The Learning with Rounding Problem: Reductions and Applications

Alon Rosen
IDC Herzliya

(Thanks: Chris Peikert)

Mysore Park Theory Workshop
August 15, 2013

Pseudorandom Functions [GGM'84]

- A family $\mathcal{F} = \{F_s : \{0,1\}^k \to D\}$ s.t. given adaptive query access,

$$\boxed{F_s \leftarrow \mathcal{F}} \qquad \overset{c}{\approx} \qquad \boxed{\text{random func } U}$$

$$\uparrow x_i \quad \downarrow F_s(x_i) \qquad\qquad \uparrow x_i \quad \downarrow U(x_i)$$

??

(The "seed" or "secret key" for F_s is s.)

(Images courtesy xkcd.org)

Pseudorandom Functions [GGM'84]

- A family $\mathcal{F} = \{F_s : \{0,1\}^k \to D\}$ s.t. given adaptive query access,

$$\boxed{F_s \leftarrow \mathcal{F}} \quad \overset{c}{\approx} \quad \boxed{\text{random func } U}$$

$x_i \uparrow \quad \downarrow F_s(x_i) \qquad\qquad x_i \uparrow \quad \downarrow U(x_i)$

??

(The "seed" or "secret key" for F_s is s.)

- Many applications in symmetric cryptography:
(efficient) encryption, identification, authentication, ...

(Images courtesy xkcd.org)

How to Construct PRFs

1. Heuristically: AES, Blowfish.
   - ✔ Fast!
   - ✔ Withstand <u>known</u> cryptanalytic techniques (linear, differential, . . .)

How to Construct PRFs

1. Heuristically: AES, Blowfish.
   - ✔ Fast!
   - ✔ Withstand known cryptanalytic techniques (linear, differential, . . .)
   - ✗ PRF security is subtle: want provable (reduction) guarantees

How to Construct PRFs

1. Heuristically: AES, Blowfish.
   - ✔ Fast!
   - ✔ Withstand known cryptanalytic techniques (linear, differential, ...)
   - ✗ PRF security is subtle: want provable (reduction) guarantees
2. Goldreich-Goldwasser-Micali [GGM'84]
   - ✔ Based on any (doubling) PRG. $F_s(x_1 \cdots x_k) = G_{x_k}(\cdots G_{x_1}(s) \cdots)$

How to Construct PRFs

1. Heuristically: AES, Blowfish.
   - ✔ Fast!
   - ✔ Withstand known cryptanalytic techniques (linear, differential, ...)
   - ✗ PRF security is subtle: want provable (reduction) guarantees

2. Goldreich-Goldwasser-Micali [GGM'84]
   - ✔ Based on any (doubling) PRG. $F_s(x_1 \cdots x_k) = G_{x_k}(\cdots G_{x_1}(s) \cdots)$
   - ✗ Inherently sequential: $\geq k$ iterations (circuit depth)

How to Construct PRFs

1. Heuristically: AES, Blowfish.
   - ✔ Fast!
   - ✔ Withstand known cryptanalytic techniques (linear, differential, ...)
   - ✗ PRF security is subtle: want provable (reduction) guarantees
2. Goldreich-Goldwasser-Micali [GGM'84]
   - ✔ Based on any (doubling) PRG. $F_s(x_1 \cdots x_k) = G_{x_k}(\cdots G_{x_1}(s) \cdots)$
   - ✗ Inherently sequential: $\geq k$ iterations (circuit depth)
3. Naor-Reingold [NR'95,NR'97,NRR'00]
   - ✔ Based on "synthesizers" or number theory (DDH, factoring)
   - ✔ Low-depth: NC^2, NC^1 or even TC^0 [$O(1)$ depth w/ threshold gates]

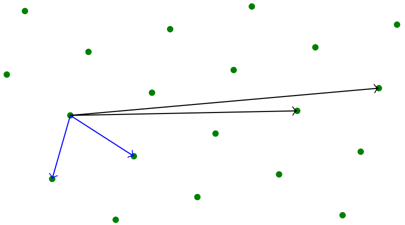
How to Construct PRFs

1. Heuristically: AES, Blowfish.
   - ✔ Fast!
   - ✔ Withstand known cryptanalytic techniques (linear, differential, ...)
   - ✗ PRF security is subtle: want provable (reduction) guarantees

2. Goldreich-Goldwasser-Micali [GGM'84]
   - ✔ Based on any (doubling) PRG. $F_s(x_1 \cdots x_k) = G_{x_k}(\cdots G_{x_1}(s) \cdots)$
   - ✗ Inherently sequential: $\geq k$ iterations (circuit depth)

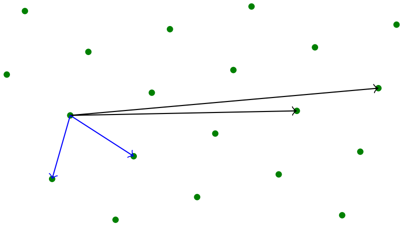
3. Naor-Reingold [NR'95,NR'97,NRR'00]
   - ✔ Based on "synthesizers" or number theory (DDH, factoring)
   - ✔ Low-depth: NC^2, NC^1 or even TC^0 [$O(1)$ depth w/ threshold gates]
   - ✗ Large circuits that need much preprocessing
   - ✗ No "post-quantum" construction under standard assumptions

Why Not Try Lattices?



$$\overset{??}{\Longrightarrow} \quad \boxed{F_s \leftarrow \mathcal{F}}$$

Why Not Try Lattices?

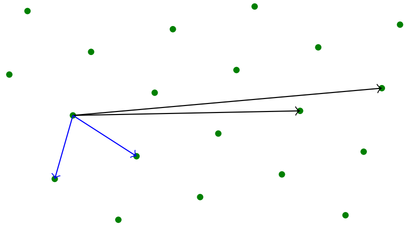


$\xRightarrow{??}$ $\boxed{F_s \leftarrow \mathcal{F}}$

Advantages of Lattice Crypto Schemes

- Simple & efficient: linear, highly parallel operations
- Resist quantum attacks (so far)
- Secure under worst-case hardness assumptions [Ajtai'96,. . .]

Why Not Try Lattices?



$$\stackrel{??}{\Longrightarrow} \quad \boxed{F_s \leftarrow \mathcal{F}}$$

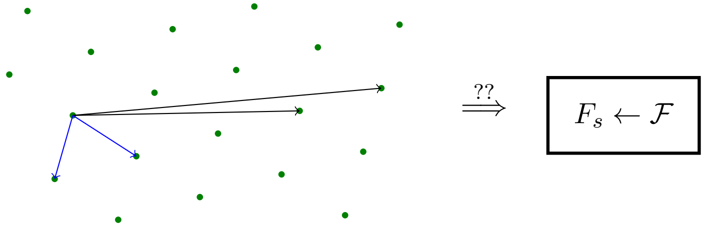
Advantages of Lattice Crypto Schemes

- Simple & efficient: linear, highly parallel operations
- Resist quantum attacks (so far)
- Secure under worst-case hardness assumptions [Ajtai'96,...]

Disadvantages

✗ Only known PRF is generic GGM (not parallel or efficient)

Why Not Try Lattices?



Advantages of Lattice Crypto Schemes

- Simple & efficient: linear, highly parallel operations
- Resist quantum attacks (so far)
- Secure under worst-case hardness assumptions [Ajtai'96,...]

Disadvantages

✗ Only known PRF is generic GGM (not parallel or efficient)

✗✗ We don't even have practical PRGs from lattices: biased errors

PRFs From Lattices [Banerjee, Peikert, Rosen'12]

1. Low-depth, relatively small-circuit PRFs from lattices / (ring-)LWE

PRFs From Lattices [Banerjee, Peikert, Rosen'12]

1. Low-depth, relatively small-circuit PRFs from lattices / (ring-)LWE
   - Synthesizer-based PRF in $\mathsf{TC}^1 \subseteq \mathsf{NC}^2$ *a la* [NR'95]
   - Direct construction in $\mathsf{TC}^0 \subseteq \mathsf{NC}^1$ analogous to [NR'97,NRR'00]

PRFs From Lattices [Banerjee, Peikert, Rosen'12]

1. Low-depth, relatively small-circuit PRFs from lattices / (ring-)LWE
   - ⋆ Synthesizer-based PRF in $\mathsf{TC}^1 \subseteq \mathsf{NC}^2$ *a la* [NR'95]
   - ⋆ Direct construction in $\mathsf{TC}^0 \subseteq \mathsf{NC}^1$ analogous to [NR'97,NRR'00]
2. Main technique: Learning With Rounding (LWR)

   "derandomization" of LWE: deterministic errors

PRFs From Lattices [Banerjee, Peikert, Rosen'12]

1. Low-depth, relatively small-circuit PRFs from lattices / (ring-)LWE
   - ⋆ Synthesizer-based PRF in $\mathsf{TC}^1 \subseteq \mathsf{NC}^2$ *a la* [NR'95]
   - ⋆ Direct construction in $\mathsf{TC}^0 \subseteq \mathsf{NC}^1$ analogous to [NR'97,NRR'00]

2. Main technique: Learning With Rounding (LWR)

   "derandomization" of LWE: deterministic errors

   Also gives more practical PRGs, GGM-type PRFs, encryption, . . .

