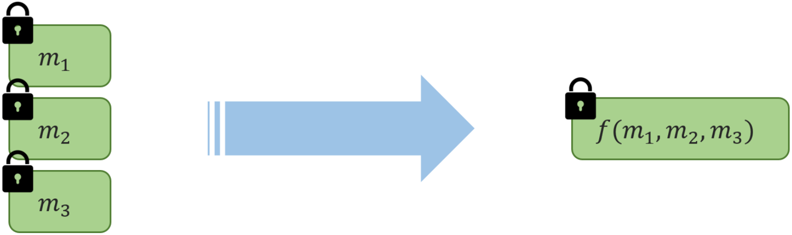
Faster TFHE Bootstrapping with Block Binary Keys

Changmin Lee[1], **Seonhong Min**[2], Jinyeong Seo[2], Yongsoo Song[2]

[1]Korea Institute for Advanced Study, Seoul
[2]Seoul National University, Seoul

Fully Homomorphic Encryption



- **Fully Homomorphic Encryption (FHE)** supports arbitrary function evaluation on encrypted data.
- **Various Applications**: privacy preserving machine learning, private information retrieval, private set intersection ...

Learning with Errors

- The most efficient FHEs to date are built on **Learning with Errors (LWE)** problem and its ring-variant **Ring-LWE (RLWE)**.

- **LWE**: $(\mathbf{a}, b) \approx_c \mathcal{U}(\mathbb{Z}_q^{n+1})$
  - $\mathbf{a} \leftarrow \mathcal{U}(\mathbb{Z}_q^n)$, $\mathbf{s} \in \mathbb{Z}^n$, $e \leftarrow$ small dist' over $\mathbb{Z}$
  - $b = -\langle \mathbf{a}, \mathbf{s} \rangle + e \pmod q$

- **RLWE**: $(a, b) \approx_c \mathcal{U}(R_q^2)$
  - Variant of LWE over $R_q = R/qR$ where $R = \mathbb{Z}[X]/(X^N + 1)$
  - $a \leftarrow \mathcal{U}(R_q)$, $s \in R$, $e \leftarrow$ small dist' over R
  - $b = -a \cdot s + e \pmod q$

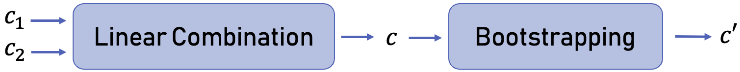
- FHE schemes based on LWE/RLWE
  - BGV / BFV / CKKS
  - **TFHE** / FHEW

TFHE description

- FHE scheme that supports bits operations (NAND, AND, OR...).
- **Secret Key**:
  - LWE secret $\mathbf{s} = (s_1, \ldots, s_n)$
  - RLWE secret $t = \sum_{i=1}^{N} t_i X^{i-1}$
  - Vectorized secret $\mathbf{t} = (t_1, \ldots, t_N)$
  - All keys are sampled from **binary distribution**
- **Encoding**: $m \in \{-1, 1\} \mapsto \mu = \frac{q}{8} m \in \mathbb{Z}_q$
- **Decoding**: $\begin{cases} 1 & \text{if } \mu > 0 \\ -1 & \text{otherwise} \end{cases}$
- **Encryption**: $c = (b, \mathbf{a}) \in \mathbb{Z}_q^{n+1}$ for $\mathbf{a} \leftarrow \mathcal{U}(\mathbb{Z}_q^n)$, $e \leftarrow$ small dist., $b = -\langle \mathbf{a}, \mathbf{s} \rangle + \mu + e$.
- **Decryption**: $b + \langle \mathbf{a}, \mathbf{s} \rangle = \mu + e$

Homomorphic Gate Evaluation

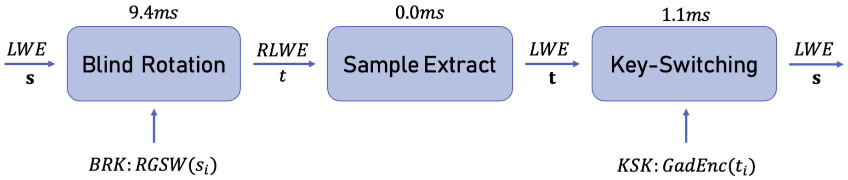
- Each bit operation consists of the following pipeline:



- **Linear Combination** : The linear combination corresponding to a Boolean gate is evaluated.
  - ex) NAND : $c = (\frac{q}{8}, \mathbf{0}) - c_1 - c_2$
  - output ciphertext contains a **large noise** e.

