Final Project Report – COMP 642

Breaking Classical Ciphers and RSA Using NLP and Machine Learning

Ishaan Iyer (ii10), Nneoma Amechi (na82)

May 2, 2025

1. Problem Description

Data is one of our most important and used assets in the modern age encryption ensures that only authorized parties can access or interpret sensitive data, forming the backbone of cybersecurity. If encryption can be easily broken, the entire system's protection and information is vulnerable and accessible to anyone with the right tools. Security is not just about hiding information, but about making it practically impossible to uncover without the key.

Classical ciphers, though no longer used for secure communication, remain relevant in Capture the Flag competitions, escape rooms, geocaching, and educational settings. Traditional decryption methods—brute force, frequency analysis, and pattern matching—are often inefficient or fail when the cipher type is unknown or the ciphertext is noisy.

This project reframes cipher-breaking as a language translation task using Transformer-based models like GPT 4O, with ongoing exploration into GPT-2. By leveraging the models' understanding of grammar and context, we aim to:

1. Identify the type of classical cipher.
2. Accurately decrypt it.
3. Handle noisy or imperfect ciphertexts.

We train these models using custom datasets of plaintext-ciphertext pairs from classical encryption schemes. Looking ahead, our long-term goal is to explore machine learning approaches to modern cryptography, particularly RSA. While breaking RSA is mathematically complex, we aim to investigate whether AI can support factorization or uncover structural patterns that aid in decryption—moving from language-based ciphers to algebraic cryptanalysis.

2. Literature Survey

2.1 Existing Work on Cipher Decryption

- Classical cryptanalysis techniques—such as frequency analysis, index of coincidence, and bigram/trigram statistics—have historically been effective for simple substitution ciphers like Caesar and Atbash. However, these approaches rely heavily on prior knowledge of the cipher type and clean, well-formatted input. They often fail in scenarios involving unknown cipher types or noisy ciphertexts.
- Quisquater & Delescaille (1989) explored statistical pattern recognition in monoalphabetic substitution ciphers using known letter frequency distributions. While effective for lengthy texts, the method struggles with short messages and obfuscation.
- Hill et al. (2004) introduced cipher-type identification using decision trees and n-gram analysis. Although this showed potential for automatic classification, the approach lacked adaptability and scalability when applied to a broader range of cipher structures or adversarial noise.

2.2 NLP & Transformer-Based Approaches

- Vaswani et al. (2017) proposed the Transformer architecture, which has since revolutionized sequence modeling tasks due to its attention mechanism and capacity for handling long-range dependencies—traits highly suitable for decryption tasks that depend on contextual understanding.
- Rijnders et al. (2022) applied supervised Transformers to solve substitution and transposition ciphers. However, their models were trained only on predefined cipher types and did not account for errors or noise often found in real-world cryptograms.
- Dugan et al. (2019) developed cipher classification models based on character frequency patterns using SVMs and random forests. While their method performed well on known cipher types, it lacked robustness under unpredictable or noisy conditions.

Our work builds upon these foundations by leveraging Transformer-based models (GPT 4O, GPT-2) to develop a noise-resilient, end-to-end system for both cipher classification and decryption. By formulating the task as a sequence-to-sequence translation problem, we enable the model to learn both structure and semantics, significantly outperforming rule-based techniques in noisy and variable environments.

Beyond classical cryptography, we also explore preliminary steps toward breaking modern encryption schemes like RSA. While breaking RSA directly remains infeasible through deep learning alone due to its mathematical hardness, we investigate whether AI can assist in tasks like prime factorization, structural vulnerability detection, or pattern-based attacks. This integration of NLP and algebraic cryptanalysis opens new avenues in both theoretical research and applied cryptography.

---

3. Hypothesis / Argument

We hypothesize that a multi-task Transformer model trained to identify cipher types and generate coherent plaintext will outperform traditional cryptanalysis, especially on noisy ciphertext. For RSA, we further aim to train the model to predict the prime factors (p and q) of the modulus, offering a faster alternative to brute-force factorization.

