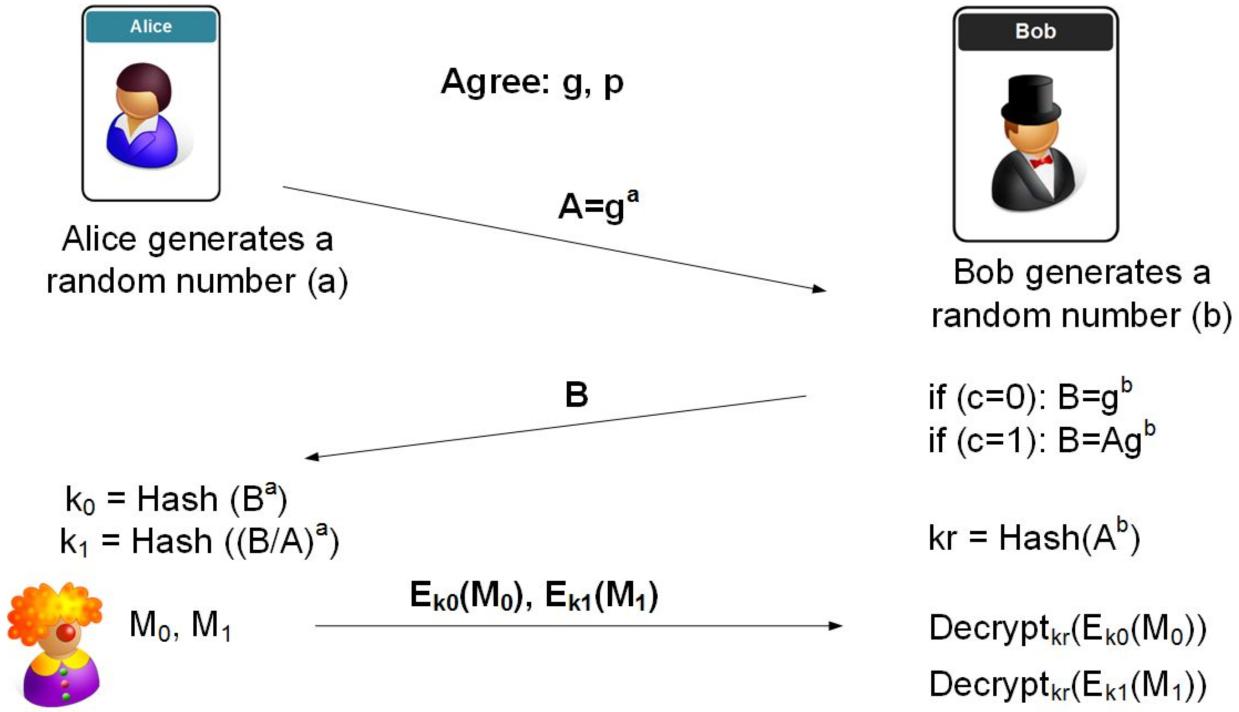
Silent OT on GPU

Zhifei Xie

Oblivious Transfer (OT)

- Sender has 2 private messages and can not reveal both to the receiver.
- Receiver requires one message from sender but can not reveal which one.
- Requires Public Key Cryptography, which can be costly when message is large.

Oblivious Transfer (OT)



Agree: g, p

Alice generates a random number (a)

$A=g^a$

Bob generates a random number (b)

if $(c=0)$: $B=g^b$
if $(c=1)$: $B=Ag^b$

B

k_0 = Hash (B^a)
k_1 = Hash $((B/A)^a)$

kr = Hash(A^b)

