Fully Homomorphic Encryption Lab

Archisman Dutta

Fully Homomorphic Encryption: A primer

Fully Homomorphic Encryption (FHE) is a cryptographic primitive that allows performing computations on encrypted data without decrypting it first.

Fully Homomorphic Encryption: A primer

Fully Homomorphic Encryption (FHE) is a cryptographic primitive that allows performing computations on encrypted data without decrypting it first.

Formally, an FHE scheme $\mathscr{H}$ is a 4-tuple of algorithms $(\mathrm{KeyGen},\ \mathrm{Enc},\ \mathrm{Dec},\ \mathrm{Eval})$ such that, for any $(\mathrm{pk},\ \mathrm{sk}) \overset{\$}{\leftarrow}$ KeyGen(λ), plaintext m and ciphertext $c = \mathrm{Enc}(\mathrm{pk},\ \mathrm{m})$, the following equality holds for all polynomial circuits C:

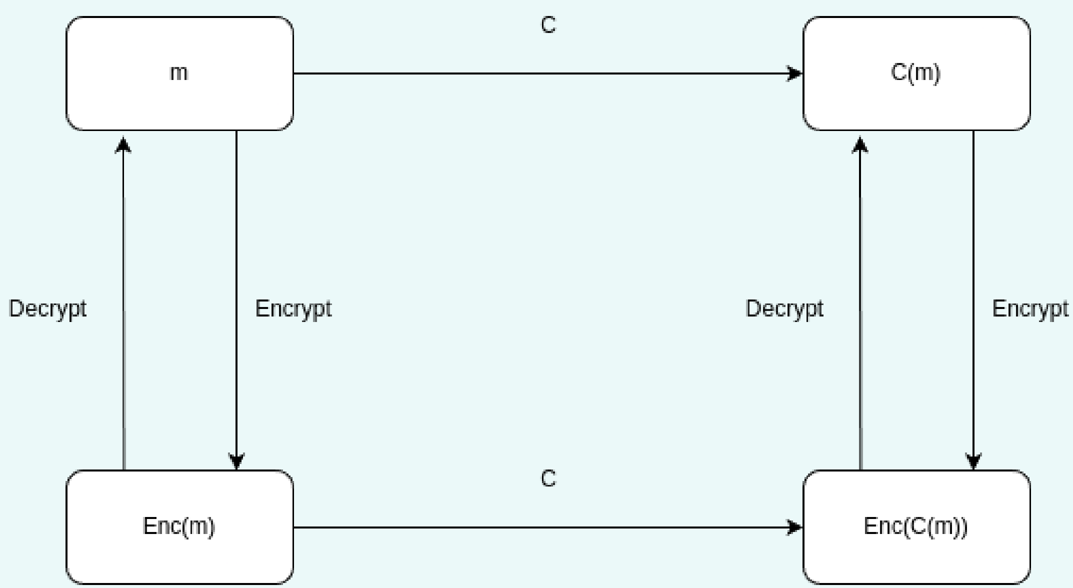
$$\mathrm{Dec}(\mathrm{sk}, \mathrm{Eval}(\mathrm{pk}, c, C)) = C(m)$$

Fully Homomorphic Encryption: A primer

Fully Homomorphic Encryption (FHE) is a cryptographic primitive that allows performing computations on encrypted data without decrypting it first.

Formally, an FHE scheme $\mathscr{H}$ is a 4-tuple of algorithms ($\mathrm{KeyGen},\ \mathrm{Enc},\ \mathrm{Dec},\ \mathrm{Eval}$) such that, for any $(\mathrm{pk},\ \mathrm{sk}) \overset{\$}{\leftarrow}$ KeyGen(λ), plaintext m and ciphertext $c = \mathrm{Enc}(\mathrm{pk},\ \mathrm{m})$, the following equality holds for all polynomial circuits C:

$$\mathrm{Dec}(\mathrm{sk}, \mathrm{Eval}(\mathrm{pk}, c, C)) = C(m)$$



Types of FHE schemes

FHE was first proposed by Gentry in [1] using ideal lattices - this finally enabled additive as well as multiplicatively homomorphic operations on encrypted data.

