KU LEUVEN

BASALISC: Programmable Hardware Accelerator for BGV Fully Homomorphic Encryption

Robin Geelen[1], **Michiel Van Beirendonck**[1]

H. V. L. Pereira[1], B. Huffman[2], T. McAuley[3], B. Selfridge[2], D. Wagner[2], G. Dimou[3], I. Verbauwhede[1], F. Vercauteren[1], D. W. Archer[2]

[1]COSIC KU Leuven, [2]Galois, Inc., and [3]Niobium Microsystems

CHES 2023, September 12

DARPA DPRIVE

- Dedicated ASIC acceleration of BGV
  - 150mm^2 12nm chip
  - 10,000$\times$ faster than software reference
  - 128-bit security
  - Support *bootstrapping*

DARPA DPRIVE

- Dedicated ASIC acceleration of BGV
  - 150mm^2 12nm chip
  - 10,000$\times$ faster than software reference
  - 128-bit security
  - Support *bootstrapping*
- Four teams of researchers
  - **Galois**, Duality, SRI, and Intel

galois

DARPA DPRIVE

- Dedicated ASIC acceleration of BGV
  - 150mm^2 12nm chip
  - 10,000$\times$ faster than software reference
  - 128-bit security
  - Support *bootstrapping*
- Four teams of researchers
  - **Galois**, Duality, SRI, and Intel
- Several phases
  - **Phase 1**: design, implementation and verification of system architecture and IP blocks

galois

DARPA DPRIVE

- Dedicated ASIC acceleration of BGV
  - 150mm^2 12nm chip
  - 10,000$\times$ faster than software reference
  - 128-bit security
  - Support *bootstrapping*
- Four teams of researchers
  - **Galois**, Duality, SRI, and Intel
- Several phases
  - **Phase 1**: design, implementation and verification of system architecture and IP blocks
  - **Phase 2**: software integration, hardware emulation, full system verification and validation, CKKS scheme

galois

Outline

Outline

BGV

- Ciphertexts are vectors of polynomials $c(X)$
  - Large polynomial multiplication through NTT: $C[k] = c(\omega^k)$
  - Large integer multiplication through RNS: $q = \prod q_i$

BGV

- Ciphertexts are vectors of polynomials $c(X)$
  - Large polynomial multiplication through NTT: $C[k] = c(\omega^k)$
  - Large integer multiplication through RNS: $q = \prod q_i$
    - 32-bit $q_i = c_i \cdot 2^k + 1$