Synthesizers and PRFs [NaorReingold'95]

Synthesizer

- A deterministic function $S\colon D \times D \to D$ s.t. for any $m = \text{poly}$:
  for $a_1, \ldots, a_m,\ b_1, \ldots, b_m \leftarrow D$,

$$\{\, S(a_i\,,\, b_j)\,\} \overset{c}{\approx} \text{Unif}(D^{m \times m}).$$

Synthesizers and PRFs [NaorReingold'95]

Synthesizer

- A deterministic function $S \colon D \times D \to D$ s.t. for <u>any</u> $m =$ poly: for $a_1, \ldots, a_m,\ b_1, \ldots, b_m \leftarrow D$,

$$\{\, S(a_i\,,\, b_j) \,\} \overset{c}{\approx} \mathsf{Unif}(D^{m \times m}).$$

$$\begin{array}{c||ccc}
 & b_1 & b_2 & \cdots \\
\hline\hline
a_1 & S(a_1, b_1) & S(a_1, b_2) & \cdots \\
a_2 & S(a_2, b_1) & S(a_2, b_2) & \cdots \\
\vdots & & \ddots &
\end{array}
\quad \text{vs.} \quad
\begin{array}{c||ccc}
 & & & \\
\hline\hline
 & U_{1,1} & U_{1,2} & \cdots \\
 & U_{2,1} & U_{2,2} & \cdots \\
 & & \ddots &
\end{array}$$

Synthesizers and PRFs [NaorReingold'95]

Synthesizer

- A deterministic function $S \colon D \times D \to D$ s.t. for any $m = \text{poly}$: for $a_1, \ldots, a_m,\ b_1, \ldots, b_m \leftarrow D$,

$$\{\, S(a_i\,,\, b_j) \,\} \overset{c}{\approx} \mathsf{Unif}(D^{m \times m}).$$

$$\begin{array}{c||ccc}
 & b_1 & b_2 & \cdots \\
\hline\hline
a_1 & S(a_1, b_1) & S(a_1, b_2) & \cdots \\
a_2 & S(a_2, b_1) & S(a_2, b_2) & \cdots \\
\vdots & & \ddots &
\end{array}
\quad \text{vs.} \quad
\begin{array}{c||ccc}
 & & & \\
\hline\hline
 & U_{1,1} & U_{1,2} & \cdots \\
 & U_{2,1} & U_{2,2} & \cdots \\
 & & \ddots &
\end{array}$$

- Alternative view: an (almost) length-squaring PRG with locality: maps $D^{2m} \to D^{m^2}$, and each output depends on only 2 inputs.

Synthesizers and PRFs [NaorReingold'95]

PRF from Synthesizer, Recursively

- Synthesizer $S \colon D \times D \to D$, where $\{ S(a_i , b_j) \} \stackrel{c}{\approx}$ Unif$(D^{m \times m})$.

Synthesizers and PRFs [NaorReingold'95]

PRF from Synthesizer, Recursively

- Synthesizer $S\colon D \times D \to D$, where $\{\, S(a_i\,,\, b_j) \,\} \overset{c}{\approx}$ Unif$(D^{m \times m})$.
- Base case: "one-bit" PRF $F_{s_0,s_1}(x) := s_x \in D$. ✔

Synthesizers and PRFs [NaorReingold'95]

PRF from Synthesizer, Recursively

- Synthesizer $S \colon D \times D \to D$, where $\{\, S(a_i\,,\, b_j)\,\} \overset{c}{\approx}$ Unif$(D^{m \times m})$.
- Base case: "one-bit" PRF $F_{s_0,s_1}(x) := s_x \in D$. ✔
- Input doubling: given k-bit PRF family $\mathcal{F} = \{F \colon \{0,1\}^k \to D\}$, define a $\{0,1\}^{2k} \to D$ function with seed $F_\ell, F_r \leftarrow \mathcal{F}$:

$$F_{(F_\ell, F_r)}(x_\ell\,,\, x_r) = S\big(\, F_\ell(x_\ell)\,,\, F_r(x_r)\,\big).$$

Synthesizers and PRFs [NaorReingold'95]

PRF from Synthesizer, Recursively

- Synthesizer $S \colon D \times D \to D$, where $\{\, S(a_i\,,\, b_j)\,\} \overset{c}{\approx} \mathsf{Unif}(D^{m \times m})$.
- <u>Base case</u>: "one-bit" PRF $F_{s_0,s_1}(x) := s_x \in D$. ✔
- <u>Input doubling</u>: given k-bit PRF family $\mathcal{F} = \{F \colon \{0,1\}^k \to D\}$, define a $\{0,1\}^{2k} \to D$ function with seed $F_\ell, F_r \leftarrow \mathcal{F}$:

$$F_{(F_\ell,F_r)}(x_\ell\,,\,x_r) = S\big(\,F_\ell(x_\ell)\,,\,F_r(x_r)\,\big).$$

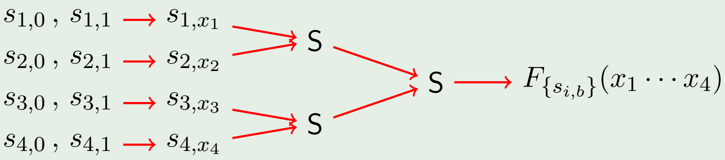
$$\begin{tikzcd}
s_{1,0}\,,\,s_{1,1} \arrow[r] & s_{1,x_1} \arrow[dr] & & & \\
& & \mathsf{S} \arrow[ddr] & & \\
s_{2,0}\,,\,s_{2,1} \arrow[r] & s_{2,x_2} \arrow[ur] & & & \\
& & & \mathsf{S} \arrow[r] & F_{\{s_{i,b}\}}(x_1 \cdots x_4) \\
s_{3,0}\,,\,s_{3,1} \arrow[r] & s_{3,x_3} \arrow[dr] & & & \\
& & \mathsf{S} \arrow[uur] & & \\
s_{4,0}\,,\,s_{4,1} \arrow[r] & s_{4,x_4} \arrow[ur] & & &
\end{tikzcd}$$

Synthesizers and PRFs [NaorReingold'95]

PRF from Synthesizer, Recursively

- Synthesizer $S \colon D \times D \to D$, where $\{\, S(a_i\,,\, b_j)\,\} \overset{c}{\approx}$ Unif$(D^{m \times m})$.
- Base case: "one-bit" PRF $F_{s_0,s_1}(x) := s_x \in D$. ✔
- Input doubling: given k-bit PRF family $\mathcal{F} = \{F \colon \{0,1\}^k \to D\}$, define a $\{0,1\}^{2k} \to D$ function with seed $F_\ell, F_r \leftarrow \mathcal{F}$:

$$F_{(F_\ell,F_r)}(x_\ell\,,\, x_r) = S\big(\, F_\ell(x_\ell)\,,\, F_r(x_r)\,\big).$$



- Security: the queries $F_\ell(x_\ell)$ and $F_r(x_r)$ define (pseudo)random inputs $a_1, a_2, \ldots \in D$ and $b_1, b_2, \ldots \in D$ to synthesizer S.

