Bootstrapping (with Small Error Growth)

Chris Peikert

University of Michigan

HEAT Summer School
12 Oct 2015

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:

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

  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.

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

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.

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

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

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

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

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

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

- Decrypting $\boxed{\mu}$ as a function of sk:

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

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

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

- Decrypting $\boxed{\mu}$ as a <u>function of sk</u>:

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

- *Homomorphically* decrypting $\boxed{\mu}$ on $\boxed{sk}$:

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

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

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

- Decrypting $\boxed{\mu}$ as a function of sk:

$$sk \longrightarrow \mathsf{Dec}\left(\cdot\,,\boxed{\mu}\right) \longrightarrow \mu$$

- *Homomorphically* decrypting $\boxed{\mu}$ on $\boxed{sk}$:

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

- Runtime of Eval(Dec) is controlled by complexity of Dec.

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

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

- Decrypting $\boxed{\mu}$ as a <u>function of sk</u>:

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

- *Homomorphically* decrypting $\boxed{\mu}$ on $\boxed{sk}$:

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

- Runtime of Eval(Dec) is controlled by complexity of Dec.
  Error growth of Eval(Dec) determines strength of cryptographic assumption – e.g., initial LWE noise "rate" of $\boxed{sk}$.

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

- *Homomorphic* decryption of $\boxed{\mu}$ on $\boxed{sk}$:

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

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

- *Homomorphic* decryption of $\boxed{\mu}$ on $\boxed{sk}$:

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

- Runtime: quasi-linear $\tilde{O}(\lambda)$ using rings [GHS'12,AP'13]

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

- *Homomorphic* decryption of $\boxed{\mu}$ on $\boxed{sk}$:

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

- Runtime: quasi-linear $\tilde{O}(\lambda)$ using rings [GHS'12,AP'13]
- Error growth using [BGV'12,B'12,GSW'13]:

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

- *Homomorphic* decryption of $\boxed{\mu}$ on $\boxed{sk}$:

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

- Runtime: quasi-linear $\tilde{O}(\lambda)$ using rings [GHS'12,AP'13]
- Error growth using [BGV'12,B'12,GSW'13]:
  - ⋆ **Homom Addition:** Error grows additively.

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

- *Homomorphic* decryption of $\boxed{\mu}$ on $\boxed{sk}$:

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

- Runtime: quasi-linear $\tilde{O}(\lambda)$ using rings [GHS'12,AP'13]
- Error growth using [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]

- *Homomorphic* decryption of $\boxed{\mu}$ on $\boxed{sk}$:

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

- Runtime: quasi-linear $\tilde{O}(\lambda)$ using rings [GHS'12,AP'13]
- Error growth using [BGV'12,B'12,GSW'13]:
  - **Homom Addition:** Error grows additively.
  - **Homom Multiplication:** Error grows by $\mathrm{poly}(\lambda)$ factor.
- Known boolean decryption circuits have logarithmic $O(\log \lambda)$ depth.

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

- *Homomorphic* decryption of $\boxed{\mu}$ on $\boxed{sk}$:

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

- Runtime: quasi-linear $\tilde{O}(\lambda)$ using rings [GHS'12,AP'13]
- Error growth using [BGV'12,B'12,GSW'13]:
  - ⋆ **Homom Addition:** Error grows additively.
  - ⋆ **Homom Multiplication:** Error grows by $\mathrm{poly}(\lambda)$ factor.
- Known boolean decryption circuits have logarithmic $O(\log \lambda)$ depth.

  $\Longrightarrow$ Quasi-polynomial $\lambda^{O(\log \lambda)}$ error growth & lattice approx factors.

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

- *Homomorphic* decryption of $\boxed{\mu}$ on $\boxed{sk}$:

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

- Runtime: quasi-linear $\tilde{O}(\lambda)$ using rings [GHS'12,AP'13]
- Error growth using [BGV'12,B'12,GSW'12]:
  - ⋆ **Homom Addition:** Error grows additively.
  - ⋆ **Homom Multiplication:** Error grows by $\mathrm{poly}(\lambda)$ factor.
- Known boolean decryption circuits have logarithmic $O(\log \lambda)$ depth.
  $\Longrightarrow$ Quasi-polynomial $\lambda^{O(\log \lambda)}$ error growth & lattice approx factors.

