Efficient FHEW Bootstrapping with Small Evaluation Keys, and Applications to Threshold Homomorphic Encryption

Yongwoo Lee[1,3] Daniele Micciancio[2] Andrey Kim[1]
Rakyong Choi[1] Maxim Deryabin[1] Jieun Eom[1] Donghoon Yoo[1,4]

[1]Samsung Advanced Institute of Technology

[2]University of California, San Diego

[3]Inha University

[4]Desilo

Eurocrypt 2023
Apr. 24 2023

Outline

Outline

FHEW-like Fully Homomorphic Encryption

- FHEW-like [DM15] schemes are the best-known bit-level HE
- Small parameter size
- Fastest bootstrapping ($\leq$ 100ms)
- Two competing approaches:
  - AP/FHEW/DM:
    all secret keys, large boot key [DM15, AP14]
  - GINX/TFHE/CGGI:
    limited secret key distribution, small boot key [GINX16, CGGI17]

FHEW-like Fully Homomorphic Encryption

- FHEW-like [DM15] schemes are the best-known bit-level HE
- Small parameter size
- Fastest bootstrapping (≤ 100ms)
- Two competing approaches:
  - AP/FHEW/DM:
    all secret keys, large boot key [DM15, AP14]
  - GINX/TFHE/CGGI:
    limited secret key distribution, small boot key [GINX16, CGGI17]

FHEW-like Fully Homomorphic Encryption

- FHEW-like [DM15] schemes are the best-known bit-level HE
- Small parameter size
- Fastest bootstrapping (≤ 100ms)
- Two competing approaches:
  - AP/FHEW/DM:
    all secret keys, large boot key [DM15, AP14]
  - GINX/TFHE/CGGI:
    limited secret key distribution, small boot key [GINX16, CGGI17]

Technical Contribution

- The third bootstrapping offering the best of both approaches

Method	Arbitrary secret	Boot key size
AP	◯	large
GINX	$\times(\triangle)$[1]	small
Proposed	◯	small

- Additional benefit: smaller noise growth
- Efficient FHEW-like threshold HE
- Source code available at OpenFHE[2]

[1] Modification for arbitrary distribution is proposed in [MP21, JP22], and another variant for ternary keys is in [KDE+21, BIP+22]

[2] https://github.com/openfheorg/openfhe-development/tree/278-new-lmkcdeys

Technical Contribution

- The third bootstrapping offering the best of both approaches

Method	Arbitrary secret	Boot key size
AP	◯	large
GINX	$\times(\triangle)$[1]	small
Proposed	◯	small

- Additional benefit: smaller noise growth
- Efficient FHEW-like threshold HE
- Source code available at OpenFHE[2]

[1] Modification for arbitrary distribution is proposed in [MP21, JP22], and another variant for ternary keys is in [KDE$^+$21, BIP$^+$22]

[2] https://github.com/openfheorg/openfhe-development/tree/278-new-lmkcdeys

Technical Contribution

- The third bootstrapping offering the best of both approaches

Method	Arbitrary secret	Boot key size
AP	◯	large
GINX	$\times(\triangle)$[1]	small
Proposed	◯	small

- Additional benefit: smaller noise growth
- Efficient FHEW-like threshold HE
- Source code available at OpenFHE[2]

[1] Modification for arbitrary distribution is proposed in [MP21, JP22], and another variant for ternary keys is in [KDE+21, BIP+22]

[2] https://github.com/openfheorg/openfhe-development/tree/278-new-lmkcdeys

Technical Contribution

- The third bootstrapping offering the best of both approaches

Method	Arbitrary secret	Boot key size
AP	◯	large
GINX	$\times(\triangle)$[1]	small
Proposed	◯	small

- Additional benefit: smaller noise growth
- Efficient FHEW-like threshold HE
- Source code available at OpenFHE[2]

[1] Modification for arbitrary distribution is proposed in [MP21, JP22], and another variant for ternary keys is in [KDE+21, BIP+22]

[2] https://github.com/openfheorg/openfhe-development/tree/278-new-lmkcdeys

FHEW Bootstrapping

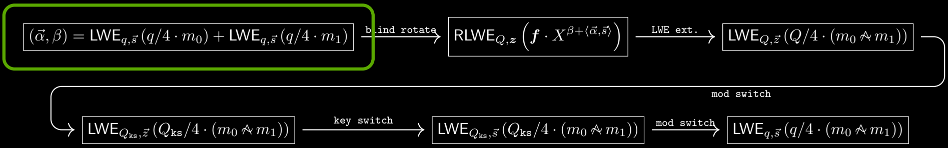


Figure: NAND gate bootstrapping procedure of FHEW scheme [DM15, MP21]

FHEW Bootstrapping

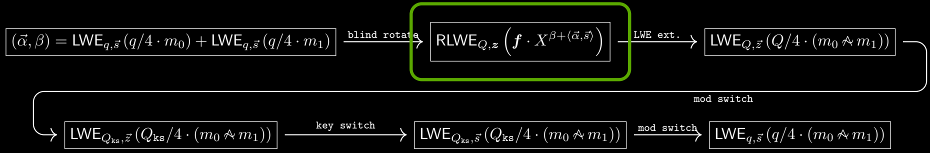


Figure: NAND gate bootstrapping procedure of FHEW scheme [DM15, MP21]

FHEW Bootstrapping

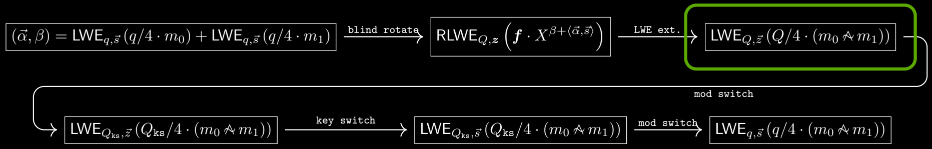


Figure: NAND gate bootstrapping procedure of FHEW scheme [DM15, MP21]

FHEW Bootstrapping

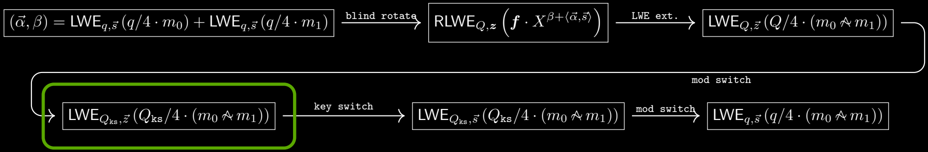


Figure: NAND gate bootstrapping procedure of FHEW scheme [DM15, MP21]

FHEW Bootstrapping

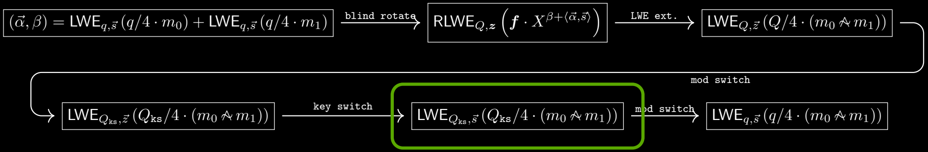


Figure: NAND gate bootstrapping procedure of FHEW scheme [DM15, MP21]

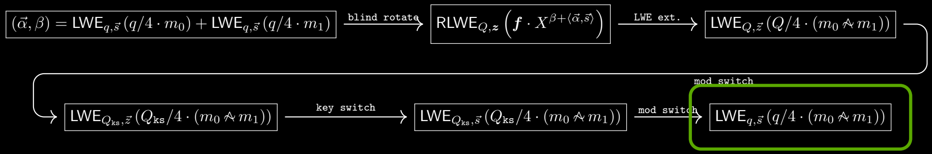


Figure: NAND gate bootstrapping procedure of FHEW scheme [DM15, MP21]

Blind Rotation

Definition (Blind Rotation)

A blind rotation is an algorithm that takes as input a ring element $\boldsymbol{f} \in \mathcal{R}_Q$, an $\mathsf{LWE}_{2N,\vec{s}}$ ciphertext $(\vec{\alpha}, \beta) \in \mathbb{Z}_{2N}^{n+1}$, and blind rotation keys $\mathtt{brk}_{\boldsymbol{z},\vec{s}}$ corresponding to secrets $\boldsymbol{z}$ and $\vec{s}$. It outputs an RLWE ciphertext of the form:

$$\mathsf{RLWE}_{Q,\boldsymbol{z}}\left(\boldsymbol{f} \cdot X^{\beta+\langle\vec{\alpha},\vec{s}\rangle}\right) \in \mathcal{R}_Q^2.$$

- A crucial component of bootstrapping for FHEW-like HE
- It enables decryption of LWE ciphertext in the exponent of the output
- The constant term of the output is f_{-u}, where $u = \beta + \langle\vec{\alpha}, \vec{s}\rangle$

Blind Rotation

- Bootstrapping for FHEW-like HE
- Machine Learning [LHH+21]
- Sign function [LMP22]
- Modular reduction for CKKS/BGV/BFV bootstrapping [KDE+21]

Blind Rotation

- Bootstrapping for FHEW-like HE
- Machine Learning [LHH+21]
- Sign function [LMP22]
- Modular reduction for CKKS/BGV/BFV bootstrapping [KDE+21]

Blind Rotation

- Bootstrapping for FHEW-like HE
- Machine Learning [LHH+21]
- Sign function [LMP22]
- Modular reduction for CKKS/BGV/BFV bootstrapping [KDE+21]

Blind Rotation

- Bootstrapping for FHEW-like HE
- Machine Learning [LHH+21]
- Sign function [LMP22]
- Modular reduction for CKKS/BGV/BFV bootstrapping [KDE+21]

Basic Building Block: RGSW Encryption of s_i

$\circledast$: **RLWE $\times$ RGSW $\rightarrow$ RLWE**

$$\mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}_1) \circledast \mathsf{RGSW}_{\boldsymbol{z}}(\boldsymbol{m}_2) = \mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}_1 \cdot \boldsymbol{m}_2 + \boldsymbol{e}_1 \cdot \boldsymbol{m}_2) \in \mathcal{R}_Q^2$$

