RSAConference2020

San Francisco | February 24 – 28 | Moscone Center

HUMAN ELEMENT

SESSION ID: **CRYP-R01**

Better Bootstrapping for Approximate Homomorphic Encryption

Dohyeong Ki

Seoul National University

#RSAC

Need for a Faster Homomorphic Encryption

Modern Data Analysis Algorithms

Machine Learning Algorithms: Covolutional Neural Network (CNN), Deep Neural Network (DNN), etc.

Extremely Complicated

Require faster homomorphic operations and faster boostrapping

What We Do

- Suggest a generalized key-switching method for the Full-RNS variant of HEAAN
- Propose a new polynomial approximation method to evaluate a sine function, which is specialized for the bootstrapping for HEAAN
- Give first implementation of bootstrapping for Full-RNS variant of approximate homomorphic encryption

RSAConference2020

Generalized Key-switching Method for the Full-RNS Variant of HEAAN

Preliminary

Full-RNS variant of HEAAN

$C = \{q_0, q_1, \ldots, q_L\}$: Base primes $\approx q$

$B = \{p_0, p_1, \ldots, p_k\}$: Temporary Moduli

→ Reduce the size of error generated from key-switching

$$P = \prod_{i=0}^{k} p_i \gg Q = \prod_{j=0}^{L} q_j$$

→ $k \approx L$

SEAL (v3.3 -)

One temporary modulus + RNS decomposition method → $k = 1$

Motivation

- k ⇧ (assume the same security level)

Advantages

1. The number of evaluation keys for key-swtiching ⇩
2. Complexity for key-switching ⇩

Disadvantages

1. The number of levels supported ⇩

Key Ideas

[1] Temporary modulus technique

$B = \{p_0, p_1, \ldots, p_k\}$: Temporary Moduli

$C = \{q_0, q_1, \ldots, q_L\}$: Base primes $\approx q$

$$\mathcal{C}' = \{Q_j\}_{0 \le j < \mathtt{dnum}} = \left[\prod_{i=j\alpha}^{(j+1)\alpha-1} q_i \right]_{0 \le j < \mathtt{dnum}}$$

for the given integer $\mathtt{dnum} > 0$ and $\alpha = (L+1)/\mathtt{dnum}$.

$$P = \prod_{i=0}^{k} p_i \gg \max_{0 \le j < dnum} Q_j \;\Rightarrow\; k \approx \alpha$$

Key Ideas

[2] RNS-decomposition method

$$\text{RNS-Decomp}_{\mathcal{C}}(a(x)) = ([a(x)\cdot \hat{q_0}^{-1}]_{q_0}, \ldots, [a(x)\cdot \hat{q_L}^{-1}]_{q_L}) \in R^{L+1}$$

$$\text{RNS-Power}_{\mathcal{C}}(b(x)) = (b(x)\cdot \hat{q}_0, b(x)\cdot \hat{q}_1, \ldots, b(x)\cdot \hat{q}_L) \in R^{L+1},$$

$$\text{where } \mathcal{C} = \{q_0, q_1, \ldots, q_L\},\ \hat{q}_i = \textstyle\prod_{j\neq i} q_j \text{ and } a(x), b(x) \in R$$

$$\Rightarrow a(x)\cdot b(x) = \langle \text{RNS-Decomp}_{\mathcal{C}}(a(x)), \text{RNS-Power}_{\mathcal{C}}(b(x))\rangle \in R_Q$$

Can perform key-switching as

$$\langle \text{RNS-Decomp}_{\mathcal{C}}(a(x)), \boxed{\text{RNS-Power}_{\mathcal{C}}(s(x)^2) + \mathtt{Enc}_{s(x)}(0)}\rangle \bmod Q$$

Complexity of Homomorphic Multiplication

Total Complexity for Homomorphic Multiplication

$$\approx N \cdot \{(l + \log N) \cdot k + (2 + \log N) \cdot l^2 \cdot (1/k)\} + (constant)$$

if we set k to α and regard other parameters as constants

The total complexity is minimized when $k = \sqrt{\frac{2+\log N}{l+\log N}} \cdot l$