Types of FHE schemes

FHE was first proposed by Gentry in [1] using ideal lattices - this finally enabled additive as well as multiplicatively homomorphic operations on encrypted data.

Since then, a number of schemes have cropped up which promise more efficient computations in such a way that the resultant noise stays below a threshold (via bootstrapping or modulus switching, for instance) and the resultant ciphertext size does not grow too large (via relinearization). Most of these schemes are based on the hardness assumption of the Ring-LWE problem.

Types of FHE schemes

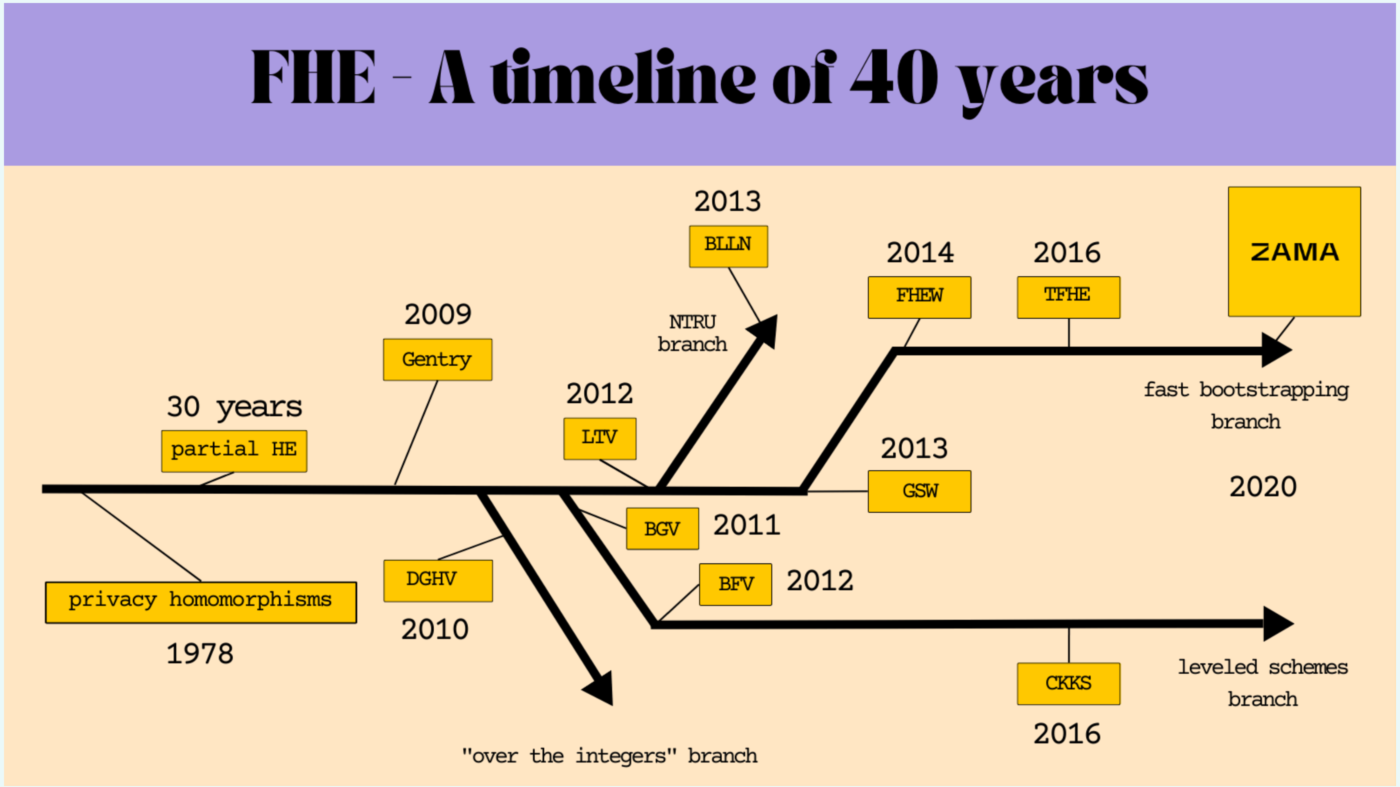
FHE was first proposed by Gentry in [1] using ideal lattices - this finally enabled additive as well as multiplicatively homomorphic operations on encrypted data.

