Addition: \[ C_1 + C_2 \text{ is encryption of } \mu_1 + \mu_2: \]
\[ C_1 + C_2 = A(R_1, R_2) + (\mu_1, \mu_2)G \]
\[ \text{New error: } R_x = R_1 + R_2, \quad ||R_x||_\infty \leq ||R_1||_\infty + ||R_2||_\infty \]

Multiplication: \[ C_1, G^{-1}(C_2) \text{ is encryption of } \mu_1 \cdot \mu_2: \]
\[ C_1G^{-1}(C_2) = (AR_1 + \mu_1 G^{-1}(C_2)) \]
\[ = AR_1G^{-1}(C_2) + \mu_1 G^{-1}(C_2) \]
\[ = AR_1G^{-1}(C_2) + \mu_1 C_2 \]
\[ = AR_1G^{-1}(C_2) + \mu_1 (AR_2 + \mu_2 G) \]
\[ = A(R_1G^{-1}(C_2) + \mu_1 R_2) + \mu_1 \mu_2 G \]
\[ \text{New error: } R_x = R_1 G^{-1}(C_2) + \mu_1 R_2, \quad ||R_x||_\infty \leq ||R_1||_\infty \cdot m + ||R_2||_\infty \]

After computing \( d \) repeated squarings: noise is \( m(\text{poly}) \) for correctness, require that \( G > 4\text{poly} \cdot ||R_1||_\infty \), so bit-length of \( G \) scales with multiplicative depth of circuit:
\[ \Rightarrow \text{also requires super-poly modulus when } d = \omega(1) \]
(Stronger assumption needed)

But not quite fully homomorphic encryption: we need a bound on the (multiplicative) depth of the computation.

From SWHE to FHE: The above construction requires imposing an a priori bound on the multiplicative depth of the computation. To obtain fully homomorphic encryption, we apply Gentry’s brilliant insight of bootstrapping.

High-level idea: Suppose we have SWHE with following properties:
1. We can evaluate functions with multiplicative depth \( d \).
2. The decryption function can be implemented by a circuit with multiplicative depth \( d' < d \).

Then, we can build an FHE scheme as follows:
- Public key of FHE scheme is public key of SWHE scheme and an encryption of the SWHE decryption key under the SWHE public key.
- We now describe a ciphertext refreshing procedure:
  - For each SWHE ciphertext, we can associate a “noisy” level that keeps track of how many more homomorphic operations can be performed on the ciphertext (while maintaining correctness).
  - For instance, we can evaluate depth-\( d \) circuits on fresh ciphertexts; after evaluating a single multiplication, we can only evaluate circuits of depth-\( (d-1) \) and so on...
  - The refresh procedure takes any valid ciphertext and produces one that supports depth-\( (d-d') \) homomorphism; since \( d > d' \), this enables unbounded (i.e., arbitrary) computations on ciphertexts.

Idea: Suppose we have a ciphertext \( \text{ct} \) where \( \text{Decrypt}(sk, \text{ct}) = X \).

To refresh the ciphertext, we define the Boolean circuit \( C_{\text{ct}} : 0,1 \mapsto 0,1 \) where \( C_{\text{ct}}(sk) := \text{Decrypt}(sk, \text{ct}) \) and homomorphically evaluate \( C_{\text{ct}} \) on the encryption of \( sk \).
\[ \Rightarrow \text{Encrypt}(pk, sk) \mapsto \text{Encrypt}(pk, C_{\text{ct}}(sk)) \]
\[ \Rightarrow \text{Refresh ciphertext still supports } d-d' \text{ levels of multiplication} \]
Security now requires that the public key includes a copy of the decryption key:

\( \text{FHE without circular security from LWE} \)

(possible from IO)

\( \Rightarrow \text{specifically: } s^T C \approx \mu^T s^T G \)

Let's take a closer look at bootstrapping for GSW encryption:

\[ \text{pk: } A \in Z_q^n \]

\[ \text{Enc}(pk, m) : C = AR + \mu s^T G \]

\[ \text{sk: } S \in Z_q^n \]

\[ \text{Dec}(sk, C) = \text{compute } s^T C \text{ and round.} \]

Consider a computation with multiplicative depth \( d \) that can support by setting \( q > m(2^d) \)

Consider depth of circuit implementing GSW decryption: circuit has ciphertext column \( C \in Z_q^n \) hard-wired and takes secret key \( s \in Z_q^n \) as input

Need to compute round \( s^T C \bmod q \) as a Boolean circuit:

\[ s^T C = \sum_{i=1}^{n} \sum_{j=1}^{d} s_i \cdot \left( 2^j \cdot c_{i,j} \bmod q \right) \]

- We can write \( i \)-th bit of \( i \)-th component of \( s \)

\[ \text{computing } s^T C \text{ over the integers can be computed by } O(n \log q) \text{ additions of values with } O(n \log n \log \log q) \text{ bits} \]

Using an addition tree, this can be computed by a circuit of depth \( O(n \log n \log \log q) \)

- Given \( s^T C \bmod q \) over the integers, need to reduce mod \( q \)

\[ s^T C \bmod q = s^T C \bmod q \text{ is just rounding} \]

- Recovering \( s^T C \bmod q \) is just rounding (checking most significant bits of binary representation - constant depth)

\[ \text{overall depth: } O(n \log n \log \log q) = O(n \log n) \text{ since we always have } q < 2^n \text{ (for security)} \]

To bootstrap, it suffices to support multiplicative depth \( O(\log n) \).

