HOGE: Homomorphic Gate on An FPGA

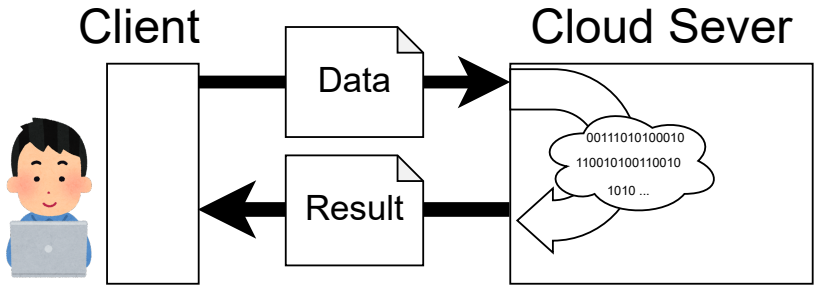
Kotaro Matsuoka[1], Song Bian[2] and Takashi Sato[1]

[1]Kyoto University, Japan [2]Beihang University, China

Jan. 23 2024

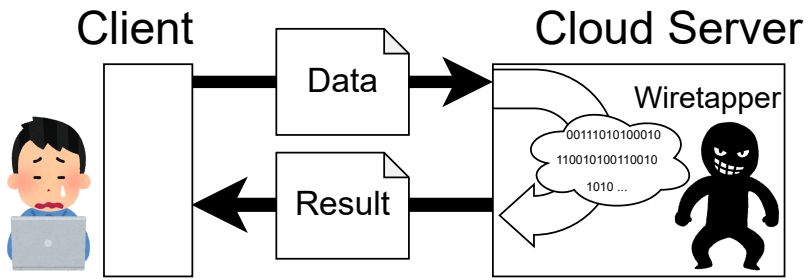
Background: Cloud Computing

- Cloud Computing is a way to outsource the computation.
  - E.g. AWS, GCP, Azure, OCI, etc.



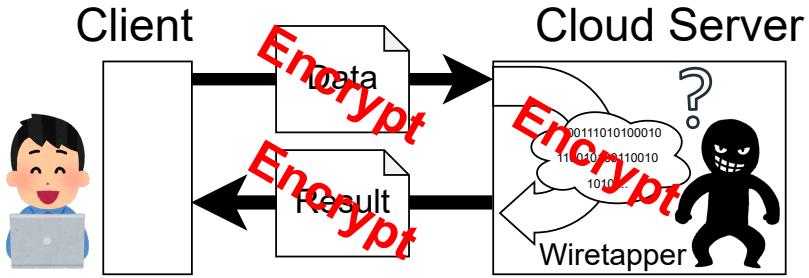
Background: Privacy Concerns

- The data are in **plaintext** at the execution time.
  - The malicious wiretapper can steal the data.



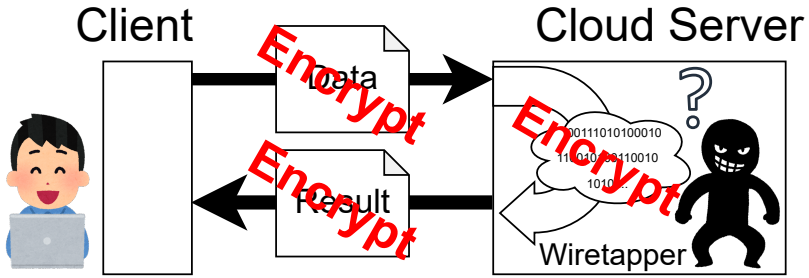
Background: Privacy-Preserving Computation

- One of the ways to prevent such wiretapping is PPC.
  - Processing the data without decryption.
- There are several PPC schemes.
  - Fully Homomorphic Encryption (FHE)
  - Secure Multi-Party Computation (SMPC)
  - Garbled Circuit (GC)
  - Trusted Execution Environment (TEE)



