Efficient Homomorphic Evaluation of Arbitrary Bivariate Integer Functions

Daisuke Maeda, Koki Morimura, Takashi Nishide

University of Tsukuba, Japan

WAHC'22, 7th Nov. 2022

https://dl.acm.org/doi/10.1145/3560827.3563378

Introduction

Fully Homomorphic Encryption (FHE)

- FHE: encryption scheme allows addition and multiplication on ciphertexts
  - $[\![a]\!] \oplus [\![b]\!] = [\![a + b]\!]$
  - $[\![a]\!] \otimes [\![b]\!] = [\![a \times b]\!]$

$[\![a]\!]$: Ciphertext of plaintext a

- We focus on BFV with packing method [SV14]:
  - Supports integer-wise operations
    - Easily calculate $\oplus$ and $\otimes$ without bit/digit encryption
  - Supports packing method (i.e., SIMD, slot-wise computation)
  - Non-trivial to compute arbitrary non-linear functions
    - Polynomial evaluation by representing f as a polynomial via polynomial interpolation is one standard approach

[SV14] Nigel P Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. Designs, codes and cryptography, 71(1):57–81, 2014.

Contributions

Existing work

- [OCHK18] achieves homomorphic evaluation of arbitrary bivariate integer functions
- Inefficient when input domain size is relatively large
  - About 59Days for 15-bit integer division in our environment

Our Contributions

- Proposed two new methods for homomorphic evaluation of arbitrary bivariate integer functions
  - ① An improved method of prior work [OCHK18]
  - ② A new approach for relatively large input domain size instead of enabling the SIMD operation

[OCHK18] Okada, Hiroki, et al. "Linear depth integer-wise homomorphic division." WISTP. Springer, 2018..

Prior Work

Prior Work [OCHK18]

Okada et al. shows how to homomorphically compute an arbitrary bivariate integer function based on polynomial evaluation.

Algorithm 1 Homomorphic Division $[\![\lfloor \frac{a}{d} \rfloor]\!]$

Input: $[\![a]\!]$, $[\![d]\!] (a, d \in \mathrm{Z}_{t'})$
Output: $[\![\lfloor \frac{a}{d} \rfloor]\!]$

1: $S \leftarrow [\![0]\!]$
2: $C_a^{pow} \leftarrow \mathrm{Pows}([\![a]\!], t'),\ C_d^{pow} \leftarrow \mathrm{Pows}([\![d]\!], t')$
3: **for** $i = 0$ to $t' - 1$ **do**
4: $\quad [\![\lfloor \frac{a}{i} \rfloor]\!] = \mathrm{ConstDiv}(C_a^{pow}, i)$
5: $\quad [\![i = d]\!] = \mathrm{ConstEq}(C_d^{pow}, i)$
6: $\quad S \leftarrow S \oplus [\![\lfloor \frac{a}{i} \rfloor]\!] \otimes [\![i = d]\!]$
7: **end for**
Output: $S = [\![\lfloor \frac{a}{d} \rfloor]\!]$

- $\mathrm{Pows}([\![a]\!], t'): \{[\![a]\!], [\![a^2]\!], \cdots, [\![a^{t'-1}]\!]\}$, $\mathrm{Pows}([\![d]\!], t')$
  - t' invocations of $ct \otimes ct$, respectively
- ConstDiv ($[\![\lfloor \frac{a}{i} \rfloor]\!]$), ConstEq ($[\![i = d]\!]$)
  - Polynomial evaluation using powers
- $[\![\lfloor \frac{a}{i} \rfloor]\!] \otimes [\![i = d]\!] = \begin{cases} [\![\lfloor \frac{a}{d} \rfloor]\!], & (\text{if } i = d) \\ [\![0]\!], & (\text{otherwise}) \end{cases}$
- $\sum_{i=0}^{t'} [\![\lfloor \frac{a}{i} \rfloor]\!] \otimes [\![i = d]\!] = [\![\lfloor \frac{a}{d} \rfloor]\!]$

Our Algorithm

Proposal 1

Proposal 1 halves # of polynomial evaluations compared to [OCHK18]

