cea list

HEGIDE: A MIMD Oblivious Processor for Private Function Evaluation over CKKS

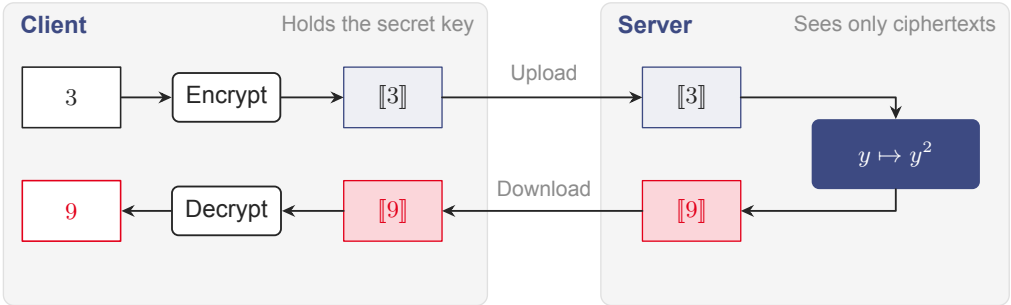
Jules Dumezy, Nicolas Ye, Pierre-Emmanuel Clet, Olive Chakraborty, and Aymen Boudguiga

ACM CCS, 18 November 2026

INSTITUT CARNOT CEA LIST

université PARIS-SACLAY

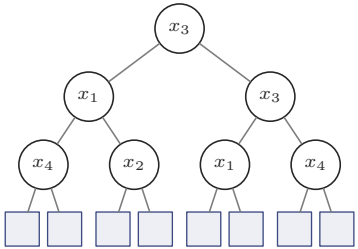
Fully homomorphic encryption in one picture



The server never sees x or $f(x)$… but it has to know f.

FHE hides the data, not the program

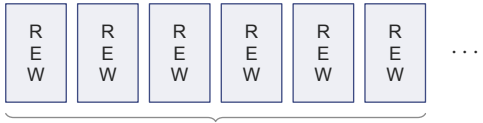
Standard FHE: the client's circuit



Depth
Size
Features used
Wiring

Comparisons on encrypted features

HEGIDE: one public processor



T identical cycles: read, execute, write

Program and data are both encrypted inputs

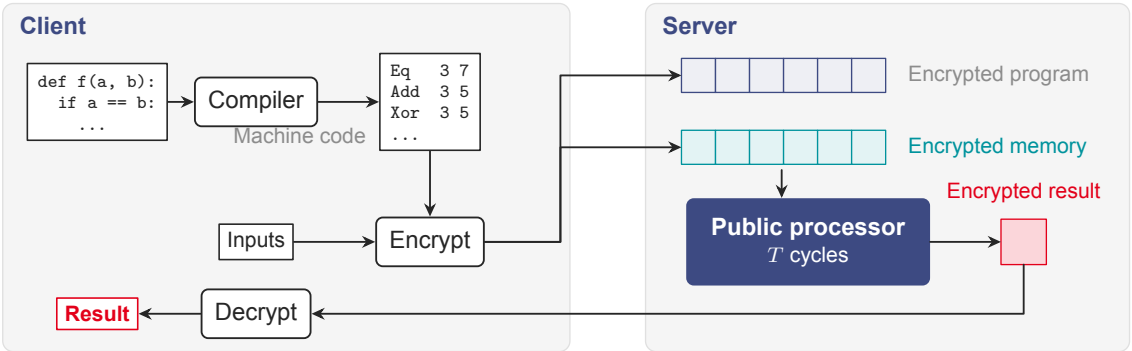
Private function evaluation

The program is the asset: hide it *and* the data.

Threat model

Honest-but-curious server, one key. Leaks only T, M, W and the instruction set.

Idea: the program becomes data



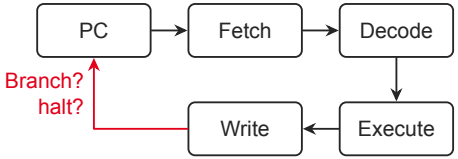
Prior encrypted processors emulate a CPU with an encrypted program counter (1.26 to 3.28 s per cycle), run one program at a time, and pay $O(M)$ for every write to an encrypted address.

“Can we design an efficient, MIMD encrypted processor? Can we bypass the write bottleneck?”

Me, a year ago.