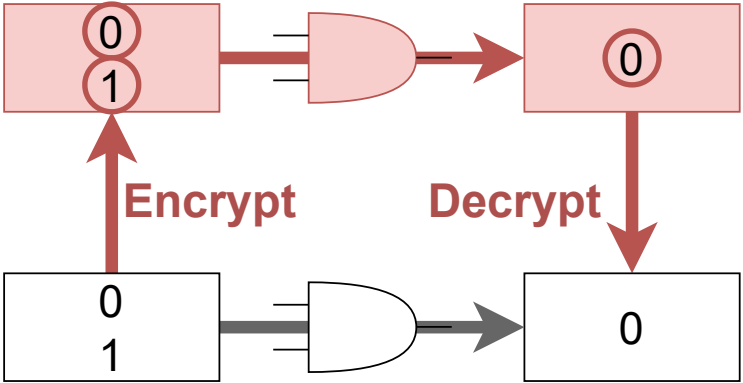
Background: Privacy-Preserving Computation

- One of the ways to prevent such wiretapping is PPC.
  - Processing the data without decryption.
- There are several PPC schemes.
  - **Fully Homomorphic Encryption (FHE)**
  - Secure Multi-Party Computation (SMPC)
  - Garbled Circuit (GC)
  - Trusted Execution Environment (TEE)



Preliminaries: Fully Homomorphic Encryption(FHE)

- FHE is the form of encryption that permits function evaluations without decryption.
  - E.g. AND evaluation without decryption.
- Pros: Only requires correctly handling the secret key.
- Cons: Slow operations



Preliminaries: BOOTSTRAPPING

- In FHE, the operations increase the *noise* in the ciphertext.
  - Too large *noise* cause the decryption failure.

- BOOTSTRAPPING refresh the *noise* to the constant amount.
  - BOOTSTRAPPING is the slowest operation of FHE.
- Speeding-up BOOTSTRAPPING is a crucial topic for FHE.

Preliminaries: TFHE

- There are several kinds of FHE schemes.
  - E.g. CKKS [1], BFV [2, 3], TFHE [4]
- TFHE provides the fastest BOOTSTRAPPING.
  - TFHE: Fully Homomorphic Encryption over the Torus
  - Around 10ms on CPU.
- TFHE is good at logic circuit evaluations.
  - We can use ordinary logic synthesis tools.

Problem: Slow operations in TFHE BOOTSTRAPPING

1. Polynomial Multiplication
   - BOOTSTRAPPING needs some hundreds of polynomial multiplications.
2. Memory bandwidth requirement
   - BOOTSTRAPPING needs streaming about 120MB key.

Problem: Slow operations in TFHE BOOTSTRAPPING

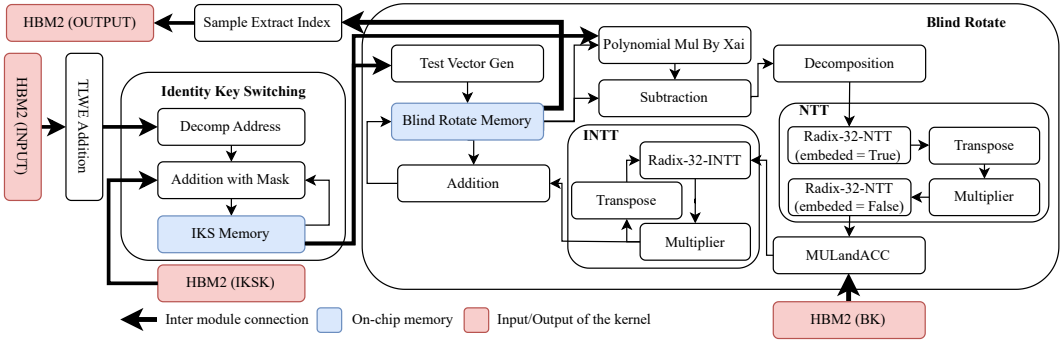
1. Polynomial Multiplication
   - BOOTSTRAPPING needs some hundreds of polynomial multiplications.