Since then, a number of schemes have cropped up which promise more efficient computations in such a way that the resultant noise stays below a threshold (via bootstrapping or modulus switching, for instance) and the resultant ciphertext size does not grow too large (via relinearization). Most of these schemes are based on the hardness assumption of the Ring-LWE problem.

These include:

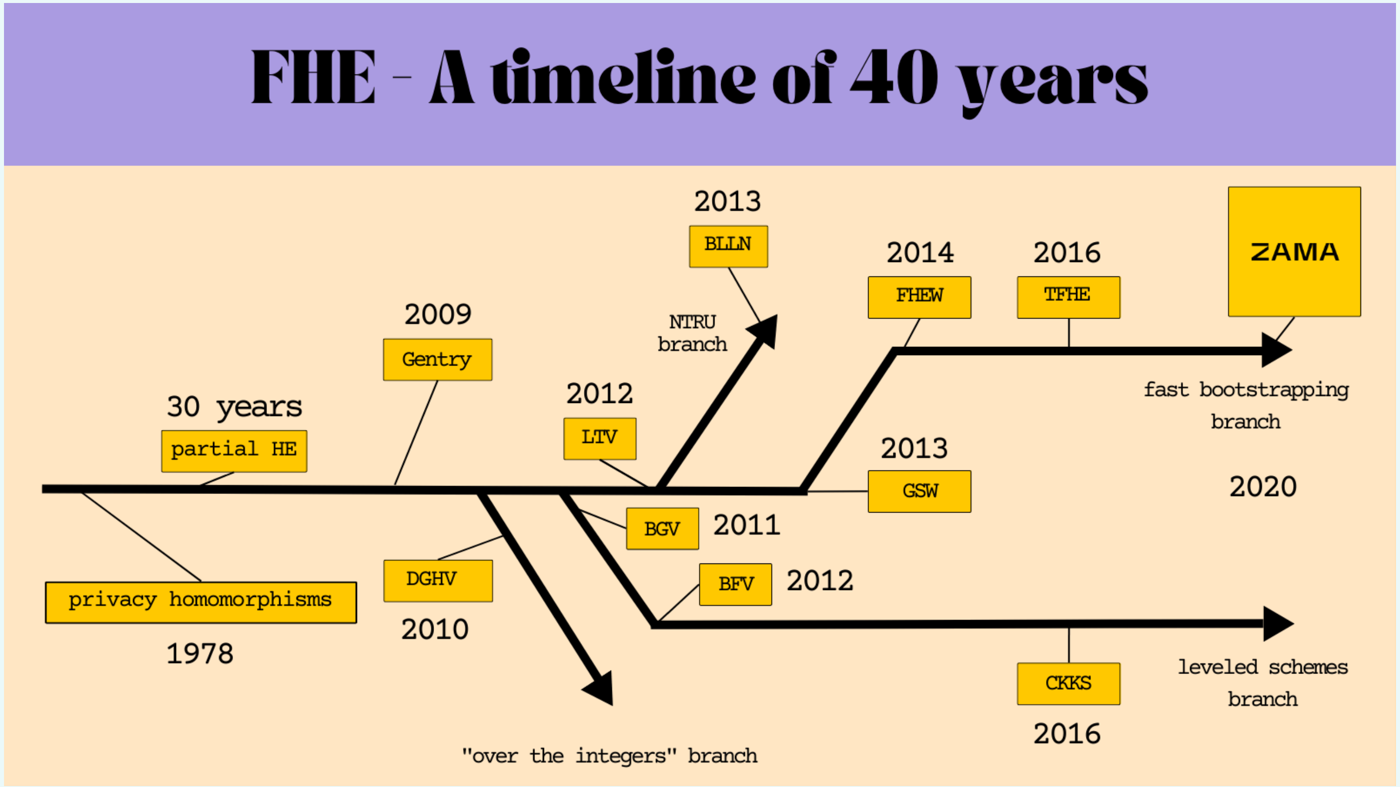
- Brakerski-Fan/Vercauteren (BFV) [2]
- Brakerski-Gentry-Vaikuntanathan (BGV) [3]
- Cheon-Kim-Kim-Song (CKKS) [4]
- Gentry-Sahai-Waters (GSW) [5]
- Homomorphic Encryption over the Torus (TFHE) [6]
- FHEW due to Ducas and Micciancio [7]

