Pseudorandom Functions and Lattices

Abhishek Banerjee[1] Chris Peikert[1] Alon Rosen[2]

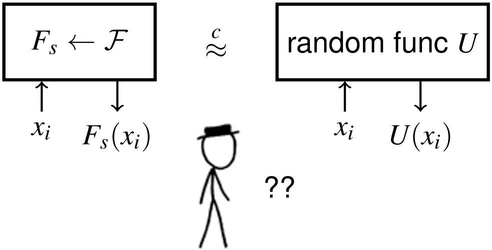
[1]Georgia Tech

[2]IDC Herzliya

Faces of Modern Cryptography
9 September 2011

Pseudorandom Functions [GGM'84]

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

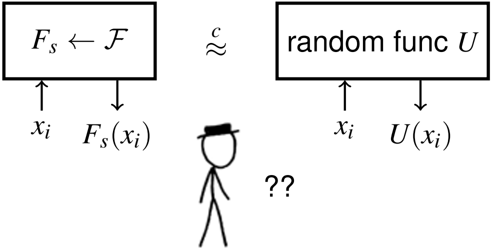


(The "seed" or "secret key" for F_s is s.)

(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,



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

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

(Images courtesy xkcd.org)

How to Construct PRFs

1. Heuristically: AES, Blowfish.
   - ✔ Fast!
   - ✔ Withstand known 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 (reductionist) guarantees

How to Construct PRFs

1. Heuristically: AES, Blowfish.
   - ✔ Fast!
   - ✔ Withstand known cryptanalytic techniques (linear, differential, . . .)
   - ✗ PRF security is subtle: want provable (reductionist) 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 (reductionist) 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 (reductionist) 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 / Naor-Reingold-Rosen [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]

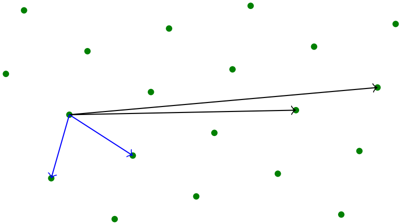
How to Construct PRFs

1. Heuristically: AES, Blowfish.
   - ✔ Fast!
   - ✔ Withstand known cryptanalytic techniques (linear, differential, . . .)
   - ✗ PRF security is subtle: want provable (reductionist) 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 / Naor-Reingold-Rosen [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]
   - ✗ Huge circuits that need mucho preprocessing
   - ✗ No "post-quantum" construction under standard assumptions

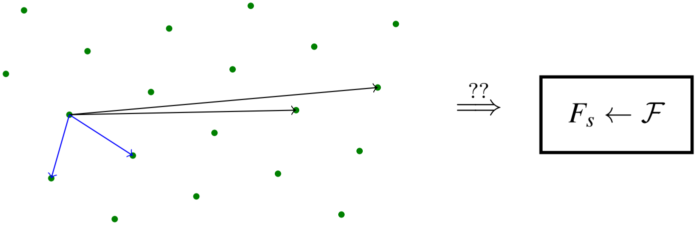
Why Not Try Lattices?



$$\overset{??}{\Longrightarrow}$$

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

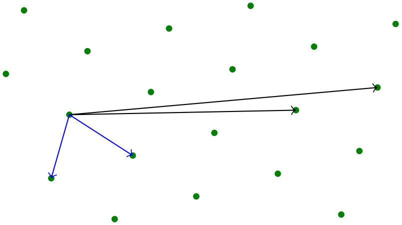
Why Not Try Lattices?



Advantages of Lattice Crypto Schemes

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

Why Not Try Lattices?



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

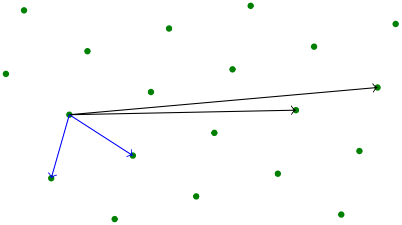
Advantages of Lattice Crypto Schemes

- 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?



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

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)

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

Our Results

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

Our Results

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]

Our Results

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: "derandomization" of LWE: deterministic errors

Our Results

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: "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 any $m = \text{poly}$: for $a_1, \ldots, a_m,\ b_1, \ldots, b_m \leftarrow D$,

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

	b_1	b_2	$\cdots$
a_1	$S(a_1, b_1)$	$S(a_1, b_2)$	$\cdots$
a_2	$S(a_2, b_1)$	$S(a_2, b_2)$	$\cdots$
$\vdots$		$\ddots$	

vs.

