Project Title: Energy-Based and Neuro-Symbolic Methods for Advanced Cryptanalysis

---

1. Project Description

Modern cryptography is built on assumptions that certain problems are too hard to solve quickly. But with AI improving fast—especially in areas like energy-based models, symbolic reasoning, and quantum-inspired search—we now have new tools that might be able to challenge these assumptions.

In this project, I plan to build a full system that can: - Try to recover AES-128 keys from known data, - Break small or toy ciphers fully, - And learn simple, human-readable logic behind unknown encryption methods.

This project will not only test how AI can be used in cryptanalysis, but will also result in a working open- source toolkit that could help others test and understand ciphers.

2. Abstract

This project is about building a tool that uses AI to study and break encryption systems. It combines energy-based models (EBMs), transformer models, quantum-inspired search methods, and logic solvers.

The tool will: - Automatically create input–output examples using AES and toy ciphers, - Model the encryption process using an energy function, - Use simulated quantum annealing to search for likely keys, - And apply logical rules to keep things interpretable.

I'll test it on AES-128 (for key recovery) and smaller ciphers (for full cracking). The system will be judged based on how well it narrows down the key space, how accurate the recovered keys are, and whether the learned rules match how the cipher really works.

Final outputs will include working code, saved models, a small web demo, and a technical report.

3. About Me & Motivation

Myself, Rishit Modi, a 2nd year B.Tech Computer Engineering student at Veermata Jijabai Technological Institute. I am currently working on a project titled Energy-Based and Neuro-Symbolic Methods for Advanced Cryptanalysis, which explores the use of AI techniques for analyzing and breaking cryptographic systems.

Through this project, I am learning about energy-based models, symbolic reasoning, and quantum-inspired approaches to build advanced cryptanalysis tools. I am developing my skills in Python and exploring libraries and frameworks relevant to AI and security research

Methodology

3.1 Overall Plan

The system will have multiple parts working together: - First, I'll generate lots of input-output data using AES and toy ciphers. - Then, I'll use machine learning to train a Hopfield network that assigns lower energy to correct keys. - Next, I'll use a quantum-inspired algorithm to search through the key space. - I'll also train a transformer model to suggest logic rules about the cipher. - These logic rules will be checked for correctness and then used to guide the search. - At the end, all components will be joined in a pipeline to try and recover encryption keys.

3.2 Phase 1: Setup and Classical Attacks (Weeks 1–2)

Goal: Prepare the tools, generate datasets, and perform some standard attacks as a baseline.

- Set up Python libraries like PyCryptodome (for AES), PyTorch (for ML), and Z3 (for logic).
- Create a script that makes 50,000 (plaintext, ciphertext, key) samples using AES-128.
- Build a basic toy cipher (e.g., small SPN) that is easy to break manually.
- Perform differential cryptanalysis on the toy cipher. This gives us a way to compare how well our AI methods do later.

By the end of this phase: - I'll have working datasets and a reliable setup for experiments. - I'll know how traditional attacks perform. - This gives me a clear baseline to beat with AI models.

3.3 Phase 2: Hopfield-Based Energy Model (Weeks 3–4)

Goal: Train a Hopfield network to recognize correct keys based on their energy values.

- Represent each 128-bit AES key as a binary vector.
- Train the network so that it gives lower energy to real keys and higher energy to wrong ones.
- Use a contrastive loss (good vs bad key examples) to teach the network to separate them.

Why Hopfield: - It acts like memory: it "remembers" patterns (keys) during training. - It's fast to run and easy to integrate into later parts of the project.

Expected outcome: - By the end of this phase, the network should narrow down the possible keys to a much smaller group (~2^20).

3.4 Phase 3: Quantum-Inspired Search (Weeks 5–6)

Goal: Use simulated quantum annealing (SQA) to explore the keyspace more effectively.

- Convert the trained Hopfield model into a format called QUBO (used in optimization).
- Use the D-Wave Ocean tools to apply SQA and sample thousands of candidate keys.
- Feed these candidates back to the Hopfield network to double-check their energy levels.

Outcome: - We'll get a list of promising key guesses that can be passed to later logic stages.

3.5 Phase 4: Neuro-Symbolic Rules (Weeks 6–8)

Goal: Add logical reasoning to make the system more explainable.

- Fine-tune a small transformer (like DistilGPT2) on sequences from encryption traces.
- Let it predict logical rules like "if this bit changes, then that key bit is likely X."
- Use Z3 to verify these rules against real data.
- Feed verified rules back into the energy model to guide future searches.

Outcome: - I'll have working logic filters that improve accuracy and help explain the results.

3.6 Phase 5: AES Key Recovery Pipeline (Weeks 8–9)

Goal: Combine everything from earlier phases to recover AES keys from data.

- Start with 1,000 (plaintext, ciphertext) examples.
- Use differential methods to get initial guesses.
- Rank those guesses using the Hopfield model.
- Apply SQA again to explore new key candidates.
- Use logic rules to filter out invalid guesses.
- If the space is small enough (~2^20), finish with brute-force.

Outcome: - The pipeline will recover AES-128 keys in most test cases. - I'll compare its performance with earlier baselines.

3.7 Phase 6: Learning Logic from Unknown Ciphers (Weeks 9–10)

Goal: Try to reverse-engineer the structure of a cipher just from examples.

