Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller

Daniele Micciancio[1] Chris Peikert[2]

[1] UC San Diego

[2] Georgia Tech

IBM Research
8 September 2011

Lattice-Based Cryptography

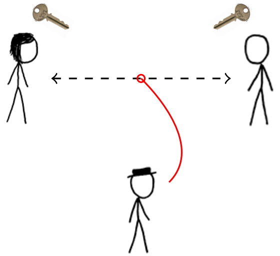
$y = g^x \bmod p$

$m^e \bmod N$

$N = p \cdot q$

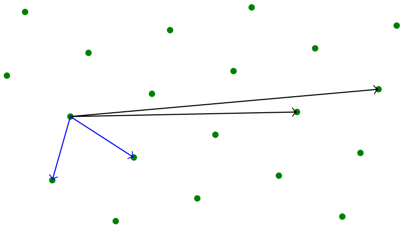
$e(g^a, g^b)$

$\Longrightarrow$

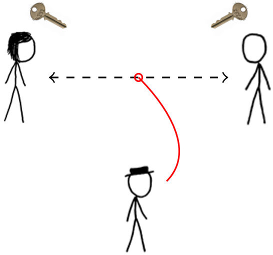


(Images courtesy xkcd.org)

Lattice-Based Cryptography

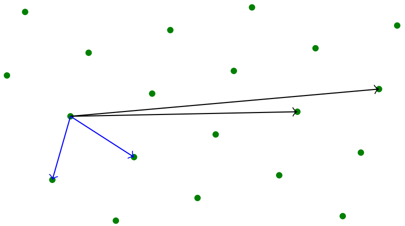


$\Longrightarrow$

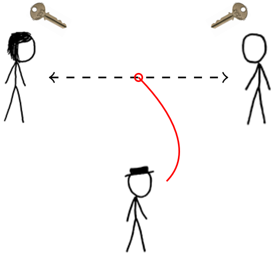


(Images courtesy xkcd.org)

Lattice-Based Cryptography



$\Longrightarrow$



Why?

- Simple & efficient: linear, highly parallel operations
- Resist quantum attacks (so far)
- Secure under worst-case hardness assumptions [Ajtai'96,...]
- Solve 'holy grail' problems like FHE [Gentry'09,...]

(Images courtesy xkcd.org)

Lattice-Based One-Way Functions

- Public key $\begin{bmatrix} \cdots & \mathbf{A} & \cdots \end{bmatrix} \in \mathbb{Z}_q^{n \times m}$ for $q = \text{poly}(n)$, $m = \Omega(n \log q)$.

Lattice-Based One-Way Functions

- Public key $\begin{bmatrix} \cdots & \mathbf{A} & \cdots \end{bmatrix} \in \mathbb{Z}_q^{n \times m}$ for $q = \text{poly}(n)$, $m = \Omega(n \log q)$.

$$f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x} \bmod q \in \mathbb{Z}_q^n$$

("short" $\mathbf{x}$, surjective)

CRHF if SIS hard [Ajtai'96,...]

Lattice-Based One-Way Functions

- Public key $\begin{bmatrix} \cdots & \mathbf{A} & \cdots \end{bmatrix} \in \mathbb{Z}_q^{n \times m}$ for $q = \text{poly}(n)$, $m = \Omega(n \log q)$.

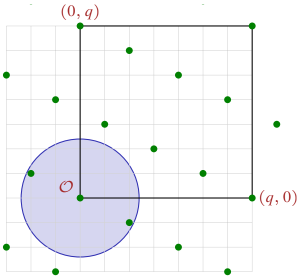
$f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x} \bmod q \in \mathbb{Z}_q^n$ ("short" $\mathbf{x}$, surjective)	$g_{\mathbf{A}}(\mathbf{s}, \mathbf{e}) = \mathbf{s}^t\mathbf{A} + \mathbf{e}^t \bmod q \in \mathbb{Z}_q^m$ ("short" $\mathbf{e}$, injective)
CRHF if SIS hard [Ajtai'96,...]	OWF if LWE hard [Regev'05,P'09]

Lattice-Based One-Way Functions

- Public key $\begin{bmatrix} \cdots & \mathbf{A} & \cdots \end{bmatrix} \in \mathbb{Z}_q^{n \times m}$ for $q = \text{poly}(n)$, $m = \Omega(n \log q)$.

$f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x} \bmod q \in \mathbb{Z}_q^n$ ("short" $\mathbf{x}$, surjective)	$g_{\mathbf{A}}(\mathbf{s}, \mathbf{e}) = \mathbf{s}^t\mathbf{A} + \mathbf{e}^t \bmod q \in \mathbb{Z}_q^m$ ("short" $\mathbf{e}$, injective)
CRHF if SIS hard [Ajtai'96,...]	OWF if LWE hard [Regev'05,P'09]

- Lattice interpretation: $\Lambda^{\perp}(\mathbf{A}) = \{\mathbf{x} \in \mathbb{Z}^m : f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x} = \mathbf{0} \bmod q\}$

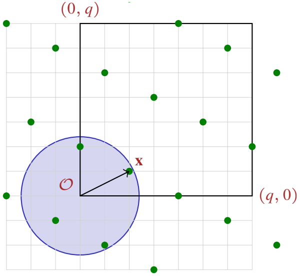


Lattice-Based One-Way Functions

- Public key $\begin{bmatrix} \cdots & \mathbf{A} & \cdots \end{bmatrix} \in \mathbb{Z}_q^{n \times m}$ for $q = \text{poly}(n)$, $m = \Omega(n \log q)$.

$f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x} \bmod q \in \mathbb{Z}_q^n$	$g_{\mathbf{A}}(\mathbf{s}, \mathbf{e}) = \mathbf{s}^t\mathbf{A} + \mathbf{e}^t \bmod q \in \mathbb{Z}_q^m$
("short" $\mathbf{x}$, surjective)	("short" $\mathbf{e}$, injective)
CRHF if SIS hard [Ajtai'96,...]	OWF if LWE hard [Regev'05,P'09]

- Lattice interpretation: $\Lambda^\perp(\mathbf{A}) = \{\mathbf{x} \in \mathbb{Z}^m : f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x} = \mathbf{0} \bmod q\}$



Lattice-Based One-Way Functions

- Public key $\begin{bmatrix} \cdots & \mathbf{A} & \cdots \end{bmatrix} \in \mathbb{Z}_q^{n \times m}$ for $q =$ poly(n), $m = \Omega(n \log q)$.

$f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x} \bmod q \in \mathbb{Z}_q^n$	$g_{\mathbf{A}}(\mathbf{s}, \mathbf{e}) = \mathbf{s}^t\mathbf{A} + \mathbf{e}^t \bmod q \in \mathbb{Z}_q^m$
("short" $\mathbf{x}$, surjective)	("short" $\mathbf{e}$, injective)
CRHF if SIS hard [Ajtai'96,...]	OWF if LWE hard [Regev'05,P'09]

- $f_{\mathbf{A}}$, $g_{\mathbf{A}}$ in forward direction yield CRHFs, CPA-secure encryption

  ...and not much else.

Trapdoor Inversion

- Many cryptographic applications need to invert $f_{\mathbf{A}}$ and/or $g_{\mathbf{A}}$.

Trapdoor Inversion

- Many cryptographic applications need to invert $f_{\mathbf{A}}$ and/or $g_{\mathbf{A}}$.

Invert $g_{\mathbf{A}}(\mathbf{s}, \mathbf{e}) = \mathbf{s}^t\mathbf{A} + \mathbf{e}^t \bmod q$:

find the unique preimage $\mathbf{s}$

(equivalently, $\mathbf{e}$)

Trapdoor Inversion

- Many cryptographic applications need to invert $f_{\mathbf{A}}$ and/or $g_{\mathbf{A}}$.

Invert $\mathbf{u} = f_{\mathbf{A}}(\mathbf{x}') = \mathbf{A}\mathbf{x}' \bmod q$:

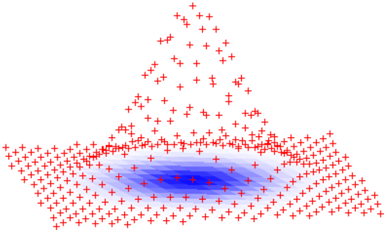
sample random $\mathbf{x} \leftarrow f_{\mathbf{A}}^{-1}(\mathbf{u})$

with prob $\propto \exp(-\|\mathbf{x}\|^2/s^2)$.

Invert $g_{\mathbf{A}}(\mathbf{s}, \mathbf{e}) = \mathbf{s}^t\mathbf{A} + \mathbf{e}^t \bmod q$:

find the unique preimage $\mathbf{s}$

(equivalently, $\mathbf{e}$)



Trapdoor Inversion

- Many cryptographic applications need to invert $f_{\mathbf{A}}$ and/or $g_{\mathbf{A}}$.