M_0, M_1

$E_{k0}(M_0)$, $E_{k1}(M_1)$

Decrypt$_{kr}(E_{k0}(M_0))$

Decrypt$_{kr}(E_{k1}(M_1))$

OT Extension

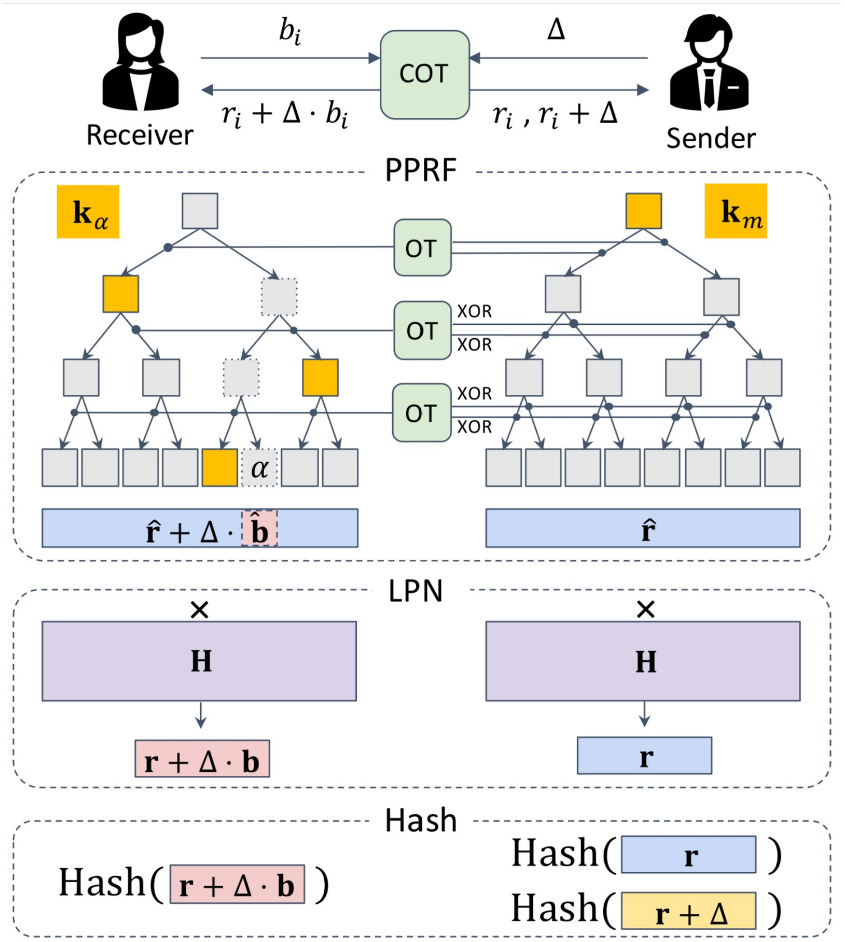
- Naive OT requires a lot of communication between sender and receiver. It is not ideal in Wide-Area Network (WAN) settings
- Use base OTs (with PKC) to securely communicate precomputed masks as part of offline pre-processing.
- Use these masks as one-time pads for actual messages during the online phase.
-