- Generate many input–output pairs from a black-box cipher.
- Use a transformer to find patterns and describe the cipher in rule form.
- Check how well the rules explain the outputs.
- Improve them in steps until the whole cipher is described clearly.
- If accuracy is good enough, convert rules into logic gates using Pyverilog.

Outcome: - A logic-level model of the cipher. - Shows the AI can figure out unknown designs, not just break known ones.

3.8 Phase 7: Using Side-Channel Data (Weeks 10–11)

Goal: Use extra information like power consumption to improve recovery.

- Use the ASCAD dataset, which has power traces from real AES runs.
- Compress the traces using PCA or an autoencoder.
- Attach these features to key guesses and check which ones match the hardware leakage.

Outcome: - Better accuracy in real-world settings. - A more complete model that works with physical as well as digital data

Phase 8: Evaluation

I'll test the system using: - Key rank (where the real key appears on the list) - Success rate (% of times the full key is recovered) - Logic accuracy (how close the learned rules are to the real ones) - Time and compute costs

3.9 Phase 9: Final Demo and Packaging (Week 11)

- Create a Streamlit interface to run tests and show results visually.
- Add Docker support so the project can be run easily on any machine.
- Record a short screencast demo and write a full project report.

All files, code, and setup steps will be added to a GitHub repo for easy access.

4. Timeline (12 Weeks)

Week 1 – Setting Up Tools and Reading Background Material

- Install the basic tools and Python libraries (like PyTorch for AI, PyCryptodome for encryption, Z3 for logic solving).
- Read introductory papers and watch videos to understand AES encryption, Hopfield networks, and symbolic AI.
- Create folders and files for each part of the project.
- Understand how AES works and explore example datasets like ASCAD (contains power traces from real AES runs).

Week 2 – Creating Data and Testing Old-School Attacks

- Write a Python script that generates thousands of encrypted messages using AES-128.
- Build a small, simplified encryption system (called a "toy cipher") for testing.
- Try basic cryptographic attacks (like differential cryptanalysis) on the toy cipher to see how traditional methods work.
- Save the results — these will be used later to see if our AI-based methods are better.

Week 3 – Starting Hopfield Network Training

- Represent encryption keys (like long passwords) as binary numbers.
- Begin training a special AI model called a Hopfield network, which learns to "remember" the right key by giving it a low energy value.
- Use pairs of "correct" and "wrong" keys to help the network learn the difference.

Week 4 – Testing the Hopfield Network

- See how well the Hopfield network can identify correct keys from a large list.
- Adjust the training settings to improve accuracy.
- Make sure the model reduces the number of possible keys from billions to something manageable.
- Save this model for the next step.

Week 5 – Simulated Quantum Search (Part 1)

- Convert the AI model into a format that can be used in optimization algorithms.
- Use a tool that mimics quantum computers (called Simulated Quantum Annealing) to search for possible keys more efficiently.
- Try running it on test data and record results.

Week 6 – Quantum Search (Part 2)

- Fine-tune the quantum search settings to improve results.
- Let the algorithm suggest thousands of likely key guesses.
- Recheck these guesses using the Hopfield network.
- Keep only the most promising keys.

Week 7 – Learning Logic Rules from Patterns

- Use a small AI model (like DistilGPT2) to learn simple rules from how bits change during encryption.
- For example: "If input bit 5 is 1, then output bit 12 is likely to be 0."

- Train the model using patterns found in encryption data.

Week 8 – Verifying and Using Logic Rules

- Use a logic solver (Z3) to check if the AI-generated rules actually make sense.
- Discard any rules that don't match real data.
- Use the verified rules to remove impossible key guesses from the list.
- This makes the final key search faster and more accurate.

Week 9 – Combining All Parts into a Working System

- Connect all the pieces: Hopfield network, quantum search, and logic rules.
- Use this full system to try recovering AES-128 keys from test examples.
- Compare its performance to the basic attacks done earlier (Week 2).
- Track how often it finds the correct key and how long it takes.

Week 10 – Trying to Understand a Cipher with No Prior Info

- Act like we've found a mysterious encryption method and want to figure out how it works just by looking at examples.
- Use AI to find patterns and write rules explaining how this unknown cipher might work.
- Convert these rules into logic gates (like in digital circuits) using tools like Pyverilog.
- This shows the system isn't just solving, but also understanding encryption logic.

Week 11 – Real-World Power Data and Demo Setup

- Use power traces (electrical signals) collected from real AES hardware to help in key recovery.
- Process the signals to find patterns that match different key guesses.
- Build a simple app using Streamlit to demonstrate how the system works.
- Add Docker support so it can run anywhere easily.

Week 12 – Wrapping Up and Final Touches

- Write a complete report explaining the project, results, and what was learned.
- Clean up the code, upload everything to GitHub.
- Record a short video showing how the system works.
- Use any extra time to fix bugs or improve results.

5. References

- https://arxiv.org/pdf/2008.02217.pdf

- https://github.com/ZhouKanglei/Cryptanalysis-of-MHM
- https://iacr.org/archive/ches2007/47270450/47270450.pdf
- https://arxiv.org/abs/1706.03762
- https://www.nature.com/articles/ncomms3067
- https://arxiv.org/abs/1710.03599
- https://youtu.be/piF6D6CQxUw
- https://youtu.be/zvfkXjzzYOo
- AES Explained (Advanced Encryption Standard) - Computerphile
- https://www.youtube.com/watch?v=EacYNe7moSs&pp=ygUJejMgc29sdmVy0gcJCcMJAYcqIYzv
- Transformers Explained Visually