Learning With Errors [Regev'05]

- Dimension n (security param), modulus $q \geq 2$

Learning With Errors [Regev'05]

- Dimension n (security param), modulus $q \geq 2$
- **Search:** find $\mathbf{s} \in \mathbb{Z}_q^n$ given 'noisy random inner products'

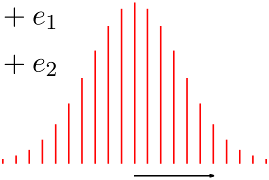
$$\mathbf{a}_1 \leftarrow \mathbb{Z}_q^n \ , \ b_1 = \langle \mathbf{s} \ , \ \mathbf{a}_1 \rangle + e_1$$
$$\mathbf{a}_2 \leftarrow \mathbb{Z}_q^n \ , \ b_2 = \langle \mathbf{s} \ , \ \mathbf{a}_2 \rangle + e_2$$
$$\vdots$$

Learning With Errors [Regev'05]

- Dimension n (security param), modulus $q \geq 2$, 'error rate' $\alpha \ll 1$
- **Search:** find $\mathbf{s} \in \mathbb{Z}_q^n$ given 'noisy random inner products'

$$\mathbf{a}_1 \leftarrow \mathbb{Z}_q^n \ , \ b_1 = \langle \mathbf{s} \ , \ \mathbf{a}_1 \rangle + e_1$$
$$\mathbf{a}_2 \leftarrow \mathbb{Z}_q^n \ , \ b_2 = \langle \mathbf{s} \ , \ \mathbf{a}_2 \rangle + e_2$$
$$\vdots$$

Errors $e_i \leftarrow \chi =$ Gaussian over $\mathbb{Z}$, param αq



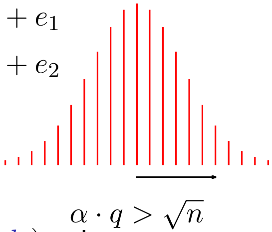
$\alpha \cdot q > \sqrt{n}$

Learning With Errors [Regev'05]

- Dimension n (security param), modulus $q \geq 2$, 'error rate' $\alpha \ll 1$
- **Search:** find $\mathbf{s} \in \mathbb{Z}_q^n$ given 'noisy random inner products'

$$\mathbf{a}_1 \leftarrow \mathbb{Z}_q^n \ , \ b_1 = \langle \mathbf{s} \ , \ \mathbf{a}_1 \rangle + e_1$$
$$\mathbf{a}_2 \leftarrow \mathbb{Z}_q^n \ , \ b_2 = \langle \mathbf{s} \ , \ \mathbf{a}_2 \rangle + e_2$$
$$\vdots$$

Errors $e_i \leftarrow \chi =$ Gaussian over $\mathbb{Z}$, param αq

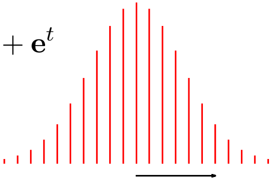


- **Decision:** distinguish $(\mathbf{a}_i, b_i)$ from uniform $(\mathbf{a}_i, b_i)$ pairs

Learning With Errors [Regev'05]

- Dimension n (security param), modulus $q \geq 2$, 'error rate' $\alpha \ll 1$
- **Search:** find $\mathbf{s} \in \mathbb{Z}_q^n$ given 'noisy random inner products'

$$\mathbf{A} = \begin{pmatrix} | & & | \\ \mathbf{a}_1 & \cdots & \mathbf{a}_m \\ | & & | \end{pmatrix}, \quad \mathbf{b}^t = \mathbf{s}^t\mathbf{A} + \mathbf{e}^t$$



Errors $e_i \leftarrow \chi =$ Gaussian over $\mathbb{Z}$, param αq

$\alpha \cdot q > \sqrt{n}$

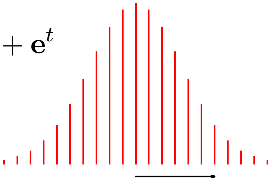
- **Decision:** distinguish $(\mathbf{a}_i, b_i)$ from uniform $(\mathbf{a}_i, b_i)$ pairs

Learning With Errors [Regev'05]

- Dimension n (security param), modulus $q \geq 2$, 'error rate' $\alpha \ll 1$
- **Search:** find $\mathbf{s} \in \mathbb{Z}_q^n$ given 'noisy random inner products'

$$\mathbf{A} = \begin{pmatrix} | & & | \\ \mathbf{a}_1 & \cdots & \mathbf{a}_m \\ | & & | \end{pmatrix}, \quad \mathbf{b}^t = \mathbf{s}^t \mathbf{A} + \mathbf{e}^t$$

Errors $e_i \leftarrow \chi =$ Gaussian over $\mathbb{Z}$, param αq



$\alpha \cdot q > \sqrt{n}$

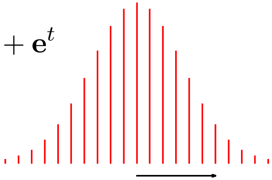
- **Decision:** distinguish $(\mathbf{a}_i, b_i)$ from uniform $(\mathbf{a}_i, b_i)$ pairs

Generalizes LPN ($q = 2$, Bernoulli noise) [AL'88,BFKL'94,...]

Learning With Errors [Regev'05]

- Dimension n (security param), modulus $q \geq 2$, 'error rate' $\alpha \ll 1$
- **Search:** find $\mathbf{s} \in \mathbb{Z}_q^n$ given 'noisy random inner products'

$$\mathbf{A} = \begin{pmatrix} | & & | \\ \mathbf{a}_1 & \cdots & \mathbf{a}_m \\ | & & | \end{pmatrix}, \quad \mathbf{b}^t = \mathbf{s}^t \mathbf{A} + \mathbf{e}^t$$



Errors $e_i \leftarrow \chi =$ Gaussian over $\mathbb{Z}$, param αq

$\alpha \cdot q > \sqrt{n}$

- **Decision:** distinguish $(\mathbf{a}_i, b_i)$ from uniform $(\mathbf{a}_i, b_i)$ pairs

  Generalizes LPN ($q = 2$, Bernoulli noise) [AL'88,BFKL'94,...]

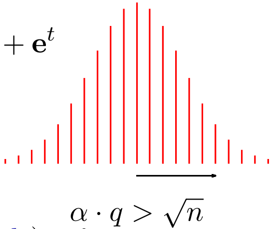
- Why error $\alpha q > \sqrt{n}$?
  - ⋆ Required by worst-case hardness proofs [R'05,P'09,MP'12,BLPRS'13]

Learning With Errors [Regev'05]

- Dimension n (security param), modulus $q \geq 2$, 'error rate' $\alpha \ll 1$
- **Search:** find $\mathbf{s} \in \mathbb{Z}_q^n$ given 'noisy random inner products'

$$\mathbf{A} = \begin{pmatrix} | & & | \\ \mathbf{a}_1 & \cdots & \mathbf{a}_m \\ | & & | \end{pmatrix}, \quad \mathbf{b}^t = \mathbf{s}^t \mathbf{A} + \mathbf{e}^t$$

Errors $e_i \leftarrow \chi =$ Gaussian over $\mathbb{Z}$, param αq



- **Decision:** distinguish $(\mathbf{a}_i, b_i)$ from uniform $(\mathbf{a}_i, b_i)$ pairs

  Generalizes LPN ($q = 2$, Bernoulli noise) [AL'88,BFKL'94,...]

- Why error $\alpha q > \sqrt{n}$?
  - ⋆ Required by worst-case hardness proofs [R'05,P'09,MP'12,BLPRS'13]
  - ⋆ There's an $\exp((\alpha q)^2)$-time attack! [AG'11]

Simple Properties of LWE

1. Check a candidate solution $\mathbf{s}' \in \mathbb{Z}_q^n$:

Simple Properties of LWE

1. Check a candidate solution $\mathbf{s}' \in \mathbb{Z}_q^n$: test if all $b - \langle \mathbf{s}', \mathbf{a} \rangle$ 'small.'

Simple Properties of LWE

1. Check a candidate solution $\mathbf{s}' \in \mathbb{Z}_q^n$: test if all $b - \langle \mathbf{s}', \mathbf{a} \rangle$ 'small.'

   If $\mathbf{s}' \neq \mathbf{s}$, then $b - \langle \mathbf{s}', \mathbf{a} \rangle = \langle \mathbf{s} - \mathbf{s}', \mathbf{a} \rangle + e$ is 'well-spread' in $\mathbb{Z}_q$.

Simple Properties of LWE

1. Check a candidate solution $\mathbf{s}' \in \mathbb{Z}_q^n$: test if all $b - \langle \mathbf{s}', \mathbf{a} \rangle$ 'small.'

   If $\mathbf{s}' \neq \mathbf{s}$, then $b - \langle \mathbf{s}', \mathbf{a} \rangle = \langle \mathbf{s} - \mathbf{s}', \mathbf{a} \rangle + e$ is 'well-spread' in $\mathbb{Z}_q$.

2. 'Shift' the secret by any $\mathbf{t} \in \mathbb{Z}_q^n$:

Simple Properties of LWE

1. Check a candidate solution $\mathbf{s}' \in \mathbb{Z}_q^n$: test if all $b - \langle \mathbf{s}', \mathbf{a} \rangle$ 'small.'

   If $\mathbf{s}' \neq \mathbf{s}$, then $b - \langle \mathbf{s}', \mathbf{a} \rangle = \langle \mathbf{s} - \mathbf{s}', \mathbf{a} \rangle + e$ is 'well-spread' in $\mathbb{Z}_q$.

2. 'Shift' the secret by any $\mathbf{t} \in \mathbb{Z}_q^n$: given $(\mathbf{a}, b = \langle \mathbf{s}, \mathbf{a} \rangle + e)$, output

$$\begin{aligned} \mathbf{a}\ ,\ b' &= b + \langle \mathbf{t}, \mathbf{a} \rangle \\ &= \langle \mathbf{s} + \mathbf{t}, \mathbf{a} \rangle + e. \end{aligned}$$

Simple Properties of LWE

1. Check a candidate solution $\mathbf{s}' \in \mathbb{Z}_q^n$: test if all $b - \langle \mathbf{s}', \mathbf{a} \rangle$ 'small.'