Invert $\mathbf{u} = f_{\mathbf{A}}(\mathbf{x}') = \mathbf{A}\mathbf{x}' \bmod q$:

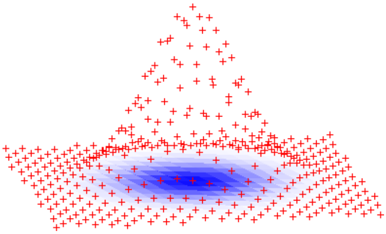
sample random $\mathbf{x} \leftarrow f_{\mathbf{A}}^{-1}(\mathbf{u})$
with prob $\propto \exp(-\|\mathbf{x}\|^2/s^2)$.

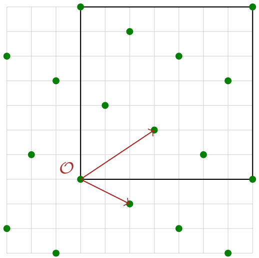
Invert $g_{\mathbf{A}}(\mathbf{s},\mathbf{e}) = \mathbf{s}^t\mathbf{A} + \mathbf{e}^t \bmod q$:

find the unique preimage $\mathbf{s}$
(equivalently, $\mathbf{e}$)

- How? Use a "strong trapdoor" for $\mathbf{A}$: a short basis of $\Lambda^{\perp}(\mathbf{A})$

[Babai'86,GGH'97,Klein'01,GPV'08,P'10]





Applications of Strong Trapdoors

Canonical App: [GPV'08] Signatures

- $pk = \mathbf{A}$, $sk =$ short basis for $\mathbf{A}$, random oracle $H \colon \{0,1\}^* \to \mathbb{Z}_q^n$.

Applications of Strong Trapdoors

Canonical App: [GPV'08] Signatures

- $pk = \mathbf{A}$, $sk =$ short basis for $\mathbf{A}$, random oracle $H \colon \{0,1\}^* \to \mathbb{Z}_q^n$.
- Sign(m): let $\mathbf{u} = H(m)$ and output Gaussian $\mathbf{x} \leftarrow f_{\mathbf{A}}^{-1}(\mathbf{u})$

Applications of Strong Trapdoors

Canonical App: [GPV'08] Signatures

- $pk = \mathbf{A}$, $sk =$ short basis for $\mathbf{A}$, random oracle $H \colon \{0,1\}^* \to \mathbb{Z}_q^n$.
- Sign(m): let $\mathbf{u} = H(m)$ and output Gaussian $\mathbf{x} \leftarrow f_{\mathbf{A}}^{-1}(\mathbf{u})$
- Verify($m, \mathbf{x}$): check $f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x} = H(m)$ and $\mathbf{x}$ "short enough"

Applications of Strong Trapdoors

Canonical App: [GPV'08] Signatures

- $pk = \mathbf{A}$, $sk =$ short basis for $\mathbf{A}$, random oracle $H \colon \{0,1\}^* \to \mathbb{Z}_q^n$.
- Sign(m): let $\mathbf{u} = H(m)$ and output Gaussian $\mathbf{x} \leftarrow f_{\mathbf{A}}^{-1}(\mathbf{u})$
- Verify($m, \mathbf{x}$): check $f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x} = H(m)$ and $\mathbf{x}$ "short enough"
- Security: finding "short enough" preimages in $f_{\mathbf{A}}$ must be hard

Applications of Strong Trapdoors

Canonical App: [GPV'08] Signatures

- $pk = \mathbf{A}$, $sk =$ short basis for $\mathbf{A}$, random oracle $H \colon \{0,1\}^* \to \mathbb{Z}_q^n$.
- Sign(m): let $\mathbf{u} = H(m)$ and output Gaussian $\mathbf{x} \leftarrow f_{\mathbf{A}}^{-1}(\mathbf{u})$
- Verify($m, \mathbf{x}$): check $f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x} = H(m)$ and $\mathbf{x}$ "short enough"
- Security: finding "short enough" preimages in $f_{\mathbf{A}}$ must be hard

Other "Black-Box" Applications of f^{-1}, g^{-1}

- Standard model signatures [CHKP'10,R'10,B'10]
- CCA-secure encryption [PW'08,P'09]
- (Hierarchical) ID-based encryption [GPV'08,CHKP'10,ABB'10a,ABB'10b]
- Much more:
  [PVW'08,PV'08,GHV'10,GKV'10,BF'10a,BF'10b,OPW'11,AFV'11,ABVVW'11,...]

Applications of Strong Trapdoors

Canonical App: [GPV'08] Signatures

- $pk = \mathbf{A}$, $sk =$ short basis for $\mathbf{A}$, random oracle $H\colon \{0,1\}^* \to \mathbb{Z}_q^n$.
- Sign(m): let $\mathbf{u} = H(m)$ and output Gaussian $\mathbf{x} \leftarrow f_{\mathbf{A}}^{-1}(\mathbf{u})$
- Verify($m, \mathbf{x}$): check $f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x} = H(m)$ and $\mathbf{x}$ "short enough"
- Security: finding "short enough" preimages in $f_{\mathbf{A}}$ must be hard

Some Drawbacks...

✗ Generating $\mathbf{A}$ w/ short basis is complicated and slow [Ajtai'99,AP'09]

Applications of Strong Trapdoors

Canonical App: [GPV'08] Signatures

- $pk = \mathbf{A}$, $sk =$ short basis for $\mathbf{A}$, random oracle $H\colon \{0,1\}^* \to \mathbb{Z}_q^n$.
- Sign(m): let $\mathbf{u} = H(m)$ and output Gaussian $\mathbf{x} \leftarrow f_{\mathbf{A}}^{-1}(\mathbf{u})$
- Verify($m, \mathbf{x}$): check $f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x} = H(m)$ and $\mathbf{x}$ "short enough"
- <u>Security</u>: finding "short enough" preimages in $f_{\mathbf{A}}$ must be hard

Some Drawbacks. . .

- ✗ Generating $\mathbf{A}$ w/ short basis is complicated and slow [Ajtai'99,AP'09]
- ✗ Known algorithms trade quality for efficiency

Applications of Strong Trapdoors

Canonical App: [GPV'08] Signatures

- $pk = \mathbf{A}$, $sk =$ short basis for $\mathbf{A}$, random oracle $H \colon \{0,1\}^* \to \mathbb{Z}_q^n$.
- Sign(m): let $\mathbf{u} = H(m)$ and output Gaussian $\mathbf{x} \leftarrow f_{\mathbf{A}}^{-1}(\mathbf{u})$
- Verify($m, \mathbf{x}$): check $f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x} = H(m)$ and $\mathbf{x}$ "short enough"
- Security: finding "short enough" preimages in $f_{\mathbf{A}}$ must be hard

Some Drawbacks...

✗ Generating $\mathbf{A}$ w/ short basis is complicated and slow [Ajtai'99,AP'09]

✗ Known algorithms trade quality for efficiency

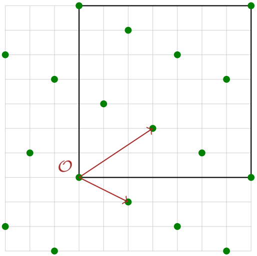
$g_{\mathbf{A}}^{-1}$: [Babai'86] (tight,iterative,fp) vs [Babai'86] (looser,parallel,offline)

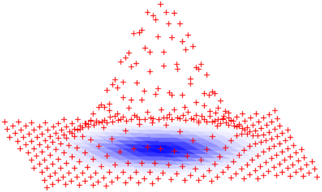
$f_{\mathbf{A}}^{-1}$: [Klein'01,GPV'08] (ditto) vs [P'10] (ditto)

Taming the Parameters

$$n\left\{\underbrace{\begin{bmatrix}\cdots & \mathbf{A} & \cdots\end{bmatrix}}_{m}\right.$$

$$f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x}$$

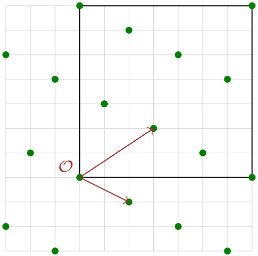


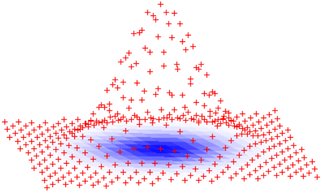


Taming the Parameters

$$n\left\{\underbrace{\begin{bmatrix} \cdots & \mathbf{A} & \cdots \end{bmatrix}}_{m}\right.$$

$$f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x}$$



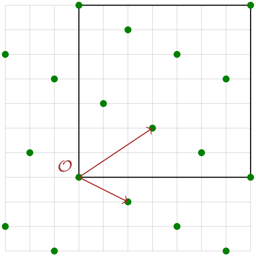


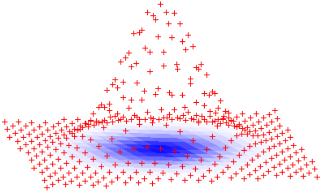
1. Trapdoor construction yields some lattice dim $m = \Omega(n \log q)$.