- **Bootstrapping** : Reduces the size of noise for further evaluation.
  - ex) $\|e\| < \frac{q}{8} \to \|e'\| < \frac{q}{16}$

TFHE Bootstrapping



- **Blind Rotation** : Homomorphically computes the decryption circuit on the exponent of X *i.e.*, $X^{b+\langle \mathbf{a},\mathbf{s}\rangle}$.
  - Need Blind Rotation Key : RGSW encryptions of s_i $(1 \leq i \leq n)$
- **Sample Extract** : Extract the constant term of the plaintext from the resulting RLWE ciphertext.
- **Key-Switching** : Switch the dimension of the LWE ciphertext.
  - Need Key-Switching Key : Gadget encryptions of t_i $(1 \leq i \leq N)$

Our Contribution

Motivation : Most FHE schemes (BGV/FV/CKKS) make an additional assumption on key structure to obtain better efficiency.

- BGV/FV : Small noise growth in homomorphic multiplication.
- BGV/CKKS : Small depth for bootstrapping.

Our Result : We adapt similar approach to accelerate TFHE bootstrapping.

1. **Faster Blind Rotation**
   - Sample LWE key from **block binary key distribution**
   - Reduce the number of FFT operations
2. **Compact Key-Switching**
   - Re-use the LWE key as a part of RLWE key
   - Improve both time and space complexity

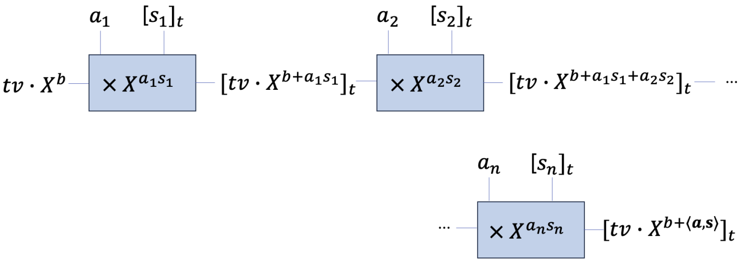
Blind Rotation

Functionality

- Homomorphic evaluation of $tv \cdot X^{b+\sum_{i=1}^{n} a_i s_i} = tv \cdot X^{\frac{q}{8}m+e} \in R_q$.
  - $tv = -\frac{q}{8}(1 + X + \cdots + X^{N-1}) \in R_q$.
  - Constant term of $tv \cdot X^{\frac{q}{8}m+e} = \frac{q}{8}m$.

- Homomorphically multiply monomials $X^{a_i s_i}$ to $tv \cdot X^b$ iteratively.
- We need **n external products** total.

Previous Blind Rotation



- $X^{a_i s_i} = \begin{cases} X^{a_i} & (s_i = 1) \\ 1 & (s_i = 0) \end{cases} = 1 + (X^{a_i} - 1)s_i$
  - Using this **key formula**, we have $[X^{a_i s_i}]_t = 1 + (X^{a_i} - 1)[s_i]_t$
  - We iteratively multiply one monomial $X^{a_i s_i}$ for **n** times.

Observation

- Can we multiply 2 monomials simultaneously?

$$
\begin{aligned}
&X^{a_1 s_1 + a_2 s_2} \\
&\quad = (1 + (X^{a_1} - 1)s_1)(1 + (X^{a_2} - 1)s_2) \\
&\quad = 1 + (X^{a_1} - 1)s_1 + (X^{a_2} - 1)s_2 + (X^{a_1} - 1)(X^{a_2} - 1)s_1 s_2
\end{aligned}
$$

- With this formula, the number of homomorphic mult reduces by **half**.
  - Requires RGSW encryption of $s_1 s_2$
  - $+$ the number of linear evaluation grows.
- What if we can ignore the case where $s_1 = s_2 = 1$?
  - No additional blind rotation keys are required.
  - The number of linear evaluation remains same.
- **Generalization**: How about ℓ monomials?
  $\rightarrow$ Possible. If $\mathbf{s}$ is sampled from **Block Binary Key Distribution**...

