Privacy Preserving Machine Learning

By Matthew Townsend

NCS Capstone Presentation

SUNY Polytechnic Institute

Index

- The Need
- Approaches
- Homomorphic Encryption
- Small Scale implementation

The need

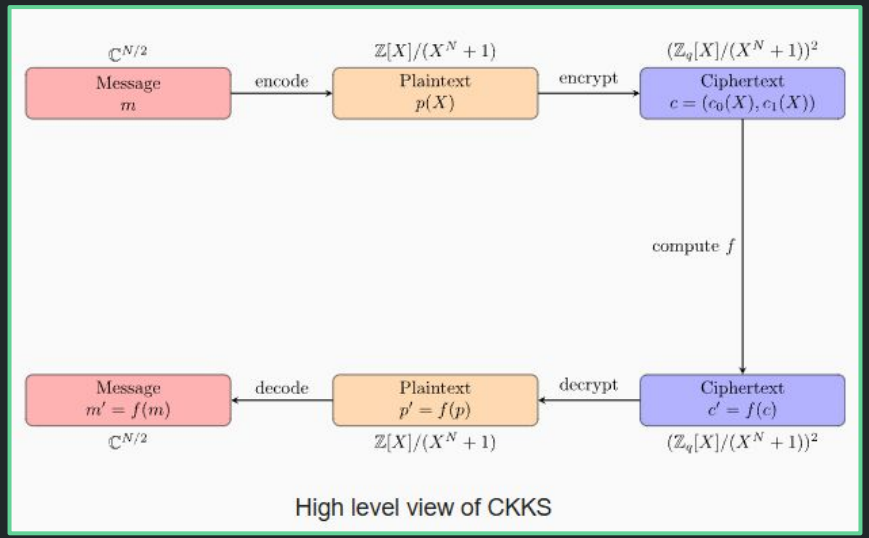
- While Machine Learning has advanced many fields such as data analytics and Medical computation, it has a central weakness which is the data it relies on,
- A model is only as good as the data it is given.
- This data can be your user information from social media, finance app, etc.
- As with all software there are methods to learn this data.

Approaches

- Federated Learning
  - Decentralized training on edge devices, data never leaves device.
  - User Downloads model, trains, encrypt, sends to the ‘cloud’, finally decrypt.
- Anonymization
  - Remove direct identifies, K-anonymity and l-diversity.
  - On a scale from most secure/bad data to least secure/good data
- Differential Privacy
  - Done by adding random noise through laplace, exponential, and randomised response mechanism.
- Lastly: Fully Homomorphic Encryption (FHE)