Can we do better??

Agenda for the Talk

1. Branching program bootstrapping with (large) polynomial runtime and error growth [BrakerskiVaikuntanathan'14]

Agenda for the Talk

1. Branching program bootstrapping with (large) polynomial runtime and error growth [BrakerskiVaikuntanathan'14]

2. Arithmetic bootstrapping with small polynomial runtime and growth [Alperin-SheriffPeikert'14]

Agenda for the Talk

1. Branching program bootstrapping with (large) polynomial runtime and error growth [BrakerskiVaikuntanathan'14]
2. Arithmetic bootstrapping with small polynomial runtime and growth [Alperin-SheriffPeikert'14]
3. Fast (< 1s) ring-based implementation [DucasMicciancio'15]

Somewhat Homomorphic Encryption [GentrySahaiWaters'13]

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

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

Somewhat Homomorphic Encryption [GentrySahaiWaters'13]

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

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

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

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

Somewhat Homomorphic Encryption [GentrySahaiWaters'13]

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

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

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

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

- Homomorphic add: $\mathbf{C}_1 \boxplus \mathbf{C}_2 := \mathbf{C}_1 + \mathbf{C}_2$.

Somewhat Homomorphic Encryption [GentrySahaiWaters'13]

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

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

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

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

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

Somewhat Homomorphic Encryption [GentrySahaiWaters'13]

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

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

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

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

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

$$\mathbf{s} \cdot \mathbf{C}_1 \cdot \mathbf{G}^{-1}(\mathbf{C}_2) = (\mu_1 \cdot \mathbf{sG} + \mathbf{e}_1) \cdot \mathbf{G}^{-1}(\mathbf{C}_2)$$

Somewhat Homomorphic Encryption [GentrySahaiWaters'13]

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

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

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

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

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

$$\begin{aligned} \mathbf{s} \cdot \mathbf{C}_1 \cdot \mathbf{G}^{-1}(\mathbf{C}_2) &= (\mu_1 \cdot \mathbf{s}\mathbf{G} + \mathbf{e}_1) \cdot \mathbf{G}^{-1}(\mathbf{C}_2) \\ &= \mu_1 \cdot \mathbf{s}\mathbf{C}_2 + \mathbf{e}_1 \cdot \mathbf{G}^{-1}(\mathbf{C}_2) \end{aligned}$$

Somewhat Homomorphic Encryption [GentrySahaiWaters'13]

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

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

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

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

- Homomorphic add: $\mathbf{C}_1 \boxplus \mathbf{C}_2 := \mathbf{C}_1 + \mathbf{C}_2$.

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

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

Somewhat Homomorphic Encryption [GentrySahaiWaters'13]

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

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

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

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

- Homomorphic add: $\mathbf{C}_1 \boxplus \mathbf{C}_2 := \mathbf{C}_1 + \mathbf{C}_2$.

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

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

- (Can randomize $\mathbf{G}^{-1}$ for tighter error growth, full rerandomization.)

Bootstrapping with Polynomial Error [BrakerskiVaikuntanathan'14]

- Error growth for multiplication is asymmetric and "quasi-additive:"

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

Bootstrapping with Polynomial Error [BrakerskiVaikuntanathan'14]

- Error growth for multiplication is asymmetric and "quasi-additive:"

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

- Right-associative multiplication: for $\mathbf{C}_i$ encrypting $\mu_i \in \{0, \pm 1\}$,

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

Bootstrapping with Polynomial Error [BrakerskiVaikuntanathan'14]

- Error growth for multiplication is asymmetric and "quasi-additive:"

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

- Right-associative multiplication: for $\mathbf{C}_i$ encrypting $\mu_i \in \{0, \pm 1\}$,