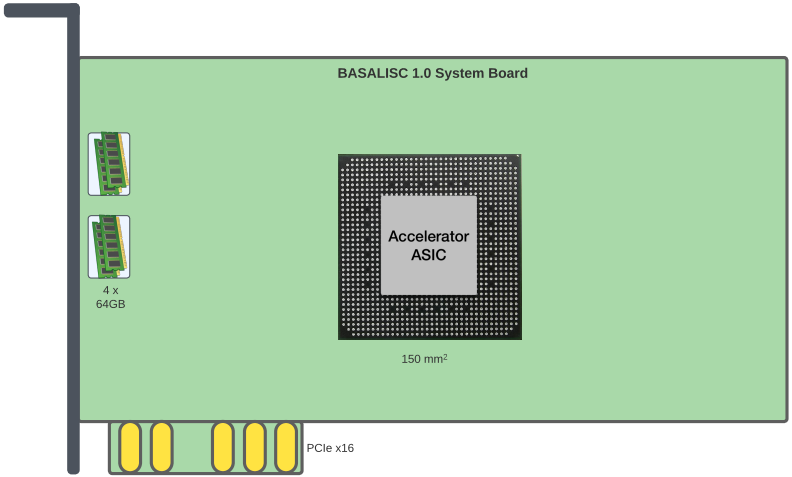
BGV

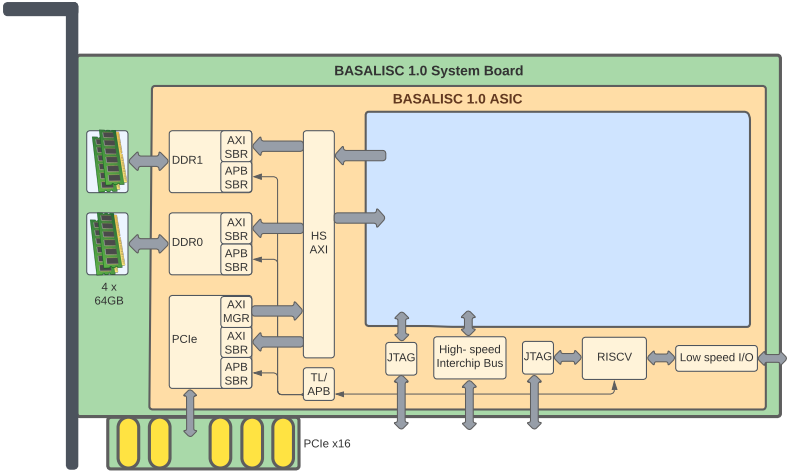
- Ciphertexts are vectors of polynomials $c(X)$
  - Large polynomial multiplication through NTT: $C[k] = c(\omega^k)$
  - Large integer multiplication through RNS: $q = \prod q_i$
    - 32-bit $q_i = c_i \cdot 2^k + 1$
- Basic Homomorphic Operations
  - Addition, multiplication, permutation
- Auxiliary Homomorphic Operations
  - Modulus switching, key switching
- Bootstrapping

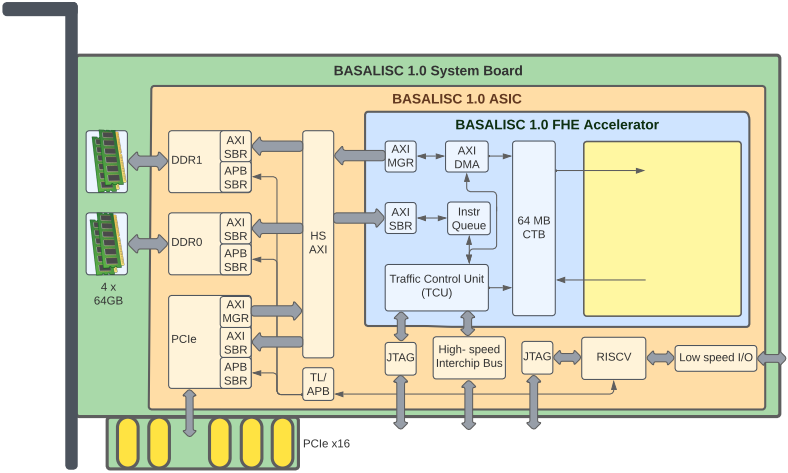
BGV 128-bit Security Parameters

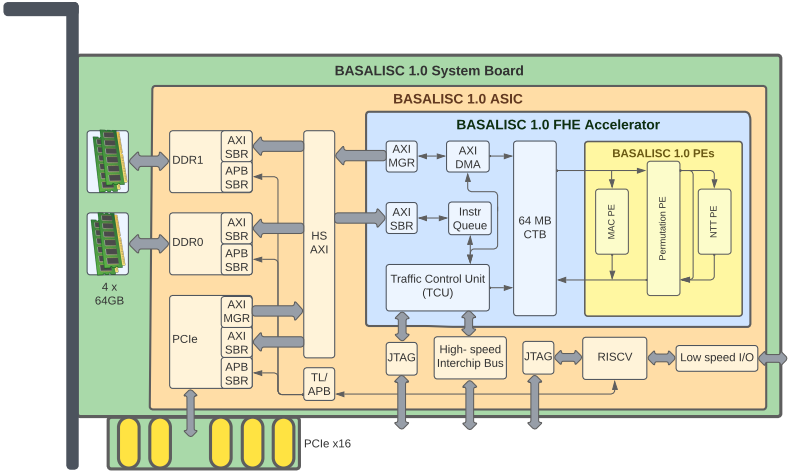
Parameter	Range	Example
Ring dimension N	$2^9 - 2^{16}$	$\mathbf{2^{16}}$
Ciphertext packing ℓ	$2 - 2^{16}$	64 slots
Max $\log_2(QP)$ for key switching	$20 - 1782$	**1782 bits**
Max $\log_2(Q)$ for ciphertext	$20 - 1782$	**1263 bits**
Max multiplicative depth L	N/A	31

