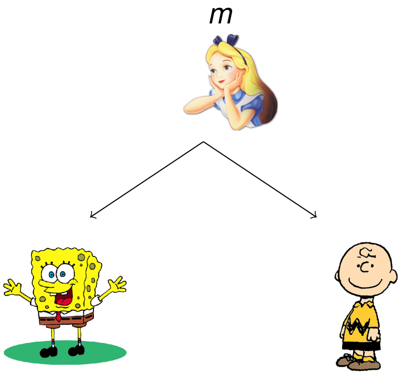
Functional Encryption for Inner Product Predicates from Learning with Errors

Shweta Agrawal[1], **David Mandell Freeman**[2], and Vinod Vaikuntanathan[3]

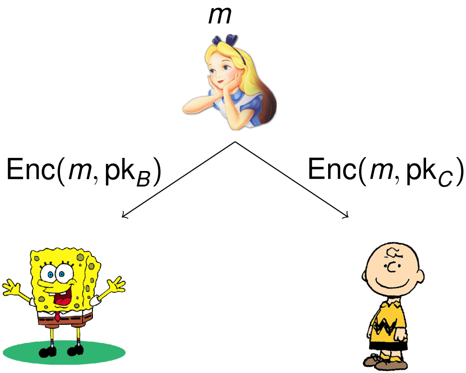
[1]UCLA, USA; [2]Stanford University, USA; [3]University of Toronto, Canada

Asiacrypt 2011
Seoul, Korea
5 December 2011

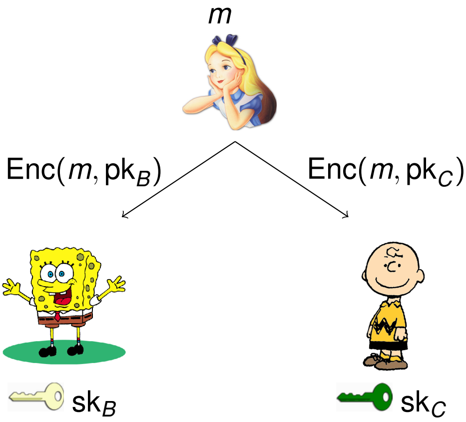
Traditional Public-Key Encryption [DH76,RSA78,...]



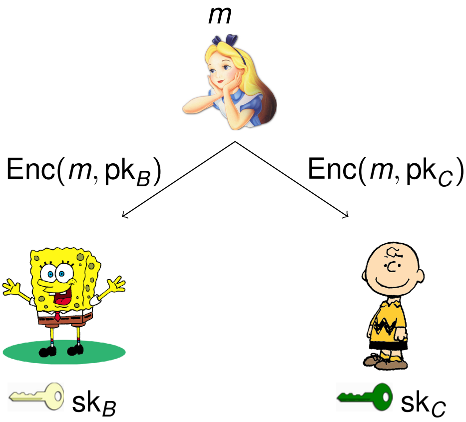
Traditional Public-Key Encryption [DH76,RSA78,...]



Traditional Public-Key Encryption [DH76,RSA78,...]

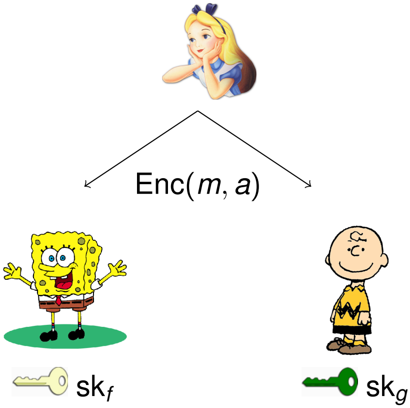


Traditional Public-Key Encryption [DH76,RSA78,...]



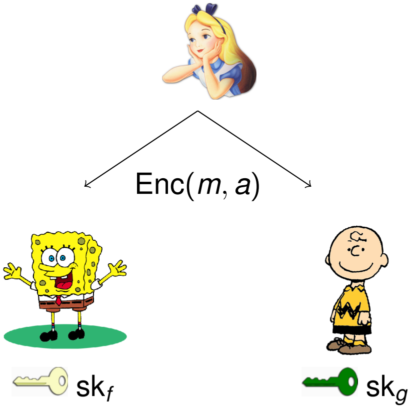
- m must be encrypted separately to each user.
- Recipient set must be decided in advance.

Functional Encryption [SW05,GPSW05,BW07,KSW08,...]



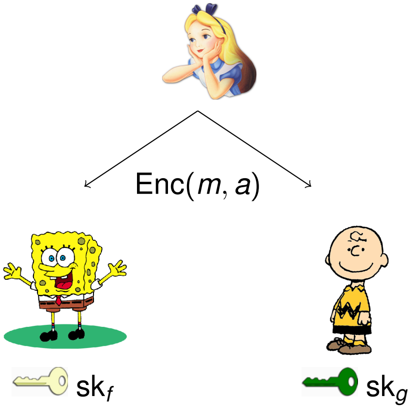
- Ciphertext equipped with attribute a.
- sk equipped with predicate f.
- User with sk_f can decrypt iff $f(a) = 1$.