- When $\boldsymbol{m}_2$ is small (e.g., monomial) noise is only additive
- Note: Multiplying monomial X^k == adding k in exponent
- RGSW encryptions of partial secret key as blind rotation keys

Using RGSW keys in prior arts

- AP: decompose $\alpha_i \Rightarrow$ many RGSW keys required
- GINX: decompose $s_i \Rightarrow$ distribution of s_i limited

Basic Building Block: RGSW Encryption of s_i

$\circledast$: **RLWE** $\times$ **RGSW** $\rightarrow$ **RLWE**

$$\mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}_1) \circledast \mathsf{RGSW}_{\boldsymbol{z}}(\boldsymbol{m}_2) = \mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}_1 \cdot \boldsymbol{m}_2 + \boldsymbol{e}_1 \cdot \boldsymbol{m}_2) \in \mathcal{R}_Q^2$$

- When $\boldsymbol{m}_2$ is small (e.g., monomial) noise is only additive
- Note: Multiplying monomial X^k == adding k in exponent
- RGSW encryptions of partial secret key as blind rotation keys

Using RGSW keys in prior arts

- AP: decompose $\alpha_i \Rightarrow$ many RGSW keys required
- GINX: decompose $s_i \Rightarrow$ distribution of s_i limited

Basic Building Block: RGSW Encryption of s_i

$\circledast$: **RLWE** $\times$ **RGSW** $\rightarrow$ **RLWE**

$$\mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}_1) \circledast \mathsf{RGSW}_{\boldsymbol{z}}(\boldsymbol{m}_2) = \mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}_1 \cdot \boldsymbol{m}_2 + \boldsymbol{e}_1 \cdot \boldsymbol{m}_2) \in \mathcal{R}_Q^2$$

- When $\boldsymbol{m}_2$ is small (e.g., monomial) noise is only additive
- Note: Multiplying monomial X^k $==$ adding k in exponent
- RGSW encryptions of partial secret key as blind rotation keys

Using RGSW keys in prior arts

- AP: decompose $\alpha_i \Rightarrow$ many RGSW keys required
- GINX: decompose $s_i \Rightarrow$ distribution of s_i limited

Basic Building Block: RGSW Encryption of s_i

$\circledast$: **RLWE** $\times$ **RGSW** $\rightarrow$ **RLWE**

$$\mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}_1) \circledast \mathsf{RGSW}_{\boldsymbol{z}}(\boldsymbol{m}_2) = \mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}_1 \cdot \boldsymbol{m}_2 + \boldsymbol{e}_1 \cdot \boldsymbol{m}_2) \in \mathcal{R}_Q^2$$

- When $\boldsymbol{m}_2$ is small (e.g., monomial) noise is only additive
- Note: Multiplying monomial X^k == adding k in exponent
- RGSW encryptions of partial secret key as blind rotation keys

Using RGSW keys in prior arts

- AP: decompose $\alpha_i \Rightarrow$ many RGSW keys required
- GINX: decompose $s_i \Rightarrow$ distribution of s_i limited

Basic Building Block: RGSW Encryption of s_i

$\circledast$: **RLWE** $\times$ **RGSW** $\rightarrow$ **RLWE**

$$\mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}_1) \circledast \mathsf{RGSW}_{\boldsymbol{z}}(\boldsymbol{m}_2) = \mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}_1 \cdot \boldsymbol{m}_2 + \boldsymbol{e}_1 \cdot \boldsymbol{m}_2) \in \mathcal{R}_Q^2$$

- When $\boldsymbol{m}_2$ is small (e.g., monomial) noise is only additive
- Note: Multiplying monomial X^k == adding k in exponent
- RGSW encryptions of partial secret key as blind rotation keys

Using RGSW keys in prior arts

- AP: decompose $\alpha_i \Rightarrow$ many RGSW keys required
- GINX: decompose $s_i \Rightarrow$ distribution of s_i limited

Basic Building Block: RGSW Encryption of s_i

$\circledast$: **RLWE** $\times$ **RGSW** $\rightarrow$ **RLWE**

$$\mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}_1) \circledast \mathsf{RGSW}_{\boldsymbol{z}}(\boldsymbol{m}_2) = \mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}_1 \cdot \boldsymbol{m}_2 + \boldsymbol{e}_1 \cdot \boldsymbol{m}_2) \in \mathcal{R}_Q^2$$

- When $\boldsymbol{m}_2$ is small (e.g., monomial) noise is only additive
- Note: Multiplying monomial X^k == adding k in exponent
- RGSW encryptions of partial secret key as blind rotation keys

Using RGSW keys in prior arts

- AP: decompose $\alpha_i \Rightarrow$ many RGSW keys required
- GINX: decompose $s_i \Rightarrow$ distribution of s_i limited

Basic Building Block: RGSW Encryption of s_i

$\circledast$: **RLWE** $\times$ **RGSW** $\rightarrow$ **RLWE**

$$\mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}_1) \circledast \mathsf{RGSW}_{\boldsymbol{z}}(\boldsymbol{m}_2) = \mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}_1 \cdot \boldsymbol{m}_2 + \boldsymbol{e}_1 \cdot \boldsymbol{m}_2) \in \mathcal{R}_Q^2$$

- When $\boldsymbol{m}_2$ is small (e.g., monomial) noise is only additive
- Note: Multiplying monomial X^k $==$ adding k in exponent
- RGSW encryptions of partial secret key as blind rotation keys

Using RGSW keys in prior arts

- AP: decompose $\alpha_i \Rightarrow$ many RGSW keys required
- GINX: decompose $s_i \Rightarrow$ distribution of s_i limited

Outline

Another Building Block: Ring Automorphisms

- We use ring automorphism as another building block[3]
- Constant multiplication in the exponent
- $\texttt{EvalAuto}_t\left(\mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}), \texttt{ak}_t\right)$:

$$\mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}(X)) = (\boldsymbol{a}(X), \boldsymbol{b}(X)) \xrightarrow{\psi_t} \mathsf{RLWE}_{\boldsymbol{z}(X^t)}(\boldsymbol{m}(X^t)) = (\boldsymbol{a}(X^t), \boldsymbol{b}(X^t))$$

$$\texttt{KS}_{\boldsymbol{z}(X^t) \to \boldsymbol{z}(X)}\left(\mathsf{RLWE}_{\boldsymbol{z}(X^t)}(\boldsymbol{m}(X^t))\right) = \mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}(X^t))$$

[3] Bonnoron et al. first used automorphisms to reduce the key size of a variant of the FHEW cryptosystem [BDF18].

Another Building Block: Ring Automorphisms

- We use ring automorphism as another building block[3]
- Constant multiplication in the exponent
- $\texttt{EvalAuto}_t\left(\mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}), \texttt{ak}_t\right)$:

$$\mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}(X)) = (\boldsymbol{a}(X), \boldsymbol{b}(X)) \xrightarrow{\psi_t} \mathsf{RLWE}_{\boldsymbol{z}(X^t)}(\boldsymbol{m}(X^t)) = (\boldsymbol{a}(X^t), \boldsymbol{b}(X^t))$$

$$\texttt{KS}_{\boldsymbol{z}(X^t)\to\boldsymbol{z}(X)}\left(\mathsf{RLWE}_{\boldsymbol{z}(X^t)}(\boldsymbol{m}(X^t))\right) = \mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}(X^t))$$

[3] Bonnoron et al. first used automorphisms to reduce the key size of a variant of the FHEW cryptosystem [BDF18].

Another Building Block: Ring Automorphisms

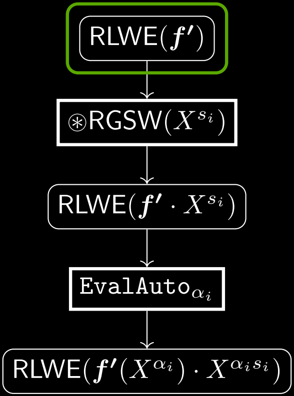
- We use ring automorphism as another building block[3]
- Constant multiplication in the exponent
- $\texttt{EvalAuto}_t\,(\mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}), \texttt{ak}_t)$:

$$\mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}(X)) = (\boldsymbol{a}(X), \boldsymbol{b}(X)) \xrightarrow{\psi_t} \mathsf{RLWE}_{\boldsymbol{z}(X^t)}(\boldsymbol{m}(X^t)) = (\boldsymbol{a}(X^t), \boldsymbol{b}(X^t))$$

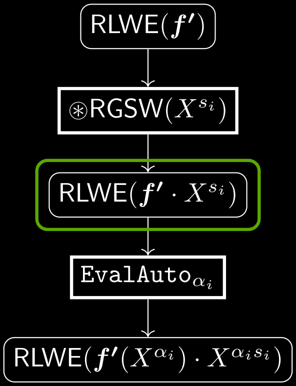
$$\texttt{KS}_{\boldsymbol{z}(X^t)\to\boldsymbol{z}(X)}\left(\mathsf{RLWE}_{\boldsymbol{z}(X^t)}(\boldsymbol{m}(X^t))\right) = \mathsf{RLWE}_{\boldsymbol{z}}(\boldsymbol{m}(X^t))$$

[3] Bonnoron et al. first used automorphisms to reduce the key size of a variant of the FHEW cryptosystem [BDF18].