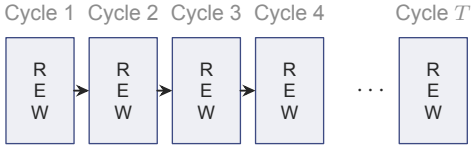
Oblivious deciders: everything is known in advance

Standard processor



Next step depends on the data:
under FHE, an encrypted PC

Oblivious decider

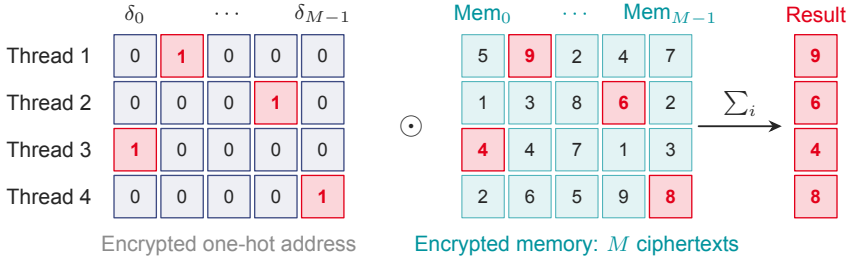


Length, addresses and schedule
fixed at compile time

Non-interactive and oblivious: the server can neither branch nor stop on encrypted data.
$\Rightarrow$ fixed run length T: **no halting problem** $\Rightarrow$ trace known at compile time: **no PC, fetch or decode**.

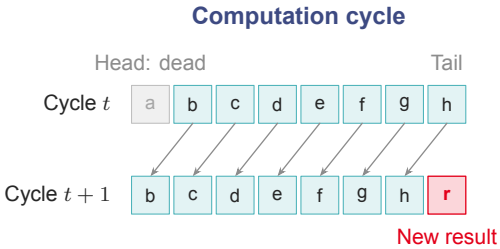
Programs get there by if-arithmetization ($x = b + c\,(a - b)$) and unrolled loops.

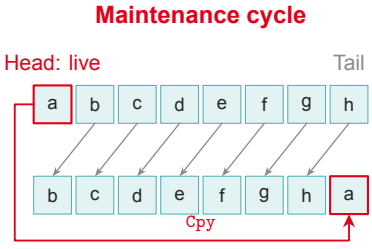
Reading at an encrypted address



REG $= \sum_i \delta_{x,i} \cdot \text{Mem}_i$: M multiplications, **depth 1**, **one thread per slot**.

OSReM: writing is a shift





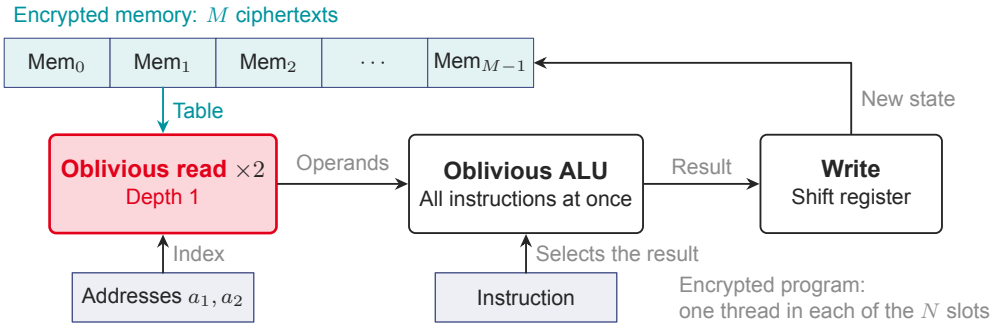
RAM write at an encrypted address

Touches all M cells and costs depth: the whole memory must be bootstrapped.

Shift-register write

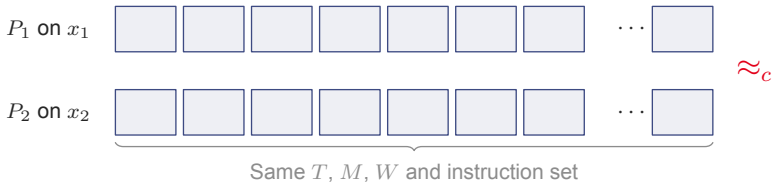
Public, zero homomorphic operations, about 200 µs. With $M = \gamma K$ cells for K live values, at most $1/\gamma$ of the cycles are copies.

One cycle, N threads



ALU: bootstrap the operands (cleans the noise), run all 11 instructions, keep one.
MIMD: each of the $N = 2^{16}$ slots is an independent thread.

