Faster Bootstrapping with Polynomial Error

Jacob Alperin-Sheriff   Chris Peikert

School of Computer Science
Georgia Tech

CRYPTO 2014
19 August 2014

Fully Homomorphic Encryption [RAD'78,Gentry'09]

- FHE lets you do this:

$$\boxed{\mu} \longrightarrow \boxed{\mathsf{Eval}(f)} \longrightarrow \boxed{f(\mu)}$$

A cryptographic "holy grail" with countless applications.

First solved in [Gentry'09], followed by

[vDGHV'10,BV'11a,BV'11b,BGV'12,B'12,GSW'13,...]

Fully Homomorphic Encryption [RAD'78,Gentry'09]

- FHE lets you do this:

$$\boxed{\mu} \longrightarrow \boxed{\mathsf{Eval}(f)} \longrightarrow \boxed{f(\mu)}$$

  A cryptographic "holy grail" with countless applications.

  First solved in [Gentry'09], followed by

  [vDGHV'10,BV'11a,BV'11b,BGV'12,B'12,GSW'13,. . .]

- "Naturally occurring" schemes are somewhat homomorphic (SHE): can only evaluate functions of an *a priori* bounded depth.

$$\boxed{\mu} \to \boxed{\mathsf{Eval}(f)} \to \boxed{f(\mu)} \to \boxed{\mathsf{Eval}(g)} \to \boxed{g(f(\mu))}$$

Fully Homomorphic Encryption [RAD'78,Gentry'09]

- FHE lets you do this:

$$\boxed{\mu} \longrightarrow \boxed{\mathsf{Eval}(f)} \longrightarrow \boxed{f(\mu)}$$

  A cryptographic "holy grail" with countless applications.

  First solved in [Gentry'09], followed by

  [vDGHV'10,BV'11a,BV'11b,BGV'12,B'12,GSW'13,...]

- "Naturally occurring" schemes are somewhat homomorphic (SHE): can only evaluate functions of an *a priori* bounded depth.

$$\boxed{\mu} \rightarrow \boxed{\mathsf{Eval}(f)} \rightarrow \boxed{f(\mu)} \rightarrow \boxed{\mathsf{Eval}(g)} \rightarrow \boxed{g(f(\mu))}$$

- Thus far, "bootstrapping" is required to achieve unbounded FHE.

Bootstrapping: SHE $\rightarrow$ FHE [Gentry'09]

- Homomorphically evaluates the SHE decryption function to "refresh" a ciphertext $\boxed{\mu}$, allowing further homomorphic operations.

$$\boxed{sk} \longrightarrow \boxed{\mathsf{Eval}\Big(\ \mathsf{Dec}\big(\cdot\ , \boxed{\mu}\big)\ \Big)} \longrightarrow \boxed{\mu}$$

Bootstrapping: SHE $\rightarrow$ FHE [Gentry'09]

- Homomorphically evaluates the SHE decryption function to "refresh" a ciphertext $\boxed{\mu}$, allowing further homomorphic operations.

$$\boxed{sk} \longrightarrow \boxed{\mathsf{Eval}\Big(\;\mathsf{Dec}\Big(\cdot\,,\boxed{\mu}\Big)\;\Big)} \longrightarrow \boxed{\mu}$$

- Error growth of bootstrapping determines cryptographic assumptions.

Bootstrapping: SHE $\rightarrow$ FHE [Gentry'09]

- Homomorphically evaluates the SHE decryption function to "refresh" a ciphertext $\boxed{\mu}$, allowing further homomorphic operations.

$$\boxed{sk} \longrightarrow \boxed{\mathsf{Eval}\Big(\ \mathsf{Dec}\big(\cdot\,,\boxed{\mu}\big)\ \Big)} \longrightarrow \boxed{\mu}$$

- Error growth of bootstrapping determines cryptographic assumptions. State of the art [BGV'12,B'12,GSW'13]:

Bootstrapping: SHE $\to$ FHE [Gentry'09]

- Homomorphically evaluates the SHE decryption function to "refresh" a ciphertext $\boxed{\mu}$, allowing further homomorphic operations.

$$\boxed{sk} \longrightarrow \boxed{\mathsf{Eval}\Big(\ \mathsf{Dec}\Big(\cdot\ ,\boxed{\mu}\Big)\ \Big)} \longrightarrow \boxed{\mu}$$

- Error growth of bootstrapping determines cryptographic assumptions. State of the art [BGV'12,B'12,GSW'13]:
  - ⋆ **Homom Addition:** Error grows additively.

Bootstrapping: SHE $\rightarrow$ FHE [Gentry'09]

- Homomorphically evaluates the SHE decryption function to "refresh" a ciphertext $\boxed{\mu}$, allowing further homomorphic operations.

$$\boxed{sk} \longrightarrow \boxed{\mathsf{Eval}\Big(\ \mathsf{Dec}\big(\cdot\ ,\boxed{\mu}\big)\ \Big)} \longrightarrow \boxed{\mu}$$

- Error growth of bootstrapping determines cryptographic assumptions. State of the art [BGV'12,B'12,GSW'13]:
  - ⋆ **Homom Addition:** Error grows additively.
  - ⋆ **Homom Multiplication:** Error grows by $\mathrm{poly}(\lambda)$ factor.