Taming the Parameters

$$n\left\{\underbrace{\begin{bmatrix} \cdots & \mathbf{A} & \cdots \end{bmatrix}}_{m}\right.$$

$$f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x}$$



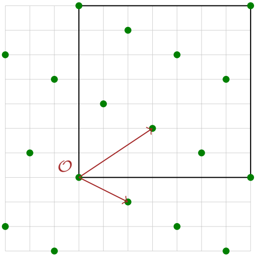


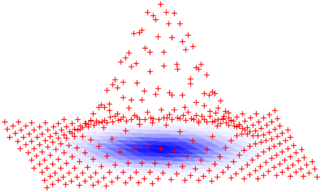
1. Trapdoor construction yields some lattice dim $m = \Omega(n \log q)$.
2. Basis "quality" $\approx$ lengths of basis vectors $\approx$ Gaussian std dev s.

Taming the Parameters

$$n\left\{\underbrace{\begin{bmatrix} \cdots & \mathbf{A} & \cdots \end{bmatrix}}_{m}\right.$$

$$f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x}$$



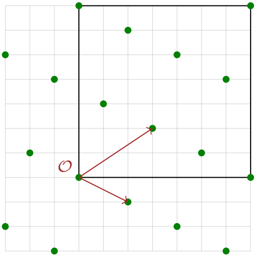


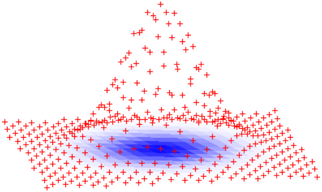
1. Trapdoor construction yields some lattice dim $m = \Omega(n \log q)$.
2. Basis "quality" $\approx$ lengths of basis vectors $\approx$ Gaussian std dev s.
3. Dimension m, std dev $s \implies$ preimage length $\beta = \|\mathbf{x}\| \approx s\sqrt{m}$.

Taming the Parameters

$$n\left\{\underbrace{\begin{bmatrix} \cdots & \mathbf{A} & \cdots \end{bmatrix}}_{m}\right.$$

$$f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x}$$



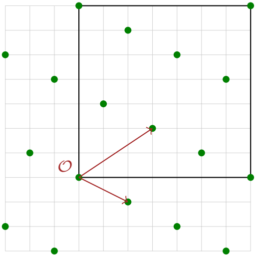


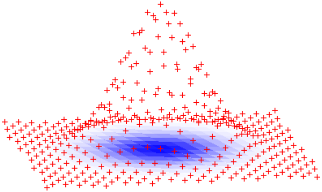
1. Trapdoor construction yields some lattice dim $m = \Omega(n \log q)$.
2. Basis "quality" $\approx$ lengths of basis vectors $\approx$ Gaussian std dev s.
3. Dimension m, std dev $s \Longrightarrow$ preimage length $\beta = \|\mathbf{x}\| \approx s\sqrt{m}$.
4. Choose n, q so that finding β-bounded preimages is hard.

Taming the Parameters

$$n\left\{\underbrace{\begin{bmatrix} \cdots & \mathbf{A} & \cdots \end{bmatrix}}_{m}\right.$$

$$f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x}$$





1. Trapdoor construction yields some lattice dim $m = \Omega(n \log q)$.
2. Basis "quality" $\approx$ lengths of basis vectors $\approx$ Gaussian std dev s.
3. Dimension m, std dev $s \implies$ preimage length $\beta = \|\mathbf{x}\| \approx s\sqrt{m}$.
4. Choose n, q so that finding β-bounded preimages is hard.

✔ Better dimension m & quality s

$\implies$ "win-win-win" in security-keysize-runtime

Our Contributions

New “strong” trapdoor generation and inversion algorithms:

Our Contributions

New "strong" trapdoor generation and inversion algorithms:

✔ Very simple & fast

- ★ Generation: one matrix mult. No HNF or inverses (cf. [A'99,AP'09])
- ★ Inversion: practical, parallel, & mostly offline
- ★ No more efficiency-vs-quality tradeoff

Our Contributions

New “strong” trapdoor generation and inversion algorithms:

✔ Very simple & fast

- ⋆ Generation: one matrix mult. No HNF or inverses (cf. [A’99,AP’09])
- ⋆ Inversion: practical, parallel, & mostly offline
- ⋆ No more efficiency-vs-quality tradeoff

✔ Tighter parameters m and s

- ⋆ Asymptotically optimal with small constant factors
- ⋆ Ex improvement: 32x in dim m, 25x in quality $s \Rightarrow$ 67x in keysize

Our Contributions

New "strong" trapdoor generation and inversion algorithms:

✔ Very simple & fast

- ⋆ Generation: one matrix mult. No HNF or inverses (cf. [A'99,AP'09])
- ⋆ Inversion: practical, parallel, & mostly offline
- ⋆ No more efficiency-vs-quality tradeoff

✔ Tighter parameters m and s

- ⋆ Asymptotically optimal with small constant factors
- ⋆ Ex improvement: 32x in dim m, 25x in quality $s \Rightarrow$ 67x in keysize

✔ New kind of trapdoor — not a basis! (But just as powerful.)

- ⋆ Half the dimension of a basis $\Rightarrow$ 4x size improvement
- ⋆ Delegation: size grows as $O(\text{dim})$, versus $O(\text{dim}^2)$ [CHKP'10]

Our Contributions

New “strong” trapdoor generation and inversion algorithms:

- ✔ Very simple & fast
  - ⋆ Generation: one matrix mult. No HNF or inverses (cf. [A’99,AP’09])
  - ⋆ Inversion: practical, parallel, & mostly offline
  - ⋆ No more efficiency-vs-quality tradeoff
- ✔ Tighter parameters m and s
  - ⋆ Asymptotically optimal with small constant factors
  - ⋆ Ex improvement: 32x in dim m, 25x in quality $s \Rightarrow$ 67x in keysize
- ✔ New kind of trapdoor — not a basis! (But just as powerful.)
  - ⋆ Half the dimension of a basis $\Rightarrow$ 4x size improvement
  - ⋆ Delegation: size grows as $O(\text{dim})$, versus $O(\text{dim}^2)$ [CHKP’10]
- ✔ More efficient applications (beyond “black-box” improvements)

Concrete Parameter Improvements

	Before [AP'09]	**Now** (fast f^{-1})	**Improvement**
Dim m	slow f^{-1}: $> 5n \log q$ fast f^{-1}: $> n \log^2 q$	$2n \log q$ ($\overset{s}{\approx}$) $n(1+\log q)$ ($\overset{c}{\approx}$)	$2.5 - \log q$

Concrete Parameter Improvements

	Before [AP'09]	**Now** (fast f^{-1})	**Improvement**
Dim m	slow f^{-1}: $> 5n \log q$ fast f^{-1}: $> n \log^2 q$	$2n \log q$ ($\stackrel{s}{\approx}$) $n(1 + \log q)$ ($\stackrel{c}{\approx}$)	$2.5 - \log q$
Quality s	slow f^{-1}: $20\sqrt{n \log q}$ fast f^{-1}: $16\sqrt{n \log^2 q}$	$1.6\sqrt{n \log q}$	$12.5 - 10\sqrt{\log q}$

Concrete Parameter Improvements

	Before [AP'09]	**Now** (fast f^{-1})	**Improvement**
Dim m	slow f^{-1}: $> 5n \log q$ fast f^{-1}: $> n \log^2 q$	$2n \log q$ ($\stackrel{s}{\approx}$) $n(1 + \log q)$ ($\stackrel{c}{\approx}$)	$2.5 - \log q$
Quality s	slow f^{-1}: $20\sqrt{n \log q}$ fast f^{-1}: $16\sqrt{n \log^2 q}$	$1.6\sqrt{n \log q}$	$12.5 - 10\sqrt{\log q}$

Example parameters for (ring-based) GPV signatures:

	n	q	δ to break	pk size (bits)
Before (fast f^{-1})	436	2^{32}	1.007	$\approx 17 \times 10^6$
Now	284	2^{24}	1.007	$\approx 360 \times 10^3$

Bottom line: $\approx$ 45-fold improvement in key size.

Overview of Methods

1. Design a fixed, public lattice defined by "gadget" $\mathbf{G}$.

   Give fast, parallel, offline algorithms for $f_{\mathbf{G}}^{-1}$, $g_{\mathbf{G}}^{-1}$.

Overview of Methods

1. Design a fixed, public lattice defined by "gadget" $\mathbf{G}$.

   Give fast, parallel, offline algorithms for $f_{\mathbf{G}}^{-1}$, $g_{\mathbf{G}}^{-1}$.

2. Randomize $\mathbf{G} \leftrightarrow \mathbf{A}$ via a "nice" unimodular transformation.

   (The transformation is the trapdoor!)

Overview of Methods

1. Design a fixed, public lattice defined by “gadget” $\mathbf{G}$.
   Give fast, parallel, offline algorithms for $f_{\mathbf{G}}^{-1}$, $g_{\mathbf{G}}^{-1}$.

2. Randomize $\mathbf{G} \leftrightarrow \mathbf{A}$ via a “nice” unimodular transformation.
   (The transformation is the trapdoor!)

3. Reduce $f_{\mathbf{A}}^{-1}, g_{\mathbf{A}}^{-1}$ to $f_{\mathbf{G}}^{-1}, g_{\mathbf{G}}^{-1}$ plus pre-/post-processing.

