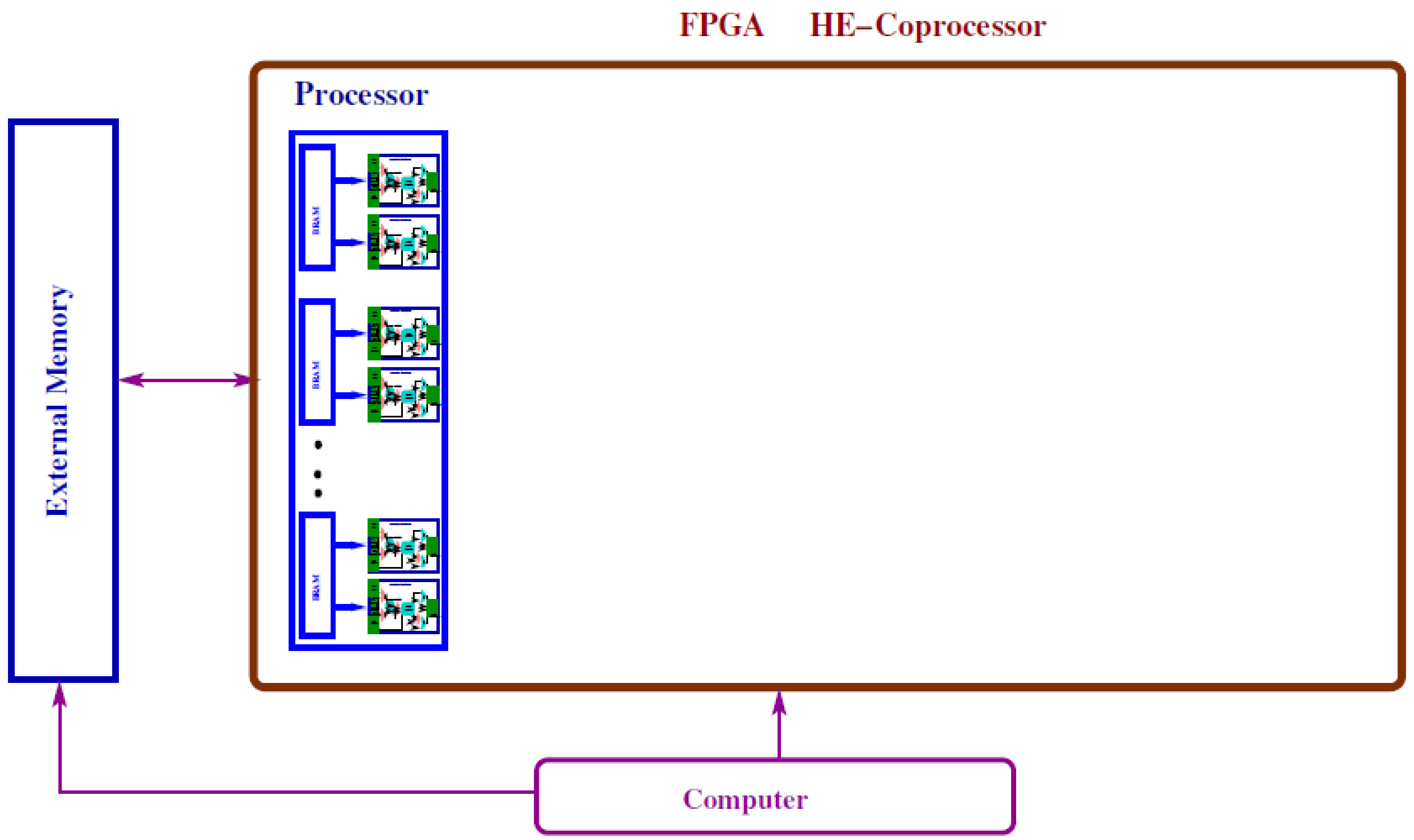


coming back to the overall architecture ….