Functional Encryption [SW05,GPSW05,BW07,KSW08,...]



- Ciphertext equipped with attribute a.
- sk equipped with predicate f.
- User with sk_f can decrypt iff $f(a) = 1$.

E.g.: attribute $a =$ (conf="Asiacrypt", year=2011),

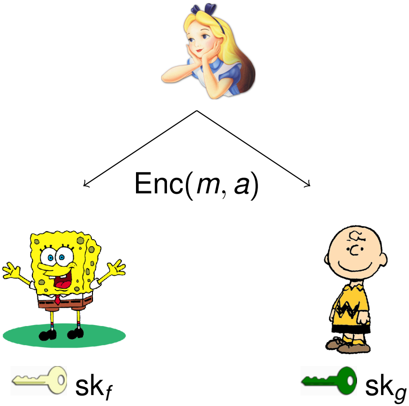


- Ciphertext equipped with attribute a.
- sk equipped with predicate f.
- User with sk_f can decrypt iff $f(a) = 1$.

E.g.: attribute $a =$ (conf="Asiacrypt", year=2011),

predicates $f =$ (conf="Asiacrypt" AND year$\geq$2000),

Functional Encryption [SW05,GPSW05,BW07,KSW08,...]



- Ciphertext equipped with attribute a.
- sk equipped with predicate f.
- User with sk_f can decrypt iff $f(a) = 1$.

E.g.: attribute $a =$ (conf="Asiacrypt", year=2011),

predicates $f =$ (conf="Asiacrypt" AND year$\geq$2000),

$g =$ (conf="Eurocrypt" OR year=2011)

Prior Work on Functional Encryption

Identity-based encryption is functional encryption for equality predicates.

- Ciphertexts & keys equipped with identity *id*.
- Decrypt succeeds iff (key *id*) $=$ (CT *id*).
- Achieved using pairings, QR, and lattices.
  [BF01,BB04ab,...], [C01,BGH07], [GPV08,CHKP10,ABB10ab]

Prior Work on Functional Encryption

Identity-based encryption is functional encryption for equality predicates.

- Ciphertexts & keys equipped with identity *id*.
- Decrypt succeeds iff (key *id*) $=$ (CT *id*).
- Achieved using pairings, QR, and lattices.
  [BF01,BB04ab,...], [C01,BGH07], [GPV08,CHKP10,ABB10ab]

Inner product predicates [KSW08,OT09,LOSTW10,...]:

- CT $\leftrightarrow$ vector $\vec{w}$; key $\leftrightarrow$ vector $\vec{v}$
- Key for $\vec{v}$ can decrypt CT for $\vec{w}$ iff $\langle \vec{v}, \vec{w} \rangle = 0$.
- Achieved using pairings.

Prior Work on Functional Encryption

Identity-based encryption is functional encryption for equality predicates.

- Ciphertexts & keys equipped with identity *id*.
- Decrypt succeeds iff (key *id*) $=$ (CT *id*).
- Achieved using pairings, QR, and lattices.
  [BF01,BB04ab,...], [C01,BGH07], [GPV08,CHKP10,ABB10ab]

Inner product predicates [KSW08,OT09,LOSTW10,...]:

- CT $\leftrightarrow$ vector $\vec{w}$; key $\leftrightarrow$ vector $\vec{v}$
- Key for $\vec{v}$ can decrypt CT for $\vec{w}$ iff $\langle \vec{v}, \vec{w} \rangle = 0$.
- Achieved using pairings.

[KSW08]: Inner product predicates allow us to instantiate range, conjunction, disjunction, and polynomial evaluation predicates.

Our Contribution

Functional encryption for inner product predicates based on the *learning with errors* (LWE) assumption.

- Achieves functionality of [KSW08].
- Worst-case reduction, (conjectured) quantum security.
- Allows inner products over small fields.

Our Contribution

Functional encryption for inner product predicates based on the *learning with errors* (LWE) assumption.

- Achieves functionality of [KSW08].
- Worst-case reduction, (conjectured) quantum security.
- Allows inner products over small fields.

Privacy property: CT attribute is hidden from users who cannot decrypt ("*weakly attribute hiding*").

- [KSW08] construction hides attribute from all users.
- Open problem: achieve same privacy property from LWE.

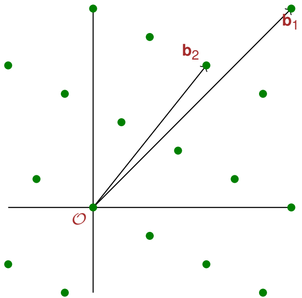
Building Blocks

Lattice-based PKE [GPV08 “dual Regev”]:

Building Blocks

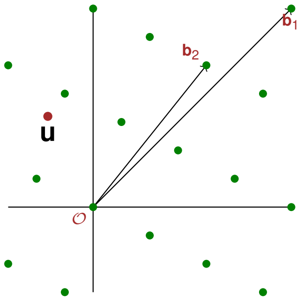
Lattice-based PKE [GPV08 “dual Regev”]:

pk: lattice $\Lambda \subset \mathbb{Z}^m$



Lattice-based PKE [GPV08 “dual Regev”]:

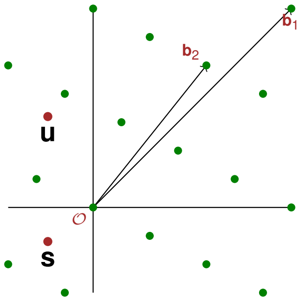
pk: lattice $\Lambda \subset \mathbb{Z}^m$, vector $\mathbf{u}$.