Step 1: Gadget $\mathbf{G}$ and Inversion Algorithms

- Let $q = 2^k$. Define 1-by-k "parity check" vector

$$\mathbf{g} := \begin{bmatrix} 1 & 2 & 4 & \cdots & 2^{k-1} \end{bmatrix} \in \mathbb{Z}_q^{1\times k}.$$

Step 1: Gadget $\mathbf{G}$ and Inversion Algorithms

- Let $q = 2^k$. Define 1-by-k "parity check" vector

$$\mathbf{g} := \begin{bmatrix} 1 & 2 & 4 & \cdots & 2^{k-1} \end{bmatrix} \in \mathbb{Z}_q^{1 \times k}.$$

- Invert LWE function $g_{\mathbf{g}} \colon \mathbb{Z}_q \times \mathbb{Z}^k \to \mathbb{Z}_q^k$

$$g_{\mathbf{g}}(s, \mathbf{e}) := s \cdot \mathbf{g} + \mathbf{e} = \begin{bmatrix} s + e_0 & 2s + e_1 & \cdots & 2^{k-1}s + e_{k-1} \end{bmatrix} \bmod q.$$

Step 1: Gadget $\mathbf{G}$ and Inversion Algorithms

- Let $q = 2^k$. Define 1-by-k "parity check" vector

$$\mathbf{g} := \begin{bmatrix} 1 & 2 & 4 & \cdots & 2^{k-1} \end{bmatrix} \in \mathbb{Z}_q^{1 \times k}.$$

- Invert LWE function $g_{\mathbf{g}} \colon \mathbb{Z}_q \times \mathbb{Z}^k \to \mathbb{Z}_q^k$

$$g_{\mathbf{g}}(s, \mathbf{e}) := s \cdot \mathbf{g} + \mathbf{e} = \begin{bmatrix} s + e_0 & 2s + e_1 & \cdots & 2^{k-1}s + e_{k-1} \end{bmatrix} \bmod q.$$

  - ⋆ Get lsb(s), e_{k-1} from $2^{k-1}s + e_{k-1}$. Then get next bit of s, etc.
    Works exactly when $\mathbf{e} \in [-\frac{q}{4}, \frac{q}{4})^k$.

Step 1: Gadget $\mathbf{G}$ and Inversion Algorithms

- Let $q = 2^k$. Define 1-by-k "parity check" vector

$$\mathbf{g} := \begin{bmatrix} 1 & 2 & 4 & \cdots & 2^{k-1} \end{bmatrix} \in \mathbb{Z}_q^{1 \times k}.$$

- Invert LWE function $g_{\mathbf{g}} \colon \mathbb{Z}_q \times \mathbb{Z}^k \to \mathbb{Z}_q^k$

$$g_{\mathbf{g}}(s, \mathbf{e}) := s \cdot \mathbf{g} + \mathbf{e} = \begin{bmatrix} s + e_0 & 2s + e_1 & \cdots & 2^{k-1}s + e_{k-1} \end{bmatrix} \bmod q.$$

  - ★ Get lsb(s), e_{k-1} from $2^{k-1}s + e_{k-1}$. Then get next bit of s, etc. Works exactly when $\mathbf{e} \in [-\frac{q}{4}, \frac{q}{4})^k$.
  - ★ <u>OR</u> round to $\frac{q}{8}$-multiple and lookup in size-q^3 table.

Step 1: Gadget $\mathbf{G}$ and Inversion Algorithms

- Let $q = 2^k$. Define 1-by-k “parity check” vector

$$\mathbf{g} := \begin{bmatrix} 1 & 2 & 4 & \cdots & 2^{k-1} \end{bmatrix} \in \mathbb{Z}_q^{1\times k}.$$

- Invert LWE function $g_{\mathbf{g}} \colon \mathbb{Z}_q \times \mathbb{Z}^k \to \mathbb{Z}_q^k$

$$g_{\mathbf{g}}(s, \mathbf{e}) := s \cdot \mathbf{g} + \mathbf{e} = \begin{bmatrix} s + e_0 & 2s + e_1 & \cdots & 2^{k-1}s + e_{k-1} \end{bmatrix} \bmod q.$$

  - ⋆ Get $\mathrm{lsb}(s)$, e_{k-1} from $2^{k-1}s + e_{k-1}$. Then get next bit of s, etc. Works exactly when $\mathbf{e} \in [-\frac{q}{4}, \frac{q}{4})^k$.
  - ⋆ OR round to $\frac{q}{8}$-multiple and lookup in size-q^3 table.
  - ⋆ OR a hybrid of the two approaches.

Step 1: Gadget $\mathbf{G}$ and Inversion Algorithms

- Let $q = 2^k$. Define 1-by-k "parity check" vector

$$\mathbf{g} := \begin{bmatrix} 1 & 2 & 4 & \cdots & 2^{k-1} \end{bmatrix} \in \mathbb{Z}_q^{1\times k}.$$

- Invert LWE function $g_{\mathbf{g}} : \mathbb{Z}_q \times \mathbb{Z}^k \to \mathbb{Z}_q^k$

$$g_{\mathbf{g}}(s, \mathbf{e}) := s \cdot \mathbf{g} + \mathbf{e} = \begin{bmatrix} s + e_0 & 2s + e_1 & \cdots & 2^{k-1}s + e_{k-1} \end{bmatrix} \bmod q.$$

  - ⋆ Get lsb(s), e_{k-1} from $2^{k-1}s + e_{k-1}$. Then get next bit of s, etc. Works exactly when $\mathbf{e} \in [-\frac{q}{4}, \frac{q}{4})^k$.
  - ⋆ OR round to $\frac{q}{8}$-multiple and lookup in size-q^3 table.
  - ⋆ OR a hybrid of the two approaches.

- Sample Gaussian preimage for $u = f_{\mathbf{g}}(\mathbf{x}) := \langle \mathbf{g}, \mathbf{x} \rangle \bmod q$.

Step 1: Gadget $\mathbf{G}$ and Inversion Algorithms

- Let $q = 2^k$. Define 1-by-k "parity check" vector

$$\mathbf{g} := \begin{bmatrix} 1 & 2 & 4 & \cdots & 2^{k-1} \end{bmatrix} \in \mathbb{Z}_q^{1 \times k}.$$

- Invert LWE function $g_{\mathbf{g}} \colon \mathbb{Z}_q \times \mathbb{Z}^k \to \mathbb{Z}_q^k$

$$g_{\mathbf{g}}(s, \mathbf{e}) := s \cdot \mathbf{g} + \mathbf{e} = \begin{bmatrix} s + e_0 & 2s + e_1 & \cdots & 2^{k-1}s + e_{k-1} \end{bmatrix} \bmod q.$$

  - ⋆ Get lsb(s), e_{k-1} from $2^{k-1}s + e_{k-1}$. Then get next bit of s, etc. Works exactly when $\mathbf{e} \in [-\frac{q}{4}, \frac{q}{4})^k$.
  - ⋆ <u>OR</u> round to $\frac{q}{8}$-multiple and lookup in size-q^3 table.
  - ⋆ <u>OR</u> a hybrid of the two approaches.

- Sample Gaussian preimage for $u = f_{\mathbf{g}}(\mathbf{x}) := \langle \mathbf{g}, \mathbf{x} \rangle \bmod q$.
  - ⋆ For $i \leftarrow 0, \ldots, k-1$: choose $x_i \leftarrow (2\mathbb{Z} + u)$, let $u \leftarrow (u - x_i)/2 \in \mathbb{Z}$.

Step 1: Gadget $\mathbf{G}$ and Inversion Algorithms

- Let $q = 2^k$. Define 1-by-k "parity check" vector

$$\mathbf{g} := \begin{bmatrix} 1 & 2 & 4 & \cdots & 2^{k-1} \end{bmatrix} \in \mathbb{Z}_q^{1\times k}.$$

- Invert LWE function $g_{\mathbf{g}} \colon \mathbb{Z}_q \times \mathbb{Z}^k \to \mathbb{Z}_q^k$

$$g_{\mathbf{g}}(s, \mathbf{e}) := s \cdot \mathbf{g} + \mathbf{e} = \begin{bmatrix} s + e_0 & 2s + e_1 & \cdots & 2^{k-1}s + e_{k-1} \end{bmatrix} \bmod q.$$

  - ★ Get $\mathrm{lsb}(s)$, e_{k-1} from $2^{k-1}s + e_{k-1}$. Then get next bit of s, etc. Works exactly when $\mathbf{e} \in [-\frac{q}{4}, \frac{q}{4})^k$.
  - ★ OR round to $\frac{q}{8}$-multiple and lookup in size-q^3 table.
  - ★ OR a hybrid of the two approaches.

- Sample Gaussian preimage for $u = f_{\mathbf{g}}(\mathbf{x}) := \langle \mathbf{g}, \mathbf{x} \rangle \bmod q$.
  - ★ For $i \leftarrow 0, \ldots, k-1$: choose $x_i \leftarrow (2\mathbb{Z} + u)$, let $u \leftarrow (u - x_i)/2 \in \mathbb{Z}$.
  - ★ OR presample many $\mathbf{x} \leftarrow \mathbb{Z}^k$ and store in 'buckets' $f_{\mathbf{g}}(\mathbf{x})$ for later.