   If $\mathbf{s}' \neq \mathbf{s}$, then $b - \langle \mathbf{s}', \mathbf{a} \rangle = \langle \mathbf{s} - \mathbf{s}', \mathbf{a} \rangle + e$ is 'well-spread' in $\mathbb{Z}_q$.

2. 'Shift' the secret by any $\mathbf{t} \in \mathbb{Z}_q^n$: given $(\mathbf{a}, b = \langle \mathbf{s}, \mathbf{a} \rangle + e)$, output

$$\begin{aligned} \mathbf{a}\ ,\ b' &= b + \langle \mathbf{t}, \mathbf{a} \rangle \\ &= \langle \mathbf{s} + \mathbf{t}, \mathbf{a} \rangle + e. \end{aligned}$$

   Random $\mathbf{t}$'s (with fresh samples) $\Rightarrow$ random self-reduction.

   Lets us amplify success probabilities (both search & decision):

   non-negl on uniform $\mathbf{s} \leftarrow \mathbb{Z}_q^n \quad \Longrightarrow \quad \approx 1$ on <u>any</u> $\mathbf{s} \in \mathbb{Z}_q^n$

Simple Properties of LWE

1. Check a candidate solution $\mathbf{s}' \in \mathbb{Z}_q^n$: test if all $b - \langle \mathbf{s}', \mathbf{a} \rangle$ 'small.'

   If $\mathbf{s}' \neq \mathbf{s}$, then $b - \langle \mathbf{s}', \mathbf{a} \rangle = \langle \mathbf{s} - \mathbf{s}', \mathbf{a} \rangle + e$ is 'well-spread' in $\mathbb{Z}_q$.

2. 'Shift' the secret by any $\mathbf{t} \in \mathbb{Z}_q^n$: given $(\mathbf{a}, b = \langle \mathbf{s}, \mathbf{a} \rangle + e)$, output

$$\begin{aligned} \mathbf{a} \ , \ b' &= b + \langle \mathbf{t}, \mathbf{a} \rangle \\ &= \langle \mathbf{s} + \mathbf{t}, \mathbf{a} \rangle + e. \end{aligned}$$

   Random $\mathbf{t}$'s (with fresh samples) $\Rightarrow$ random self-reduction.

   Lets us amplify success probabilities (both search & decision):

   non-negl on uniform $\mathbf{s} \leftarrow \mathbb{Z}_q^n \quad \Longrightarrow \quad \approx 1$ on <u>any</u> $\mathbf{s} \in \mathbb{Z}_q^n$

3. Multiple secrets: $(\mathbf{a}, b_1 \approx \langle \mathbf{s}_1, \mathbf{a} \rangle, \ldots, b_t \approx \langle \mathbf{s}_t, \mathbf{a} \rangle)$ vs. $(\mathbf{a}, b_1, \ldots, b_t)$.

   Simple hybrid argument, since $\mathbf{a}$'s are *public*.

Search/Decision Equivalence [BFKL'94,R'05]

- Suppose $\mathcal{D}$ solves decision-LWE: it 'perfectly' distinguishes between pairs $(\mathbf{a}, b = \langle \mathbf{s}, \mathbf{a} \rangle + e)$ and $(\mathbf{a}, b)$.

Search/Decision Equivalence [BFKL'94,R'05]

- Suppose $\mathcal{D}$ solves decision-LWE: it 'perfectly' distinguishes between pairs $(\mathbf{a}, b = \langle \mathbf{s}, \mathbf{a} \rangle + e)$ and $(\mathbf{a}, b)$.

  We want to solve search-LWE: given pairs $(\mathbf{a}, b)$, find $\mathbf{s}$.

Search/Decision Equivalence [BFKL'94,R'05]

- Suppose $\mathcal{D}$ solves decision-LWE: it 'perfectly' distinguishes between pairs $(\mathbf{a}, b = \langle \mathbf{s}, \mathbf{a} \rangle + e)$ and $(\mathbf{a}, b)$.

  We want to solve search-LWE: given pairs $(\mathbf{a}, b)$, find $\mathbf{s}$.

- If $\boxed{q = \text{poly}(n)}$, to find $s_1 \in \mathbb{Z}_q$ it suffices to test whether $s_1 \stackrel{?}{=} 0$, because we can shift s_1 by $0, 1, \ldots, q-1$. Same for $s_2, s_3, \ldots, s_n$.

Search/Decision Equivalence [BFKL'94,R'05]

- Suppose $\mathcal{D}$ solves decision-LWE: it 'perfectly' distinguishes between pairs $(\mathbf{a}, b = \langle \mathbf{s}, \mathbf{a} \rangle + e)$ and $(\mathbf{a}, b)$.

  We want to solve search-LWE: given pairs $(\mathbf{a}, b)$, find $\mathbf{s}$.

- If $\boxed{q = \text{poly}(n)}$, to find $s_1 \in \mathbb{Z}_q$ it suffices to test whether $s_1 \stackrel{?}{=} 0$, because we can shift s_1 by $0, 1, \ldots, q-1$. Same for $s_2, s_3, \ldots, s_n$.

The test: for each $(\mathbf{a}, b)$, choose fresh $r \leftarrow \mathbb{Z}_q$. Invoke $\mathcal{D}$ on pairs

$$(\mathbf{a}' = \mathbf{a} - (r, 0, \ldots, 0)\,,\, b).$$

Search/Decision Equivalence [BFKL'94,R'05]

- Suppose $\mathcal{D}$ solves decision-LWE: it 'perfectly' distinguishes between pairs $(\mathbf{a}, b = \langle \mathbf{s}, \mathbf{a} \rangle + e)$ and $(\mathbf{a}, b)$.

  We want to solve search-LWE: given pairs $(\mathbf{a}, b)$, find $\mathbf{s}$.

- If $\boxed{q = \text{poly}(n)}$, to find $s_1 \in \mathbb{Z}_q$ it suffices to test whether $s_1 \stackrel{?}{=} 0$, because we can shift s_1 by $0, 1, \ldots, q-1$. Same for $s_2, s_3, \ldots, s_n$.

The test: for each $(\mathbf{a}, b)$, choose fresh $r \leftarrow \mathbb{Z}_q$. Invoke $\mathcal{D}$ on pairs

$$(\mathbf{a}' = \mathbf{a} - (r, 0, \ldots, 0)\,,\, b).$$

- Notice: $b = \langle \mathbf{s}, \mathbf{a}' \rangle + s_1 \cdot r + e$.

Search/Decision Equivalence [BFKL'94,R'05]

- Suppose $\mathcal{D}$ solves decision-LWE: it 'perfectly' distinguishes between pairs $(\mathbf{a}, b = \langle \mathbf{s}, \mathbf{a} \rangle + e)$ and $(\mathbf{a}, b)$.

  We want to solve search-LWE: given pairs $(\mathbf{a}, b)$, find $\mathbf{s}$.

- If $\boxed{q = \text{poly}(n)}$, to find $s_1 \in \mathbb{Z}_q$ it suffices to test whether $s_1 \stackrel{?}{=} 0$, because we can shift s_1 by $0, 1, \ldots, q-1$. Same for $s_2, s_3, \ldots, s_n$.

<u>The test</u>: for each $(\mathbf{a}, b)$, choose fresh $r \leftarrow \mathbb{Z}_q$. Invoke $\mathcal{D}$ on pairs

$$(\mathbf{a}' = \mathbf{a} - (r, 0, \ldots, 0)\,,\, b).$$

- Notice: $b = \langle \mathbf{s}, \mathbf{a}' \rangle + s_1 \cdot r + e$.
  - ⋆ If $s_1 = 0$, then $b = \langle \mathbf{s}, \mathbf{a}' \rangle + e \Rightarrow \mathcal{D}$ accepts.

Search/Decision Equivalence [BFKL'94,R'05]

- Suppose $\mathcal{D}$ solves decision-LWE: it 'perfectly' distinguishes between pairs $(\mathbf{a}, b = \langle \mathbf{s}, \mathbf{a} \rangle + e)$ and $(\mathbf{a}, b)$.

  We want to solve search-LWE: given pairs $(\mathbf{a}, b)$, find $\mathbf{s}$.

- If $\boxed{q = \mathrm{poly}(n)}$, to find $s_1 \in \mathbb{Z}_q$ it suffices to test whether $s_1 \stackrel{?}{=} 0$, because we can shift s_1 by $0, 1, \ldots, q-1$. Same for $s_2, s_3, \ldots, s_n$.