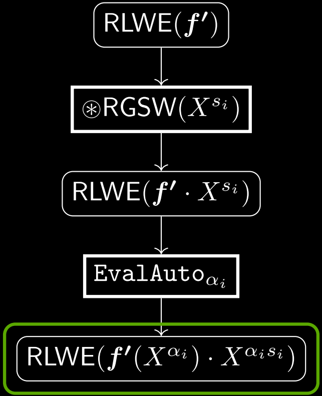
Linear Transformation with $+$ and $\times$



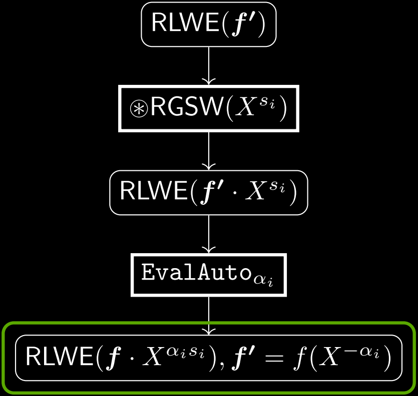
Linear Transformation with $+$ and $\times$



Linear Transformation with + and ×



Linear Transformation with $+$ and $\times$



Proposed Technique: Toy Example

- $\vec{\alpha} = (\alpha_0, \alpha_1, \alpha_2, \alpha_3) = (5, 25, 5, 1)$

- $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f'})$ ▷ $\boldsymbol{f'} = \boldsymbol{f}(X^{-25})$
- $\texttt{acc} \leftarrow \texttt{acc} \circledast \mathsf{RGSW}(X^{s_1})$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f'} \cdot X^{s_1})$
- $\texttt{EvalAuto}_5(\texttt{acc}, \texttt{ak}_5)$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f'}(X^5) \cdot X^{5s_1})$
- $\texttt{acc} \leftarrow \texttt{acc} \circledast \mathsf{RGSW}(X^{s_0})$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f'}(X^5) \cdot X^{s_0+5s_1})$
- $\texttt{acc} \leftarrow \texttt{acc} \circledast \mathsf{RGSW}(X^{s_2})$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f'}(X^5) \cdot X^{s_0+5s_1+s_2})$
- $\texttt{EvalAuto}_5(\texttt{acc}, \texttt{ak}_5)$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f'}(X^{25}) \cdot X^{5s_0+25s_1+5s_2})$
- $\texttt{acc} \leftarrow \texttt{acc} \circledast \mathsf{RGSW}(X^{s_3})$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f'}(X^{25}) \cdot X^{5s_0+25s_1+5s_2+s_3})$

- $\texttt{acc} = X^{\beta} \cdot \texttt{acc}$
- $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f} \cdot X^{\beta + \langle \vec{\alpha}, \vec{s} \rangle})$

Proposed Technique: Toy Example

- $\vec{\alpha} = (\alpha_0, \alpha_1, \alpha_2, \alpha_3) = (5, 25, 5, 1)$

- $\texttt{acc} = \text{RLWE}(\boldsymbol{f}')$ ▷ $\boldsymbol{f}' = \boldsymbol{f}(X^{-25})$
- $\texttt{acc} \leftarrow \texttt{acc} \circledast \text{RGSW}(X^{s_1})$ ▷ $\texttt{acc} = \text{RLWE}(\boldsymbol{f}' \cdot X^{s_1})$
- $\texttt{EvalAuto}_5(\texttt{acc}, \texttt{ak}_5)$ ▷ $\texttt{acc} = \text{RLWE}(\boldsymbol{f}'(X^5) \cdot X^{5s_1})$
- $\texttt{acc} \leftarrow \texttt{acc} \circledast \text{RGSW}(X^{s_0})$ ▷ $\texttt{acc} = \text{RLWE}(\boldsymbol{f}'(X^5) \cdot X^{s_0+5s_1})$
- $\texttt{acc} \leftarrow \texttt{acc} \circledast \text{RGSW}(X^{s_2})$ ▷ $\texttt{acc} = \text{RLWE}(\boldsymbol{f}'(X^5) \cdot X^{s_0+5s_1+s_2})$
- $\texttt{EvalAuto}_5(\texttt{acc}, \texttt{ak}_5)$ ▷ $\texttt{acc} = \text{RLWE}(\boldsymbol{f}'(X^{25}) \cdot X^{5s_0+25s_1+5s_2})$
- $\texttt{acc} \leftarrow \texttt{acc} \circledast \text{RGSW}(X^{s_3})$ ▷ $\texttt{acc} = \text{RLWE}(\boldsymbol{f}'(X^{25}) \cdot X^{5s_0+25s_1+5s_2+s_3})$

- $\texttt{acc} = X^{\beta} \cdot \texttt{acc}$
- $\texttt{acc} = \text{RLWE}(\boldsymbol{f} \cdot X^{\beta+\langle\vec{\alpha},\vec{s}\rangle})$

Proposed Technique: Toy Example

- $\vec{\alpha} = (\alpha_0, \alpha_1, \alpha_2, \alpha_3) = (5, 25, 5, 1)$

- $\mathsf{acc} = \mathsf{RLWE}(\boldsymbol{f'})$ ▷ $\boldsymbol{f'} = \boldsymbol{f}(X^{-25})$
- $\mathsf{acc} \leftarrow \mathsf{acc} \circledast \mathsf{RGSW}(X^{s_1})$ ▷ $\mathsf{acc} = \mathsf{RLWE}(\boldsymbol{f'} \cdot X^{s_1})$
- $\mathtt{EvalAuto}_5(\mathsf{acc}, \mathtt{ak}_5)$ ▷ $\mathsf{acc} = \mathsf{RLWE}(\boldsymbol{f'}(X^5) \cdot X^{5s_1})$
- $\mathsf{acc} \leftarrow \mathsf{acc} \circledast \mathsf{RGSW}(X^{s_0})$ ▷ $\mathsf{acc} = \mathsf{RLWE}(\boldsymbol{f'}(X^5) \cdot X^{s_0+5s_1})$
- $\mathsf{acc} \leftarrow \mathsf{acc} \circledast \mathsf{RGSW}(X^{s_2})$ ▷ $\mathsf{acc} = \mathsf{RLWE}(\boldsymbol{f'}(X^5) \cdot X^{s_0+5s_1+s_2})$
- $\mathtt{EvalAuto}_5(\mathsf{acc}, \mathtt{ak}_5)$ ▷ $\mathsf{acc} = \mathsf{RLWE}(\boldsymbol{f'}(X^{25}) \cdot X^{5s_0+25s_1+5s_2})$
- $\mathsf{acc} \leftarrow \mathsf{acc} \circledast \mathsf{RGSW}(X^{s_3})$ ▷ $\mathsf{acc} = \mathsf{RLWE}(\boldsymbol{f'}(X^{25}) \cdot X^{5s_0+25s_1+5s_2+s_3})$

- $\mathsf{acc} = X^{\beta} \cdot \mathsf{acc}$
- $\mathsf{acc} = \mathsf{RLWE}(\boldsymbol{f} \cdot X^{\beta + \langle \vec{\alpha}, \vec{s} \rangle})$

Proposed Technique: Toy Example

- $\vec{\alpha} = (\alpha_0, \alpha_1, \alpha_2, \alpha_3) = (5, 25, 5, 1)$

- $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f}')$ ▷ $\boldsymbol{f}' = \boldsymbol{f}(X^{-25})$
- $\texttt{acc} \leftarrow \texttt{acc} \circledast \mathsf{RGSW}(X^{s_1})$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f}' \cdot X^{s_1})$
- $\texttt{EvalAuto}_5(\texttt{acc}, \texttt{ak}_5)$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f}'(X^5) \cdot X^{5s_1})$
- $\texttt{acc} \leftarrow \texttt{acc} \circledast \mathsf{RGSW}(X^{s_0})$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f}'(X^5) \cdot X^{s_0+5s_1})$
- $\texttt{acc} \leftarrow \texttt{acc} \circledast \mathsf{RGSW}(X^{s_2})$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f}'(X^5) \cdot X^{s_0+5s_1+s_2})$
- $\texttt{EvalAuto}_5(\texttt{acc}, \texttt{ak}_5)$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f}'(X^{25}) \cdot X^{5s_0+25s_1+5s_2})$
- $\texttt{acc} \leftarrow \texttt{acc} \circledast \mathsf{RGSW}(X^{s_3})$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f}'(X^{25}) \cdot X^{5s_0+25s_1+5s_2+s_3})$

- $\texttt{acc} = X^{\beta} \cdot \texttt{acc}$
- $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f} \cdot X^{\beta + \langle \vec{\alpha}, \vec{s} \rangle})$

Proposed Technique: Toy Example

- $\vec{\alpha} = (\alpha_0, \alpha_1, \alpha_2, \alpha_3) = (5, 25, 5, 1)$

- $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f}')$ ▷ $\boldsymbol{f}' = \boldsymbol{f}(X^{-25})$
- $\texttt{acc} \leftarrow \texttt{acc} \circledast \mathsf{RGSW}(X^{s_1})$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f}' \cdot X^{s_1})$
- $\texttt{EvalAuto}_5(\texttt{acc}, \texttt{ak}_5)$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f}'(X^5) \cdot X^{5s_1})$
- $\texttt{acc} \leftarrow \texttt{acc} \circledast \mathsf{RGSW}(X^{s_0})$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f}'(X^5) \cdot X^{s_0+5s_1})$
- $\texttt{acc} \leftarrow \texttt{acc} \circledast \mathsf{RGSW}(X^{s_2})$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f}'(X^5) \cdot X^{s_0+5s_1+s_2})$
- $\texttt{EvalAuto}_5(\texttt{acc}, \texttt{ak}_5)$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f}'(X^{25}) \cdot X^{5s_0+25s_1+5s_2})$
- $\texttt{acc} \leftarrow \texttt{acc} \circledast \mathsf{RGSW}(X^{s_3})$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f}'(X^{25}) \cdot X^{5s_0+25s_1+5s_2+s_3})$

- $\texttt{acc} = X^{\beta} \cdot \texttt{acc}$
- $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f} \cdot X^{\beta+\langle\vec{\alpha},\vec{s}\rangle})$