Input: $[\![a]\!]$, $[\![d]\!]$ $(a, d \in \mathbb{Z}_{t'})$

Output: $\left[\!\left[\left\lfloor \frac{a}{d} \right\rfloor\right]\!\right]$

- If d is fixed, we can precompute coefficients $(c_{0,d}, c_{1,d}, \cdots, c_{t'-1,d})$ of $f_d(x)\colon \mathbb{Z}_{t'} \to \mathbb{Z}_{t'}$ via polynomial interpolation.

$$f_d(x) = \left\lfloor \frac{x}{d} \right\rfloor = c_{0,d} + c_{1,d}x + \cdots + c_{t'-1,d}x^{t'-1} \bmod t$$

- **Also,** for $0 \le j \le t' - 1$ we define $g_j(x)\colon \mathbb{Z}_{t'} \to \mathbb{Z}_{t'}$ that, given d as x, returns j-th coefficient $c_{j,d}$ of $f_d(x)$.

$$g_j(x) = c'_{0,j} + c'_{1,j}x + \cdots + c'_{t'-1,j}x^{t'-1} \bmod t$$

Proposal 1

$$\left\lfloor \frac{a}{d} \right\rfloor = f_d(a)$$

$$f_d(x) = c_{0,d} + c_{1,d}\, x + c_{2,d}\, x^2 + \cdots + c_{t'-1,d}\, x^{t'-1} \bmod t$$

Proposal 1

$$\left\lfloor \frac{a}{d} \right\rfloor = f_d(a)$$

$$f_d(x) = c_{0,d} + c_{1,d}\,x + c_{2,d}\,x^2 + \cdots + c_{t'-1,d}\,x^{t'-1} \bmod t$$

$$g_0(x) = c'_{0,0} + c'_{1,0}x + \cdots + c'_{t'-1,0}x^{t'-1} \bmod t$$

← to compute $c_{0,d} = g_0(d)$

Proposal 1

$$\left\lfloor \frac{a}{d} \right\rfloor = f_d(a)$$

$$f_d(x) = c_{0,d} + c_{1,d}\,x + c_{2,d}\,x^2 + \cdots + c_{t'-1,d}\,x^{t'-1} \bmod t$$

$$g_0(x) = c'_{0,0} + c'_{1,0}x + \cdots + c'_{t'-1,0}x^{t'-1} \bmod t$$

$$g_1(x) = c'_{0,1} + c'_{1,1}x + \cdots + c'_{t'-1,1}x^{t'-1} \bmod t$$

← to compute $c_{1,d} = g_1(d)$

Proposal 1

$$\left\lfloor \frac{a}{d} \right\rfloor = f_d(a)$$

$$f_d(x) = c_{0,d} + c_{1,d}\,x + c_{2,d}\,x^2 + \cdots + c_{t'-1,d}\,x^{t'-1} \bmod t$$

$$g_0(x) = c'_{0,0} + c'_{1,0}x + \cdots + c'_{t'-1,0}x^{t'-1} \bmod t$$

$$g_1(x) = c'_{0,1} + c'_{1,1}x + \cdots + c'_{t'-1,1}x^{t'-1} \bmod t$$

$$g_2(x) = c'_{0,2} + c'_{1,2}x + \cdots + c'_{t'-1,2}x^{t'-1} \bmod t$$

$$g_{t'-1}(x) = c'_{0,t'-1} + c'_{1,t'-1}x + \cdots + c'_{t'-1,t'-1}x^{t'-1} \bmod t$$

Total: t' Polynomial Evaluations

Proposal 1

- Proposal 1 requires **t' polynomial evaluations**, which **halves** the computational cost of [OCHK18]
  - [OCHK18] computes $2t$' polynomial evaluations (t' invocations of equality check and division)
  - Overall execution time is also halved