Lattice-based PKE [GPV08 “dual Regev”]:

pk: lattice $\Lambda \subset \mathbb{Z}^m$, vector $\mathbf{u}$.

sk: short vector $\mathbf{s}$ in coset $\Lambda^\perp + \mathbf{u}$ of dual lattice.

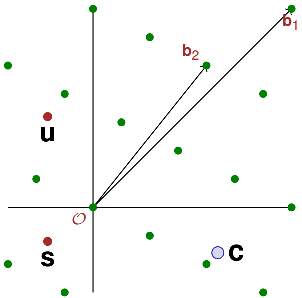


Lattice-based PKE [GPV08 "dual Regev"]:

pk: lattice $\Lambda \subset \mathbb{Z}^m$, vector $\mathbf{u}$.

sk: short vector $\mathbf{s}$ in coset $\Lambda^{\perp} + \mathbf{u}$ of dual lattice.

Enc: vector $\mathbf{c}$ close to Λ, scalar c' encoding $m \in \{0, 1\}$.



Building Blocks

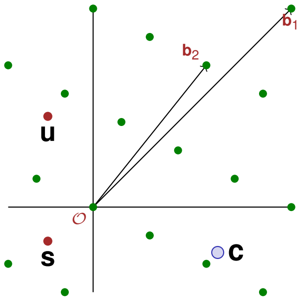
Lattice-based PKE [GPV08 “dual Regev”]:

pk: lattice $\Lambda \subset \mathbb{Z}^m$, vector $\mathbf{u}$.

sk: short vector $\mathbf{s}$ in coset $\Lambda^\perp + \mathbf{u}$ of dual lattice.

Enc: vector $\mathbf{c}$ close to Λ, scalar c' encoding $m \in \{0, 1\}$.

Dec: use $\langle \mathbf{s}, \mathbf{c} \rangle$ to decode c'.



Building Blocks

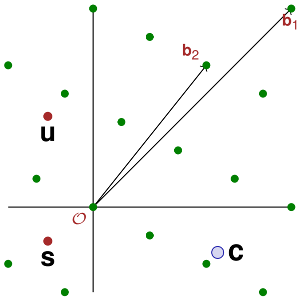
Lattice-based PKE [GPV08 "dual Regev"]:

pk: lattice $\Lambda \subset \mathbb{Z}^m$, vector $\mathbf{u}$.

sk: short vector $\mathbf{s}$ in coset $\Lambda^\perp + \mathbf{u}$ of dual lattice.

Enc: vector $\mathbf{c}$ close to Λ, scalar c' encoding $m \in \{0, 1\}$.

Dec: use $\langle \mathbf{s}, \mathbf{c} \rangle$ to decode c'.



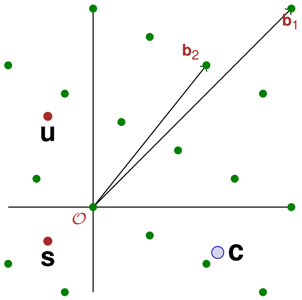
Two ways to generate keys for Λ:

- Choose short sk $\mathbf{s}$, compute pk vector $\mathbf{u}$.

Building Blocks

Lattice-based PKE [GPV08 "dual Regev"]:

- pk: lattice $\Lambda \subset \mathbb{Z}^m$, vector $\mathbf{u}$.
- sk: short vector $\mathbf{s}$ in coset $\Lambda^\perp + \mathbf{u}$ of dual lattice.
- Enc: vector $\mathbf{c}$ close to Λ, scalar c' encoding $m \in \{0, 1\}$.
- Dec: use $\langle \mathbf{s}, \mathbf{c} \rangle$ to decode c'.



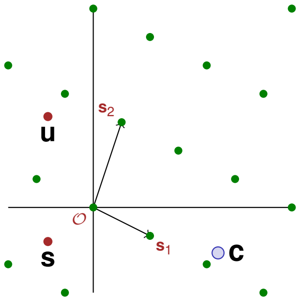
Two ways to generate keys for Λ:

- Choose short sk $\mathbf{s}$, compute pk vector $\mathbf{u}$.
- Given $\mathbf{u}$, use short basis of $\Lambda^\perp$ to find $\mathbf{s}$.

Building Blocks

Lattice-based PKE [GPV08 “dual Regev”]:

- pk: lattice $\Lambda \subset \mathbb{Z}^m$, vector $\mathbf{u}$.
- sk: short vector $\mathbf{s}$ in coset $\Lambda^\perp + \mathbf{u}$ of dual lattice.
- Enc: vector $\mathbf{c}$ close to Λ, scalar c' encoding $m \in \{0, 1\}$.
- Dec: use $\langle \mathbf{s}, \mathbf{c} \rangle$ to decode c'.