Security: full obliviousness



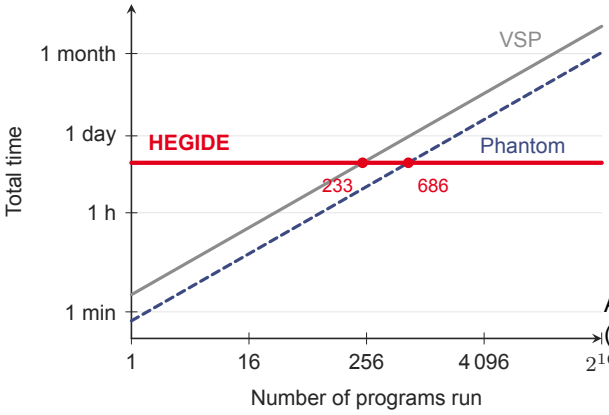
Theorem

If CKKS is IND-CPA secure, the server's views of P_1 on x_1 and P_2 on x_2 are indistinguishable when T, M, W and the instruction set match.

Why

Every cycle runs the same operations; reads and the instruction choice are weighted sums with encrypted weights; writes are a public shift.

Decision tree: one program or many

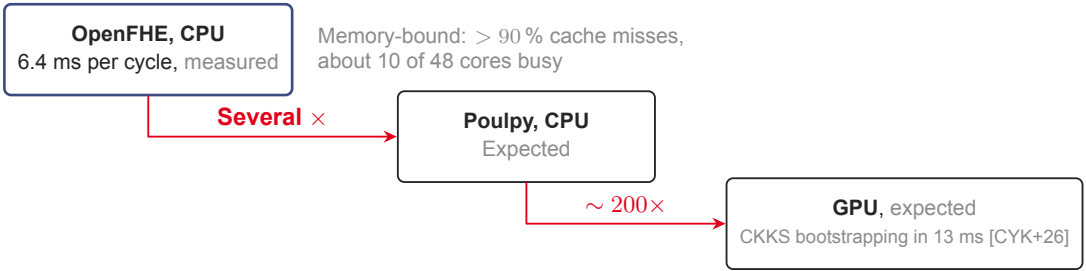


	VSP	Phantom	HEGIDE
Cycles	41	30	60
Per cycle	3.28 s	1.26 s	**7.2 ms***
Per prog.	122 s	41 s	**0.43 s***
RAM per prog.	1.6 GB	1.9 GB	**1.4 MB***

*amortized over 2^{16} programs

Ahead from **233** programs (VSP) and **686** (Phantom). At 2^{16}: $96\times$ Phantom's throughput.

Where it stands, where it goes



Lower latency: a drop-in scheme

About 8 h for one program today. The cleanest fix is a drop-in: our small-batch scheme [ePrint 2026/2513] keeps the CKKS keys and formats and bootstraps few values about $5\times$ faster. Leaner ALUs [Kim25] help too.

Memory and bandwidth

90 GB of RAM, 50 GB of it keys. $O(M)$ indicators per read, compressible to one RLWE ciphertext.

Takeaways

Program privacy, at throughput
6.4 ms per 16-bit cycle, amortized: two orders of magnitude beyond encrypted processors, for batch and multi-program workloads.

Oblivious means known in advance
No program counter, no fetch or decode; writes become a public shift (OSReM).

CKKS slots as threads
2^{16} independent programs per ciphertext (MIMD).

Next
Faster backends (Poulpy, GPU), lower latency with our small-batch scheme, a MIMD-aware compiler, floating point.

Paper
ACM CCS 2026. Code not yet public (administrative hurdles).

Thank you !

CEA Nano-INNOV
91 120 Palaiseau Cedex
France
jules.dumezy@cea.fr

AI disclosure. Anthropic Claude Opus 5.5 was used to assist in the preparation of these slides and in producing the figures.

Backup: why this is not iO

	Indistinguishability obfuscation	HEGIDE
Who runs it	Anyone holding the obfuscated program	The server
Inputs	The evaluator's own, in the clear	The client's, encrypted
Output	In the clear, to the evaluator	Encrypted: only the client decrypts
Hides the program from	An evaluator who sees the outputs	A server that sees no output
Assumptions	Bilinear maps, LPN, local PRGs [JLS21]; far from practical	Ring-LWE: CKKS is IND-CPA
Leaks	Circuit size	T, M, W, instruction set

HEGIDE is private function evaluation: FHE on a universal circuit, made practical.
Only the key holder ever learns anything.

