BREAKING CIPHER ENCRYPTIONS

Ishaan Iyer ii10
Nneoma Amechi na82

AGENDA

INTRODUCTION

PROBLEM STATEMENT

METHOD

EXPERIMENTS

CONCLUSION

"I told her she was **"Euxwlixo"** in Caesar +3. She blocked me. Turns out she thought I called her a Pokémon."

INTRODUCTION

- 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.
- Encryption has come a long way from substitution ciphers like Caesar Cipher in ancient Rome to mathematically driven ones like RSA.

PROBLEM STATEMENT

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 mini, with exploration into GPT-2 to be run locally.

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 and identify patterns

Moreover, this is a fun idea, to see , by using AI, how close we can get to breaking such encryptions that are mathematically so complex, that modern computers just cannot brute force their way in.

METHOD

METHOD OVERVIEW

Classical Ciphers

- Detect if an input contains a hidden cipher
- Identify the type of cipher used
- Extract the encoded word
- Decrypt the word using the corresponding model of the cipher used (e.g. Atbash or Caesar)
- Done using Langchain OpenAi (GPT 4o mini)

RSA (Rivest–Shamir–Adleman)

- Given an input n, e and c, try to use GPT 2 model stored locally
- “Guess” the factors p and q such that p*q = n through the GPT model.
- Better shot at cracking RSA than Brute force as it will computationally take forever to prime factorize the given n.

METHOD – CLASSICAL CIPHERS

DETECTION AND IDENTIFICATION

- LLM used : GPT 4o mini (OpenAI Langchain)
- Prompt: ("system", SYSTEM_PROMPT), ("human", "{question}")
- The question is a Natural Language Question like :

"What is the decrypted cipher text for: GSV XZG QFNKVW LEVI GSV UVMXV?"

Returned:

- suspected_ciphers: ["cipher_suspected_by_model"]
- Probabilities: confidence per prediction
- New_question: cipher text extracted

METHOD – CLASSICAL CIPHERS

DECRYPTION

- Method: Rule based decryption
  - Caesar – 26 possible shifts, scores based on English words and determines the most optimal one
  - Atbash – maps A↔Z, B↔Y... According to what the model finds most logical
  - Affine – tries (a, b) pair combinations (where a is coprime with 26), uses modular inverse
  - Bacon – treats input as binary, decodes using a 5-bit character mapping
- Selection Logic: Use english_words to rank decryption quality
- Return the best match

METHOD – RSA (WORKING)

Encryption

- Given: p, q, e (public key), m (plaintext)
- Take two prime numbers p and q, generally 512 to 4096 bits, and compute n (= p*q)
- Compute $\phi(n) = (p-1)(q-1)$
- Compute Private key d such that

  $$d*e = 1 \bmod \phi(n)$$

- c = m^e mod(n)
- C = Ciphertext

Decryption

- Given: n, e, c
- We decrypt the ciphertext c using the relation:
- m = c^d mod(n)
- Now, if were an authenticated party receiving the message, we already have the private key (d), but if a hacker were to decrypt the code, they would need p and q, to calculate d.

METHOD – RSA

Dataset creation:

- Generated 10000 prime p and q pairs and multiplied them to get the corresponding n (32 and 256 bits)
- Used random.getrandbits(256), and isprime() to do so. Converted to string and added ‘a’ before each number to avoid any short automatic notations.