$U_{1,1}$	$U_{1,2}$	$\cdots$
$U_{2,1}$	$U_{2,2}$	$\cdots$
	$\ddots$	

Synthesizers and PRFs [NaorReingold'95]

Synthesizer

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

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

	b_1	b_2	$\cdots$
a_1	$S(a_1, b_1)$	$S(a_1, b_2)$	$\cdots$
a_2	$S(a_2, b_1)$	$S(a_2, b_2)$	$\cdots$
$\vdots$		$\ddots$	

vs.

$U_{1,1}$	$U_{1,2}$	$\cdots$
$U_{2,1}$	$U_{2,2}$	$\cdots$
	$\ddots$	

- 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 $\{ S(a_i , b_j) \} \stackrel{c}{\approx}$ 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: choose $F_\ell, F_r \leftarrow \mathcal{F}$ and let

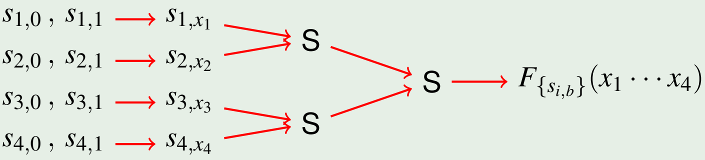
$$F_{(F_\ell,F_r)}(x_\ell , x_r) = S\big(F_\ell(x_\ell) , F_r(x_r) \big).$$

Synthesizers and PRFs [NaorReingold'95]

PRF from Synthesizer, Recursively

- Synthesizer $S \colon D \times D \to D$, where $\{ S(a_i , b_j) \} \stackrel{c}{\approx}$ 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: choose $F_\ell, F_r \leftarrow \mathcal{F}$ and let

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

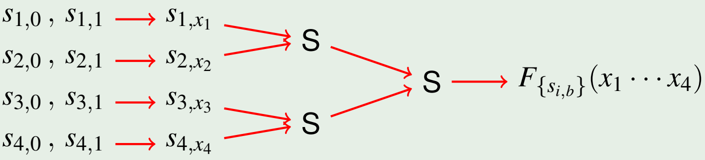


Synthesizers and PRFs [NaorReingold'95]

PRF from Synthesizer, Recursively

- Synthesizer $S\colon D \times D \to D$, where $\{\, S(a_i\,,\, b_j)\,\} \overset{c}{\approx} \text{Unif}(D^{m\times m})$.
- 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: choose $F_\ell, F_r \leftarrow \mathcal{F}$ and let

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



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

(Ring) Learning With Errors (RLWE) [Regev'05,LPR'10]

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

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

(Ring) Learning With Errors (RLWE) [Regev'05,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."

(Ring) Learning With Errors (RLWE) [Regev'05,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."

- By hybrid argument, for $s_1, s_2, \ldots \leftarrow R_q$ can't distinguish m tuples $(a_i \, , \, a_i \cdot s_1 + e_{i,1} \, , \, a_i \cdot s_2 + e_{i,2} \, , \, \ldots)$ from uniform.

(Ring) Learning With Errors (RLWE) [Regev'05,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."

- By hybrid argument, for $s_1, s_2, \ldots \leftarrow R_q$ can't distinguish m tuples $(a_i \, , \, a_i \cdot s_1 + e_{i,1} \, , \, a_i \cdot s_2 + e_{i,2} \, , \, \ldots)$ from uniform.

An RLWE-Based Synthesizer?

	s_1	s_2	$\cdots$
a_1	$a_1 \cdot s_1 + e_{1,1}$	$a_1 \cdot s_2 + e_{1,2}$	$\cdots$
a_2	$a_2 \cdot s_1 + e_{2,1}$	$a_2 \cdot s_2 + e_{2,2}$	$\cdots$
$\vdots$		$\ddots$	

(Ring) Learning With Errors (RLWE) [Regev'05,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."

- By hybrid argument, for $s_1, s_2, \ldots \leftarrow R_q$ can't distinguish m tuples $(a_i\ ,\ a_i \cdot s_1 + e_{i,1}\ ,\ a_i \cdot s_2 + e_{i,2}\ ,\ \ldots)$ from uniform.