   ⇒ Efficient specific moduli NTT/INTT modules
2. Memory bandwidth requirement
   - BOOTSTRAPPING needs streaming about 120MB key.

   ⇒ Utilizing HBM2 (High Bandwidth Memory 2).

Proposed: Abstract architecture of HOGE

- Target FPGA: AMD Alveo U280 (equipped with HBM2)
- The architecture is divided into **Identity Key Switching** and **Blind Rotate**.
  - All BOOTSTRAPPING operations are placed on FPGA.
- **Identity Key Switching** accumulates the vectors (IKSK).
  - Highly memory-intensive but simple computations.
- **Blind Rotate** includes polynomial multiplications.



Preliminaries: Polynomial Multiplication using NTT

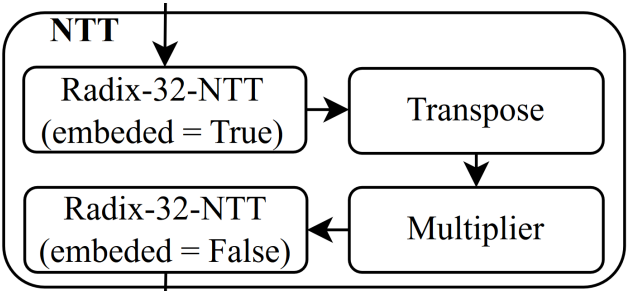
- NTT: Number Theoretic Transform
  - Discrete Fourier Transform (DFT) over integers modulo prime p.
- We can use primitive N-th root of unity ω as *twiddle*.
  - $\omega^N \equiv 1 \bmod p$
- The convolution theorem holds for NTT.
  - $\odot$ is Hadamard product.
  - σ is *twist* to do negacyclic multiplications.
    - $\bmod X^N + 1$ instead of $\bmod X^N - 1$

$$a[x] \cdot b[X] \bmod X^N + 1 =$$
$$\sigma^{-1} \odot INTT\big(NTT(\sigma \odot a[X]) \odot NTT(\sigma \odot b[X])\big) \quad (1)$$

Proposed: Four-step Radix-32 NTT

- We can do an FFT-like algorithm in NTT calculation.
- Four-step NTT: Applying $\sqrt{N}$-radix NTT twice and one data transpose.
  - In our implementation, $N = 1024$, $\sqrt{N} = 32$.
  - Pros: It provides low latency NTT implementation.
  - Cons: It requires a large area.

- Utilizing specific moduli $2^{64} - 2^{32} + 1$.



Characteristics of $2^{64} - 2^{32} + 1$

- This prime is also utilized in the GPU implementation [5].
  - We applied it to an FPGA with improvements.

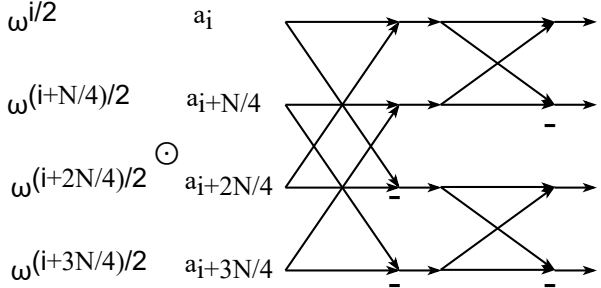
1. The modulo operation is easy.
   - Only addition, subtraction, and shift are used.
2. Up to 64-radix efficient NTT.
   - $8^{64} \equiv 1 \bmod 2^{64} - 2^{32} + 1$
     - We can replace *twiddle* multiplications with left shifts.

- We use only 32-radix NTT while 64-radix NTT is possible.

- We can embed *twist* multiplication into 32-radix NTT.

