SCIENCE
PASSION
TECHNOLOGY

Aloha-HE

A Low-Area Hardware Accelerator for Client-Side Operations in Homomorphic Encryption

Florian Krieger, Florian Hirner, Ahmet Can Mert, Sujoy Sinha Roy
Institute of Applied Information Processing and Communications, TU Graz

January 31, 2024

Outline

Table of Contents

Homomorphic Encryption (in short HE)

- Special encryption technique
- Computations directly on encrypted data
  - Confidential cloud computing
  - Sensitive data processing on untrusted third parties
  - Novel opportunities

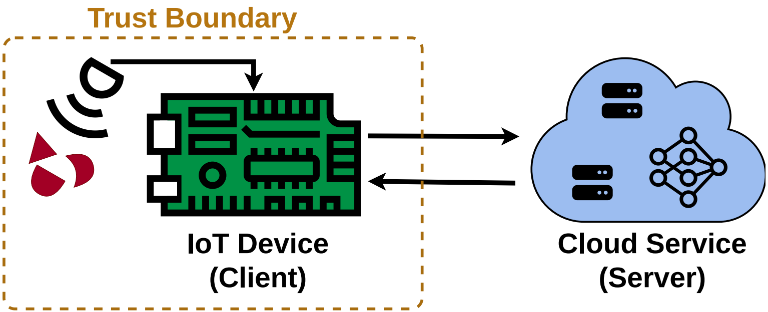
Homomorphic Encryption (in short HE)

- Special encryption technique
- Computations directly on encrypted data
  - Confidential cloud computing
  - Sensitive data processing on untrusted third parties
  - Novel opportunities

Homomorphic Encryption (in short HE)

- Special encryption technique
- Computations directly on encrypted data
  - Confidential cloud computing
  - Sensitive data processing on untrusted third parties
  - Novel opportunities

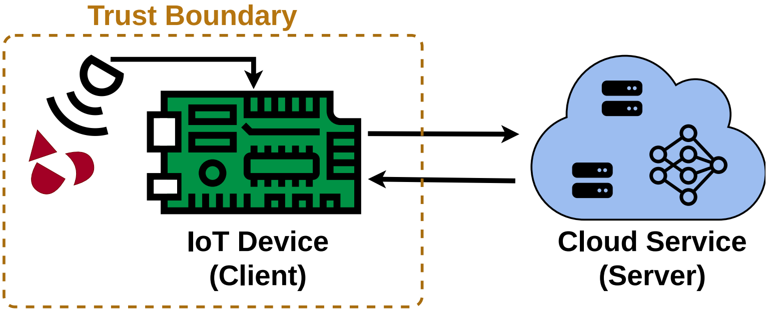
Homomorphic Encryption: An Example



- Encrypts sensitive data
- Sends it to cloud

- Computes on encrypted data
  - E.g.: AI application
- Returns result to client

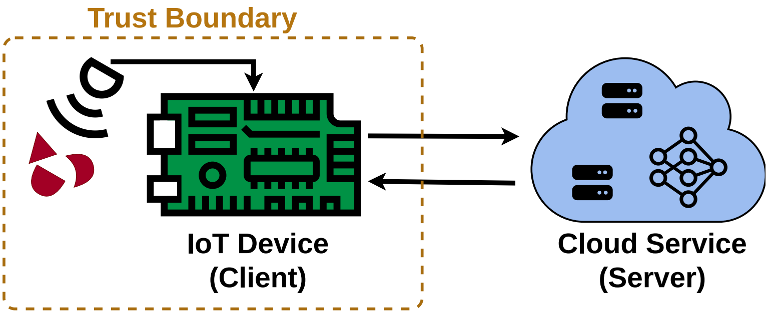
Homomorphic Encryption: An Example



- Encrypts sensitive data
- Sends it to cloud

- Computes on encrypted data
  - E.g.: AI application
- Returns result to client

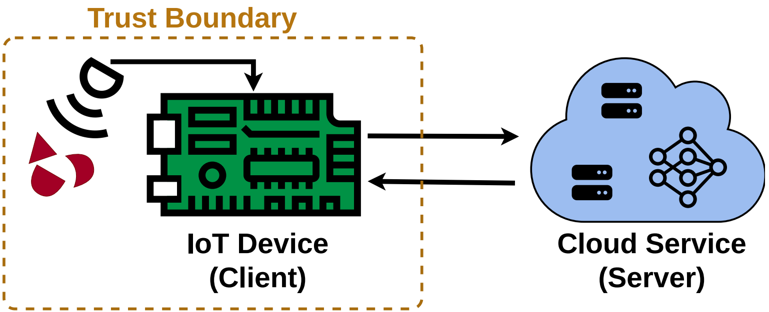
Homomorphic Encryption: An Example



- Encrypts sensitive data
- Sends it to cloud

- Computes on encrypted data
  - E.g.: AI application
- Returns result to client

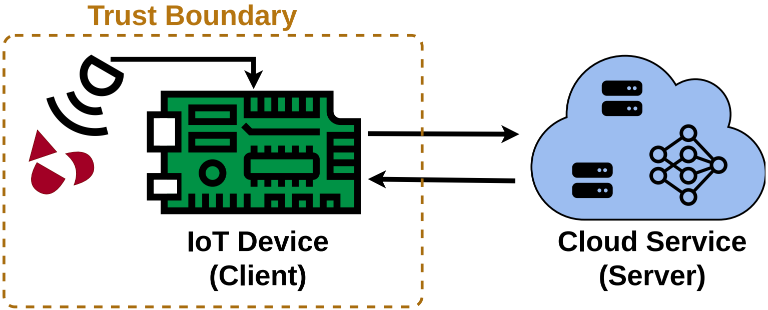
Homomorphic Encryption: An Example



- Encrypts sensitive data
- Sends it to cloud

- Computes on encrypted data
  - E.g.: AI application
- Returns result to client

Homomorphic Encryption: An Example



- Encrypts sensitive data
- Sends it to cloud

- Computes on encrypted data
  - E.g.: AI application
- Returns result to client

Homomorphic Encryption: The Drawback

- Huge computational overhead
  - Large polynomials
- Simple computations take seconds
  - Hardens practical adoption