An RLWE-Based Synthesizer?

	s_1	s_2	$\cdots$
a_1	$a_1 \cdot s_1 + e_{1,1}$	$a_1 \cdot s_2 + e_{1,2}$	$\cdots$
a_2	$a_2 \cdot s_1 + e_{2,1}$	$a_2 \cdot s_2 + e_{2,2}$	$\cdots$
$\vdots$		$\ddots$	

✔ $\{a_i \cdot s_j + e_{i,j}\} \stackrel{c}{\approx}$ Uniform, but…

(Ring) Learning With Errors (RLWE) [Regev'05,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."
- By hybrid argument, for $s_1, s_2, \ldots \leftarrow R_q$ can't distinguish m tuples $(a_i \ , \ a_i \cdot s_1 + e_{i,1} \ , \ a_i \cdot s_2 + e_{i,2} \ , \ \ldots)$ from uniform.

An RLWE-Based Synthesizer?

	s_1	s_2	$\cdots$
a_1	$a_1 \cdot s_1 + e_{1,1}$	$a_1 \cdot s_2 + e_{1,2}$	$\cdots$
a_2	$a_2 \cdot s_1 + e_{2,1}$	$a_2 \cdot s_2 + e_{2,2}$	$\cdots$
$\vdots$		$\ddots$	

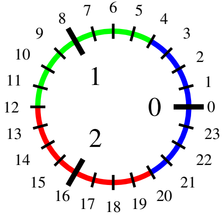
✔ $\{a_i \cdot s_j + e_{i,j}\} \stackrel{c}{\approx}$ Uniform, but...

✗ Where do $e_{i,j}$ come from? Synthesizer must be deterministic...

"Learning With Rounding" (LWR) [This work]

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

  (Common in decryption to remove error.)

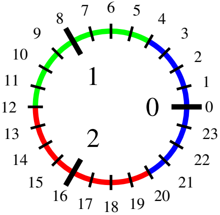


"Learning With Rounding" (LWR) [This work]

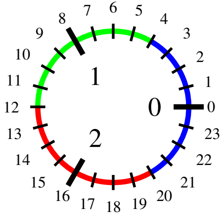
- IDEA: generate errors deterministically by rounding $\mathbb{Z}_q$ to a "sparse" subset (e.g. subgroup).

  (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) [This work]



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

  (Common in decryption to remove error.)

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

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

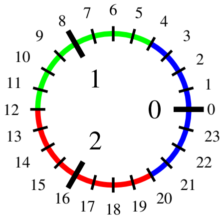
$$\left(a_i\ ,\ \lfloor a_i \cdot s \rceil_p\right) \in R_q \times R_p$$ from uniform

“Learning With Rounding” (LWR) [This work]

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

  (Common in decryption to remove error.)

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

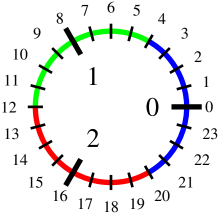


- Ring-LWR problem: distinguish any $m =$ poly pairs

$$\left(a_i\ ,\ \lfloor a_i \cdot s \rceil_p\right) \in R_q \times R_p \quad \text{from uniform}$$

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

"Learning With Rounding" (LWR) [This work]



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

  (Common in decryption to remove error.)

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

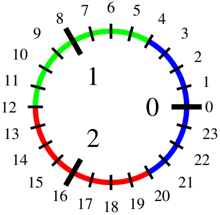
- Ring-LWR problem: distinguish any $m =$ poly pairs

$$\left(a_i \ , \ \lfloor a_i \cdot s \rceil_p\right) \in R_q \times R_p \quad \text{from uniform}$$

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

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

“Learning With Rounding” (LWR) [This work]



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

  (Common in decryption to remove error.)

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

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

$$\left(a_i\ ,\ \lfloor a_i \cdot s \rceil_p\right) \in R_q \times R_p \quad \text{from uniform}$$

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

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

  Main idea: w.h.p. $(a, \lfloor a \cdot s + e \rceil_p) = (a, \lfloor a \cdot s \rceil_p)$

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

LWR-Based Synthesizer & PRF

- Synthesizer $S\colon R_q \times R_q \to R_p$ is $S(a, s) = \lfloor a \cdot s \rceil_p$.

  Note: range R_p slightly smaller than domain R_q. (Limits composition.)

LWR-Based Synthesizer & PRF

- Synthesizer $S\colon R_q \times R_q \to R_p$ is $S(a, s) = \lfloor a \cdot s \rceil_p$.

  Note: range R_p slightly smaller than domain R_q. (Limits composition.)