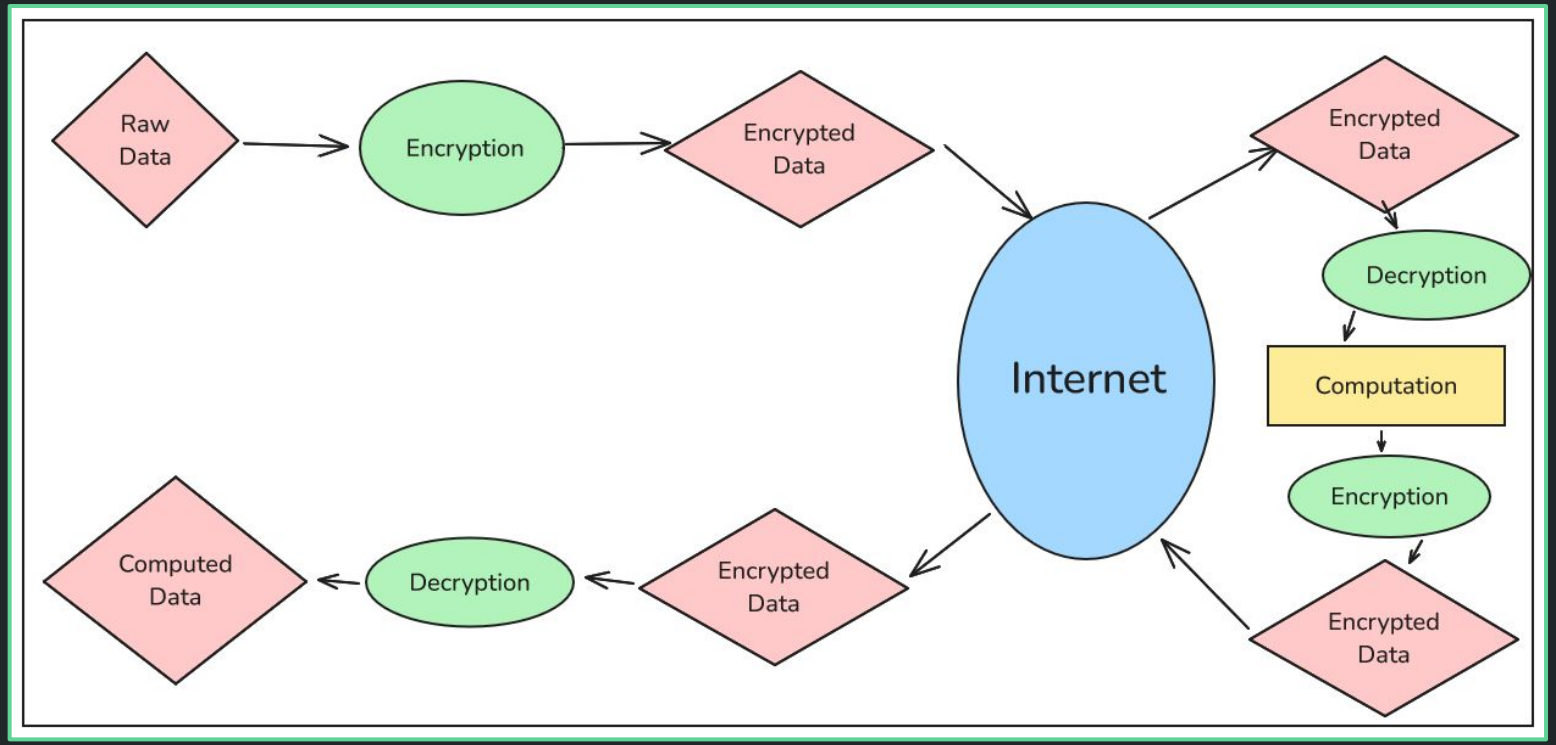
Homomorphic Encryption

- Version of encryption that allows for mathematical operations while in an encrypted state.
- The scheme used is CKKS as it allows for Arithmancy of Approximate Numbers