RSAConference2020

A New Polynomial Approximation Method to Evaluate a Sine Function

- Specialized to the Bootstrapping for HEAAN

Preliminary : Bootstrapping for HEAAN

Goal

$m(X) = [\langle \mathsf{ct}, \mathsf{sk} \rangle]_q \Longrightarrow m(X) = [\langle \mathsf{ct}', \mathsf{sk} \rangle]_Q \quad Q > q$

Steps

$ct = Enc\big(m(X)\big)(mod\ q)$

Modulus Raising

$ct = Enc\big(t(X)\big)(mod\ Q_0)$

$where\ t(X) = qI(X) + m(X)$

Coefficients to Slots

$Enc\big(t_0, \ldots, t_{N/2-1}\big)$
$Enc\big(t_{N/2}, \ldots, t_{N-1}\big)$

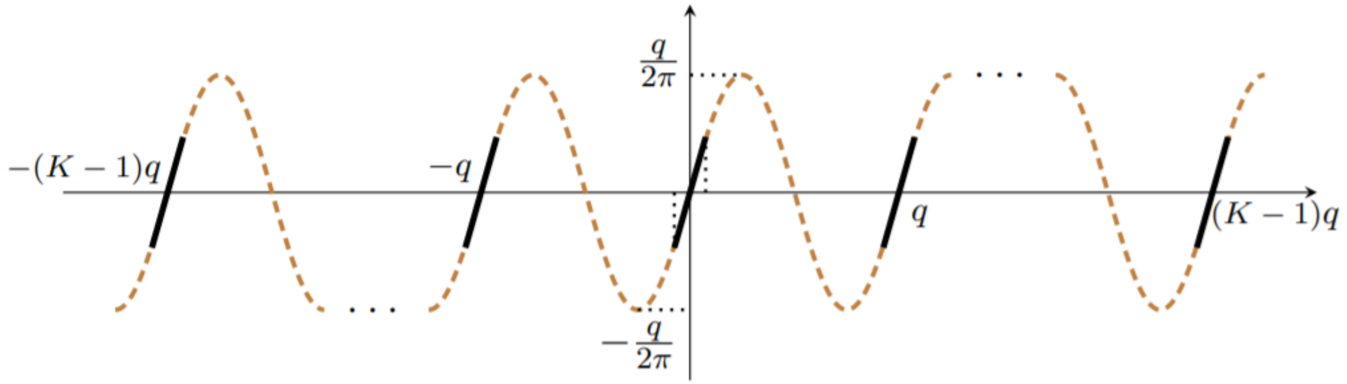
Sine Evaluation

$Enc\big(m_0, \ldots, m_{N/2-1}\big)$
$Enc\big(m_{N/2}, \ldots, m_{N-1}\big)$

Slots to Coefficients

$ct' = Enc\big(m(X)\big)(mod\ Q)$

Previous Works



$$[t]_q \simeq \frac{q}{2\pi}\sin\left(\frac{2\pi}{q}t\right) \simeq p(t)$$

Polynomial

1. [CHKKS18] : Taylor Approximation Method
2. [CCS 19] : Chebyshev Approximation Method

[CHKKS18] : Cheon et al., Bootstrapping for Approximate Homomorphic Encryption, *Eurocrypt,* 2018.
[CCS19] : Chen, H., Ilaria Chillotti and Yongsoo Song, Improved Bootstrapping for Approximate Homomorphic Encryption, *Eurocrypt,* 2019.