Two ways to generate keys for Λ:

- Choose short sk $\mathbf{s}$, compute pk vector $\mathbf{u}$.
- Given $\mathbf{u}$, use short basis of $\Lambda^\perp$ to find $\mathbf{s}$.

Building Blocks

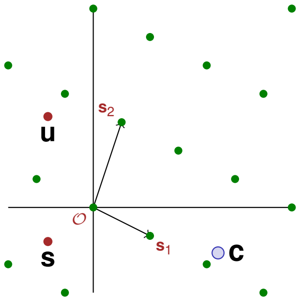
Lattice-based PKE [GPV08 "dual Regev"]:

pk: lattice $\Lambda \subset \mathbb{Z}^m$, vector $\mathbf{u}$.

sk: short vector $\mathbf{s}$ in coset $\Lambda^\perp + \mathbf{u}$ of dual lattice.

Enc: vector $\mathbf{c}$ close to Λ, scalar c' encoding $m \in \{0, 1\}$.

Dec: use $\langle \mathbf{s}, \mathbf{c} \rangle$ to decode c'.



Two ways to generate keys for Λ:

- Choose short sk $\mathbf{s}$, compute pk vector $\mathbf{u}$.
- Given $\mathbf{u}$, use short basis of $\Lambda^\perp$ to find $\mathbf{s}$.

[A99,AP09]: Can generate a random lattice Λ along with short basis of $\Lambda^\perp$ = trapdoor for Λ.

Building Block: Lattice-Based IBE [CHKP10,ABB10ab]

Building Block: Lattice-Based IBE [CHKP10,ABB10ab]

Each identity *id* defines a lattice Λ_{id}.

- CT is GPV encryption relative to Λ_{id}.
- Trapdoor for Λ_{id} used to derive sk for *id*.
- Can decrypt iff sk lattice matches CT lattice.

Building Block: Lattice-Based IBE [CHKP10,ABB10ab]

Each identity *id* defines a lattice Λ_{id}.

- CT is GPV encryption relative to Λ_{id}.
- Trapdoor for Λ_{id} used to derive sk for *id*.
- Can decrypt iff sk lattice matches CT lattice.

IBE schemes don't seem to generalize to functional encryption:

- In functional encryption, many sk can decrypt each CT.

  CT for $\vec{w}$ decryptable by sk for any $\vec{v}$ with $\langle \vec{v}, \vec{w} \rangle = 0$.

Building Block: Lattice-Based IBE [CHKP10,ABB10ab]

Each identity *id* defines a lattice Λ_{id}.

- CT is GPV encryption relative to Λ_{id}.
- Trapdoor for Λ_{id} used to derive sk for *id*.
- Can decrypt iff sk lattice matches CT lattice.

IBE schemes don't seem to generalize to functional encryption:

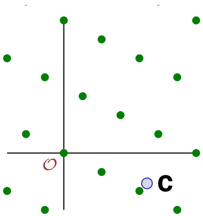
- In functional encryption, many sk can decrypt each CT.

  CT for $\vec{w}$ decryptable by sk for any $\vec{v}$ with $\langle \vec{v}, \vec{w} \rangle = 0$.

Conclude: can't require CT lattice to match sk lattice.

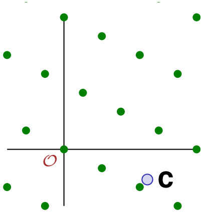
Building Functional Encryption

Encrypt relative to
attribute lattice $\Lambda_{\vec{w}} \subset \mathbb{Z}^r$

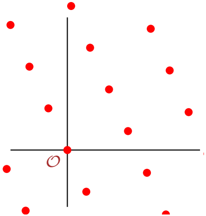


Building Functional Encryption

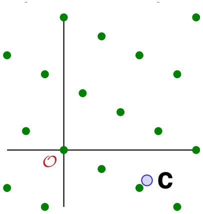
Encrypt relative to
attribute lattice $\Lambda_{\vec{w}} \subset \mathbb{Z}^r$



sk corresponds to
predicate lattice $\Lambda_{\vec{v}} \subset \mathbb{Z}^s$



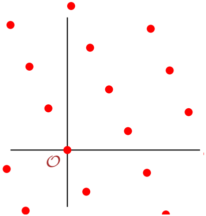
Encrypt relative to
attribute lattice $\Lambda_{\vec{w}} \subset \mathbb{Z}^r$



"short"
linear map

$T_{\vec{v}} : \mathbb{Z}^r \to \mathbb{Z}^s$

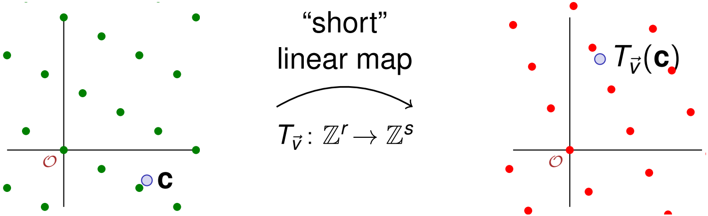
sk corresponds to
predicate lattice $\Lambda_{\vec{v}} \subset \mathbb{Z}^s$



$$T_{\vec{v}}(\Lambda_{\vec{w}}) = \Lambda_{\vec{v}} \quad \text{iff} \quad \langle \vec{v}, \vec{w} \rangle = 0$$