Step 1: Gadget $\mathbf{G}$ and Inversion Algorithms

- Let $q = 2^k$. Define 1-by-k "parity check" vector

$$\mathbf{g} := \begin{bmatrix} 1 & 2 & 4 & \cdots & 2^{k-1} \end{bmatrix} \in \mathbb{Z}_q^{1\times k}.$$

- Invert LWE function $g_{\mathbf{g}} \colon \mathbb{Z}_q \times \mathbb{Z}^k \to \mathbb{Z}_q^k$

$$g_{\mathbf{g}}(s, \mathbf{e}) := s \cdot \mathbf{g} + \mathbf{e} = \begin{bmatrix} s + e_0 & 2s + e_1 & \cdots & 2^{k-1}s + e_{k-1} \end{bmatrix} \bmod q.$$

  - ⋆ Get $\mathrm{lsb}(s)$, e_{k-1} from $2^{k-1}s + e_{k-1}$. Then get next bit of s, etc. Works exactly when $\mathbf{e} \in [-\frac{q}{4}, \frac{q}{4})^k$.
  - ⋆ OR round to $\frac{q}{8}$-multiple and lookup in size-q^3 table.
  - ⋆ OR a hybrid of the two approaches.

- Sample Gaussian preimage for $u = f_{\mathbf{g}}(\mathbf{x}) := \langle \mathbf{g}, \mathbf{x} \rangle \bmod q$.
  - ⋆ For $i \leftarrow 0, \ldots, k-1$: choose $x_i \leftarrow (2\mathbb{Z} + u)$, let $u \leftarrow (u - x_i)/2 \in \mathbb{Z}$.
  - ⋆ OR presample many $\mathbf{x} \leftarrow \mathbb{Z}^k$ and store in 'buckets' $f_{\mathbf{g}}(\mathbf{x})$ for later.
  - ⋆ OR a hybrid of the two approaches.

Step 1: Gadget $\mathbf{G}$ and Inversion Algorithms

- Another view: for $\mathbf{g} = \begin{bmatrix} 1 & 2 & \cdots & 2^{k-1} \end{bmatrix}$ the lattice $\Lambda^\perp(\mathbf{g})$ has basis

$$\mathbf{S} = \begin{bmatrix} 2 & & & & \\ -1 & 2 & & & \\ & -1 & \ddots & & \\ & & & 2 & \\ & & & -1 & 2 \end{bmatrix} \in \mathbb{Z}^{k \times k}, \quad \text{with } \widetilde{\mathbf{S}} = 2 \cdot \mathbf{I}_k.$$

Step 1: Gadget $\mathbf{G}$ and Inversion Algorithms

- Another view: for $\mathbf{g} = \begin{bmatrix} 1 & 2 & \cdots & 2^{k-1} \end{bmatrix}$ the lattice $\Lambda^\perp(\mathbf{g})$ has basis

$$\mathbf{S} = \begin{bmatrix} 2 & & & & \\ -1 & 2 & & & \\ & -1 & \ddots & & \\ & & & 2 & \\ & & & -1 & 2 \end{bmatrix} \in \mathbb{Z}^{k \times k}, \quad \text{with } \widetilde{\mathbf{S}} = 2 \cdot \mathbf{I}_k.$$

The iterative inversion algorithms for $f_{\mathbf{g}}$, $g_{\mathbf{g}}$ are special cases of the (randomized) "nearest-plane" algorithm [Babai'86,Klein'01,GPV'08].

Step 1: Gadget $\mathbf{G}$ and Inversion Algorithms

- Another view: for $\mathbf{g} = \begin{bmatrix} 1 & 2 & \cdots & 2^{k-1} \end{bmatrix}$ the lattice $\Lambda^\perp(\mathbf{g})$ has basis

$$\mathbf{S} = \begin{bmatrix} 2 & & & & \\ -1 & 2 & & & \\ & -1 & \ddots & & \\ & & & 2 & \\ & & & -1 & 2 \end{bmatrix} \in \mathbb{Z}^{k \times k}, \quad \text{with } \widetilde{\mathbf{S}} = 2 \cdot \mathbf{I}_k.$$

  The iterative inversion algorithms for $f_{\mathbf{g}}$, $g_{\mathbf{g}}$ are special cases of the (randomized) "nearest-plane" algorithm [Babai'86,Klein'01,GPV'08].

- Define $\mathbf{G} = \mathbf{I}_n \otimes \mathbf{g} = \begin{bmatrix} \cdots \mathbf{g} \cdots & & & \\ & \cdots \mathbf{g} \cdots & & \\ & & \ddots & \\ & & & \cdots \mathbf{g} \cdots \end{bmatrix} \in \mathbb{Z}_q^{n \times nk}$.

Step 1: Gadget $\mathbf{G}$ and Inversion Algorithms

- Another view: for $\mathbf{g} = \begin{bmatrix} 1 & 2 & \cdots & 2^{k-1} \end{bmatrix}$ the lattice $\Lambda^{\perp}(\mathbf{g})$ has basis

$$\mathbf{S} = \begin{bmatrix} 2 & & & & \\ -1 & 2 & & & \\ & -1 & \ddots & & \\ & & & 2 & \\ & & & -1 & 2 \end{bmatrix} \in \mathbb{Z}^{k \times k}, \quad \text{with } \widetilde{\mathbf{S}} = 2 \cdot \mathbf{I}_k.$$

The iterative inversion algorithms for $f_{\mathbf{g}}$, $g_{\mathbf{g}}$ are special cases of the (randomized) "nearest-plane" algorithm [Babai'86,Klein'01,GPV'08].

- Define $\mathbf{G} = \mathbf{I}_n \otimes \mathbf{g} = \begin{bmatrix} \cdots \mathbf{g} \cdots & & & \\ & \cdots \mathbf{g} \cdots & & \\ & & \ddots & \\ & & & \cdots \mathbf{g} \cdots \end{bmatrix} \in \mathbb{Z}_q^{n \times nk}$.

Now $f_{\mathbf{G}}^{-1}$, $g_{\mathbf{G}}^{-1}$ reduce to n parallel (and offline) calls to $f_{\mathbf{g}}^{-1}$, $g_{\mathbf{g}}^{-1}$.

Step 1: Gadget $\mathbf{G}$ and Inversion Algorithms

- Another view: for $\mathbf{g} = \begin{bmatrix} 1 & 2 & \cdots & 2^{k-1} \end{bmatrix}$ the lattice $\Lambda^{\perp}(\mathbf{g})$ has basis

$$\mathbf{S} = \begin{bmatrix} 2 & & & & \\ -1 & 2 & & & \\ & -1 & \ddots & & \\ & & & 2 & \\ & & & -1 & 2 \end{bmatrix} \in \mathbb{Z}^{k \times k}, \quad \text{with } \widetilde{\mathbf{S}} = 2 \cdot \mathbf{I}_k.$$

  The iterative inversion algorithms for $f_{\mathbf{g}}$, $g_{\mathbf{g}}$ are special cases of the (randomized) "nearest-plane" algorithm [Babai'86,Klein'01,GPV'08].

- Define $\mathbf{G} = \mathbf{I}_n \otimes \mathbf{g} = \begin{bmatrix} \cdots \mathbf{g} \cdots & & & \\ & \cdots \mathbf{g} \cdots & & \\ & & \ddots & \\ & & & \cdots \mathbf{g} \cdots \end{bmatrix} \in \mathbb{Z}_q^{n \times nk}$.

  Now $f_{\mathbf{G}}^{-1}$, $g_{\mathbf{G}}^{-1}$ reduce to n parallel (and offline) calls to $f_{\mathbf{g}}^{-1}$, $g_{\mathbf{g}}^{-1}$.

  Also applies to $\mathbf{H} \cdot \mathbf{G}$ for any invertible $\mathbf{H} \in \mathbb{Z}_q^{n \times n}$.

Step 2: Randomize $\mathbf{G} \leftrightarrow \mathbf{A}$

1. Define semi-random $[\bar{\mathbf{A}} \mid \mathbf{G}]$ for uniform (universal) $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n \times \bar{m}}$.

   (Computing f^{-1}, g^{-1} easily reduce to $f_{\mathbf{G}}^{-1}$, $g_{\mathbf{G}}^{-1}$.)

Step 2: Randomize $\mathbf{G} \leftrightarrow \mathbf{A}$

1. Define semi-random $[\bar{\mathbf{A}} \mid \mathbf{G}]$ for uniform (universal) $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n \times \bar{m}}$.

   (Computing f^{-1}, g^{-1} easily reduce to $f_{\mathbf{G}}^{-1}$, $g_{\mathbf{G}}^{-1}$.)