Ciphertext: 21 MB, Key-switch key: 112 MB

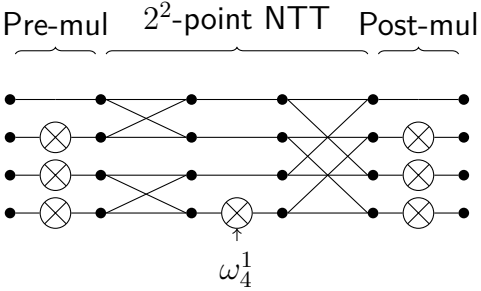

BASALISC 1.0 System Board
Accelerator ASIC
150 mm²
4 x 64GB
PCIe x16


BASALISC 1.0 System Board
BASALISC 1.0 ASIC
DDR1
DDR0
PCIe
AXI SBR
APB SBR
AXI MGR
HS AXI
TL/ APB
4 x 64GB
JTAG
High- speed Interchip Bus
RISCV
Low speed I/O
PCIe x16


BASALISC 1.0 System Board
BASALISC 1.0 ASIC
BASALISC 1.0 FHE Accelerator
DDR1
DDR0
AXI SBR
APB SBR
4 x 64GB
PCIe
AXI MGR
HS AXI
TL/ APB
AXI MGR
AXI DMA
AXI SBR
Instr Queue
64 MB CTB
Traffic Control Unit (TCU)
JTAG
High- speed Interchip Bus
JTAG
RISCV
Low speed I/O
PCIe x16


BASALISC 1.0 System Board
BASALISC 1.0 ASIC
BASALISC 1.0 FHE Accelerator
BASALISC 1.0 PEs
DDR1
DDR0
PCIe
AXI SBR
APB SBR
AXI MGR
4 x 64GB
HS AXI
TL/ APB
AXI DMA
Instr Queue
64 MB CTB
MAC PE
Permutation PE
NTT PE
Traffic Control Unit (TCU)
JTAG
High- speed Interchip Bus
RISCV
Low speed I/O
PCIe x16

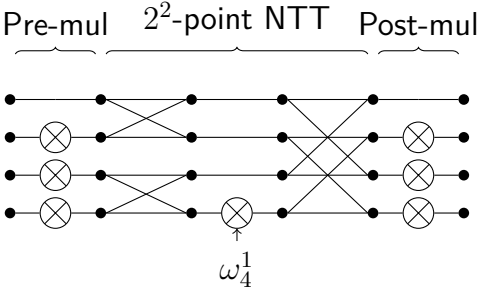
BASALISC High-Throughput NTT PE



Example radix-2^2 butterfly

BASALISC High-Throughput NTT PE

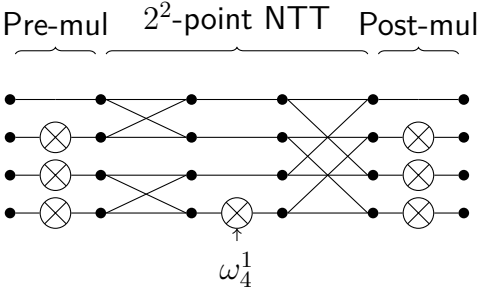
1. Radix-2^8 butterfly
   - 2^{16}-point NTT with 2 passes
   - 32 Tb/s NTT throughput



Example radix-2^2 butterfly

BASALISC High-Throughput NTT PE

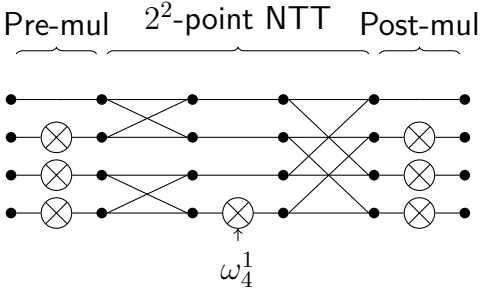
1. Radix-2^8 butterfly
   - 2^{16}-point NTT with 2 passes
   - 32 Tb/s NTT throughput
2. Twiddle-factor factory unit



Example radix-2^2 butterfly

BASALISC High-Throughput NTT PE

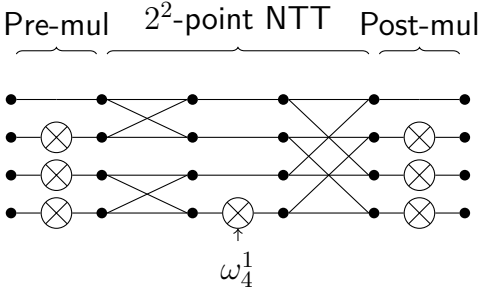
1. Radix-2^8 butterfly
   - 2^{16}-point NTT with 2 passes
   - 32 Tb/s NTT throughput
2. Twiddle-factor factory unit
3. Optimized multipliers
   - $q_i = c_i \cdot 2^k + 1$



