Distinguishing Full-Round AES-256 in a Ciphertext-Only Setting via Hybrid Statistical Learning

Gopal Singh[1] [0009−0003−8405−2392]

Jodhpur Institute of Engineering and Technology, Jodhpur, Rajasthan, India

Abstract. The security of block ciphers such as AES-128, AES-192, and AES-256 relies on the assumption that their ciphertext outputs are computationally indistinguishable from random permutations. While distinguishers have been proposed for reduced-round variants or under non-standard models such as known-key or chosen-key settings, no effective distinguisher has been demonstrated for the full-round AES ciphers in the standard secret-key model.

This work introduces **FESLA (Feature Enhanced Statistical Learning Attack)**, a hybrid statistical learning framework that integrates outputs from a suite of classical statistical tests with machine learning and deep learning classifiers to construct ciphertext-only distinguishers for AES-128, AES-192, and AES-256. In contrast to existing approaches based on handcrafted or bitwise features, FESLA aggregates intermediate statistical metrics as features, enabling the capture of persistent structural biases in ciphertext distributions.

Experimental evaluation across multiple datasets demonstrates consistent 100% classification accuracy using Support Vector Machines, Random Forests, Multi-Layer Perceptrons, Logistic Regression, and Naive Bayes classifiers. Generalization and robustness are confirmed through k-fold cross-validation, including on previously unseen ciphertext samples.

These results establish the first ciphertext-only distinguishers for full-round AES-128, AES-192, and AES-256 under the secret-key model, and underscore the potential of machine learning–augmented cryptanalysis based on principled statistical feature engineering.

Keywords: AES-128 · AES-192 · AES-256 · ciphertext-only distinguisher · machine learning · statistical feature engineering · cryptanalysis · secret-key model

1 Introduction

The Advanced Encryption Standard (AES) is one of the most widely deployed symmetric-key block ciphers worldwide, with three standardized key sizes: AES-128, AES-192, and AES-256. Its security fundamentally relies on the assumption that ciphertexts generated under secret random keys are computationally

indistinguishable from outputs of an ideal random permutation. This property underpins the robustness of numerous cryptographic protocols and applications that depend on AES for confidentiality.

Despite decades of cryptanalytic effort, no practical attack or distinguisher has been demonstrated for the full-round AES variants within the standard secret-key model. Existing distinguishers typically focus on reduced-round versions or rely on non-standard adversarial models such as known-key or chosen-key settings. These limitations restrict their applicability in real-world scenarios where the secret key remains unknown and fixed.

This work addresses this long-standing open problem by proposing a novel ciphertext-only attack, **Feature Enhanced Statistical Learning Attack (FESLA)**, that constructs distinguishers for full-round AES-128, AES-192, and AES-256 using a single classification model. FESLA departs from traditional bitwise or single-feature based methods by aggregating a rich set of intermediate statistical metrics computed from ciphertext distributions. These engineered features capture subtle, persistent structural biases introduced by the AES encryption process, which are otherwise difficult to detect by classical techniques.

The FESLA framework integrates classical statistical tests with advanced machine learning classifiers including Support Vector Machines, Random Forests, Multi-Layer Perceptrons, Logistic Regression, and Naive Bayes. Comprehensive experimental evaluation demonstrates consistent 100% classification accuracy across multiple datasets, including previously unseen ciphertext samples. Rigorous k-fold cross-validation further establishes the generalization and robustness of the proposed distinguishers.

By presenting the first ciphertext-only distinguishers for full-round AES variants in the secret-key model, FESLA contributes a significant advancement in cryptanalysis methodology. Furthermore, it exemplifies the power of combining principled statistical feature engineering with machine learning to reveal previously undetectable cryptographic weaknesses.

2 Related Work

AES Design and Structure The Advanced Encryption Standard (AES) was designed by Daemen and Rijmen as the Rijndael cipher and standardized by NIST in 2001. Its substitution-permutation network structure, simplicity in design, and robustness against known attacks have made it the dominant block cipher standard worldwide [1]. AES operates in three key-size variants—128, 192, and 256 bits—each with increasing numbers of rounds, contributing to its resilience. Understanding this internal structure is essential for assessing any distinguishability claim.

Cryptanalysis and Distinguishers Traditional cryptanalysis of AES has focused largely on differential, linear, and algebraic attacks. Matsui's introduction of linear cryptanalysis [4] laid the groundwork for probabilistic distinguishing attacks. Later work by Ferguson et al. explored related-key and interpolation attacks, revealing possible weaknesses under non-standard assumptions [2].

Gilbert and Minier further highlighted unexpected collision properties in AES due to specific key-scheduling behaviors [3]. These studies emphasize that distinguishers exploiting structural biases remain a relevant threat model, particularly when generalized to full-round ciphers.