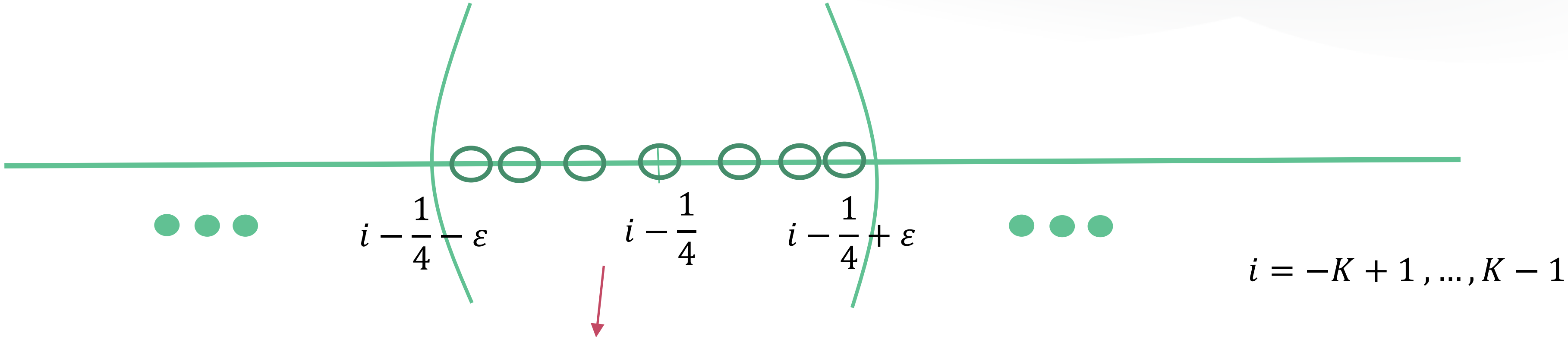
Key Ideas

Theorem 1 (polynomial interpolation). *Let f be a function in $C^{n+1}[a,b]$ and p_n be a polynomial of degree $\leq n$ that interpolates the function f at $n+1$ distinct points $t_0, t_1, \cdots, t_n \in [a,b]$, i.e. $p_n(t_i) = f(t_i)$ for all $0 \leq i \leq n$. Then, for each $t \in [a,b]$, there exists a point $\psi_t \in [a,b]$ such that*

$$f(t) - p_n(t) = \underbrace{\frac{f^{(n+1)}(\psi_t)}{(n+1)!}}_{\text{Bounded}} \cdot \boxed{\prod_{i=0}^{n}(t - t_i).} \tag{2}$$

Goal : Choose appropriate t_i's that minimize the maximum value of $w(t) = \prod_{i=0}^{n}(t - t_i)$ in a specified domain of t.

Our Method



Choose d_i nodes as the Chebyshev method

$$t_{i,j} = i - \tfrac{1}{4} + \epsilon \cdot \cos\left(\frac{2j-1}{2d_i}\pi\right) \qquad 1 \leq j \leq d_i$$

$$||w(t)|| \leq \frac{1}{2^{d_i - 1}} \cdot \epsilon^{d_i} \cdot \prod_{j=1}^{K-1-i} (j+\epsilon)^{d_{i+j}} \cdot \prod_{j=1}^{K-1+i} (j+\epsilon)^{d_{i-j}}$$

$$\text{when } t \in I_i = [i - \tfrac{1}{4} - \epsilon, i - \tfrac{1}{4} + \epsilon]$$

How to Choose the Number of Nodes in each Interval?

Algorithm 1 Choosing the number of nodes in each interval

Input : Target degree n
Initialize $d_i = 1$ for all i
while $\sum d_i \leq n$ **do**
  Compute M_i for each i
  Find $i_0 = \text{argmax } M_i$
  $d_{i_0} \leftarrow d_{i_0} + 1$
end while
Output : d_i's

