Phase 3: Implementation + Security Analysis

Security Analysis of BFV & CKKS Homomorphic Encryption Schemes

CS6530 – Applied Cryptography

Introduction to Homomorphic Encryption

Main Concept:

> Homomorphic Encryption (HE) enables computations directly on encrypted data without decryption, preserving data privacy in untrusted environments.

Classification of HE Schemes:

- **Partial HE (PHE):** Supports only addition OR multiplication
- **Somewhat HE (SHE):** Limited operations before noise overwhelms the ciphertext
- **Leveled FHE:** Supports fixed-depth circuits with predetermined operation limits
- **Fully HE (FHE):** Unlimited computations via bootstrapping techniques

Key Benefits:

- Process sensitive data while maintaining end-to-end encryption
- Secure cloud computations without trusting service providers
- Enable privacy-preserving analytics and machine learning
- Regulatory compliance with data protection laws (HIPAA, GDPR)

BFV Scheme Overview

Full Name:

> Brakerski-Fan-Vercauteren Scheme

Primary Characteristics:

- **Designed for:** Exact integer arithmetic (no approximation)
- **Message space:** Integers modulo plaintext modulus t
- **Ciphertext structure:** Polynomials in ring $Z_q[x]/(x^n + 1)$
- **Classification:** Leveled Fully Homomorphic Encryption

Supported Operations:

- Homomorphic addition
- Homomorphic multiplication

Typical Applications:

- Financial computations requiring precision
- Encrypted medical records
- Secure voting systems
- Encrypted database queries

BFV Mathematical Structure

Polynomial Rings:

- **Plaintext Ring:** $R_t = Z_t[x]/(x^N + 1)$
- **Ciphertext Ring:** $R_q = Z_q[x]/(x^N + 1)$
- N is polynomial degree, t is plaintext modulus, q is ciphertext modulus

Key Generation:

- Sample secret key: $s \leftarrow \chi$ (error distribution)
- Sample random polynomial: $a \leftarrow R_q$
- Sample error: $e \leftarrow \chi$
- **Public Key:** $(pk_0, pk_1) = (-as + e, a)$
- **Secret Key:** s

Message Encoding:

- Scale plaintext: $\tilde{m} = \lfloor q/t \rfloor \cdot m$
- Ensures proper decryption after operations

Encryption Process:

- Sample randomness: $v, e_1, e_2 \leftarrow \chi$
- Compute ciphertext components:

```
c₀ = pk₀·v + e₁ + m̃
c₁ = pk₁·v + e₂
Ciphertext: c = (c₀, c₁)
```

Decryption Process:

```
1. m' = c₀ + c₁·s (mod q)
2. m = ⌊(t/q)·m'⌉ (mod t)
```

Homomorphic Operations:

- **Addition:** $(c_0, c_1) + (d_0, d_1) = (c_0 + d_0, c_1 + d_1)$
- Noise grows linearly
- **Multiplication:** Requires relinearization
- Produces 3 components initially
- Noise grows quadratically

BFV Noise Growth & Decryption Error Condition

Decryption Recovery:

- After decryption, BFV recovers: $m' = \tilde{m} + e_{total} \pmod q$
- Where e_{total} is the accumulated noise from all operations

Critical Correctness Condition:

$|e_{total}| < q/(2t)$
This is the noise budget threshold for correct decryption

Why This Bound Matters:

- Decoding process transforms ciphertext back to message space
- Noise must be small enough to avoid "wrap-around" modulo q
- Exceeding threshold causes bits to flip in decoded result

Key Design Principle:

- Ciphertext modulus q must be sufficiently large relative to plaintext modulus t
- Must accommodate all noise growth throughout computation circuit
- **Balance:** Larger q → more noise capacity but requires larger n for security

Decoding Process:

```
Recovered message: m̂ = ⌊(t/q)·m'⌉
Substituting m': m̂ = ⌊m + (t/q)·e_total⌉
```

Case 1: Noise Within Bound

$|e_{total}| < q/(2t)$
The term $(t/q)\cdot e_{total}$ remains small (< 0.5)
Rounding $\lfloor\cdot\rceil$ correctly recovers m
Result: Successful decryption ✓

Case 2: Noise Exceeds Bound

$|e_{total}| \geq q/(2t)$
Coefficients "wrap around" modulo q
Centered noise representation exceeds $q/(2t)$
The term $(t/q)\cdot e_{total}$ pushes rounding past 0.5
Rounding flips to incorrect plaintext value
Result: Decryption failure ✗

Practical Implication:

- Circuit depth is fundamentally limited by noise budget
- Deep circuits require larger q and correspondingly larger n
- This trade-off defines BFV's computational limits

CKKS Scheme Overview

Full Name:

> Cheon-Kim-Kim-Song (CKKS) Scheme

Primary Characteristics:

- **Designed for approximate arithmetic** on real/complex numbers
- Encodes floating-point vectors as complex polynomials
- Classification: **Approximate Leveled FHE**
- Trades minor precision loss for computational efficiency

Key Innovation:

> Controlled approximation through scaling factors and rescaling operations

Ideal Use Cases:

- Machine learning inference
- Statistical analytics
- Signal processing
- Applications tolerant to small numerical errors

CKKS Scaling and Rescaling

Encoding Real Numbers with Scaling:

- **Scaling factor Δ** converts reals to large integers
- Example: 3.14159 with $\Delta = 2^{40} \approx 3{,}454{,}217{,}652{,}188$

$$\tilde{m} = \lfloor m \cdot \Delta \rceil$$

Scale Behavior in Operations:

- **Addition:** Scale unchanged

$$(m_1 \cdot \Delta) + (m_2 \cdot \Delta) = (m_1 + m_2) \cdot \Delta$$

- **Multiplication:** Scale squares

$$(m_1 \cdot \Delta) \times (m_2 \cdot \Delta) = (m_1 \cdot m_2) \cdot \Delta^2$$

The Rescaling Operation:

- **Purpose:** Manage scale explosion
- Returns scale from Δ^2 back to Δ

Rescaling Mechanics:

```
Rescale(c) = c/Δ
```

- **Effects:**
- 1. Scale adjustment: $\Delta^2 \to \Delta$
- 2. Drops one prime from modulus chain ($q \to q/p$)
- 3. Reduces noise magnitude proportionally
- 4. Preserves approximate plaintext value

Modulus Chain Consumption:

- Each multiplication + rescale consumes one level
- With modulus $q = q_0 \cdot q_1 \cdot q_2 \cdot q_3$:
- → Supports 3 multiplications (4 levels - 1 encoding)

Trade-offs:

- Larger Δ → better precision but requires larger q
- More rescaling levels → deeper circuits but slower
- Balance between accuracy, depth, and performance

Microsoft SEAL Library Overview

Open-source HE Library

- **MIT License** — Industry-standard crypto library
- **Supported Schemes:** BFV (exact) and CKKS (approximate)
- **Backend:** RNS representation + NTT for polynomial ops

Key Features

- Parameter validation against HE security standards
- SIMD batching for parallel operations
- Pre-tuned parameter sets for 128/192/256-bit security
- Comprehensive API for complete HE workflows

```
// BFV setup example
EncryptionParameters parms(scheme_type::bfv);
parms.set_poly_modulus_degree(8192);
parms.set_coeff_modulus(CoeffModulus::BFVDefault(8192));
parms.set_plain_modulus(PlainModulus::Batching(8192, 20));
```

Implementation Performance

8192 Standard poly degree	**~50-60 bits** Prime moduli size
128-bit Default security	**BFV/CKKS** Supported schemes

Core SEAL Components We Use

Parameter Setup

- **EncryptionParameters** — Configures scheme type, polynomial degree, modulus chain
- **SEALContext** — Validates parameter security levels and builds internal structures

Key Management

- **KeyGenerator** — Creates secret/public key pairs
- **RelinKeys** — For multiplication operations
- **GaloisKeys** — For rotation operations on encrypted vectors

Encoding & Encryption

- **BatchEncoder** (BFV) — Encodes integer vectors
- **CKKSEncoder** (CKKS) — Encodes real/complex vectors
- **Encryptor** — Encrypts plaintexts using public or secret key

```
// Basic SEAL usage pattern
auto context = SEALContext(parms); // validate params
KeyGenerator keygen(context);
auto secret_key = keygen.secret_key();
auto public_key = keygen.public_key();
auto relin_keys = keygen.relin_keys();
```

1. **Evaluation Operations**
   Evaluator performs homomorphic operations: add, multiply, rotate, relinearize, rescale. Core of all HE computations.
2. **Decryption & Results**
   Decryptor recovers plaintext from ciphertexts using secret key.
3. **Monitoring**
   Use print_noise_budget() for BFV and ciphertext.scale() for CKKS to track remaining noise tolerance.

Configuring Circuit Depth in SEAL

Depth Control Parameters

- **Polynomial Modulus Degree (N):** Higher values allow deeper circuits but increase computational cost
- **Coefficient Modulus Chain:** Each prime in chain supports one multiplication level
- **BFV Constraints:** Choose t and q so q/t is large for noise margin

Scheme-Specific Behavior

- **CKKS:** Each mul+rescale consumes one modulus level; explicit scale management
- **BFV:** No automatic rescaling; faster noise accumulation; more restrictive depth limits
- **Relinearization:** Apply after each multiplication to control noise and ciphertext size

```
// CKKS modulus chain configuration
size_t poly_modulus_degree = 16384;
parms.set_poly_modulus_degree(poly_modulus_degree);
// Prime chain for 5 multiplications
parms.set_coeff_modulus(CoeffModulus::Create(
poly_modulus_degree, {60, 40, 40, 40, 40, 60}));
```

