Name: Anushka Yadav
E-mail: aryadav_b24@it.vjti.ac.in

Energy-Based and Neuro-Symbolic Methods for Advanced Cryptanalysis

Introduction:

Myself, Anushka Yadav, Second Year B.Tech Information Technology Student, Veermata Jijabai Technological Institute (VJTI)
I have a keen interest in learning about AI (Artificial Intelligence) and encryption. I enjoy working with programming languages like Python and C++, and I love exploring ways that technology can help solve real-world problems. I am always eager to learn more and apply my skills in interesting ways.

DESCRIPTION OF THE PROJECT:

Energy-Based and Neuro-Symbolic Methods for Advanced explores cutting-edge AI and neuro-symbolic approaches to analyze and break cryptographic systems. It aims to combine energy-based models, quantum-inspired learning mechanisms, and symbolic reasoning to develop advanced cryptanalysis techniques. The ultimate goal is to evaluate and challenge the strength of modern encryption systems, such as AES, and improve our understanding of cipher structures through learning and attack simulation.

Motivation of the Project:

This project aims to explore how AI can help in understanding and breaking encryption. Even though I was new to concepts like Transformers and symbolic learning, I was interested in learning how these advanced methods work together. By combining machine learning and logic-based techniques, this project gives me a chance to apply AI in a challenging field like cryptography and learn new things along the way.

GOALS:

1. Cipher Cracking:

Rationale:

Breaking existing ciphers can help us find weaknesses in old or popular encryption methods. This knowledge allows us to improve security and create stronger encryption.

Approach:

We will use Hopfield Neural Networks (HNNs) and Transformer models to identify the encryption method used on a ciphertext

- HNNs work like memory systems that compare unknown ciphertexts to stored cipher patterns, which helps identify the type of cipher.
- Transformers can learn patterns by paying attention to different parts of the text, allowing them to find repeating structures that hint at specific cipher types.

Outcome:

Develop techniques capable of identifying and decrypting ciphertexts without prior key knowledge. By combining pattern memory (HNN) and sequence learning (Transformer), we improve the accuracy and adaptability of cipher detection.

2. Key Recovery Attacks:

Objective:

To evaluate the effectiveness of advanced key recovery methods, we will focus on AES(Advanced Encryption Standard), which is known for its high resilience. Through testing AES, we can conclude how effective, efficient, and practical these key recovery methods are.

Approach:

We propose two key recovery strategies:

- **Simulated Quantum Annealing (SQA):** Models that search for the correct key as an optimization problem by minimizing decryption error in a large keyspace.

- **Side-Channel AI Attacks**: Uses AI models trained on external signals to learn and predict the secret key used during encryption

Importance:

Crucial for evaluating the robustness of modern cryptographic protocols.

3. Symbolic Cipher Learning: (Reverse Engineering)

Purpose:

Learn the logic behind symbolic ciphers and decode them without prior knowledge of the cipher

Approach:

We use a Transformer-based model trained on ciphertext–plaintext pairs to directly reverse-engineer the cipher transformation.

- This model does not require prior key information and instead learns the mapping through symbolic sequence patterns.

- In future work, neuro-symbolic techniques can enhance the model by combining learned data with logical rules (like XOR behavior).

Outcome:

Develop models capable of decrypting ciphertext directly, simulating how an attacker might reverse-engineer an unknown encryption system.

Methodology:

➢ Execution Overview:

We start by encrypting messages using the AES algorithm and randomly generated 128-bit keys. For each message, we store the encrypted output (ciphertext), the key, and the original text. This dataset is used to train a Transformer model that takes ciphertext as input and predicts the original message, helping with reverse engineering.

To identify which cipher may have been used, we study Hopfield Networks that store and recall known encryption patterns. For key recovery, we will use two techniques: simulated quantum annealing (to search keyspace) and side-channel AI attacks (to predict keys from leaked data). We also use neuro-symbolic learning to help the model understand the logic behind cipher operations like XOR.

Finally, we evaluate each method's performance using test samples to measure accuracy and effectiveness in all three tasks — cipher cracking, key recovery, and reverse engineering.

1. Cipher Generation:

We will use the Advanced Encryption Standard (AES) algorithm to generate ciphertexts from plaintext messages. AES is a symmetric block cipher that encrypts data in fixed-size blocks of 128 bits (16 bytes). For each encryption, we generate a new random 128-bit key to simulate a realistic cryptographic setting where keys are unknown to attackers.

We use the PyCryptodome library in Python to perform the encryption. PyCryptodome allows us to generate secure random keys and apply the AES algorithm in different modes. For simplicity and ease, we will use the Electronic Codebook (ECB) mode of AES. ECB mode encrypts each block of plaintext independently, making it easier to trace how the input is transformed into ciphertext, although it is less secure than other modes like CBC or CTR.

To create each data sample:

- We first select a plaintext message.
- We pad the message, meaning we will make its length a multiple of 16 bytes, as required by AES.
- We generate a random 128-bit key using the get_random_bytes(16) function.
- We encrypt the padded plaintext using AES in ECB mode.
- We convert the key and ciphertext into Base64-encoded strings to make them easier to read, store, and feed into ML models.

