Week 14: Advanced Lattice-Based Primitives

So far, we have seen how to leverage lattice homomorphisms to get homomorphic encryption and homomorphic signatures. This week, we will continue and look at another primitive that makes use of homomorphism: attribute-based encryption.

**Attribute-based encryption**: Generalization of identity-based encryption:
- Ciphertexts are associated with an attribute \( x \) and a message \( m \).
- Secret keys are associated with functions \( f \).

- Useful notion for enforcing access control (e.g., attributes might be "CONFIDENTIAL" and "TOP-SECRET") and decryption key corresponds to access level.

**Schema**:
\[
\begin{align*}
\text{Setup}(1^n) &\rightarrow (\text{mpk}, \text{msk}) \\
\text{KeyGen}(\text{msk}, C) &\rightarrow \text{sk}_C \\
\text{Encrypt}(\text{mpk}, x, m) &\rightarrow \text{ct} \\
\text{Decrypt}(\text{sk}_C, \text{ct}) &\rightarrow m / 1
\end{align*}
\]

- **Correctness**: for any attribute \( x \) and circuit \( C \) where \( C(x) = 1 \),
\[
(\text{mpk}, \text{msk}) \leftarrow \text{Setup}(1^n)
\]
\[
\text{sk}_C \leftarrow \text{KeyGen}(\text{msk}, C) \\
\text{ct} \leftarrow \text{Encrypt}(\text{mpk}, x, m)
\]
\[
\text{Pr} [\text{Decrypt}(\text{sk}_C, \text{ct}) = m] = 1
\]

- **Semantic Security**:

\[
\begin{align*}
\text{adversary} &\downarrow \\
\text{challenger} \downarrow & (\text{mpk}, \text{msk}) \leftarrow \text{Setup}(1^n) \\
\text{sk}_C &\leftarrow \text{KeyGen}(\text{msk}, C) \\
X, m_0, m_1 &\rightarrow \text{ct} \leftarrow \text{Encrypt}(\text{mpk}, x, m) \\
\text{adversary} &\downarrow
\end{align*}
\]

- **Requirement**: \( C(x) = 0 \) for all circuits queried by the adversary.

- **Selective Security**: Adversary commits to the challenge attribute \( X^* \) at the beginning of the security game.

- Selective security implies adaptive security via technique called "complexity leveraging" [reduction guesses the challenge attribute at the beginning of the game] - this incurs a subexponential loss in the security reduction so will require a subexponential hardness assumption:

\[
\text{AdaptiveAdv}[B] \leq \frac{1}{2} \cdot \text{SelectiveAdv}[A]
\]
where \( l \) is the attribute length.
Starting point: dual version of Regev's encryption (interchange ciphertexts with secret key):

\[
\begin{align*}
\text{Setup (1)} & : \text{Sample } A \in \mathbb{Z}_q^{n \times n}, \ r \in \mathbb{Z}_q \text{ and compute } u = Ar \in \mathbb{Z}_q^n \\
\text{Output} \ pk = (A, u) \text{ and } sk = r
\end{align*}
\]

- Encrypt (pk, \mu): Sample \ s \in \mathbb{Z}_q^n, \ e \in \mathbb{Z}_q, \ e' \in \mathbb{Z}_q \text{ and output } ct = (s^\top A + e, s^\top u + e' + \mu \cdot \frac{d}{2})
- Decrypt (sk, ct): Output \ lct \ = \langle ct, r \rangle_a

Correctness: \ \lct - \langle ct, r \rangle_a = s^\top u + e' + \mu \cdot \frac{d}{2} = s^\top Ar - e' r \\
\text{Correct as long as } |e' - e' r| < \frac{d}{q}

Security: By LHL, public key statistically indistinguishable from sampling \ A \in \mathbb{Z}_q^{n \times n}, \ u \in \mathbb{Z}_q^n