Circuit Depth Capabilities

4096 N: ~2-3 muls	**8192** N: ~4-5 muls
16384 N: ~7-9 muls	**32768** N: ~10+ muls

```
// Depth monitoring (CKKS)
// Each operation consumes levels:
auto level = context.get_context_data(
encrypted.parms_id())->chain_index();
// Scale tracking:
double scale = encrypted.scale();
```

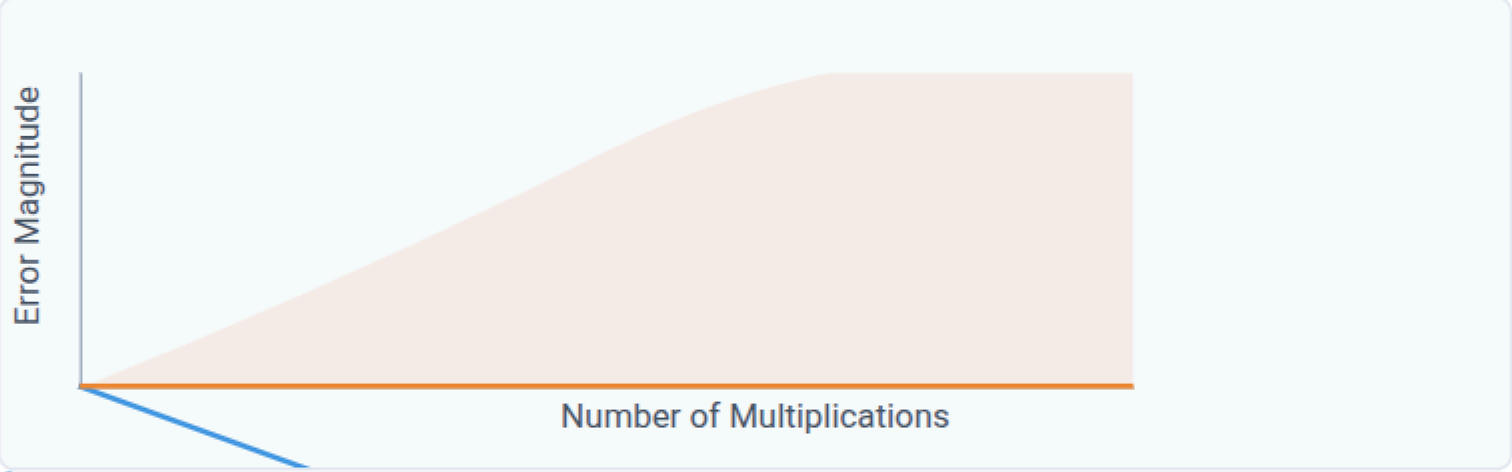
Error Accumulation in CKKS (Experimental)

Error Sources in CKKS

- **Approximation error** — Inherent to encoding/scaling operations
- **RLWE noise** — Cryptographic noise from encryption
- **Cumulative effect** — Both sources compound with each operation

Observed Behavior

- Error grows significantly with multiplicative depth
- Scale decreases after each rescaling operation
- Modulus levels are consumed progressively
- Past critical threshold → significant accuracy loss




```
// CKKS error monitoring
auto context = SEALContext(parms);
// Check remaining scale after operations
double current_scale = encrypted.scale();
if (current_scale < 1e15) {
cout << "Warning: Scale degrading" << endl;
}
```

Error Indicators in SEAL

scale()
Monitor decreasing scale

parms_id
Track chain position

~10^{-6}
Initial relative error

~10^{-2}
Critical error threshold

Security-Critical Parameter Selection

Core Security Parameters

- **Polynomial Modulus Degree (N):** Higher values strengthen RLWE security but increase computation cost
- **Coefficient Modulus (q):** Larger q provides more noise capacity but can weaken security if N is insufficient
- **Plaintext Modulus (t):** BFV-specific; must maintain proper ratio with coefficient modulus
- **Scaling Factor (Δ):** CKKS-specific; affects precision and noise consumption rate

Security Standard Compliance

- All parameters must guarantee ≥128-bit RLWE security level
- SEALContext validates parameter security automatically
- Follow HomomorphicEncryption.org standard guidelines

Critical Security Constraints

Parameter choices directly impact RLWE problem hardness and scheme security

Recommended Parameters

≥ 4096 Minimum N value	**128-bit** Minimum security
q << 2^N Modulus constraint	**t << q** BFV plaintext ratio

1. SEAL presets: N ∈ {4096, 8192, 16384, 32768}
2. Maximum $\log_2(q)$ bits by security level:
   N=4096: 109 bits (128-bit security)
3. Use SEALContext.parameters_validated() to verify security

