Background of Fully Homomorphic Encryption

CKKS and TFHE

HU Qi
PhD Student in HKU

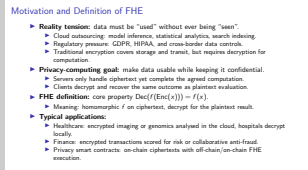
The University of Hong Kong

November 7, 2025

Background of Fully Homomorphic Encryption
CKKS and TFHE
HU Qi
PhD Student in HKU
The University of Hong Kong
November 7, 2025

Motivation and Definition of FHE

- **Reality tension:** data must be "used" without ever being "seen".
  - Cloud outsourcing: model inference, statistical analytics, search indexing.
  - Regulatory pressure: GDPR, HIPAA, and cross-border data controls.
  - Traditional encryption covers storage and transit, but requires decryption for computation.
- **Privacy-computing goal:** make data usable while keeping it confidential.
  - Servers only handle ciphertext yet complete the agreed computation.
  - Clients decrypt and recover the same outcome as plaintext evaluation.
- **FHE definition:** core property $\mathrm{Dec}(f(\mathrm{Enc}(x))) = f(x)$.
  - Meaning: homomorphic f on ciphertext, decrypt for the plaintext result.
- **Typical applications:**
  - Healthcare: encrypted imaging or genomics analysed in the cloud, hospitals decrypt locally.
  - Finance: encrypted transactions scored for risk or collaborative anti-fraud.
  - Privacy smart contracts: on-chain ciphertexts with off-chain/on-chain FHE execution.



Start with the question: why compute under encryption? Cloud inference, analytics, and risk scoring clash with GDPR, HIPAA, and cross-border rules that forbid revealing plaintext. Traditional encryption protects storage and transit yet forces decryption right before computation—the riskiest moment. Privacy computing keeps servers on ciphertext while clients decrypt to recover the same output; the identity $\mathrm{Dec}(f(\mathrm{Enc}(x))) = f(x)$ uses the same logical f, with ciphertexts processed through matching homomorphic operations. FHE keeps data encrypted end-to-end, removing the server-side decrypt-compute step and shrinking leakage on untrusted infrastructure. Applications span encrypted medical AI, financial risk scoring on ciphertexts, encrypted inputs to cloud models with local decryption, and privacy smart contracts.

Evolution of Fully Homomorphic Encryption

- 2009: Gentry's breakthrough first-generation scheme based on ideal lattices.
- 2011–2014: Second generation (BGV, BFV) improves efficiency via leveled FHE and bootstrapping optimizations.
- 2016: CKKS introduces approximate arithmetic for real-number workloads such as machine learning inference.
- 2016+: TFHE enables fast gate-by-gate Boolean computation with rapid bootstrapping.
- Current landscape: hybrid approaches, GPU acceleration, and practical libraries (HElib, SEAL, Concrete, OpenFHE).

Evolution of Fully Homomorphic Encryption
- 2009: Gentry's breakthrough first-generation scheme based on ideal lattices.
- 2011–2014: Second generation (BGV, BFV) improves efficiency via leveled FHE and bootstrapping optimizations.
- 2016: CKKS introduces approximate arithmetic for real-number workloads such as machine learning inference.
- 2016+: TFHE enables fast gate-by-gate Boolean computation with rapid bootstrapping.
- Current landscape: hybrid approaches, GPU acceleration, and practical libraries (HElib, SEAL, Concrete, OpenFHE).

Here I sketch the trajectory of FHE. After Gentry's 2009 construction, leveled schemes like BGV and BFV reduced overhead. CKKS later delivered approximate arithmetic suited for ML, while TFHE focused on fast Boolean gates with efficient bootstrapping. Today the ecosystem blends these ideas, pushing toward practical deployments with optimized libraries and hardware acceleration.