Block Binary Keys

Definition (Block Binary Key)

- $n = k\ell$ for two positive integers $k, \ell > 0$
- $\mathbf{s} = (B_1, \ldots, B_k) \in \{0,1\}^n$
- $B_i \leftarrow \mathcal{U}((1,0,\ldots,0),\ldots,(0,0,\ldots,1),(0,\ldots,0))$
- **At most one 1 in each block**

1	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	1	0

Figure: Block Binary Key with $\ell = 3$ and $k = 6$

Block Binary Keys

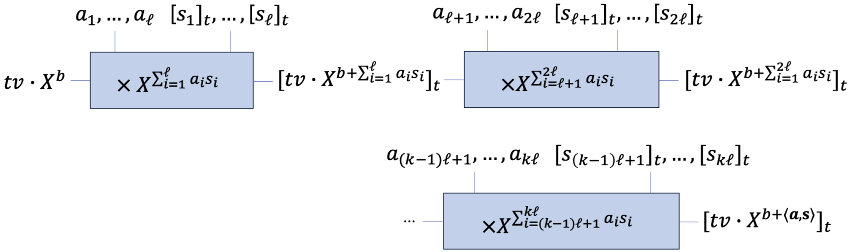
- $$X^{a_1 s_1} = \begin{cases} X^{a_1} & (s_1 = 1) \\ 1 & (s_1 = 0) \end{cases}$$
  $$= 1 + (X^{a_1} - 1)s_1$$

  $\rightarrow$ Multiply 1 monomial with 1 mult and 1 add.

- $$X^{\sum_{i=1}^{\ell} a_i s_i} = \begin{cases} X^{a_1} & (s_1 = 1, s_2 = 0, \ldots, s_\ell = 0) \\ \vdots & \\ X^{a_\ell} & (s_1 = 0, s_2 = 0, \ldots, s_\ell = 1) \\ 1 & (s_1 = 0, s_2 = 0, \ldots, s_\ell = 0) \end{cases}$$
  $$= 1 + \sum_{i=1}^{\ell} (X^{a_i} - 1)s_i$$

  $\rightarrow$ Multiply ℓ monomials with 1 mult and ℓ add.

Our Blind Rotation



- Iteratively multiplies ℓ monomials with one homomorphic multiplication.
- Only **k** external products are required!!
- However, not direct ℓ-times speedup due to other operations.

Algorithm

Algorithm 1 New Blind Rotation

1: **Input:** The blind rotation key $\mathtt{BK}$ and a TLWE ciphertext $\mathbf{c} = (b, \mathbf{a}) \in \mathbb{T}^{n+1}$

2: **Output:** A TRLWE ciphertext $\mathtt{ACC} \in \mathbb{T}_N[X]^2$

3: $\mathtt{tv} \leftarrow -\frac{1}{8} \cdot (1 + X + \cdots + X^{N-1}) \in \mathbb{T}_N[X]$

4: Let $\overline{b} = \lfloor 2Nb \rceil$ and $\overline{a}_i = \lfloor 2Na_i \rceil$ for $0 \leq i < n$

5: $\mathtt{ACC} \leftarrow (X^{\overline{b}} \cdot \mathtt{tv}, 0) \in \mathbb{T}_N[X]^2$

6: **for** $0 \leq j < k$ **do**

7: $\quad \mathtt{ACC} \leftarrow \mathtt{ACC} + \mathtt{ACC} \boxdot \left[\sum_{i \in I_j} (X^{\overline{a}_i} - 1) \cdot \mathtt{BK}_i \right]$

8: **end for**

Optimization (Hoisting)

- This algorithm requires more Floating point operations than the original blind rotation algorithm.
- Instead, we re-use the gadget decomposition of ACC for each external products. *i.e.*, $h(\texttt{ACC})$
-
  - **Previous**: $\texttt{ACC} \leftarrow \texttt{ACC} + \left\langle h(\texttt{ACC}), \sum_{i \in I_j} (X^{\bar{a}_i} - 1) \cdot \texttt{BK}_i \right\rangle$
  - **Modified**: $\texttt{ACC} \leftarrow \texttt{ACC} + \sum_{i \in I_j} (X^{\bar{a}_i} - 1) \cdot \langle h(\texttt{ACC}), \texttt{BK}_i \rangle$
- Then, the number of FFT operations is reduced with the same number of Floating point operations.