Example radix-2^2 butterfly

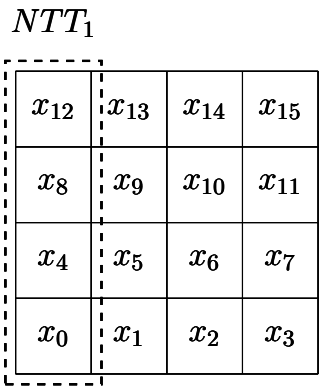
BASALISC High-Throughput NTT PE

1. Radix-2^8 butterfly
   - 2^{16}-point NTT with 2 passes
   - 32 Tb/s NTT throughput
2. Twiddle-factor factory unit
3. Optimized multipliers
   - $q_i = c_i \cdot 2^k + 1$
4. Conflict-free data layout

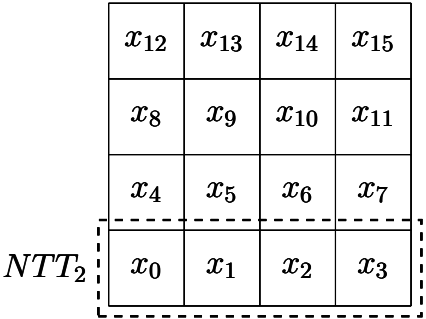


Example radix-2^2 butterfly

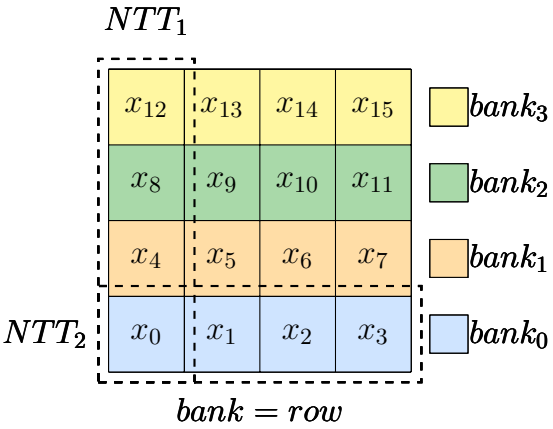
BASALISC Conflict-free CTB Layout



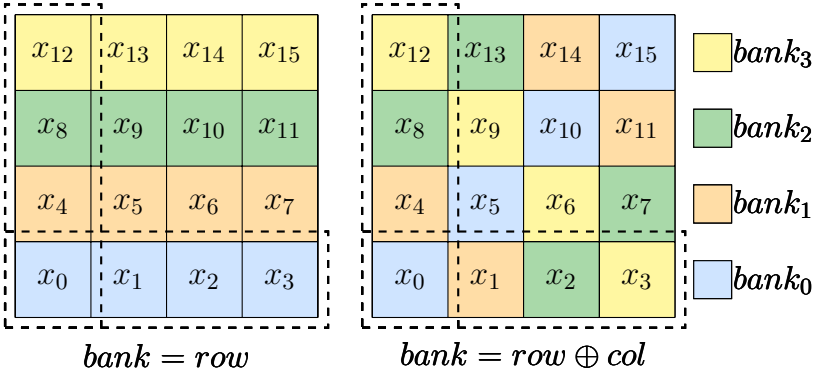
BASALISC Conflict-free CTB Layout



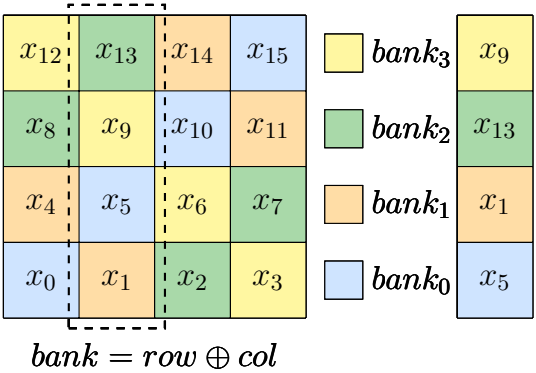
