FHE Circuit Construction and Synthesis

Takashi Sato / Kotaro Matsuoka

Jul. 11, 2022

Kyoto University

Motivation: Security in cloud computing

- Both the program and data are decrypted on the server.
  - Wiretappers can take a peek!
- The data can be protected by homomorphic encryption (HE).
  - How about the **program**?
  - Program may be reused or reverse engineered

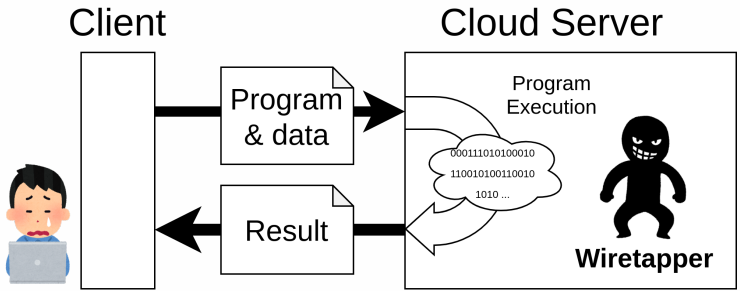


Figure 1: Conventional use case

Our solution: Virtual secure platform (VSP)

- Virtualization: VSP evaluates logic circuits over TFHE, such as RISC-V-like CPU
- Security: VSP protects both **program** and **data**
  - Program and data are indistinguishable
  - The circuits comprising the CPU are independent of the program
- **Standard C programs** are executed in an encrypted manner with encrypted data

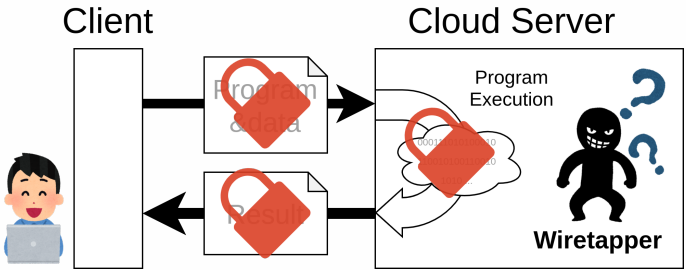


Figure 2: Virtual Secure Platform

Why we use TFHE?

TFHE: Fully Homomorphic Encryption over the Torus[1]

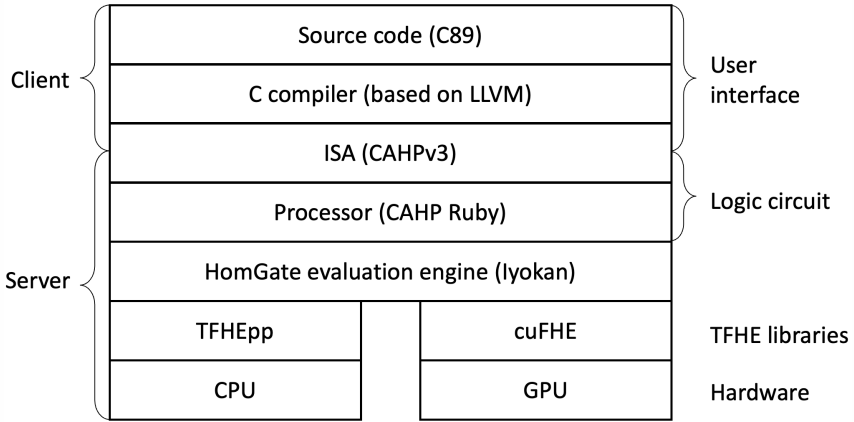
- natively supports evaluation of **logic function** over ciphertexts.
- allows logic circuit evaluation of **infinite depth** via **relatively fast Bootstrapping**.
- supported gates:
  basic 2-input gates, such as NAND, AND, OR, XOR, etc.

$\Rightarrow$ We can utilize existing tool-chain to build logic functions over ciphertexts.
e.g., CPU architecture exploration, high level synthesis, etc.

[1] Chillotti, et al. J. Cryptol. 33, pp.34–91 (2020).