Comparison with the Chebyshev Method

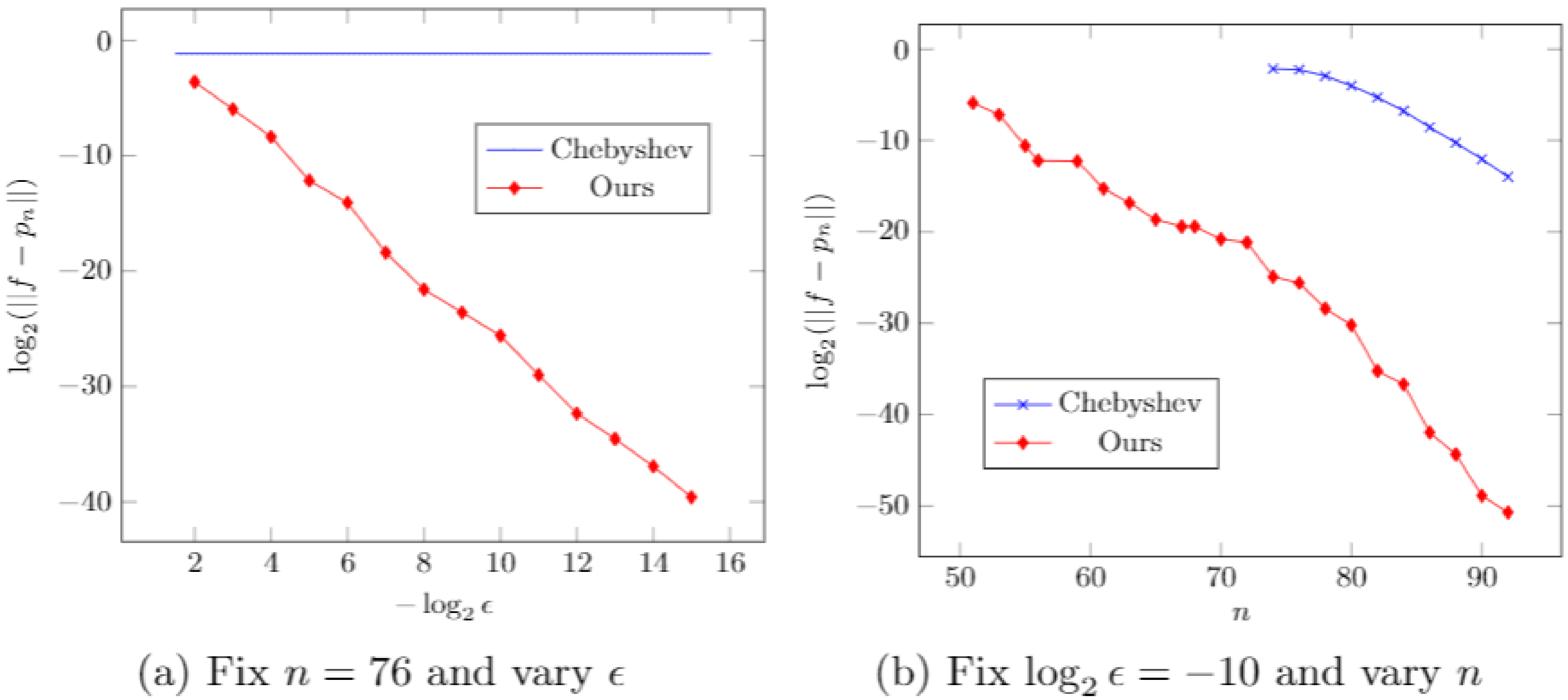


Fig. 2. Error bounds $\log_2(||f - p_n||)$ for our optimized interpolation $(K = 12)$.

Homomorphic Evaluation of a Polynomial

Navie approach

$$p(t) = \sum_{i=0}^{n} p_i t^i \longrightarrow p_i' s \text{ are unstable} \longrightarrow$$

1. Yield a lot of numerical errors
2. Make homomorphic evaluation difficult

Our method

$$T_0(t) = 1, T_1(t) = t,$$
$$T_i(t) = 2tT_{i-1}(t) - T_{i-2}(t) \text{ for } i \geq 2 \longrightarrow |T_i(t)| \leq 1 \text{ for all } |t| \leq 1$$

$$\tilde{T}_i(t) = T_i(\tfrac{t}{K}) \longrightarrow |\tilde{T}_i(t)| \leq 1 \text{ for all } |t| \leq K$$

$$p_n(t) = \sum_{i=0}^{n} c_i \cdot \tilde{T}_i(t) \Rightarrow$$

Compute with the Baby-step Giant-step algorithm $\Rightarrow$ $2\sqrt{2n} + \frac{1}{2}\log_2 n + O(1)$ non-scalar multiplications