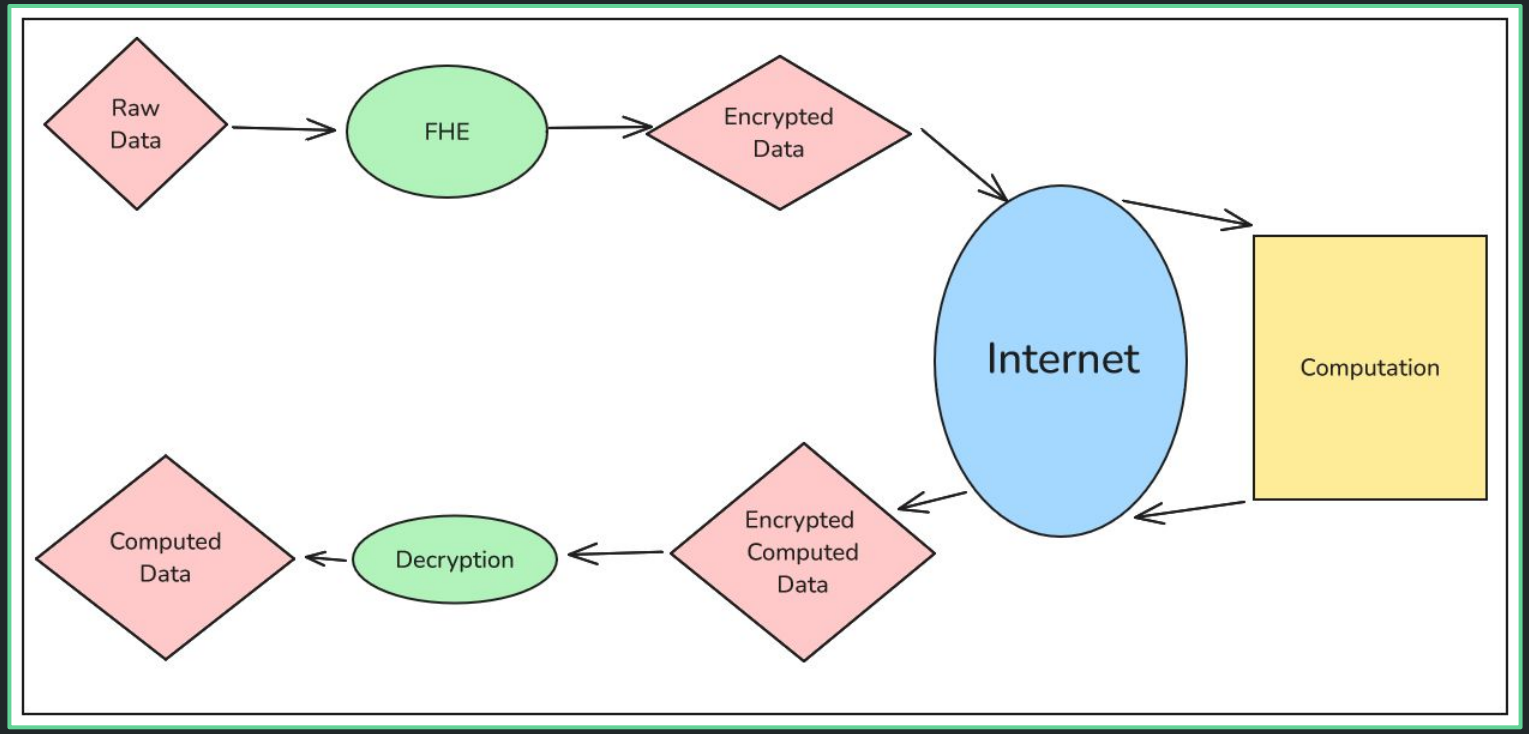


https://blog.openmined.org/ckks-explained-part-1-simple-encoding-and-decoding/

Overview - Traditional Encryption



Overview FHE



CKKS / HEEN

- As described, CKKS also called, Homomorphic Encryption for Arithmetic of Approximate Numbers (HEAAN)
- Here plaintext is taken as the payload and loss combined.
- RESCALE: reduce the magnitude of plaintext
- MODSWITCH: Switching modulus to add resizing functionality

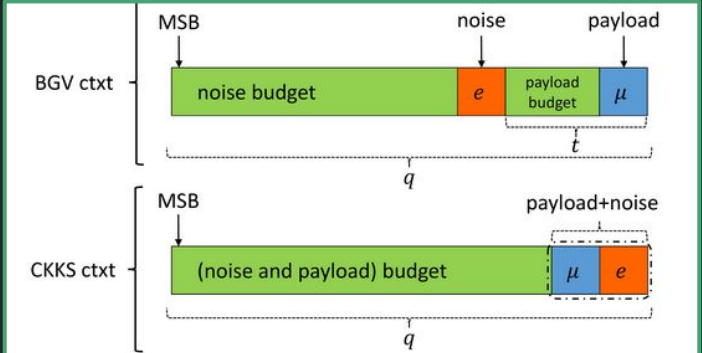


Figure 1: Ciphertext structure in the BGV and CKKS schemes. MSB stands for most significant bit. Adapted from [CKKS17].

https://inferati.com/blog/fhe-schemes-ckks#sec-bgv-struct

Overview

- In order to get a working model running encrypted data on training required complicated model defining.
- I used torch to build my model but for the encrypted model, the standard functions did not work.
- The accuracy is point to note for this model as the variance in noise affects it. But it has stayed within 0.3 of the Non-Encrypted Accuracy.

