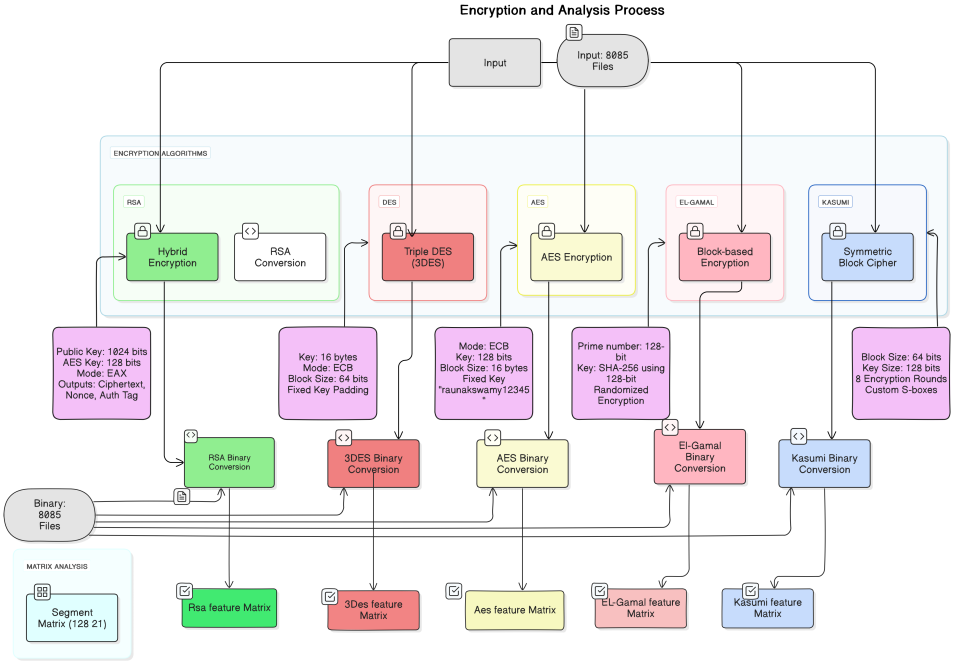
Cryptonet

Data collection

Source - https://anc.org/data/oanc/download/

There are a total of 8085 txt files from source. The available is diverse in terms of length.

Getting the cipher texts



Moreover the cipher texts are converted to binary so that machine can process it.

Technical aspects of the encryption algorithms

RSA

To get the secret key required to decrypt that data, authorized recipients publish a public key while retaining an associated private key that only they know. The sender then uses that public key and RSA to encrypt and transmit to each recipient their own secret AES key, which can be used to decrypt the data.

Hybrid Encryption Algorithm Details:

1. Key Sizes and Encryption Methods:
    - RSA Public Key: 1024 bits (base64 encoded)
    - AES Key: 128 bits (16 bytes)
    - AES Mode: EAX (Encryption with Authentication)

- Encryption Type: Hybrid (Asymmetric + Symmetric)

Encryption Process:

2. a) AES Encryption:
    - Generates a random 128-bit AES key
    - Encrypts plaintext using AES-EAX mode
    - Produces:
        - Ciphertext
        - Nonce (Number used once)
        - Authentication Tag

2. b) RSA Encryption:

- Converts encrypted AES key to hexadecimal string
- Encrypts the AES key using the public RSA key

3. Output Sizes:
    - AES Key: 16 bytes
    - RSA Encrypted AES Key: Approximately 128 bytes (hexadecimal representation)
    - Nonce: 16 bytes (typical for EAX mode)
    - Authentication Tag: 16 bytes
    - Ciphertext: Same size as original plaintext

6. Block Size:

- AES Block Size: 128 bits (16 bytes)
- RSA Public Key Block Size: 1024 bits

KASUMI

Here's a concise breakdown of the Kasumi encryption algorithm implementation:

Algorithm Overview:

- Type: Symmetric Block Cipher (Kasumi)
- Origin: Used in 3G mobile communications security

Key Characteristics:

1. Block Size: 64 bits (8 bytes)
2. Key Size: 128 bits
3. Number of Rounds: 8

Key Generation Process:

- Derives multiple subkeys from master key
- Uses bit shifting and XOR operations

- Creates 9 different key arrays:

- KL1, KL2

- KO1, KO2, KO3

- KI1, KI2, KI3

Encryption Mechanism:

1. Key Setup:

- Transforms master key

- Generates multiple round-specific keys

- Uses bit manipulation techniques

2. Encryption Steps:

- Splits 64-bit block into left and right halves

- Applies 8 rounds of:

- Function FI (substitution)

- Function FO (Feistel network)

- Function FL (linear transformation)

3. Key Transformation:

- XORs master key with predefined constant

- Applies complex key scheduling algorithm

Unique Features:

- Uses two custom S-boxes (S7, S9)
- Implements non-linear transformations
- Provides confusion and diffusion

Output Specifications:

- Input: Variable length text
- Output: Hexadecimal representation
- Padding: PKCS7-like padding to 64-bit blocks
- Binary conversion of hex output

3DES

Encryption Algorithm Details:

1. Key Characteristics:

- Encryption Algorithm: Triple DES (3DES)
- Mode of Operation: ECB (Electronic Codebook)
- Key Length: 16 bytes (128 bits)

- Block Size: 64 bits (8 bytes)

2. Encryption Process:

a) Key Preparation:

- Uses a fixed key: `raunakswamy12345`
- Key is padded to meet 3DES key requirements

b) Encryption Steps:

- Pad input text to block size
- Encrypt using 3DES-ECB mode
- Convert encrypted text to hexadecimal
- Convert hexadecimal to binary string

3. Output Sizes:

- Key Size: 16 bytes
- Block Size: 64 bits
- Ciphertext: Padded to multiple of block size (< 8 bytes to 8, < 16 to 16)
- Hexadecimal Output: Variable (depends on input text length)
- Binary Output: Conversion of hexadecimal

4. Folder Processing:

- Recursively finds .txt files
- Encrypts each file individually

AES

Algorithm Working:

1. The script uses AES (Advanced Encryption Standard) in ECB (Electronic Codebook) mode to encrypt text files.
2. It walks through a specified input folder, finds all .txt files, and encrypts them.
3. Encryption process:
    - Removes trailing spaces from the plaintext
    - Pads the text to match AES block size
    - Converts the text to bytes
    - Encrypts using a fixed key

Key Details:

- Encryption Mode: AES-ECB (Electronic Codebook)
- Key Size: 128 bits (16 bytes)
- Block Size: 16 bytes (128 bits)

- Key Used: "raunakswamy12345" (exactly 16 bytes)

Output Characteristics:

- Input file: Original .txt file
- Output file: .bin file with hexadecimal encrypted content
- Encrypted text will be larger than input due to padding

Each 16-byte block of input is encrypted independently

ElGamal

Here's a concise breakdown of the encryption algorithm:

Algorithm Type: El Gamal Encryption with Custom Implementations

Key Generation Process:

- Uses SHA-256 to generate a 128-bit key
- Generates a prime number (p) close to 2^127
- Finds a primitive root (g) for the prime
- Creates public key: (p, g, h)
- Generates a private key (x)

Encryption Mechanism:

1. Converts input text to bytes
2. Breaks message into blocks
3. For each block:

- Generates random k

- Computes c1 = g^k mod p

- Computes shared secret s = h^k mod p

- Encrypts block as c2 = (message * s) mod p

Key Characteristics:

- Prime (p): 128-bit
- Primitive Root (g): Variable length
- Public Key Components:

- p: 128-bit prime

- g: Primitive root

- h: Public key element

- Private Key (x): Derived from hash, < p-1

Output Specifications:

- Converts encrypted blocks to binary string

- Output saved as .bin file
- Binary representation of encrypted blocks
- Output length varies with input text length

Security Features:

- Randomized encryption (each encryption differs)
- Uses Miller-Rabin primality testing
- Derives keys from input passphrase

Unique Aspects:

- Custom prime generation
- Efficient primitive root finding
- Block-based encryption approach

Feature extraction from binary sequence

> According to NIST we extract 21 features from 3 methods

NIST Document link : (A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications)
https://drive.google.com/file/d/1xFVxq44DxGM1COcbeucTxRZMAJ5Eo20k/view?usp=sharing

3 methods and corresponding features

Frequency within the block analysis	Chi-Square Statistic
	p-value
	Average Proportion of ones across blocks
	Standard deviation of proportions
	Maximum block proportion
	Minimum block proportion
	Number of blocks with proportion > 0.5

Run test analysis	Z-statistic
	Total number of runs
	Number of runs of length 1
	Number of runs of length 2
	Number of runs of length 3
	Number of runs of length 4+
	Proportion of 1s in the sequence

Serial test analysis	$\nabla\psi^2\_m$ statisti
	p-value
	Entropy of m-bit patterns
	Maximum m-b pattern frequency
	Minimum m-bi pattern frequency
	Number of missing m-bit patterns
	Ratio of observed to expected most frequent pattern

How each feature is extracted?