Then, \ \langle s^\top A + e, s^\top u + e' + \mu \cdot \frac{d}{2} \rangle \cong \langle r, r' \rangle

where \ r \in \mathbb{Z}_q^n, \ r' \in \mathbb{Z}_q \text{ by LWE}

Comparison with standard (i.e., primal) Regev:

<table>
<thead>
<tr>
<th>primal Regev</th>
<th>dual Regev</th>
</tr>
</thead>
<tbody>
<tr>
<td>public key:</td>
<td>sample \ s \in \mathbb{Z}_q^n, \ e \in \mathbb{Z}_q^n</td>
</tr>
<tr>
<td>secret key:</td>
<td>\mu</td>
</tr>
<tr>
<td>ciphertext:</td>
<td>sample \ r \in \mathbb{Z}_q^n</td>
</tr>
<tr>
<td>(Ar, s^\top Ar + e' + \mu \cdot \frac{d}{2})</td>
<td>(s^\top A + e, s^\top u + e' + \mu \cdot \frac{d}{2})</td>
</tr>
</tbody>
</table>

Trapdoor extension: 1. Suppose we have a gadget trapdoor \ G \in \mathbb{Z}_q^{n \times m} \text{ for } A \in \mathbb{Z}_q^{n \times n} \text{ (i.e., } AR = G).

Then, \ \begin{bmatrix} A & Ar \end{bmatrix} \text{ is a trapdoor for any extension } [A | Ar] \text{ of } A: [A | Ar] \begin{bmatrix} G \\ \mathbb{R} \end{bmatrix} = ARG.

2. For a matrix \ A \in \mathbb{Z}_q^{n \times n} \text{ and } U = AR + G \in \mathbb{Z}_q^{n \times m}, \text{ then } \begin{bmatrix} A | U \end{bmatrix} \begin{bmatrix} -\mathbb{R} \\ \mathbb{G} \end{bmatrix} = -ARR + U = -ARR + AR + G = G

Useful observation: Two possible trapdoors for a lattice \ [A | Ar] : either know a trapdoor for \ A \text{ or have a short } R \text{ such that } A_r = AR + G

This is a useful tool in many security proofs relying on the "puncturing" technique.

- Real scheme will use the trapdoor for \ A
- Reduction (simulation) will set up parameters so it knows \ R \text{ such that } A_r = AR + G

\Rightarrow \text{Since reduction likely will reduce to LWE (and no trapdoor is provided!)}

We will write \ SampleLeft (A, A_r, \text{td}_A, u, p) \text{ to denote an algorithm that samples a pre-image } u \text{ such that } [A | Ar] u = v \text{ and } \|u\| \leq \beta

We will write \ SampleRight (A, B, R, v, p) \text{ to denote an algorithm that samples a pre-image } u \text{ such that } [A | B] u = v \text{ and } \|u\| \leq \beta

provided that \ B = AR + G. \text{ In both cases, the allowable value of } \beta \text{ depends on the quality of the trapdoor (} \text{td}_A \text{ or } R)
We start with an abstraction for the homomorphic operations we have examined so far.

**Matrix embeddings:** Let \( A_1, \ldots, A_k \in \mathbb{Z}_p^{m \times n} \). Take \( x \in \{0,1\}^k \). We can "encode" \( x \) as follows:

\[
v_i = S^T(A_i + x_i G) + e_i^T
\]

\[
v_k = S^T(A_k + x_k G) + e_k^T
\]

**Addition:**
Given \( v_i = S^T(A_i + x_i G) + e_i^T \) and \( v_j = S^T(A_j + x_j G) + e_j^T \),

\[
\frac{v_i + v_j}{v_k} = S^T((A_i + A_j) + (x_i + x_j) G) + e_i^T + e_j^T
\]

**Multiplication:**
Given \( v_i = S^T(A_i + x_i G) + e_i^T \) and \( v_j = S^T(A_j + x_j G) + e_j^T \),

\[
\frac{x_j v_i - v_j G^{-1}(A_i)}{v_k} = S^T(x_j A_i + x_j G) + x_j e_i^T - S^T(A_j G^{-1}(A_i)) + x_j A_i + e_j G^{-1}(A_i)
\]