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

- Generalizes to orthogonal matrices over $\mathbb{Z}$, e.g., permutation matrices. Encrypt bitwise:

$$\underbrace{\begin{pmatrix} \boxed{0} & \boxed{1} \\ \boxed{1} & \boxed{0} \end{pmatrix}}_{\mathbf{P}_1} \boxdot \underbrace{\begin{pmatrix} \boxed{0} & \boxed{1} \\ \boxed{1} & \boxed{0} \end{pmatrix}}_{\mathbf{P}_2} = \underbrace{\begin{pmatrix} \boxed{1} & \boxed{0} \\ \boxed{0} & \boxed{1} \end{pmatrix}}_{\mathbf{P}_1 \cdot \mathbf{P}_2}$$

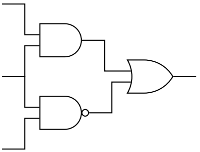
$$\underbrace{\begin{pmatrix} \mathbf{e}_{1,1} & \mathbf{e}_{1,2} \\ \mathbf{e}_{2,1} & \mathbf{e}_{2,2} \end{pmatrix}}_{\mathbf{E}}, \underbrace{\begin{pmatrix} \mathbf{f}_{1,1} & \mathbf{f}_{1,2} \\ \mathbf{f}_{2,1} & \mathbf{f}_{2,2} \end{pmatrix}}_{\mathbf{F}} \to \mathbf{E} \cdot \mathrm{poly}(\lambda) + \underbrace{\begin{pmatrix} \mathbf{f}_{2,1} & \mathbf{f}_{2,2} \\ \mathbf{f}_{1,1} & \mathbf{f}_{1,2} \end{pmatrix}}_{\mathbf{P}_1 \cdot \mathbf{F}}$$

Bootstrapping with Polynomial Error [BrakerskiVaikuntanathan'14]

- Polynomial error growth for any product of encrypted permutations.

Bootstrapping with Polynomial Error [BrakerskiVaikuntanathan'14]

- Polynomial error growth for any product of encrypted permutations.
- Barrington's Theorem: boolean circuit $\rightarrow$ branching program:



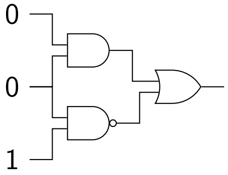
$$\begin{array}{ccccc} \mathbf{P}_{0,1} & \mathbf{P}_{1,1} & \cdots & \mathbf{P}_{14,1} & \mathbf{P}_{15,1} \\ \mathbf{P}_{0,0} & \mathbf{P}_{1,0} & \cdots & \mathbf{P}_{14,0} & \mathbf{P}_{15,0} \end{array}$$

depth d

length 4^d

Bootstrapping with Polynomial Error [BrakerskiVaikuntanathan'14]

- Polynomial error growth for any product of encrypted permutations.
- Barrington's Theorem: boolean circuit $\rightarrow$ branching program:



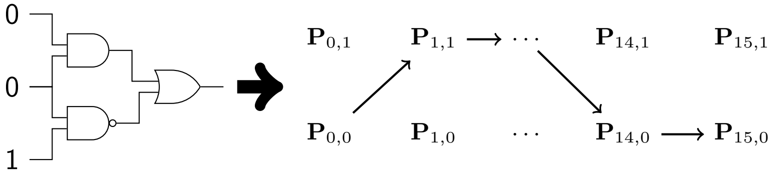
$$\begin{tikzcd}
\mathbf{P}_{0,1} & \mathbf{P}_{1,1} \arrow[r] & \cdots \arrow[dr] & \mathbf{P}_{14,1} & \mathbf{P}_{15,1} \\
\mathbf{P}_{0,0} \arrow[ur] & \mathbf{P}_{1,0} & \cdots & \mathbf{P}_{14,0} \arrow[r] & \mathbf{P}_{15,0}
\end{tikzcd}$$

depth d

length 4^d

Bootstrapping with Polynomial Error [BrakerskiVaikuntanathan'14]