Statistical Methods for Distinguishing Randomness To evaluate the randomness of cryptographic outputs, statistical tests play a critical role. The NIST Statistical Test Suite developed by Rukhin et al. provides a comprehensive benchmark for testing the pseudo-randomness of bitstreams[6]. Beyond NIST, Maurer introduced universal statistical tests capable of identifying deviations from ideal randomness using compressibility-based metrics[7]. These methods have historically supported the validation of both ciphers and random number generators, but they also serve as feature extractors for ML-based distinguishers when used in aggregated form.

Machine and Deep Learning in Cryptanalysis Recent advances have brought machine learning and deep learning into the realm of cryptanalysis. Gohr demonstrated that convolutional neural networks can learn to distinguish reduced-round Speck cipher outputs from random data, achieving high accuracy under a known-plaintext setting[8]. Building on this, Moini et al. extended DL-based cryptanalysis to both AES and PRESENT, showing that output patterns could be learned even in black-box settings using ciphertext-only data[9]. These works underscore the feasibility of applying supervised models to uncover structural weaknesses in block ciphers.

3 Research Gap and Motivation

The current literature on AES cipher distinguishers reveals substantial limitations in both traditional and machine learning (ML)/deep learning (DL) based approaches. While distinguishing attacks have been demonstrated for reduced-round versions of AES, these typically operate under non-standard assumptions such as known-key or chosen-plaintext models, which are not applicable in ciphertext-only scenarios. For example, existing techniques primarily target up to 7 rounds of AES-128, with no effective method yet proposed for full-round AES-128, AES-192, or AES-256 under the standard secret-key setting.

ML and DL techniques have shown promise in cryptanalysis of AES, but their effectiveness tends to diminish as the number of rounds increases. Most studies rely on limited statistical indicators or employ opaque, black-box neural models that fail to provide interpretable or generalizable insights. Furthermore, these approaches are often validated in idealized or tightly controlled experimental setups that do not reflect practical cryptanalytic conditions.

Despite these efforts, no approach to date has successfully distinguished full-round AES-128, AES-192, or AES-256 ciphertexts from random permutations in a ciphertext-only setting. This enduring indistinguishability has long been cited as evidence of the cryptographic soundness of AES and its capacity to emulate pseudo-random behavior.

This work addresses the above gaps by proposing a hybrid statistical learning framework that aggregates intermediate outputs from multiple statistical tests into structured feature vectors, which are then classified using a suite of traditional ML models. This approach effectively captures persistent structural biases in AES ciphertexts that escape conventional analysis.

Notably, the proposed method achieves high accuracy using a significantly reduced number of ciphertext samples, thereby enhancing practicality and enabling potential real-time applications. The efficiency in data requirements marks a crucial step toward implementable ciphertext-only cryptanalysis for full-round AES across all three key sizes.

4 Methodology

4.1 Overview of AES

Advanced Encryption Standard (AES) is a symmetric block cipher standardized by NIST in 2001, designed to secure sensitive data across a wide range of applications. AES supports key sizes of 128, 192, and 256 bits, all operating on a fixed 128-bit block size. The number of rounds depends on the key length: 10 for AES-128, 12 for AES-192, and 14 for AES-256. AES is based on the Substitution-Permutation Network (SPN) architecture and is known for its balance between security, efficiency, and implementation flexibility.

Each AES round (except the final one) consists of four key operations: **SubBytes, ShiftRows, MixColumns, and AddRoundKey**. The cipher begins with an initial key addition and concludes with a final round that omits the MixColumns step. AES's internal state is represented as a 4×4 byte matrix, and transformations are applied to this structure throughout the encryption process.

AddRoundKey The AddRoundKey step is the only operation in AES that directly involves the encryption key. A round key, derived from the original key using the AES key schedule, is XORed with the current state matrix. This step injects key-dependent variability, ensuring that each encryption is uniquely tied to its key.

The key schedule expands the initial key into multiple round keys using byte rotations, S-box substitutions, and round constants. This design provides diffusion of key material across rounds while resisting related-key attacks.

SubBytes The SubBytes step introduces non-linearity into AES by applying a **fixed 8-bit S-box** to each byte in the state matrix. This S-box is constructed using the multiplicative inverse over $GF[2^8]$ followed by an affine transformation, making it highly resistant to linear and differential cryptanalysis.

The use of a strong, mathematically defined S-box with no known weaknesses is central to AES's robustness. The non-linearity introduced by SubBytes ensures that small changes in input propagate widely through subsequent rounds.