<u>The test</u>: for each $(\mathbf{a}, b)$, choose fresh $r \leftarrow \mathbb{Z}_q$. Invoke $\mathcal{D}$ on pairs

$$(\mathbf{a}' = \mathbf{a} - (r, 0, \ldots, 0)\,,\, b).$$

- Notice: $b = \langle \mathbf{s}, \mathbf{a}' \rangle + s_1 \cdot r + e$.
  - ⋆ If $s_1 = 0$, then $b = \langle \mathbf{s}, \mathbf{a}' \rangle + e \Rightarrow \mathcal{D}$ accepts.
  - ⋆ If $s_1 \neq 0$ and $\boxed{q \text{ prime}}$ then $b =$ uniform $\Rightarrow \mathcal{D}$ rejects.

Search/Decision Equivalence [BFKL'94,R'05]

- Suppose $\mathcal{D}$ solves decision-LWE: it 'perfectly' distinguishes between pairs $(\mathbf{a}, b = \langle \mathbf{s}, \mathbf{a} \rangle + e)$ and $(\mathbf{a}, b)$.

  We want to solve search-LWE: given pairs $(\mathbf{a}, b)$, find $\mathbf{s}$.

- If $\boxed{q = \text{poly}(n)}$, to find $s_1 \in \mathbb{Z}_q$ it suffices to test whether $s_1 \stackrel{?}{=} 0$, because we can shift s_1 by $0, 1, \ldots, q-1$. Same for $s_2, s_3, \ldots, s_n$.

The test: for each $(\mathbf{a}, b)$, choose fresh $r \leftarrow \mathbb{Z}_q$. Invoke $\mathcal{D}$ on pairs

$$(\mathbf{a}' = \mathbf{a} - (r, 0, \ldots, 0)\,,\, b).$$

- Notice: $b = \langle \mathbf{s}, \mathbf{a}' \rangle + s_1 \cdot r + e$.
  - ⋆ If $s_1 = 0$, then $b = \langle \mathbf{s}, \mathbf{a}' \rangle + e \Rightarrow \mathcal{D}$ accepts.
  - ⋆ If $s_1 \neq 0$ and $\boxed{q \text{ prime}}$ then $b = \text{uniform} \Rightarrow \mathcal{D}$ rejects.
- Don't really need $\boxed{\text{prime } q = \text{poly}(n)}$ [P'09,ACPS'09,MM'11,MP'12]

LWE $\Rightarrow$ Synthesizer?

Learning With Errors (LWE) [Regev'05]

- Hard to distinguish $(\mathbf{a}_i \in \mathbb{Z}_q^n\ ,\ b_i = \langle \mathbf{a}_i, \mathrm{s} \rangle + e_i)$ from $(\mathbf{a}_i\ ,\ b_i)$, where $\mathbf{a}_i, b_i, \mathrm{s}$ uniform and $e_i \leftarrow \chi =$ Gaussian over $\mathbb{Z}$ w/ param $\alpha q > \sqrt{n}$

LWE $\Rightarrow$ Synthesizer?

Learning With Errors (LWE) [Regev'05]

- Hard to distinguish $(\mathbf{a}_i \in \mathbb{Z}_q^n\ ,\ b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i)$ from $(\mathbf{a}_i\ ,\ b_i)$, where $\mathbf{a}_i, b_i, \mathbf{s}$ uniform and $e_i \leftarrow \chi =$ Gaussian over $\mathbb{Z}$ w/ param $\alpha q > \sqrt{n}$
- By hybrid argument, can't distinguish tuples

  $$(\mathbf{A}_i \in \mathbb{Z}_q^{n \times n}\ ,\ \mathbf{A}_i \cdot \mathbf{S}_1 + \mathbf{E}_{i,1} \in \mathbb{Z}_q^{n \times n}\ ,\ \mathbf{A}_i \cdot \mathbf{S}_2 + \mathbf{E}_{i,2} \in \mathbb{Z}_q^{n \times n}\ ,\ \ldots)$$

LWE $\Rightarrow$ Synthesizer?

Learning With Errors (LWE) [Regev'05]

- Hard to distinguish $(\mathbf{a}_i \in \mathbb{Z}_q^n \ , \ b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i)$ from $(\mathbf{a}_i \ , \ b_i)$, where $\mathbf{a}_i, b_i, \mathbf{s}$ uniform and $e_i \leftarrow \chi =$ Gaussian over $\mathbb{Z}$ w/ param $\alpha q > \sqrt{n}$
- By hybrid argument, can't distinguish tuples

$$(\mathbf{A}_i \in \mathbb{Z}_q^{n \times n} \ , \ \mathbf{A}_i \cdot \mathbf{S}_1 + \mathbf{E}_{i,1} \in \mathbb{Z}_q^{n \times n} \ , \ \mathbf{A}_i \cdot \mathbf{S}_2 + \mathbf{E}_{i,2} \in \mathbb{Z}_q^{n \times n} \ , \ \ldots)$$

An LWE-Based Synthesizer?

	$\mathbf{S}_1$	$\mathbf{S}_2$	$\cdots$
$\mathbf{A}_1$	$\mathbf{A}_1 \cdot \mathbf{S}_1 + \mathbf{E}_{1,1}$	$\mathbf{A}_1 \cdot \mathbf{S}_2 + \mathbf{E}_{1,2}$	$\cdots$
$\mathbf{A}_2$	$\mathbf{A}_2 \cdot \mathbf{S}_1 + \mathbf{E}_{2,1}$	$\mathbf{A}_2 \cdot \mathbf{S}_2 + \mathbf{E}_{2,2}$	$\cdots$
$\vdots$		$\ddots$	

LWE $\Rightarrow$ Synthesizer?

Learning With Errors (LWE) [Regev'05]

- Hard to distinguish $(\mathbf{a}_i \in \mathbb{Z}_q^n \ , \ b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i)$ from $(\mathbf{a}_i \ , \ b_i)$, where $\mathbf{a}_i, b_i, \mathbf{s}$ uniform and $e_i \leftarrow \chi =$ Gaussian over $\mathbb{Z}$ w/ param $\alpha q > \sqrt{n}$
- By hybrid argument, can't distinguish tuples

$$(\mathbf{A}_i \in \mathbb{Z}_q^{n\times n} \ , \ \mathbf{A}_i \cdot \mathbf{S}_1 + \mathbf{E}_{i,1} \in \mathbb{Z}_q^{n\times n} \ , \ \mathbf{A}_i \cdot \mathbf{S}_2 + \mathbf{E}_{i,2} \in \mathbb{Z}_q^{n\times n} \ , \ \ldots)$$

An LWE-Based Synthesizer?

	$\mathbf{S}_1$	$\mathbf{S}_2$	$\cdots$
$\mathbf{A}_1$	$\mathbf{A}_1 \cdot \mathbf{S}_1 + \mathbf{E}_{1,1}$	$\mathbf{A}_1 \cdot \mathbf{S}_2 + \mathbf{E}_{1,2}$	$\cdots$
$\mathbf{A}_2$	$\mathbf{A}_2 \cdot \mathbf{S}_1 + \mathbf{E}_{2,1}$	$\mathbf{A}_2 \cdot \mathbf{S}_2 + \mathbf{E}_{2,2}$	$\cdots$
$\vdots$		$\ddots$	

✔ $\{\mathbf{A}_i \cdot \mathbf{S}_j + \mathbf{E}_{i,j}\} \stackrel{c}{\approx}$ Uniform, but...

LWE $\Rightarrow$ Synthesizer?

Learning With Errors (LWE) [Regev'05]

- Hard to distinguish $(\mathbf{a}_i \in \mathbb{Z}_q^n \ , \ b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i)$ from $(\mathbf{a}_i \ , \ b_i)$, where $\mathbf{a}_i, b_i, \mathbf{s}$ uniform and $e_i \leftarrow \chi =$ Gaussian over $\mathbb{Z}$ w/ param $\alpha q > \sqrt{n}$
- By hybrid argument, can't distinguish tuples

$$(\mathbf{A}_i \in \mathbb{Z}_q^{n \times n} \ , \ \mathbf{A}_i \cdot \mathbf{S}_1 + \mathbf{E}_{i,1} \in \mathbb{Z}_q^{n \times n} \ , \ \mathbf{A}_i \cdot \mathbf{S}_2 + \mathbf{E}_{i,2} \in \mathbb{Z}_q^{n \times n} \ , \ \ldots)$$

An LWE-Based Synthesizer?