Proposed Technique: Toy Example

- $\vec{\alpha} = (\alpha_0, \alpha_1, \alpha_2, \alpha_3) = (5, 25, 5, 1)$

- $\mathsf{acc} = \mathsf{RLWE}(\boldsymbol{f}')$ ▷ $\boldsymbol{f}' = \boldsymbol{f}(X^{-25})$
- $\mathsf{acc} \leftarrow \mathsf{acc} \circledast \mathsf{RGSW}(X^{s_1})$ ▷ $\mathsf{acc} = \mathsf{RLWE}(\boldsymbol{f}' \cdot X^{s_1})$
- $\mathtt{EvalAuto}_5(\mathsf{acc}, \mathtt{ak}_5)$ ▷ $\mathsf{acc} = \mathsf{RLWE}(\boldsymbol{f}'(X^5) \cdot X^{5s_1})$
- $\mathsf{acc} \leftarrow \mathsf{acc} \circledast \mathsf{RGSW}(X^{s_0})$ ▷ $\mathsf{acc} = \mathsf{RLWE}(\boldsymbol{f}'(X^5) \cdot X^{s_0+5s_1})$
- $\mathsf{acc} \leftarrow \mathsf{acc} \circledast \mathsf{RGSW}(X^{s_2})$ ▷ $\mathsf{acc} = \mathsf{RLWE}(\boldsymbol{f}'(X^5) \cdot X^{s_0+5s_1+s_2})$
- $\mathtt{EvalAuto}_5(\mathsf{acc}, \mathtt{ak}_5)$ ▷ $\mathsf{acc} = \mathsf{RLWE}(\boldsymbol{f}'(X^{25}) \cdot X^{5s_0+25s_1+5s_2})$
- $\mathsf{acc} \leftarrow \mathsf{acc} \circledast \mathsf{RGSW}(X^{s_3})$ ▷ $\mathsf{acc} = \mathsf{RLWE}(\boldsymbol{f}'(X^{25}) \cdot X^{5s_0+25s_1+5s_2+s_3})$

- $\mathsf{acc} = X^{\beta} \cdot \mathsf{acc}$
- $\mathsf{acc} = \mathsf{RLWE}(\boldsymbol{f} \cdot X^{\beta + \langle \vec{\alpha}, \vec{s} \rangle})$

Proposed Technique: Toy Example

- $\vec{\alpha} = (\alpha_0, \alpha_1, \alpha_2, \alpha_3) = (5, 25, 5, 1)$

- $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f}')$ ▷ $\boldsymbol{f}' = \boldsymbol{f}(X^{-25})$
- $\texttt{acc} \leftarrow \texttt{acc} \circledast \mathsf{RGSW}(X^{s_1})$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f}' \cdot X^{s_1})$
- $\texttt{EvalAuto}_5(\texttt{acc}, \texttt{ak}_5)$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f}'(X^5) \cdot X^{5s_1})$
- $\texttt{acc} \leftarrow \texttt{acc} \circledast \mathsf{RGSW}(X^{s_0})$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f}'(X^5) \cdot X^{s_0+5s_1})$
- $\texttt{acc} \leftarrow \texttt{acc} \circledast \mathsf{RGSW}(X^{s_2})$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f}'(X^5) \cdot X^{s_0+5s_1+s_2})$
- $\texttt{EvalAuto}_5(\texttt{acc}, \texttt{ak}_5)$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f}'(X^{25}) \cdot X^{5s_0+25s_1+5s_2})$
- $\texttt{acc} \leftarrow \texttt{acc} \circledast \mathsf{RGSW}(X^{s_3})$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f}'(X^{25}) \cdot X^{5s_0+25s_1+5s_2+s_3})$

- $\texttt{acc} = X^{\beta} \cdot \texttt{acc}$
- $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f} \cdot X^{\beta + \langle \vec{\alpha}, \vec{s} \rangle})$

Proposed Technique: Toy Example

- $\vec{\alpha} = (\alpha_0, \alpha_1, \alpha_2, \alpha_3) = (5, 25, 5, 1)$

- $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f'})$ ▷ $\boldsymbol{f'} = \boldsymbol{f}(X^{-25})$
- $\texttt{acc} \leftarrow \texttt{acc} \circledast \mathsf{RGSW}(X^{s_1})$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f'} \cdot X^{s_1})$
- $\texttt{EvalAuto}_5(\texttt{acc}, \texttt{ak}_5)$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f'}(X^5) \cdot X^{5s_1})$
- $\texttt{acc} \leftarrow \texttt{acc} \circledast \mathsf{RGSW}(X^{s_0})$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f'}(X^5) \cdot X^{s_0+5s_1})$
- $\texttt{acc} \leftarrow \texttt{acc} \circledast \mathsf{RGSW}(X^{s_2})$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f'}(X^5) \cdot X^{s_0+5s_1+s_2})$
- $\texttt{EvalAuto}_5(\texttt{acc}, \texttt{ak}_5)$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f'}(X^{25}) \cdot X^{5s_0+25s_1+5s_2})$
- $\texttt{acc} \leftarrow \texttt{acc} \circledast \mathsf{RGSW}(X^{s_3})$ ▷ $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f'}(X^{25}) \cdot X^{5s_0+25s_1+5s_2+s_3})$

- $\texttt{acc} = X^{\beta} \cdot \texttt{acc}$
- $\texttt{acc} = \mathsf{RLWE}(\boldsymbol{f} \cdot X^{\beta+\langle\vec{\alpha},\vec{s}\rangle})$

Proposed Technique: Toy Example

- $\vec{\alpha} = (\alpha_0, \alpha_1, \alpha_2, \alpha_3) = (5, 25, 5, 1)$

- $\mathsf{acc} = \mathsf{RLWE}(\boldsymbol{f'})$ ▷ $\boldsymbol{f'} = \boldsymbol{f}(X^{-25})$
- $\mathsf{acc} \leftarrow \mathsf{acc} \circledast \mathsf{RGSW}(X^{s_1})$ ▷ $\mathsf{acc} = \mathsf{RLWE}(\boldsymbol{f'} \cdot X^{s_1})$
- $\mathtt{EvalAuto}_5(\mathsf{acc}, \mathtt{ak}_5)$ ▷ $\mathsf{acc} = \mathsf{RLWE}(\boldsymbol{f'}(X^5) \cdot X^{5s_1})$
- $\mathsf{acc} \leftarrow \mathsf{acc} \circledast \mathsf{RGSW}(X^{s_0})$ ▷ $\mathsf{acc} = \mathsf{RLWE}(\boldsymbol{f'}(X^5) \cdot X^{s_0+5s_1})$
- $\mathsf{acc} \leftarrow \mathsf{acc} \circledast \mathsf{RGSW}(X^{s_2})$ ▷ $\mathsf{acc} = \mathsf{RLWE}(\boldsymbol{f'}(X^5) \cdot X^{s_0+5s_1+s_2})$
- $\mathtt{EvalAuto}_5(\mathsf{acc}, \mathtt{ak}_5)$ ▷ $\mathsf{acc} = \mathsf{RLWE}(\boldsymbol{f'}(X^{25}) \cdot X^{5s_0+25s_1+5s_2})$
- $\mathsf{acc} \leftarrow \mathsf{acc} \circledast \mathsf{RGSW}(X^{s_3})$ ▷ $\mathsf{acc} = \mathsf{RLWE}(\boldsymbol{f'}(X^{25}) \cdot X^{5s_0+25s_1+5s_2+s_3})$

- $\mathsf{acc} = X^{\beta} \cdot \mathsf{acc}$
- $\mathsf{acc} = \mathsf{RLWE}(\boldsymbol{f} \cdot X^{\beta + \langle \vec{\alpha}, \vec{s} \rangle})$

Proposed Technique

In the toy example:

- RGSW(X^{s_i}) to add s_i to the exponent
- `EvalAuto` to multiply α_i in the exponent
- Only one automorphism key $\mathtt{ak}_5$ is required,
  - as 5 and 25 are powers of 5

Let's extend it to full blind rotation

- $\{5, -1\}$ generates $\mathbb{Z}_{2N}^*$ (say, $g = 5$)
- Only $\mathtt{ak}_g$ and $\mathtt{ak}_{-1}$ is required: $O(1)$ automorphism keys[4]
- Computations:
  - n multiplication of RGSW(X^{s_i})
  - (at most) N `EvalAutos`

[4] In practice, using $\mathtt{ak}_{-g}$ instead of $\mathtt{ak}_{-1}$ improves the performance

Proposed Technique

In the toy example:

- RGSW(X^{s_i}) to add s_i to the exponent
- `EvalAuto` to multiply α_i in the exponent
- Only one automorphism key $\mathtt{ak}_5$ is required,
  - as 5 and 25 are powers of 5

Let's extend it to full blind rotation

- $\{5, -1\}$ generates $\mathbb{Z}_{2N}^*$ (say, $g = 5$)
- Only $\mathtt{ak}_g$ and $\mathtt{ak}_{-1}$ is required: $O(1)$ automorphism keys[4]
- Computations:
  - n multiplication of RGSW(X^{s_i})
  - (at most) N `EvalAutos`

[4] In practice, using $\mathtt{ak}_{-g}$ instead of $\mathtt{ak}_{-1}$ improves the performance

Proposed Technique

In the toy example:

- RGSW(X^{s_i}) to add s_i to the exponent
- `EvalAuto` to multiply α_i in the exponent
- Only one automorphism key $\mathtt{ak}_5$ is required,
  - as 5 and 25 are powers of 5

Let's extend it to full blind rotation

- $\{5, -1\}$ generates $\mathbb{Z}_{2N}^*$ (say, $g = 5$)
- Only $\mathtt{ak}_g$ and $\mathtt{ak}_{-1}$ is required: $O(1)$ automorphism keys[4]
- Computations:
  - n multiplication of RGSW(X^{s_i})
  - (at most) N `EvalAutos`

[4] In practice, using $\mathtt{ak}_{-g}$ instead of $\mathtt{ak}_{-1}$ improves the performance