Bootstrapping: SHE $\to$ FHE [Gentry'09]

- Homomorphically evaluates the SHE decryption function to "refresh" a ciphertext $\boxed{\mu}$, allowing further homomorphic operations.

$$\boxed{sk} \longrightarrow \boxed{\mathsf{Eval}\Big(\ \mathsf{Dec}\big(\,\cdot\,,\boxed{\mu}\big)\ \Big)} \longrightarrow \boxed{\mu}$$

- Error growth of bootstrapping determines cryptographic assumptions. State of the art [BGV'12,B'12,GSW'13]:
  - ⋆ **Homom Addition:** Error grows additively.
  - ⋆ **Homom Multiplication:** Error grows by $\mathrm{poly}(\lambda)$ factor.

- Known decryption circuits have logarithmic $O(\log \lambda)$ depth.

Bootstrapping: SHE $\to$ FHE [Gentry'09]

- Homomorphically evaluates the SHE decryption function to "refresh" a ciphertext $\boxed{\mu}$, allowing further homomorphic operations.

$$\boxed{sk} \longrightarrow \boxed{\mathsf{Eval}\Big(\,\mathsf{Dec}\big(\cdot\,, \boxed{\mu}\big)\,\Big)} \longrightarrow \boxed{\mu}$$

- Error growth of bootstrapping determines cryptographic assumptions. State of the art [BGV'12,B'12,GSW'13]:
  - ⋆ **Homom Addition:** Error grows additively.
  - ⋆ **Homom Multiplication:** Error grows by $\mathrm{poly}(\lambda)$ factor.

- Known decryption circuits have logarithmic $O(\log \lambda)$ depth.
  $\Longrightarrow$ Quasi-polynomial $\lambda^{O(\log \lambda)}$ error growth and lattice approx factors

Bootstrapping: SHE $\rightarrow$ FHE [Gentry'09]

- Homomorphically evaluates the SHE decryption function to "refresh" a ciphertext $\boxed{\mu}$, allowing further homomorphic operations.

$$\boxed{sk} \longrightarrow \boxed{\mathsf{Eval}\Big(\,\mathsf{Dec}\big(\cdot\,,\boxed{\mu}\big)\Big)} \longrightarrow \boxed{\mu}$$

- Error growth of bootstrapping determines cryptographic assumptions. State of the art [BGV'12,B'12,GSW'13]:
  - ⋆ **Homom Addition:** Error grows additively.
  - ⋆ **Homom Multiplication:** Error grows by $\mathrm{poly}(\lambda)$ factor.

- Known decryption circuits have logarithmic $O(\log \lambda)$ depth.
$\Longrightarrow$ Quasi-polynomial $\lambda^{O(\log \lambda)}$ error growth and lattice approx factors

- Can we do better?

Bootstrapping with Polynomial Error [BrakerskiVaikuntanathan'14]

- Error growth for multiplication in [GSW'13] is asymmetric:

Error in $\mathbf{C} := \mathbf{C}_1 \boxdot \mathbf{C}_2$ is $\mathbf{e} := \mathbf{e}_1 \cdot \mathrm{poly}(\lambda) + \mu_1 \cdot \mathbf{e}_2$.

Bootstrapping with Polynomial Error [BrakerskiVaikuntanathan'14]

- Error growth for multiplication in [GSW'13] is asymmetric:

$$\text{Error in } \mathbf{C} := \mathbf{C}_1 \boxdot \mathbf{C}_2 \text{ is } \mathbf{e} := \mathbf{e}_1 \cdot \mathsf{poly}(\lambda) + \mu_1 \cdot \mathbf{e}_2.$$

- Make multiplication right-associative:

$$\mathbf{C}_1 \boxdot (\cdots (\mathbf{C}_{t-2} \boxdot (\mathbf{C}_{t-1} \boxdot \mathbf{C}_t)) \cdots) \text{ has error } \sum_i \mathbf{e}_i \cdot \mathsf{poly}(\lambda)$$

Bootstrapping with Polynomial Error [BrakerskiVaikuntanathan'14]

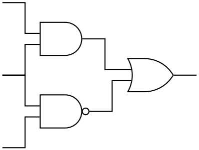
- Error growth for multiplication in [GSW'13] is asymmetric:

$$\text{Error in } \mathbf{C} := \mathbf{C}_1 \boxdot \mathbf{C}_2 \text{ is } \mathbf{e} := \mathbf{e}_1 \cdot \mathsf{poly}(\lambda) + \mu_1 \cdot \mathbf{e}_2.$$

- Make multiplication right-associative:

$$\mathbf{C}_1 \boxdot (\cdots(\mathbf{C}_{t-2} \boxdot (\mathbf{C}_{t-1} \boxdot \mathbf{C}_t))\cdots) \text{ has error } \textstyle\sum_i \mathbf{e}_i \cdot \mathsf{poly}(\lambda)$$