Using these elementary operations, we can define functions

\[
\text{Eval}_{PK}(C, A_1, \ldots, A_k) \rightarrow A_c
\]

\[
\text{Eval}_{CT}(C, A_1, \ldots, A_k, v_i, \ldots, v_k, x_i, \ldots, x_k) \rightarrow v_c
\]

such that for any collection of matrices \( A_1, \ldots, A_k \in \mathbb{Z}_p^{m \times n} \), if

\[
v_i = S^T(A_i + x_i G) + e_i^T \quad \text{for all } i \in \{k\},
\]

then if we take

\[
A_c \leftarrow \text{Eval}_{PK}(C, A_1, \ldots, A_k)
\]

\[
v_c \leftarrow \text{Eval}_{CT}(C, A_1, \ldots, A_k, v_i, \ldots, v_k, x_i, \ldots, x_k),
\]

it follows that

\[
v_c = S^T(A_c + C(x) \cdot G) + e_c^T
\]

Next, if \( A_i = A R_i - x_i G \), then observe that

\[
A_i + A_j = A(R_i + R_j - (x_i + x_j) G)
\]

\[
-AR_j G^{-1}(A_i) = -AR_j G^{-1}(A_i) + x_j A_i
\]

\[
-AR_j G^{-1}(A_i) = -AR_j G^{-1}(A_i) + x_j A_i = -AR_j G^{-1}(A_i) + x_j A_i - x_j G
\]

\[
= A(-R_j G^{-1}(A_i) + x_j A_i - x_j G)
\]

Then \( A_c = A R_c - C(x) \cdot G \)

\[ \leftarrow \text{function of } C, A, R_1, \ldots, R_k, \text{and } x \]
ABE from lattices: - use modular encoding to encode attributes:
\[ v_i = s^T(A_i + x_i G) + e_i^T \] enables computation of \[ s^T(A_c + C(x) G) + e_c^T \] \( \Rightarrow \) if \( C(x) = 0 \) this becomes \[ s^T A_c + e_c^T \] looks like part of a dual Regev encryption (with public key \( A_c \))

- encrypt a message \( \mu \) with respect to vector \( u \) (e.g., dual Regev style)
\[ s^T u + e^T + \mu \cdot \ell_c \]
- ciphertext is essentially dual Regev encryption with respect to \( A_c \) and \( u \); to decrypt, we need to give out a short vector \( \ell_c \) such that \( u^T A_c \ell_c \) seems challenging unless we have a trapdoor for \( A_c \)
- will use basis extension to make this easier: instead of encrypting to \( A_c \), we instead encrypt with respect to \([A | A_c]\) and let master secret key be trapdoor for \( A \) (which can be used to generate trapdoors for \([A | A_c]\) - these will be the ABE decryption keys

Setup (\( 2^\alpha \)): Sample \((A, \text{td}_A) \leftarrow \text{TrapGen}(1^\alpha)\)
\[ A_1, \ldots, A_\alpha \in \mathbb{Z}_q^{\cdot m} \quad \text{mpk} : (A, A_1, \ldots, A_\alpha, u) \]
\[ u \in \mathbb{Z}_q^{\cdot m} \quad \text{mk} : \text{td}_A \]

Encrypt (mpk, \( x \), \( \mu \)): Sample \( s \in \mathbb{Z}_q^{\cdot m}, e, e_1, \ldots, e_\alpha \leftarrow \mathbb{X}^\alpha, e^T \leftarrow X^T \) and compute
\[ v = s^T A + e^T \]
\[ v_1 = s^T (A_1 + x_1 G) + e_1^T \]
\[ \vdots \]
\[ v_\alpha = s^T (A_\alpha + x_\alpha G) + e_\alpha^T \]
\[ v' = s^T u + e^T + \mu \cdot \ell_c \]
\[ \text{ct} : (v_1, \ldots, v_\alpha, v', x) \]