Each sample includes three parts:

- plaintext
- ciphertext,
- encryption key.

These are saved together in a JSON file and form the foundation of our dataset. We generate hundreds or thousands of such samples to provide enough data for training and testing AI models.

2. Cipher Cracking (Goal 1):

Cipher cracking is the task of identifying which cipher was used to encrypt the message.

Techniques used:

- **Hopfield Neural Networks:**

Hopfield Neural Network enhances the ability of our model to remember and recall patterns found in the encrypted data.

A Hopfield network is a type of Recurrent Neural Network designed for associative memory. This means it can store certain patterns and retrieve them even if the input is incomplete. In our project, this ability is useful because encrypted data often contains recurring structures or patterns that can be remembered and used to assist in decryption.

We aim to use Hopfield Networks to:

- Store known ciphertext–plaintext pairs, acting like memory units.
- Recall similar patterns when a new ciphertext appears that partially matches a stored one.
- Support the transformer model, especially when the model struggles to decode certain inputs on its own.

3. Key Recovery Attack (Goal 2):

Key recovery involves searching the encryption key space to find the correct key used in AES.

Techniques used:

- **Simulated Quantum Annealing:**

To improve our method, we can use Simulated Quantum Annealing (SQA) to enhance decryption and key recovery.

Simulated Quantum Annealing is a technique that looks for the lowest "energy" state in a system. Here, "energy" refers to the difference or error between the decrypted result and the expected plaintext.

Implementation:

I. Define an energy function that evaluates how close a decrypted output is to the true plaintext.
II. Let the algorithm search through possible outputs to find the one with the minimum error.

- **Side-Channel AI Attacks:**

Side-channel attacks use physical information leaked by devices during encryption, such as power use or timing, to guess the secret key. Even if the encryption algorithm is strong, these leaks can reveal key details. In this project, we use AI models to learn from such leakage data.

Approach:

- We will simulate simple side-channel data, like how many bits change during certain encryption steps.

- Then, we will train a neural network on this data so it can learn how to predict parts of the secret key.

4. Symbolic Cipher Learning: (Goal 3)

Techniques used:

Al model: Transformer:

The Transformer is a type of neural network. Unlike RNN(Recurrent Neural Network), Transformers process the entire input sequence in parallel. They rely solely on a mechanism called Attention, which allows the model to focus on the most important parts of the input.

The Transformer's main strength is its self-attention mechanism, which allows the model to focus on different parts of the input sequence when making predictions. In our case, the model learns which parts of the ciphertext are most helpful for predicting each part of the plaintext.

In our project, we will be using a Transformer to learn how to convert AES-encrypted ciphertext into its original plaintext message.

Model Architecture:

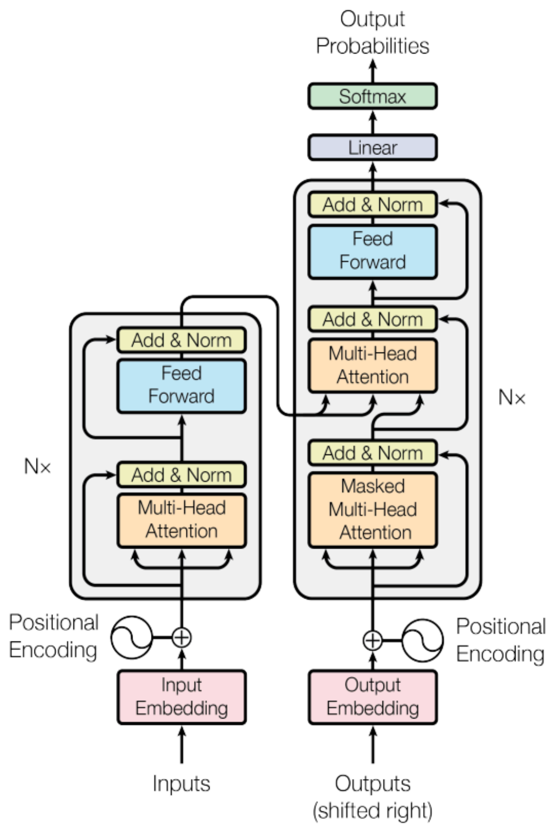


Figure 1: The Transformer - model architecture.

Components:

1. Input Embedding:

Machine learning models work with numbers, not text. So it converts each token, or in simple words, character in the ciphertext, into a high-dimensional numerical vector. Each ciphertext character is embedded into a vector that holds its features. Embeddings capture relationships between tokens.

2. Positional Encoding

Transformers don’t process data in order by default, so this tells the model the correct sequence. It adds information about the position of each token in the sequence. Thus, the model understands the order of characters in the ciphertext, which is important for decryption.

3. Encoder Blocks:

a. Multi-Head Self-Attention:

Helps the model learn patterns by checking how each token relates to others, and Learns relationships between encrypted characters to decode the message.

b. Feed-Forward Neural Network:

Applies transformations to the attention output and helps the model make complex decisions about the input pattern.