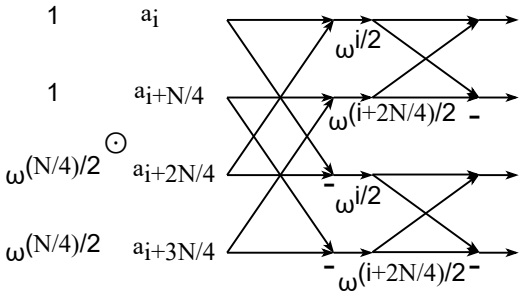
Proposed: Twist embedding with $2^{64} - 2^{32} + 1$

- The actual value of *twist* is $\sigma_i = \omega^{\frac{i}{2}}$.
- The below figure is a radix-4 case.
  - Pushing common factor to the later stage.
- At last, *twist* multiplication is replaced by $\omega^{\frac{N}{2 \cdot radix}}$.
  - If $radix \leq 64/2 = 32$, $\omega^{\frac{N}{2 \cdot radix}}$ is left shift.
  - Other constants in the butterfly are also left shifts.
- Reducing latency and area by removing multipliers.



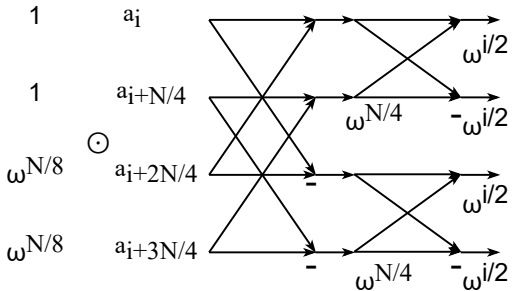
Proposed: Twist embedding with $2^{64} - 2^{32} + 1$

- The actual value of *twist* is $\sigma_i = \omega^{\frac{i}{2}}$.
- The below figure is a radix-4 case.
  - Pushing common factor to the later stage.
- At last, *twist* multiplication is replaced by $\omega^{\frac{N}{2 \cdot radix}}$.
  - If $radix \leq 64/2 = 32$, $\omega^{\frac{N}{2 \cdot radix}}$ is left shift.
  - Other constants in the butterfly are also left shifts.
- Reducing latency and area by removing multipliers.



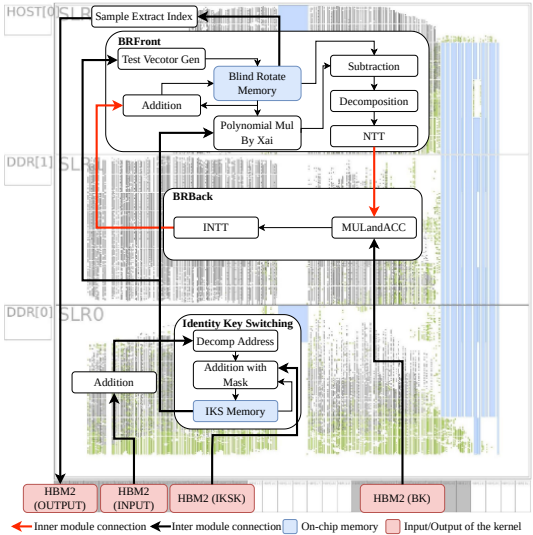
Proposed: Twist embedding with $2^{64} - 2^{32} + 1$

- The actual value of *twist* is $\sigma_i = \omega^{\frac{i}{2}}$.
- The below figure is a radix-4 case.
  - Pushing common factor to the later stage.
- At last, *twist* multiplication is replaced by $\omega^{\frac{N}{2 \cdot radix}}$.
  - If $radix \leq 64/2 = 32$, $\omega^{\frac{N}{2 \cdot radix}}$ is left shift.
  - Other constants in the butterfly are also left shifts.
- Reducing latency and area by removing multipliers.



Proposed: Module Placements

- Alveo U280 is divided into three SLRs.
- The modules are divided into three.
- Because of routing difficulty, HBM2 is not fully utilized.
  - 10 ch. for IKSK.
  - 8 ch. for BK.