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

- Public moduli $q_d > q_{d-1} > \cdots > q_0$.
- Secret key is $2k$ ring elements $s_{i,b} \in R_{q_d}$ for $i \in [k]$, $b \in \{0,1\}$.

LWR-Based Synthesizer & PRF

- Synthesizer $S \colon R_q \times R_q \to R_p$ is $S(a, s) = \lfloor a \cdot s \rceil_p$.

  Note: range R_p slightly smaller than domain R_q. (Limits composition.)

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

- Public moduli $q_d > q_{d-1} > \cdots > q_0$.
- Secret key is $2k$ ring elements $s_{i,b} \in R_{q_d}$ for $i \in [k]$, $b \in \{0,1\}$.
- Depth $d = \lg k$ tree of LWR synthesizers:

$$F_{\{s_{i,b}\}}(x_1 \cdots x_8) =$$
$$\Big\lfloor \Big\lfloor \lfloor s_{1,x_1} \cdot s_{2,x_2} \rceil_{q_2} \cdot \lfloor s_{3,x_3} \cdot s_{4,x_4} \rceil_{q_2} \Big\rceil_{q_1} \cdot \Big\lfloor \lfloor s_{5,x_5} \cdot s_{6,x_6} \rceil_{q_2} \cdot \lfloor s_{7,x_7} \cdot s_{8,x_8} \rceil_{q_2} \Big\rceil_{q_1} \Big\rceil_{q_0}$$

LWR-Based Synthesizer & PRF

- Synthesizer $S\colon R_q \times R_q \to R_p$ is $S(a,s) = \lfloor a \cdot s \rceil_p$.

  Note: range R_p slightly smaller than domain R_q. (Limits composition.)

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

- Public moduli $q_d > q_{d-1} > \cdots > q_0$.
- Secret key is $2k$ ring elements $s_{i,b} \in R_{q_d}$ for $i \in [k]$, $b \in \{0,1\}$.
- Depth $d = \lg k$ tree of LWR synthesizers:

$$F_{\{s_{i,b}\}}(x_1 \cdots x_8) = \Big\lfloor \big\lfloor \lfloor s_{1,x_1} \cdot s_{2,x_2} \rceil_{q_2} \cdot \lfloor s_{3,x_3} \cdot s_{4,x_4} \rceil_{q_2} \big\rceil_{q_1} \cdot \big\lfloor \lfloor s_{5,x_5} \cdot s_{6,x_6} \rceil_{q_2} \cdot \lfloor s_{7,x_7} \cdot s_{8,x_8} \rceil_{q_2} \big\rceil_{q_1} \Big\rceil_{q_0}$$

- Craig's talk: deja vu…

Shallower? More Efficient?

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

Shallowver? More Efficient?

- Synth-based PRF is $\log k$ levels of NC^1 synthesizers $\Rightarrow$ NC^2.
- [NR'97,NRR'00]: direct PRFs from DDH / factoring, in $\mathrm{TC}^0 \subseteq \mathrm{NC}^1$.

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

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

Shallower? More Efficient?

- Synth-based PRF is $\log k$ levels of NC^1 synthesizers $\Rightarrow$ NC^2.
- [NR'97,NRR'00]: 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 huge 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? More Efficient?

- Synth-based PRF is $\log k$ levels of NC^1 synthesizers $\Rightarrow$ NC^2.
- [NR'97,NRR'00]: 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 huge 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? More Efficient?

- Synth-based PRF is $\log k$ levels of NC^1 synthesizers $\Rightarrow$ NC^2.
- [NR'97,NRR'00]: 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 huge 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

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

  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

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

  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 $s_1, \ldots, s_k \in R$.

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

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

  Thought experiment: answer queries with

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

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

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

Open Questions

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

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

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

Open Questions

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

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

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

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

Open Questions

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

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

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

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

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

2. Synth-based PRF relies on approx factor and modulus $= n^{\Theta(\log k)}$.
   Direct construction 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 $q/p = \text{poly}(n)$.

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

Open Questions

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

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

2. Synth-based PRF relies on approx factor and modulus $= n^{\Theta(\log k)}$.
   Direct construction 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 $q/p = \text{poly}(n)$.

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

4. Efficient PRF from subset sum?

Open Questions

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

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

2. Synth-based PRF relies on approx factor and modulus $= n^{\Theta(\log k)}$.
   Direct construction 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 $q/p = \text{poly}(n)$.

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

4. Efficient PRF from subset sum?

Thanks!

Full paper: ePrint report #2011/401