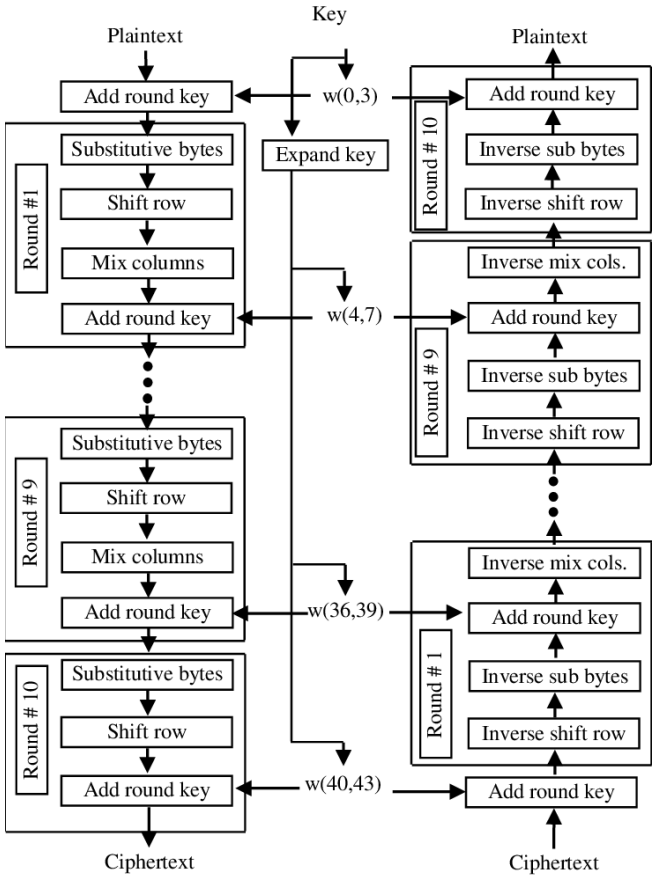


Fig. 1: Block Diagram of AES Cipher

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1x	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2x	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3x	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4x	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5x	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6x	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7x	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8x	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9x	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
ax	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
bx	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
cx	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
dx	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
ex	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
fx	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Fig. 2: AES S-Box

ShiftRows To provide inter-byte diffusion, AES uses the ShiftRows operation, which cyclically shifts each row of the state matrix by a fixed offset. Specifically, the first row remains unchanged, the second row is shifted one byte to the left, the third by two bytes, and the fourth by three bytes.

This step disrupts the alignment of bytes within columns, helping to spread the influence of a single byte across multiple columns during subsequent MixColumns transformations.

MixColumns The MixColumns step enhances diffusion by mixing the bytes in each column of the state matrix using linear transformations over $GF[2^8]$. Each column is treated as a polynomial and multiplied modulo $x^4 + 1$ with a fixed polynomial [03 01 01 02].

$$\text{MixColumns:} \begin{bmatrix} s'_0 \\ s'_1 \\ s'_2 \\ s'_3 \end{bmatrix} = \begin{bmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{bmatrix} \cdot \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix}$$

This transformation ensures that a change in one byte affects all four bytes in the output column, promoting the avalanche effect across rounds.

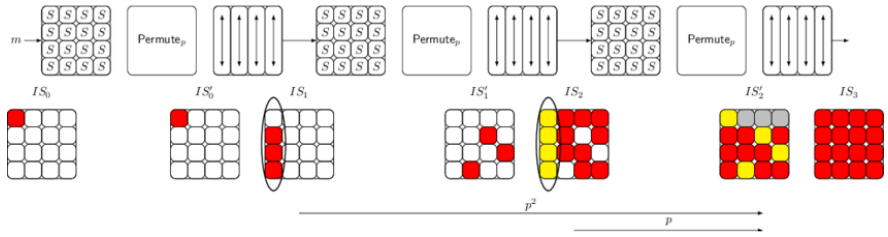


Fig. 3: Diffusion over Round Structure of AES

AES's modular structure, strong algebraic foundations, and high resistance to known cryptanalytic techniques make it the de facto standard for block cipher encryption globally. Its uniformity across key sizes allows the same core algorithm to be scaled for different security levels, making it suitable for applications ranging from embedded systems to high-throughput network security.

4.2 Overview of Cryptographic Distinguishers

A cryptographic distinguisher is an algorithm or method designed to differentiate between the output of a cryptographic primitive—such as a block cipher or hash function—and truly random data. In the context of block ciphers like AES, distinguishers attempt to **detect non-random patterns or statistical biases in the ciphertexts** produced after a certain number of rounds, thereby challenging the cipher's security claims.

The goal of a distinguisher is to identify whether a given dataset is more likely to be produced by the cipher under analysis or to be purely random. An **effective distinguisher must perform significantly better than random guessing**; that is, it must exhibit an advantage, however small, over 50% accuracy in binary classification settings.

Distinguishers are critically important because **they expose structural weaknesses or non-ideal behaviors in cryptographic designs**. While a successful distinguisher does not directly lead to key recovery, it indicates that the cipher is not behaving like a **pseudorandom permutation (PRP)** over its full round configuration, which is a fundamental requirement for secure block ciphers.