Security Foundation: RLWE Problem

Ring Learning With Errors (RLWE):

- Mathematical hardness assumption underlying both BFV and CKKS
- Defined over polynomial rings $R_q = Z_q[x]/(x^N + 1)$
- N is a power-of-two integer (typically 2048-32768)

```
Problem: Given (a, b = a·s + e mod q)
where s, e are small (from error distribution)
Distinguish from uniform random pairs (a, u)
```

Relation to Lattice Problems:

- RLWE reduces to Shortest Vector Problem (SVP)
- SVP is computationally hard for classical/quantum computers
- Provides concrete security with appropriate parameters

Security Guarantees:

Classical Security: No known efficient algorithm for breaking RLWE at standard parameters

Post-Quantum Security: Resistant to attacks by quantum computers (unlike RSA, ECC)

Attack Vectors:

- BKZ lattice reduction algorithms
- Potential weaknesses if parameters poorly chosen
- Security depends on N, q, error width σ, and secret distribution

Current Consensus:

- RLWE-based schemes remain secure with proper parameters
- HE Standard recommendations ensure ≥128-bit security
- Actively researched but no significant practical breaks

Vulnerability #1: Parameter Misconfiguration

Critical Issue: Incorrect parameters directly undermine RLWE hardness and scheme security

Common Misconfigurations:

- **Insufficient Polynomial Degree:** Choosing $n < 4096$ makes schemes vulnerable to lattice reduction attacks
- **Oversized Plaintext Modulus (BFV):** Setting t too close to q risks information leakage via modulus reduction
- **Improper Coefficient Modulus:** Using insecure bit-lengths or violating ratio requirements between q and n
- **Non-NTT-Friendly Primes:** Selecting primes that don't support efficient Number Theoretic Transform

Impact:

- Security level drops below acceptable threshold
- Attacker may recover secret key through mathematical attacks
- Practical attacks become feasible within reasonable compute bounds

Mitigation:

SEAL provides built-in parameter validation against HE Standard
Always use SEALContext to verify security levels
Follow HomomorphicEncryption.org standard guidelines for parameter selection
Use library's recommended parameter sets for specific security levels

Vulnerability #2: Noise Overflow & Decryption Failure

Nature:

> Correctness/availability failure (not a confidentiality breach) — when noise exceeds threshold, decryption produces incorrect results

Noise Growth Patterns:

- BFV: Rapid noise increase due to lack of rescaling
- Addition: Linear noise growth
- Multiplication: Quadratic noise growth
- CKKS: More controlled noise growth with rescaling
- Still accumulates over deep circuits
- Precision degrades alongside noise increase

Causes:

- Deep circuits exceeding noise budget
- Missing relinearization after multiplication
- Large plaintext magnitudes (especially in BFV)
- Insufficient modulus size relative to circuit depth

Mitigations:

- Reduce multiplicative depth through circuit redesign
- Relinearize after each multiplication
- Apply modulus switching (BFV) or rescaling (CKKS)
- Increase polynomial modulus degree (N) and modulus chain
- Use alternative algorithms (e.g., Horner's method) to minimize depth

Vulnerability #3: Precision Degradation in CKKS

Symptom:

- Loss of significant bits in computation results
- Machine learning accuracy drop in encrypted inference
- Unstable analytics and statistical results
- Results that fall outside acceptable error margins

Causes:

- Short modulus chain for deep computation circuits
- Small scaling factor (Δ) providing insufficient precision
- Excessive rescaling operations and rotations
- Scale mismatch before addition operations

Detection Signs:

- Rapidly shrinking ciphertext.scale() values
- Few remaining levels in modulus chain
- Large relative error after decryption

Mitigation Strategies:

- Use $\Delta \approx 2^{40}$–2^{60} for adequate precision
- Implement balanced 40-bit primes in modulus chain
- Align scales before addition operations
- Minimize rescale count through circuit optimization
- Apply model quantization/normalization techniques
- Use CKKS bootstrapping where available

Conclusion: Key Takeaways

- **BFV vs CKKS:** BFV provides exact integer arithmetic while CKKS offers efficient approximate computation on real/complex numbers
- **Security Foundation:** Both schemes rely on RLWE hardness assumption, providing post-quantum security with proper parameters
- **Depth Limits:** Circuit depth is dictated by noise budget (BFV) and modulus chain length (CKKS)
- **Implementation:** Microsoft SEAL provides secure defaults, parameter validation, and noise monitoring tools

> Best Practices for Secure Implementation

- Always validate parameters against HE Standard recommendations (≥128-bit security)
- Monitor noise budgets and scales throughout computation
- Design circuits within depth limits and relinearize after multiplications
- Test decryption accuracy throughout development lifecycle
- Choose parameter sets carefully based on application requirements (precision vs. performance)

Thank You