Backup: how does it compare to zkVMs?

	zkVM (e.g. RISC Zero, SP1)	HEGIDE
Goal	Integrity: prove the program ran correctly	Confidentiality: the server learns nothing
Program and data	Seen in the clear by the prover	Seen by nobody but the client
The client gets	Output and a succinct proof	Encrypted output
Trust in the server	None for correctness	Honest-but-curious for correctness, none for privacy
Cost driver	Proof generation	Bootstrapping

Complementary: a zkVM proves what the server did, HEGIDE hides it.
Both at once is verifiable FHE, an open and costly direction.

Backup: why not ORAM or PIR?

	ORAM	PIR	OSReM
Addresses	Visible, pattern shuffled	Encrypted	Encrypted
Writes	Yes, $O(\log N)$ amortised	No: read-only	Yes, a public shift
Interaction	Client in the loop	One query, one answer	None: the server runs alone
Read cost	$O(\log N)$	$O(M)$	$O(M)$, depth 1

Encrypted indices change the problem
The server cannot route a request to a cell anyway. $O(M)$ is the price of computing on an encrypted index, not of hiding a pattern.

FHE-RAM [BM25a]
Arbitrary writes need a read-before-write traversal: 871 ms per write against 302 ms per read, 1 MB heap.

Backup: why not a trusted execution environment?

TEE (SGX, TDX, SEV, GPU enclaves)
Native speed. But trust moves to the hardware vendor and its attestation, and side channels keep breaking enclaves.

HEGIDE
No hardware trust: security reduces to Ring-LWE. The price is speed: 6.4 ms per cycle amortised, about 8 h for one program alone.

Different trade-offs: a TEE trusts a chip, HEGIDE trusts a lattice problem.

Setting: private function evaluation

Use case
A bank owns a trading strategy (the program) and its positions (the data). It wants the cloud's compute, not its eyes.

Model
One key for program and data.
Non-interactive: upload, run, download.
Honest-but-curious server for correctness; privacy holds even if it misbehaves.

The server learns

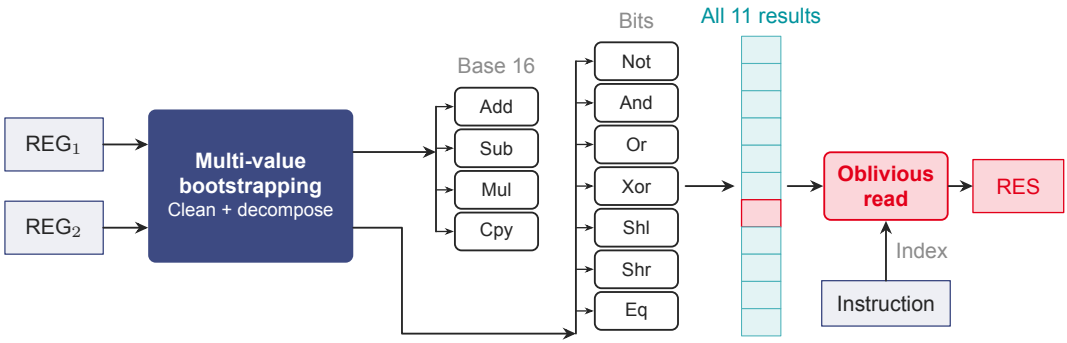
Cycles T | Memory size M | Word size W | Instruction set

And nothing else

Inputs | Outputs | Instructions | Addresses | Control flow

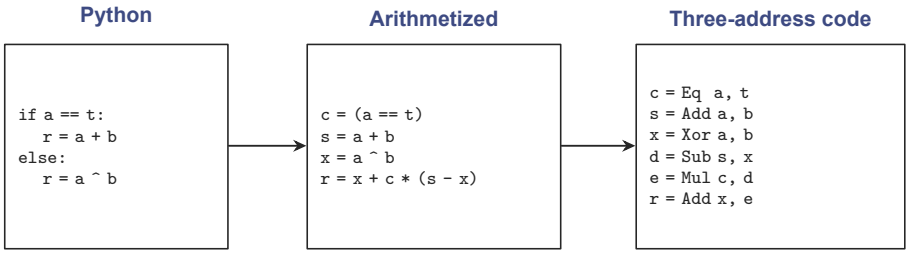
Not covered
What the decrypted output reveals. The client gives no feedback on decryption, so the server gets no chosen-ciphertext oracle.