KeyGen (mk, \( C \)): \( A_c \leftarrow \text{EvalPK}(C, A_1, \ldots, A_\alpha) \)
Output \( \ell_c \leftarrow \text{SampleLeft}(A, A_c, \text{td}_A, u, p) \) \( [p \text{ is some bound chosen to satisfy correctness and security] } \)
\[ \ell_c \] in particular \( [A | A_c] \ell_c = u \)

Decrypt (sk, \( \text{ct} \)): if \( C(x) = 0 \), output \( \bot \)
\[ v_c' = \text{EvalCT}(C, A_1, \ldots, A_\alpha, v_1, \ldots, v_\alpha, x_1, \ldots, x_\alpha) \]
Output \( v' = [v' | v_c'] \ell_c \)

Correctness: Take any \( x \in \text{Sym}_j \) and circuit \( C : \text{Sym}_j \rightarrow \{0,1\} \), where \( C(x) = 0 \). Consider decrypting a valid ciphertext with attribute \( x \) and message \( \mu \). Then,
\[ v = s^T A + e^T \]
\[ v_c = s^T (A_c + (x G)) + e_c^T = s^T A_c + e_c^T \]
\[ \Rightarrow [v' | v_c'] \ell_c = s^T [A | A_c] \ell_c + [e' | e_c] \ell_c \]
\[ = s^T u + [e' | e_c] \ell_c \]
\[ \Rightarrow v' = [v' | v_c'] \ell_c = s^T u + e' + \mu \cdot \ell_c \]
\[ \leq \mu \cdot \ell_c \]
Correct as long as \( |e| < \frac{\ell_c}{\mu} \)

\( e \) will be small since \( e', e_c^T \) are small, as is \( \ell_c \)

size grows with \( m \cdot \alpha \) where \( m \) is the depth of the computation

boundary by \( p \) (quality of trapdoor)
Security: Give a reduction to LWE. High level idea:
- Use LWE to argue that ciphertext components are uniformly random:
  \[ \begin{align*}
  & \text{observe: } A_i, A_1, ..., A_k, u \text{ one random so } \\
  & \text{will appear indistinguishable from uniform under LWE} \\
  & \text{problem: how do we simulate the decryption keys (we do not know trapdoor for }
  \\
  & s' u + e' + \mu \cdot \frac{L}{T}, A) \\
\end{align*} \]

  Proof technique will rely on the "puncturing" technique that will allow us to generate keys for all admissible functions; \( f(x^*) = 0 \) \( x^* \) is the challenge attribute and does not require trapdoor for \( A \).
- Will consider selective security game where \( A \) chooses its attribute in advance (implies adaptive security via complexity leveraging - though relies on subexponential hardness assumption) - major open problem is to obtain adaptive security without complexity leveraging / subexponential hardness.

We use a hybrid argument:

\( \text{Hyb}_r: \) Real game
\( \text{Hyb}_a: \) After adversary commits to \( x^* \), we set public parameters as follows:
\[ \begin{align*}
  A_i & \leftarrow R_i \cdot x^* \cdot G \in \mathbb{Z}_b^{n \times m} \\
  A_k & \leftarrow R_k \cdot x^* \cdot G \in \mathbb{Z}_b^{n \times m} \\
  u & \leftarrow \mathbb{Z}_b \\
\end{align*} \]
\[ \text{mpk} = (A, A_1, ..., A_k, u) \]
\( \text{msk} = T \)

For challenge ciphertext, compute:
\[ \begin{align*}
  & [v_1, v_2, ..., v_k] = s^T A [I | R_1, ..., R_k] + e^T [I | R_1, ..., R_k] \\
  & v' = s^T u + e' + \mu \cdot \frac{L}{T}
\end{align*} \]
\( \text{Hyb}_s: \) Switch challenge ciphertext to uniformly random vectors:
\[ \begin{align*}
  & v, v_1, ..., v_k \leftarrow \mathbb{Z}_b^n, v' \leftarrow \mathbb{Z}_b \\
  & c' = (v_1, v_2, ..., v_k, r^*)
\end{align*} \]