Hybrid Method

Compute $\cos\left(2\pi \frac{t}{2^r}\right)$ and use double angle formula

Degree	Depth	# of scaling					
		1		2		3	
		Degree	Depth	Degree	Depth	Degree	Depth
76	7	49	6+1	**31**	**5+2**	24	5+3
86	7	**57**	**6+1**	40	6+2	28	5+3
96	7	65	7+1	45	6+2	34	6+3
106	7	72	7+1	51	6+2	38	6+3
116	7	80	7+1	57	6+2	43	6+3
126	7	88	7+1	63	6+2	49	6+3
136	8	**94**	**7+1**	70	7+2	55	6+3

Table 3. Minimum degree of an approximate polynomials to ensure the same level of error bound for each number of scaling and corresponding depth consumption. ($K = 12$ and $\log_2 \epsilon = -10$)

Comparison with the Previous Work [CCS19]

Method	Degree	# of Scaling	Degree (After scaling)	Non-scalar Multiplication	Depth
Ours	74	0	74	24 (Alg 2)	7
		1	49	16+1 (Alg 2)	6+1
		2	**30**	**11+2 (Alg 2)**	**5+2**
[5]	119	-	-	20 (PS alg)	7

Table 4. Comparison between our method and the previous work. [5] ($K = 12$ and $\log_2 \epsilon = -10$)

[CCS19] : Chen, H., Ilaria Chillotti and Yongsoo Song, Improved Bootstrapping for Approximate Homomorphic Encryption, *Eurocrypt,* 2019.

Experimental Results

Performance of Basic Operations

	$\log q_i$	L	dnum	Enc	Dec	Mult	Rescale	
$N = 2^{16}$	45	23	1	103 ms	5 ms	773 ms	60 ms	HEAAN-RNS
			4			**436 ms**		
			6			487 ms		
			12			660 ms		
			24			958 ms		SEAL v3.3

Table 5. Performance of Our Full-RNS variant of HEAAN with 2^{15} slots

Bootstrapping Performance

	ns	Boot Time	Precision	After Level	Amortized Time
Param 1	2^0	6.8 s	15.5	5	7.1 s
	2^1	7.0 s	16.8	3	3.5 s
	2^2	7.5 s	15.0	3	1.87 s
Param 2	2^5	28 s	18.5	9	0.87 s
	2^{10}	37.6 s	15.3	7	0.036 s
	2^{14}	$\boxed{52.8\text{ s}}$	10.8	7	0.0032 s

Table 7. Performance of the bootstrapping in our scheme

ns : the number of slots

Precision $:= -\log_2 e$, where e is average noise generated

Amortized Time : bootstrapping time per each slot

vs

[CCS 19] : 158s

[HHC 19] : 127s

[CCS19] : Chen, H., Ilaria Chillotti and Yongsoo Song, Improved Bootstrapping for Approximate Homomorphic Encryption, *Eurocrypt*, 2019.
[HHC 19] : Han, K.,Seungwan Hong and Jung Hee Cheon, Improved Homomorphic Discrete Fourier Transforms and FHE bootstrapping, *IEEE Access*, 2019.