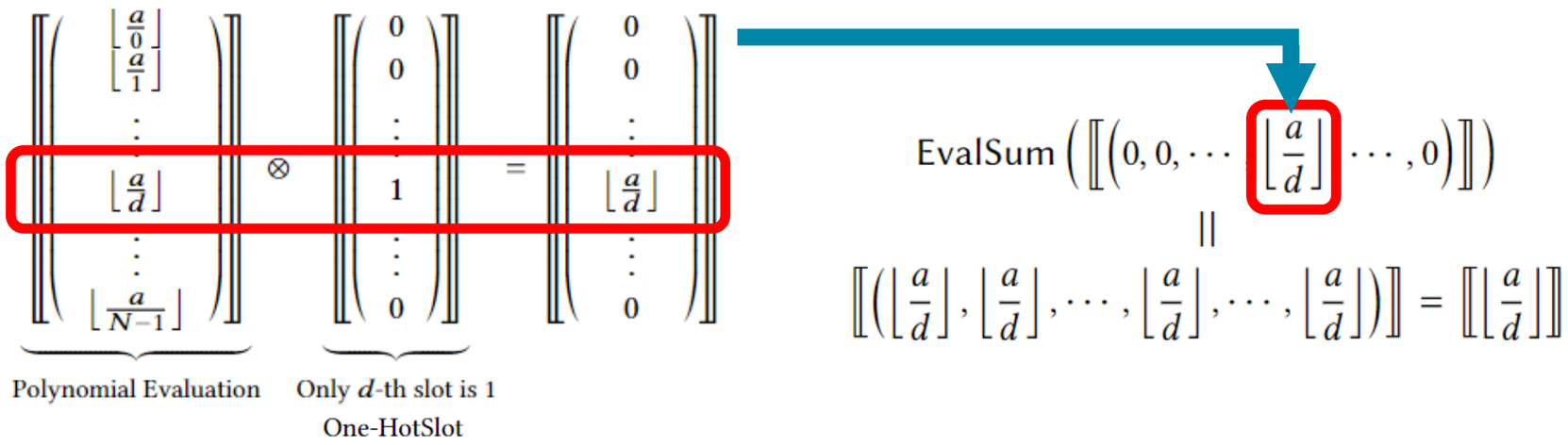
But…

- The larger domain size, the slower computation
  - Run-time of each operation is small, but total # of these is quadratic in t'
    - Each polynomial evaluation includes t' invocations of $pt \otimes ct$ and $ct \oplus ct$
  - **When $t' = 2^{15}$**, estimation of total computation is **about 59 days** in our experimental environment
    - Details will be discussed later
- Packing method allows SIMD-style parallel processing of bivariate functions, but it needs huge time to get all the results

Overview of Proposal 2

Efficient way to compute bivariate functions **for relatively large input domain size** instead of enabling SIMD operation

- Building Blocks
  - **EvalSum**: computes sum of values in each slot
  - **One-HotSlot**: given $[\![d]\!]$, computes a packed ciphertext in which only the d-th slot is 1 and the others are 0
- Rough Overview of Proposal 2 (Case of Homomorphic Division $\left[\!\left[\left\lfloor \frac{a}{d} \right\rfloor\right]\!\right]$)



EvalSum

Notation	Description
$t,\ N$	t: plaintext modulus s.t. $2N \mid t-1$. Typical settings: $N = 2^{15}, t = 2N+1$
$\sigma_i(\cdot)$	Automorphism mapping for $i \in \mathbb{Z}^*_{2N}$
$[\![(a'_0, a'_1, \ldots, a'_{N-1})]\!]$	Packed ciphetext of $[\![f(X)]\!]$

Input: $[\![(a'_0, a'_1, \cdots, a'_{N-1})]\!]$: Packed ciphertext of polynomial $f(X)$

Output: $[\![S]\!] = [\![(S, S, \cdots, S)]\!]$ $\left(S = \sum_{i=0}^{N-1} a'_i\right)$

- Invokes $\log_2(N)$ automorphism mappings and $ct \oplus ct$ to compute $N \times f(0)$

Proposition

Let $[\![(a'_0, a'_1, \cdots, a'_{N-1})]\!]$ as packed ciphertext of $[\![f(X)]\!]$.

$$\left[\!\!\left[\sum_{i=0}^{N-1} a'_i\right]\!\!\right] = [\![N \times f(0)]\!]$$

Algorithm 3 EvalSum

Input: $[\![f(X)]\!] = [\![(a'_0, a'_1, \ldots, a'_{N-1})]\!]$

1: $c \leftarrow [\![f(X)]\!]$
2: **for** $k = 0$ to $\log_2(N) - 1$ **do**
3: $\quad c' \leftarrow \sigma_{\frac{N}{2^k}+1}(c)$
4: $\quad c \leftarrow c \oplus c'$
5: **end for**