- Polynomial error growth for any product of encrypted permutations.
- Barrington's Theorem: boolean circuit $\to$ branching program:

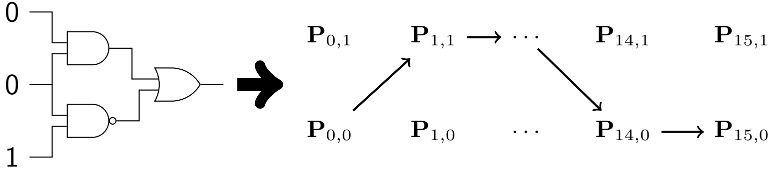


depth d length 4^d

- To refresh $\boxed{\mu}$: convert $\mathrm{Dec}(\cdot, \boxed{\mu})$ to BP; homomorphically evaluate using encrypted bits of sk to select from pairs $\mathbf{P}_{i,0}, \mathbf{P}_{i,1}$.

Bootstrapping with Polynomial Error [BrakerskiVaikuntanathan'14]

- Polynomial error growth for any product of encrypted permutations.
- Barrington's Theorem: boolean circuit $\to$ branching program:



depth $d \approx 3 \log \lambda$ length $4^d \approx \lambda^6$

- To refresh $\boxed{\mu}$: convert Dec$(\cdot, \boxed{\mu})$ to BP; homomorphically evaluate using encrypted bits of sk to select from pairs $\mathbf{P}_{i,0}, \mathbf{P}_{i,1}$.

✗ Drawback: Barrington's transformation is very inefficient.

More Efficient Bootstrapping [Alperin-SheriffPeikert'14]

- Faster algorithm with small polynomial error growth

More Efficient Bootstrapping [Alperin-SheriffPeikert'14]

- Faster algorithm with small polynomial error growth

  Result: quasi-optimal $\tilde{O}(\lambda)$ homom ops; $\tilde{O}(\lambda^2)$ error growth.

More Efficient Bootstrapping [Alperin-SheriffPeikert'14]

- Faster algorithm with small polynomial error growth

  Result: quasi-optimal $\tilde{O}(\lambda)$ homom ops; $\tilde{O}(\lambda^2)$ error growth.

- Treats decryption as an arithmetic function over $\mathbb{Z}_q$, not a circuit.

More Efficient Bootstrapping [Alperin-SheriffPeikert'14]

- Faster algorithm with small polynomial error growth

  Result: quasi-optimal $\tilde{O}(\lambda)$ homom ops; $\tilde{O}(\lambda^2)$ error growth.

- Treats decryption as an arithmetic function over $\mathbb{Z}_q$, not a circuit.

  Avoids Barrington's Theorem – but still uses permutation matrices!

More Efficient Bootstrapping [Alperin-SheriffPeikert'14]

- Faster algorithm with small polynomial error growth

  Result: quasi-optimal $\tilde{O}(\lambda)$ homom ops; $\tilde{O}(\lambda^2)$ 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 a small symmetric group.

Overview of Bootstrapping Algorithm [AP'14]

- Decryption in LWE-based schemes is a "rounded inner product:"

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

Overview of Bootstrapping Algorithm [AP'14]

- Decryption in LWE-based schemes is a "rounded inner product:"

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

1. Prepare: Encrypt each $s_j \in \mathbb{Z}_q$, embedded into a certain group G.

Overview of Bootstrapping Algorithm [AP'14]

- Decryption in LWE-based schemes is a "rounded inner product:"

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

1. **Prepare**: Encrypt each $s_j \in \mathbb{Z}_q$, embedded into a certain group G.