Encrypt relative to
attribute lattice $\Lambda_{\vec{w}} \subset \mathbb{Z}^r$

sk corresponds to
predicate lattice $\Lambda_{\vec{v}} \subset \mathbb{Z}^s$



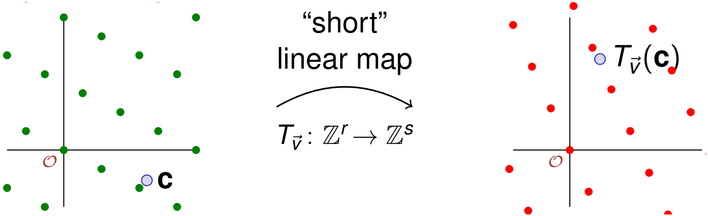
$$T_{\vec{v}}(\Lambda_{\vec{w}}) = \Lambda_{\vec{v}} \quad \text{iff} \quad \langle \vec{v}, \vec{w} \rangle = 0$$

If $\langle \vec{v}, \vec{w} \rangle = 0$, $T_{\vec{v}}(\mathbf{c})$ is a CT relative to $\Lambda_{\vec{v}}$

Building Functional Encryption

Encrypt relative to *attribute lattice* $\Lambda_{\vec{w}} \subset \mathbb{Z}^r$

sk corresponds to *predicate lattice* $\Lambda_{\vec{v}} \subset \mathbb{Z}^s$



$$T_{\vec{v}}(\Lambda_{\vec{w}}) = \Lambda_{\vec{v}} \quad \text{iff} \quad \langle \vec{v}, \vec{w} \rangle = 0$$

If $\langle \vec{v}, \vec{w} \rangle = 0$, $T_{\vec{v}}(\mathbf{c})$ is a CT relative to $\Lambda_{\vec{v}}$
$\Rightarrow$ key for $\Lambda_{\vec{v}}$ can decrypt $T_{\vec{v}}(\mathbf{c})$.

What Lattices are Used?

Regev/GPV lattice Λ defined by matrix $\mathbf{A}_0 \in \mathbb{Z}_q^{n \times m}$, $n < m$:

$$\Lambda = \Lambda_q(\mathbf{A}_0) = \{\mathbf{v} \in \mathbb{Z}^m : \mathbf{v} \bmod q = \mathbf{r}^t \cdot \mathbf{A}_0 \text{ for some } \mathbf{r} \in \mathbb{Z}_q^n\}$$

- i.e., vectors in $\mathbb{Z}^m$ that (mod q) are linear combinations of rows of $\mathbf{A}_0$.

What Lattices are Used?

Regev/GPV lattice Λ defined by matrix $\mathbf{A}_0 \in \mathbb{Z}_q^{n\times m}$, $n < m$:

$$\Lambda = \Lambda_q(\mathbf{A}_0) = \{\mathbf{v} \in \mathbb{Z}^m : \mathbf{v} \bmod q = \mathbf{r}^t \cdot \mathbf{A}_0 \text{ for some } \mathbf{r} \in \mathbb{Z}_q^n\}$$

- i.e., vectors in $\mathbb{Z}^m$ that (mod q) are linear combinations of rows of $\mathbf{A}_0$.

[ABB10a] IBE: to encrypt to identity *id*, use lattice

$$\Lambda_{id} = \Lambda_q(\mathbf{A}_0 \parallel \mathbf{A}_1 + H(id)\mathbf{B}) \subset \mathbb{Z}^{2m}.$$

- public $\mathbf{A}_0, \mathbf{A}_1, \mathbf{B} \in \mathbb{Z}_q^{n\times m}$.
- $H\colon \{0,1\}^* \to \mathbb{Z}_q^{n\times n}$ is a hash function.

Secret key for Λ_{id} can be computed using trapdoor for $\mathbf{A}_0$.

A Functional Encryption Scheme

A Functional Encryption Scheme

To compute CT for vector $\vec{w} = (w_1, \ldots, w_\ell)$, use lattice

$$\Lambda_{\vec{w}} = \Lambda_q(\mathbf{A}_0 \parallel \mathbf{A}_1 + w_1\mathbf{B} \parallel \cdots \parallel \mathbf{A}_\ell + w_\ell\mathbf{B}) \subset \mathbb{Z}^{(1+\ell)m}.$$

- public $\mathbf{A}_i, \mathbf{B} \in \mathbb{Z}_q^{n \times m}$.

A Functional Encryption Scheme

To compute CT for vector $\vec{w} = (w_1, \ldots, w_\ell)$, use lattice

$$\Lambda_{\vec{w}} = \Lambda_q(\mathbf{A}_0 \parallel \mathbf{A}_1 + w_1\mathbf{B} \parallel \cdots \parallel \mathbf{A}_\ell + w_\ell\mathbf{B}) \subset \mathbb{Z}^{(1+\ell)m}.$$

- public $\mathbf{A}_i, \mathbf{B} \in \mathbb{Z}_q^{n \times m}$.