	$\mathbf{S}_1$	$\mathbf{S}_2$	$\cdots$
$\mathbf{A}_1$	$\mathbf{A}_1 \cdot \mathbf{S}_1 + \mathbf{E}_{1,1}$	$\mathbf{A}_1 \cdot \mathbf{S}_2 + \mathbf{E}_{1,2}$	$\cdots$
$\mathbf{A}_2$	$\mathbf{A}_2 \cdot \mathbf{S}_1 + \mathbf{E}_{2,1}$	$\mathbf{A}_2 \cdot \mathbf{S}_2 + \mathbf{E}_{2,2}$	$\cdots$
$\vdots$		$\ddots$	

✔ $\{\mathbf{A}_i \cdot \mathbf{S}_j + \mathbf{E}_{i,j}\} \overset{c}{\approx}$ Uniform, but...

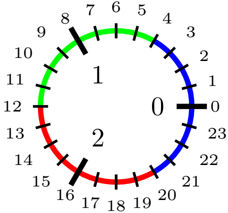
✗ What about $\mathbf{E}_{i,j}$?

Synthesizer must be deterministic...

“Learning With Rounding” (LWR) [Banerjee Peikert Rosen’12]

- IDEA: generate errors deterministically by rounding $\mathbb{Z}_q$ to a “sparse” subset (e.g. $\mathbb{Z}_p$).

  (Common in decryption to remove error.)

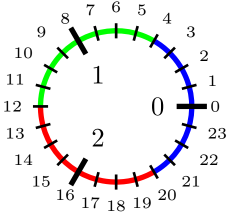


"Learning With Rounding" (LWR) [Banerjee Peikert Rosen'12]

- IDEA: generate errors deterministically by rounding $\mathbb{Z}_q$ to a "sparse" subset (e.g. $\mathbb{Z}_p$).

  (Common in decryption to remove error.)

  Let $p < q$ and define $\lfloor x \rceil_p = \lfloor (p/q) \cdot x \rceil \bmod p$.

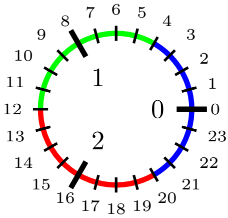


"Learning With Rounding" (LWR) [Banerjee Peikert Rosen'12]

- IDEA: generate errors deterministically by rounding $\mathbb{Z}_q$ to a "sparse" subset (e.g. $\mathbb{Z}_p$).

  (Common in decryption to remove error.)

  Let $p < q$ and define $\lfloor x \rceil_p = \lfloor (p/q) \cdot x \rceil \bmod p$.



- LWR problem: distinguish any $m =$ poly pairs

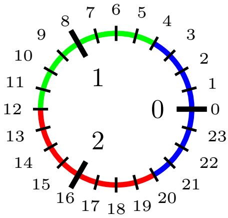
$$\left(\mathbf{a}_i \ , \ \lfloor \langle \mathbf{a}_i, \mathbf{s} \rangle \rceil_p\right) \in \mathbb{Z}_q \times \mathbb{Z}_p \quad \text{from uniform}$$

"Learning With Rounding" (LWR) [Banerjee Peikert Rosen'12]

- IDEA: generate errors deterministically by rounding $\mathbb{Z}_q$ to a "sparse" subset (e.g. $\mathbb{Z}_p$).

  (Common in decryption to remove error.)

  Let $p < q$ and define $\lfloor x \rceil_p = \lfloor (p/q) \cdot x \rceil \bmod p$.



- LWR problem: distinguish any $m =$ poly pairs

$$\left(\mathbf{a}_i\ ,\ \lfloor \langle \mathbf{a}_i, \mathbf{s} \rangle \rceil_p\right) \in \mathbb{Z}_q \times \mathbb{Z}_p \quad \text{from uniform}$$

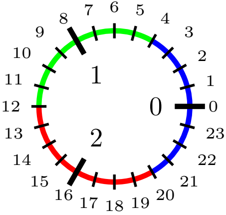
Interpretation: LWE conceals low-order bits by adding small random error. LWR just discards those bits instead.

“Learning With Rounding” (LWR) [Banerjee Peikert Rosen’12]

- IDEA: generate errors deterministically by rounding $\mathbb{Z}_q$ to a “sparse” subset (e.g. $\mathbb{Z}_p$).

  (Common in decryption to remove error.)

  Let $p < q$ and define $\lfloor x \rceil_p = \lfloor (p/q) \cdot x \rceil \bmod p$.



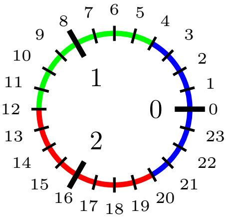
- LWR problem: distinguish any $m =$ poly pairs

$$\left(\mathbf{a}_i \ , \ \lfloor \langle \mathbf{a}_i, \mathbf{s} \rangle \rceil_p\right) \in \mathbb{Z}_q \times \mathbb{Z}_p \quad \text{from uniform}$$

  Interpretation: LWE conceals low-order bits by adding small random error. LWR just discards those bits instead.

- Theorem: LWE $\leq$ LWR for $q \geq p \cdot n^{\omega(1)}$ [but it seems 2^n-hard for $q \geq p\sqrt{n}$]

"Learning With Rounding" (LWR) [Banerjee Peikert Rosen'12]



- IDEA: generate errors deterministically by rounding $\mathbb{Z}_q$ to a "sparse" subset (e.g. $\mathbb{Z}_p$).

  (Common in decryption to remove error.)

  Let $p < q$ and define $\lfloor x \rceil_p = \lfloor (p/q) \cdot x \rceil \bmod p$.

- LWR problem: distinguish any $m =$ poly pairs

$$\left(\mathbf{a}_i\ ,\ \lfloor \langle \mathbf{a}_i, \mathbf{s} \rangle \rceil_p\right) \in \mathbb{Z}_q \times \mathbb{Z}_p \quad \text{from uniform}$$

  Interpretation: LWE conceals low-order bits by adding small random error. LWR just discards those bits instead.

- Theorem: LWE $\leq$ LWR for $q \geq p \cdot n^{\omega(1)}$ [but it seems 2^n-hard for $q \geq p\sqrt{n}$]

  Proof idea: w.h.p., $(\,\mathbf{a}\,,\ \lfloor \langle \mathbf{a}, \mathbf{s} \rangle + e \rceil_p\,) = (\,\mathbf{a}\,,\ \lfloor \langle \mathbf{a}, \mathbf{s} \rangle \rceil_p\,)$

  and $(\,\mathbf{a}\,,\ \lfloor \text{Unif}(\mathbb{Z}_q) \rceil_p\,) = (\,\mathbf{a}\,,\ \text{Unif}(\mathbb{Z}_p)\,)$

Properties of LWR

1. Random Self Reducubility:
   On input $A, R(As)$, output $AX, R(As)$, for random $X \in \mathbb{Z}_q^{n \times n}$,

   $$AX, R(As) \sim A, R(AX^{-1}s) = A, R(A(X^{-1}s)).$$

   Similar to LWE, but shift labels (not secret).

Properties of LWR

1. Random Self Reducubility:
   On input $A, R(As)$, output $AX, R(As)$, for random $X \in \mathbb{Z}_q^{n\times n}$,
   $$AX, R(As) \sim A, R(AX^{-1}s) = A, R(A(X^{-1}s)).$$
   Similar to LWE, but shift labels (not secret).

2. Search/Decision:
   On input A, b output $A + ue_1^T$ where u is a random vector,
   If $s_1 = 0$,
   $$A + ue_1^T, R(As) \sim A, R(As + (s_1)u) = A, R(As)$$
   If $s1 \neq 0$,
   $$A + ue_1^T, R(As) \sim A, R(As + (s_1)u) = A, R(u)$$

LWR-Based Synthesizer & PRF

- Synthesizer $S\colon \mathbb{Z}_q^{n\times n} \times \mathbb{Z}_q^{n\times n} \to \mathbb{Z}_p^{n\times n}$ is $\quad S(\mathbf{A}, \mathbf{S}) = \lfloor \mathbf{A} \cdot \mathbf{S} \rceil_p$.

  (Note: range $\mathbb{Z}_p$ is slightly smaller than domain $\mathbb{Z}_q$. Only limits composition.)

LWR-Based Synthesizer & PRF

- Synthesizer $S \colon \mathbb{Z}_q^{n \times n} \times \mathbb{Z}_q^{n \times n} \to \mathbb{Z}_p^{n \times n}$ is $\quad S(\mathbf{A}, \mathbf{S}) = \lfloor \mathbf{A} \cdot \mathbf{S} \rceil_p$.

  (Note: range $\mathbb{Z}_p$ is slightly smaller than domain $\mathbb{Z}_q$. Only limits composition.)

PRF on Domain $\{0,1\}^{k=2^d}$

- "Tower" of public moduli $q_d > q_{d-1} > \cdots > q_0$.
- Secret key is $2k$ square matrices $\mathbf{S}_{i,b}$ over $\mathbb{Z}_{q_d}$ for $i \in [k]$, $b \in \{0,1\}$.