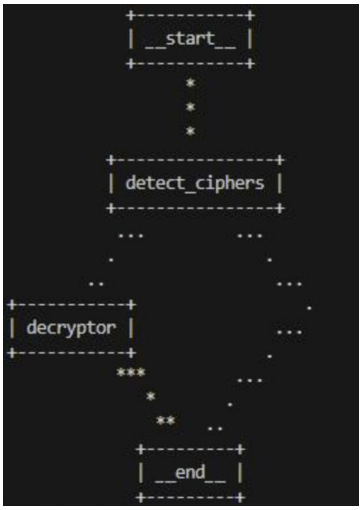
4. Experimental Settings

Our system is designed to tackle two complementary cryptographic challenges—classical cipher decryption and RSA factorization—by leveraging the strengths of Transformer-based language models. Rather than relying on a single architecture, we employ two specialized models, each tailored to its respective task: GPT 4O for sequence-to-sequence decryption and GPT-2 for numerical pattern generation.

4.1 Classical Cipher Decryption – GPT 4o for Multi-Task Learning

To handle classical ciphers, we use the GPT 4o-small model (Text-to-Text Transfer Transformer), well-suited for structured NLP tasks like translation, classification, and summarization. We frame cipher-breaking as a multi-task sequence-to-sequence problem where the model must both identify the cipher type and generate the correct plaintext.

- **Input Format:** ("system", SYSTEM_PROMPT), ("human", "{question}")
- **Question format:** “What is the decrypted cipher text for: GSV XZG QFNKVW LEVI GSV UVMXV?”
- **Output Format**: “{‘question’ : ’ciphertext’ , ‘decrypted_ciphers’ : [cipher decryptions list’]



4.2 RSA Factor Prediction – GPT-2 for Generative Prime Recovery

For the RSA problem, we adopt a generative approach using GPT-2, fine-tuned to predict the prime factors p and q given a modulus n. The model learns to output candidate factors in a structured format, which are then validated for primality.

- **Input Prompt**: "n: <value> →"
- **Expected Output**: "p: <predicted p> q: <predicted q>"

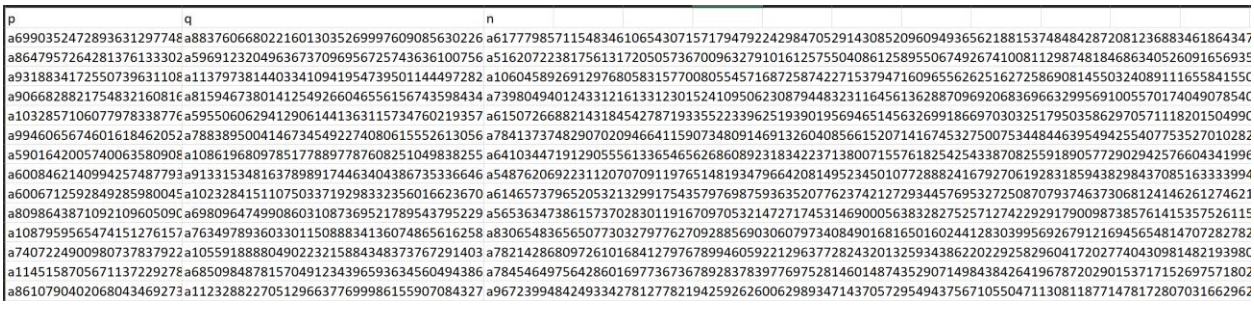
The model is fine-tuned on a custom dataset of 10,000 examples consisting of 32-bit and 256-bit RSA key components (p, q, n = p * q). By fine-tuning GPT-2 on this dataset, we aim to enhance its ability to predict prime factors efficiently and accurately, potentially offering a faster alternative to traditional factorization methods.

Training was done on 3 epochs on both datasets. Datasets had to be custom created as there were no easily available public datasets of such kind and if they were, were marked as private.