Output: $c\ (= [\![N \cdot f(0)]\!] = [\![\sum_{i=0}^{N-1} a'_i]\!]$
$= [\![(\sum_{i=0}^{N-1} a_i, \sum_{i=0}^{N-1} a_i, \ldots, \sum_{i=0}^{N-1} a_i)]\!])$

✓ No additional $ct \otimes ct$

- does not consume multiplicative depth

✗ About 6 times slower than run-time of $ct \otimes ct$ in our environment

One-HotSlot

Input: $[\![a]\!] = [\![(a, a, \cdots a)]\!]$ $(a \in \mathbb{Z}_N)$

Output: $\underbrace{[\![(0, \cdots, 0, 1, 0, \cdots, 0)]\!]}_{\text{only } a\text{-}th \text{ slot is } 1}$

- ℓ^{t-1}: computed by repeated squaring modulo t
  - Multiplicative depth = $\log_2(t-1)$
    - fixed regardless of input domain size
- One-Hot packed ciphertext can be used for table lookup

Algorithm 2 One-HotSlot

Input: Packed ciphertext $[\![a]\!] = [\![(a, a, \ldots, a)]\!]$

1: $T = [(0, 1, 2, \ldots, N-1)]$

2: $\ell \leftarrow [\![a]\!] \ominus T$

$= [\![\underbrace{(a, a-1, a-2, \ldots, 0, \ldots, a-N+1)}_{\text{Only } a\text{-th slot is } 0}]\!]$

3: $m \leftarrow \ell^{t-1}$

$= [\![(1, 1, \ldots, 0, 1, \ldots, 1)]\!]$

4: $n \leftarrow [(1, 1, \ldots, 1)] \ominus m$

$= \underbrace{[\![(0, 0, \ldots, 1, 0, \ldots, 0)]\!]}_{\text{Only } a\text{-th slot is } 1}$

Output: n

Homomorphic Evaluation of Arbitrary Univariate Function

Input: $[\![a]\!], T = [f(0), f(1), \cdots, f(N-1)]$ $(a \in \mathbb{Z}_N)$

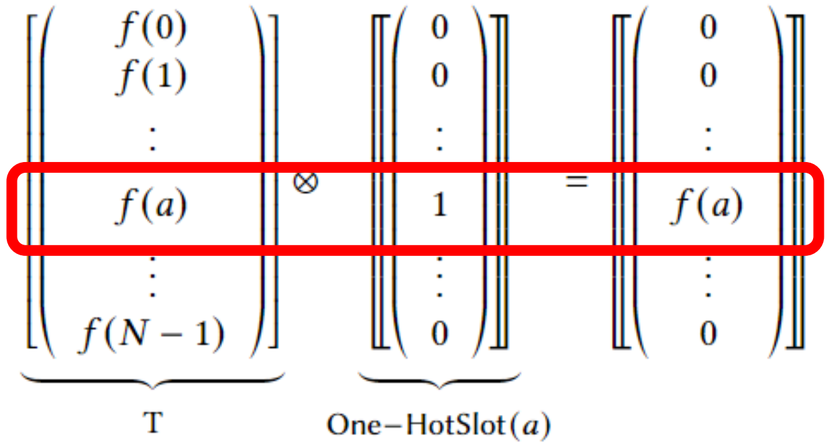
Output: $[\![f(a)]\!]$

Algorithm 4 Homomorphic Evaluation of Arbitrary Univariate Function

Input: $[\![a]\!] = [\![(a, a, \ldots, a)]\!]$ and packed plaintext $T = [(f(0), f(1), \ldots, f(N-1))]$ representing a lookup table of function $f: \mathbb{Z}_N \to \mathbb{Z}_N$

1: $\ell \leftarrow$ One-HotSlot($[\![a]\!]$)
2: $m \leftarrow \ell \otimes T = \ell \otimes [(f(0), f(1), \ldots, f(N-1))]$
3: $n \leftarrow$ EvalSum(m)