- Barrington's Theorem



$$\Rightarrow \quad \begin{array}{ccccc} (P_{0,1}) & (P_{1,1}) & \cdots & (P_{14,1}) & (P_{15,1}) \\ \\ (P_{0,0}) & (P_{1,0}) & \cdots & (P_{14,0}) & (P_{15,0}) \end{array}$$

depth d

length 4^d

Bootstrapping with Polynomial Error [BrakerskiVaikuntanathan'14]

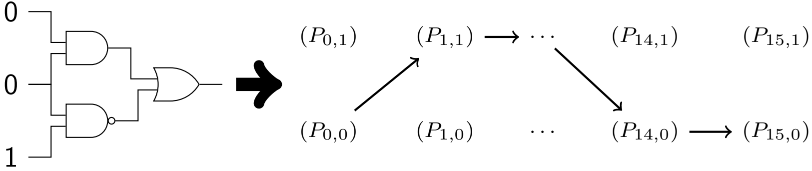
- Error growth for multiplication in [GSW'13] is asymmetric:

$$\text{Error in } \mathbf{C} := \mathbf{C}_1 \boxdot \mathbf{C}_2 \text{ is } \mathbf{e} := \mathbf{e}_1 \cdot \mathrm{poly}(\lambda) + \mu_1 \cdot \mathbf{e}_2.$$

- Make multiplication right-associative:

$$\mathbf{C}_1 \boxdot (\cdots(\mathbf{C}_{t-2} \boxdot (\mathbf{C}_{t-1} \boxdot \mathbf{C}_t))\cdots) \text{ has error } \textstyle\sum_i \mathbf{e}_i \cdot \mathrm{poly}(\lambda)$$

- Barrington's Theorem



depth d

length 4^d

Bootstrapping with Polynomial Error [BrakerskiVaikuntanathan'14]

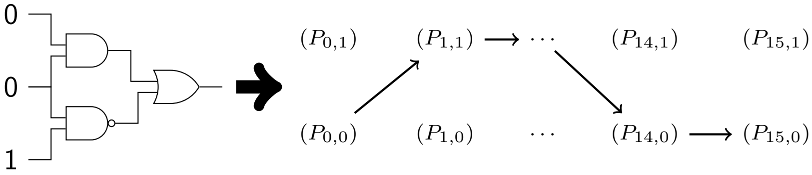
- Error growth for multiplication in [GSW'13] is asymmetric:

$$\text{Error in } \mathbf{C} := \mathbf{C}_1 \boxdot \mathbf{C}_2 \text{ is } \mathbf{e} := \mathbf{e}_1 \cdot \mathsf{poly}(\lambda) + \mu_1 \cdot \mathbf{e}_2.$$

- Make multiplication right-associative:

$$\mathbf{C}_1 \boxdot (\cdots (\mathbf{C}_{t-2} \boxdot (\mathbf{C}_{t-1} \boxdot \mathbf{C}_t)) \cdots) \text{ has error } \textstyle\sum_i \mathbf{e}_i \cdot \mathsf{poly}(\lambda)$$

- Barrington's Theorem



depth $d \approx 3 \log \lambda$

length $4^d \approx \lambda^6$

✗ Problem: Barrington's transformation is very inefficient.

Our Results

1. Faster bootstrapping with small polynomial error growth

Our Results

1. Faster bootstrapping with small polynomial error growth
   - Treats decryption as an arithmetic function over $\mathbb{Z}_q$, not a circuit.

Our Results

1. Faster bootstrapping with small polynomial error growth
   - ⋆ Treats decryption as an arithmetic function over $\mathbb{Z}_q$, not a circuit. Avoids Barrington's Theorem – but still uses permutation matrices!

Our Results

1. Faster bootstrapping with small polynomial error growth
   - Treats decryption as an arithmetic function over $\mathbb{Z}_q$, not a circuit. Avoids Barrington's Theorem – but still uses permutation matrices!
   - Key Idea: Embed additive group $(\mathbb{Z}_q, +)$ into small symmetric group

Our Results

1. Faster bootstrapping with small polynomial error growth
   - ⋆ Treats decryption as an arithmetic function over $\mathbb{Z}_q$, not a circuit. Avoids Barrington's Theorem – but still uses permutation matrices!
   - ⋆ Key Idea: Embed additive group $(\mathbb{Z}_q, +)$ into small symmetric group

Reference	**# Homom Ops**	**Noise Growth**
[GHS'12,AP'13] (packing)	$\tilde{O}(1)$ ✔	$\lambda^{O(\log \lambda)}$
[BV'14]	$\tilde{O}(\lambda^6)$	large $\mathrm{poly}(\lambda)$
This work	$\tilde{O}(\lambda)$ ✔	$\tilde{O}(\lambda^2)$