Frequency within blocks analysis

The method analyzes the distribution of 0s and 1s within sub-blocks of the ciphertext. It helps identify if there are any biases in the distribution of bits within the blocks.

In a perfectly random ciphertext, we would expect each block to have about 50% 1s and 50% 0s.

Working

1. Divide the input sequence into ‘n’ non-overlapping blocks of ‘M’ bits.
2. Determine the proportion of ones in each block.
3. Compare these proportions to the expected value of 0.5 using a chi-square test.

Notations and assumptions

For a sequence of length n divided into N = ⌊n/M⌋ blocks of M bits:

- The test statistic follows a χ^2 distribution with N degrees of freedom. Significance alpha = 0.05
- Calculate π_i = (ones in i-th block) / M for each block
- Compute $\chi^2 = 4M * \Sigma(\pi_i - 0.5)^2$ for i = 1 to N

Example calculation

Consider the binary sequence: 1100001111000011110000111100 (n=28) and M=7 (block size)

So N=4 (Degree of freedom)

	Block	π
1	1100001	3/7=0.428
2	1110000	3/7=0.428
3	1111000	4/7=0.571
4	0111100	4/7=0.571

Chi-Square Statistic	$\chi^2 = 4 * 7 * [(0.428 - 0.5)^2+(0.428 - 0.5)^2+(0.571 - 0.5)^2+(0.571 - 0.5)^2] = 0.5726$
p-value	$1 - CDF(\chi^2(4), 0.5726) = .966059$
Average Proportion of ones across blocks	0.4995
Standard deviation of proportions	0.0715
Maximum block proportion	0.571
Minimum block	0.428

proportion	
Number of blocks with proportion > 0.5	2

Run test analysis

This index looks at consecutive runs of the same bit (0 or 1) in the ciphertext. A run is a maximal sequence of consecutive bits of either all ones or all zeros.

A higher runs index might indicate less randomness in the ciphertext, as it suggests longer stretches of the same bit.

Working

1. Count the number of runs of each length.
2. Compare these counts to the expected counts for a random sequence.

Notations and assumptions

For a random sequence of length n with n_1 ones and n_o zeros:

- Expected number of runs: $e = 1 + (2n_1n_o) / n$
- Variance of the number of runs: $\sigma^2 = (2n_1n_o(2n_1n_o - n)) / (n^2(n - 1))$

The test statistic is: $Z = (r - e) / \sigma$

Where r is the observed number of runs.

Example calculation

Consider the binary sequence: 1011010111

1. Count runs

Iteratively traverse the sequence	Length of run
1	1
0	1
11	2
0	1
1	1
0	1
111	3

Total observed number of runs: 7

2. Calculate expected runs: $n_1 = 7$, $n_o = 3$, $n = 10$
   $e = 1 + (2 * 7 * 3) / 10 = 5.2$
3. Calculate variance: $\sigma^2 = (2 * 7 * 3 * (2 * 7 * 3 - 10)) / (10^2 * 9) \approx 1.29$
4. Calculate test statistic: $Z = (7 - 5.2) / \sqrt{1.29} \approx 1.58$

Z-statistic	1.58
Total number of runs	7

Number of runs of length 1	5
Number of runs of length 2	1
Number of runs of length 3	1
Number of runs of length 4+	0
Proportion of 1s in the sequence	7/10=0.7

Serial test analysis

This index examines all possible sub-sequences within the ciphertext. This index helps identify patterns or biases in the sub-sequences of the ciphertext. In a truly random sequence, you'd expect each possible sub-sequence to occur with roughly equal frequency.

Working

1. Count occurrences of all possible overlapping m-bit patterns.
2. Compare these counts to the expected counts for a random sequence.
3. Compute test statistics based on the differences between observed and expected frequencies.