A Brief History of ~~Time~~ FHE



Credits: https://tfhe.com

A Brief History of ~~Time~~ FHE



Credits: https://tfhe.com

Jun 04, '24: First proposal of QFHE construction from generic FHE schemes by [8]

Introduction to OpenFHE

OpenFHE is a comprehensive library for employing FHE in code that provides APIs in C/C++, Python and Rust, supporting schemes like BGV, CKKS, and BFV. Here, we demonstrate usage using the Python bindings for OpenFHE's BFV scheme implementation.

Introduction to OpenFHE

OpenFHE is a comprehensive library for employing FHE in code that provides APIs in C/C++, Python and Rust, supporting schemes like BGV, CKKS, and BFV. Here, we demonstrate usage using the Python bindings for OpenFHE's BFV scheme implementation.

Documentation:

- `https://github.com/openfheorg/openfhe-python`
- `https://openfheorg.github.io/openfhe-python/html/index.html`

Introduction to OpenFHE

OpenFHE is a comprehensive library for employing FHE in code that provides APIs in C/C++, Python and Rust, supporting schemes like BGV, CKKS, and BFV. Here, we demonstrate usage using the Python bindings for OpenFHE's BFV scheme implementation.

Documentation:

- `https://github.com/openfheorg/openfhe-python`
- `https://openfheorg.github.io/openfhe-python/html/index.html`

Some common methods:

- `MakePackedPlaintext(pt) -> openfhe.Plaintext` - Encode the plaintext vector
- `Eval{Add, Sub, Mult}(ct1, ct2) -> ct3` - Perform homomorphic add/sub/mult
- `EvalMultKeyGen(sk)` - Generate the relinearization key used for `EvalMult`
- `EvalSum(ct, batchSize) -> ct1` - Evaluate the sum of all components in a vector
- `GetPackedValue(openfhe.plaintext) -> List[int]` - Decode packed pt. to vector
- `EvalAtIndex(ct, index) -> ct1` - Rotate by index (+ve/-ve corr. to left/right shift)
- `EvalAtIndexKeyGen(sk)` - Generate the rotation keys used for `EvalAtIndex`