Our Results

1. Faster bootstrapping with small polynomial error growth
   - ★ Treats decryption as an arithmetic function over $\mathbb{Z}_q$, not a circuit. Avoids Barrington's Theorem – but still uses permutation matrices!
   - ★ Key Idea: Embed additive group $(\mathbb{Z}_q, +)$ into small symmetric group

Reference	# Homom Ops	Noise Growth
[GHS'12,AP'13] (packing)	$\tilde{O}(1)$ ✔	$\lambda^{O(\log \lambda)}$
[BV'14]	$\tilde{O}(\lambda^6)$	large $\mathrm{poly}(\lambda)$
This work	$\tilde{O}(\lambda)$ ✔	$\tilde{O}(\lambda^2)$

2. Variant of [GSW'13] encryption scheme

Our Results

1. Faster bootstrapping with small polynomial error growth
   - ★ Treats decryption as an arithmetic function over $\mathbb{Z}_q$, not a circuit. Avoids Barrington's Theorem – but still uses permutation matrices!
   - ★ Key Idea: Embed additive group $(\mathbb{Z}_q, +)$ into small symmetric group

Reference	# Homom Ops	Noise Growth
[GHS'12,AP'13] (packing)	$\tilde{O}(1)$ ✔	$\lambda^{O(\log \lambda)}$
[BV'14]	$\tilde{O}(\lambda^6)$	large $\mathrm{poly}(\lambda)$
This work	$\tilde{O}(\lambda)$ ✔	$\tilde{O}(\lambda^2)$

2. Variant of [GSW'13] encryption scheme
   - ★ Very simple description and error analysis

Our Results

1. Faster bootstrapping with small polynomial error growth
   - ★ Treats decryption as an arithmetic function over $\mathbb{Z}_q$, not a circuit. Avoids Barrington's Theorem – but still uses permutation matrices!
   - ★ Key Idea: Embed additive group $(\mathbb{Z}_q, +)$ into small symmetric group

Reference	# Homom Ops	Noise Growth
[GHS'12,AP'13] (packing)	$\tilde{O}(1)$ ✔	$\lambda^{O(\log \lambda)}$
[BV'14]	$\tilde{O}(\lambda^6)$	large $\mathrm{poly}(\lambda)$
This work	$\tilde{O}(\lambda)$ ✔	$\tilde{O}(\lambda^2)$

2. Variant of [GSW'13] encryption scheme
   - ★ Very simple description and error analysis
   - ★ Enjoys full re-randomization of error as a natural side effect

     Cf. [BV'14]: partial re-randomization, using extra key material

Simpler GSW Variant

- "Gadget" $\mathbb{Z}_q$-matrix $\mathbf{G}$ [MP'12]: for any $\mathbb{Z}_q$-matrix $\mathbf{A}$,

  $\mathbf{G}^{-1}(\mathbf{A})$ is short and $\mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{A}) = \mathbf{A} \pmod{q}$.

Simpler GSW Variant

- "Gadget" $\mathbb{Z}_q$-matrix $\mathbf{G}$ [MP'12]: for any $\mathbb{Z}_q$-matrix $\mathbf{A}$,

$$\mathbf{G}^{-1}(\mathbf{A}) \text{ is short} \qquad \text{and} \qquad \mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{A}) = \mathbf{A} \pmod q.$$

- Ciphertext encrypting $\mu \in \{0,1\}$ under $\mathbf{s}$ is a $\mathbb{Z}_q$-matrix $\mathbf{C}$ satisfying

$$\mathbf{sC} = \mu \cdot \mathbf{sG} + \mathbf{e} \pmod q.$$

Simpler GSW Variant

- "Gadget" $\mathbb{Z}_q$-matrix $\mathbf{G}$ [MP'12]: for any $\mathbb{Z}_q$-matrix $\mathbf{A}$,

$$\mathbf{G}^{-1}(\mathbf{A}) \text{ is short} \qquad \text{and} \qquad \mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{A}) = \mathbf{A} \pmod{q}.$$

- Ciphertext encrypting $\mu \in \{0,1\}$ under $\mathbf{s}$ is a $\mathbb{Z}_q$-matrix $\mathbf{C}$ satisfying

$$\mathbf{s}\mathbf{C} = \mu \cdot \mathbf{s}\mathbf{G} + \mathbf{e} \pmod{q}.$$

- Homomorphic multiplication: $\mathbf{C}_1 \boxdot \mathbf{C}_2 := \mathbf{C}_1 \cdot \mathbf{G}^{-1}(\mathbf{C}_2)$.

$$\begin{aligned}
\mathbf{s}\mathbf{C}_1 \cdot \mathbf{G}^{-1}(\mathbf{C}_2) &= (\mu_1 \cdot \mathbf{s}\mathbf{G} + \mathbf{e}_1) \cdot \mathbf{G}^{-1}(\mathbf{C}_2) \\
&= \mu_1 \cdot \mathbf{s}\mathbf{C}_2 + \mathbf{e}_1 \cdot \mathbf{G}^{-1}(\mathbf{C}_2) \\
&= \mu_1\mu_2 \cdot \mathbf{s}\mathbf{G} + \underbrace{\mu_1 \cdot \mathbf{e}_2 + \mathbf{e}_1 \cdot \mathbf{G}^{-1}(\mathbf{C}_2)}_{\text{new error}}.
\end{aligned}$$