Output: $n = [\![f(a)]\!]$



$$\text{EvalSum}([\![(0, 0, \cdots, \underbrace{f(a)}_{a\text{-th slot}}, \cdots, 0)]\!])$$

$$\| $$

$$[\![(f(a), f(a), \cdots, f(a))]\!] = [\![f(a)]\!]$$

- **Why using One-HotSlot & EvalSum?**
  - One-hot Packed ciphertext looks simple at glance
    - It is actually a complex polynomial
  - By applying EvalSum, we can get a polynomial with only a constant term
    - I.e., ciphertext of just an integer

Proposal 2

Input: $[\![a]\!], [\![d]\!]$ $(a, d \in \mathbb{Z}_N)$

Output: $\left[\!\left[\left\lfloor \frac{a}{d} \right\rfloor\right]\!\right]$

0. Define polynomial $f_d(x) = \left\lfloor \frac{x}{d} \right\rfloor = c_{0,d} + c_{1,d}x + \cdots + c_{N-1,d}x^{N-1} \bmod t$ and coefficients $c_{i,d}$ can be precomputed via polynomial interpolation

1. Precompute all the coefficients of $f_0(x), \cdots, f_{N-1}(x)$ and make packed plaintexts[1] c_i as follows:

$$c_0 = \left[\left(c_{0,0}, c_{0,1}, \ldots, c_{0,d}, \ldots, c_{0,N-1}\right)\right]$$ ← **0-th coefficients of all the polynomials $f_0(x)$**

$$c_1 = \left[\left(c_{1,0}, c_{1,1}, \ldots, c_{1,d}, \ldots, c_{1,N-1}\right)\right]$$ ← **1-st coefficients of all the polynomials $f_1(x)$**

$$\vdots$$

$$c_{N-1} = \left[\left(c_{N-1,0}, c_{N-1,1}, \ldots, c_{N-1,d}, \ldots, c_{N-1,N-1}\right)\right]$$ ← **$N-1$-th coefficients of all the polynomials $f_{N-1}(x)$**

1) To make packed plaintext, MakePackedPlaintext() function can be used in PALISADE library.

Proposal 2

Lemma (Paterson-Stockmeyer method)

Polynomial of degree N can be computed by using $O(\sqrt{N})$ mult.

2. Compute polynomial evaluation $\boldsymbol{f_0(x), \cdots, f_{N-1}(x)}$ **in parallel** using PS-method where $p, s \approx \sqrt{N}$

$$
\begin{aligned}
&[\![(f_0(a), \ldots, f_{d_i}(a), \ldots f_{N-1}(a))]\!] \leftarrow \\
&\quad (c_0 + c_1[\![a]\!] + \cdots + c_{p-1}[\![a^{p-1}]\!]) \\
&\quad + \left(c_p + c_{p+1}[\![a]\!] + \cdots + c_{2p-1}[\![a^{p-1}]\!]\right) \times [\![a^p]\!] \\
&\quad + \left(c_{2p} + c_{2p+1}[\![a]\!] + \cdots + c_{3p-1}[\![a^{p-1}]\!]\right) \times [\![(a^p)^2]\!] \\
&\qquad\qquad \vdots \\
&\quad + \left(c_{(s-1)p} + c_{(s-1)p+1}[\![a]\!] + \cdots + c_{(s-1)p+p-1}[\![a^{p-1}]\!]\right) \times [\![(a^p)^{s-1}]\!]
\end{aligned}
\quad \Rightarrow \quad
[\![(f_0(a), f_1(a), \ldots, f_d(a), \ldots, f_{N-1}(a))]\!] = [\![(\lfloor \tfrac{a}{0} \rfloor, \lfloor \tfrac{a}{1} \rfloor, \ldots, \lfloor \tfrac{a}{d} \rfloor, \ldots, \lfloor \tfrac{a}{N-1} \rfloor)]\!]
$$

- PS-method decreases # of powers for polynomial evaluation
  - $N \rightarrow 2\sqrt{N}$