Proposed Technique

In the toy example:

- $\mathsf{RGSW}(X^{s_i})$ to add s_i to the exponent
- `EvalAuto` to multiply α_i in the exponent
- Only one automorphism key $\mathtt{ak}_5$ is required,
  - as 5 and 25 are powers of 5

Let's extend it to full blind rotation

- $\{5, -1\}$ generates $\mathbb{Z}_{2N}^*$ (say, $g = 5$)
- Only $\mathtt{ak}_g$ and $\mathtt{ak}_{-1}$ is required: $O(1)$ automorphism keys[4]
- Computations:
  - n multiplication of $\mathsf{RGSW}(X^{s_i})$
  - (at most) N `EvalAutos`

[4] In practice, using $\mathtt{ak}_{-g}$ instead of $\mathtt{ak}_{-1}$ improves the performance

Proposed Technique

In the toy example:

- RGSW(X^{s_i}) to add s_i to the exponent
- `EvalAuto` to multiply α_i in the exponent
- Only one automorphism key $\texttt{ak}_5$ is required,
  - as 5 and 25 are powers of 5

Let's extend it to full blind rotation

- $\{5, -1\}$ generates $\mathbb{Z}_{2N}^*$ (say, $g = 5$)
- Only $\texttt{ak}_g$ and $\texttt{ak}_{-1}$ is required: $O(1)$ automorphism keys[4]
- Computations:
  - n multiplication of RGSW(X^{s_i})
  - (at most) N `EvalAutos`

[4] In practice, using $\texttt{ak}_{-g}$ instead of $\texttt{ak}_{-1}$ improves the performance

Proposed Technique

In the toy example:

- RGSW(X^{s_i}) to add s_i to the exponent
- `EvalAuto` to multiply α_i in the exponent
- Only one automorphism key $\mathtt{ak}_5$ is required,
  - as 5 and 25 are powers of 5

Let's extend it to full blind rotation

- $\{5, -1\}$ generates $\mathbb{Z}_{2N}^*$ (say, $g = 5$)
- Only $\mathtt{ak}_g$ and $\mathtt{ak}_{-1}$ is required: $O(1)$ automorphism keys[4]
- Computations:
  - n multiplication of RGSW(X^{s_i})
  - (at most) N `EvalAutos`

[4] In practice, using $\mathtt{ak}_{-g}$ instead of $\mathtt{ak}_{-1}$ improves the performance

Proposed Technique: Core Algorithm

- Let $I_\ell^+ = \{i : \alpha_i = g^\ell\}$ and $I_\ell^- = \{i : \alpha_i = -g^\ell\}$, for $\ell \in [0, N/2-1]$
- Using the fact that $g^{N/2} = 1 \pmod{2N}$ we have the following decomposition

$$\sum_i \alpha_i s_i = \left(\sum_{j \in I_0^+} s_j + \cdots + g\left(\sum_{j \in I_{N/2-1}^+} s_j - g\left(\sum_{j \in I_0^-} s_j + \cdots + g\left(\sum_{j \in I_{N/2-1}^-} s_j\right)\right)\right)\right)$$

Proposed Technique: Core Algorithm

- Let $I_\ell^+ = \{i : \alpha_i = g^\ell\}$ and $I_\ell^- = \{i : \alpha_i = -g^\ell\}$, for $\ell \in [0, N/2-1]$
- Using the fact that $g^{N/2} = 1 \pmod{2N}$ we have the following decomposition

$$\sum_i \alpha_i s_i = \left(\sum_{j\in I_0^+} s_j + \cdots + g\left(\sum_{j\in I^+_{N/2-1}} s_j - g\left(\sum_{j\in I_0^-} s_j + \cdots + g\left(\sum_{j\in I^-_{N/2-1}} s_j\right)\right)\right)\right)$$

Proposed Technique: Core Algorithm

- Let $I_\ell^+ = \{i : \alpha_i = g^\ell\}$ and $I_\ell^- = \{i : \alpha_i = -g^\ell\}$, for $\ell \in [0, N/2-1]$
- Using the fact that $g^{N/2} = 1 \pmod{2N}$ we have the following decomposition

$$\sum_i \alpha_i s_i = \left(\sum_{j \in I_0^+} s_j + \cdots + g\left(\sum_{j \in I_{N/2-1}^+} s_j - g\left(\sum_{j \in I_0^-} s_j + \cdots + g\left(\sum_{j \in I_{N/2-1}^-} s_j\right)\right)\right)\right)$$

The Core Blind Rotation Algorithm

- Given an initial ciphertext $\texttt{acc} = \mathsf{RLWE}^0_{\boldsymbol{z}}\left(\boldsymbol{f}'(X)\right)$,
- we first multiply it by $\texttt{brk}_j$ for all $j \in I^-_{N/2-1}$, $\texttt{brk}_j := \mathsf{RGSW}_{\boldsymbol{z}}\left(X^{s_j}\right)$

$$\texttt{acc} = \mathsf{RLWE}_{\boldsymbol{z}}\left(\boldsymbol{f}' \cdot X^{\sum_{j \in I^-_{N/2-1}} s_j}\right)$$

- then apply automorphism $\texttt{EvalAuto}_g$ to acc and obtain

$$\texttt{acc} = \mathsf{RLWE}_{\boldsymbol{z}}\left(\boldsymbol{f}'(X^g) \cdot X^{g \cdot \sum_{j \in I^-_{N/2-1}} s_j}\right)$$

- Then we multiply the accumulator by $\texttt{brk}_j$ for $j \in I^-_{N/2-2}$ and again apply automorphism $\texttt{EvalAuto}_g$ to acc
- This process is repeated for both I^-_ℓ and I^+_ℓ for all $\ell = N/2-1, \ldots, 0$

The Core Blind Rotation Algorithm

- Given an initial ciphertext $\texttt{acc} = \mathsf{RLWE}^0_{\boldsymbol{z}}\left(\boldsymbol{f}'(X)\right)$,
- we first multiply it by $\texttt{brk}_j$ for all $j \in I^-_{N/2-1}$, $\texttt{brk}_j := \mathsf{RGSW}_{\boldsymbol{z}}\left(X^{s_j}\right)$

$$\texttt{acc} = \mathsf{RLWE}_{\boldsymbol{z}}\left(\boldsymbol{f}' \cdot X^{\sum_{j \in I^-_{N/2-1}} s_j}\right)$$

- then apply automorphism $\texttt{EvalAuto}_g$ to acc and obtain

$$\texttt{acc} = \mathsf{RLWE}_{\boldsymbol{z}}\left(\boldsymbol{f}'(X^g) \cdot X^{g \cdot \sum_{j \in I^-_{N/2-1}} s_j}\right)$$

- Then we multiply the accumulator by $\texttt{brk}_j$ for $j \in I^-_{N/2-2}$ and again apply automorphism $\texttt{EvalAuto}_g$ to acc
- This process is repeated for both I^-_ℓ and I^+_ℓ for all $\ell = N/2-1, ..., 0$

The Core Blind Rotation Algorithm

- Given an initial ciphertext $\texttt{acc} = \mathsf{RLWE}^0_{\boldsymbol{z}}\left(\boldsymbol{f}'(X)\right)$,
- we first multiply it by $\texttt{brk}_j$ for all $j \in I^-_{N/2-1}$, $\texttt{brk}_j := \mathsf{RGSW}_{\boldsymbol{z}}\left(X^{s_j}\right)$

$$\texttt{acc} = \mathsf{RLWE}_{\boldsymbol{z}}\left(\boldsymbol{f}' \cdot X^{\sum_{j \in I^-_{N/2-1}} s_j}\right)$$

- then apply automorphism $\texttt{EvalAuto}_g$ to acc and obtain

$$\texttt{acc} = \mathsf{RLWE}_{\boldsymbol{z}}\left(\boldsymbol{f}'(X^g) \cdot X^{g \cdot \sum_{j \in I^-_{N/2-1}} s_j}\right)$$

- Then we multiply the accumulator by $\texttt{brk}_j$ for $j \in I^-_{N/2-2}$ and again apply automorphism $\texttt{EvalAuto}_g$ to acc
- This process is repeated for both I^-_ℓ and I^+_ℓ for all $\ell = N/2-1, \ldots, 0$

The Core Blind Rotation Algorithm

- Given an initial ciphertext $\texttt{acc} = \mathsf{RLWE}^0_{\boldsymbol{z}}(\boldsymbol{f}'(X))$,
- we first multiply it by $\texttt{brk}_j$ for all $j \in I^-_{N/2-1}$, $\texttt{brk}_j := \mathsf{RGSW}_{\boldsymbol{z}}(X^{s_j})$

$$\texttt{acc} = \mathsf{RLWE}_{\boldsymbol{z}}\left(\boldsymbol{f}' \cdot X^{\sum_{j \in I^-_{N/2-1}} s_j}\right)$$

- then apply automorphism $\texttt{EvalAuto}_g$ to acc and obtain

$$\texttt{acc} = \mathsf{RLWE}_{\boldsymbol{z}}\left(\boldsymbol{f}'(X^g) \cdot X^{g \cdot \sum_{j \in I^-_{N/2-1}} s_j}\right)$$

- Then we multiply the accumulator by $\texttt{brk}_j$ for $j \in I^-_{N/2-2}$ and again apply automorphism $\texttt{EvalAuto}_g$ to acc
- This process is repeated for both I^-_ℓ and I^+_ℓ for all $\ell = N/2-1, \ldots, 0$

The Core Blind Rotation Algorithm

- Given an initial ciphertext $\texttt{acc} = \mathsf{RLWE}^0_{\boldsymbol{z}}(\boldsymbol{f}'(X))$,
- we first multiply it by $\texttt{brk}_j$ for all $j \in I^-_{N/2-1}$, $\texttt{brk}_j := \mathsf{RGSW}_{\boldsymbol{z}}(X^{s_j})$

$$\texttt{acc} = \mathsf{RLWE}_{\boldsymbol{z}}\left(\boldsymbol{f}' \cdot X^{\sum_{j \in I^-_{N/2-1}} s_j}\right)$$