Simpler GSW Variant

- "Gadget" $\mathbb{Z}_q$-matrix $\mathbf{G}$ [MP'12]: for any $\mathbb{Z}_q$-matrix $\mathbf{A}$,

$$\mathbf{G}^{-1}(\mathbf{A}) \text{ is short} \qquad \text{and} \qquad \mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{A}) = \mathbf{A} \pmod q.$$

- Ciphertext encrypting $\mu \in \{0,1\}$ under $\mathbf{s}$ is a $\mathbb{Z}_q$-matrix $\mathbf{C}$ satisfying

$$\mathbf{sC} = \mu \cdot \mathbf{sG} + \mathbf{e} \pmod q.$$

- Homomorphic multiplication: $\mathbf{C}_1 \boxdot \mathbf{C}_2 := \mathbf{C}_1 \cdot \mathbf{G}^{-1}(\mathbf{C}_2)$.

$$\begin{aligned} \mathbf{sC}_1 \cdot \mathbf{G}^{-1}(\mathbf{C}_2) &= (\mu_1 \cdot \mathbf{sG} + \mathbf{e}_1) \cdot \mathbf{G}^{-1}(\mathbf{C}_2) \\ &= \mu_1 \cdot \mathbf{sC}_2 + \mathbf{e}_1 \cdot \mathbf{G}^{-1}(\mathbf{C}_2) \\ &= \mu_1\mu_2 \cdot \mathbf{sG} + \underbrace{\mu_1 \cdot \mathbf{e}_2 + \mathbf{e}_1 \cdot \mathbf{G}^{-1}(\mathbf{C}_2)}_{\text{new error}}. \end{aligned}$$

- Old method [GSW'13]: $\mathbf{G}^{-1}$ is deterministic bit decomposition.

Simpler GSW Variant

- "Gadget" $\mathbb{Z}_q$-matrix $\mathbf{G}$ [MP'12]: for any $\mathbb{Z}_q$-matrix $\mathbf{A}$,

$$\mathbf{G}^{-1}(\mathbf{A}) \text{ is short} \qquad \text{and} \qquad \mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{A}) = \mathbf{A} \pmod q.$$

- Ciphertext encrypting $\mu \in \{0,1\}$ under $\mathbf{s}$ is a $\mathbb{Z}_q$-matrix $\mathbf{C}$ satisfying

$$\mathbf{sC} = \mu \cdot \mathbf{sG} + \mathbf{e} \pmod q.$$

- Homomorphic multiplication: $\mathbf{C}_1 \boxdot \mathbf{C}_2 := \mathbf{C}_1 \cdot \mathbf{G}^{-1}(\mathbf{C}_2)$.

$$\begin{aligned} \mathbf{sC}_1 \cdot \mathbf{G}^{-1}(\mathbf{C}_2) &= (\mu_1 \cdot \mathbf{sG} + \mathbf{e}_1) \cdot \mathbf{G}^{-1}(\mathbf{C}_2) \\ &= \mu_1 \cdot \mathbf{sC}_2 + \mathbf{e}_1 \cdot \mathbf{G}^{-1}(\mathbf{C}_2) \\ &= \mu_1\mu_2 \cdot \mathbf{sG} + \underbrace{\mu_1 \cdot \mathbf{e}_2 + \mathbf{e}_1 \cdot \mathbf{G}^{-1}(\mathbf{C}_2)}_{\text{new error}}. \end{aligned}$$

- Old method [GSW'13]: $\mathbf{G}^{-1}$ is deterministic bit decomposition.
- New: $\mathbf{G}^{-1}$ samples a (random) subgaussian preimage.
  - $\Rightarrow$ Tight $O(\sqrt{n})$ error growth, full rerandomization of error

Overview of Our Bootstrapping Algorithm

- Decryption in LWE-based schemes can be expressed as

$$\mathsf{Dec}_{\mathbf{s}}(\mathbf{c}) := \lfloor \langle \mathbf{s}, \mathbf{c} \rangle \rceil_2 \in \{0,1\} \text{ with } \mathbf{s} \in \mathbb{Z}_q^n,\ \mathbf{c} \in \{0,1\}^n$$

Overview of Our Bootstrapping Algorithm

- Decryption in LWE-based schemes can be expressed as
$$\mathsf{Dec}_{\mathbf{s}}(\mathbf{c}) := \lfloor \langle \mathbf{s}, \mathbf{c} \rangle \rceil_2 \in \{0,1\} \text{ with } \mathbf{s} \in \mathbb{Z}_q^n,\ \mathbf{c} \in \{0,1\}^n$$