Silent OT

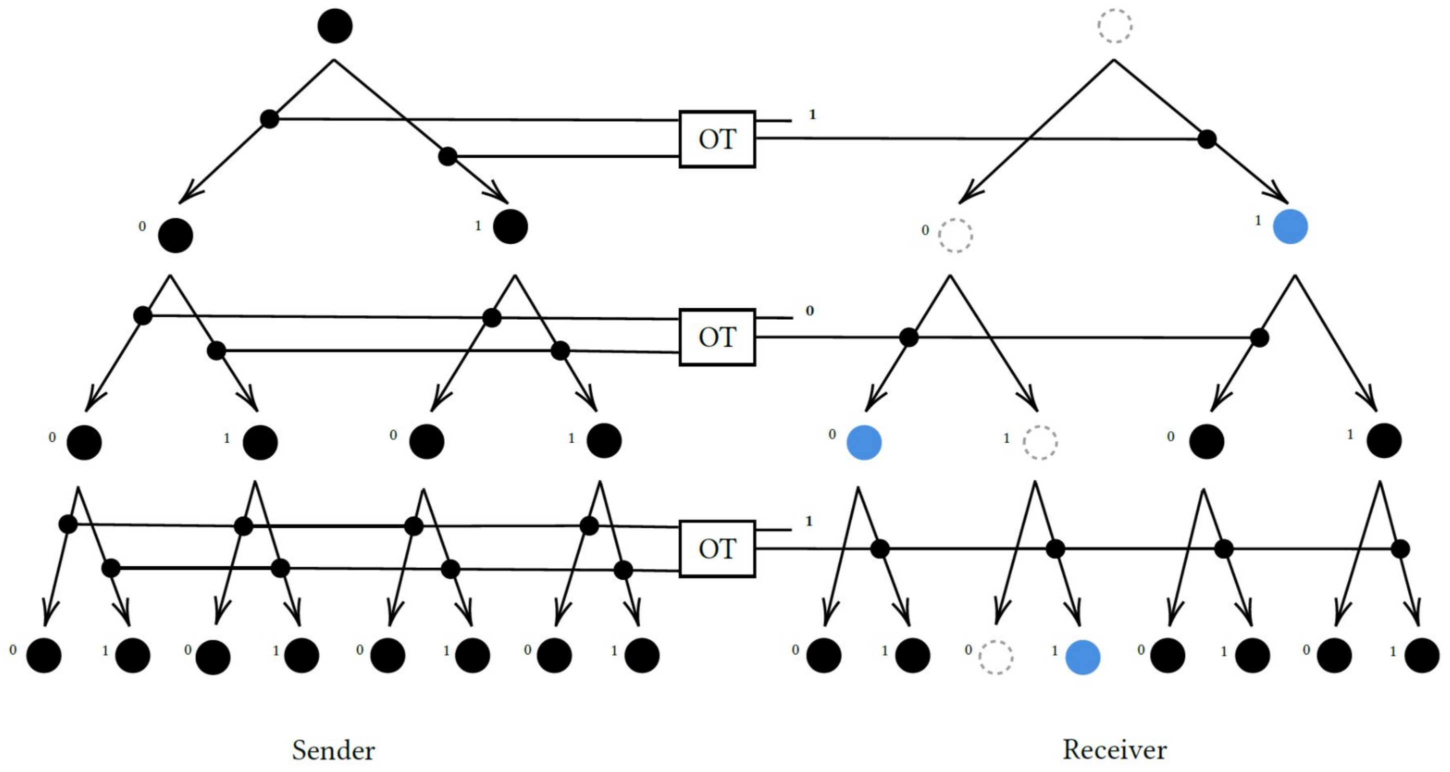
- Use base OTs (base OTs) to securely communicate short random seeds.
- Expand the seeds locally to generate the masks (OTPs).
- Use these OTPs during online phase.

Silent OT

- LPN was one of the most computationally costly part, but recent researches has address the issue and not a costly software program anymore
- PPRF is now the headache.