Prompt format: n: a<value> -> p: a<value> q: a<value>

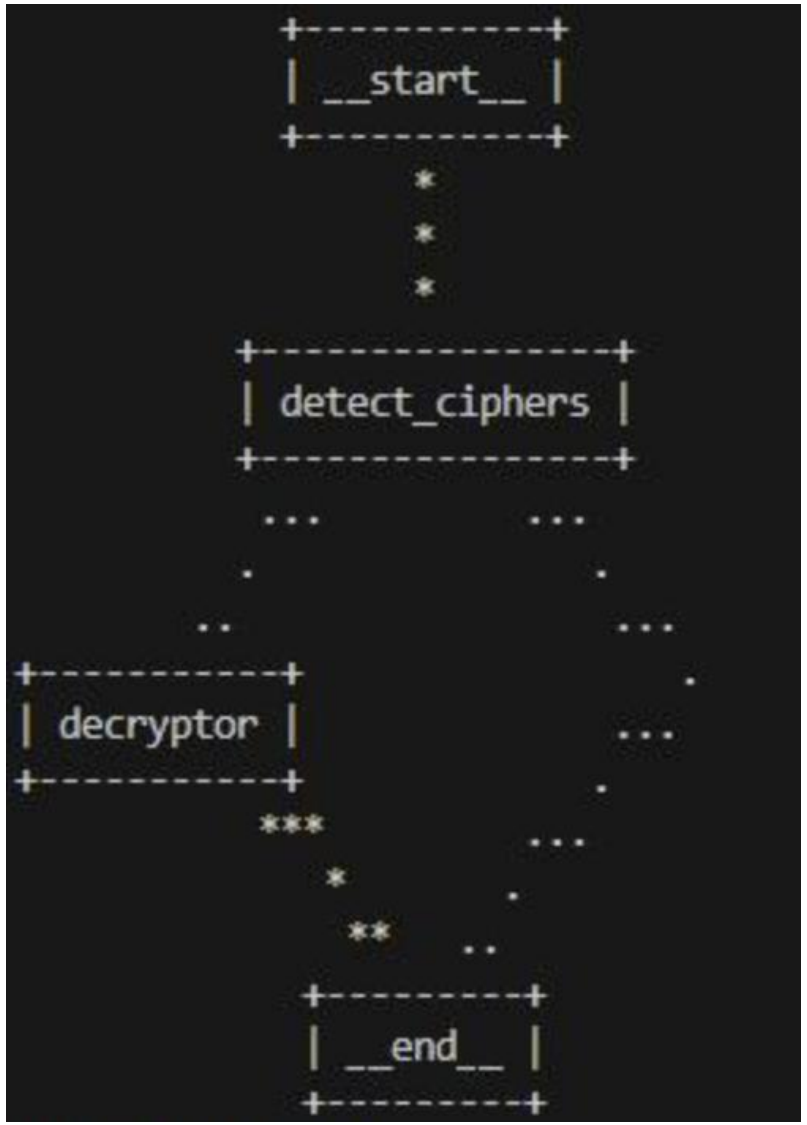
Training: HF trainer API. Model finetuned and saved locally for immediate usage.

Model: GPT2 (local)

Prediction input: n ->

Output: n -> p: value q: value

EXPERIMENTS



CLASSICAL CIPHERS

- The model identifies the possible ciphers used in the encryption and stores them in an array.
- Then, using the decryption models of all the ciphers in the array, we get the possible corresponding decoded plaintexts
- Format:

DECRYPTOR: [cipher_type list]

{‘question’ : ’ciphertext’ , ‘decrypted_ciphers’ : [cipher decryptions list’]

CLASSICAL CIPHERS

Caesar:

Payload: Vemrhvstw iglsih sr xli vssjxst

Actual Cipher: Caesar

Predicted Ciphers: ["caesar"]Predicted

Probabilities: ["0.9"]

Decrypted Output: ['Raindrops echoed on the rooftop']

```
--GET CIPHER TYPE--
--RUN DECRYPTOR--
--DECRYPTOR:  ['caesar'] --
{'question': 'Vemrhvstw iglsih sr xli vssjxst', 'decrypted_ciphers': ['Raindrops echoed on the rooftop'], 'suspected_ciphers': ['caesar'], 'probabilities': [0.9]}
```

CLASSICAL CIPHERS

Atbash:

Payload: IZRMWILKH VXSLVW LM GSV ILLUGLK

Actual Cipher: Atbash

Predicted Ciphers: ["atbash"]Predicted

Probabilities: ["0.95"]Decrypted

Outputs: ['RAINDROPS ECHOED ON THE ROOFTOP']

```
--GET CIPHER TYPE--
--RUN DECRYPTOR--
--DECRYPTOR:  ['atbash'] --
{'question': 'IZRMWILKH VXSLVW LM GSV ILLUGLK', 'decrypted_ciphers': [('RAINDROPS ECHOED ON THE ROOFTOP', 2)], 'suspected_ciphers': ['atbash'], 'probabilities': [0.95]}
```

CLASSICAL CIPHERS

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']

```
--GET CIPHER TYPE--
--RUN DECRYPTOR--
--DECRYPTOR:  ['atbash', 'transposition'] --
{'question': 'thgindim kcurts kcolc ehT', 'decrypted_ciphers': [('GSTRMWRN PXFIGH PXLOX VSG', 0), 'The clock struck midnight'], 'suspected_ciphers': ['atbash', 'transposition'], 'probabili
ties': [0.85, 0.75]}
```

RSA

Dataset:

32 bits →

256 bits

↓

	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

p	q	n
a699035247289363129774	a883760668022160130352699976090856302	a617779857115483461065430715717947922429847052914308520960949365621881537484842872081236883461864347
a864795726428137613330	a596912320496367370969567257436361007	a516207223817561317205057367009632791016125755040861258955067492674100811298748184686340526091656935
a931883417255073963110	a113797381440334109419547395011444972	a106045892691297680583157700805545716872587422715379471609655626251627258690814550324089111655841550
a906682882175483216081	a815946738014125492660465561567435984	a739804940124331216133123015241095062308794483231164561362887096920683696632995691005570174049078540
a103285710607797833877	a595506062941290614413631157347602193	a615072668821431845427871933552233962519390195694651456326991866970303251795035862970571118201504990
a994606567460161846205	a788389500414673454922740806155526130	a784137374829070209466411590734809146913260408566152071416745327500753448446395494255407753527010282
a590164200574006358090	a108619680978517788977876082510498382	a641034471912905556133654656268608923183422371380071557618254254338708255918905772902942576604341996
a600846214099425748779	a913315348163789891744634043867353366	a548762069223112070709119765148193479664208149523450107728882416792706192831859438298437085163333994
a600671259284928598004	a102328415110750337192983323560166236	a614657379652053213299175435797698759363520776237421272934457695327250870793746373068124146261274621
a809864387109210960509	a698096474990860310873695217895437952	a565363473861573702830119167097053214727174531469000563832827525712742292917900987385761415357526115
a108795956547415127615	a763497893603301150888341360748656162	a830654836565077303279776270928856903060797340849016816501602441283039956926791216945654814707282782
a740722490098073783792	a105591888804902232158843483737672914	a782142868097261016841279767899460592212963772824320132593438622022925829604172027740430981482193980
a114515870567113722927	a685098487815704912343965936345604943	a784546497564286016977367367892837839776975281460148743529071498438426419678720290153717152697571802
a861079040206804346927	a112328822705129663776999861559070843	a967239948424933427812778219425926260062989347143705729549437567105504711308118771478172807031662962

RSA

Finetuning the model:

Prompt format: n: value -> p: value q: value

[1665/1665 45:36, Epoch 3/3]

Step	Training Loss
500	3.280800
1000	3.170900
1500	3.132100

RSA

- Unsurprisingly, the accuracy of finetuned model is 5%
- However, the predicted p and q for a given n, are similar to the actual values of p and q.
- For the combination:

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

- We get the predicted

```
n: a330529065548171568 61 -> 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
```

RSA

- For the combination:

```
p = 8021120633
q = 7885603247
n = 63251374908163495351
```

- We get the predicted

```
🔍 Output: n: a63251374908163495351 -> p: a6395875463 q: a8005913693
n: a55238814253377771273 -> p: a7212371439 q: a7700262073
n: a22941848774428981067 -> p: a4399602381 q: a4665524097
n: a33159517675060352963 -> p: a5945384583 q: a6036783683
n: a643545950020385836351 -> p: a8276686173 q: a8371369707
n: a204545507924792899059 -> p: a4468221847 q: a4969640079
n: a39129545890029097557 -> p: a5624786921 q: a7378493561
n: a634456589051845340069 -> p: a7766696769 q: a7402768669
n: a30206544862216306073 -> p: a4530682857 q: a7216992677
n: a28681669263023681347 -> p: a4450139889 q
```

RSA

```python
# Given RSA public key components
p = 8021120633
q = 7885603247
n = 63251374908163495351
e = 65537

# Message to encrypt
message = "MORNING"

# Convert message to numbers (A=1 to Z=26), zero-padded to 2 digits each
number_str = ''.join([f"{ord(char)-64:02d}" for char in message.upper()])
m = int(number_str)

print("Plaintext number:", m)

# Encrypt using RSA: c = m^e mod n
c = pow(m, e, n)

print("Encrypted ciphertext:", c)
```

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

This output took around 3.5 hours of training and `1.5 hours of testing on 10000 samples and two models (earlier used T5) showcasing the strength of RSA in modern cryptography.

```python
# Split into 2-digit chunks
decoded_chars = []
for i in range(0, len(m_str), 2):
    val = int(m_str[i:i+2]) % 26
    if val == 0:
        val = 26
    decoded_chars.append(chr(val + 64))  # 1->A, 2->B, ..., 26->Z

decoded_message = ''.join(decoded_chars)
print("Decrypted text message:", decoded_message)
```

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

CONCLUSION & FUTURE

CONCLUSION

Classical

- Overall probability metrics for prediction of decoded text across all the classical ciphers is 91% in confidence
- Our model performed well with classical ciphers as they are encrypted using substitution of letters in a certain pattern

RSA

- GPT2 performed decent for 32-bit prime p and q pairs as we were expecting worse results, and was similarfor 256-bit numbers.
- 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).

FUTURE

- 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 qs, such that we could try to decode the RSA, be it with wrong values, but to get an answer close to the original.
- 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 cored word.

GUNAX LBH

IN ROT +13