Homomorphic Encryption: The Drawback

- Huge computational overhead
  - Large polynomials
- Simple computations take seconds
  - Hardens practical adoption

Homomorphic Encryption: The Drawback

- Huge computational overhead
  - ➔ Large polynomials
- Simple computations take seconds
  - ➔ Hardens practical adoption

Homomorphic Encryption: The Drawback

- Huge computational overhead
  - ➔ Large polynomials
- Simple computations take seconds
  - ➔ Hardens practical adoption

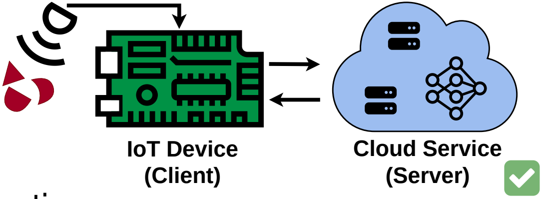
Homomorphic Encryption: The Drawback

- Huge computational overhead
  - Large polynomials
- Simple computations take seconds
  - Hardens practical adoption

Hardware acceleration!

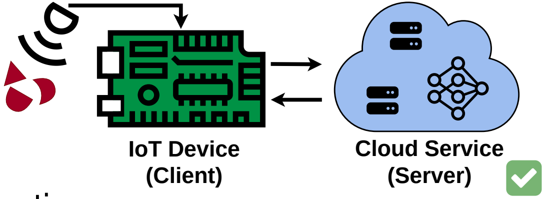
State of the art in accelerating HE

State of the art in accelerating HE



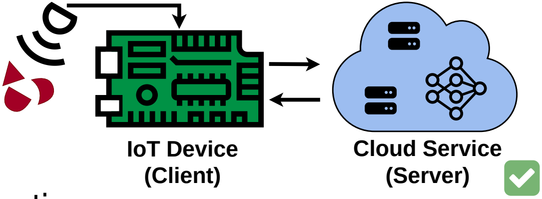
- Strong focus on server-side operations
  - Usually the limiting factor
- **But:** Client-side is also crucial in constrained scenarios

State of the art in accelerating HE



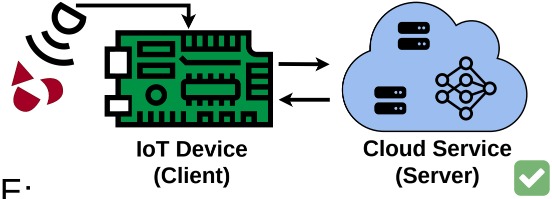
- Strong focus on server-side operations
  - ➔ Usually the limiting factor
- **But:** Client-side is also crucial in constrained scenarios

State of the art in accelerating HE



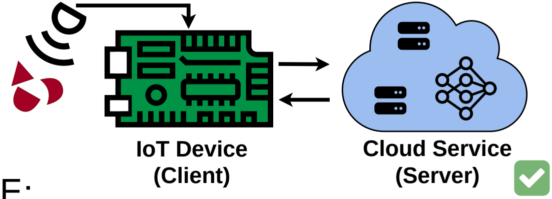
- Strong focus on server-side operations
  - ➔ Usually the limiting factor
- **But:** Client-side is also crucial in constrained scenarios

State of the art in accelerating HE



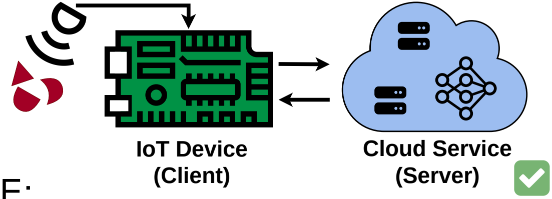
- Consider IoT application with HE:
  - Timely tasks require fast encryption/decryption
  - Limited energy budget
  - Low-cost devices
- Only few works targeting client-side operations
  - ➔ Insufficient hardware support

State of the art in accelerating HE



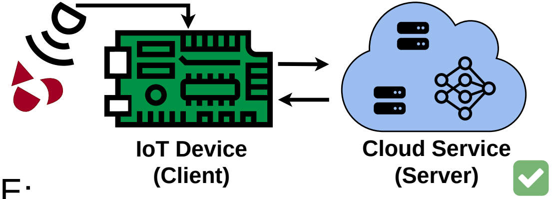
- Consider IoT application with HE:
  - Timely tasks require fast encryption/decryption
  - Limited energy budget
  - Low-cost devices
- Only few works targeting client-side operations
  - Insufficient hardware support

State of the art in accelerating HE



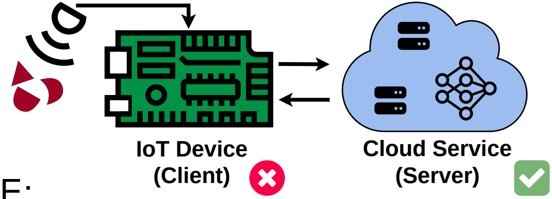
- Consider IoT application with HE:
  - Timely tasks require fast encryption/decryption
  - Limited energy budget
  - Low-cost devices
- Only few works targeting client-side operations
  - Insufficient hardware support

State of the art in accelerating HE



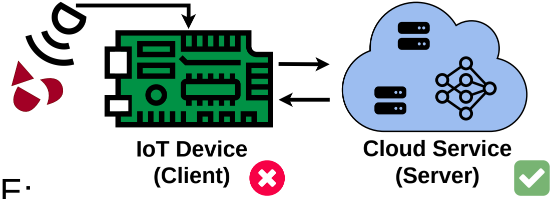
- Consider IoT application with HE:
  - Timely tasks require fast encryption/decryption
  - Limited energy budget
  - Low-cost devices
- Only few works targeting client-side operations
  - Insufficient hardware support