Constructing distinguishers becomes increasingly difficult as the number of cipher rounds increases. This is because additional rounds are designed to diffuse input differences, mask statistical structures, and ensure pseudorandomness. Traditional statistical methods often fail beyond a few rounds, as biases become too weak to detect. Similarly, machine learning and deep learning models have seen limited success in penetrating deeper into well-designed ciphers without requiring access to intermediate rounds, secret keys, or unrealistic assumptions.

The challenge lies in creating distinguishers that are both powerful and practical to identify meaningful patterns with minimal information and generalize across variations in keys or plaintexts. In the case of AES, the **ruggedised design and efficient diffusion mechanisms make full-round distinguishers particularly difficult to construct**, especially without access to the internal state or key. This motivates the exploration of novel approaches, such as hybrid statistical-learning techniques, to amplify weak signals and construct effective full-round distinguishers.

4.3 Distinguisher Design.

Data Generation : The dataset required for designing the distinguisher needed to have ciphertext generated by AES cipher and random data of 128 bits which will resemble the ciphertext structurally but will be complete random in distribution.

- **Random Data Generation :**The random dataset was generated using the `secrets` module from Python 3's standard library. This module was chosen specifically because it provides **cryptographically secure random number generation**, making it highly suitable for applications where unpredictability is critical. Unlike the `random` module, which relies on the Mersenne Twister and is deterministic, the `secrets` module draws entropy from the underlying operating system's secure random number generator[1]. This ensures that the output is both **non-deterministic** and **non-reproducible**, making it resistant to prediction and suitable for cryptographic applications. Using this module, a total of 2^{21} secure random values were generated in each set. In total, 1024 such sets were created. Out of these 2^{10} sets, four were

[1] Typically `/dev/urandom` on Unix-like systems (MacOS in this case).

randomly selected to ensure the inclusion of representative and unbiased samples of pseudo-random data for comparison against ciphertexts. The inherent unpredictability and security guarantees of the `secrets` module make it an ideal choice for generating robust random datasets in cryptographic research.

- **Algorithm Used :** The existing python module `PyCryptodome` was used, from where, the AES module was selected. The modeule has been verified over a period of time, however, to be absolutely sure, the encryption process was checked using the AES Encryption Validation Suite[10] standard validation vectors. Once confirmed, the module was used to generate the ciphertext.
- **Plaintext Generation :** To create ciphertext of truly random nature, it was decided that two sets of 2^{21} randomly generated plaintext will be created, one to train and validate the model, the other to reconfirm the entire process.
- **Key Generation :** The key generation was done for two different scenarios, first one key for each version $(128, 192, 256)$ encrypting the entire 2^{21} plaintexts and other scenario where one random key will be used for one plaintext, ensuring a complete random plaintext and key combination, which entailed generating 2^{21} keys of lengths 128, 192, 256 bits each. **The plaintext and keys were generated using the same code which was used to generate the random numbers for random dataset**.
- **Ciphertext Generation :** The two sets of 2^{21} plaintext were encrypted using a single key of each version length and then using 2^{21} random keys of respective lengths. Thus, there were a total of 12 ciphertext datasets, two for each version per set of plaintexts. This provided a variety of datasets for training, testing and valdation of results from the model.

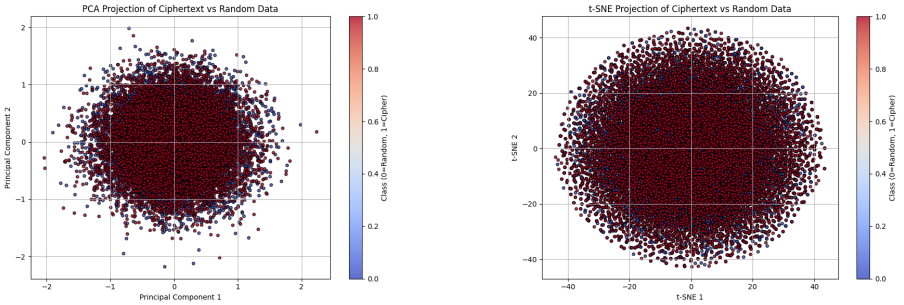
Phase I : ML/DL Based Modeling. The first choice for designing the distinguisher was ML/DL based methods, which can observe deeper patterns in the dataset and reveal the subtle differences in the types of data in both the datasets. This entailed extracting features and studying them to understand the underlying properties of the distribution. The dataset due to its larger size was sub divided into batches for processing. The batch size was decided as 2^{12} due to the constraints of RAM and processing power in a simple PC[2]. The same batch size has been used for all the tests mentioned below. The total number of batches for the given batch size were 512. The following methods and tests were conducted :-

- **Machine Learning Models :** Training of these models was selected to be supervised training. The labeled data was passed to the models for training using Support Vector Machines (SVM) and Random Forest models. The design was based around bit wise inputs of ciphertext and random data, both of 128 bits and hence the models had 128 inputs, one for each bit.