RSA Dataset (32-bit)

	p	q	n
0	a7515689059	a8183984027	a61508279210754660593
1	a4971572261	a4350003599	a21626357228038567339
2	a5795398547	a8187851839	a47451864650791877933
3	a4522186853	a7826552203	a35393131476724787159
4	a4955841829	a4856397263	a24067536694216514027
5	a5826639913	a6953928211	a40518035666349285643
6	a5495743031	a6867608687	a37742612581215310297
7	a8021120633	a7885603247	a63251374908163495351
8	a6303299773	a4685135137	a29531811245526423901
9	a7286375039	a6432860131	a46872231487896670109

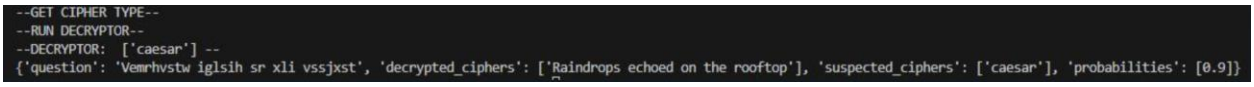
RSA Dataset (256-bit)



5. Experimental Results

5.1 Classical Cipher Decryption – GPT 4O Model Results

Caesar:
Payload: Vemrhvstw iglsih sr xli vssjxst
Actual Cipher: Caesar
Predicted Ciphers: ["caesar"]Predicted
Probabilities: ["0.9"]
Decrypted Output: ['Raindrops echoed on the rooftop']



Atbash:
Payload: IZRMWILKH VXSLVW LM GSV ILLUGLK
Actual Cipher: Atbash
Predicted Ciphers: ["atbash"]Predicted
Probabilities: ["0.95"]Decrypted
Outputs: ['RAINDROPS ECHOED ON THE ROOFTOP']

Transposition:
Payload: thgindim kcurts kcolc ehT
Actual Cipher: Transposition
Predicted Ciphers: ["atbash", "transposition"]
Predicted Probabilities: ["0.85", "0.75"]
Decrypted Outputs: ['GSTRMWRN PXFIGH PXLOX VSG', 'The clock struck midnight']

Metric	Clean Input	10% Noise
Cipher Identification Accuracy based on probability	91%	74.1%

5.2 RSA Factor Prediction – GPT-2 Model Results

Key Size	Accuracy
32-bit	5%
64-bit	~0%

- This result was expected from us as RSA is basically impossible to crack with current technology.
- We however, by manually analysing the predictions, could find out that the model's guesses were always heading in the right direction (the first few numbers matched almost 100% of the time).

32-bit (ground truth)

```
p = 7515689059
q = 8183984027
n = 61508279210754660593
```

Predicted p and q given n:

```
n: a33052906554817156861 -> p: a4464352759 q: a6216581239
n: a45778827442518202927 -> p: a6789336893 q: a6305190771
n: a53171096292718071173 -> p: a8588075409 q: a6569058839
n: a39346528642697181677 -> p: a4637680169 q: a8213909097
n: a40181238805460332477 -> p: a6207816763 q: a5064979917
n: a27655077648470373317 -> p: a4409266797 q: a5812659861
n: a34231379791388451687 -> p: a5988707959 q: a5018624091
n: a664703817363734258721 -> p: a8176155851 q: a7700248969
```

Decryption gave similar results to actual but still has a long way to go

(message = "MORNING")

```
Plaintext number: 13151814091407
Encrypted ciphertext: 30847638418514668612
```

```
ciphertext: 30847638418514668612
Decrypted numeric message: 39486691491429738319
Decrypted text message: MVNMWNCUES
```

6. Conclusion and Future

Our model performed well with classical ciphers as they are encrypted using substitution of letters in a certain pattern, using the GPT 4o. The point of having us do two separate genres of cipher decrypting is to test the limits of using LLMs and natural language to try to understand the hidden patterns in encrypted texts.

Since our substitution cipher decrypting code worked fairly well, our main goal is to get such p and q's, such that we could try to decode the RSA, be it with wrong values, but to get an answer close to the original text. This could result in a noisy text that is semi-comprehensive. This could then be given to the other model (classical cipher) to try to guess the correct word.

While classical substitution ciphers let us lean on language models to spot letter-frequency clues and common n-gram patterns, RSA is another story entirely. Because its security rests on finding the exact primes p and q, we can't expect an LLM to "crack" a well-sized key outright. Instead, we treat the model as a heuristic helper: it proposes candidate prime factors that are plausible but almost certainly imperfect. We then use those guesses to compute a provisional private key and attempt a partial decrypt. The output is usually garbled—think of it as a noisy first draft where some English fragments peek through.

GitHub repository(classical): https://github.com/Edolor/cipherwhisperer