We need two homomorphic algorithms for $\mathbb{Z}_q \subseteq G$:

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

Overview of Bootstrapping Algorithm [AP'14]

- Decryption in LWE-based schemes is a "rounded inner product:"

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

1. Prepare: Encrypt each $s_j \in \mathbb{Z}_q$, embedded into a certain group G.

We need two homomorphic algorithms for $\mathbb{Z}_q \subseteq G$:

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

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

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

Overview of Bootstrapping Algorithm [AP'14]

- Decryption in LWE-based schemes is a "rounded inner product:"

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

1. Prepare: Encrypt each $s_j \in \mathbb{Z}_q$, embedded into a certain group G.

We need two homomorphic algorithms for $\mathbb{Z}_q \subseteq G$:

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

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

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

3. Round: compute $\boxed{\lfloor v \rceil_2} := \boxplus_{z\colon \lfloor z \rceil_2 = 1} \mathsf{Equal?}(\boxed{v}, z)$

Overview of Bootstrapping Algorithm [AP'14]

- Decryption in LWE-based schemes is a "rounded inner product:"

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

1. Prepare: Encrypt each $s_j \in \mathbb{Z}_q$, embedded into a certain group G.

We need two homomorphic algorithms for $\mathbb{Z}_q \subseteq G$:

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

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

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

3. Round: compute $\boxed{\lfloor v \rceil_2} := \boxplus_{z\colon \lfloor z \rceil_2 = 1} \mathsf{Equal?}(\boxed{v}, z)$

- It remains to define the group G and $\boxplus$, $\mathsf{Equal?}$ operations

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

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

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

$$
\begin{array}{ccccc}
\mathbb{Z}_q & 0 & 1 & \dots & q-1 \\
S_q &
\begin{pmatrix} \boxed{1} & & & \\ \boxed{0} & 1 & & \\ \vdots & & \ddots & \\ \boxed{0} & & & 1 \end{pmatrix} &
\begin{pmatrix} \boxed{0} & & & 1 \\ \boxed{1} & & & \\ \vdots & \ddots & & \\ \boxed{0} & & 1 & \end{pmatrix} &
\dots &
\begin{pmatrix} \boxed{0} & 1 & & \\ \vdots & & \ddots & \\ \boxed{0} & & & 1 \\ \boxed{1} & & & \end{pmatrix} \\
& \boxed{\mathbf{P}_0} & \boxed{\mathbf{P}_1} & \dots & \boxed{\mathbf{P}_{q-1}}
\end{array}
$$

- Embed $s \in \mathbb{Z}_q$ as $\mathbf{P}_s$ and encrypt entry-wise (only need first column).

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

$$
\begin{array}{ccccc}
\mathbb{Z}_q & 0 & 1 & \cdots & q-1 \\
S_q &
\begin{pmatrix} \boxed{1} & & & \\ \boxed{0} & 1 & & \\ \vdots & & \ddots & \\ \boxed{0} & & & 1 \end{pmatrix} &
\begin{pmatrix} \boxed{0} & & & 1 \\ \boxed{1} & & & \\ \vdots & \ddots & & \\ \boxed{0} & & 1 & \end{pmatrix} &
\cdots &
\begin{pmatrix} \boxed{0} & 1 & & \\ \vdots & & \ddots & \\ \boxed{0} & & & 1 \\ \boxed{1} & & & \end{pmatrix} \\
& \boxed{\mathbf{P}_0} & \boxed{\mathbf{P}_1} & \cdots & \boxed{\mathbf{P}_{q-1}}
\end{array}
$$

- Embed $s \in \mathbb{Z}_q$ as $\mathbf{P}_s$ and encrypt entry-wise (only need first column).
- Addition: $\boxed{a} \boxplus \boxed{b}$ implemented as $\boxed{\mathbf{P}_a} \boxdot \boxed{\mathbf{P}_b} = \boxed{\mathbf{P}_a \cdot \mathbf{P}_b}$
  - ⋆ Recall: Right-associative multiplication yields polynomial error growth.