1. Prepare: Encrypt each $s_j \in \mathbb{Z}_q$ under a certain group embedding.

Overview of Our Bootstrapping Algorithm

- Decryption in LWE-based schemes can be expressed as

$$\mathsf{Dec}_{\mathbf{s}}(\mathbf{c}) := \lfloor \langle \mathbf{s}, \mathbf{c} \rangle \rceil_2 \in \{0,1\} \text{ with } \mathbf{s} \in \mathbb{Z}_q^n,\ \mathbf{c} \in \{0,1\}^n$$

1. Prepare: Encrypt each $s_j \in \mathbb{Z}_q$ under a certain group embedding.

Bootstrapping procedure uses two homomorphic algorithms:

$$\boxed{a} \boxplus \boxed{b} = \boxed{a+b} \quad \text{and} \quad \mathsf{Equals}(\boxed{v}, z) = \begin{cases} \boxed{1} & \text{if } v = z \\ \boxed{0} & \text{otherwise} \end{cases}$$

Overview of Our Bootstrapping Algorithm

- Decryption in LWE-based schemes can be expressed as
$$\mathsf{Dec}_{\mathbf{s}}(\mathbf{c}) := \lfloor \langle \mathbf{s}, \mathbf{c} \rangle \rceil_2 \in \{0,1\} \text{ with } \mathbf{s} \in \mathbb{Z}_q^n,\ \mathbf{c} \in \{0,1\}^n$$

1. Prepare: Encrypt each $s_j \in \mathbb{Z}_q$ under a certain group embedding.

Bootstrapping procedure uses two homomorphic algorithms:

$$\boxed{a} \boxplus \boxed{b} = \boxed{a+b} \quad \text{and} \quad \mathsf{Equals}(\boxed{v}, z) = \begin{cases} \boxed{1} & \text{if } v = z \\ \boxed{0} & \text{otherwise} \end{cases}$$

Given ciphertext $\mathbf{c} \in \{0,1\}^n$ and encryptions $\boxed{s_j}$, evaluate:

2. Inner Product: compute $\boxed{v} := \langle \boxed{\mathbf{s}}, \mathbf{c} \rangle = \boxed{\sum}_{j:\, c_j = 1} \boxed{s_j}$

Overview of Our Bootstrapping Algorithm

- Decryption in LWE-based schemes can be expressed as

$$\mathsf{Dec}_{\mathbf{s}}(\mathbf{c}) := \lfloor \langle \mathbf{s}, \mathbf{c} \rangle \rceil_2 \in \{0,1\} \text{ with } \mathbf{s} \in \mathbb{Z}_q^n,\ \mathbf{c} \in \{0,1\}^n$$

1. Prepare: Encrypt each $s_j \in \mathbb{Z}_q$ under a certain group embedding.

Bootstrapping procedure uses two homomorphic algorithms:

$$\boxed{a} \boxplus \boxed{b} = \boxed{a+b} \quad \text{and} \quad \mathsf{Equals}(\boxed{v}, z) = \begin{cases} \boxed{1} & \text{if } v = z \\ \boxed{0} & \text{otherwise} \end{cases}$$

Given ciphertext $\mathbf{c} \in \{0,1\}^n$ and encryptions $\boxed{s_j}$, evaluate:

2. Inner Product: compute $\boxed{v} := \langle \boxed{\mathbf{s}}, \mathbf{c} \rangle = \underset{j:\ c_j = 1}{\boxplus} \boxed{s_j}$

3. Round: compute $\boxed{\lfloor v \rceil_2} := \underset{z:\ \lfloor z \rceil_2 = 1}{\boxplus} \mathsf{Equals}(\boxed{v}, z)$

Overview of Our Bootstrapping Algorithm

- Decryption in LWE-based schemes can be expressed as

$$\mathsf{Dec}_{\mathbf{s}}(\mathbf{c}) := \lfloor \langle \mathbf{s}, \mathbf{c} \rangle \rceil_2 \in \{0,1\} \text{ with } \mathbf{s} \in \mathbb{Z}_q^n,\ \mathbf{c} \in \{0,1\}^n$$

1. Prepare: Encrypt each $s_j \in \mathbb{Z}_q$ under a certain group embedding.

Bootstrapping procedure uses two homomorphic algorithms:

$$\boxed{a} \boxplus \boxed{b} = \boxed{a+b} \quad \text{and} \quad \mathsf{Equals}(\boxed{v}, z) = \begin{cases} \boxed{1} & \text{if } v = z \\ \boxed{0} & \text{otherwise} \end{cases}$$

Given ciphertext $\mathbf{c} \in \{0,1\}^n$ and encryptions $\boxed{s_j}$, evaluate:

2. Inner Product: compute $\boxed{v} := \langle \boxed{\mathbf{s}}, \mathbf{c} \rangle = \underset{j:\ c_j = 1}{\boxplus} \boxed{s_j}$
3. Round: compute $\boxed{\lfloor v \rceil_2} := \underset{z:\ \lfloor z \rceil_2 = 1}{\boxplus} \mathsf{Equals}(\boxed{v}, z)$

- Remains to implement $\boxplus$ and Equals for plaintext space $\mathbb{Z}_q$.