\( \text{Hyb}_a \) and \( \text{Hyb}_r \) are statistically indistinguishable by LHL. Namely, \( (A, Ar, ..., Ar_k) \approx (A, U_1, ..., U_k) \) if \( Ar, U_1, ..., U_k \in \mathbb{Z}_b^{n \times m} \) and \( R_1, ..., R_k \in \mathbb{Z}_b^{1 \times m} \) and \( m \geq 3 \log b \).

- Strictly speaking, we require a generalization of the LHL that says that \( (A, Ar, e^T R) \approx (A, U, e^T R) \)
Hyb₁ and Hyb₂ are computationally indistinguishable by LWE. Suppose there is an adversary A that can distinguish Hyb₁ and Hyb₂. We use A to construct an algorithm B for LWE:

1. Algorithm B receives an LWE challenge \((\langle a_1, \cdots, a_n \rangle, \langle v_1, \cdots, v_n \rangle) \in \mathbb{Z}_q^n \times \mathbb{Z}_q^n\) where \(b = s^T a + e \pmod{q}\) or \(b = s^T e \pmod{q}\).
2. Let \(x^*\) be the attribute \(A\) chooses for the semantic security game.
3. Algorithm B samples \(R_1, \ldots, R_n \in \mathbb{Z}_q^n\) and sets \(A_i = A_{i-1} R_i - x^*_i G\) and sets the mpk as \((A, A_1, \ldots, A_n, u)\) and gives mpk to A.
4. Suppose A makes a key-generation query on a circuit \(C\). It must be the case that \(C(x^*) = 1\). This means that
   \[
   A_c = \text{EvalPK}(C, A_1, \ldots, A_n) \\
   = \text{EvalPK}(C, A_{i-1} R_i - x^*_i G, \ldots, A_{i-1} R_i - x^*_i G) \\
   = A_{c_{i-1}} + C(x^*) G \\
   = A_{c_{i-1}} + G
   \]
   [This will allow the reduction to sample keys whenever \(C(x^*) = 1\), but not when \(C(x^*) = 0\).]

   By design, \(G\) is small. To simulate a key, algorithm B needs to compute a short \(r_c\) such that
   \[
   [A] A_{c_{i-1}} r_c = u. \text{ This is possible since } B \text{ knows } r_c \text{ such that } A_{c_{i-1}} = A_{c_{i-1}} r_c + G \text{ so } B \text{ computes}
   
   r_c \leftarrow \text{SampleRight}(A, A_{c_{i-1}}, r_c, u, b^{'}) \text{ which is indistinguishable from a real key (output by the actual}
   
   \text{KeyGen algorithm)}
   \]
5. For the challenge ciphertext, set
   \[
   v = b \quad \text{and} \quad v_i = b^T R_i \quad \text{for } i \in [L] \\
   v' = b' + \mu \cdot [\frac{1}{L}]
   \]
   and output \(ct = (v, v_1, \ldots, v_L, v')\).

Two possibilities:
- Suppose \(b^T = s^T A + e^T\) and \(b' = s^T u + e'\). Then,
  \[
  v_i = (s^T A + e^T) R_i = s^T A R_i + e^T R_i \\
  = s^T A (R_i + x_i^* G) + e^T R_i
  \]
  Thus, ciphertexts distributed exactly as in Hyb₁.
- Suppose \(b^T\) and \(b'\) are uniformly random. Then, by LHL, all of the \(v_i\) are uniform over \(\mathbb{Z}_q^n\) and the
  ciphertext is distributed according to the specification in Hyb₂.

Thus, assuming LWE, Hyb₁ and Hyb₂ is computationally indistinguishable.