RSAConference2020

Appendix

Key Generation in Our Scheme

KSGen$(s_1, s_2, \mathtt{dnum})$. For given secret polynomials $s_1, s_2 \in R$, sample $(a'^{(0)}, \ldots, a'^{(k+L)}) \leftarrow U\left(\prod_{i=0}^{k-1} R_{p_i} \times \prod_{j=0}^{L} R_{q_j}\right)$ and sample an error $e' \leftarrow \chi_{\mathsf{err}}$. Output the switching keys $\boxed{\{\mathsf{swk}_j\}_{0 \le j < \mathtt{dnum}}}$ as

$$\left(\mathsf{swk}_j^{(0)} = (b'^{(0)}_j, a'^{(0)}_j), \ldots, \mathsf{swk}_j^{(k+L)} = (b'^{(k+L)}_j, a'^{(k+L)}_j)\right) \in \prod_{i=0}^{k-1} R_{p_i}^2 \times \prod_{i=0}^{L} R_{q_i}^2$$

where $b'^{(i)}_j \leftarrow -a'^{(i)}_j \cdot s_2 + e' \pmod{p_i}$ for $0 \le i < k$ and $b'^{(k+i)}_j \leftarrow -a'^{(k+i)}_j \cdot s_2 + [P]_{q_i} \cdot \boxed{[\hat{Q}_j]_{q_i}} \cdot s_1 + e' \pmod{q_i}$ for $0 \le i \le L$.

KeyGen. First, sample $s \leftarrow \chi_{\mathsf{key}}$ and set the secret key as $\mathsf{sk} \leftarrow (1, s)$ and the evaluation key as $\boxed{\{\mathsf{evk}_i\}_{0 \le j < \mathtt{dnum}}} \leftarrow$ KSGen(s^2, s).

Multiplication in Our Scheme

$\underline{\mathtt{Mult}_{\mathsf{evk}}(\mathsf{ct}, \mathsf{ct}')}$. Given two ciphertexts $\mathsf{ct} = \left(\mathsf{ct}^{(j)} = (c_0^{(j)}, c_1^{(j)})\right)_{0 \le j \le \ell}$ and $\mathsf{ct}' = \left(\mathsf{ct}'^{(j)} = (c_0'^{(j)}, c_1'^{(j)})\right)_{0 \le j \le \ell}$, perform the following procedures and return the ciphertext $\mathsf{ct}_{\mathsf{mult}} \in \prod_{j=0}^{\ell} R_{q_j}^2$.

1. For $0 \le j \le \ell$, compute

$$
\begin{aligned}
d_0^{(j)} &\leftarrow c_0^{(j)} c_0'^{(j)} \pmod{q_j}, \\
d_1^{(j)} &\leftarrow c_0^{(j)} c_1'^{(j)} + c_1^{(j)} c_0'^{(j)} \pmod{q_j}, \\
d_2^{(j)} &\leftarrow c_1^{(j)} c_1'^{(j)} \pmod{q_j}.
\end{aligned}
$$

Multiplication in Our Scheme

2. RNS-Decompose:

   2-1. Zero-padding and Split: Let $\beta = \lceil (\ell+1)/\alpha \rceil$,

$$d'^{(i)}_{2,j} = \begin{cases} d_2^{(j\alpha+i)} \cdot [Q']_{q_{j\alpha+i}} & \text{if } j\alpha + i \leq \ell \\ 0 & \text{otherwise} \end{cases}$$

   for $0 \leq i < \alpha$, $0 \leq j < \beta$ and $Q' = \prod_{i=\ell+1}^{L} q_i$.

   2-2. RNS-Decompose:

$$d'^{(i)}_{2,j} \leftarrow d'^{(i)}_{2,j} \cdot [\hat{Q}_j^{-1}]_{q_{j\alpha+i}}$$

   for $0 \leq i < \alpha$ and $0 \leq j < \beta$ with $j\alpha + i \leq \ell$.