- then apply automorphism $\texttt{EvalAuto}_g$ to acc and obtain

$$\texttt{acc} = \mathsf{RLWE}_{\boldsymbol{z}}\left(\boldsymbol{f}'(X^g) \cdot X^{g \cdot \sum_{j \in I^-_{N/2-1}} s_j}\right)$$

- Then we multiply the accumulator by $\texttt{brk}_j$ for $j \in I^-_{N/2-2}$ and again apply automorphism $\texttt{EvalAuto}_g$ to acc
- This process is repeated for both I^-_ℓ and I^+_ℓ for all $\ell = N/2 - 1, ..., 0$

The Core Blind Rotation Algorithm

Algorithm 1 Core Blind Rotation Sub Algorithm for odd α_i

1: **procedure** BLINDROTATECORE$\Big(\texttt{acc}, \vec{\alpha}, \{\texttt{brk}_i\}_{i\in[0,n-1]}, \{\texttt{ak}_{g^u}\}_{u\in[1,w]}, \texttt{ak}_{-g}\Big)$
2: $\quad v \leftarrow 0$
3: $\quad$ **for** $(\ell = N/2 - 1; \ell > 0; \ell = \ell - 1)$ **do**
4: $\quad\quad$ **for** $j \in I_\ell^-$ **do**
5: $\quad\quad\quad \texttt{acc} \leftarrow \texttt{acc} \circledast \texttt{brk}_j$
6: $\quad\quad v \leftarrow v + 1$
7: $\quad\quad$ **if** $(I_{\ell-1}^- \neq \emptyset$ or $v = w$ or $l = 1)$ **then**
8: $\quad\quad\quad \texttt{acc} \leftarrow \texttt{EvalAuto}_{g^v}(\texttt{acc}, \texttt{ak}_{g^v})$
9: $\quad\quad\quad v \leftarrow 0$
10: $\quad$ **for** $j \in I_0^-$ **do**
11: $\quad\quad \texttt{acc} \leftarrow \texttt{acc} \circledast \texttt{brk}_j$
12: $\quad \texttt{acc} \leftarrow \texttt{EvalAuto}_{-g}(\texttt{acc}, \texttt{ak}_{-g})$
13: $\quad$ **for** $(\ell = N/2 - 1; \ell > 0; \ell = \ell - 1)$ **do**
14: $\quad\quad$ Repeat Line 4 – 9
15: $\quad$ **for** $j \in I_0^+$ **do**
16: $\quad\quad \texttt{acc} \leftarrow \texttt{acc} \circledast \texttt{brk}_j$
17: **return** acc

Odd α_i: Round-to-odds

- Limitation: automorphism exists only for odd numbers in $\mathbb{Z}_{2N}$
- Each α_i should be odd
- Several variants are proposed

Odd α_i: Round-to-odds

- Limitation: automorphism exists only for odd numbers in $\mathbb{Z}_{2N}$
- Each α_i should be odd
- Several variants are proposed

Recall: FHEW Bootstrapping

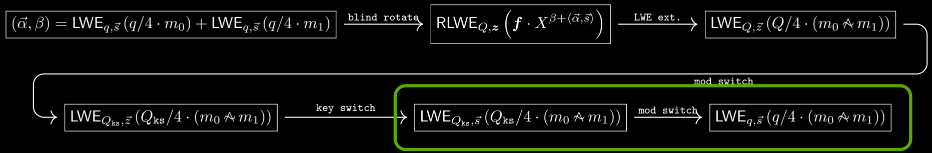


Figure: NAND gate bootstrapping procedure of FHEW scheme [DM15, MP21]

Odd α_i: Round-to-odds

- Previously: for $(\vec{\alpha}', \beta') = \mathsf{LWE}_{Q_{\mathtt{ks}}}(Q_{\mathtt{ks}}/4 \cdot m)$,

$$\left(\vec{\alpha} = \left\lfloor \frac{q}{Q_{\mathtt{ks}}} \cdot \vec{\alpha}' \right\rceil, \beta = \left\lfloor \frac{q}{Q_{\mathtt{ks}}} \cdot \beta' \right\rceil\right) = \mathsf{LWE}_q(q/4 \cdot m)$$

- New modulus reduction:

$$\left(\vec{\alpha} = \left\lfloor \frac{2N}{Q_{\mathtt{ks}}} \cdot \vec{\alpha}' \right\rceil_{\mathtt{odd}}, \beta = \left\lfloor \frac{2N}{Q_{\mathtt{ks}}} \cdot \beta' \right\rceil_{\mathtt{odd}}\right) = \mathsf{LWE}_{2N}(q/4 \cdot m)$$

- $\lfloor \cdot \rceil_{\mathtt{odd}}$ finds the nearest odd integer

Odd α_i: Round-to-odds

- Previously: for $(\vec{\alpha}', \beta') = \mathsf{LWE}_{Q_{\mathtt{ks}}}(Q_{\mathtt{ks}}/4 \cdot m)$,

$$\left(\vec{\alpha} = \left\lfloor \frac{q}{Q_{\mathtt{ks}}} \cdot \vec{\alpha}' \right\rceil, \beta = \left\lfloor \frac{q}{Q_{\mathtt{ks}}} \cdot \beta' \right\rceil\right) = \mathsf{LWE}_q(q/4 \cdot m)$$

- New modulus reduction:

$$\left(\vec{\alpha} = \left\lfloor \frac{2N}{Q_{\mathtt{ks}}} \cdot \vec{\alpha}' \right\rceil_{\mathtt{odd}}, \beta = \left\lfloor \frac{2N}{Q_{\mathtt{ks}}} \cdot \beta' \right\rceil_{\mathtt{odd}}\right) = \mathsf{LWE}_{2N}(q/4 \cdot m)$$

- $\lfloor \cdot \rceil_{\mathtt{odd}}$ finds the nearest odd integer

Odd α_i: Round-to-odds

- Previously: for $(\vec{\alpha}', \beta') = \mathsf{LWE}_{Q_{\texttt{ks}}}(Q_{\texttt{ks}}/4 \cdot m)$,

$$\left(\vec{\alpha} = \left\lfloor \frac{q}{Q_{\texttt{ks}}} \cdot \vec{\alpha}' \right\rceil, \beta = \left\lfloor \frac{q}{Q_{\texttt{ks}}} \cdot \beta' \right\rceil\right) = \mathsf{LWE}_q(q/4 \cdot m)$$

- New modulus reduction:

$$\left(\vec{\alpha} = \left\lfloor \frac{2N}{Q_{\texttt{ks}}} \cdot \vec{\alpha}' \right\rceil_{\texttt{odd}}, \beta = \left\lfloor \frac{2N}{Q_{\texttt{ks}}} \cdot \beta' \right\rceil_{\texttt{odd}}\right) = \mathsf{LWE}_{2N}(q/4 \cdot m)$$

- $\lfloor \cdot \rceil_{\texttt{odd}}$ finds the nearest odd integer

Multiple Automorphism Keys

When $I_\ell^+ = \emptyset$,

- $\dots$
- Multiply RGSW(X^{s_j}) for $j \in I_{\ell+1}^+$
- $\texttt{EvalAuto}_g$
- (Nothing to do): multiply RGSW(X^{s_j}) for $j \in I_\ell^+$
- $\texttt{EvalAuto}_g$
- Multiply RGSW(X^{s_j}) for $j \in I_{\ell-1}^+$
- $\dots$

If we have $\texttt{ak}_{g^2}$

- $\dots$
- Multiply RGSW(X^{s_j}) for $j \in I_{\ell+1}^+$
- $\texttt{EvalAuto}_{g^2}$
- Multiply RGSW(X^{s_j}) for $j \in I_{\ell-1}^+$
- $\dots$

Multiple Automorphism Keys

When $I_\ell^+ = \emptyset$,

- $\dots$
- Multiply $\mathsf{RGSW}(X^{s_j})$ for $j \in I_{\ell+1}^+$
- $\texttt{EvalAuto}_g$
- (Nothing to do): multiply $\mathsf{RGSW}(X^{s_j})$ for $j \in I_\ell^+$
- $\texttt{EvalAuto}_g$
- Multiply $\mathsf{RGSW}(X^{s_j})$ for $j \in I_{\ell-1}^+$
- $\dots$

If we have $\texttt{ak}_{g^2}$

- $\dots$
- Multiply $\mathsf{RGSW}(X^{s_j})$ for $j \in I_{\ell+1}^+$
- $\texttt{EvalAuto}_{g^2}$
- Multiply $\mathsf{RGSW}(X^{s_j})$ for $j \in I_{\ell-1}^+$
- $\dots$

Multiple Automorphism Keys

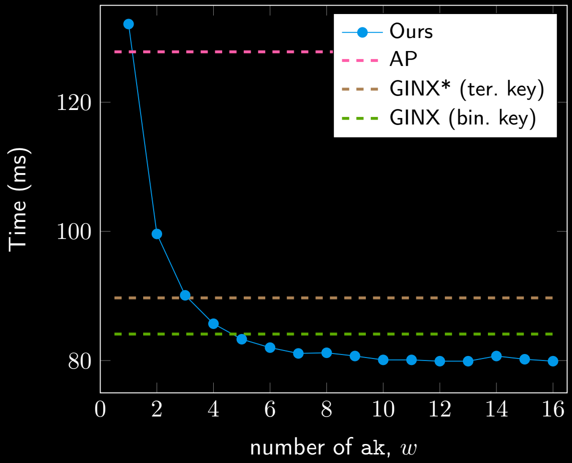


Figure: Bootstrapping performance by number of ak. [5]

[5] $\#\mathtt{ak} = \log N$ is enough. Analysis is available on paper.