For correctness, we thus require that \( q = n(\log n) \), so this is easily satisfiable.

\[ \Rightarrow \text{FHE from LWE \& circular security} \]

But... we did require super-polynomial modulus for correctness: \( q > n(\log n) \).

\[ \Rightarrow \text{Hardness based on worst-case lattice problems with super-polynomial approximation factor - stronger assumption than for PKE} \]

Can do better by relying on asymmetric noise growth of GSW multiplication:

\[ C_i = AR_i + \mu_i G \]

\[ C_j = AR_j + \mu_j G \]

\[ = A(R_i G^i(C_i) + \mu_i R_i) + \mu_j G \]

\[ = A(R_i G^i(C_i) + \mu_i R_i) + \mu_j G \]

Observe: \( R_i \) only needs \( R_i \), dependence on \( R_i \) is additive

\[ \Rightarrow \text{each multiplication increases noise by additive factor } 3 \cdot m \]

Suppose we have \( C_1, \ldots, C_t \) with noise \( R_1, \ldots, R_t \) where \( \| R_i \|_2 \leq B \) for all \( i \leq t \).

Consider sequence of homomorphic multiplications where each multiplication involves one of \( C_1, \ldots, C_t \). Then, noise accumulation after \( T \) multiplications is bounded by \( T \cdot B \cdot m \)

\[ \Rightarrow \text{each multiplication increases noise by additive factor } B \cdot m \]

Key takeaway: if input to every multiplication is a fresh ciphertext, then noise growth is additive not multiplicative in the depth.
Asymmetric noise growth extremely useful both theoretically and practically! Very efficient private information retrieval protocols base security on vector assumptions (% PKE!).

How to exploit in the case of bootstrapping? Rounded inner product does not necessarily have this form...

**Branching programs**: one way to capture space-bounded computations

\[ \text{State can be expressed as an indicator vector } v \in \{0,1\}^n. \]

\[ \text{Transition can be expressed as matrix product corresponding to} \]

- **Important**: some bit of input can be read multiple times

**Theorem (Barrington)**: Let \( C : \{0,1\}^5 \to \{0,1\}^2 \) be a Boolean circuit with depth \( d \) and fan-in \( 2 \) (i.e., each gate has two inputs). Then, we can compute \( C \) using a permuted branching program of length \( l \leq 4^d \) and width 5.

In particular, if \( d = O(\log n) \), the length of the branching program is \( l \leq 4^d = 4^{O(\log n)} = \text{poly}(n) \).

Let \( \text{BP} = (\text{inp}, M_{i,n}, M_{i,i}) \) be a branching program on input \( x \in \{0,1\}^n \) with length \( l \) and width \( w \):

- \( \text{inp} : \{l\} \to \{w\} \) specifies which bit of input to read in given layer
- \( M_{i,n}, M_{i,i} \in \{0,1\}^w \) and specifies transition for reading 0 or 1 in layer \( i \)
- Let \( v_0 \in \{1\}^l \) be initial state.
- Let \( t \in \{0,1\}^w \) be indicator for accepting states in output layer

- Can compute \( \text{BP}(x) \) as:
  \[ \text{BP}(x) = t^T . A_{l,\text{inp}(x)} . A_{l-1,\text{inp}(x)} . \ldots . A_{1,\text{inp}(x)} . v_0 \]

To compute homomorphically: given fresh encryptions of bits of \( x \), homomorphically compute:

\[ A_{i,\text{inp}(x)} = x_i . A_{i-1} + (1-x_i) . A_{i,0} \quad \text{if encryptions of } x \text{ have noise at most } B \]

then encryptions of \( A_{i,\text{inp}(x)} \) has noise at most \( 2B \)

**Homomorphically compute sequence of products**

\[ t^T . A_{l,\text{inp}(x)} . A_{l-1,\text{inp}(x)} . \ldots . A_{1,\text{inp}(x)} \cdot v \]

Observe: Each product involves at least one "fresh" ciphertext \( A_{i,\text{inp}(x)} \), so by asymmetric noise growth of GSW multipliers, overall noise is \( l \cdot B \cdot \text{poly}(n) \).

Decryption circuit has depth \( O(\log n) \) so associated branching program \( \text{BP} \) has length \( 4^d = \text{poly}(n) \).

\( \Rightarrow \) Overall noise from bootstrapping: \( l \cdot B \cdot \text{poly}(n) = \text{poly}(n) \)

For correctness, it now suffices to use \( q = \text{poly}(n) \), so can get EHE with polynomial modulus \( q \). Further improvements possible!