State of the art in accelerating HE



- Consider IoT application with HE:
  - Timely tasks require fast encryption/decryption
  - Limited energy budget
  - Low-cost devices
- Only few works targeting client-side operations
  - ➔ Insufficient hardware support

State of the art in accelerating HE



- Consider IoT application with HE:
  - Timely tasks require fast encryption/decryption
  - Limited energy budget
  - Low-cost devices

} Motivation for Aloha-HE

- Only few works targeting client-side operations
  - ➔ Insufficient hardware support

The CKKS Scheme

The CKKS Scheme

- State-of-the-art homomorphic encryption scheme
- Allows computations on complex numbers ($\mathbb{C}$)
  - Required in machine learning applications
- Software support such as in Microsoft SEAL library

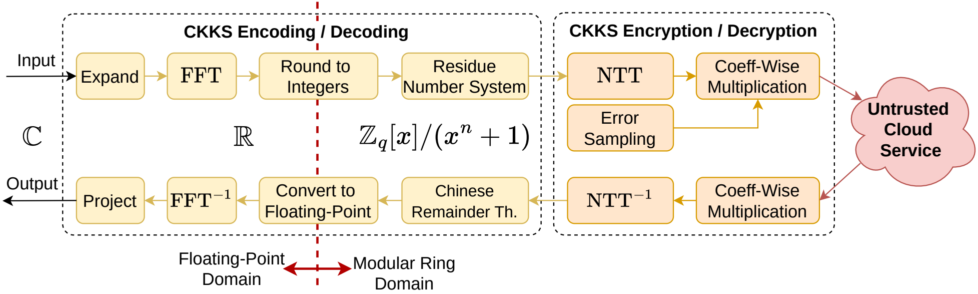
The CKKS Scheme

- State-of-the-art homomorphic encryption scheme
- Allows computations on complex numbers ($\mathbb{C}$)
  - ➔ Required in machine learning applications
- Software support such as in Microsoft SEAL library

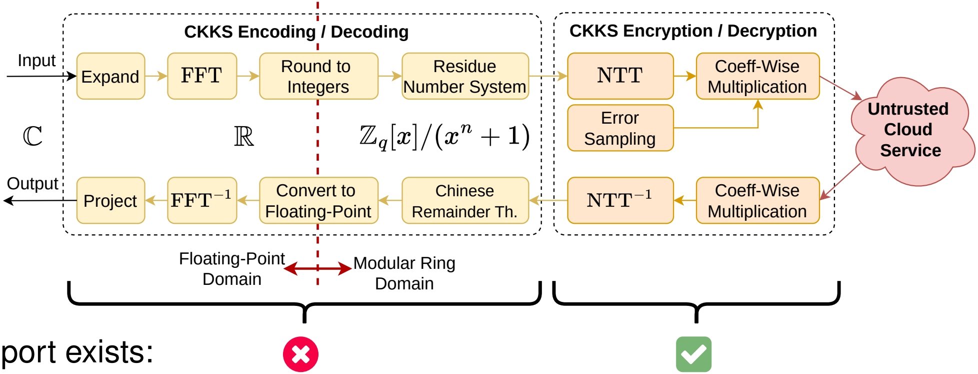
The CKKS Scheme

- State-of-the-art homomorphic encryption scheme
- Allows computations on complex numbers ($\mathbb{C}$)
  - ➔ Required in machine learning applications
- Software support such as in Microsoft SEAL library

The CKKS Scheme: Operations involved



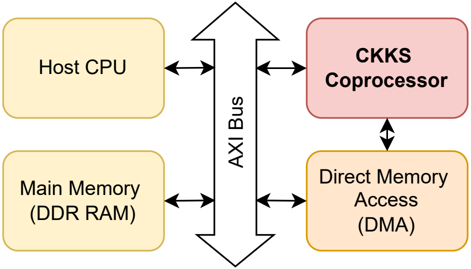
The CKKS Scheme: Operations involved



HW support exists:

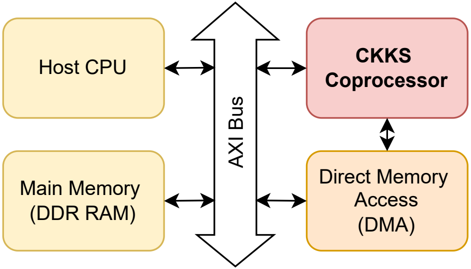
Table of Contents

Overall Architecture



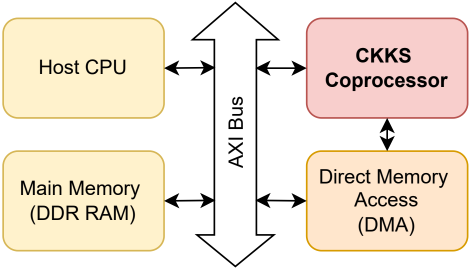
- CKKS coprocessor performing all client-side operations
- DMA for data streaming
- CPU controls DMA and coprocessor
  - Instruction set based

Overall Architecture



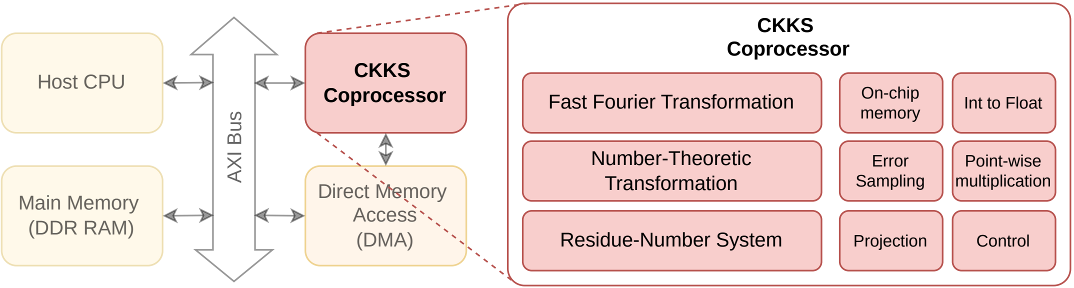
- CKKS coprocessor performing all client-side operations
- DMA for data streaming
- CPU controls DMA and coprocessor
  - Instruction set based