Outline

Comparison of Blind Rotation Techniques

Table: Key size, complexity, and error variance of each technique (normalized). $|U| = 1$ for binary, and 2 for ternary.

Method	# keys	# mult	$\sigma^2_{\text{acc}}/\sigma^2_{\odot}$
AP [AP14, DM15]	$2d_r(B_r - 1)n$	$2d_r\left(1 - \frac{1}{B_r}\right)n$	$2d_r\left(1 - \frac{1}{B_r}\right)n$
GINX [GINX16, CGGI20, MP21]	$2\lvert U\rvert n$	$2\lvert U\rvert n$	$4\lvert U\rvert n$
GINX* [KDE+21, BIP+22]	$4n$	$2n$	$8n$
Proposed	$2n + w + 1$	$2n + \frac{w-1}{w}\kappa + \frac{N}{w}$	$2n + \frac{w-1}{w}\kappa + \frac{N}{w}$

Gaussian Secrets for Efficiency

Table: Optimized parameter sets for FHEW schemes.[6]

Parameter set	key	n	q	N	Q	d_g	d_{ks}	$\lambda^{\dagger}_{\min}$
128_Ours/AP	$\sigma = 3.2$	458	1024	1024	2^{28}	3	2	128.2
128_tGINX	ternary	531	2048	1024	2^{26}	4	2	128.5
128_bGINX	binary	571	2048	1024	2^{25}	4	2	128.1
STD128_OPT [MP21]	ternary	502	1024	1024	2^{27}	4	2	121.0
TFHE [TFH]	binary	630	$\sigma = 2^{-15}$	1024	$\sigma = 2^{-25}$	3	2	115.11

Gaussian secret improves the efficiency!

[6] Security is measured by lattice estimator.

Gaussian Secrets for Efficiency

Table: Optimized parameter sets for FHEW schemes.[6]

Parameter set	key	n	q	N	Q	d_g	d_{ks}	$\lambda^{\dagger}_{min}$
128_Ours/AP	$\sigma = 3.2$	458	1024	1024	2^{28}	3	2	128.2
128_tGINX	ternary	531	2048	1024	2^{26}	4	2	128.5
128_bGINX	binary	571	2048	1024	2^{25}	4	2	128.1
STD128_OPT [MP21]	ternary	502	1024	1024	2^{27}	4	2	121.0
TFHE [TFH]	binary	630	$\sigma = 2^{-15}$	1024	$\sigma = 2^{-25}$	3	2	115.11

Gaussian secret improves the efficiency!

[6] Security is measured by lattice estimator.

Implementation Results with Optimized Parameters

Table: Implementation result (average of 400, $\#\mathtt{ak} = 10$ for our method), blind rotation key size, and failure probability for FHEW bootstrapping (NAND gate).

Parameter set	Method	Runtime [ms]	Key size [MB]	Fail. prob. $(\leq 2^{-32})$
128_Ours/AP	Proposed	80.1	12.67	$2^{-85.68}$
128_Ours/AP	AP	127.8	776.45	$2^{-77.74}$
128_tGINX	GINX*	89.7	40.45	$2^{-93.84}$
128_bGINX	GINX	84.1	20.91	$2^{-79.82}$

Faster bootstrapping, smaller bootstrapping key size

Implementation Results with Optimized Parameters

Table: Implementation result (average of 400, $\#\mathtt{ak} = 10$ for our method), blind rotation key size, and failure probability for FHEW bootstrapping (NAND gate).

Parameter set	Method	Runtime [ms]	Key size [MB]	Fail. prob. $(\leq 2^{-32})$
128_Ours/AP	Proposed	80.1	12.67	$2^{-85.68}$
128_Ours/AP	AP	127.8	776.45	$2^{-77.74}$
128_tGINX	GINX*	89.7	40.45	$2^{-93.84}$
128_bGINX	GINX	84.1	20.91	$2^{-79.82}$

Faster bootstrapping, smaller bootstrapping key size

Outline

Threshold Homomorphic Encryption

- A more compelling motivation to use larger secret keys
- Distribute a secret key s among a set of participants, say $P_1, \ldots, P_k$
- each holding a share s_i, and they can collaboratively decrypt messages.

Threshold Homomorphic Encryption

- A more compelling motivation to use larger secret keys
- Distribute a secret key s among a set of participants, say $P_1, \ldots, P_k$
- each holding a share s_i, and they can collaboratively decrypt messages.

Shared Secret Key and Public Key

Each participant $j \in J$ has (J: set of participants)

1. the secret keys $\vec{s}_j$ for LWE encryption
2. and z_j for RLWE encryption,

The common secret keys:

- $\vec{s}_* = \sum_{j \in J} \vec{s}_j$
- $\boldsymbol{z}_* = \sum_{j \in J} \boldsymbol{z}_j$.

The public key:

- $\mathrm{pk}_{\boldsymbol{z}_*}^{\mathsf{RLWE}} = (\boldsymbol{a}_{\mathtt{crs}}, \sum_{j \in J} \boldsymbol{b}_j)$, where $\boldsymbol{b}_j = -\boldsymbol{a}_{\mathtt{crs}} \cdot \boldsymbol{z}_j + \boldsymbol{e}_j$

Shared Secret Key and Public Key

Each participant $j \in J$ has (J: set of participants)

1. the secret keys $\vec{s}_j$ for LWE encryption
2. and z_j for RLWE encryption,

The common secret keys:

- $\vec{s}_* = \sum_{j \in J} \vec{s}_j$
- $\boldsymbol{z}_* = \sum_{j \in J} \boldsymbol{z}_j$.

The public key:

- $\mathrm{pk}^{\mathsf{RLWE}}_{\boldsymbol{z}_*} = (\boldsymbol{a}_{\mathrm{crs}}, \sum_{j \in J} \boldsymbol{b}_j)$, where $\boldsymbol{b}_j = -\boldsymbol{a}_{\mathrm{crs}} \cdot \boldsymbol{z}_j + \boldsymbol{e}_j$

Shared Secret Key and Public Key

Each participant $j \in J$ has (J: set of participants)

1. the secret keys $\vec{s}_j$ for LWE encryption
2. and $\boldsymbol{z}_j$ for RLWE encryption,

The common secret keys:

- $\vec{s}_* = \sum_{j \in J} \vec{s}_j$
- $\boldsymbol{z}_* = \sum_{j \in J} \boldsymbol{z}_j$.

The public key:

- $\mathrm{pk}_{\boldsymbol{z}_*}^{\mathsf{RLWE}} = (\boldsymbol{a}_{\mathtt{crs}}, \sum_{j \in J} \boldsymbol{b}_j)$, where $\boldsymbol{b}_j = -\boldsymbol{a}_{\mathtt{crs}} \cdot \boldsymbol{z}_j + \boldsymbol{e}_j$

Generation of Automorphism Keys

Generation of $\mathsf{RLWE}'_{z_*}\left(z_*(X^i)\right)$:

- Using the shared public key $\mathtt{pk}_{z_*}^{\mathsf{RLWE}}$, j generates

$$\mathtt{ak}_{j,k}^{Thr} = \mathsf{RLWE}'_{z_*}\left(z_j(X^k)\right)$$

- Next, each participant sends $\mathtt{ak}_{j,k}^{Thr}$ to the computing party.
- The computing party generates automorphism keys $\mathtt{ak}_k^{Thr}$ as follows

$$\mathtt{ak}_k^{Thr} := \sum_{j \in J} \mathtt{ak}_{j,k}^{Thr} = \sum_{j \in J} \mathsf{RLWE}'_{z_*}\left(z_j(X^k)\right) = \mathsf{RLWE}'_{z_*}\left(z_*(X^k)\right).$$

Generation of Automorphism Keys

Generation of $\mathsf{RLWE}'_{z_*}\left(z_*(X^i)\right)$:

- Using the shared public key $\mathtt{pk}^{\mathsf{RLWE}}_{z_*}$, j generates

$$\mathtt{ak}^{Thr}_{j,k} = \mathsf{RLWE}'_{z_*}\left(z_j(X^k)\right)$$

- Next, each participant sends $\mathtt{ak}^{Thr}_{j,k}$ to the computing party.
- The computing party generates automorphism keys $\mathtt{ak}^{Thr}_{k}$ as follows

$$\mathtt{ak}^{Thr}_{k} := \sum_{j \in J} \mathtt{ak}^{Thr}_{j,k} = \sum_{j \in J} \mathsf{RLWE}'_{z_*}\left(z_j(X^k)\right) = \mathsf{RLWE}'_{z_*}\left(z_*(X^k)\right).$$

Generation of Automorphism Keys

Generation of $\mathsf{RLWE}'_{\boldsymbol{z}_*}\left(z_*(X^i)\right)$:

- Using the shared public key $\mathtt{pk}^{\mathsf{RLWE}}_{\boldsymbol{z}_*}$, j generates

$$\mathtt{ak}^{Thr}_{j,k} = \mathsf{RLWE}'_{\boldsymbol{z}_*}\left(\boldsymbol{z}_j(X^k)\right)$$

- Next, each participant sends $\mathtt{ak}^{Thr}_{j,k}$ to the computing party.
- The computing party generates automorphism keys $\mathtt{ak}^{Thr}_{k}$ as follows

$$\mathtt{ak}^{Thr}_{k} := \sum_{j \in J} \mathtt{ak}^{Thr}_{j,k} = \sum_{j \in J} \mathsf{RLWE}'_{\boldsymbol{z}_*}\left(\boldsymbol{z}_j(X^k)\right) = \mathsf{RLWE}'_{\boldsymbol{z}_*}\left(\boldsymbol{z}_*(X^k)\right).$$

Generation of $\mathsf{RGSW}_{\boldsymbol{z}_*}(X^{s_{*,i}})$

The difference:

- The sum of components $s_{j,i}$ is done in the exponent.
- The merging is done by RGSW $\circledast$ RGSW multiplications

Generation of $\mathsf{RGSW}_{\boldsymbol{z}_*}(X^{s}_{*,i})$:

- Each participant generates the partial encryption

$$\mathtt{brk}_{j,i}^{Thr} = \mathsf{RGSW}_{\boldsymbol{z}_*}(X^{s_{j,i}})$$

- Then, each party sends $\mathtt{brk}_{j,i}^{Thr}$ to the computing party.
- The computing party calculates $\mathtt{brk}_{i}^{Thr} = \mathsf{RGSW}_{\boldsymbol{z}_*}(X^{s_{*,i}})$:

$$\mathtt{brk}_{i}^{Thr} := \prod_{j \in J} \mathtt{brk}_{j,i}^{Thr} = \prod_{j \in J} \mathsf{RGSW}_{\boldsymbol{z}_*}(X^{s_{j,i}}) = \mathsf{RGSW}_{\boldsymbol{z}_*}(X^{s_{*,i}}).$$

Generation of $\mathsf{RGSW}_{\boldsymbol{z}_*}(X^{s_{*,i}})$

The difference:

- The sum of components $s_{j,i}$ is done in the exponent.
- The merging is done by RGSW $\circledast$ RGSW multiplications

Generation of RGSW$_{\boldsymbol{z}_*}(X^s_{*,i})$:

- Each participant generates the partial encryption

$$\mathtt{brk}_{j,i}^{Thr} = \mathsf{RGSW}_{\boldsymbol{z}_*}(X^{s_{j,i}})$$

- Then, each party sends $\mathtt{brk}_{j,i}^{Thr}$ to the computing party.
- The computing party calculates $\mathtt{brk}_i^{Thr} = \mathsf{RGSW}_{\boldsymbol{z}_*}(X^{s_{*,i}})$:

$$\mathtt{brk}_i^{Thr} := \prod_{j \in J} \mathtt{brk}_{j,i}^{Thr} = \prod_{j \in J} \mathsf{RGSW}_{\boldsymbol{z}_*}(X^{s_{j,i}}) = \mathsf{RGSW}_{\boldsymbol{z}_*}(X^{s_{*,i}}).$$

Generation of $\mathsf{RGSW}_{\boldsymbol{z}_*}(X^{s_{*,i}})$

The difference:

- The sum of components $s_{j,i}$ is done in the exponent.
- The merging is done by RGSW $\circledast$ RGSW multiplications

Generation of $\mathsf{RGSW}_{\boldsymbol{z}_*}(X^{s}_{*,i})$:

- Each participant generates the partial encryption

$$\mathtt{brk}_{j,i}^{Thr} = \mathsf{RGSW}_{\boldsymbol{z}_*}(X^{s_{j,i}})$$

- Then, each party sends $\mathtt{brk}_{j,i}^{Thr}$ to the computing party.
- The computing party calculates $\mathtt{brk}_{i}^{Thr} = \mathsf{RGSW}_{\boldsymbol{z}_*}(X^{s_{*,i}})$:

$$\mathtt{brk}_{i}^{Thr} := \prod_{j \in J} \mathtt{brk}_{j,i}^{Thr} = \prod_{j \in J} \mathsf{RGSW}_{\boldsymbol{z}_*}(X^{s_{j,i}}) = \mathsf{RGSW}_{\boldsymbol{z}_*}(X^{s_{*,i}}).$$

Generation of $\mathsf{RGSW}_{\boldsymbol{z}_*}(X^{s_{*,i}})$

The difference:

- The sum of components $s_{j,i}$ is done in the exponent.
- The merging is done by RGSW $\circledast$ RGSW multiplications

Generation of RGSW$_{\boldsymbol{z}_*}(X^{s}_{*,i})$:

- Each participant generates the partial encryption

$$\mathtt{brk}_{j,i}^{Thr} = \mathsf{RGSW}_{\boldsymbol{z}_*}(X^{s_{j,i}})$$

- Then, each party sends $\mathtt{brk}_{j,i}^{Thr}$ to the computing party.
- The computing party calculates $\mathtt{brk}_{i}^{Thr} = \mathsf{RGSW}_{\boldsymbol{z}_*}(X^{s_{*,i}})$:

$$\mathtt{brk}_{i}^{Thr} := \prod_{j \in J} \mathtt{brk}_{j,i}^{Thr} = \prod_{j \in J} \mathsf{RGSW}_{\boldsymbol{z}_*}(X^{s_{j,i}}) = \mathsf{RGSW}_{\boldsymbol{z}_*}(X^{s_{*,i}}).$$

All Keys for FHEW-like Threshold HE Design

The computing party locates the evaluation keys:

1. $\mathtt{brk}_i^{Thr} = \mathsf{RGSW}_{\boldsymbol{z}_*}\left(X^{s_{*,i}}\right), \quad i \in [0, n-1]$
2. $\mathtt{ak}_u^{Thr} = \mathsf{RLWE}'_{\boldsymbol{z}_*}\left(\boldsymbol{z}_*(X^{g^u})\right), \quad u \in [1, w]$
3. $\mathtt{ak}_{-1}^{Thr} = \mathsf{RLWE}'_{\boldsymbol{z}_*}\left(\boldsymbol{z}_*(X^{-1})\right)$

Conclusion: FHEW-like Threshold HE Design

All Keys for FHEW-like Threshold HE Design

The computing party locates the evaluation keys:

1. $\mathtt{brk}_i^{Thr} = \mathsf{RGSW}_{\boldsymbol{z}_*}\left(X^{s_{*,i}}\right), \quad i \in [0, n-1]$
2. $\mathtt{ak}_u^{Thr} = \mathsf{RLWE}'_{\boldsymbol{z}_*}\left(\boldsymbol{z}_*(X^{g^u})\right), \quad u \in [1, w]$
3. $\mathtt{ak}_{-1}^{Thr} = \mathsf{RLWE}'_{\boldsymbol{z}_*}\left(\boldsymbol{z}_*(X^{-1})\right)$

Conclusion: FHEW-like Threshold HE Design

Outline

Conclusion

A new blind rotation technique

- Offers the best of both previous AP and GINX (further improves on them)
- Several variants which provide tradeoffs between key size and complexity
- Simple threshold HE scheme based on FHEW
  - Takes advantage of the new blind rotation: secret keys wider than ternary

Future work

- Apply it to schemes of other structures such as NTRU and Torus variants
- Batched (or amortized) bootstrapping

Conclusion

A new blind rotation technique

- Offers the best of both previous AP and GINX (further improves on them)
- Several variants which provide tradeoffs between key size and complexity
- Simple threshold HE scheme based on FHEW
  - Takes advantage of the new blind rotation: secret keys wider than ternary

Future work

- Apply it to schemes of other structures such as NTRU and Torus variants
- Batched (or amortized) bootstrapping

Conclusion

A new blind rotation technique

- Offers the best of both previous AP and GINX (further improves on them)
- Several variants which provide tradeoffs between key size and complexity
- Simple threshold HE scheme based on FHEW
  - Takes advantage of the new blind rotation: secret keys wider than ternary

Future work

- Apply it to schemes of other structures such as NTRU and Torus variants
- Batched (or amortized) bootstrapping

Conclusion

A new blind rotation technique

- Offers the best of both previous AP and GINX (further improves on them)
- Several variants which provide tradeoffs between key size and complexity
- Simple threshold HE scheme based on FHEW
  - Takes advantage of the new blind rotation: secret keys wider than ternary

Future work

- Apply it to schemes of other structures such as NTRU and Torus variants
- Batched (or amortized) bootstrapping

Reference I

[AP14] Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In *CRYPTO 2014*, pages 297–314. Springer, 2014.

[BDF18] Guillaume Bonnoron, Léo Ducas, and Max Fillinger. Large FHE gates from tensored homomorphic accumulator. In *Progress in Cryptology – AFRICACRYPT 2018*, pages 217–251. Springer, 2018.

[BIP+22] Charlotte Bonte, Ilia Iliashenko, Jeongeun Park, Hilder V. L. Pereira, and Nigel P. Smart. FINAL: Faster FHE instantiated with NTRU and LWE. In *Advances in Cryptology - ASIACRYPT 2022*, pages 188–215, 2022.

[CGGI17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In *Advances in Cryptology – ASIACRYPT 2017*, pages 377–408. Springer, 2017.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.

Reference II

[DM15] Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *EUROCRYPT 2015*, pages 617–640. Springer, 2015.

[GINX16] Nicolas Gama, Malika Izabachene, Phong Q Nguyen, and Xiang Xie. Structural lattice reduction: Generalized worst-case to average-case reductions and homomorphic cryptosystems. In *EUROCRYPT 2016*, pages 528–558. Springer, 2016.

[JP22] Marc Joye and Pascal Paillier. Blind rotation in fully homomorphic encryption with extended keys. In *International Symposium on Cyber Security, Cryptology, and Machine Learning*, pages 1–18. Springer, 2022.

[KDE+21] Andrey Kim, Maxim Deryabin, Jieun Eom, Rakyong Choi, Yongwoo Lee, Whan Ghang, and Donghoon Yoo. General bootstrapping approach for RLWE-based homomorphic encryption. *Cryptol. ePrint Arch.*, 2021/691, 2021.

Reference III

[LHH$^+$21] Wen-jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. PEGASUS: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. In *2021 IEEE symposium on Security and Privacy (S&P)*, pages 1057–1073. IEEE, 2021.

[LMP22] Zeyu Liu, Daniele Micciancio, and Yuriy Polyakov. Large-precision homomorphic sign evaluation using FHEW/TFHE bootstrapping. In *Advances in Cryptology - ASIACRYPT 2022*, pages 130–160, 2022.

[MP21] Daniele Micciancio and Yuriy Polyakov. Bootstrapping in FHEW-like cryptosystems. In *WAHC'21*, pages 17–28. ACM, 2021.

[TFH] TFHE. Fast fully homomorphic encryption library over the torus. `https://tfhe.github.io/tfhe/`.

Thank You!