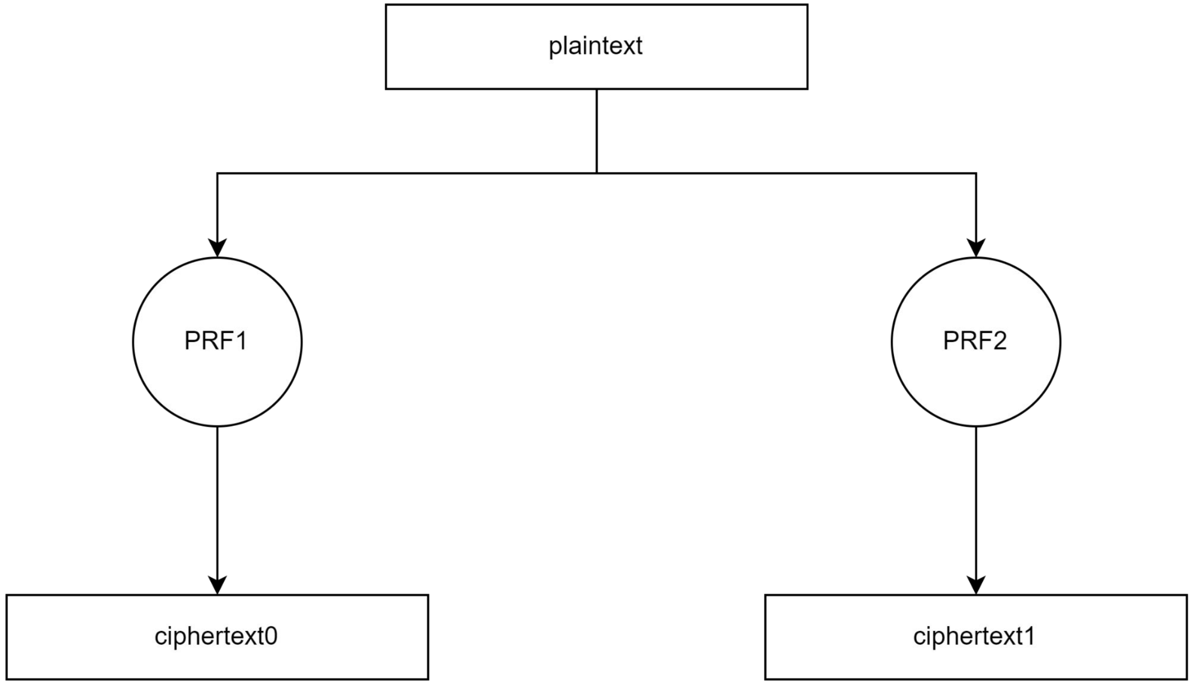


GGM Tree Based PPRF Expansion



Single Expand View

Pseudo-random Function used to do the expansion. It can be any function as long as it is mutually agreed and secure.

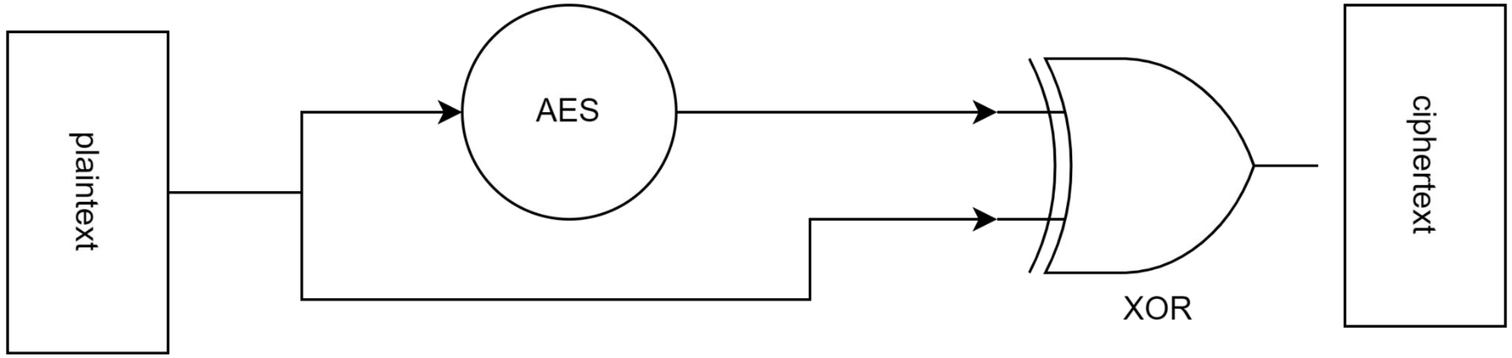


Potential GPU Benefit

- As shown in the diagram, the expansion function only takes the plaintext as input.
- Which means inside of one round of expansion, there is no dependency among PRFs; the input of PRFs only depends on the result of previous round of expansion, not on other PRFs in the same round.
- Huge opportunity for parallelism.