[2] iMac with 1.8 GHz Dual-Core Intel Core i5 processor and 8 GB 1600 MHz DDR3 RAM, No GPUs were used

These models failed to converge repeatedly, indicating that the data may not possess features that can be distinguished using ML methods. This meant use of DL models.

- **Deep Learning Models :** For Deep Learning the model selected was Multi Layered Perceptron (MLP) model. The inputs were designed same as the ML models, i.e. 128 inputs, one per bit of ciphertext or random data. The output was one bit, classification labels of 0 (random) and 1 (ciphertext). The dense layers(3) were made of 1024 neurons each. Loss function set as Binary Cross Entropy and Adam used as optimiser. The training was designed to be supervised. The model failed to converge after numerous epochs indicating a weak or no feature separation for the datasets of ciphertext and random data. This indicating the need for using other methods.
- **PCA and tSNE Analysis of datasets.** To understand the separability or the lack of Principal Component Analysis (PCA) and t-distributed Stochastic Neighbor Embedding (tSNE) methods of dimensionality reduction were used and plots were made, which made it clear that the data is not separable using any method used on data directly.



(a) PCA of Random Data and Ciphertext

(b) tSNE of Random Data and Ciphertext

Fig. 4: PCA and tSNE of Random Data and Ciphertext

Phase II : Statistical Tests & Analysis. To analyse the characteristics of the datasets and the distribution of data it was decided to go in for various statistical tests to determine whether the distribution is random or has some kind of structure associated with it, which will identify them as ciphertext. To do so, a total of 30 different tests were conducted, which are listed at Appendix A. The following was revealed from the tests:-

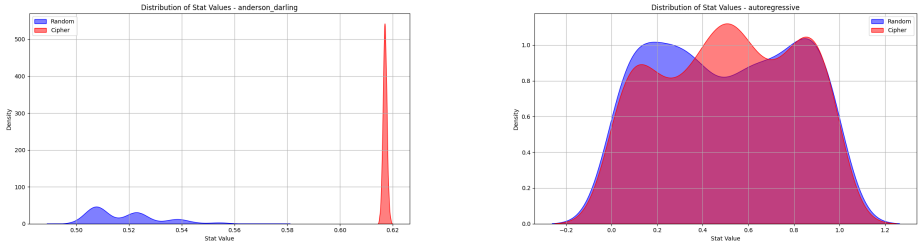
- **Randomness Detection.** These above tests were used to test whether the random behaviour of the two different datasets of ciphertext and random data differ in the test out come, thus, be distinguishable from each other. However, the results of **probability value (p value)** were found same for

both data sets. Which meant that the results for both datasets were either random or non-random, but similar to each other.

– **Deeper Analysis.** Analysis of statistical values calculated, revealed a certain underlying difference in the outcomes for ciphertext and random data. To confirm the distribution Kernel Distribution Plots were made for each test with random data and ciphertext in the same plot. **Kernel Density Estimation (KDE) Plot** is a non-parametric method to estimate the probability density function of a dataset, providing a smooth curve to visualize data distribution. The KDE at point x is given by:

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

where K is the kernel function (e.g., Gaussian), h is the bandwidth, and n is the number of data points. The sample results of two tests are given below, the others are attached as Appendix B.



(a) Anderson Darling Test : Distinguishable

(b) Autoregressive Test: Indistinguishable

Fig. 5: Statistical Values Comparison

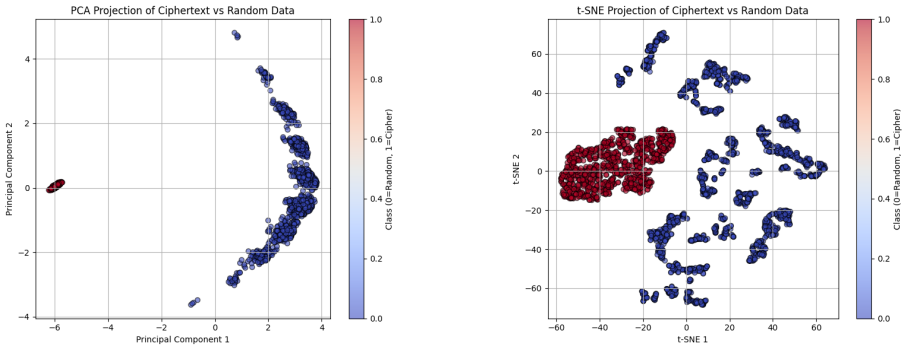
Phase III : Selective Statistical Learning. The above results clearly indicated that the tests failed in producing a distinguishable difference in the datasets as the calculated results, though different from each other when viewed at a lower scale, are in distinguishable due to minuscule value in comparison to the large size of dataset (2^{21}). It was concluded that these statistical values must be extracted and used as features to train models of ML and DL. The extraction, comparison and training was done in following steps:-