Evaluation: Par Gate runtime

- Spped-up ratio is normalized by TFHEpp's runtime.
- Our implementation is $5-6\times$ faster than CPU implementations.

Table: Runtime comparisons of NAND over ciphertexts

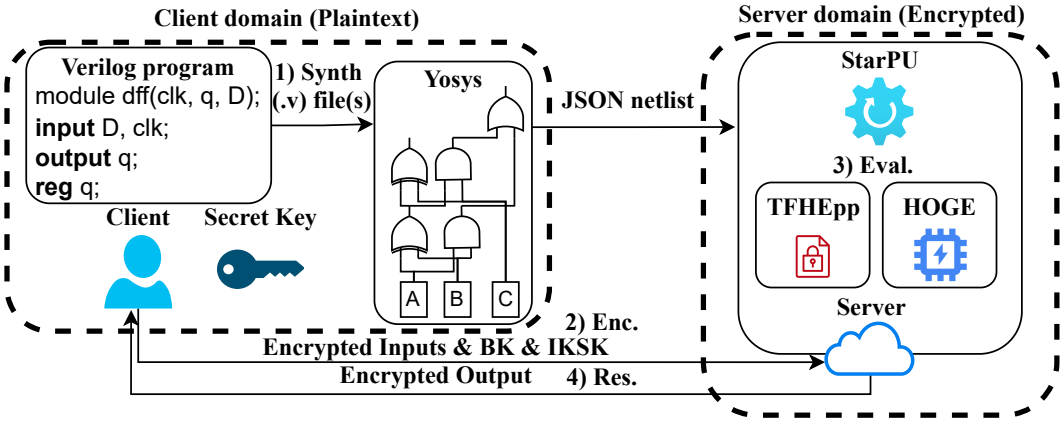
	CPU[1] Impl.		GPU Impl.	
	TFHEpp [6]	TFHE [7]	cuFHE [5]	CPU-GPU-TFHE [8]
Latency [ms]	9.3	11.5	11.3	11.4
Speed-up	1.0	0.8	0.8	0.8

	FPGA/ASIC Impl.			
Latency [ms]	TVE [9]	YKP [10]	MATCHA [11]	HOGE
	14 [11]	1.9	0.16 (est.)	1.6
Speed-up	0.7	4.9	58	5.8

[1] Ryzen 5950X

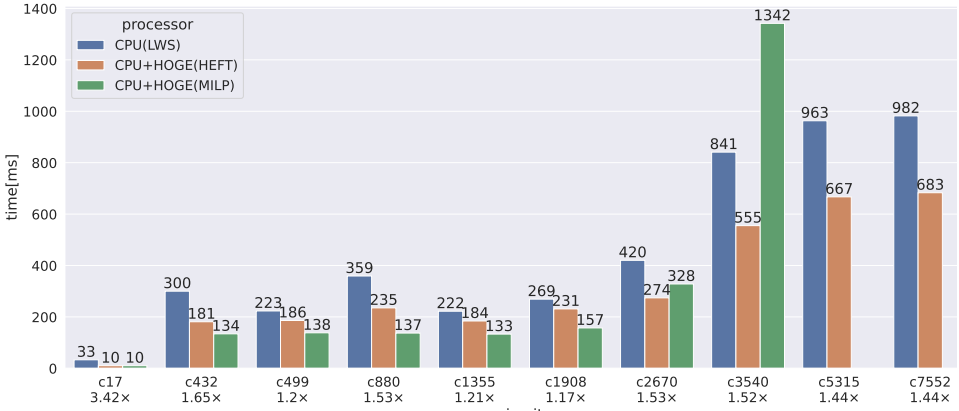
Proposed: Logic circuit evaluation engine with HOGE

- Logic circuits evaluation needs software to handle HOGE.
  - Tangor: StarPU-based [12] engine that can handle HOGE.
- The circuit is fed in the Verilog format.