Overall Architecture

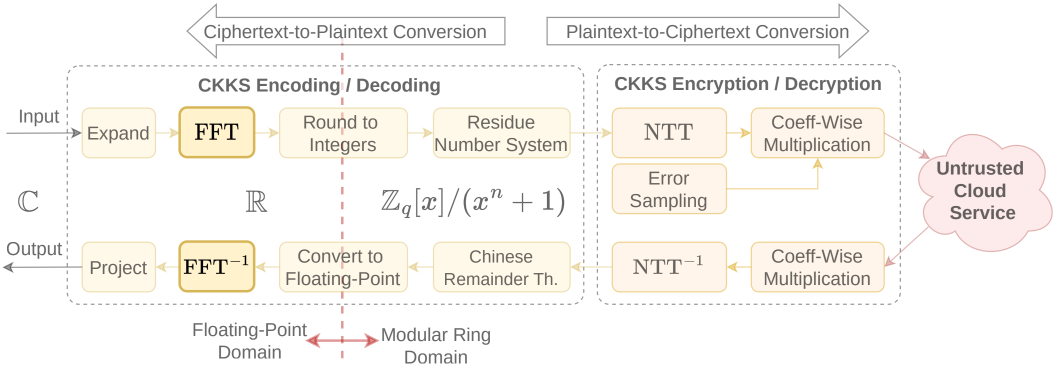


- CKKS coprocessor performing all client-side operations
- DMA for data streaming
- CPU controls DMA and coprocessor
  - Instruction set based

Overall Architecture

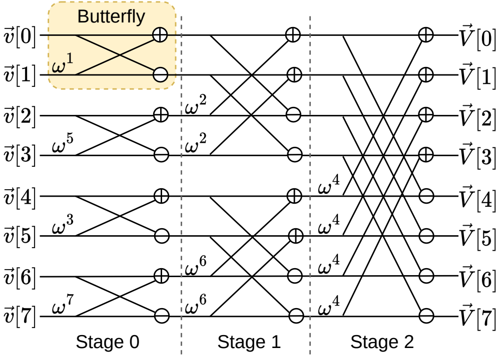


Fast Fourier Transformation



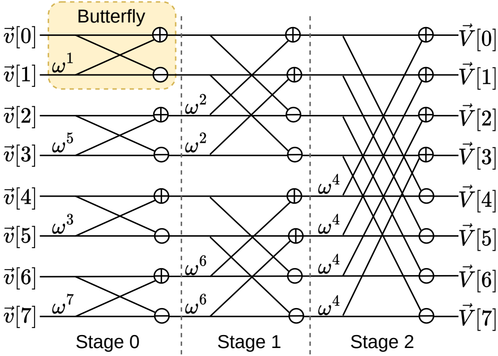
Fast Fourier Transformation: Data Flow

- Basic operation: Butterfly
- Iterated within $\log_2(n)$ stages
- Requires twiddle factors ω^i
  - Roots of unity in $\mathbb{C}$



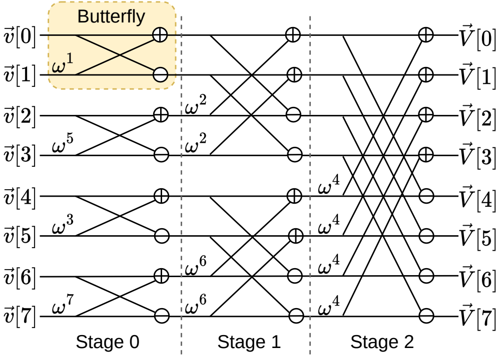
Fast Fourier Transformation: Data Flow

- Basic operation: Butterfly
- Iterated within $\log_2(n)$ stages
- Requires twiddle factors ω^i
  - Roots of unity in $\mathbb{C}$



Fast Fourier Transformation: Data Flow

- Basic operation: Butterfly
- Iterated within $\log_2(n)$ stages
- Requires twiddle factors ω^i
  - Roots of unity in $\mathbb{C}$



Fast Fourier Transformation: Complex Arithmetic

- Complex number arithmetic: Requires a floating-point unit!
  - ➔ Challenging to efficiently implement FPU
  - Needs to support complex addition and multiplication
  - Must fit on small area
  - Optimized for FFT computation

Fast Fourier Transformation: Complex Arithmetic

- Complex number arithmetic: Requires a floating-point unit!
  - ➔ Challenging to efficiently implement FPU
  - Needs to support complex addition and multiplication
  - Must fit on small area
  - Optimized for FFT computation

Fast Fourier Transformation: Complex Arithmetic

- Complex number arithmetic: Requires a floating-point unit!
  - ➔ Challenging to efficiently implement FPU
  - Needs to support complex addition and multiplication
  - Must fit on small area
  - Optimized for FFT computation

Fast Fourier Transformation: Complex Arithmetic

- Complex number arithmetic: Requires a floating-point unit!
  - ➔ Challenging to efficiently implement FPU
  - Needs to support complex addition and multiplication
  - Must fit on small area
  - Optimized for FFT computation

Fast Fourier Transformation: Optimizations

- Sharing of resources within the FFT butterfly
  - ➔ lowering area consumption
- Stored *or* on-the-fly generated twiddle factors
  - ➔ increasing flexibility
- Reducing the number of stored twiddle factors by 75%
  - ➔ lowering BRAM consumption