AES-128 ECB Mode Based PRF Function Units

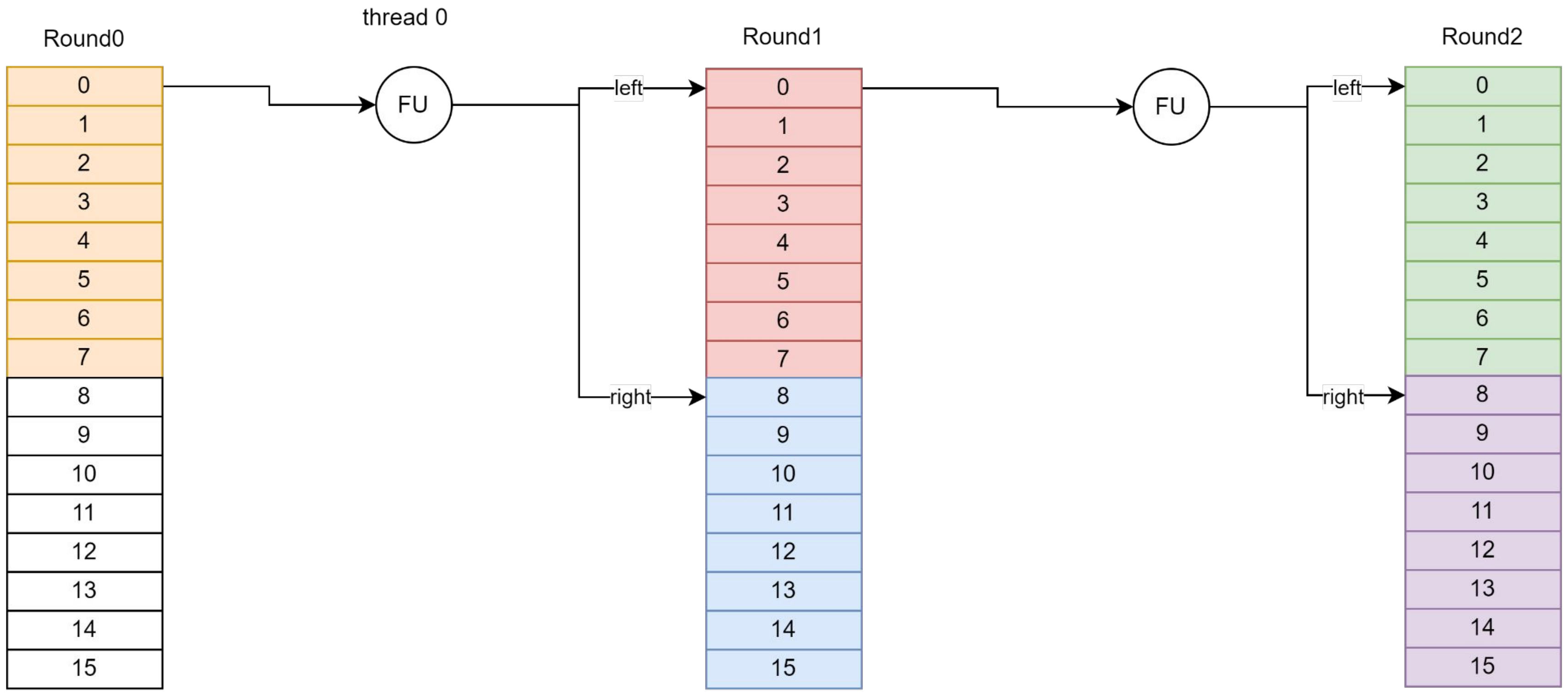
- AES 128 bit ECB mode is used for Pseudo-random function.
- It is widely used, easy to verify.
- Can gives same result from both parties.
- For more patterns, also XOR the plaintext and AES ciphertext to get the final ciphertext.



GPU Implementation

- The host CPU allocate mask memory on GPU
- Since the key is the same across all same type of PRFs (all left PRFs has one key and all right PRFs has one key), AES key expansion is done by CPU and stored in GPU memory since it is same across all same type of PRFs (all left PRFs has one key and all right PRFs has one key), reduces wasteful recomputation of key expansion
- A shared seed is transferred to GPU and GPU use it for PRNG generation (also AES-128 based) to get the initial 8 blocks (a block is 128 bit message)
- Then GPU using the initial 8 blocks to expand till the desire amount of mask blocks.
- Total memory transfer from host to GPU: 544 bytes. 3 expanded AES keys (11 blocks each, PRNG, left and right), plus the PRNG seed. Very few memory transfer for the nature of Silent OT.