– **Feature Selection.** The statistics of the various tests were compared and checked for being similar or different using KS Two Sample Test. **Kolmogorov–Smirnov Two-Sample Test**: Tests whether two independent samples come from the same continuous distribution by comparing their empirical CDFs. The test statistic is given by:

$$D = \sup_x |F_1(x) - F_2(x)|$$

The results showed that out of 30 tests that were conducted, 22 tests showed significant to marginal differences in the statistical calculation distribution through the 512 batches, This was further filtered down to 20 tests that demonstrated a distinct difference in statistical values across batches as shown in Figure 5(a). The list of these shortlisted tests is placed at Appendix C. These selected tests were identified as input features, with each feature having 512 values, which are enough to train models. However, before starting model training it was imperative to check the separation of the features selected for classifying the datasets.

– **PCA & tSNE Plots of Features.** The PCA and tSNE plots of features revealed a distinct separation of all the selected features between random and ciphertext. The plot indicated that the statistical features, if used in training a ML or DL Model will make it feasible to classify the data with high accuracy. The plots can be seen below



(a) PCA of Features : Random vs Ciphertext

(b) tSNE of Features : Random vs Ciphertext

Fig. 6: PCA & tSNE of Features from Random Data & Ciphertext

5 Results

Support Vector Machine (SVM) was the choice of Model selected. Te training of model was carried out using one dataset of ciphertext and three datasets of random data. This ensured that there existed at least one dataset of ciphertext and random data each that was unseen by the model and would present a high level of validation for the model.

5.1 Model Metrics

The model was trained to a very high accuracy, as expected. The model was tested and validated using he data mentioned above and the details of F1, Precision and recall are given below:-

Class	Precision	Recall	F1-Score	Support
Random	1.0000	1.0000	1.0000	512
Cipher	1.0000	1.0000	1.0000	512
Accuracy			1.0000	1024
Macro Avg	1.0000	1.0000	1.0000	1024
Weighted Avg	1.0000	1.0000	1.0000	1024

Table 1: SVM Classification Metrics

Feature	Coeff	Feature	Coeff	Feature	Coeff
Anderson-Darling	-0.0200	KS Test	-0.0200	Shapiro-Wilk	-0.0871
Approx. Entropy	-0.0798	Lilliefors	-0.0204	T-Test	-0.1374
Chi-Square Dist.	0.1016	Monobit	-0.0014	Hamming Weight	-0.0553
Chi-Square Unif.	0.1016	Runs Test (Blocks)	0.1497	Frequency (Block)	-0.0869
Cumulative Sums	0.0171	Serial	0.0871	Autocorrelation	-0.0109
Entropy	-0.0866	Jarque-Bera	0.1842	Mann-Whitney	0.2026
F-Test	0.1313	Gap Test	0.1280		

Table 2: SVM Feature Coefficients

5.2 Multi Model Validation

The results with 100% accuracy across training, validation and test did raise questions about overfitting. To confirm the same fresh data was created for ciphertext and random data was selected from already generated 2^{10} sets to get a completely new perspective. Further, the models used for classification were also changed. The new methods of classification used were Random Forest, Naive Bayes, Logistic Regression and Multi Layer Perceptron. The results achieved for the above tests are given below:-

Class	Precision	Recall	F1-Score	Support
Random	1.0000	1.0000	1.0000	512
Cipher	1.0000	1.0000	1.0000	512
Accuracy			1.0000	1024
Macro Avg	1.0000	1.0000	1.0000	1024
Weighted Avg	1.0000	1.0000	1.0000	1024

Table 3: Random Forest Classification Report

Feature	**Imp**	**Feature**	**Imp**	**Feature**	**Imp**
Anderson-Darling	0.070	KS Test	0.060	Shapiro-Wilk	0.070
Approx. Entropy	0.080	Lilliefors	0.050	T-Test	0.090
Chi-Square Dist.	0.030	Monobit	0.040	Hamming Weight	0.000
Chi-Square Unif.	0.050	Runs Test (Blocks)	0.040	Frequency (Block)	0.000
Cumulative Sums	0.050	Serial	0.120	Autocorrelation	0.000
Entropy	0.070	Jarque-Bera	0.070	Mann-Whitney	0.000
F-Test	0.060	Gap Test	0.050		

Table 4: Random Forest Feature Importance

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Random	1.0000	1.0000	1.0000	512
Cipher	1.0000	1.0000	1.0000	512
Accuracy			1.0000	1024
Macro Avg	1.0000	1.0000	1.0000	1024
Weighted Avg	1.0000	1.0000	1.0000	1024