Notations and assumptions

For a binary sequence of length n and pattern length m:

- Expected frequency of each m-bit pattern: n/2^m
- Compute ψ^2_m and $\psi^2_{(m-1)}$: $\psi^2_m = (2^m/n) * \Sigma v^2_i - n$, where v_i is the frequency of the i-th m-bit pattern $\psi^2_{(m-1)}$ is calculated similarly for (m-1)-bit patterns
- Test statistic: $\nabla\psi^2_m = \psi^2_m - \psi^2_{(m-1)}$
- $\nabla\psi^2_m$ follows a χ^2 distribution with 2^(m-1) degrees of freedom. Significance alpha = 0.05

Example calculation

Consider the binary sequence: 0011011101 (length 10)
Let's use m = 3 (pattern length)
Degree of freedoms = 4

1. Count 3-bit patterns

Pattern	Count
000	0
001	1
010	0
011	2
100	0
101	2
110	2
111	1

2. Count 2 bit patterns

Pattern	Count
00	1
01	3
10	2
11	3

3. Compute $\psi^2_3 = (2^3/10) * (0^2 + 1^2 + 0^2 + 2^2 + 0^2 + 2^2 + 2^2 + 1^2) - 10 \approx 1.2$
4. Compute $\psi^2_2 = (2^2/10) * (1^2 + 3^2 + 2^2 + 3^2) - 10 \approx 0.4$
5. Calculate $\nabla\psi^2_3 = \psi^2_3 - \psi^2_2 \approx 0.8$
6. Calculate p-value: p-value = $1 - \text{CDF}(\chi^2(4), 0.8) \approx 0.9384$

$\nabla\psi^2_3$ statistic	0.8
p-value	0.9883
Entropy of m bit patterns	2.75
Maximum 3-bit pattern frequency	2
Minimum 3-bit pattern frequency	0
Number of missing 3-bit patterns	3
Ratio of observed to expected most frequent pattern	2/(10/8)=1.6

How one datapoint looks?

Let's say a binary cipher text is of length N.
We have taken a block of 128 and generate 21 features as demonstrated in previous section.
Total number of blocks = N/128 = B

One datapoint is matrix of (B, 21)

```
# 21 features
{
    'chi_square': ...,
    'p-value': ...,
    'avg_prop': ...,
    'std_dev_prop': ...,
    'max_prop': ...,
    'min_prop': ...,
    'num_props_over_05': ...,

    'z_statistic': ...,
    'num_runs': ...,
    'num_runs_1': ...,
    'num_runs_2': ...,
    'num_runs_3': ...,
    'num_runs_4_plus': ...,
    'prop_1s': ...,

    'mbit_chisquare': ...,
    'mbit_pvalue': ...,
    'mbit_entropy': ...,
    'max_mbit_freq': ...,
```

```
    'min_mbit_freq': ...,
    'num_missing_mbits': ...,
    'ratio_observed_to_expected': ...,
}
```

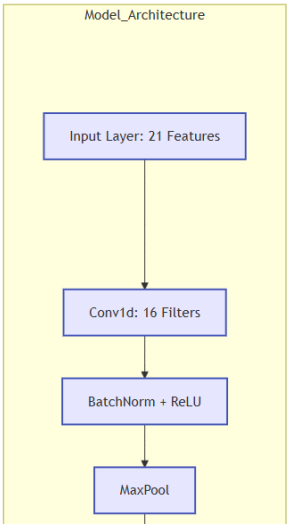
Why deep learning?

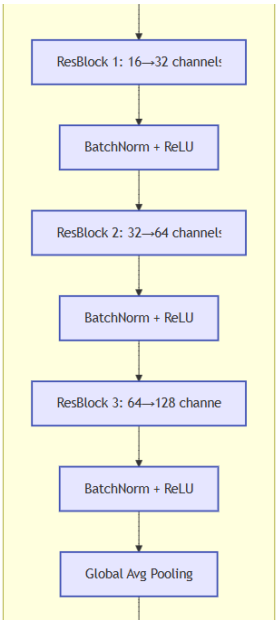
In our research, we opted for a deep learning model instead of a traditional machine learning model due to the complexity and cluttered nature of the features in our dataset. With 21 features, a 3D visualization using techniques like t-SNE and PCA revealed that the feature space was highly non-linear and lacked clear separability. This level of complexity made it challenging for traditional machine learning algorithms to effectively capture the intricate patterns and relationships within the data.