Evaluation: Circuit Evaluations

- Environment: Ryzen 5950X + Alveo U280
- Evaluated with ISCAS'85 circuits.
  - Introducing HOGE improved the performance.
- 1.44× improvement ≈ the throughput improvement.
  - Higher improvement means fast critical path evaluations.



Conclusion

- Proposed FPGA implementation $5-6\times$ faster than CPU implementations.
- Employed low latency four-step NTT with the prime $2^{64}-2^{32}+1$ and the twist embedding.
- Evaluated with ISCAS'85 circuits to demonstrate improvements with logic circuit evaluations.

Reference I

[1] 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), pp. 409–437, Springer International Publishing, 2017.

[2] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) Fully Homomorphic Encryption without Bootstrapping," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, (New York, NY, USA), p. 309–325, Association for Computing Machinery, 2012.

[3] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption." Cryptology ePrint Archive, Report 2012/144, 2012.
`https://ia.cr/2012/144`.

[4] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *Advances in Cryptology – ASIACRYPT 2016* (J. H. Cheon and T. Takagi, eds.), (Berlin, Heidelberg), pp. 3–33, Springer Berlin Heidelberg, 2016.

[5] W. Dai, "Original implementation of cuFHE." `https://github.com/vernamlab/cuFHE`, 2018.

[6] K. Matsuoka, "TFHEpp: pure C++ implementation of TFHE cryptosystem."
`https://github.com/virtualsecureplatform/TFHEpp`, 2020.

[7] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast fully homomorphic encryption library," August 2016.
https://tfhe.github.io/tfhe/.

Reference II

[8] T. Morshed, M. Aziz, and N. Mohammed, “CPU and GPU Accelerated Fully Homomorphic Encryption,” in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, (Los Alamitos, CA, USA), pp. 142–153, IEEE Computer Society, dec 2020.

[9] S. Gener, P. Newton, D. Tan, S. Richelson, G. Lemieux, and P. Brisk, “An FPGA-based Programmable Vector Engine for Fast Fully Homomorphic Encryption over the Torus,” *SPSL: Secure and Private Systems for Machine Learning (ISCA Workshop)*.

[10] T. Ye, R. Kannan, and V. K. Prasanna, “FPGA Acceleration of Fully Homomorphic Encryption over the Torus,” in *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–7, 2022.

[11] L. Jiang, Q. Lou, and N. Joshi, “MATCHA: A Fast and Energy-Efficient Accelerator for Fully Homomorphic Encryption over the Torus,” in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, DAC ’22, (New York, NY, USA), p. 235–240, Association for Computing Machinery, 2022.

[12] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, “StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures,” *CCPE - Concurrency and Computation: Practice and Experience, Special Issue: Euro-Par 2009*, vol. 23, pp. 187–198, Feb. 2011.

Backup: Resource Utilization

Table: Resource utilization for each kernel

Name	IKS	BRFront	BRBack	Kernel Total
LUT	116590 (10.50%)	154961 (13.96%)	170476 (15.36%)	442027 (39.82%)
LUTAsMem	72733 (12.68%)	19641 (3.42%)	46189 (8.05%)	138563 (24.16%)
Register	45382 (1.94%)	242273 (10.34%)	250379 (10.69%)	538034 (22.97%)
BRAM	177 (9.75%)	14 (0.77%)	57 (3.14%)	248 (13.66%)
DSP	0 (0.0%)	512 (5.68%)	1536 (17.03%)	2048 (22.71%)

Backup: YKP comparison

Table: Runtime comparisons of NAND with YKP in different parameters

	YKP		HOGE	
BKU parameter m	1	4	1	1
Security	80-bit	80-bit	128-bit	80-bit
Per Gate Latency [ms]	7.5	1.9	1.6	1.3
Speed-up	1.0	3.9	4.7	5.7