4. Linear + Softmax Layer:

Converts processed vectors into probabilities for each possible output token. Then the model predicts the most likely next character and outputs the predicted plaintext character at each position.

5. Neuro-Symbolic Learning:

In our project, we will combine two ideas: Neural Network and symbolic reasoning. This is known as Neuro-symbolic learning.

Neural => Learning from data (like ciphertext–plaintext pairs)

Symbolic => Understanding logic rules (like XOR, AND, or modular arithmetic)

Neuro-Symbolic => Combining both to get better learning and reasoning.

We can use symbolic expressions alongside the neural outputs. These symbolic parts act as a guide to correct or support what the Transformer model predicts.

6. Model Training and Evaluation:

Now, after defining our transformer model, the next step is to train the model using our dataset. The goal is to enable the model to learn how to map ciphertext back to its original text.

Each training sample contains:

- Input: Ciphertext
- Target: Corresponding original text

The model learns by comparing its predicted plaintext against the actual plaintext and gradually improving through repeated training.

★ Evaluation Strategy:

After training, the model will be tested on unseen ciphertext-plaintext pairs. To evaluate performance, we will measure:

- Character Accuracy: The percentage of individual characters correctly predicted.
- Exact Match Score: The number of complete plaintexts correctly predicted without any error.

  These metrics evaluate how well the model can understand and perform the decryption task using new examples, not just ones it has memorized.

Workflow:

Week 1:

Set up the environment and generate AES-encrypted datasets (plaintext, ciphertext, key).

Week 2:

Process data and prepare it for model input (tokenization, padding, encoding).

Week 3:

Build and train the Transformer model for reverse engineering (ciphertext => plaintext)

Week 4:

Evaluate the Transformer performance and adjust the model.

Week 5:

Integrate a Hopfield Neural Network for cipher pattern identification (cipher cracking).

Week 6:

Simulate side-channel leakage data and train a neural network for key recovery

Week 7:

Implement simulated quantum annealing to optimize model predictions.

Week 8:

Final testing, evaluation, and result analysis.

Implementation:

As part of the initial implementation, the following is Python code using PyCryptodome to encrypt and decrypt AES messages. This includes generating a random 128-bit key, padding the plaintext, encrypting in ECB mode, and decoding the ciphertext. This code is used to build our dataset of ciphertext, plaintext, and key samples.

```
from Crypto.Cipher import AES

from Crypto.Util.Padding import pad, unpad

from Crypto.Random import get_random_bytes

import base64
```

This will create a random key of 16 bytes:

```
key = get_random_bytes(16)
```

This will create an AES cipher in ECB mode:

```
cipher = AES.new(key, AES.MODE_ECB)
```

AES only works with byte data, not normal strings. So we write b '......' to make it a bytes object:

```
data = b'Project-X'
```

AES needs the message to be a multiple of 16 bytes. For AES, block size is fixed- 16 bytes

```
padded_data = pad(data, AES.block_size)
```

Now, we encrypt the padded message using the cipher.

```
ciphertext = cipher.encrypt(padded_data)
```

Convert the ciphertext to Base64 format. So, it becomes a readable string.

```
encoded_ciphertext = base64.b64encode(ciphertext).decode()
```

Decryption using the same key

```
cipher_dec = AES.new(key, AES.MODE_ECB)
```

```python
decrypted_padded_data = cipher_dec.decrypt(ciphertext)
```

Remove padding

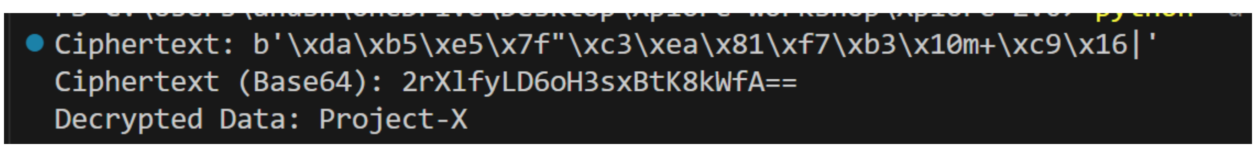
```python
decrypted_data = unpad(decrypted_padded_data, AES.block_size)
```

```python
print(f"Ciphertext: {ciphertext}")

print(f"Ciphertext (Base64): {encoded_ciphertext}")

print(f"Decrypted Data: {decrypted_data.decode()}")
```

Output :



```
Ciphertext: b'\xda\xb5\xe5\x7f"\xc3\xea\x81\xf7\xb3\x10m+\xc9\x16|'
Ciphertext (Base64): 2rXlfyLD6oH3sxBtK8kWfA==
Decrypted Data: Project-X
```

References:

1) pycryptodome:

   https://www.geeksforgeeks.org/python/what-is-pycryptodome-in-python/

2) Hopfield Neural Network:

   https://arxiv.org/pdf/2008.02217

3) Neuro-Symbolic Learning:

   https://arxiv.org/abs/2110.01639/

4) Transformers paper:

   https://arxiv.org/abs/1706.03762

5) Simulated Quantum Annealing Article –
   https://www.nature.com/articles/ncomms3067

6) Side-Channel AI Techniques –

   https://eprint.iacr.org/2018/053