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

$$
\begin{array}{ccccc}
\mathbb{Z}_q & 0 & 1 & \dots & q-1 \\
S_q & \begin{pmatrix} \boxed{1} & & & \\ \boxed{0} & 1 & & \\ \vdots & & \ddots & \\ \boxed{0} & & & 1 \end{pmatrix} & \begin{pmatrix} \boxed{0} & & & 1 \\ \boxed{1} & & & \\ \vdots & \ddots & & \\ \boxed{0} & & 1 & \end{pmatrix} & \dots & \begin{pmatrix} \boxed{0} & 1 & & \\ \vdots & & \ddots & \\ \boxed{0} & & & 1 \\ \boxed{1} & & & \end{pmatrix} \\
 & \boxed{\mathbf{P}_0} & \boxed{\mathbf{P}_1} & \dots & \boxed{\mathbf{P}_{q-1}}
\end{array}
$$

- Embed $s \in \mathbb{Z}_q$ as $\mathbf{P}_s$ and encrypt entry-wise (only need first column).
- Addition: $\boxed{a} \boxplus \boxed{b}$ implemented as $\boxed{\mathbf{P}_a} \boxdot \boxed{\mathbf{P}_b} = \boxed{\mathbf{P}_a \cdot \mathbf{P}_b}$
  - ★ Recall: Right-associative multiplication yields polynomial error growth.
- Equality test: Equal?$(\boxed{\mathbf{P}_a}, b)$: output bth entry.

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

$$\begin{array}{ccccc}
\mathbb{Z}_q & 0 & 1 & \dots & q-1 \\
S_q & \begin{pmatrix} \boxed{1} & & & \\ \boxed{0} & 1 & & \\ \vdots & & \ddots & \\ \boxed{0} & & & 1 \end{pmatrix} & \begin{pmatrix} \boxed{0} & & & 1 \\ \boxed{1} & & & \\ \vdots & \ddots & & \\ \boxed{0} & & 1 & \end{pmatrix} & \dots & \begin{pmatrix} \boxed{0} & 1 & & \\ \vdots & & \ddots & \\ \boxed{0} & & & 1 \\ \boxed{1} & & & \end{pmatrix} \\
 & \boxed{\mathbf{P}_0} & \boxed{\mathbf{P}_1} & \dots & \boxed{\mathbf{P}_{q-1}}
\end{array}$$

- Embed $s \in \mathbb{Z}_q$ as $\mathbf{P}_s$ and encrypt entry-wise (only need first column).
- Addition: $\boxed{a} \boxplus \boxed{b}$ implemented as $\boxed{\mathbf{P}_a} \boxdot \boxed{\mathbf{P}_b} = \boxed{\mathbf{P}_a \cdot \mathbf{P}_b}$
  - ⋆ Recall: Right-associative multiplication yields polynomial error growth.
- Equality test: Equal?$(\boxed{\mathbf{P}_a}, b)$: output bth entry.
- Bottom line: $\tilde{O}(\lambda^3)$ homomorphic operations to bootstrap.

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

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

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

- Use $q = p_1 \cdots p_t = \tilde{O}(\lambda)$ 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

- Use $q = p_1 \cdots p_t = \tilde{O}(\lambda)$ 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:

$$\mathbb{Z}_q \to S_{p_1} \times \cdots \times S_{p_t} \quad \left[\subseteq S_{\sum p_i}\right]$$
$$x \mapsto (\mathbf{P}_{x \bmod p_1}, \ldots, \mathbf{P}_{x \bmod p_t})$$

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

- Use $q = p_1 \cdots p_t = \tilde{O}(\lambda)$ 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:

$$\begin{aligned} \mathbb{Z}_q &\to S_{p_1} \times \cdots \times S_{p_t} \quad \left[\subseteq S_{\sum p_i}\right] \\ x &\mapsto (\mathbf{P}_{x \bmod p_1}, \ldots, \mathbf{P}_{x \bmod p_t}) \end{aligned}$$

- Addition $\boxplus$: same as in warmup, but component-wise

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

- Use $q = p_1 \cdots p_t = \tilde{O}(\lambda)$ 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:

$$\begin{gathered} \mathbb{Z}_q \to S_{p_1} \times \cdots \times S_{p_t} \quad \left[\subseteq S_{\sum p_i}\right] \\ x \mapsto (\mathbf{P}_{x \bmod p_1}, \ldots, \mathbf{P}_{x \bmod p_t}) \end{gathered}$$