Virtual secure platform (software/hardware stack)

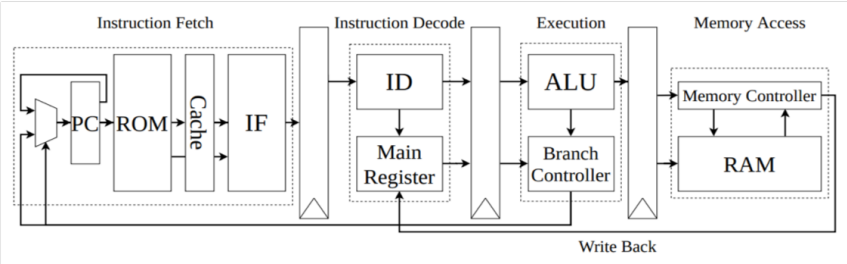
Virtual secure platform (VSP) software stack[2]



[2] https://github.com/virtualsecureplatform/kvsp

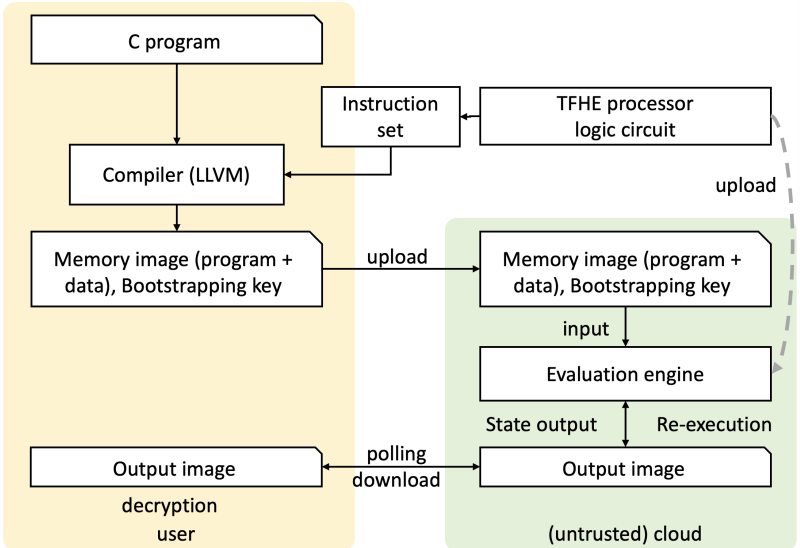
Virtual secure platform (processor)

An example processor implementation in VSP[3]



[3]K. Matsuoka, R. Banno, N. Matsumoto, T. Sato, and S. Bian, "Virtual secure platform: A five-stage pipeline processor over TFHE," in Usenix Security Symposium, pp.4007-4024, Aug. 2021.

Virtual secure platform (execution flow)



Processor design using HomGates

Successfully followed a regular logic design flow

- Architecture: Simple pipeline processor based on RISC-V ISA
- C compiler support using LLVM backend
- "Logical" design only, does not need "physical" design
  - HDL: Chisel, Logic synthesis: Yosis, etc.
  - Use gates available in HomGates only (even memories – naively, MUX tree)
  - DFF does not require logic gates
  - No wires, no clock tree, no timing violations, no scan chain, ...
- Run on CPU, GPU, or their hybrid environments

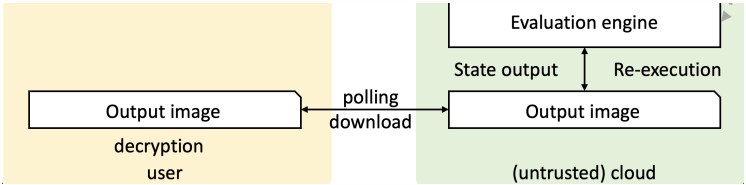
Runs as expected, ... but extremely slow

Design considerations/optimizations for TFHE processor

- Protocols (e.g. state-leakage problem)
- HomGate evaluation and scheduling (Iyokan)
- Parallelism pursuit in gate evaluation
- ISA design suitable for HE
- Memory design
- Additional logic gates suitable for building processors

Knowing program termination

- Program state (running/in halt) is considered as information leakage
- Customize `halt` instruction 1) to set an encrypted termination flag either on register or memory, and 2) to enter an infinite loop (at halt address).
- Protocol:
  1. pass # clock cycles N_c to evaluation engine
  2. evaluation engine performs N_c cycles of processing
  3. user checks the termination flag
  4. if set, terminate and retrieve memory image; else goto 2



- Logic circuit is represented as a directed acyclic graph (DAG)
- Replacing the gates with HomGates, computation is encrypted
  - Logic gates operate in parallel
  - Naturally reaches steady state after a while