Security of Block Binary Keys

- **Asymptotic Security** : If the entropy of key distribution is sufficiently large, LWE is secure (Goldwasser et al).
  - Entropy of block binary keys : $(\ell + 1)^k$
- **Concrete Security** : We conducted cryptanalysis considering the best-known lattice attacks.
  - Dual attack
  - Meet-in-the-Middle
  - Tailor-made

Dual Attack

- **Dual Attack**
  - Dual Attack is effective for sparse secret.
  - Run lattice-estimator with respect to (expected) Hamming weight $n/(\ell+1)$ and LWE dimension n.
- **Modified Dual Attack**
  - With one guessing, one can reduce ℓ dimension at once.
  - Therefore, one can reduce t blocks by guessing and then exploit dual attack.
  - Then, the cost is $O((\ell+1)^t \cdot \mathcal{T})$ where $\mathcal{T}$ is the cost of dual attack on LWE of dimension $n - t\ell$ under secret with (expected) Hamming weight $(n-t\ell)/(\ell+1)$.

MitM attack

- MitM algorithm
  - Given secret $\mathbf{s}$, split the secret vector into $\mathbf{s} = \mathbf{s}_0 + \mathbf{s}_1$.
  - For an LWE instance $(b, \mathbf{a})$, $b + \langle \mathbf{s}_0, \mathbf{a} \rangle \approx - \langle \mathbf{s}_1, \mathbf{a} \rangle$ since $b + \langle \mathbf{a}, \mathbf{s} \rangle$ is small.
  - Therefore, we can find the collision between two sets in time $\mathcal{S}^{0.5}$:

$$\mathcal{R}_0 = \{ b + \langle \mathbf{x}_0, \mathbf{a} \rangle \mid \|\mathbf{x}_0\|_1 = \|\mathbf{s}\|_1/2 \}$$
$$\mathcal{R}_1 = \{ - \langle \mathbf{x}_1, \mathbf{a} \rangle \mid \|\mathbf{x}_1\|_1 = \|\mathbf{s}\|_1/2 \}$$

- May et al. (2021)
  - Inductively perform MitM algorithm to $\mathbf{s}_0, \mathbf{s}_1$.
  - Overall cost requires $\geq \mathcal{S}^{0.28}$ time complexity.
- Since $\mathcal{S} = (\ell + 1)^k$, the cost is $2^{O(0.28k \log(\ell+1))}$.
- In other words, it achieves $0.28k \log(\ell + 1)$-bit security.

Key-Switching

Functionality

- Switch the secret key of LWE ciphertext from $\mathbf{t}$ to $\mathbf{s}$.

- For LWE ciphertext $\mathbf{c} = (b, a_1, \ldots, a_N)$ encrypted under $\mathbf{t}$, we compute $\mathbf{c}' = (b, 0, \ldots, 0) + \sum_{i=1}^{N} a_i \odot \mathrm{Enc}_{\mathbf{s}}(t_i)$.
  - $\mathrm{Enc}_{\mathbf{s}}(t_i)$: Gadget encryptions of t_i under $\mathbf{s}$ $(1 \leq i \leq N)$.
  - $Dec_{\mathbf{s}}(\mathbf{c}') \approx b + \sum_{i=1}^{N} a_i t_i = Dec_{\mathbf{t}}(\mathbf{c})$.
- **Complexity**
  - Time : **N** homomorphic scalar multiplications.
  - Space: **N** key-switching keys

Compact Key-Switching

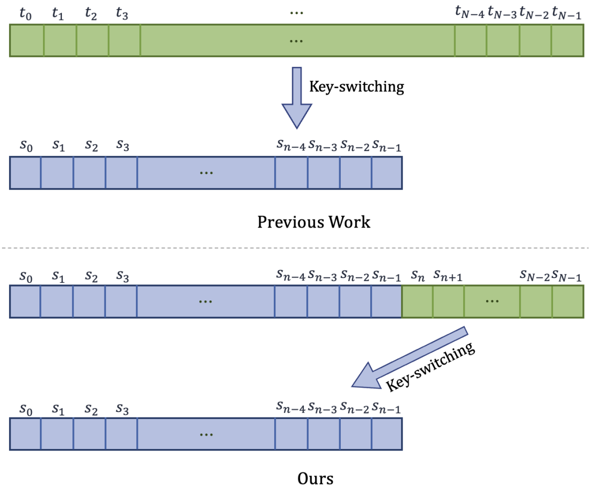
- If $t_i = s_i$ $(1 \le i \le n)$, we can replace $\mathbf{c}'$ by

$$(b, a_1, \dots, a_n) + \sum_{i=n+1}^{N} a_i \odot \mathrm{Enc}_{\mathbf{s}}(t_i)$$

  - $Dec_{\mathbf{s}}(\mathbf{c}') \approx b + \sum_{i=1}^{n} a_i s_i + \sum_{i=n+1}^{N} a_i t_i = b + \sum_{i=1}^{N} a_i t_i = Dec_{\mathbf{t}}(\mathbf{c})$.