OpenFHE: Basic usage

This is a sample code template that demonstrates usage of the BFV-RNS (residue number system) scheme for performing FHE (`add` and `mult`) on two plaintext vectors:

```
from openfhe import *

# Set CryptoContext
parameters = CCParamsBFVRNS() # Create instance of the BFV-RNS scheme
parameters.SetPlaintextModulus(65537) # Define plaintext space
parameters.SetMultiplicativeDepth(4) # Max no. of mults w/o bootstrapping

crypto_context = GenCryptoContext(parameters)
crypto_context.Enable(PKESchemeFeature.PKE) # Allow public-key encryption
crypto_context.Enable(PKESchemeFeature.LEVELEDSHE) # Enable leveled FHE w/o
bootstrapping
crypto_context.Enable(PKESchemeFeature.KEYSWITCH) # Enable key switching /
relinearization

# Generate (pk, sk)
key_pair = crypto_context.KeyGen()
```

```python
# Generate the relinearization key
crypto_context.EvalMultKeyGen(key_pair.secretKey)

# Encode first plaintext vector
vec1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
pt1 = crypto_context.MakePackedPlaintext(vec1)

# Encode second plaintext vector
vec2 = [11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
pt2 = crypto_context.MakePackedPlaintext(vec2)

# Encrypt the two vectors using the same public key
ct1 = crypto_context.Encrypt(key_pair.publicKey, pt1)
ct2 = crypto_context.Encrypt(key_pair.publicKey, pt2)

# Homomorphic addition
ct_add = crypto_context.EvalAdd(ct1, ct2)

# Homomorphic multiplication
ct_mult = crypto_context.EvalMult(ct1, ct2)

# Decrypt the result of the addition
```

```python
pt_add = crypto_context.Decrypt(ct_add, key_pair.secretKey)

# Decrypt the result of the multiplication
pt_mult = crypto_context.Decrypt(ct_mult ,key_pair.secretKey)

print("Plaintext #1: " + str(pt1))
print("Plaintext #2: " + str(pt2))

# Output results
print("#1 + #2 = " + str(pt_add))
print("#1 * #2 = " + str(pt_mult))
```

Output:

```
Plaintext #1: ( 1 2 3 4 5 6 7 8 9 10 ... )
Plaintext #2: ( 11 12 13 14 15 16 17 18 19 20 ... )
#1 + #2 = ( 12 14 16 18 20 22 24 26 28 30 ... )
#1 * #2 = ( 11 24 39 56 75 96 119 144 171 200 ... )
```

Refer to the OpenFHE GitHub repository for more detailed examples that also demonstrate bootstrapping and Threshold-FHE, both of which are beyond the scope of this lab.

Exercises on FHE

You are encouraged to play around with the OpenFHE library! BFV and BGV are two of the simpler FHE schemes with wide-ranging applications. Here are some exercises you can try out:

Exercises on FHE

You are encouraged to play around with the OpenFHE library! BFV and BGV are two of the simpler FHE schemes with wide-ranging applications. Here are some exercises you can try out:

- Integer comparison:
  - Encrypt two arbitrary integers $a = 6, b = 8$
  - Compare them homomorphically
  - Decrypt and verify the result

Exercises on FHE

You are encouraged to play around with the OpenFHE library! BFV and BGV are two of the simpler FHE schemes with wide-ranging applications. Here are some exercises you can try out:

- Integer comparison:
  - Encrypt two arbitrary integers $a = 6, b = 8$
  - Compare them homomorphically
  - Decrypt and verify the result
- Arithmetic mean:
  - Encrypt a dataset $A = [412, 8423, 66, 891, 277, 84, 5, 9]$
  - Homomorphically compute the arithmetic mean of A
  - Decrypt and verify the result

Exercises on FHE

You are encouraged to play around with the OpenFHE library! BFV and BGV are two of the simpler FHE schemes with wide-ranging applications. Here are some exercises you can try out:

- Integer comparison:
  - Encrypt two arbitrary integers $a = 6, b = 8$
  - Compare them homomorphically
  - Decrypt and verify the result
- Arithmetic mean:
  - Encrypt a dataset $A = [412, 8423, 66, 891, 277, 84, 5, 9]$
  - Homomorphically compute the arithmetic mean of A
  - Decrypt and verify the result
- Polynomial evaluation:
  - Take a multivariate polynomial, say, $P(x, y) = 2x^2 + 3xy + 4y^2 + 5x + 6y + 7$
  - Evaluate the polynomial homomorphically on, say, $x = 3, y = 4$
  - Decrypt and verify the result $P(3, 4) = 164$

Exercises on FHE (contd.)