To generate sk for vector $\vec{v} = (v_1, \ldots, v_\ell)$, use lattice

$$\Lambda_{\vec{v}} = \Lambda_q(\mathbf{A}_0 \parallel \sum v_i\mathbf{A}_i) \subset \mathbb{Z}^{2m}.$$

- Use trapdoor for $\mathbf{A}_0$ + [CHKP10] “delegation” technique.

A Functional Encryption Scheme

To compute CT for vector $\vec{w} = (w_1, \ldots, w_\ell)$, use lattice

$$\Lambda_{\vec{w}} = \Lambda_q(\mathbf{A}_0 \parallel \mathbf{A}_1 + w_1\mathbf{B} \parallel \cdots \parallel \mathbf{A}_\ell + w_\ell\mathbf{B}) \subset \mathbb{Z}^{(1+\ell)m}.$$

- public $\mathbf{A}_i, \mathbf{B} \in \mathbb{Z}_q^{n\times m}$.

To generate sk for vector $\vec{v} = (v_1, \ldots, v_\ell)$, use lattice

$$\Lambda_{\vec{v}} = \Lambda_q(\mathbf{A}_0 \parallel \sum v_i\mathbf{A}_i) \subset \mathbb{Z}^{2m}.$$

- Use trapdoor for $\mathbf{A}_0$ + [CHKP10] "delegation" technique.

To decrypt, apply transformation $T_{\vec{v}} : \mathbb{Z}^{(1+\ell)m} \to \mathbb{Z}^{2m}$ given by

$$T_{\vec{v}}(\mathbf{c}_0, \ldots, \mathbf{c}_\ell) = (\mathbf{c}_0, \sum v_i\mathbf{c}_i).$$

A Functional Encryption Scheme

To compute CT for vector $\vec{w} = (w_1, \ldots, w_\ell)$, use lattice

$$\Lambda_{\vec{w}} = \Lambda_q(\mathbf{A}_0 \parallel \mathbf{A}_1 + w_1\mathbf{B} \parallel \cdots \parallel \mathbf{A}_\ell + w_\ell\mathbf{B}) \subset \mathbb{Z}^{(1+\ell)m}.$$

- public $\mathbf{A}_i, \mathbf{B} \in \mathbb{Z}_q^{n \times m}$.

To generate sk for vector $\vec{v} = (v_1, \ldots, v_\ell)$, use lattice

$$\Lambda_{\vec{v}} = \Lambda_q(\mathbf{A}_0 \parallel \sum v_i\mathbf{A}_i) \subset \mathbb{Z}^{2m}.$$

- Use trapdoor for $\mathbf{A}_0$ + [CHKP10] "delegation" technique.

To decrypt, apply transformation $T_{\vec{v}} : \mathbb{Z}^{(1+\ell)m} \to \mathbb{Z}^{2m}$ given by

$$T_{\vec{v}}(\mathbf{c}_0, \ldots, \mathbf{c}_\ell) = (\mathbf{c}_0, \sum v_i\mathbf{c}_i).$$

Then

$$T_{\vec{v}}(\Lambda_{\vec{w}}) = \Lambda_q(\mathbf{A}_0 \parallel \sum v_i\mathbf{A}_i + \langle \vec{v}, \vec{w} \rangle \mathbf{B})$$

So sk for $\Lambda_{\vec{v}}$ can decrypt $T_{\vec{v}}(CT)$ iff $\langle \vec{v}, \vec{w} \rangle = 0$ (and $\vec{v}$ is short).