Oblivious ALU: compute everything, keep one



11 instructions, any word size (16 to 64 bits as base-16 digits), shifts by an encrypted amount. Bootstrapping every operand also cleans the noise: **programs run for any number of cycles**.

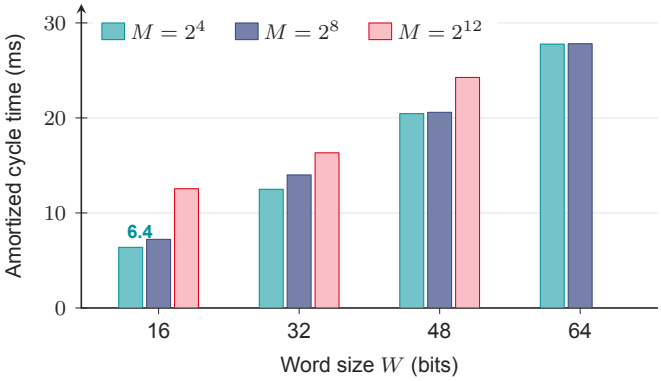
Compiling Python



Both branches are computed:
no data-dependent control flow

One instruction per cycle,
`Cpy` inserted when a live value reaches the head

Results: 6.4 ms per cycle, amortized



157 Hz
A virtual 16-bit CPU per thread, 2^{16} of them.

Where time goes
The ALU's bootstrappings: 90 to 95 %. Writes: 200 µs.

Memory
90 to 180 GB in total, 1.4 to 2.7 MB per thread.

$2\times$ Xeon Platinum 8268, 192 GB, OpenFHE 1.4.2 with HEXL (AVX2), $N = 2^{16}$, 128-bit security. $M = 2^{12}$ and $W = 64$ use the compressed memory mode.

Backup: bandwidth and compression modes

Mode	Plaintext modulus	Memory	Per address	Per instruction	Status
Off	16	$MW/4$	M	$\|I\|$	
Low	16	$MW/4$	$d \log_d M$	$\|I\|$	**Default**
Medium	2^{16}	$MW/16$	$d \log_d M$	$\|I\|$	Experimental
High	2^{16}	$MW/16$	1 RLWE	1 RLWE	Not yet implemented

Number of ciphertexts. $d = 16$ is the ALU radix, W the word size, $|I| = 11$ instructions.

Low and Medium

The client sends the one-hot digits of each address; the server only runs the collapsed multiplexer tree.

High

One small RLWE ciphertext per address: the server decomposes it itself, with large lookup tables [TCHES 2026].

Backup: extensions

n-ary instructions

Fused multiply-add and the like: n parallel reads, ALU depth growing only logarithmically with n.

Read-only global memory

Constants and tables (S-boxes, tree thresholds) in a second, static memory: fewer maintenance cycles.

Communicating threads

A Beneš network of CKKS rotations, $2 \log N - 1$ stages: oblivious sorting, FFTs across slots.

Floating point

Swap the integer ALU for IEEE 754 emulation; deeper, with more bootstrapping.

Encrypted processors so far

	Scheme	Model	Limitation	Per cycle
VSP [MBM+21]	TFHE	5-stage pipeline	512 B of RAM	3.28 s
Phantom [BM25b]	**TFHE (Poulpy)**	**RISC-V, encrypted PC**	**one program at a time**	**1.26 s**
OTM, OUF [ADK24, DK25]	TFHE	Turing machine	tape of 16 to 32 values	seconds
FHE-RAM [BM25a]		encrypted memory	writes: 871 ms vs. 302 ms reads	

Phantom: best latency so far. VSP and Phantom measured on our server, decision-tree program; FHE-RAM with a 1 MB heap.

Two costs to remove: emulating a CPU (PC, fetch, decode), and writing at encrypted addresses.

Backup: experimental setup

- $2\times$ Intel Xeon Platinum 8268 (48 cores), 192 GB of RAM, Ubuntu 24.04
- OpenFHE 1.4.2, clang++, Intel HEXL with AVX2 kernels (no AVX-512 IFMA)
- $N = 2^{16}$, all slots used, scaling factor 2^{48}, $\log Q'P' \leq 1533$ for 128-bit security
- Sparse-secret encapsulation: decryption failure below 2^{-128}
- Plaintext modulus and ALU radix 16: a W-bit word is $W/4$ memory banks
- Decision tree: depth 4, 15 internal nodes; $W = 16$, $M = 256$, 60 cycles