Encrypted Model

- The Encrypted LR model was not compatible with the several functions due to its limited operations and data structure.
- Sigmoid:

```
@staticmethod
def sigmoid(enc_x):
    # sigmoid = 0.5 + 0.197 * x - 0.004 * x^3
    # this is a degree 3 polynomial approximation of the sigmoid function
    # it's used to keep the output of the linear layer in the range of the sigmoid approximation
    return enc_x.polyval([0.5, 0.197, 0, -0.004])
```

- Forward Pass:

```
#Forward pass
def forward(self, enc_x):
    enc_out = enc_x.dot(self.weight) + self.bias
    #Calculates linear combination of input and weight, adds bias
    enc_out = EncryptedLR.sigmoid(enc_out)
    #Applies sigmoid function
    return enc_out
```

Encryption

- CKKS, implemented through TenSEAL, allows for more complex mathematical operations
- Define context, polynomials, coefficients, global scale
- Create keys

```
# the degree of the polynomial modulus
poly_mod_degree = 8192
# the bit-length of the modulus chain
coeff_mod_bit_sizes = [40, 21, 21, 21, 21, 21, 21, 40]
# create TenSEALContext
enc_training = ts.context(ts.SCHEME_TYPE.CKKS, poly_mod_degree, -1, coeff_mod_bit_sizes)
# generate keys
enc_training.global_scale = 2 ** 21
enc_training.generate_galois_keys()
```

Encryption 2

- After defining the TenSEAL context, then define the ckks_vector with the context and the x,y data in x_train and y_train

```
t_start = time()
enc_x_train = [ts.ckks_vector(enc_training, x.tolist()) for x in x_train]
enc_y_train = [ts.ckks_vector(enc_training, y.tolist()) for y in y_train]
t_end = time()
print(f"Encryption of the training_set took {int(t_end - t_start)} seconds")
```

```
Encryption of the training_set took 18 seconds
```

Results

Non-Encrypted

```
Loss at epoch 1: 0.8512
Loss at epoch 2: 0.6511
Loss at epoch 3: 0.5554
Loss at epoch 4: 0.5111
Loss at epoch 5: 0.4860

############# Non-Encrypted Training #############
Average time per epoch: 0 seconds
Non-Encrypted Accuracy: 0.7277
Memory usage: 183.01 MB
################################################
```

Encrypted

```
Accuracy at epoch #0 is 0.3125
Accuracy at epoch #1 is 0.7277
Accuracy at epoch #2 is 0.7277
Accuracy at epoch #3 is 0.7366
Accuracy at epoch #4 is 0.7321
Memory usage: 2073.83 MB
Accuracy at epoch #5 is 0.7366
############# Encrypted Training #############

Average time per epoch: 58 seconds
Accuracy 0.7366
Difference between plain and encrypted accuracies: -0.0089
Memory usage: 2074.06 MB
################################################
```

Challenges - Everything

- The TenSEAL library is a python wrapper for SEAL, a now EOL Microsoft Library.
- Sklearn in its standard form, does not operate with ‘ckks_vector’, the best I was able to get was simulated encrypted training.
- Coefficients and Polynomial modulus

Conclusion

- Receives similar accuracy but at the time penalty of 60 times in this scenario and increased resource utilization.
- Calculations by IBM place the penalties at 50-1 compute cost and 20-1 memory cost.
- This is a relatively new and advanced method of encryption, with most of its development being done in the past 15 years.
- I plan to continue studying FHE as I begin my masters degree.

Sources

https://arstechnica.com/gadgets/2020/07/ibm-completes-successful-field-trials-on-fully-homomorphic-encryption/

https://arxiv.org/html/2408.15231v1

https://www.ftc.gov/system/files/ftc_gov/pdf/Social-Media-6b-Report-9-11-2024.pdf

https://digitalprivacy.ieee.org/publications/topics/homomorphic-encryption-use-cases
https://inferati.com/blog/fhe-schemes-ckks#sec-bgv-struct
@article{Acar2018,title = {A survey on homomorphic encryption schemes: Theory and implementation},journal = {ACM Computing Surveys},year = {2018},volume = {51},number = {4},author = {Acar, A. and Aksu, H. and Uluagac, A.S. and Conti, M.}}