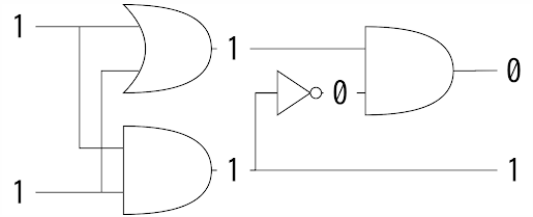


Figure 3: Example combinational circuit (plain)

Iyokan: A gate evaluation engine

- Logic circuit is represented as a directed acyclic graph (DAG)
- Replacing the gates with HomGates, computation is encrypted
  - **Evaluation engine** needs to take care of signal flow (topological order). Similar to static timing analysis or logic circuit emulation.

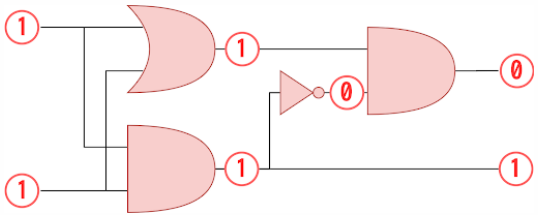


Figure 3: Example combinational circuit (encrypted)

How the performance of gate evaluation is determined?

- The total number of gates determines the total calculation effort
- Parallelizable as long as signal flow constraint is satisfied
  - Single worker: Circuit latency $\propto$ **total number** of gates.
  - Infinite workers: Circuit latency $\propto$ **logic depth**.
  - Compact and shallow logic realization is desirable

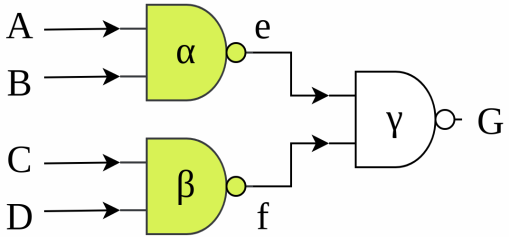


Figure 4: Example combinational circuit (green gates can be evaluated in parallel)

How to increase number of workers?

- In VSP, we used up to 64 vCPU and 8 NVIDIA V100 GPU.
- To utilize multiple vCPUs and GPUs, **scheduling** is needed.
  - Different from physical circuit (naturally parallel).
  - A worker (1 CPU or 1 streaming processor) computes one logic gate at a time
  - Can dispatch only evaluatable HomGates to available workers

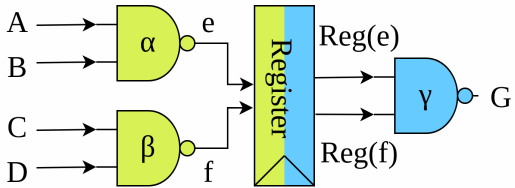
Iyokan: Our execution engine

- Simultaneous execution using CPU (TFHEpp) and GPU (cuFHE) are supported.
- The circuit is described by JSON format
  - Synthesized by Yosys[a] from Verilog.

[a] `https://github.com/YosysHQ/yosys`

Findings through processor design

- Memory is the most important component for efficiency.
  - The memory (a MUX tree) dominates the circuit evaluation time.
- RAM is computationally much heavier than ROM.
  - For the sake of oblivious access, all memory contents need refreshing.
    i.e., all RAM contents require Bootstrapping every clock cycle.
- Reducing data/instruction length can reduce **width** and the memory footprint.
- Pipelining can reduce **depth** (by increasing **width**).



Design considerations/optimizations for TFHE processor

- Protocols (e.g. state-leakage problem)
- HomGate evaluation and scheduling (Iyokan)
- Parallelism pursuit in gate evaluation
- ISA design suitable for HE
- Memory design
- Additional logic gates suitable for building processors

Example: Solution in our work (VSP)

- Use RISC-V based customized ISA.
  - 16-bit data length, 16/24-bit instruction length.
  - Ease the compiler support. (C is supported in VSP)
- Introduce pipelining while keeping **width** and the gate counts low enough.
- Employ the Harvard architecture to use ROM for storing instructions.
- CMUX Memory: Cryptographic acceleration of memory evaluation.