LWR-Based Synthesizer & PRF

- Synthesizer $S \colon \mathbb{Z}_q^{n \times n} \times \mathbb{Z}_q^{n \times n} \to \mathbb{Z}_p^{n \times n}$ is $\quad S(\mathbf{A}, \mathbf{S}) = \lfloor \mathbf{A} \cdot \mathbf{S} \rceil_p$.

  (Note: range $\mathbb{Z}_p$ is slightly smaller than domain $\mathbb{Z}_q$. Only limits composition.)

PRF on Domain $\{0,1\}^{k=2^d}$

- "Tower" of public moduli $q_d > q_{d-1} > \cdots > q_0$.
- Secret key is $2k$ square matrices $\mathbf{S}_{i,b}$ over $\mathbb{Z}_{q_d}$ for $i \in [k]$, $b \in \{0,1\}$.
- Depth $d = \lg k$ tree of LWR synthesizers:

$$F_{\{\mathbf{S}_{i,b}\}}(x_1 \cdots x_8) =$$

$$\Big\lfloor \Big\lfloor \lfloor \mathbf{S}_{1,x_1} \cdot \mathbf{S}_{2,x_2} \rceil_{q_2} \cdot \lfloor \mathbf{S}_{3,x_3} \cdot \mathbf{S}_{4,x_4} \rceil_{q_2} \Big\rceil_{q_1} \cdot \Big\lfloor \lfloor \mathbf{S}_{5,x_5} \cdot \mathbf{S}_{6,x_6} \rceil_{q_2} \cdot \lfloor \mathbf{S}_{7,x_7} \cdot \mathbf{S}_{8,x_8} \rceil_{q_2} \Big\rceil_{q_1} \Big\rceil_{q_0}$$

Even Better

- Synthesizer $S\colon \mathbb{Z}_q^{m\times n} \times \mathbb{Z}_q^{n\times m} \to \mathbb{Z}_p^{m\times m}$ is $\quad S(\mathbf{A}, \mathbf{S}) = \lfloor \mathbf{A} \cdot \mathbf{S} \rceil_p$.

  Idea: to match range and domain sizes take $m = 2n$ and $q = p^2$.

Even Better

- Synthesizer $S\colon \mathbb{Z}_q^{m\times n} \times \mathbb{Z}_q^{n\times m} \to \mathbb{Z}_p^{m\times m}$ is $\quad S(\mathbf{A},\mathbf{S}) = \lfloor \mathbf{A}\cdot\mathbf{S} \rceil_p$.
  Idea: to match range and domain sizes take $m = 2n$ and $q = p^2$.

PRF on Domain $\{0,1\}^{k=2^d}$

- Public modulus $q = p^2$.
- Secret key is $2k$ $m \times n$ matrices $\mathbf{S}_{i,b}$ over $\mathbb{Z}_q$ for $i \in [k]$, $b \in \{0,1\}$.
- Given $\mathbf{S}_1, \mathbf{S}_2 \in \mathbb{Z}_q^{2n\times n}$ "cast" $\left\lfloor \mathbf{S}_1 \cdot \mathbf{S}_2^t \right\rceil_p \in \mathbb{Z}_p^{2n\times 2n}$ into $\mathbb{Z}_q^{2n\times n}$.
  (Works because $\|\mathbf{S}_i\| = \| \left\lfloor \mathbf{S}_1 \cdot \mathbf{S}_2 \right\rceil_p \| = 4n^2 \log p$.)

Even Better

- Synthesizer $S\colon \mathbb{Z}_q^{m\times n} \times \mathbb{Z}_q^{n\times m} \to \mathbb{Z}_p^{m\times m}$ is $S(\mathbf{A}, \mathbf{S}) = \lfloor \mathbf{A} \cdot \mathbf{S} \rceil_p$.
  Idea: to match range and domain sizes take $m = 2n$ and $q = p^2$.

PRF on Domain $\{0,1\}^{k=2^d}$

- Public modulus $q = p^2$.
- Secret key is $2k$ $m \times n$ matrices $\mathbf{S}_{i,b}$ over $\mathbb{Z}_q$ for $i \in [k]$, $b \in \{0,1\}$.
- Given $\mathbf{S}_1, \mathbf{S}_2 \in \mathbb{Z}_q^{2n\times n}$ "cast" $\left\lfloor \mathbf{S}_1 \cdot \mathbf{S}_2^t \right\rceil_p \in \mathbb{Z}_p^{2n\times 2n}$ into $\mathbb{Z}_q^{2n\times n}$.
  (Works because $\|\mathbf{S}_i\| = \| \lfloor \mathbf{S}_1 \cdot \mathbf{S}_2 \rceil_p \| = 4n^2 \log p$.)
- Depth $d = \lg k$ tree of LWR synthesizers:

$$\left\lfloor \left\lfloor \left\lfloor \mathbf{S}_{1,x_1} \cdot \mathbf{S}_{2,x_2}^t \right\rceil_q \cdot \left\lfloor \mathbf{S}_{3,x_3} \cdot \mathbf{S}_{4,x_4}^t \right\rceil_q \right\rceil_q \cdot \left\lfloor \left\lfloor \mathbf{S}_{5,x_5} \cdot \mathbf{S}_{6,x_6}^t \right\rceil_q \cdot \left\lfloor \mathbf{S}_{7,x_7} \cdot \mathbf{S}_{8,x_8}^t \right\rceil_q \right\rceil_q \right\rceil_q$$

More Efficient?

Ring Learning With Errors (RLWE) [LPR'10]

- For (e.g.) n a power of 2, define "cyclotomic" polynomial rings

$$R := \mathbb{Z}[x]/(x^n + 1) \quad \text{and} \quad R_q := R/qR = \mathbb{Z}_q[x]/(x^n + 1).$$

More Efficient?

Ring Learning With Errors (RLWE) [LPR'10]

- For (e.g.) n a power of 2, define "cyclotomic" polynomial rings

$$R := \mathbb{Z}[x]/(x^n + 1) \quad \text{and} \quad R_q := R/qR = \mathbb{Z}_q[x]/(x^n + 1).$$

- Hard to distinguish m pairs $(a_i\ ,\ a_i \cdot s + e_i) \in R_q \times R_q$ from uniform, where $a_i, s \leftarrow R_q$ uniform and e_i "short."

More Efficient?

Ring Learning With Errors (RLWE) [LPR'10]

- For (e.g.) n a power of 2, define "cyclotomic" polynomial rings

$$R := \mathbb{Z}[x]/(x^n + 1) \quad \text{and} \quad R_q := R/qR = \mathbb{Z}_q[x]/(x^n + 1).$$

- Hard to distinguish m pairs $(a_i\ ,\ a_i \cdot s + e_i) \in R_q \times R_q$ from uniform, where $a_i, s \leftarrow R_q$ uniform and e_i "short."
- Shorter description/faster computation (using FFT/NTT).

Shallower?

- Synth-based PRF is $\log k$ levels of NC^1 synthesizers $\Rightarrow$ NC^2.

Shallower?

- Synth-based PRF is $\log k$ levels of NC^1 synthesizers $\Rightarrow \mathsf{NC}^2$.
- [NR'97]: direct PRFs from DDH / factoring, in $\mathsf{TC}^0 \subseteq \mathsf{NC}^1$.

$$F_{g,s_1,\ldots,s_k}(x_1 \cdots x_k) = g^{\prod s_i^{x_i}}$$

(Computing this in TC^0 needs large circuits, though...)

Shallower?

- Synth-based PRF is $\log k$ levels of NC^1 synthesizers $\Rightarrow \mathsf{NC}^2$.
- [NR'97]: direct PRFs from DDH / factoring, in $\mathsf{TC}^0 \subseteq \mathsf{NC}^1$.

$$F_{g,s_1,\ldots,s_k}(x_1 \cdots x_k) = g^{\prod s_i^{x_i}}$$

(Computing this in TC^0 needs large circuits, though...)

Direct LWE-Based Construction

- Public moduli $q > p$.
- Secret key is uniform $a \leftarrow R_q$ and short $s_1, \ldots, s_k \in R$.

Shallower?

- Synth-based PRF is $\log k$ levels of NC^1 synthesizers $\Rightarrow$ NC^2.
- [NR'97]: direct PRFs from DDH / factoring, in TC$^0 \subseteq$ NC^1.

$$F_{g,s_1,\ldots,s_k}(x_1 \cdots x_k) = g^{\prod s_i^{x_i}}$$

(Computing this in TC^0 needs large circuits, though...)

Direct LWE-Based Construction

- Public moduli $q > p$.
- Secret key is uniform $a \leftarrow R_q$ and short $s_1, \ldots, s_k \in R$.
- "Rounded subset-product" function:

$$F_{a,s_1,\ldots,s_k}(x_1 \cdots x_k) = \left\lfloor a \cdot \prod_{i=1}^{k} s_i^{x_i} \bmod q \right\rceil_p$$

Shallower?