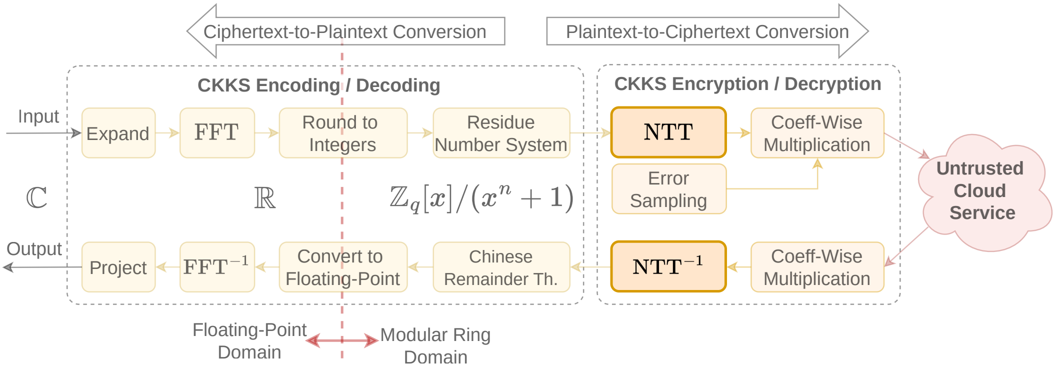
Fast Fourier Transformation: Optimizations

- Sharing of resources within the FFT butterfly
  - ➔ lowering area consumption
- Stored *or* on-the-fly generated twiddle factors
  - ➔ increasing flexibility
- Reducing the number of stored twiddle factors by 75%
  - ➔ lowering BRAM consumption

Fast Fourier Transformation: Optimizations

- Sharing of resources within the FFT butterfly
  - ➔ lowering area consumption
- Stored *or* on-the-fly generated twiddle factors
  - ➔ increasing flexibility
- Reducing the number of stored twiddle factors by 75%
  - ➔ lowering BRAM consumption

Number-Theoretic Transformation



Number-Theoretic Transformation

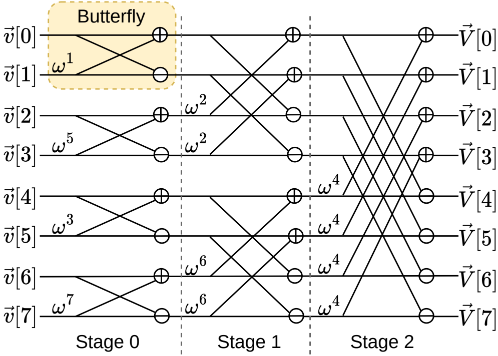
- Operates over modular ring domain $\mathbb{Z}_q$
  - Ring arithmetic modulo prime q
- Special properties of NTT:
  - Improves runtime of polynomial multiplication: $\mathcal{O}(n^2) \rightarrow \mathcal{O}(n \log(n))$
  - Commonly used in post-quantum cryptography

Number-Theoretic Transformation

- Operates over modular ring domain $\mathbb{Z}_q$
  - Ring arithmetic modulo prime q
- Special properties of NTT:
  - Improves runtime of polynomial multiplication: $\mathcal{O}(n^2) \to \mathcal{O}(n \log(n))$
  - Commonly used in post-quantum cryptography

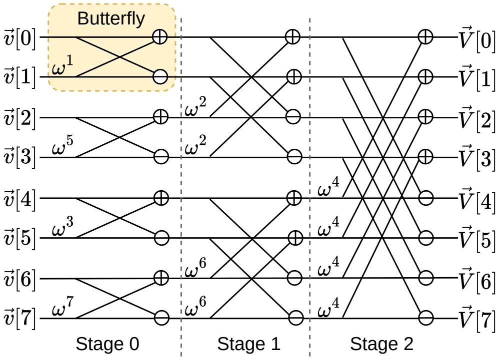
Exploiting Similarities of FFT and NTT

- Execution flow of NTT is identical to FFT
  - Share control logic
  - Share load and store logic
- Just underlying arithmetic is different
  - But still allows resource sharing!



Exploiting Similarities of FFT and NTT

- Execution flow of NTT is identical to FFT
  - ➔ Share control logic
  - ➔ Share load and store logic
- Just underlying arithmetic is different
  - ➔ But still allows resource sharing!



Sharing of Integer Multipliers

- FFT instantiates four floating-point multipliers

$$(a_r + \mathrm{i}a_i)(b_r + \mathrm{i}b_i) = (a_r b_r - a_i b_i) + \mathrm{i}(a_r b_i + a_i b_r)$$

➔ each contains one integer multiplier

Use the existing integer multipliers in NTT!

Sharing of Integer Multipliers

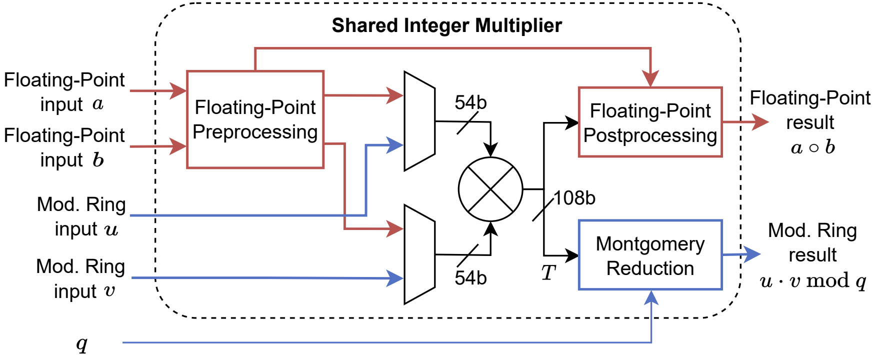
- FFT instantiates four floating-point multipliers

$$(a_r + \mathrm{i}a_i)(b_r + \mathrm{i}b_i) = (a_r b_r - a_i b_i) + \mathrm{i}(a_r b_i + a_i b_r)$$

➔ each contains one integer multiplier

Use the existing integer multipliers in NTT!