(Selective) Security Model

Challenger

Adversary

(Selective) Security Model

attributes $\vec{w}_0, \vec{w}_1$ $\longleftarrow$

Challenger

Adversary

(Selective) Security Model

Challenger

Adversary

attributes $\vec{w}_0, \vec{w}_1$ ←

 pk →

(Selective) Security Model

Challenger

Adversary

attributes $\vec{w}_0, \vec{w}_1$ ←

 pk →

predicate $\vec{v} : \langle \vec{v}, \vec{w}_b \rangle \neq 0$ ←

Challenger

attributes $\vec{w}_0, \vec{w}_1$ ←

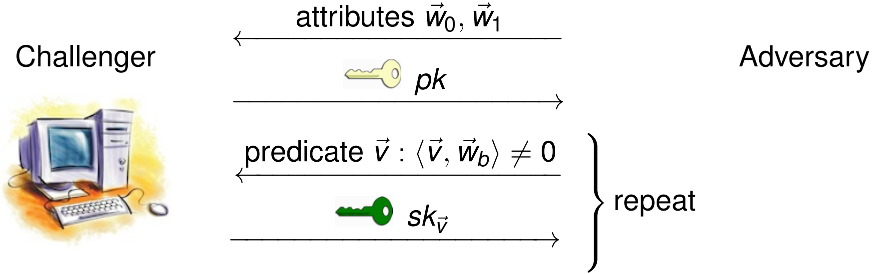
 pk →

predicate $\vec{v} : \langle \vec{v}, \vec{w}_b \rangle \neq 0$ ←

 $sk_{\vec{v}}$ →

Adversary

(Selective) Security Model



(Selective) Security Model

Challenger

Adversary

attributes $\vec{w}_0, \vec{w}_1$ ←

 pk →

predicate $\vec{v} : \langle \vec{v}, \vec{w}_b \rangle \neq 0$ ←

 $sk_{\vec{v}}$ →

} repeat

messages m_0, m_1 ←

(Selective) Security Model

Challenger

b $\xleftarrow{R}$ {0,1}

attributes $\vec{w}_0, \vec{w}_1$ ←

 pk →

predicate $\vec{v}$: $\langle \vec{v}, \vec{w}_b \rangle \neq 0$ ←

 $sk_{\vec{v}}$ →

} repeat

messages m_0, m_1 ←

Adversary

(Selective) Security Model

Challenger

$b \xleftarrow{R} \{0,1\}$

attributes $\vec{w}_0, \vec{w}_1$ ←

 pk →

predicate $\vec{v} : \langle \vec{v}, \vec{w}_b \rangle \neq 0$ ←

 $sk_{\vec{v}}$ →

} repeat

messages m_0, m_1 ←

$\mathrm{Enc}(m_b, \vec{w}_b)$ →

Adversary

(Selective) Security Model

Challenger

b $\xleftarrow{R} \{0,1\}$

Adversary

attributes $\vec{w}_0, \vec{w}_1$ ←

 pk →

predicate $\vec{v} : \langle \vec{v}, \vec{w}_b \rangle \neq 0$ ←

 $sk_{\vec{v}}$ →

} repeat

messages m_0, m_1 ←

$\text{Enc}(m_b, \vec{w}_b)$ →

predicate $\vec{v} : \langle \vec{v}, \vec{w}_b \rangle \neq 0$ ←

 $sk_{\vec{v}}$ →

} repeat

(Selective) Security Model

Challenger

$b \xleftarrow{R} \{0,1\}$

Adversary

$b' \in \{0, 1\}$

attributes $\vec{w}_0, \vec{w}_1$ ←

 pk →

predicate $\vec{v} : \langle \vec{v}, \vec{w}_b \rangle \neq 0$ ←

 $sk_{\vec{v}}$ →

} repeat

messages m_0, m_1 ←

$\mathrm{Enc}(m_b, \vec{w}_b)$ →

predicate $\vec{v} : \langle \vec{v}, \vec{w}_b \rangle \neq 0$ ←

 $sk_{\vec{v}}$ →

} repeat

Challenger

$b \xleftarrow{R} \{0,1\}$

attributes $\vec{w}_0, \vec{w}_1$ ←

 pk →

predicate $\vec{v} : \langle \vec{v}, \vec{w}_b \rangle \neq 0$ ←

 $sk_{\vec{v}}$ →

} repeat

messages m_0, m_1 ←

$\text{Enc}(m_b, \vec{w}_b)$ →

Adversary

$b' \in \{0, 1\}$

Definition

Scheme is *weakly attribute hiding* if $\left| \Pr[b' = b] - \frac{1}{2} \right|$ is negligible for all efficient $\mathcal{A}$.

Security Theorem

Learning With Errors (LWE) assumption [R05]

For fixed $\mathbf{s} \in \mathbb{Z}_q^n$, "noisy inner products" with $\mathbf{s}$ are indistinguishable from random:

$$\{\mathbf{a}_i, \langle \mathbf{s}, \mathbf{a}_i \rangle + e_i\}_{i=1}^{m} \approx_c \{\mathbf{a}_i, r_i\}_{i=1}^{m}$$

for random $\mathbf{a}_i \in \mathbb{Z}_q^n$, small $e_i \in \mathbb{Z}$, and random $r_i \in \mathbb{Z}_q$.

Security Theorem

Learning With Errors (LWE) assumption [R05]

For fixed $\mathbf{s} \in \mathbb{Z}_q^n$, "noisy inner products" with $\mathbf{s}$ are indistinguishable from random:

$$\left\{\mathbf{a}_i, \langle \mathbf{s}, \mathbf{a}_i \rangle + e_i\right\}_{i=1}^{m} \approx_c \left\{\mathbf{a}_i, r_i\right\}_{i=1}^{m}$$

for random $\mathbf{a}_i \in \mathbb{Z}_q^n$, small $e_i \in \mathbb{Z}$, and random $r_i \in \mathbb{Z}_q$.

- [R05,P09]: Algorithms that break LWE assumption can be used to solve worst-case lattice problems.

Security Theorem

Learning With Errors (LWE) assumption [R05]

For fixed $\mathbf{s} \in \mathbb{Z}_q^n$, "noisy inner products" with $\mathbf{s}$ are indistinguishable from random:

$$\{\mathbf{a}_i, \langle \mathbf{s}, \mathbf{a}_i \rangle + e_i\}_{i=1}^{m} \approx_c \{\mathbf{a}_i, r_i\}_{i=1}^{m}$$

for random $\mathbf{a}_i \in \mathbb{Z}_q^n$, small $e_i \in \mathbb{Z}$, and random $r_i \in \mathbb{Z}_q$.

- [R05,P09]: Algorithms that break LWE assumption can be used to solve worst-case lattice problems.