Proposal 2

3. By exploiting One-HotSlot, we can compute a bivariate function similarly to a **uni**variate function

$$\left[\!\!\left[\begin{pmatrix} \lfloor \frac{a}{0} \rfloor \\ \lfloor \frac{a}{1} \rfloor \\ \vdots \\ \lfloor \frac{a}{d} \rfloor \\ \vdots \\ \lfloor \frac{a}{N-1} \rfloor \end{pmatrix}\right]\!\!\right] \otimes \underbrace{\left[\!\!\left[\begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix}\right]\!\!\right]}_{\mathrm{One-HotSlot}(d)} = \left[\!\!\left[\begin{pmatrix} 0 \\ 0 \\ \vdots \\ \lfloor \frac{a}{d} \rfloor \\ \vdots \\ 0 \end{pmatrix}\right]\!\!\right]$$

$$\mathrm{EvalSum}\left(\left[\!\!\left[\left(0, 0, \cdots, \left\lfloor \frac{a}{d} \right\rfloor, \cdots, 0\right)\right]\!\!\right]\right)$$

$$\| $$

$$\left[\!\!\left[\left(\left\lfloor \frac{a}{d} \right\rfloor, \left\lfloor \frac{a}{d} \right\rfloor, \cdots, \left\lfloor \frac{a}{d} \right\rfloor, \cdots, \left\lfloor \frac{a}{d} \right\rfloor\right)\right]\!\!\right] = \left[\!\!\left[\left\lfloor \frac{a}{d} \right\rfloor\right]\!\!\right]$$

Here we focused on division, but arbitrary bivariate integer functions can be computed by modifying the part of polynomial interpolation.

Complexity Analysis

Complexity Comparison of Computing $f: \mathbb{Z}_{t'} \times \mathbb{Z}_{t'} \to \mathbb{Z}_{t'}$

of each operation

	[OCHK18]	Proposal 1	Proposal 2
$pt \times ct$	$2t'^2$	t'^2	t'
$ct + ct$	$2t'^2 + t'$	$t'^2 + t'$	$t' + \log_2(N) + s$
$ct \times \mathrm{ct}$	$3t'$	$3t'$	$\log_2(t-1) + p + 2s$
MakePackedPlaintext	$2t'^2$	t'^2	t'

($p, s \approx \sqrt{t'}$ for PS method)

- In [OCHK18] and Proposal 1, each polynomial evaluation includes t'^2 invocations of $pt \otimes ct, ct \oplus ct$, and MakePackedPlaintext
  - The number of these operations in Proposal 1 is half compared to [OCHK18]
  - When $t' = 2^{15}$, # of these operation becomes quite large
- No $O(t'^2)$ operations in Proposal 2
  - Computed with PS method, One-HotSlot, and EvalSum
  - Leading to the efficiency gap among the schemes

Implementation Results

Experiment Environment

- PC: Ryzen 5 3600@3.6GHz and 128GB RAM
- Library: PALISADE (ver.1.10.6) multi-threads
- FHE Scheme: BFVrns
- Security Parameter: SecurityLevel = HEStd_128_classic

Implementation

- ℓ: input domain size 2^{ℓ}
- L_i: Level parameter for
  - L_0: [OCHK18]
  - L_1: Proposal 1
  - L_2: Proposal 2

ℓ	L_0	L_1	L_2	[OCHK18](sec)	Proposal 1(sec)	Proposal 2(sec)
3	4	4	17	0.65	0.61	10.74
4	5	5	17	1.75	1.27	11.00
5	6	6	17	6.44	3.76	11.74
6	7	7	17	18.89	10.96	12.56
7	8	8	17	69.62	36.35	14.52
8	9	9	17	268.33	131.82	16.01
9	10	10	17	1083.41	537.65	20.79
10	11	11	17	4236.46	2140.61	25.90
11	12	12	17	43703.7	21499.1	38.86
12	-	-	17	-	-	57.49
13	-	-	17	-	-	98.72
14	-	-	17	-	-	163.28
15	-	-	17	-	-	306.93

58.7 days
EST

- Proposal 1 can support SIMD, so achieve higher throughput, but its latency (response time) is prohibitively high with large input domain size
  - Proposal 2 cannot support SIMD, but its latency is much lower

References

[SV14] Nigel P Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. Designs, codes and cryptography, 71(1):57–81, 2014.

[OCHK18] Okada, Hiroki, et al. "Linear depth integer-wise homomorphic division." IFIP. Springer, Cham, 2018..