- Prove some basic algebraic identities:
  - $(a \pm b)^2 = a^2 \pm 2ab + b^2$
  - $(a^2 - b^2) = (a + b)(a - b)$
  - $(a \pm b)^3 = a^3 \pm 3a^2b + 3ab^3 + b^3$
  - $(a + b + c)^2 = a^2 + b^2 + c^2 + 2(ab + bc + ca)$

Exercises on FHE (contd.)

- Prove some basic algebraic identities:
  - $(a \pm b)^2 = a^2 \pm 2ab + b^2$
  - $(a^2 - b^2) = (a + b)(a - b)$
  - $(a \pm b)^3 = a^3 \pm 3a^2 b + 3ab^3 + b^3$
  - $(a + b + c)^2 = a^2 + b^2 + c^2 + 2(ab + bc + ca)$
- Matrix multiplication:
  - Encrypt two matrices $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$
  - Perform homomorphic matrix multiplication
  - Decrypt and verify the result matches the product $C = A \cdot B = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$

Exercises on FHE (contd.)

- Prove some basic algebraic identities:
  - $(a \pm b)^2 = a^2 \pm 2ab + b^2$
  - $(a^2 - b^2) = (a + b)(a - b)$
  - $(a \pm b)^3 = a^3 \pm 3a^2b + 3ab^3 + b^3$
  - $(a + b + c)^2 = a^2 + b^2 + c^2 + 2(ab + bc + ca)$
- Matrix multiplication:
  - Encrypt two matrices $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$
  - Perform homomorphic matrix multiplication
  - Decrypt and verify the result matches the product $C = A \cdot B = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$
- Inner product:
  - Encrypt two vectors $v_1 = [1, 2, 3], v_2 = [4, 5, 6]$
  - Compute the homomorphic inner product $p = \langle v_1, v_2 \rangle$
  - Decrypt and verify the result matches $p = 32$

Exercises on FHE (contd.)

- Prove some basic algebraic identities:
  - $(a \pm b)^2 = a^2 \pm 2ab + b^2$
  - $(a^2 - b^2) = (a + b)(a - b)$
  - $(a \pm b)^3 = a^3 \pm 3a^2b + 3ab^3 + b^3$
  - $(a + b + c)^2 = a^2 + b^2 + c^2 + 2(ab + bc + ca)$
- Matrix multiplication:
  - Encrypt two matrices $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$
  - Perform homomorphic matrix multiplication
  - Decrypt and verify the result matches the product $C = A \cdot B = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$
- Inner product:
  - Encrypt two vectors $v_1 = [1, 2, 3], v_2 = [4, 5, 6]$
  - Compute the homomorphic inner product $p = \langle v_1, v_2 \rangle$
  - Decrypt and verify the result matches $p = 32$
- Determinant:
  - Encrypt a matrix $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$
  - Compute the determinant $\det(A) = ad - bc$
  - Decrypt and verify the results match $\det(A) = -2$

Bibliography

[1] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 2009, pp. 169–178.

[2] J. Fan and F. Vercauteren, "Somewhat Practical Fully Homomorphic Encryption." [Online]. Available: https://eprint.iacr.org/2012/144

[3] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "Fully Homomorphic Encryption without Bootstrapping." [Online]. Available: https://eprint.iacr.org/2011/277

[4] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic Encryption for Arithmetic of Approximate Numbers." [Online]. Available: https://eprint.iacr.org/2016/421

[5] C. Gentry, A. Sahai, and B. Waters, "Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based." [Online]. Available: https://eprint.iacr.org/2013/340

[6] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast Fully Homomorphic Encryption over the Torus." [Online]. Available: https://eprint.iacr.org/2018/421

[7] L. Ducas and D. Micciancio, "FHEW: Bootstrapping Homomorphic Encryption in less than a second." [Online]. Available: https://eprint.iacr.org/2014/816

[8] A. Gupte and V. Vaikuntanathan, "How to Construct Quantum FHE, Generically." [Online]. Available: https://eprint.iacr.org/2024/893