Theorem

If the LWE assumption holds, then our inner product encryption scheme is weakly attribute hiding.

Proof Idea

CT lattice: $\Lambda_{\vec{w}} = \Lambda_q(\mathbf{A}_0 \parallel \mathbf{A}_1 + w_1\mathbf{B} \parallel \cdots \parallel \mathbf{A}_\ell + w_\ell\mathbf{B})$.
sk lattice: $\Lambda_{\vec{v}} = \Lambda_q(\mathbf{A}_0 \parallel \sum v_i\mathbf{A}_i)$.

Proof Idea

CT lattice: $\Lambda_{\vec{w}} = \Lambda_q(\mathbf{A}_0 \parallel \mathbf{A}_1 + w_1\mathbf{B} \parallel \cdots \parallel \mathbf{A}_\ell + w_\ell\mathbf{B})$.
sk lattice: $\Lambda_{\vec{v}} = \Lambda_q(\mathbf{A}_0 \parallel \sum v_i\mathbf{A}_i)$.

[ABB10a] technique: Trapdoor for **B** can be used to answer sk queries for $\vec{v}$ with $\langle \vec{v}, \vec{w} \rangle \neq 0$.

Proof Idea

CT lattice: $\Lambda_{\vec{w}} = \Lambda_q(\mathbf{A}_0 \parallel \mathbf{A}_1 + w_1\mathbf{B} \parallel \cdots \parallel \mathbf{A}_\ell + w_\ell\mathbf{B})$.
sk lattice: $\Lambda_{\vec{v}} = \Lambda_q(\mathbf{A}_0 \parallel \sum v_i\mathbf{A}_i)$.

[ABB10a] technique: Trapdoor for **B** can be used to answer sk queries for $\vec{v}$ with $\langle \vec{v}, \vec{w} \rangle \neq 0$.

Embed LWE challenge in the matrix $\mathbf{A}_0$.

- If LWE challenge is "noisy inner products" $\langle \mathbf{s}, \mathbf{a}_i \rangle + e_i$, obtain real CT.
- If LWE challenge is random r_i, obtain uniformly random CT (no info. about message or attribute).

Proof Idea

CT lattice: $\Lambda_{\vec{w}} = \Lambda_q(\mathbf{A}_0 \parallel \mathbf{A}_1 + w_1\mathbf{B} \parallel \cdots \parallel \mathbf{A}_\ell + w_\ell\mathbf{B})$.
sk lattice: $\Lambda_{\vec{v}} = \Lambda_q(\mathbf{A}_0 \parallel \sum v_i\mathbf{A}_i)$.

[ABB10a] technique: Trapdoor for **B** can be used to answer sk queries for $\vec{v}$ with $\langle \vec{v}, \vec{w} \rangle \neq 0$.

Embed LWE challenge in the matrix $\mathbf{A}_0$.

- If LWE challenge is "noisy inner products" $\langle \mathbf{s}, \mathbf{a}_i \rangle + e_i$, obtain real CT.
- If LWE challenge is random r_i, obtain uniformly random CT (no info. about message or attribute).

Adversary that breaks system can break LWE assumption.

Open Questions

1. Fully attribute-hiding system.
   - Answer sk queries for $\vec{v}$ when $\langle \vec{v}, \vec{w} \rangle = 0$. [requires $m_0 = m_1$]

Open Questions

1. Fully attribute-hiding system.
   - Answer sk queries for $\vec{v}$ when $\langle \vec{v}, \vec{w} \rangle = 0$. [requires $m_0 = m_1$]
2. Fully secure system.
   - Allow adversary to make key queries before choosing attributes $\vec{w}_j$.

Open Questions

1. Fully attribute-hiding system.
   - Answer sk queries for $\vec{v}$ when $\langle \vec{v}, \vec{w} \rangle = 0$. [requires $m_0 = m_1$]
2. Fully secure system.
   - Allow adversary to make key queries before choosing attributes $\vec{w}_j$.
3. Improve efficiency.
   - Current system is efficient for $\vec{v}, \vec{w}$ over small fields.

Open Questions

1. Fully attribute-hiding system.
   - Answer sk queries for $\vec{v}$ when $\langle \vec{v}, \vec{w} \rangle = 0$. [requires $m_0 = m_1$]
2. Fully secure system.
   - Allow adversary to make key queries before choosing attributes $\vec{w}_j$.
3. Improve efficiency.
   - Current system is efficient for $\vec{v}, \vec{w}$ over small fields.
4. Functional encryption for larger class of predicates.
   - Leverage techniques from fully homomorphic encryption?

Open Questions

1. Fully attribute-hiding system.
   - Answer sk queries for $\vec{v}$ when $\langle \vec{v}, \vec{w} \rangle = 0$. [requires $m_0 = m_1$]
2. Fully secure system.
   - Allow adversary to make key queries before choosing attributes $\vec{w}_j$.
3. Improve efficiency.
   - Current system is efficient for $\vec{v}, \vec{w}$ over small fields.
4. Functional encryption for larger class of predicates.
   - Leverage techniques from fully homomorphic encryption?

Thank you!