BASALISC Conflict-free CTB Layout



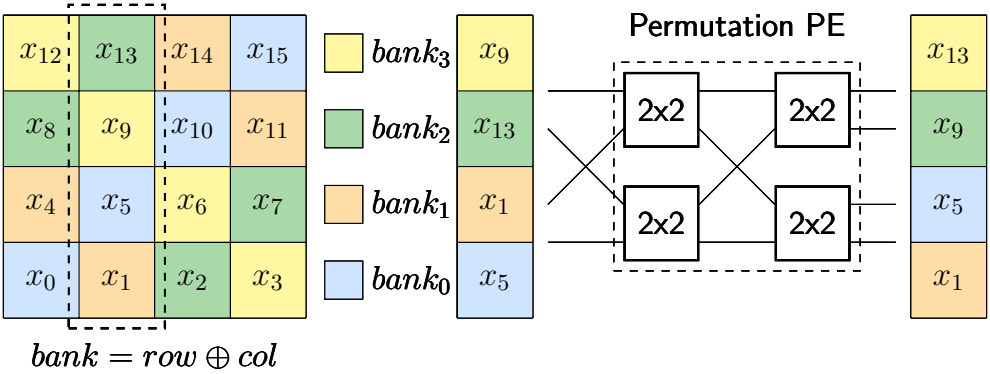
BASALISC Conflict-free CTB Layout



BASALISC Permutation PE

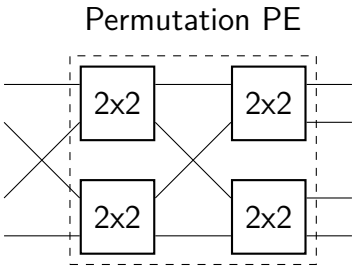


BASALISC Permutation PE



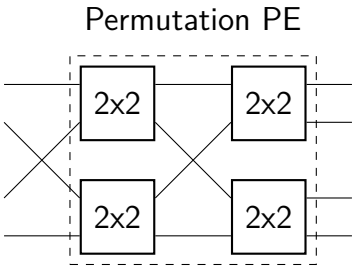
BASALISC Permutation PE

- Implements permutations: $i \mapsto i \oplus c$



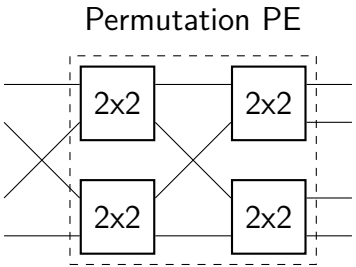
BASALISC Permutation PE

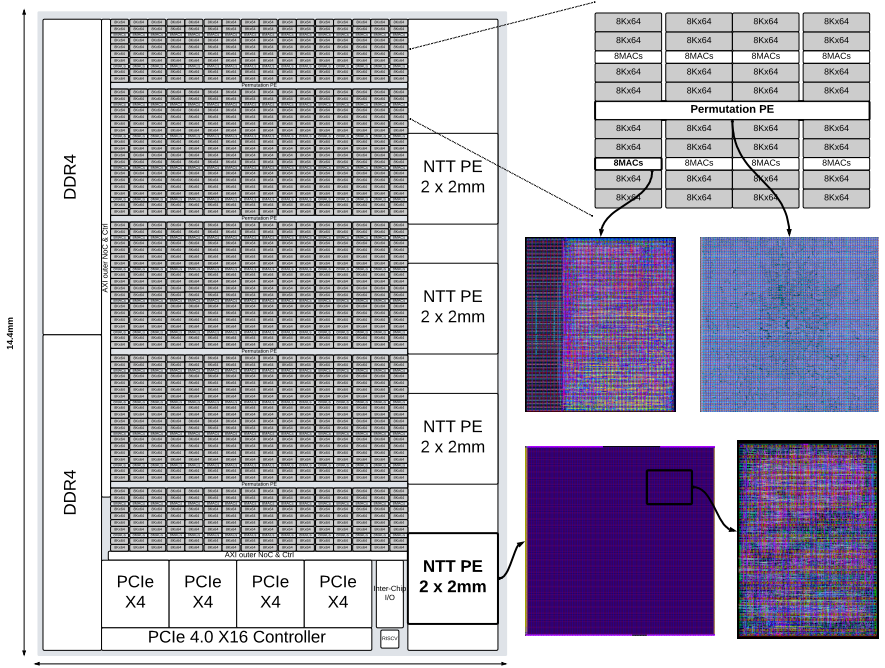
- Implements permutations: $i \mapsto i \oplus c$
- We slightly generalized: $i \mapsto (i \cdot a + b) \oplus c$