GPU Implementation



Techniques in GPU Implementation: SMEM and Synchronization

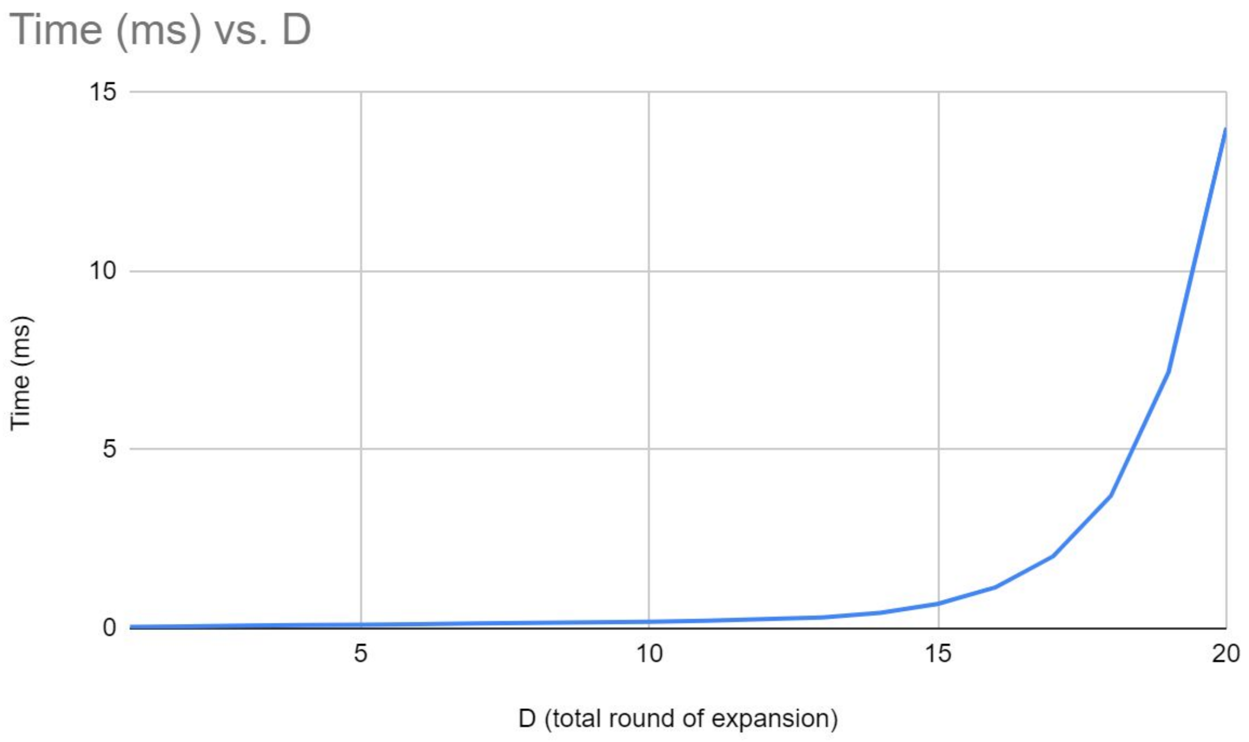
- Cannot let function units run for than rounds, otherwise it might read data that is not from the previous round
- Using Shared Memory to temporarily store the expanded value, and store them into the vector (in global memory) with __synthreads()

GPU Device and Benchmarking Method

- Host CPU: Intel i7-12700k, 32 GB DDR4-3200 CL14 memory
- GPU Device: RTX 4090, 16384 cuda cores, 24GB GDDR6X memory
- Following NVIDIA official technical blog https://developer.nvidia.com/blog/how-implement-performance-metrics-cuda-cc/, measuring runtime using events