- **Complexity**
  - Time : $\mathbf{N} - \mathbf{n}$ scalar multiplications
  - Space : $\mathbf{N} - \mathbf{n}$ key-switching keys

Compact Key-Switching



Security Analysis of Compact Key-Switching

- **Dual Attack :** Run the lattice estimator with LWE dimension N, (expected) Hamming weight $n/(\ell + 1) + (N - n)/2$.

- **MitM Attack :** The security relies on the LWE security.

Parameter Selection

- We set the parameters with respect to the Dual and MitM attack.
- Given ℓ, we can set $k = \lceil 457.143/\log(\ell+1) \rceil$.

$n = k\ell$	N	ℓ	Dual	MitM
630	1024	2	130.7	139.7
687	1024	3	130.7	128.2
788	1024	4	129.9	128.0
885	1024	5	128.9	128.1
978	1024	6	128.0	128.1

Implementation & Result

	ℓ	n	Bootstrapping	Key Size
TFHE	·	630	10.53 *ms*	109 MB
Ours	2	630	7.05 *ms*	60 MB
	3	687	6.49 *ms*	
	4	788	6.70 *ms*	
	5	885	6.82 *ms*	56 MB
	6	978	7.12 *ms*	52 MB

Table: 128-bit Security level

- Implemented based on **the TFHE library**.
- We achieve **1.5-1.6x** SPEEDUP!
- Key size is reduced by **1.8x**!

Further Applications

- This technique can be applied to many TFHE-like cryptosystems.
- It works as long as the **algebraic structure** remains the same.
  - **O** PBS, WoP-PBS, Chimera...
  - **O** MK-TFHE
    - The secret key for MK ciphertexts is the concatenated vector of each secret key.
  - **O** AP/FHEW
    - Secret key sampled from block n-ary distribution.
    - Originally, keys were given by RGSW encryptions of $X^{jB^k \cdot s_i}$ (X^{s_i} in LMKC+22).
    - Instead, provide RGSW encryptions of 0 if s_i is zero.
  - **Δ** MP-TFHE (n-out-of-n Threshold TFHE)
    - The secret key for MP ciphertexts is the sum of each secret key.
    - Can be applied to a naïve solution (AKÖ23).
    - Cannot be applied to the state-of-the art schemes (LMKC+22, PR23).

Multi-Key TFHE

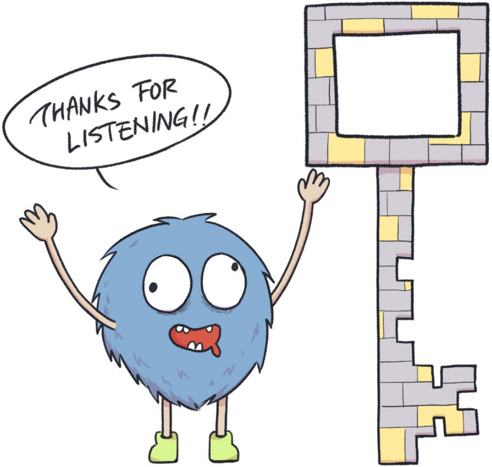
#Parties	2	4	8	16	32
KMS	0.25 s	0.87 s	2.24 s	5.62 s	14.04 s
Block	0.14 s	0.49 s	1.17 s	3.30 s	7.68 s

Table: 128-bit Security level

- We achieve **1.7-1.9x** SPEEDUP.
- The performance improvement is better than single-key scheme.
- The size of the key-switching key is also reduced.

Implementations

- Source code is available at **github.com/SNUCP/blockkey-tfhe**
- MK implementation (Julia) : **github.com/SNUCP/MKTFHE**
- PBS implementation (Go) : **github.com/sp301415/tfhe-go**


THANKS FOR LISTENING!!