- Synth-based PRF is $\log k$ levels of NC^1 synthesizers $\Rightarrow$ NC^2.
- [NR'97]: direct PRFs from DDH / factoring, in $\mathsf{TC}^0 \subseteq \mathsf{NC}^1$.

$$F_{g,s_1,\ldots,s_k}(x_1 \cdots x_k) = g^{\prod s_i^{x_i}}$$

(Computing this in TC^0 needs large circuits, though. . .)

Direct LWE-Based Construction

- Public moduli $q > p$.
- Secret key is uniform $a \leftarrow R_q$ and short $s_1, \ldots, s_k \in R$.
- "Rounded subset-product" function:

$$F_{a,s_1,\ldots,s_k}(x_1 \cdots x_k) = \left\lfloor a \cdot \prod_{i=1}^{k} s_i^{x_i} \bmod q \right\rceil_p$$

Has small(ish) TC^0 circuit, via CRT and reduction to subset-sum.

Proof Outline

- Seed is uniform $a \in R_q$ and short $s_1, \ldots, s_k \in R$.

$$F_{a,s_1,\ldots,s_k}(x_1 \cdots x_k) = \left\lfloor a \cdot s_1^{x_1} \cdots s_k^{x_k} \bmod q \right\rceil_p$$

Proof Outline

- Seed is uniform $a \in R_q$ and short $s_1, \ldots, s_k \in R$.

$$F_{a,s_1,\ldots,s_k}(x_1 \cdots x_k) = \left\lfloor a \cdot s_1^{x_1} \cdots s_k^{x_k} \bmod q \right\rceil_p$$

- Like the LWE $\leq$ LWR proof, but "souped up" to handle queries.

Proof Outline

- Seed is uniform $a \in R_q$ and short $s_1, \ldots, s_k \in R$.

$$F_{a,s_1,\ldots,s_k}(x_1 \cdots x_k) = \left\lfloor a \cdot s_1^{x_1} \cdots s_k^{x_k} \bmod q \right\rceil_p$$

- Like the LWE $\leq$ LWR proof, but "souped up" to handle queries.

  Thought experiment: answer queries with

$$\tilde{F}(x) := \left\lfloor (a \cdot s_1^{x_1} + x_1 \cdot e_{x_1}) \cdot s_2^{x_2} \cdots s_k^{x_k} \right\rceil_p = \left\lfloor a \prod_{i=1}^{k} s_i^{x_i} + x_1 \cdot e_{x_1} \cdot \prod_{i=2}^{k} s_i^{x_i} \right\rceil$$

  W.h.p., $\tilde{F}(x) = F(x)$ on all queries due to "small" error & rounding.

Proof Outline

- Seed is uniform $a \in R_q$ and short $s_1, \ldots, s_k \in R$.

$$F_{a,s_1,\ldots,s_k}(x_1 \cdots x_k) = \left\lfloor a \cdot s_1^{x_1} \cdots s_k^{x_k} \bmod q \right\rceil_p$$

- Like the LWE $\leq$ LWR proof, but "souped up" to handle queries.

  Thought experiment: answer queries with

$$\tilde{F}(x) := \left\lfloor (a \cdot s_1^{x_1} + x_1 \cdot e_{x_1}) \cdot s_2^{x_2} \cdots s_k^{x_k} \right\rceil_p = \left\lfloor a \prod_{i=1}^{k} s_i^{x_i} + x_1 \cdot e_{x_1} \cdot \prod_{i=2}^{k} s_i^{x_i} \right\rceil$$

  W.h.p., $\tilde{F}(x) = F(x)$ on all queries due to "small" error & rounding.

- Replace $(a, a \cdot s_1 + e_{x_1})$ with uniform (a_0, a_1) [ring-LWE].

  $\Rightarrow$ New function $F'(x) = \lfloor a_{x_1} \cdot s_2^{x_2} \cdots s_k^{x_k} \rceil_p$.

Proof Outline

- Seed is uniform $a \in R_q$ and short $s_1, \ldots, s_k \in R$.

$$F_{a,s_1,\ldots,s_k}(x_1 \cdots x_k) = \left\lfloor a \cdot s_1^{x_1} \cdots s_k^{x_k} \bmod q \right\rceil_p$$

- Like the LWE $\leq$ LWR proof, but "souped up" to handle queries.

  Thought experiment: answer queries with

$$\tilde{F}(x) := \left\lfloor (a \cdot s_1^{x_1} + x_1 {\cdot} e_{x_1}) \cdot s_2^{x_2} \cdots s_k^{x_k} \right\rceil_p = \left\lfloor a \prod_{i=1}^{k} s_i^{x_i} + x_1 {\cdot} e_{x_1} \cdot \prod_{i=2}^{k} s_i^{x_i} \right\rceil$$

  W.h.p., $\tilde{F}(x) = F(x)$ on all queries due to "small" error & rounding.

- Replace $(a, a \cdot s_1 + e_{x_1})$ with uniform (a_0, a_1) [ring-LWE].

  $\Rightarrow$ New function $F'(x) = \lfloor a_{x_1} \cdot s_2^{x_2} \cdots s_k^{x_k} \rceil_p$.

- Repeat for $s_2, s_3, \ldots$ until $F''''''(x) = \lfloor a_x \rceil_p =$ Uniform func. $\square$

Open Questions 1

1. Better (worst-case) hardness for LWR, e.g. for $q/p = \sqrt{n}$?

   (The proof from LWE relies on approx factor and modulus $= n^{\omega(1)}$.)

   [AKPW'13]: LWE $\leq$ LWR for $q = n^{O(1)}$ (bounded $\#$samples).

Open Questions 1

1. Better (worst-case) hardness for LWR, e.g. for $q/p = \sqrt{n}$?

   (The proof from LWE relies on approx factor and modulus $= n^{\omega(1)}$.)

   [AKPW'13]: LWE $\leq$ LWR for $q = n^{O(1)}$ (bounded #samples).

2. Non-trivial algorithms for LWR?

Open Questions 1

1. Better (worst-case) hardness for LWR, e.g. for $q/p = \sqrt{n}$?

   (The proof from LWE relies on approx factor and modulus $= n^{\omega(1)}$.)

   [AKPW'13]: LWE $\leq$ LWR for $q = n^{O(1)}$ (bounded #samples).

2. Non-trivial algorithms for LWR?

   [BCGR'13]:

   - LWR $\leq$ LWE for $\lceil q/p \rceil = n^{O(1)}$ (uses ideas from [FGKP'06]).
   - Adaptations of [AG'11] and [BKL'03] to LWR.

Open Questions 2

1. Synth-based PRF can rely on approx factor and modulus $= n^{\Theta(1)}$.
   Direct construction still relies on approx factor and modulus $= n^{\Theta(k)}$.

Open Questions 2

1. Synth-based PRF can rely on approx factor and modulus $= n^{\Theta(1)}$.
   Direct construction still relies on approx factor and modulus $= n^{\Theta(k)}$.

   Are such strong assumptions necessary (even for these constructions)?

Open Questions 2

1. Synth-based PRF can rely on approx factor and modulus $= n^{\Theta(1)}$.
   Direct construction still relies on approx factor and modulus $= n^{\Theta(k)}$.

   Are such strong assumptions necessary (even for these constructions)?

   Conjecture (?): direct PRF is secure for integral $q/p = \text{poly}(n)$.

Open Questions 2

1. Synth-based PRF can rely on approx factor and modulus $= n^{\Theta(1)}$.

   Direct construction still relies on approx factor and modulus $= n^{\Theta(k)}$.

   Are such strong assumptions necessary (even for these constructions)?

   Conjecture (?): direct PRF is secure for integral $q/p = \text{poly}(n)$.

2. Efficient PRF from parity with noise (LPN)?

Open Questions 2

1. Synth-based PRF can rely on approx factor and modulus $= n^{\Theta(1)}$.

   Direct construction still relies on approx factor and modulus $= n^{\Theta(k)}$.

   Are such strong assumptions necessary (even for these constructions)?

   Conjecture (?): direct PRF is secure for integral $q/p = \mathrm{poly}(n)$.

2. Efficient PRF from parity with noise (LPN)?

3. Efficient PRF from subset sum?

Open Questions 2

1. Synth-based PRF can rely on approx factor and modulus $= n^{\Theta(1)}$.
   Direct construction still relies on approx factor and modulus $= n^{\Theta(k)}$.

   Are such strong assumptions necessary (even for these constructions)?

   Conjecture (?): direct PRF is secure for integral $q/p = \text{poly}(n)$.

2. Efficient PRF from parity with noise (LPN)?

3. Efficient PRF from subset sum?

http://factcenter.org