2. Choose "short" (Gaussian) $\mathbf{R} \leftarrow \mathbb{Z}^{\bar{m} \times n \log q}$ and let

$$\mathbf{A} := [\bar{\mathbf{A}} \mid \mathbf{G}] \underbrace{\begin{bmatrix} \mathbf{I} & -\mathbf{R} \\ & \mathbf{I} \end{bmatrix}}_{\text{unimodular}} = [\bar{\mathbf{A}} \mid \mathbf{G} - \bar{\mathbf{A}}\mathbf{R}].$$

Step 2: Randomize $\mathbf{G} \leftrightarrow \mathbf{A}$

1. Define semi-random $[\bar{\mathbf{A}} \mid \mathbf{G}]$ for uniform (universal) $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n \times \bar{m}}$.

   (Computing f^{-1}, g^{-1} easily reduce to $f_{\mathbf{G}}^{-1}$, $g_{\mathbf{G}}^{-1}$.)

2. Choose "short" (Gaussian) $\mathbf{R} \leftarrow \mathbb{Z}^{\bar{m} \times n \log q}$ and let

$$\mathbf{A} := [\bar{\mathbf{A}} \mid \mathbf{G}] \underbrace{\begin{bmatrix} \mathbf{I} & -\mathbf{R} \\ & \mathbf{I} \end{bmatrix}}_{\text{unimodular}} = [\bar{\mathbf{A}} \mid \mathbf{G} - \bar{\mathbf{A}}\mathbf{R}].$$

   ⋆ $\mathbf{A}$ is uniform if $[\bar{\mathbf{A}} \mid \bar{\mathbf{A}}\mathbf{R}]$ is: leftover hash lemma for $\bar{m} \approx n \log q$.

Step 2: Randomize $\mathbf{G} \leftrightarrow \mathbf{A}$

1. Define semi-random $[\bar{\mathbf{A}} \mid \mathbf{G}]$ for uniform (universal) $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n \times \bar{m}}$.

   (Computing f^{-1}, g^{-1} easily reduce to $f_{\mathbf{G}}^{-1}$, $g_{\mathbf{G}}^{-1}$.)

2. Choose "short" (Gaussian) $\mathbf{R} \leftarrow \mathbb{Z}^{\bar{m} \times n \log q}$ and let

$$\mathbf{A} := [\bar{\mathbf{A}} \mid \mathbf{G}] \underbrace{\begin{bmatrix} \mathbf{I} & -\mathbf{R} \\ & \mathbf{I} \end{bmatrix}}_{\text{unimodular}} = [\bar{\mathbf{A}} \mid \mathbf{G} - \bar{\mathbf{A}}\mathbf{R}].$$

   ⋆ $\mathbf{A}$ is uniform if $[\bar{\mathbf{A}} \mid \bar{\mathbf{A}}\mathbf{R}]$ is: leftover hash lemma for $\bar{m} \approx n \log q$.

   With $\mathbf{G} = \mathbf{0}$, we get Ajtai's original method for constructing $\mathbf{A}$ with a "weak" trapdoor of ≥ 1 short vector (but not a full basis).

Step 2: Randomize $\mathbf{G} \leftrightarrow \mathbf{A}$

1. Define semi-random $[\bar{\mathbf{A}} \mid \mathbf{G}]$ for uniform (universal) $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n \times \bar{m}}$.

   (Computing f^{-1}, g^{-1} easily reduce to $f_{\mathbf{G}}^{-1}$, $g_{\mathbf{G}}^{-1}$.)

2. Choose "short" (Gaussian) $\mathbf{R} \leftarrow \mathbb{Z}^{\bar{m} \times n \log q}$ and let

$$\mathbf{A} := [\bar{\mathbf{A}} \mid \mathbf{G}] \underbrace{\begin{bmatrix} \mathbf{I} & -\mathbf{R} \\ & \mathbf{I} \end{bmatrix}}_{\text{unimodular}} = [\bar{\mathbf{A}} \mid \mathbf{G} - \bar{\mathbf{A}}\mathbf{R}].$$

   - ⋆ $\mathbf{A}$ is uniform if $[\bar{\mathbf{A}} \mid \bar{\mathbf{A}}\mathbf{R}]$ is: leftover hash lemma for $\bar{m} \approx n \log q$.

     With $\mathbf{G} = \mathbf{0}$, we get Ajtai's original method for constructing $\mathbf{A}$ with a "weak" trapdoor of ≥ 1 short vector (but not a full basis).

   - ⋆ $[\mathbf{I} \mid \bar{\mathbf{A}} \mid -(\bar{\mathbf{A}}\mathbf{R}_1 + \mathbf{R}_2)]$ is pseudorandom (under LWE) for $\bar{m} = n$.

A New Trapdoor Notion

- We constructed $\mathbf{A} = [\bar{\mathbf{A}} \mid \mathbf{G} - \bar{\mathbf{A}}\mathbf{R}]$.

A New Trapdoor Notion

- We constructed $\mathbf{A} = [\bar{\mathbf{A}} \mid \mathbf{G} - \bar{\mathbf{A}}\mathbf{R}]$.

Definition

- $\mathbf{R}$ is a trapdoor for $\mathbf{A}$ with tag $\mathbf{H} \in \mathbb{Z}_q^{n\times n}$ (invertible) if

$$\mathbf{A} \cdot \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} = \mathbf{H} \cdot \mathbf{G}.$$

A New Trapdoor Notion

- We constructed $\mathbf{A} = [\bar{\mathbf{A}} \mid \mathbf{G} - \bar{\mathbf{A}}\mathbf{R}]$.

Definition

- $\mathbf{R}$ is a trapdoor for $\mathbf{A}$ with tag $\mathbf{H} \in \mathbb{Z}_q^{n \times n}$ (invertible) if

$$\mathbf{A} \cdot \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} = \mathbf{H} \cdot \mathbf{G}.$$

- The quality of $\mathbf{R}$ is $s_1(\mathbf{R}) := \max_{\|\mathbf{u}\|=1} \|\mathbf{R}\mathbf{u}\|$. (smaller is better.)

A New Trapdoor Notion

- We constructed $\mathbf{A} = [\bar{\mathbf{A}} \mid \mathbf{G} - \bar{\mathbf{A}}\mathbf{R}]$.

Definition

- $\mathbf{R}$ is a trapdoor for $\mathbf{A}$ with tag $\mathbf{H} \in \mathbb{Z}_q^{n \times n}$ (invertible) if

$$\mathbf{A} \cdot \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} = \mathbf{H} \cdot \mathbf{G}.$$

- The quality of $\mathbf{R}$ is $s_1(\mathbf{R}) := \max_{\|\mathbf{u}\|=1} \|\mathbf{R}\mathbf{u}\|$. (smaller is better.)
- Fact: $s_1(\mathbf{R}) \approx (\sqrt{\text{rows}} + \sqrt{\text{cols}}) \cdot r$ for Gaussian entries w/ std dev r.

A New Trapdoor Notion

- We constructed $\mathbf{A} = [\bar{\mathbf{A}} \mid \mathbf{G} - \bar{\mathbf{A}}\mathbf{R}]$.

Definition

- $\mathbf{R}$ is a trapdoor for $\mathbf{A}$ with tag $\mathbf{H} \in \mathbb{Z}_q^{n \times n}$ (invertible) if

$$\mathbf{A} \cdot \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} = \mathbf{H} \cdot \mathbf{G}.$$

- The quality of $\mathbf{R}$ is $s_1(\mathbf{R}) := \max_{\|\mathbf{u}\|=1} \|\mathbf{R}\mathbf{u}\|$. (smaller is better.)
- Fact: $s_1(\mathbf{R}) \approx (\sqrt{\text{rows}} + \sqrt{\text{cols}}) \cdot r$ for Gaussian entries w/ std dev r.
- Note: $\mathbf{R}$ is a trapdoor for $\mathbf{A} - [\mathbf{0} \mid \mathbf{H}' \cdot \mathbf{G}]$ w/ tag $(\mathbf{H} - \mathbf{H}')$ [ABB'10].

A New Trapdoor Notion

- We constructed $\mathbf{A} = [\bar{\mathbf{A}} \mid \mathbf{G} - \bar{\mathbf{A}}\mathbf{R}]$.

Definition

- $\mathbf{R}$ is a trapdoor for $\mathbf{A}$ with tag $\mathbf{H} \in \mathbb{Z}_q^{n \times n}$ (invertible) if

$$\mathbf{A} \cdot \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} = \mathbf{H} \cdot \mathbf{G}.$$

- The quality of $\mathbf{R}$ is $s_1(\mathbf{R}) := \max_{\|\mathbf{u}\|=1} \|\mathbf{Ru}\|$. (smaller is better.)
- Fact: $s_1(\mathbf{R}) \approx (\sqrt{\text{rows}} + \sqrt{\text{cols}}) \cdot r$ for Gaussian entries w/ std dev r.
- Note: $\mathbf{R}$ is a trapdoor for $\mathbf{A} - [\mathbf{0} \mid \mathbf{H}' \cdot \mathbf{G}]$ w/ tag $(\mathbf{H} - \mathbf{H}')$ [ABB'10].