Sharing of Integer Multipliers



Sharing of Integer Multipliers: Benefits

- 1 complex multiplier $\leftrightarrow$ 4 modular ring multipliers
- Reused for 3 concurrent NTT + 1 twiddle factor generation
  - Saves 44% of DSPs

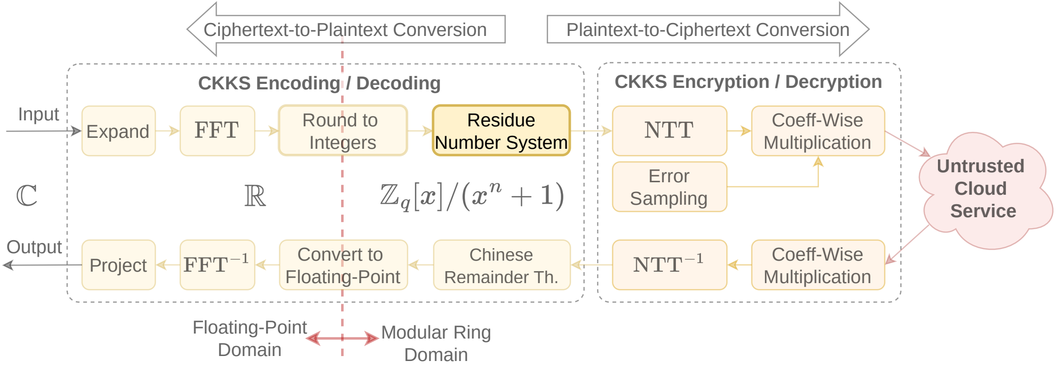
Sharing of Integer Multipliers: Benefits

- 1 complex multiplier $\leftrightarrow$ 4 modular ring multipliers
- Reused for 3 concurrent NTT + 1 twiddle factor generation
  - Saves 44% of DSPs

Sharing of Integer Multipliers: Benefits

- 1 complex multiplier $\leftrightarrow$ 4 modular ring multipliers
- Reused for 3 concurrent NTT + 1 twiddle factor generation
  - ➔ Saves 44% of DSPs

Residue Number System



Residue Number System (in short RNS)

- Reduces computational complexity
- Splits large encoded value x into multiple smaller values $(x_0, \ldots, x_{L-1})$

$$x_i \equiv x \bmod q_i$$

- Challenging to find an efficient hardware solution
  - Large operand sizes x; small q_i

Residue Number System (in short RNS)

- Reduces computational complexity
- Splits large encoded value x into multiple smaller values $(x_0, \ldots, x_{L-1})$

$$x_i \equiv x \bmod q_i$$

- Challenging to find an efficient hardware solution
  - Large operand sizes x; small q_i

Residue Number System (in short RNS)

- Reduces computational complexity
- Splits large encoded value x into multiple smaller values $(x_0, \ldots, x_{L-1})$

$$x_i \equiv x \bmod q_i$$

- Challenging to find an efficient hardware solution
  - Large operand sizes x; small q_i

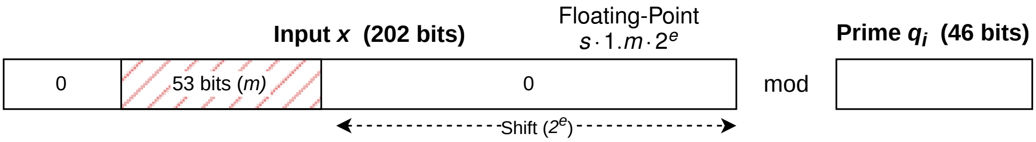
Residue Number System: Our solution

Input x (202 bits)		**Prime q_i (46 bits)**
	mod	

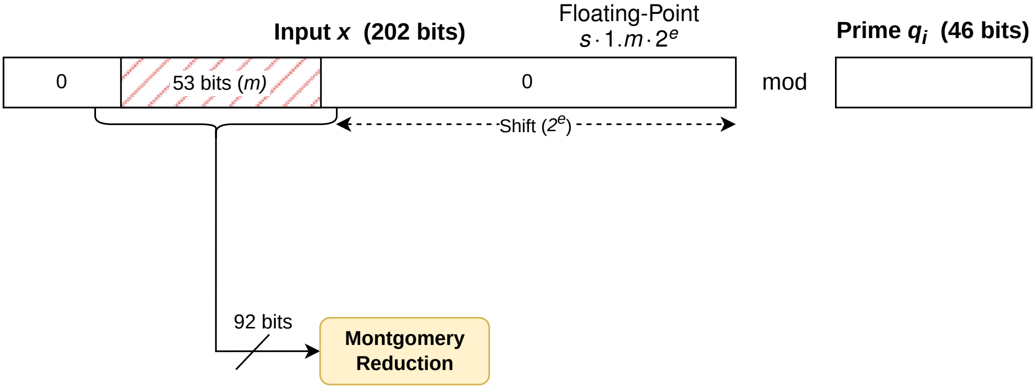
Residue Number System: Our solution

Input *x* (202 bits)		**Prime q_i (46 bits)**
Comes from Floating-Point: $s \cdot 1.m \cdot 2^e$	mod	

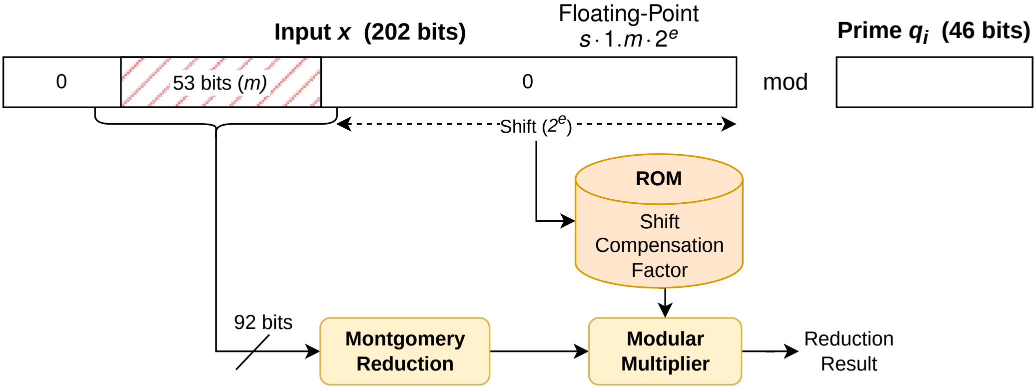
Residue Number System: Our solution



Residue Number System: Our solution



Residue Number System: Our solution



Residue Number System: Our solution

- Benefits compared to existing approach:
  - Only reuses existing datapath resources
  - Reduces area consumption significantly
  - Low latency
  - Scales well for different parameter sets