BASALISC Permutation PE

- Implements permutations: $i \mapsto i \oplus c$
- We slightly generalized: $i \mapsto (i \cdot a + b) \oplus c$
  - Supports BGV automorphisms: $\phi_k : c(X) \mapsto c(X^k)$ *with the same hardware*




14.4mm
10.4mm
DDR4
DDR4
AXI outer NoC & Ctrl
Permutation PE
NTT PE
2 x 2mm
NTT PE
2 x 2mm
NTT PE
2 x 2mm
NTT PE
2 x 2mm
AXI outer NoC & Ctrl
PCIe
X4
PCIe
X4
PCIe
X4
PCIe
X4
Inter-Chip
I/O
PCIe 4.0 X16 Controller
8Kx64
8MACs
Permutation PE

Outline

Bootstrapping Procedure

- **Homomorphic decryption** reduces noise

Bootstrapping Procedure

- **Homomorphic decryption** reduces noise
- Existing approaches (using odd prime p):
  - Ciphertext modulus $q = p^e + 1$
  - Exact division by p^k

Bootstrapping Procedure

- **Homomorphic decryption** reduces noise
- Existing approaches (using odd prime p):
  - Ciphertext modulus $q = p^e + 1$
  - Exact division by p^k
- **Problem**: Montgomery multipliers restricted to *NTT-friendly* primes
  - This means bits 16:1 are tied to 0 and bit 0 is tied to 1
  - But q and p^k are not NTT-friendly

NTT-friendly Bootstrapping

- Exclusive use of NTT-friendly primes
  - Ciphertext modulus $q = u \cdot p^e + 1$
  - No exact division by p^k

NTT-friendly Bootstrapping

- Exclusive use of NTT-friendly primes
  - Ciphertext modulus $q = u \cdot p^e + 1$
  - No exact division by p^k
- Simplified decryption function:

$(\boldsymbol{c}_0, \boldsymbol{c}_1)$

NTT-friendly Bootstrapping

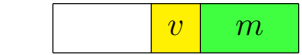
- Exclusive use of NTT-friendly primes
  - Ciphertext modulus $q = u \cdot p^e + 1$
  - No exact division by p^k
- Simplified decryption function:

$(\boldsymbol{c}_0, \boldsymbol{c}_1)$

$\boldsymbol{c}'_i \leftarrow [p^{e-r}\boldsymbol{c}_i]_q$

NTT-friendly Bootstrapping

- Exclusive use of NTT-friendly primes
  - Ciphertext modulus $q = u \cdot p^e + 1$
  - No exact division by p^k
- Simplified decryption function:



$(\boldsymbol{c}_0, \boldsymbol{c}_1)$

$\boldsymbol{c}'_i \leftarrow [p^{e-r}\boldsymbol{c}_i]_q$

$\boldsymbol{u} \leftarrow \boldsymbol{c}'_0 + \boldsymbol{c}'_1 \cdot \boldsymbol{s}$

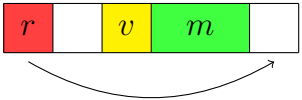
NTT-friendly Bootstrapping

- Exclusive use of NTT-friendly primes
  - Ciphertext modulus $q = u \cdot p^e + 1$
  - No exact division by p^k
- Simplified decryption function:

$(\boldsymbol{c}_0, \boldsymbol{c}_1)$

$\boldsymbol{w} \leftarrow [\boldsymbol{u}]_{p^e}$

$\boldsymbol{c}'_i \leftarrow [p^{e-r}\boldsymbol{c}_i]_q$



$\boldsymbol{u} \leftarrow \boldsymbol{c}'_0 + \boldsymbol{c}'_1 \cdot \boldsymbol{s}$

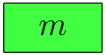
NTT-friendly Bootstrapping

- Exclusive use of NTT-friendly primes
  - Ciphertext modulus $q = u \cdot p^e + 1$
  - No exact division by p^k
- Simplified decryption function:

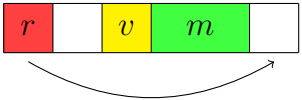
$(\boldsymbol{c}_0, \boldsymbol{c}_1)$

$\boldsymbol{w} \leftarrow [\boldsymbol{u}]_{p^e}$

$\boldsymbol{c}'_i \leftarrow [p^{e-r}\boldsymbol{c}_i]_q$



$\boldsymbol{m} \leftarrow [\lfloor \boldsymbol{w}/p^{e-r} \rceil]_{p^r}$



$\boldsymbol{u} \leftarrow \boldsymbol{c}'_0 + \boldsymbol{c}'_1 \cdot \boldsymbol{s}$

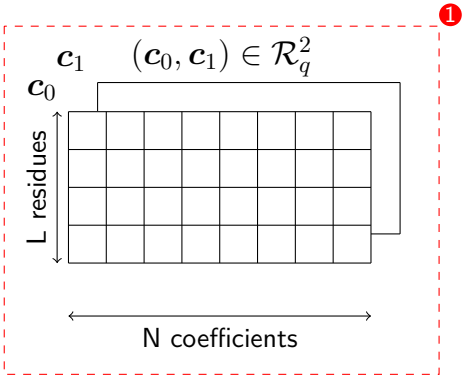
NTT-friendly Bootstrapping

- In the paper:
  - Generalization to arbitrary q
  - Correction factor management
  - Instantiation in RNS

Outline

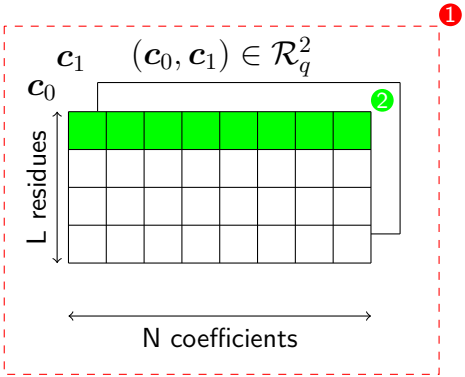
Software Toolchain

- Compiler for three-level instruction set:
  1. Macro-instructions: operations on ciphertexts and keys



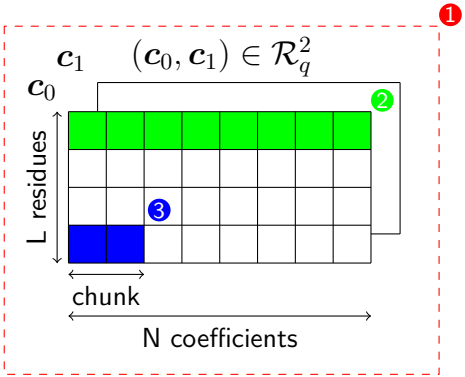
Software Toolchain

- Compiler for three-level instruction set:
  1. Macro-instructions: operations on ciphertexts and keys
  2. Mid-level instructions: operations on residue polynomials



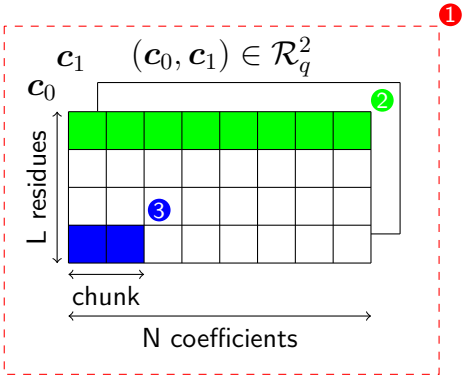
Software Toolchain

- Compiler for three-level instruction set:
  1. Macro-instructions: operations on ciphertexts and keys
  2. Mid-level instructions: operations on residue polynomials
  3. Micro-instructions: operations on smaller chunks of data



Software Toolchain

- Compiler for three-level instruction set:
  1. Macro-instructions: operations on ciphertexts and keys
  2. Mid-level instructions: operations on residue polynomials
  3. Micro-instructions: operations on smaller chunks of data
- Performance and correctness simulator: works at micro-level



Performance

- HElib on single-core Intel Xeon @2.6GHz
- BASALISC @1GHz via simulation

	HElib	BASALISC	Speedup
NTT	27 ms	11 μs	$2.5 \cdot 10^3\times$
Add/Sub	4 ms	8 μs	$5.0 \cdot 10^2\times$
Multiply	58 ms	20 μs	$2.9 \cdot 10^3\times$
Permutation	12 ms	11 μs	$1.1 \cdot 10^3\times$
Key switching	580 ms	292 μs	$2.0 \cdot 10^3\times$