Relating New and Old Trapdoors

Given a basis $\mathbf{S}$ for $\Lambda^{\perp}(\mathbf{G})$ and a trapdoor $\mathbf{R}$ for $\mathbf{A}$,

we can efficiently construct a basis $\mathbf{S_A}$ for $\Lambda^{\perp}(\mathbf{A})$

where $\|\widetilde{\mathbf{S_A}}\| \leq (s_1(\mathbf{R}) + 1) \cdot \|\widetilde{\mathbf{S}}\|$.

A New Trapdoor Notion

- We constructed $\mathbf{A} = [\bar{\mathbf{A}} \mid \mathbf{G} - \bar{\mathbf{A}}\mathbf{R}]$.

Definition

- $\mathbf{R}$ is a trapdoor for $\mathbf{A}$ with tag $\mathbf{H} \in \mathbb{Z}_q^{n \times n}$ (invertible) if

$$\mathbf{A} \cdot \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} = \mathbf{H} \cdot \mathbf{G}.$$

- The quality of $\mathbf{R}$ is $s_1(\mathbf{R}) := \max_{\|\mathbf{u}\|=1} \|\mathbf{R}\mathbf{u}\|$. (smaller is better.)
- Fact: $s_1(\mathbf{R}) \approx (\sqrt{\text{rows}} + \sqrt{\text{cols}}) \cdot r$ for Gaussian entries w/ std dev r.
- Note: $\mathbf{R}$ is a trapdoor for $\mathbf{A} - [\mathbf{0} \mid \mathbf{H}' \cdot \mathbf{G}]$ w/ tag $(\mathbf{H} - \mathbf{H}')$ [ABB'10].

Relating New and Old Trapdoors

Given a basis $\mathbf{S}$ for $\Lambda^\perp(\mathbf{G})$ and a trapdoor $\mathbf{R}$ for $\mathbf{A}$,

we can efficiently construct a basis $\mathbf{S}_\mathbf{A}$ for $\Lambda^\perp(\mathbf{A})$

where $\|\widetilde{\mathbf{S}_\mathbf{A}}\| \le (s_1(\mathbf{R}) + 1) \cdot \|\widetilde{\mathbf{S}}\|$.

(But we'll never need to.)

Step 3: Reduce $f_{\mathbf{A}}^{-1}, g_{\mathbf{A}}^{-1}$ to $f_{\mathbf{G}}^{-1}, g_{\mathbf{G}}^{-1}$

- Suppose $\mathbf{R}$ is a trapdoor for $\mathbf{A}$ (w/ tag $\mathbf{H} = \mathbf{I}$): $\mathbf{A}\left[\begin{smallmatrix}\mathbf{R}\\\mathbf{I}\end{smallmatrix}\right] = \mathbf{G}$.

Step 3: Reduce $f_{\mathbf{A}}^{-1}, g_{\mathbf{A}}^{-1}$ to $f_{\mathbf{G}}^{-1}, g_{\mathbf{G}}^{-1}$

- Suppose $\mathbf{R}$ is a trapdoor for $\mathbf{A}$ (w/ tag $\mathbf{H} = \mathbf{I}$): $\mathbf{A}\left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right] = \mathbf{G}$.

Inverting LWE Function

Given $\mathbf{b}^t = \mathbf{s}^t\mathbf{A} + \mathbf{e}^t$, recover $\mathbf{s}$ from

$$\mathbf{b}^t\left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right] = \mathbf{s}^t\mathbf{G} + \mathbf{e}^t\left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right].$$

Works if each entry of $\mathbf{e}^t\left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right]$ in $[-\frac{q}{4}, \frac{q}{4})$, e.g. if $\|\mathbf{e}\| < q/(4s_1(\left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right]))$.

Step 3: Reduce $f_{\mathbf{A}}^{-1}, g_{\mathbf{A}}^{-1}$ to $f_{\mathbf{G}}^{-1}, g_{\mathbf{G}}^{-1}$

- Suppose $\mathbf{R}$ is a trapdoor for $\mathbf{A}$ (w/ tag $\mathbf{H} = \mathbf{I}$): $\mathbf{A}\left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right] = \mathbf{G}$.

Inverting LWE Function

Given $\mathbf{b}^t = \mathbf{s}^t\mathbf{A} + \mathbf{e}^t$, recover $\mathbf{s}$ from

$$\mathbf{b}^t\left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right] = \mathbf{s}^t\mathbf{G} + \mathbf{e}^t\left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right].$$

Works if each entry of $\mathbf{e}^t\left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right]$ in $[-\frac{q}{4}, \frac{q}{4})$, e.g. if $\|\mathbf{e}\| < q/(4s_1(\left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right]))$.

Sampling Gaussian Preimages

Given $\mathbf{u} = f_{\mathbf{A}}(\mathbf{x}') = \mathbf{A}\mathbf{x}'$, sample $\mathbf{z} \leftarrow f_{\mathbf{G}}^{-1}(\mathbf{u})$ and output $\mathbf{x} = \left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right]\mathbf{z}$?

- We have $\mathbf{A}\mathbf{x} = \mathbf{G}\mathbf{z} = \mathbf{u}$ as desired.

Step 3: Reduce $f_{\mathbf{A}}^{-1}, g_{\mathbf{A}}^{-1}$ to $f_{\mathbf{G}}^{-1}, g_{\mathbf{G}}^{-1}$

- Suppose $\mathbf{R}$ is a trapdoor for $\mathbf{A}$ (w/ tag $\mathbf{H} = \mathbf{I}$): $\mathbf{A}\left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right] = \mathbf{G}$.

Inverting LWE Function

Given $\mathbf{b}^t = \mathbf{s}^t\mathbf{A} + \mathbf{e}^t$, recover $\mathbf{s}$ from

$$\mathbf{b}^t\left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right] = \mathbf{s}^t\mathbf{G} + \mathbf{e}^t\left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right].$$

Works if each entry of $\mathbf{e}^t\left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right]$ in $[-\frac{q}{4}, \frac{q}{4})$, e.g. if $\|\mathbf{e}\| < q/(4s_1(\left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right]))$.

Sampling Gaussian Preimages

Given $\mathbf{u} = f_{\mathbf{A}}(\mathbf{x}') = \mathbf{A}\mathbf{x}'$, sample $\mathbf{z} \leftarrow f_{\mathbf{G}}^{-1}(\mathbf{u})$ and output $\mathbf{x} = \left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right]\mathbf{z}$?

- We have $\mathbf{A}\mathbf{x} = \mathbf{G}\mathbf{z} = \mathbf{u}$ as desired.
- Problem: $\left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right]\mathbf{z}$ is non-spherical Gaussian, leaks $\mathbf{R}$!

Step 3: Reduce $f_{\mathbf{A}}^{-1}, g_{\mathbf{A}}^{-1}$ to $f_{\mathbf{G}}^{-1}, g_{\mathbf{G}}^{-1}$

- Suppose $\mathbf{R}$ is a trapdoor for $\mathbf{A}$ (w/ tag $\mathbf{H} = \mathbf{I}$): $\mathbf{A}\begin{bmatrix}\mathbf{R}\\ \mathbf{I}\end{bmatrix} = \mathbf{G}$.

Inverting LWE Function

Given $\mathbf{b}^t = \mathbf{s}^t\mathbf{A} + \mathbf{e}^t$, recover $\mathbf{s}$ from

$$\mathbf{b}^t\begin{bmatrix}\mathbf{R}\\ \mathbf{I}\end{bmatrix} = \mathbf{s}^t\mathbf{G} + \mathbf{e}^t\begin{bmatrix}\mathbf{R}\\ \mathbf{I}\end{bmatrix}.$$

Works if each entry of $\mathbf{e}^t\begin{bmatrix}\mathbf{R}\\ \mathbf{I}\end{bmatrix}$ in $[-\frac{q}{4}, \frac{q}{4})$, e.g. if $\|\mathbf{e}\| < q/(4s_1(\begin{bmatrix}\mathbf{R}\\ \mathbf{I}\end{bmatrix}))$.

Sampling Gaussian Preimages

Given $\mathbf{u} = f_{\mathbf{A}}(\mathbf{x}') = \mathbf{A}\mathbf{x}'$, sample $\mathbf{z} \leftarrow f_{\mathbf{G}}^{-1}(\mathbf{u})$ and output $\mathbf{x} = \begin{bmatrix}\mathbf{R}\\ \mathbf{I}\end{bmatrix}\mathbf{z}$?

- We have $\mathbf{A}\mathbf{x} = \mathbf{G}\mathbf{z} = \mathbf{u}$ as desired.
- Problem: $\begin{bmatrix}\mathbf{R}\\ \mathbf{I}\end{bmatrix}\mathbf{z}$ is non-spherical Gaussian, leaks $\mathbf{R}$!
- Solution: use offline 'perturbation' [P'10] to get spherical Gaussian w/ std dev $\approx s_1(\mathbf{R})$: output $\mathbf{x} = \mathbf{p} + \begin{bmatrix}\mathbf{R}\\ \mathbf{I}\end{bmatrix}\mathbf{z}$.

Trapdoor Delegation [CHKP'10]

- Suppose $\mathbf{R}$ is a trapdoor for $\mathbf{A}$, i.e. $\mathbf{A}\left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right] = \mathbf{H}\cdot\mathbf{G}$.