Table 5: Logistic Regression Classification Report

Feature	**Coeff**	**Feature**	**Coeff**	**Feature**	**Coeff**
Anderson-Darling	0.3327	KS Test	0.3327	Shapiro-Wilk	-0.4412
Approx. Entropy	-0.4315	Lilliefors	0.3323	T-Test	-0.4477
Chi-Square Dist.	0.4551	Monobit	0.3518	Hamming Weight	-0.2696
Chi-Square Unif.	0.4551	Runs Test (Blocks)	0.6181	Frequency (Block)	0.1079
Cumulative Sums	0.3709	Serial	0.4382	Autocorrelation	0.2001
Entropy	-0.4408	Jarque-Bera	0.5323	Mann-Whitney	0.4544
F-Test	0.4358	Gap Test	0.4461		

Table 6: Logistic Regression Feature Coefficients

Class	Precision	Recall	F1-Score	Support
Random	1.0000	1.0000	1.0000	512
Cipher	1.0000	1.0000	1.0000	512
Accuracy			1.0000	1024
Macro Avg	1.0000	1.0000	1.0000	1024
Weighted Avg	1.0000	1.0000	1.0000	1024

Table 7: Naive Bayes Classification Report

Class	Precision	Recall	F1-Score	Support
Random	1.0000	1.0000	1.0000	512
Cipher	1.0000	1.0000	1.0000	512
Accuracy			1.0000	1024
Macro Avg	1.0000	1.0000	1.0000	1024
Weighted Avg	1.0000	1.0000	1.0000	1024

Table 8: MLP Classification Report

As seen from the above results the selection of statistical values from the datasets proved to be an excellent base for modeling ML and DL models for classification. However, to verify the same further, it was decided to carry out K-Fold Cross Validation.

Note on Accuracy: *While the classification results reported in this work consistently achieve 100% accuracy across models and test sets, heir interpretation should be approached with objectivity. Such high accuracy is unusual in practical cryptanalysis and is likely a consequence of the carefully selected and well-engineered statistical features that capture subtle, persistent biases in ciphertext distributions. Extensive validation through cross-validation, unseen datasets, and alternative classifiers was conducted to rule out overfitting or data leakage.*

5.3 K-Fold Cross Validation.

K-Fold Cross Validation was carried out to ascertain the validity of the models and the classification carried out for the given multiple data sets. K-Fold validation with $K = 5$ and $K = 10$ was carried out and the results are given below:-

Model	Precision	Recall	F1 Score
SVM (CV)	1.0000	1.0000	1.0000
Random Forest (CV)	1.0000	1.0000	1.0000
Logistic Regression (CV)	1.0000	1.0000	1.0000
Naive Bayes (CV)	1.0000	1.0000	1.0000
MLP (CV)	1.0000	1.0000	1.0000
SVM (Test)	1.0000	1.0000	1.0000
Random Forest (Test)	1.0000	1.0000	1.0000
Logistic Regression (Test)	1.0000	1.0000	1.0000
Naive Bayes (Test)	1.0000	1.0000	1.0000
MLP (Test)	1.0000	1.0000	1.0000

Table 9: Precision, Recall, and F1 Scores for Cross-Validation and Test Set Evaluation

This proves beyond any doubt that the Model training is fair and the results can be generalised across all ciphertext generated using full rounds of AES for all three key lengths and respective rounds.

6 Discussion

The results obtained above have always classified the ciphertext and random data with an accuracy of 100%. Though this result seems *too good to be true*, it may be noted that multiple tests have been carried out to verify the authenticity of tests. This method does not prove the efficacy of any model, but emphasises the feature selection for the models. The test selection and the identification of intermediate values from the selected tests for features is the key to such high success of this Hybrid Statistical Learning method. This method only goes ahead and prove that the most important part about ML or DL based approaches to distinguisher tasks will succeed if the features are selected carefully and the method needs to multi-staged combination of various different techniques. This model and tests displayed here are successful for AES, however, the tests and the models for different cipher may differ, but this Hybrid Statistical Learning method can be surely applied to other ciphers and will surely make distinguishing tasks more efficient. To test this particular statement a test was carried out for the legacy DES and lightweight cipher PRESENT, used in IoTs. The distribution of the statistical values in form of KDE plot is given at Figure 7. This conclusively proves that **FESLA can be easily applied to most of the ciphers**, provided they exhibit similar subtle statistical signatures as AES. These signatures that have been hidden till now, have come forth via FESLA. It must be noted that this model does not restrict itself to just 30 tests but can incorporate up to 50+ tests, which have been carefully selected.