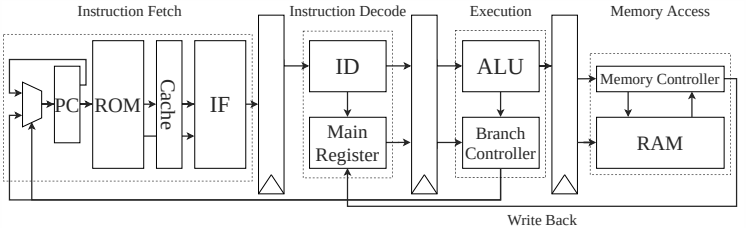


Figure 5: The architecture of processor in VSP

CMUX Memory

- The memory constructed by Leveled Homomorphic Encryption mode.
  - About $\mathbf{1000}\times$ faster multiplexer (CMUX) is supported.
- Depth is limited by introduced noise.
- The select input and the output are different formats.
  - Circuit Bootstrapping converts them.

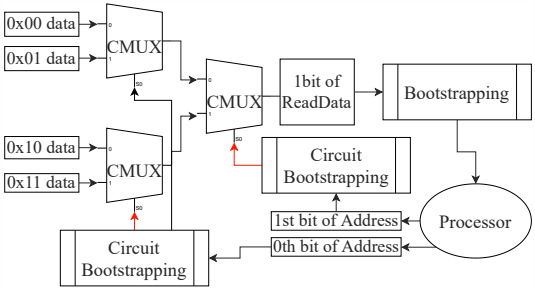


Figure 6: The read unit for 2 bit address by CMUX Memory

Performance of VSP

- Equipped with 512 bytes ROM and 512 bytes RAM.
- Pipelining degrades performance if the number of worker is not enough.
- CMUX Memory improves the peroformance.
- At the best case, we achieve around 1.25 Hz evaluation.

Table 1: Performance Evaluation Using Hamming

Machine	Pipelining?	CMUX Memory?	# of cycles	Runtime [s]	sec./cycle
AWS c5.metal	No	Yes	936	**2342.0**	**2.502**
	Yes	Yes	1216	**2773.0**	**2.280**
AWS p3.16xlarge	No	No	936	**1627.0**	**1.739**
	No	Yes	936	**1440.0**	**1.538**
	Yes	No	1216	**1566.0**	**1.28**
	Yes	Yes	1216	**965.9**	**0.794**

Constructing compound logic gates

- Proposed some three-input or multi output compound gates.
  - Half Adder, Full Adder, AOI21, OAI21 (easy to utilize in Yosys)
- They can be evaluated at a cost of one or two conventional gates.
  - Reduce **depth** and/or **width** without increasing both.

Encoded Message Space in Torus

Torus ($\mathbb{T}$): $\mathbb{R} \bmod 1 \in [-\frac{1}{2}, \frac{1}{2}ti)$

- The message space of TFHE
- Need to encode the plaintext space, Binary $\{0, 1\}$, into Torus.

$\mathbb{M}_t = \{-\frac{1}{t}, \frac{1}{t}\}$: Encoded message space

- $t \in \mathbb{N}$. Corresponds $\{0, 1\}$ respectively.
- We use $\mathbb{M}_8$ and $\mathbb{M}_{12}$
  - $\mathbb{M}_8$: Used in the original TFHE implementation
  - $\mathbb{M}_{12}$: More performance benefit but increase the decryption error rate

Blind Rotate: The core functionality of TFHE

- Can evaluate Look Up Table (LUT)
  - LUT is represented as a polynomial $\boldsymbol{TV}[\boldsymbol{X}] \in \mathbb{T}[\boldsymbol{X}]/\boldsymbol{X}^{\boldsymbol{N}} + \mathbf{1}$
  - $\boldsymbol{\rho} \in \mathbb{Z}/\boldsymbol{2N}\mathbb{Z}$ is the (encrypted) index input
- Output: The constant term of $\boldsymbol{X}^{-\rho} \cdot \boldsymbol{TV}[\boldsymbol{X}]$

$\boldsymbol{TV}[\boldsymbol{X}] = \sum_{i=0}^{N-1} \mu_i \cdot \boldsymbol{X}^i$

μ_0	μ_1	μ_2	μ_3	μ_4	μ_5	μ_6	μ_7

$\boldsymbol{X}^{-3} \cdot \boldsymbol{TV}[\boldsymbol{X}]$

μ_3	μ_4	μ_5	μ_6	μ_7	$-\mu_0$	$-\mu_1$	$-\mu_2$

Output: μ_3

Figure 7: Blind Rotate ($\boldsymbol{\rho = 3, N = 8}$)

Blind Rotate: LUT constraints

Possible LUTs for BR have two constraints.