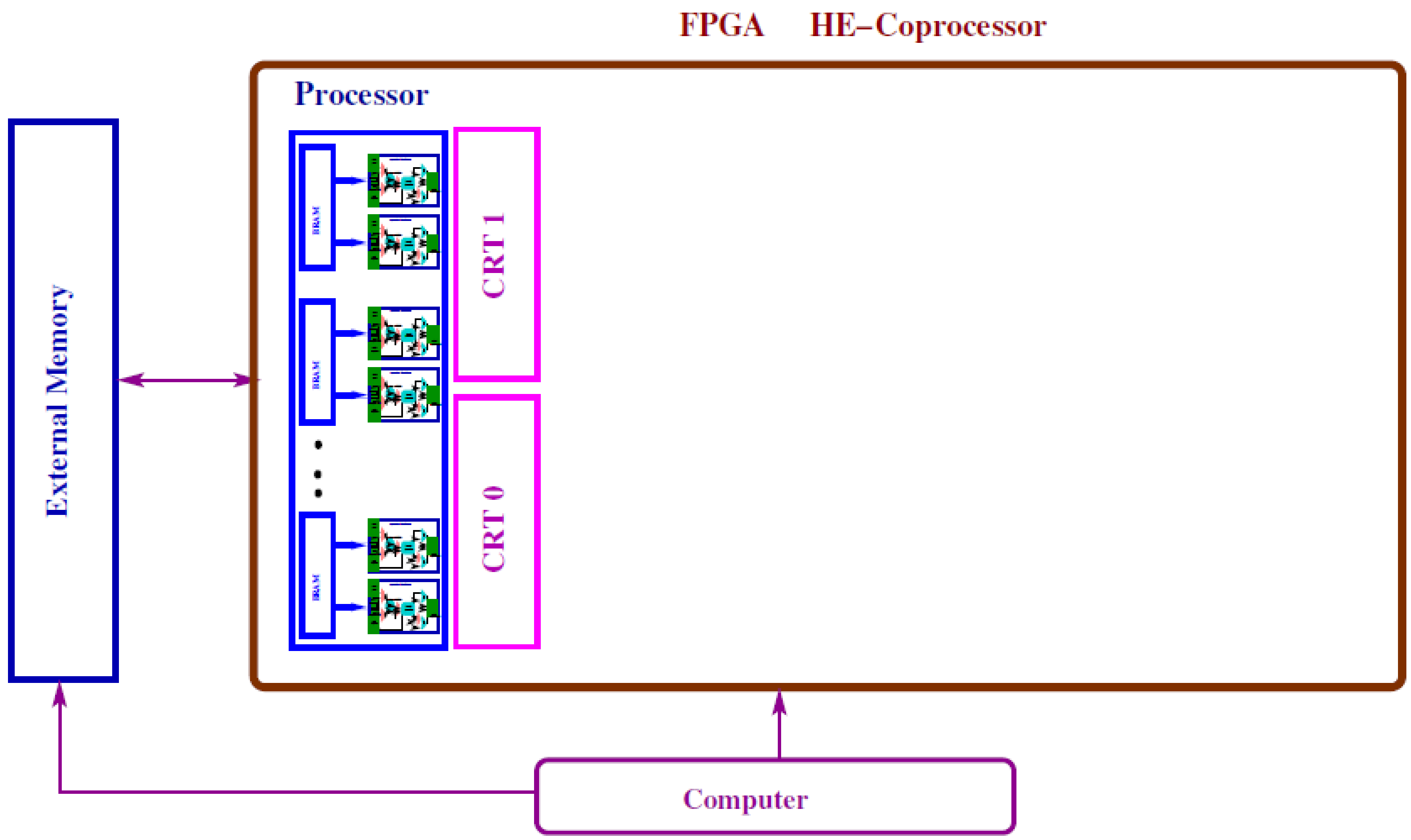
HE Architecture



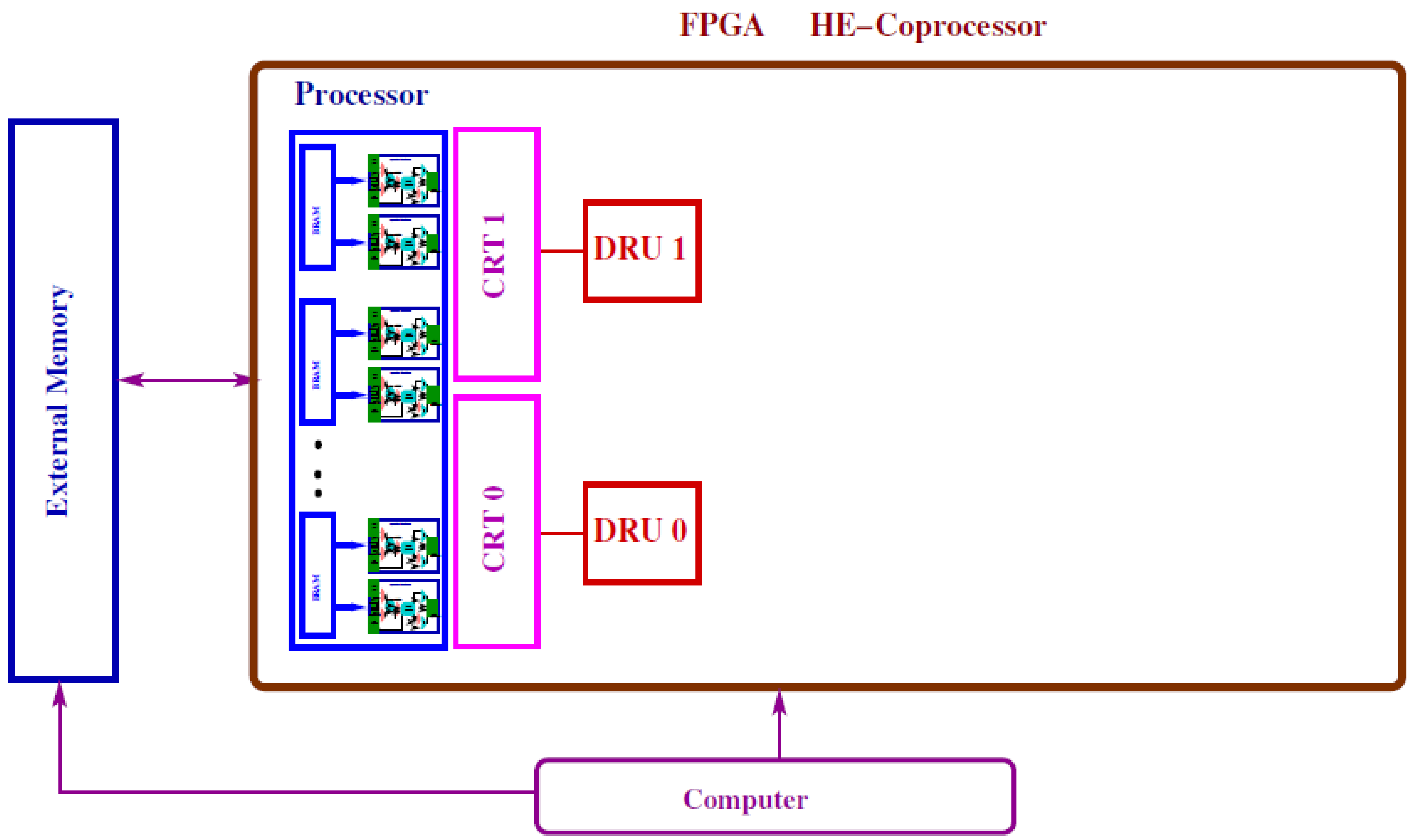
HE Architecture



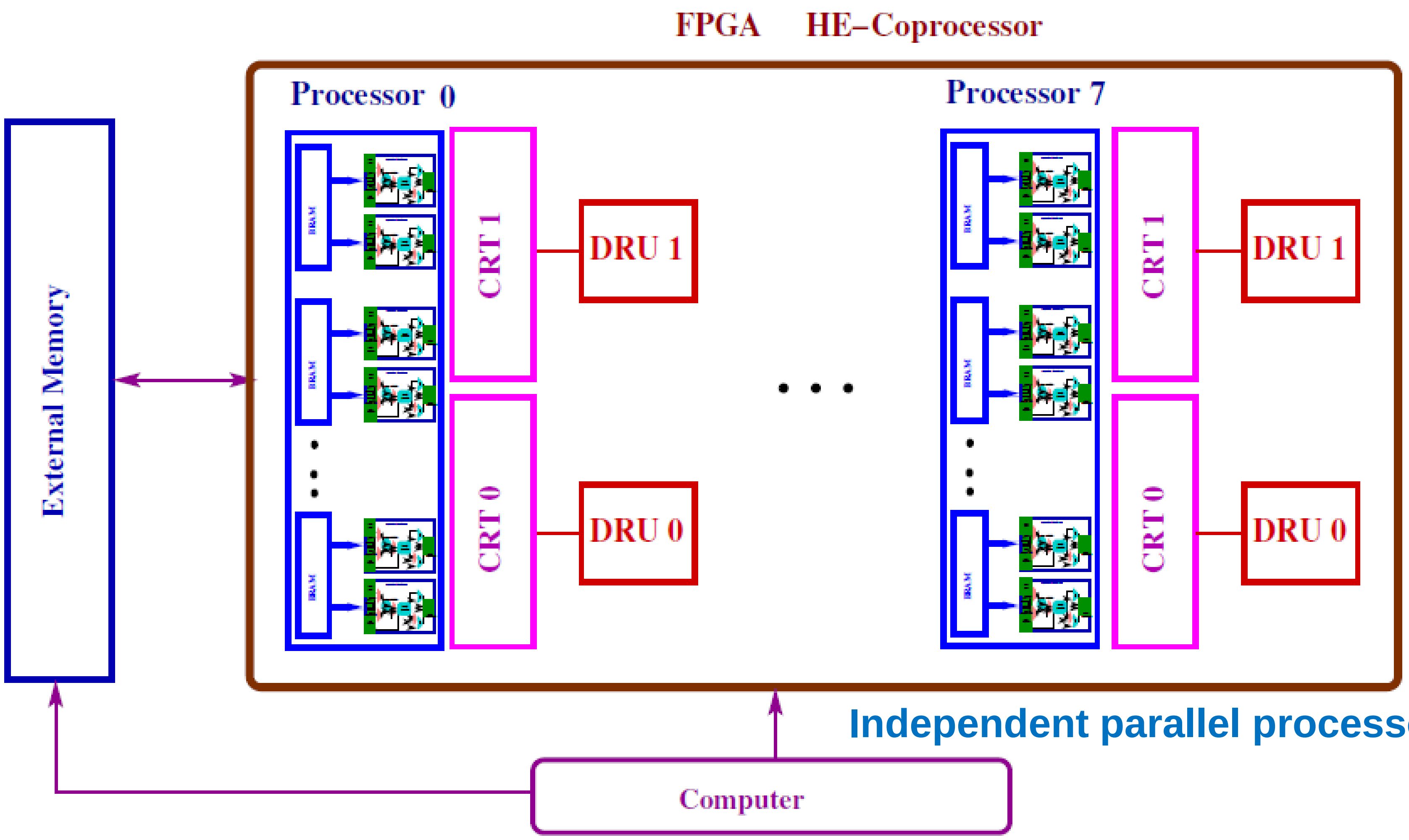
HE Architecture



HE Architecture



HE Architecture



Independent parallel processors

Results

Area Results

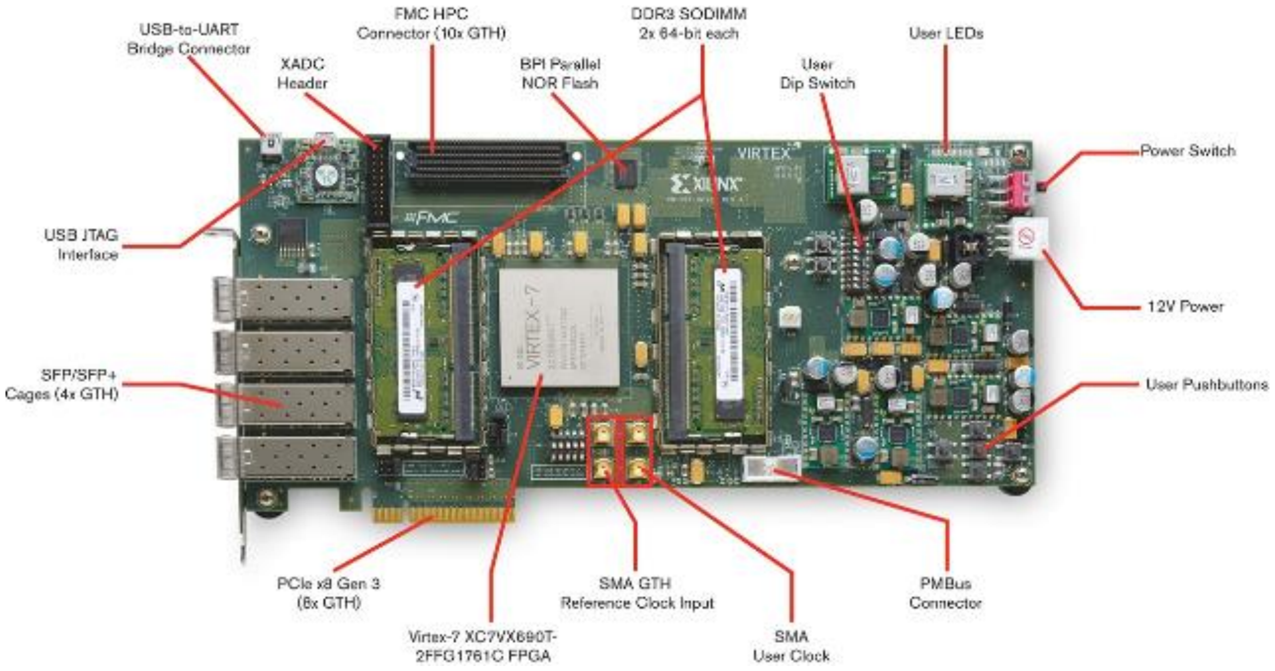
- We use the largest Virtex 7 FPGA *XCV1140TFLG1930*
- Resource consumption
  - ➢ FFs 22.6%
  - ➢ LUTs 53%
  - ➢ BRAMs 37.8%
  - ➢ DSPs 53%
- With more processors routing problem

Timing Results

- Does not include external memory--FPGA communication cost