Warmup: Embedding $(\mathbb{Z}_q, +)$ into $(S_q, \cdot)$

$$
\begin{array}{ccccc}
\mathbb{Z}_q & \mathsf{0} & \mathsf{1} & \dots & q-1 \\
S_q &
\begin{pmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{pmatrix} &
\begin{pmatrix} & & & 1 \\ 1 & & & \\ & \ddots & & \\ & & 1 & \end{pmatrix} &
\dots &
\begin{pmatrix} & 1 & & \\ & & \ddots & \\ & & & 1 \\ 1 & & & \end{pmatrix}
\end{array}
$$

Warmup: Embedding $(\mathbb{Z}_q, +)$ into $(S_q, \cdot)$

$$
\begin{array}{ccccc}
\mathbb{Z}_q & 0 & 1 & \dots & q-1 \\
S_q &
\begin{pmatrix} \boxed{1} & & & \\ \boxed{0} & 1 & & \\ \vdots & & \ddots & \\ \boxed{0} & & & 1 \end{pmatrix} &
\begin{pmatrix} \boxed{0} & & & 1 \\ \boxed{1} & & & \\ \vdots & \ddots & & \\ \boxed{0} & & 1 & \end{pmatrix} &
\dots &
\begin{pmatrix} \boxed{0} & 1 & & \\ \vdots & & \ddots & \\ \boxed{0} & & & 1 \\ \boxed{1} & & & \end{pmatrix} \\
 & \boxed{P_0} & \boxed{P_1} & \dots & \boxed{P_{q-1}}
\end{array}
$$

Warmup: Embedding $(\mathbb{Z}_q, +)$ into $(S_q, \cdot)$

$$
\begin{array}{ccccc}
\mathbb{Z}_q & 0 & 1 & \cdots & q-1 \\
S_q &
\begin{pmatrix} \boxed{1} & & & \\ \boxed{0} & 1 & & \\ \vdots & & \ddots & \\ \boxed{0} & & & 1 \end{pmatrix} &
\begin{pmatrix} \boxed{0} & & & 1 \\ \boxed{1} & & & \\ \vdots & \ddots & & \\ \boxed{0} & & 1 & \end{pmatrix} &
\cdots &
\begin{pmatrix} \boxed{0} & 1 & & \\ \vdots & & \ddots & \\ \boxed{0} & & & 1 \\ \boxed{1} & & & \end{pmatrix} \\
& \boxed{P_0} & \boxed{P_1} & \cdots & \boxed{P_{q-1}}
\end{array}
$$

- Addition: $\boxed{a} \boxplus \boxed{b}$ implemented as $\boxed{P_a} \boxdot \boxed{P_b} = \boxed{P_a \cdot P_b}$
  - ⋆ Recall: Right-associative multiplication yields polynomial error growth.

Warmup: Embedding $(\mathbb{Z}_q, +)$ into $(S_q, \cdot)$

$$
\begin{array}{ccccc}
\mathbb{Z}_q & 0 & 1 & \cdots & q-1 \\
S_q &
\begin{pmatrix} \boxed{1} & & & \\ \boxed{0} & 1 & & \\ \vdots & & \ddots & \\ \boxed{0} & & & 1 \end{pmatrix} &
\begin{pmatrix} \boxed{0} & & & 1 \\ \boxed{1} & & & \\ \vdots & \ddots & & \\ \boxed{0} & & 1 & \end{pmatrix} &
\cdots &
\begin{pmatrix} \boxed{0} & 1 & & \\ \vdots & & \ddots & \\ \boxed{0} & & & 1 \\ \boxed{1} & & & \end{pmatrix} \\
& \boxed{P_0} & \boxed{P_1} & \cdots & \boxed{P_{q-1}}
\end{array}
$$

- Addition: $\boxed{a} \boxplus \boxed{b}$ implemented as $\boxed{P_a} \boxdot \boxed{P_b} = \boxed{P_a \cdot P_b}$
  - ⋆ Recall: Right-associative multiplication yields polynomial error growth.
- Equality test: Equals$(\boxed{a}, b)$: take bth entry from first column of $\boxed{P_a}$.

Warmup: Embedding $(\mathbb{Z}_q, +)$ into $(S_q, \cdot)$

$$
\begin{array}{ccccc}
\mathbb{Z}_q & 0 & 1 & \dots & q-1 \\
S_q &
\begin{pmatrix} \boxed{1} & & & \\ \boxed{0} & 1 & & \\ \vdots & & \ddots & \\ \boxed{0} & & & 1 \end{pmatrix} &
\begin{pmatrix} \boxed{0} & & & 1 \\ \boxed{1} & & & \\ \vdots & \ddots & & \\ \boxed{0} & & 1 & \end{pmatrix} &
\dots &
\begin{pmatrix} \boxed{0} & 1 & & \\ \vdots & & \ddots & \\ \boxed{0} & & & 1 \\ \boxed{1} & & & \end{pmatrix} \\
 & \boxed{P_0} & \boxed{P_1} & \dots & \boxed{P_{q-1}}
\end{array}
$$

- Addition: $\boxed{a} \boxplus \boxed{b}$ implemented as $\boxed{P_a} \boxdot \boxed{P_b} = \boxed{P_a \cdot P_b}$
  - ★ Recall: Right-associative multiplication yields polynomial error growth.
- Equality test: Equals$(\boxed{a}, b)$: take bth entry from first column of $\boxed{P_a}$.
- Bottom line: $\tilde{O}(\lambda^3)$ homomorphic operations to bootstrap.