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

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

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

- Use $q = p_1 \cdots p_t = \tilde{O}(\lambda)$ 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:

$$\begin{gathered} \mathbb{Z}_q \to S_{p_1} \times \cdots \times S_{p_t} \quad \left[\subseteq S_{\sum p_i}\right] \\ x \mapsto (\mathbf{P}_{x \bmod p_1}, \ldots, \mathbf{P}_{x \bmod p_t}) \end{gathered}$$

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

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

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

Refinement and Implementation [DucasMicciancio'15]

- Observation [AP'14]: using ring-LWE in the mth cyclotomic ring R, can work with r-dim orthogonal matrices over R (instead of $\mathbb{Z}$): the generalized symmetric group $\mathbb{Z}_m \wr S_r$.

Refinement and Implementation [DucasMicciancio'15]

- Observation [AP'14]: using ring-LWE in the mth cyclotomic ring R, can work with r-dim orthogonal matrices over R (instead of $\mathbb{Z}$): the generalized symmetric group $\mathbb{Z}_m \wr S_r$.

  In particular, $m = q$ and $r = 1$ yields $\mathbb{Z}_q$.

Refinement and Implementation [DucasMicciancio'15]

- Observation [AP'14]: using ring-LWE in the mth cyclotomic ring R, can work with r-dim orthogonal matrices over R (instead of $\mathbb{Z}$): the generalized symmetric group $\mathbb{Z}_m \wr S_r$.

  In particular, $m = q$ and $r = 1$ yields $\mathbb{Z}_q$.

- With a clever view of NAND as a mod-4 additive threshold, [DM'15] designed a specialized "bootstrapped NAND" procedure.

Refinement and Implementation [DucasMicciancio'15]

- Observation [AP'14]: using ring-LWE in the mth cyclotomic ring R, can work with r-dim orthogonal matrices over R (instead of $\mathbb{Z}$): the generalized symmetric group $\mathbb{Z}_m \wr S_r$.

  In particular, $m = q$ and $r = 1$ yields $\mathbb{Z}_q$.

- With a clever view of NAND as a mod-4 additive threshold, [DM'15] designed a specialized "bootstrapped NAND" procedure.

- FFTW for fast ring operations $\Longrightarrow$ bootstrapping in 0.6 sec: FHEW!

Open Problems

- Can we bootstrap in sublinear # homom ops with polynomial error?
  Bottleneck in [GSW'13]: few plaintext bits / ciphertext (no "packing").

Open Problems

- Can we bootstrap in sublinear # homom ops with polynomial error?

  Bottleneck in [GSW'13]: few plaintext bits / ciphertext (no "packing").

- Circular security for unbounded FHE?

  As usual, unbounded FHE requires a "circular security" assumption: that it is safe to reveal an encryption of (embedded) sk under itself.

  Does our representation of sk help or hurt security?

Open Problems

- Can we bootstrap in sublinear $\#$ homom ops with polynomial error?

  Bottleneck in [GSW'13]: few plaintext bits / ciphertext (no "packing").

- Circular security for unbounded FHE?

  As usual, unbounded FHE requires a "circular security" assumption: that it is safe to reveal an encryption of (embedded) sk under itself.

  Does our representation of sk help or hurt security?

- Can we bootstrap FHS/ABE/PE?

  Current schemes are like "somewhat homomorphic" encryption: they have an *a priori* bound on circuits they can handle.

Open Problems

- Can we bootstrap in sublinear # homom ops with polynomial error?

  Bottleneck in [GSW'13]: few plaintext bits / ciphertext (no "packing").

- Circular security for unbounded FHE?

  As usual, unbounded FHE requires a "circular security" assumption: that it is safe to reveal an encryption of (embedded) sk under itself.

  Does our representation of sk help or hurt security?

- Can we bootstrap FHS/ABE/PE?

  Current schemes are like "somewhat homomorphic" encryption: they have an *a priori* bound on circuits they can handle.

Thanks!