- Operating frequency is 143 MHz after P&R
- YASHE.Mult requires 121.678 milliseconds
- SIMON-64/128 performs 32×44 YASHE.Mult operations
  - ➢ 171.3 seconds
- Relative time is per slot (2048 slots using SIMD)
  - ➢ 83.65 milliseconds

Future Works

- Implement interface between FPGA and external RAM
  - ➢ Serial data transfer is slow
  - ➢ Parallel 64-bit comm. between FPGA and external DDR3 RAM



Source: Xilinx Virtex-7 FPGA VC709 Connectivity Kit, www.xilinx.com

Future Works

- Architectual low-level optimization
  - ➢ Reduce pipeline bubbles [reduce cycles]
  - ➢ Increase frequency of sub blocks
  - ➢ Area optimization [more processors in FPGA]
- Higher level parallel processing
  - ➢ We have independent processors working in parallel
  - ➢ Hence more processors in several FPGAs

Thank You

Backup Slides

Homomorphic Encryption

- Enc(·,·) is homomorphic for an operation $\square$ on message space M iff

  $$\mathrm{Enc}(m_1 \,\square\, m_2, k_\mathrm{E}) = \mathrm{Enc}(m_1, k_\mathrm{E}) \circ \mathrm{Enc}(m_2, k_\mathrm{E})$$

  with $\circ$ operation on ciphertext space C
- Enc(·,·) is additively homomorphic is $\square = +$
  - eg. Caesar cipher
- Enc(·,·) is multiplicatively homomorphic is $\square = \times$
  - eg. Unpadded RSA

The YASHE Scheme

The YASHE Scheme

- Defined over a ring $R_q = \mathbb{Z}_q[\mathrm{x}]/\langle f \rangle$
- YASHE.KeyGen( )
  - (pk, sk, evk) where *pk* and *sk* $\in R_q$ and *evk* $\in R_q^{u+1}$
- YASHE.Enc (*m*, *pk*)
  - $m \in R_t$
  - $s, e \leftarrow \chi_{err}$
  - $c = \lfloor q/t \rfloor \cdot m + e + s \cdot pk \in R_q$
- YASHE.Dec(*c*, *sk*)
  - $m = \lfloor \frac{t}{q} \cdot [sk \cdot c]_q \rceil \in R_t$