Performance

- HElib on single-core Intel Xeon @2.6GHz
- BASALISC @1GHz via simulation

	HElib	BASALISC	Speedup
NTT	27 ms	11 μs	$2.5 \cdot 10^3\times$
Add/Sub	4 ms	8 μs	$5.0 \cdot 10^2\times$
Multiply	58 ms	20 μs	$2.9 \cdot 10^3\times$
Permutation	12 ms	11 μs	$1.1 \cdot 10^3\times$
Key switching	580 ms	292 μs	$2.0 \cdot 10^3\times$
Bootstrapping	160 s	40 ms	$4.0 \cdot 10^3\times$
DB lookup	2,325 s	267 ms	$8.7 \cdot 10^3\times$
LR training	217,000 s	40,500 ms	$5.4 \cdot 10^3\times$

Comparison with Different Accelerators

	BASALISC	F1	BTS	CraterLake
Scheme	**BGV**	BGV/CKKS	CKKS	CKKS
Area	**150mm^2**	150mm^2	374mm^2	472mm^2
Technology	**12nm**	12/14nm	7nm	12/14nm
Power	**115W**	180W	163W	320W
Bootstrapping speedup	**4,000**×	1,830×/1,195×	2,237×	4,400×
LR speedup	**5,400**×	—/7,200×	1,306×	2,978×

Outline

Changes Since End of DPRIVE Phase 1

- Support two schemes: CKKS and BGV
- Change from 32-bit to 64-bit q_i in RNS
  - Accommodate precision requirements in CKKS
  - Fewer NTT units due to increased area

Changes Since End of DPRIVE Phase 1

- Support two schemes: CKKS and BGV
- Change from 32-bit to 64-bit q_i in RNS
  - Accommodate precision requirements in CKKS
  - Fewer NTT units due to increased area
- On-chip PRNG eliminates 50% of key-switch memory bandwidth

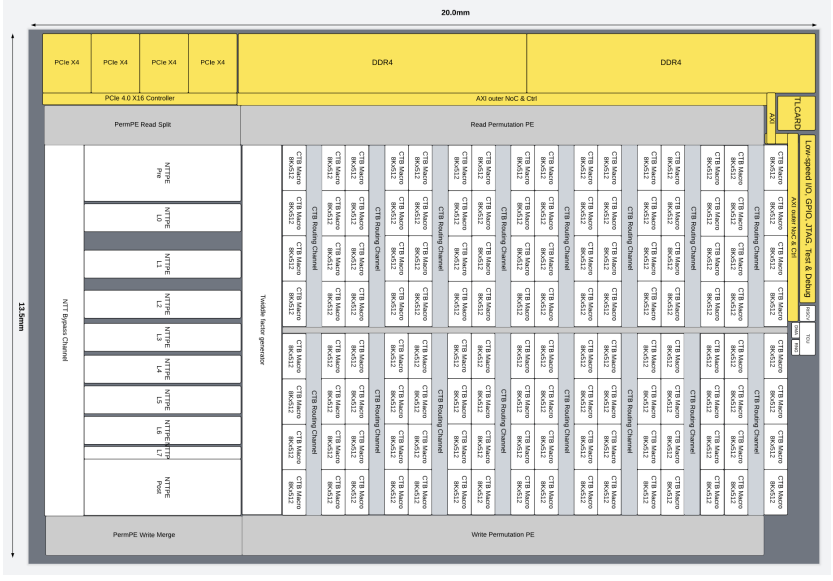
Changes Since End of DPRIVE Phase 1

- Support two schemes: CKKS and BGV
- Change from 32-bit to 64-bit q_i in RNS
  - Accommodate precision requirements in CKKS
  - Fewer NTT units due to increased area
- On-chip PRNG eliminates 50% of key-switch memory bandwidth
- Increased die size 270mm^2 = 20mm $\times$ 13.5mm

Current Status of BASALISC

- Floorplan and place and route complete
- On track for tape-out by November 30
- Verification tests in progress
- Sign-off flow on track

Latest BASALISC Floorplan



Conclusion

- New ASIC architecture for the BGV and CKKS schemes
- Combines efficiency with programmability
  - Novel processing elements
  - Bootstrapping enables optimized multipliers
  - Software toolchain
- Tape-out by the end of this year

Thank you for your attention!

robin.geelen@esat.kuleuven.be
michiel.vanbeirendonck@esat.kuleuven.be

BASALISC Die Plot