1. Negacyclic Rotation: $X^{-\rho} \cdot TV[X] = -X^{-(\rho+N)} \cdot TV[X]$
2. Linear Combination: Only $\frac{t}{4}$ degrees of freedom for LUT entries

$X^{-3} \cdot TV[X]$

μ_3	μ_4	μ_5	μ_6	μ_7	$-\mu_0$	$-\mu_1$	$-\mu_2$

$X^{-8} \cdot TV[X]$

$-\mu_0$	$-\mu_1$	$-\mu_2$	$-\mu_3$	$-\mu_4$	$-\mu_5$	$-\mu_6$	$-\mu_7$

$X^{-(3+8)} \cdot TV[X]$

$-\mu_3$	$-\mu_4$	$-\mu_5$	$-\mu_6$	$-\mu_7$	μ_0	μ_1	μ_2

Figure 8: Negacyclic Rotation ($N = 8$)

Blind Rotate: LUT constraints

Possible LUTs for BR have two constraints.

1. Negacyclic Rotation: $X^{-\rho} \cdot TV[X] = -X^{-(\rho+N)} \cdot TV[X]$
2. Linear Combination: Only $\frac{t}{4}$ degrees of freedom for LUT entries
   - There are only $\frac{t}{2}$ possible values for ρ at the maximum.
   - p : number of inputs, m_i : encoded messages of inputs
   - $a_i \in \mathbb{Z}, b \in [-\frac{t}{2}, \frac{t}{2})$
   - $\rho \approx 2N \cdot (\sum_{i=0}^{p-1} a_i \cdot m_i + \frac{b}{t} \bmod 1)$

ex.) $p = 2, m_i \in \mathbb{M}_8, a_i = -1, b = 1$

$\Rightarrow \rho \approx 2N \cdot (-m_0 - m_1 + \frac{1}{8}) \in \{2N \cdot \frac{1}{8}, 2N \cdot \frac{3}{8}, -2N \cdot \frac{1}{8}\}$

$$TV[X] = \sum_{i=0}^{1} \sum_{j=0}^{\frac{N}{2}-1} \mu_i \cdot X^{i \cdot \frac{N}{2}+j}$$

μ_0	μ_0	μ_0	μ_0	μ_1	μ_1	μ_1	μ_1

Output for $\rho = 2N \cdot \frac{1}{8}$ Output for $\rho = 2N \cdot \frac{3}{8}$

AOI21

- AOI: AND OR Inverter
- $\neg((A \wedge B) \vee C)$
  - c is not interchangeable
    - Must be treated differently from a and b
    - $a_A = a_B = 1$, $a_C = 2, b = 1$

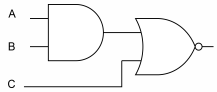


Figure 10: AOI21

Table 2: Output and $\frac{\rho}{2N}$ of AOI21

c\ab	00	01	11	10
0	$1/-\frac{3}{12}$	$1/-\frac{1}{12}$	$0/\frac{1}{12}$	$1/-\frac{1}{12}$
1	$0/\frac{1}{12}$	$0/\frac{3}{12}$	$0/\frac{5}{12}$	$0/\frac{3}{12}$

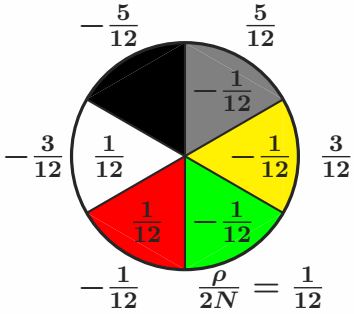


Figure 9: AOI21 LUT

Synthesis result using proposed gates

- Using our proposed gate we can reduce number of gate counts.
  - Reducing both **width** and **depth**.

Table 3: Results of synthesis by Yosys

Circuit (16-bit inputs)	# of gates (conventional)	# of gates ($\mathbb{M}_{\mathbf{12}}$)
Adder	83	16
Multiplier	1489	600

Conclusion

- TFHE is suitable for evaluating logic circuits in encrypted manner.
  - We can reuse existing tools and knowledge.
- The optimal circuit on TFHE is a bit different from physical circuits.
  - **width** should be taken into account in the circuit design.
  - The memory is heavy, so footprint should be reduced.
- Our platform can evaluate logic circuits with encrypted inputs in real environments.
  - ex. building a general processor.