Embedding $(\mathbb{Z}_q, +)$ into Smaller Symmetric Groups

- Let $q = p_1 \cdots p_t = \tilde{O}(\lambda)$ for distinct prime p_i.
  - Prime Number Theorem allows $p_i, t = O(\log \lambda)$.

Embedding $(\mathbb{Z}_q, +)$ into Smaller Symmetric Groups

- Let $q = p_1 \cdots p_t = \tilde{O}(\lambda)$ for distinct prime p_i.
  - ⋆ Prime Number Theorem allows $p_i, t = O(\log \lambda)$.

  Chinese Remainder Theorem: $\mathbb{Z}_q \cong \mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_t}$

Embedding $(\mathbb{Z}_q, +)$ into Smaller Symmetric Groups

- Let $q = p_1 \cdots p_t = \tilde{O}(\lambda)$ for distinct prime p_i.
  - ⋆ Prime Number Theorem allows $p_i, t = O(\log \lambda)$.

  Chinese Remainder Theorem: $\mathbb{Z}_q \cong \mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_t}$
- New embedding:

$$\begin{aligned} \mathbb{Z}_q &\to S_{p_1} \times \cdots \times S_{p_t} \\ x &\mapsto (P_{x \bmod p_1}, \ldots, P_{x \bmod p_t}) \end{aligned}$$

Embedding $(\mathbb{Z}_q, +)$ into Smaller Symmetric Groups

- Let $q = p_1 \cdots p_t = \tilde{O}(\lambda)$ for distinct prime p_i.
  - Prime Number Theorem allows $p_i, t = O(\log \lambda)$.

  Chinese Remainder Theorem: $\mathbb{Z}_q \cong \mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_t}$
- New embedding:

$$\begin{aligned} \mathbb{Z}_q &\to S_{p_1} \times \cdots \times S_{p_t} \\ x &\mapsto (P_{x \bmod p_1}, \ldots, P_{x \bmod p_t}) \end{aligned}$$

- Addition: same as in warmup, but component-wise

Embedding $(\mathbb{Z}_q, +)$ into Smaller Symmetric Groups

- Let $q = p_1 \cdots p_t = \tilde{O}(\lambda)$ for distinct prime p_i.
  - ⋆ Prime Number Theorem allows $p_i, t = O(\log \lambda)$.

  Chinese Remainder Theorem: $\mathbb{Z}_q \cong \mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_t}$
- New embedding:

$$\begin{aligned} \mathbb{Z}_q &\to S_{p_1} \times \cdots \times S_{p_t} \\ x &\mapsto (P_{x \bmod p_1}, \ldots, P_{x \bmod p_t}) \end{aligned}$$

- Addition: same as in warmup, but component-wise
- Equality test:

$$\mathsf{Equals}_q(\boxed{a}, b) = \underset{i}{\boxed{\bullet}} \, \mathsf{Equals}_{p_i}(\boxed{a_i}, b \bmod p_i)$$

Embedding $(\mathbb{Z}_q, +)$ into Smaller Symmetric Groups

- Let $q = p_1 \cdots p_t = \tilde{O}(\lambda)$ for distinct prime p_i.
  - ★ Prime Number Theorem allows $p_i, t = O(\log \lambda)$.

  Chinese Remainder Theorem: $\mathbb{Z}_q \cong \mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_t}$
- New embedding:

$$\begin{aligned} \mathbb{Z}_q &\to S_{p_1} \times \cdots \times S_{p_t} \\ x &\mapsto (P_{x \bmod p_1}, \ldots, P_{x \bmod p_t}) \end{aligned}$$

- Addition: same as in warmup, but component-wise
- Equality test:

$$\mathsf{Equals}_q(\boxed{a}, b) = \underset{i}{\boxed{\bullet}}\, \mathsf{Equals}_{p_i}(\boxed{a_i}, b \bmod p_i)$$

- Bottom line: $\tilde{O}(\lambda)$ homomorphic operations to bootstrap.

Open Problems

- Can we bootstrap in sublinear homom ops with polynomial error?
  - ⋆ Barrier in [GSW'13]: single-bit encryption (no "packing")

Open Problems

- Can we bootstrap in sublinear homom ops with polynomial error?
  - Barrier in [GSW'13]: single-bit encryption (no "packing")
- Circular security for unbounded FHE?
  - Does our representation help or hurt security?

Open Problems

- Can we bootstrap in sublinear homom ops with polynomial error?
  - Barrier in [GSW'13]: single-bit encryption (no "packing")
- Circular security for unbounded FHE?
  - Does our representation help or hurt security?

Thanks!