GPU Results

D	Time (ms)
1	0.043529
2	0.056169
3	0.079953
4	0.092237
5	0.106799
6	0.121495
7	0.140188
8	0.157446
9	0.172282
10	0.190031
11	0.215159
12	0.258545
13	0.310219
14	0.434699
15	0.687522
16	1.152873
17	2.021111
18	3.713297
19	7.181577
20	14.01804



Comparing Result (v.s. CPU)

- Previous we run the expansion on i7-7700k, 32GB ram computer with D = 16
- CPU runtime is 148.2 ms, whereas GPU runtime is 1.15 ms, 129 times speedup

Comparing Result (v.s. FPGA)

- Previous, expansion is also implemented on FPGA.
- Fab1 and Fab8 version, which are on different FPGAs.
- 1.74 times slower than Fab8, 6.32 times speedup versus Fab1

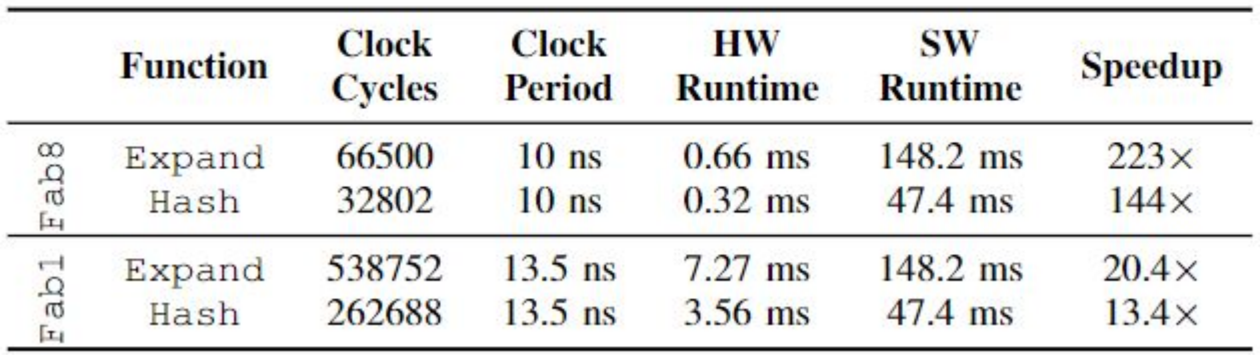


TABLE II
RUNTIME COMPARISON FABLIVIOUS

	Function	Clock Cycles	Clock Period	HW Runtime	SW Runtime	Speedup
Fab8	Expand	66500	10 ns	0.66 ms	148.2 ms	223×
	Hash	32802	10 ns	0.32 ms	47.4 ms	144×
Fab1	Expand	538752	13.5 ns	7.27 ms	148.2 ms	20.4×
	Hash	262688	13.5 ns	3.56 ms	47.4 ms	13.4×

Comparing Result (v.s. FPGA)

- However, GPU is much cheaper than FPGA, Fab8 FPGA (XCVU13P) costs more than $50,000; even the much smaller Fab1 FPGA (XCZU7CG) costs around $2,500. RTX 4090 generally cost around $1,650. Cheaper than XCZU7CG FPGA yet 6 times faster.

Other Advantages Comparing To FPGA

- Scalability. The GPU code and run on any cuda GPU without modifying anything. Whereas scaling the RTL on different FPGA can takes weeks of re-testing.
- Availability. There are a lot of GPU available since gaming PC is very common. Any PC with NVIDIA GPU can easily run this MPC protocol without any additional hardware.
- Versatility. The other part of the OT process can be transferred to GPU as well with maybe slightly performance lost but much less memory transfer to RAM. For example, we found that LPN is not suitable for FPGA implementation, so it has to run on CPU and the FPGA need to transfer the expanded data back to CPU.
- Memory Size. Even the $50,000 XCVU13P has only less than 450MB of on-board memory total, whereas a 2016 GTX 1060 has 6GB of memory. Less memory means more memory transfer, which can be a bottleneck for many programs.