3. Modulus-Raise: compute $\tilde{d}_{2,j} = \mathtt{ModUp}_{\mathcal{C}_j \to \mathcal{D}_\beta}(d'_{2,j})$.

Multiplication in Our Scheme

4. Inner Product: compute

$$\tilde{\mathsf{ct}} = (\tilde{\mathsf{ct}}^{(0)} = (\tilde{c}_0^{(0)}, \tilde{c}_1^{(0)}), \ldots, \tilde{\mathsf{ct}}^{(k+\ell)} = (\tilde{c}_0^{(k+\ell)}, \tilde{c}_1^{(k+\ell)})) \in \prod_{i=0}^{k-1} R_{p_i}^2 \times \prod_{j=0}^{\ell} R_{q_j}^2,$$

where $\tilde{\mathsf{ct}}^{(i)} = \sum_{j=0}^{\beta-1} \tilde{d}_{2,j}^{(i)} \cdot \mathsf{evk}_j^{(i)} \pmod{p_i}$ for $0 \le i < k$ and $\tilde{\mathsf{ct}}^{(k+i)} = \sum_{j=0}^{\beta-1} \tilde{d}_{2,j}^{(k+i)} \cdot \mathsf{evk}_j^{(k+i)} \pmod{q_i}$ for $0 \le i < \alpha\beta$.

5. Modulus-Down: compute

$$\left(\hat{c}_0^{(0)}, \ldots, \hat{c}_0^{(\ell)}\right) \leftarrow \mathtt{ModDown}_{\mathcal{D}_\beta \to \mathcal{C}_\ell}\left(\tilde{c}_0^{(0)}, \ldots, \tilde{c}_0^{(k+\alpha\beta)}\right),$$
$$\left(\hat{c}_1^{(0)}, \ldots, \hat{c}_1^{(\ell)}\right) \leftarrow \mathtt{ModDown}_{\mathcal{D}_\beta \to \mathcal{C}_\ell}\left(\tilde{c}_1^{(0)}, \ldots, \tilde{c}_1^{(k+\alpha\beta)}\right).$$

6. Output the ciphertext $\mathsf{ct}_{\mathsf{mult}} = (\mathsf{ct}_{\mathsf{mult}}^{(j)})_{0 \le j \le \ell}$, where $\mathsf{ct}_{\mathsf{mult}}^{(j)} \leftarrow (\hat{c}_0^{(j)} + d_0^{(j)}, \hat{c}_1^{(j)} + d_1^{(j)}) \pmod{q_j}$ for $0 \le j \le \ell$.

Baby-step Giant-step Algorithm

Algorithm 1 Baby-step Giant-step algorithm

1: **Input** : A polynomial of degree n, $p = \sum_{i=0}^{n} c_i T_i$.
2: Let m be the smallest integer satisfying $2^m > n$ and $l \approx m/2$.
3: Evaluate $T_2(t), T_3(t) \cdots, T_{2^l}(t)$ inductively.
4: Evaluate $T_{2^{l+1}}(t) \cdots, T_{2^{m-1}}(t)$ using the equation $T_{2i}(t) = 2T_i(t)^2 - 1$.
5: Find polynomials q and r of degree $< 2^{m-1}$ which satisfy $p = q \cdot T_{2^{m-1}} + r$ in forms of a linear combination of Chebyshev basis.
6: Evaluate $q(t)$ and $r(t)$ recursively. (Repeat 5 with replacing p with q and r until the degree of the quotient and the remainder become smaller than 2^l)
7: Evaluate $p(t)$ with $q(t)$, $r(t)$ and $T_{2^{m-1}}(t)$.
8: **Output** : $p(t)$

➡ $2\sqrt{2n} + \frac{1}{2}\log_2 n + O(1)$ non-scalar multiplications

➡ $2^l + 2^{m-l} + m - l - 3$ non-scalar multiplications,
where m is the smallest integer satisfying $2^m > n$ and $l \approx m/2$

Parameters Sets for Bootstrapping Performance Exp.

	L	dnum	N	$\log Q$	$\log Q + \log P$	Security
Param 1	19	10	32768	810	910	110.4
Param 2	27	7	65536	1270	1452	127.2

Table 6. Parameter Sets

Param 1 : $\log q \approx 40, \log q_0 \approx 50$

Param 2 : $\log q \approx 45, \log q_0 \approx 55$