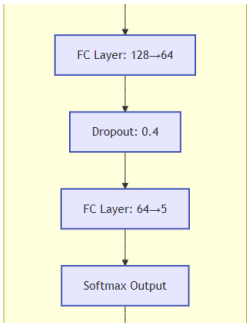
Deep learning models excel in scenarios where features are high-dimensional and non-linearly correlated, as they are capable of hierarchical feature extraction. This ability to deeply analyze and encode feature interactions is crucial for our dataset, where the underlying structure is obscured in lower dimensions.

Consequently, the deep learning model significantly outperformed traditional machine learning approaches, demonstrating its strength in understanding and leveraging the complex feature space to achieve superior performance.

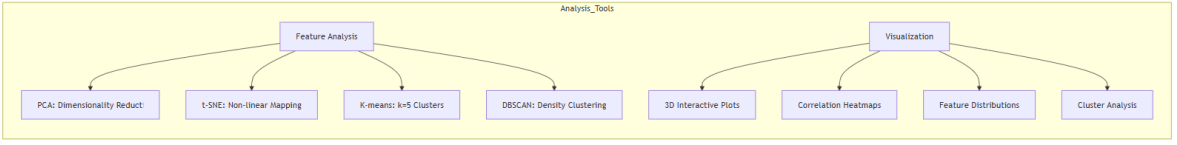
Model architecture



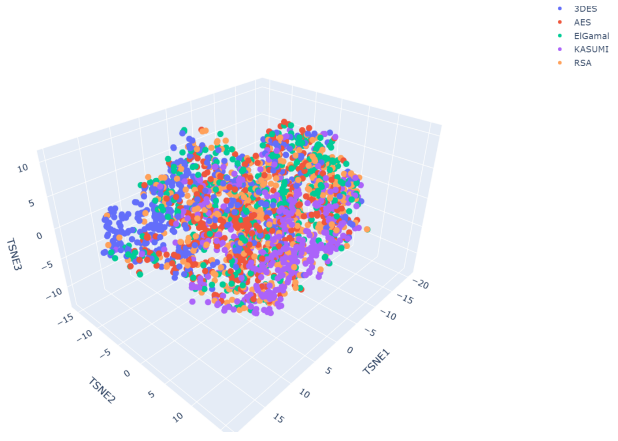




Analysis tools used

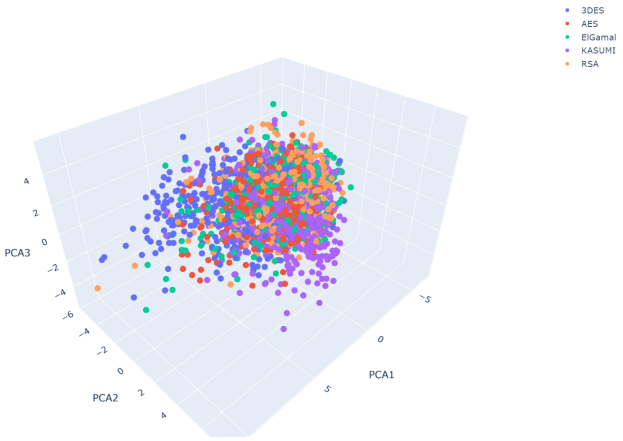


3d visualization with t-SNE



https://keesarivigneshwarreddy.github.io/Cipher-text-SNE.github.io/

3d visualization with PCA



Website for PCA Visualization - https://keesarivigneshwarreddy.github.io/Cipher-text-features-visualization-PCA.github.io/

Exploring train set

PCA Explained Variance	20.42%
kmeans_silhouette	0.094
kmeans_calinski	698.699
dbscan_silhouette	-0.192
dbscan_calinski	17.082

Evaluation results on test set

Overall Accuracy: 88.60%
Average Confidence: 0.83

Per-class analysis

	Accuracy	Precision	Recall	F1-Score	Support
3DES	90.07%	0.941	0.901	0.921	1339.0
AES	84.65%	0.965	0.847	0.902	1316.0
ElGamal	91.31%	0.958	0.913	0.935	1335.0
KASUMI	88.27%	0.980	0.883	0.929	1338.0
RSA	88.65%	0.669	0.886	0.763	1277.0

Best Validation Accuracy: 89.17%
Best Epoch: 98