Residue Number System: Our solution

- Benefits compared to existing approach:
  - Only reuses existing datapath resources
  - Reduces area consumption significantly
  - Low latency
  - Scales well for different parameter sets

Residue Number System: Our solution

- Benefits compared to existing approach:
  - Only reuses existing datapath resources
  - Reduces area consumption significantly
  - Low latency
  - Scales well for different parameter sets

Residue Number System: Our solution

- Benefits compared to existing approach:
  - Only reuses existing datapath resources
  - Reduces area consumption significantly
  - Low latency
  - Scales well for different parameter sets

Table of Contents

Implementation Results

- Implemented for low-cost FPGAs
  - ZYNQ-7000 FPGA on PYNQ-z2 board
    - 130MHz
  - Kintex-7 FPGA on Genesys2 board
    - 200MHz
- Tested and verified on real hardware

Implementation Results

- Implemented for low-cost FPGAs
  - ZYNQ-7000 FPGA on PYNQ-z2 board
    - 130MHz
  - Kintex-7 FPGA on Genesys2 board
    - 200MHz
- Tested and verified on real hardware

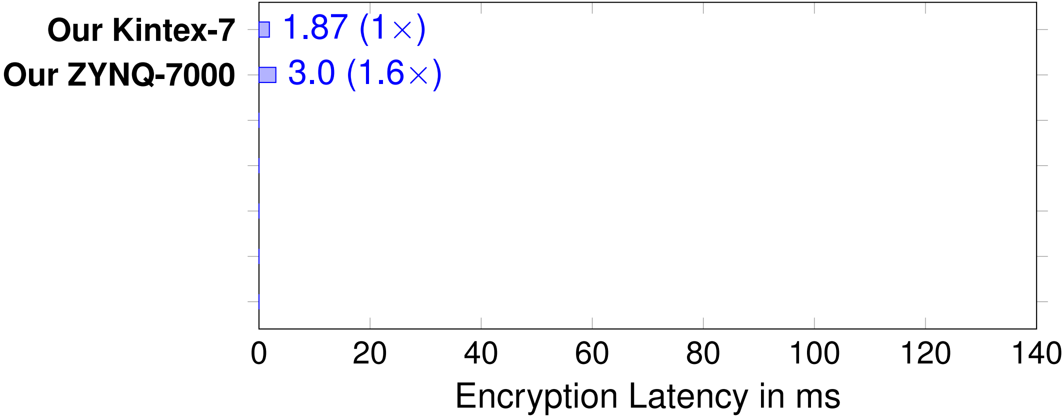
Implementation Results

- Implemented for low-cost FPGAs
  - ZYNQ-7000 FPGA on PYNQ-z2 board
    - 130MHz
  - Kintex-7 FPGA on Genesys2 board
    - 200MHz
- Tested and verified on real hardware

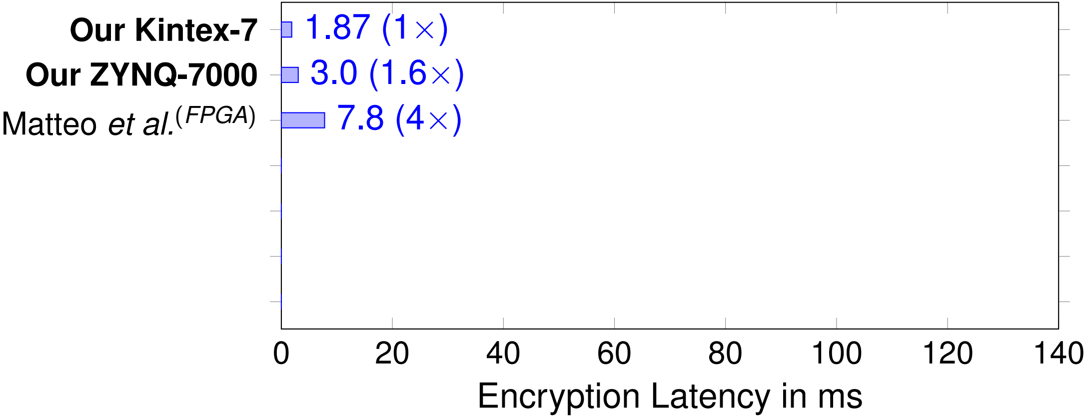
Implementation Results

- Implemented for low-cost FPGAs
  - ZYNQ-7000 FPGA on PYNQ-z2 board
    - 130MHz
  - Kintex-7 FPGA on Genesys2 board
    - 200MHz
- Tested and verified on real hardware

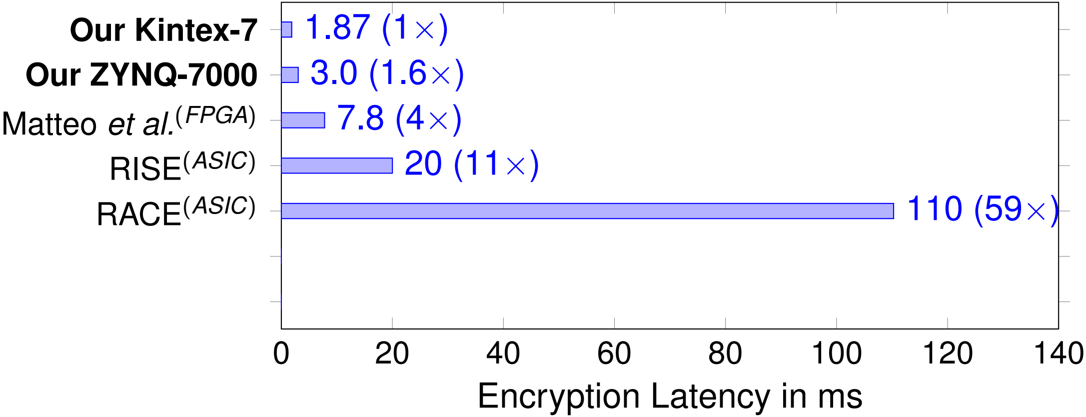
Comparison with Prior Work: Encryption Latency