The YASHE Scheme

- YASHE.Add (c_1, c_2)
  - Return $c = c_1 + c_2 \in R_q$
  - Requires one polynomial addition
- YASHE.Mult (c_1, c_2)
  - Compute *normal* polynomial multiplication $c_1 \cdot c_2$
  - Coefficients could be larger than q^2
  - Division and rounding $c' = \lfloor \frac{2}{q} c_1 c_2 \rceil$
  - Return $c = \mathsf{YASHE.KeySwitch}(c', evk) \in R_q$
  - $\mathsf{YASHE.KeySwitch}(\ )$ is u+1 poly mult and u poly add

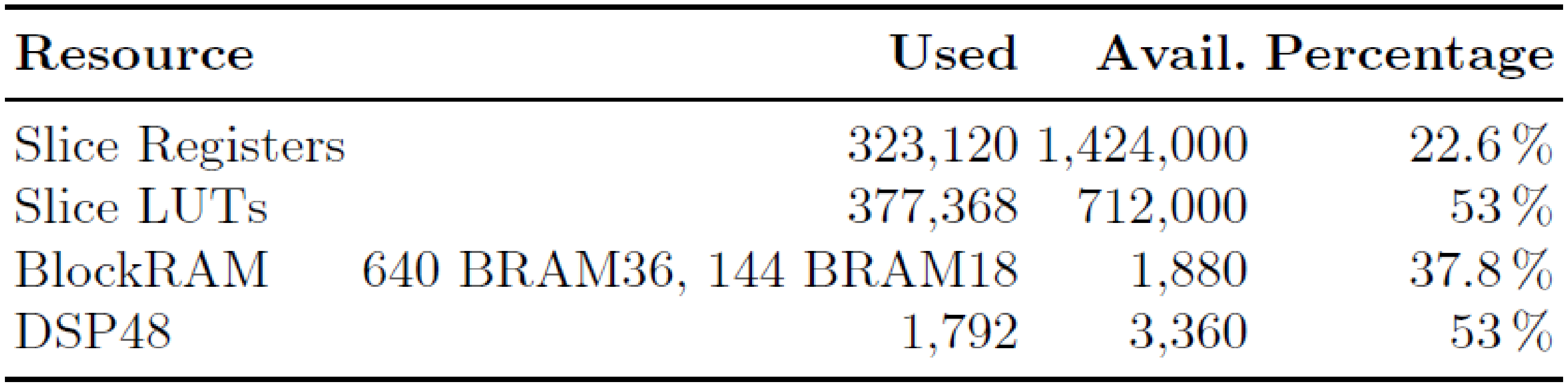
Small-CRT Computation

- Required to map polynomial coefficients *c* from R_q to R_Q
  - Remember $q = \prod_0^{l-1} q_i$ and $Q = \prod_0^{L-1} q_i$
- Compute $[c]q_j$ for l-1 < j < L
- First compute $c = ([c]q_0 \cdot b_0 + \ldots + [c]q_{l-1} \cdot b_{l-1})$ **[sum of *long* products]**
- Next $k = floor(c/q)$ **[division by q]**
- Next $[c']q_j = ([c]q_0 \cdot [b_0]q_j + \ldots + [c]q_{l-1} \cdot [b_{l-1}]q_j)$ **[sum of *short* products]**
- Finally $[c]q_j = [c']q_j - [k]q_i \cdot [q]q_j$

Area Results

- We use the largest Virtex 7 FPGA *XCV1140TFLG1930*

Table 1. The area results on Xilinx Virtex-7 XCV1140TFLG1930-2 FPGA



Resource	Used	Avail.	Percentage
Slice Registers	323,120	1,424,000	22.6 %
Slice LUTs	377,368	712,000	53 %
BlockRAM	640 BRAM36, 144 BRAM18	1,880	37.8 %
DSP48	1,792	3,360	53 %

- With more processors routing problem