Trapdoor Delegation [CHKP'10]

- Suppose $\mathbf{R}$ is a trapdoor for $\mathbf{A}$, i.e. $\mathbf{A}\left[\begin{smallmatrix}\mathbf{R}\\\mathbf{I}\end{smallmatrix}\right] = \mathbf{H}\cdot\mathbf{G}$.
- To delegate a trapdoor for an extension $[\mathbf{A} \mid \mathbf{A}']$ with tag $\mathbf{H}'$, just sample Gaussian $\mathbf{R}'$ s.t.

$$[\mathbf{A} \mid \mathbf{A}']\left[\begin{smallmatrix}\mathbf{R}'\\\mathbf{I}\end{smallmatrix}\right] = \mathbf{H}'\cdot\mathbf{G} \iff \mathbf{A}\mathbf{R}' = \mathbf{H}'\cdot\mathbf{G} - \mathbf{A}'.$$

Trapdoor Delegation [CHKP'10]

- Suppose $\mathbf{R}$ is a trapdoor for $\mathbf{A}$, i.e. $\mathbf{A}\left[\begin{smallmatrix}\mathbf{R}\\\mathbf{I}\end{smallmatrix}\right] = \mathbf{H}\cdot\mathbf{G}$.
- To delegate a trapdoor for an extension $[\mathbf{A} \mid \mathbf{A}']$ with tag $\mathbf{H}'$, just sample Gaussian $\mathbf{R}'$ s.t.

$$[\mathbf{A} \mid \mathbf{A}']\left[\begin{smallmatrix}\mathbf{R}'\\\mathbf{I}\end{smallmatrix}\right] = \mathbf{H}'\cdot\mathbf{G} \iff \mathbf{A}\mathbf{R}' = \mathbf{H}'\cdot\mathbf{G} - \mathbf{A}'.$$

- Note: $\mathbf{R}'$ is only width$(\mathbf{A}) \times$ width$(\mathbf{G})$ = $m \times n\log q$.

  So size of $\mathbf{R}'$ grows only as $O(m)$, not $\Omega(m^2)$ [CHKP'10].

  Also computationally efficient: $n\log q$ samples, no HNF or ToBasis.

Improved "Bonsai" Applications

Hierarchical IBE [CHKP'10,ABB'10]

- Setup(d): choose $\mathbf{A}_0, \ldots, \mathbf{A}_d$ (each dim $n \log q$) where $\mathbf{A}_\varepsilon = [\mathbf{A}_0 \mid \mathbf{A}_1]$ has trapdoor $\mathbf{R}_\varepsilon$ for tag $\mathbf{0}$.

  Let $msk = sk_\varepsilon = \mathbf{R}_\varepsilon$ and $mpk = \{\mathbf{A}_i\}$ $(d + 1 \text{ vs } \geq 4d + 2)$

Improved "Bonsai" Applications

Hierarchical IBE [CHKP'10,ABB'10]

- Setup(d): choose $\mathbf{A}_0, \ldots, \mathbf{A}_d$ (each dim $n \log q$) where
  $\mathbf{A}_\varepsilon = [\mathbf{A}_0 \mid \mathbf{A}_1]$ has trapdoor $\mathbf{R}_\varepsilon$ for tag $\mathbf{0}$.
  Let $msk = sk_\varepsilon = \mathbf{R}_\varepsilon$ and $mpk = \{\mathbf{A}_i\}$ $(d+1$ vs $\geq 4d+2)$
- For $id = (\mathbf{H}_1, \ldots, \mathbf{H}_t)$ of nonzero (invertible) $\mathbf{H}_i \in \mathcal{H}$, let
  $$\mathbf{A}_{id} = [\mathbf{A}_0 \mid \mathbf{A}_1 - \mathbf{H}_1\mathbf{G} \mid \cdots \mid \mathbf{A}_t - \mathbf{H}_t\mathbf{G} \mid \mathbf{A}_{t+1}].$$

Improved "Bonsai" Applications

Hierarchical IBE [CHKP'10,ABB'10]

- Setup(d): choose $\mathbf{A}_0, \ldots, \mathbf{A}_d$ (each dim $n \log q$) where

  $\mathbf{A}_\varepsilon = [\mathbf{A}_0 \mid \mathbf{A}_1]$ has trapdoor $\mathbf{R}_\varepsilon$ for tag $\mathbf{0}$.

  Let $msk = sk_\varepsilon = \mathbf{R}_\varepsilon$ and $mpk = \{\mathbf{A}_i\}$ $(d + 1$ vs $\geq 4d + 2)$

- For $id = (\mathbf{H}_1, \ldots, \mathbf{H}_t)$ of nonzero (invertible) $\mathbf{H}_i \in \mathcal{H}$, let

$$\mathbf{A}_{id} = [\mathbf{A}_0 \mid \mathbf{A}_1 - \mathbf{H}_1\mathbf{G} \mid \cdots \mid \mathbf{A}_t - \mathbf{H}_t\mathbf{G} \mid \mathbf{A}_{t+1}].$$

  and sk_{id} is a trapdoor $\mathbf{R}_{id}$ for $\mathbf{A}_{id}$ with tag $\mathbf{0}$.

  Using sk_{id}, can delegate any $sk_{id'}$ for any nontrivial extension id'.

Improved "Bonsai" Applications

Hierarchical IBE [CHKP'10,ABB'10]

- Setup(d): choose $\mathbf{A}_0, \ldots, \mathbf{A}_d$ (each dim $n \log q$) where

  $\mathbf{A}_\varepsilon = [\mathbf{A}_0 \mid \mathbf{A}_1]$ has trapdoor $\mathbf{R}_\varepsilon$ for tag $\mathbf{0}$.

  Let $msk = sk_\varepsilon = \mathbf{R}_\varepsilon$ and $mpk = \{\mathbf{A}_i\}$ ($d+1$ vs $\geq 4d+2$)

- For $id = (\mathbf{H}_1, \ldots, \mathbf{H}_t)$ of nonzero (invertible) $\mathbf{H}_i \in \mathcal{H}$, let

  $$\mathbf{A}_{id} = [\mathbf{A}_0 \mid \mathbf{A}_1 - \mathbf{H}_1\mathbf{G} \mid \cdots \mid \mathbf{A}_t - \mathbf{H}_t\mathbf{G} \mid \mathbf{A}_{t+1}].$$

  and sk_{id} is a trapdoor $\mathbf{R}_{id}$ for $\mathbf{A}_{id}$ with tag $\mathbf{0}$.

  Using sk_{id}, can delegate any $sk_{id'}$ for any nontrivial extension id'.

- Encrypt (up to $n \log q$ bits) to $\mathbf{A}_{id}$, decrypt using $\mathbf{R}_{id}$ as in [GPV'08].

Improved "Bonsai" Applications

Hierarchical IBE [CHKP'10,ABB'10]

- Setup(d): choose $\mathbf{A}_0, \ldots, \mathbf{A}_d$ (each dim $n \log q$) where

  $\mathbf{A}_\varepsilon = [\mathbf{A}_0 \mid \mathbf{A}_1]$ has trapdoor $\mathbf{R}_\varepsilon$ for tag $\mathbf{0}$.

  Let $msk = sk_\varepsilon = \mathbf{R}_\varepsilon$ and $mpk = \{\mathbf{A}_i\}$ $(d+1$ vs $\geq 4d+2)$

- For $id = (\mathbf{H}_1, \ldots, \mathbf{H}_t)$ of nonzero (invertible) $\mathbf{H}_i \in \mathcal{H}$, let

  $$\mathbf{A}_{id} = [\mathbf{A}_0 \mid \mathbf{A}_1 - \mathbf{H}_1\mathbf{G} \mid \cdots \mid \mathbf{A}_t - \mathbf{H}_t\mathbf{G} \mid \mathbf{A}_{t+1}].$$

  and sk_{id} is a trapdoor $\mathbf{R}_{id}$ for $\mathbf{A}_{id}$ with tag $\mathbf{0}$.

  Using sk_{id}, can delegate any $sk_{id'}$ for any nontrivial extension id'.

- Encrypt (up to $n \log q$ bits) to $\mathbf{A}_{id}$, decrypt using $\mathbf{R}_{id}$ as in [GPV'08].

- Security ("puncturing"): Set up mpk, trapdoor $\mathbf{R}$ with tags = id^*.

  Family $\mathcal{H}$ with "invertible differences" from extension ring of $\mathbb{Z}_q$ [DF'94,Fehr'98,ABB'10]

Conclusions

- A new, simpler, more efficient trapdoor notion and construction

Conclusions

- A new, simpler, more efficient trapdoor notion and construction
- Exposing structure of trapdoor to applications yields further efficiency improvements

Conclusions

- A new, simpler, more efficient trapdoor notion and construction
- Exposing structure of trapdoor to applications yields further efficiency improvements
- Key sizes and algorithms for "strong" trapdoors are now practical

Conclusions

- A new, simpler, more efficient trapdoor notion and construction
- Exposing structure of trapdoor to applications yields further efficiency improvements
- Key sizes and algorithms for “strong” trapdoors are now practical

Questions?