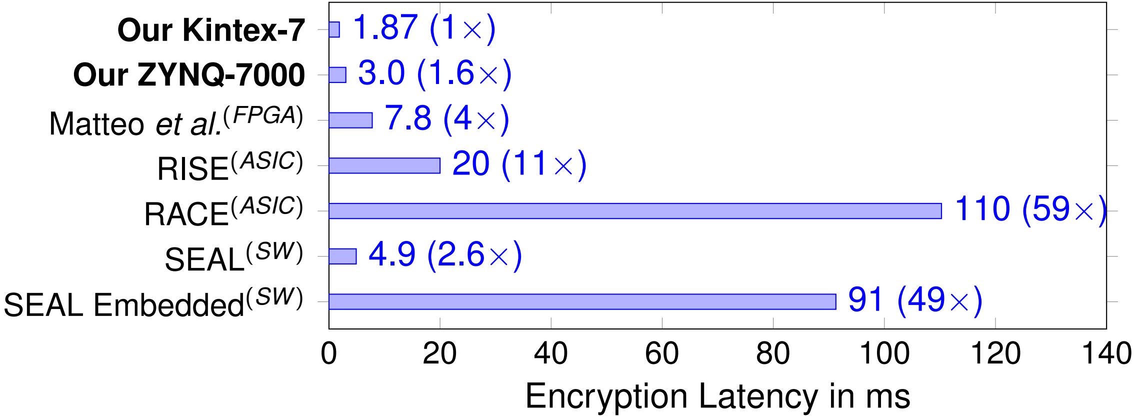
Comparison with Prior Work: Encryption Latency



Comparison with Prior Work: Encryption Latency

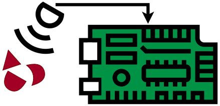


Comparison with Prior Work: Encryption Latency



Performance Figures: Encryption

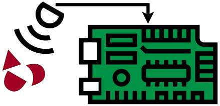
Our accelerator (FPGA)	**Intel Core CPU**	**Improvement**
200 MHz	2.3 GHz	11.5× lower
534 Enc/s	204 Enc/s	2.6×
5.8 mJ/Enc	73.5 mJ/Enc	12.7×



IoT Device (Client)

Performance Figures: Encryption

Our accelerator (FPGA)	**Intel Core CPU**	**Improvement**
200 MHz	2.3 GHz	11.5× lower
534 Enc/s	204 Enc/s	2.6×
5.8 mJ/Enc	73.5 mJ/Enc	12.7×



IoT Device (Client)

Performance Figures: Encryption

Our accelerator (FPGA)	**Intel Core CPU**	**Improvement**
200 MHz	2.3 GHz	11.5× lower
534 Enc/s	204 Enc/s	2.6×
5.8 mJ/Enc	73.5 mJ/Enc	12.7×

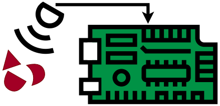


Table of Contents

Main Contributions

- First hardware accelerator covering all client operations in CKKS
- Unifies the FFT and NTT to save resources
- Novel hardware-friendly approach for RNS
- Achieves performance gain of up to 59×
- We made Aloha-HE's source code public on GitHub

Main Contributions

- First hardware accelerator covering all client operations in CKKS
- Unifies the FFT and NTT to save resources
- Novel hardware-friendly approach for RNS
- Achieves performance gain of up to 59$\times$
- We made Aloha-HE's source code public on GitHub

Main Contributions

- First hardware accelerator covering all client operations in CKKS
- Unifies the FFT and NTT to save resources
- Novel hardware-friendly approach for RNS
- Achieves performance gain of up to 59×
- We made Aloha-HE's source code public on GitHub

Main Contributions

- First hardware accelerator covering all client operations in CKKS
- Unifies the FFT and NTT to save resources
- Novel hardware-friendly approach for RNS
- Achieves performance gain of up to 59$\times$
- We made Aloha-HE's source code public on GitHub

Main Contributions

- First hardware accelerator covering all client operations in CKKS
- Unifies the FFT and NTT to save resources
- Novel hardware-friendly approach for RNS
- Achieves performance gain of up to 59$\times$
- We made Aloha-HE's source code public on GitHub

Aloha-HE

A Low-Area Hardware Accelerator for Client-Side Operations in Homomorphic Encryption

Florian Krieger, Florian Hirner, Ahmet Can Mert, Sujoy Sinha Roy
Institute of Applied Information Processing and Communications, TU Graz

January 31, 2024

References

CKKS Scheme: J. H. Cheon, A. Kim, M. Kim, and Y. Song, “Homomorphic encryption for arithmetic of approximate numbers,” in Advances in Cryptology – ASIACRYPT 2017, T. Takagi and T. Peyrin, Eds. Cham: Springer International Publishing, 2017, pp. 409–437.

Microsoft SEAL: “Microsoft SEAL (release 4.1),” https://github.com/Microsoft/SEAL, Jan 2023, microsoft Research, Redmond, WA.

SEAL-Embedded: D. Natarajan and W. Dai, “Seal-embedded: A homomorphic encryption library for the internet of things,” IACR Trans. on Cryptographic Hardware and Embedded Systems, vol. 2021, no. 3, p. 756–779, Jul. 2021.

RACE: Z. Azad et al., “Race: Risc-v soc for en/decryption acceleration on the edge for homomorphic computation,” in Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design. Association for Computing Machinery, 2022.

RISE: Z. Azad et al., “Rise: Risc-v soc for en/decryption acceleration on the edge for homomorphic encryption,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, pp. 1–14, 2023.

Matteo *et al.*: S. D. Matteo, M. L. Gerfo, and S. Saponara, “Vlsi design and fpga implementation of an ntt hardware accelerator for homomorphic seal-embedded library,” IEEE Access, vol. 11, 2023