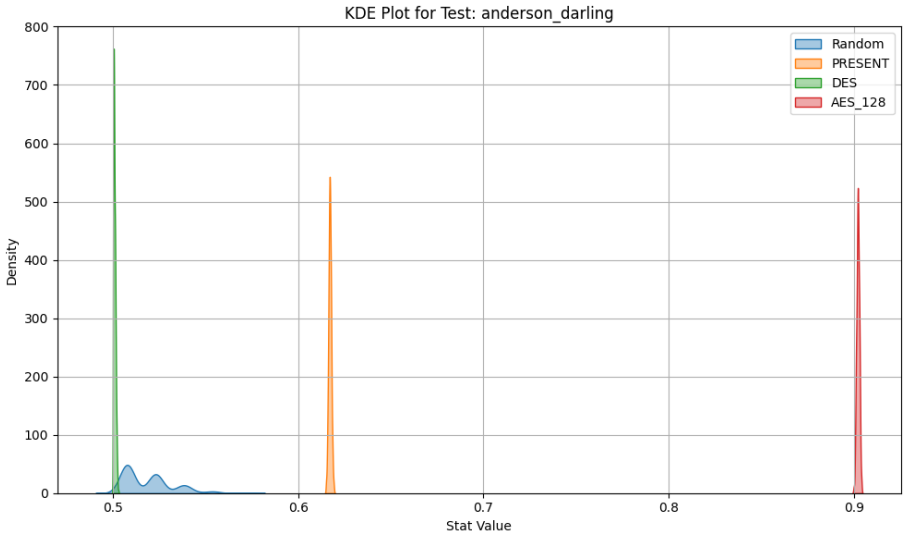


Fig. 7: Statistical Values of Different Ciphers

7 Conclusion

This work presented a novel hybrid statistical learning approach, FESLA, for constructing a full-round distinguisher for AES. By aggregating intermediate results from multiple statistical tests and using them as features for machine learning and deep learning classifiers, we demonstrated a highly effective method to distinguish AES ciphertext from truly random data. Unlike traditional cryptanalytic or ML/DL-only techniques, this method achieves perfect classification accuracy on the full round cipher, indicating that subtle statistical structures persist even at maximum round depth.

The consistent success of the classifier across multiple models and datasets underscores the critical role of feature selection, emphasizing that meaningful distinguishers can still be derived from well-engineered statistical characteristics. These findings open up promising avenues for further research, particularly the extension of FESLA to other ciphers, as seen in distinguishing DES and PRESENT above.

Future work will aim at applying this method with a battery of statistical tests to various ciphers including asymmetric ciphers, stream ciphers and various other options. This method will target to distinguish full round ciphers, like shown in this paper and extend this result to a practical model that can be applied to practical real time environments.

Appendix A - List of Statistical Tests Conducted

The following tests were conducted to check randomness in ciphertext dataset and random data dataset.

- **Monobit Test**
  Checks if the number of 1s and 0s in a bitstream are approximately equal.
  Statistic: $S = \frac{|\#\text{ones} - \#\text{zeros}|}{\sqrt{n}}$
  P-value: $\text{erfc}\left(\frac{S}{\sqrt{2}}\right)$
- **Frequency Within Block Test**
  Divides the sequence into M-bit blocks and checks the balance of 1s and 0s per block.
  Statistic: $\chi^2 = 4M \sum_{i=1}^{N} \left(\pi_i - \frac{1}{2}\right)^2$
  P-value: Based on chi-square distribution with N degrees of freedom.
- **Runs Test**
  Tests whether the number of runs of 1s and 0s is as expected.
  Statistic: $Z = \frac{|V_n - 2n\pi(1-\pi)|}{2\sqrt{2n}\pi(1-\pi)}$
  P-value: $\text{erfc}\left(\frac{Z}{\sqrt{2}}\right)$
- **Longest Run of Ones Test**
  Analyzes the longest run of 1s in fixed-size blocks.
  Statistic: Chi-square based on observed vs expected run lengths.
  P-value: From chi-square distribution.
- **Serial Test**
  Checks uniformity of overlapping m-bit patterns.
  Statistic: $\chi^2 = \sum_i \frac{(f_i - E)^2}{E}$
  P-value: From chi-square distribution.
- **Approximate Entropy Test**
  Measures frequency of overlapping m-bit patterns to detect regularity.
  Statistic: $\text{ApEn} = \phi(m) - \phi(m+1)$
  P-value: $\text{erfc}\left(\frac{\text{ApEn}}{\sqrt{2}}\right)$
- **Universal Statistical Test**
  Detects compressibility of sequences; random sequences are less compressible.
  Statistic: Based on Lempel-Ziv complexity.
  P-value: Compared against known theoretical mean and variance.
- **Linear Complexity Test**
  Evaluates whether the sequence can be generated by a linear feedback shift register (LFSR).
  Statistic: $T = (-1)^k(L - \mu) + \frac{2}{9}$
  P-value: Derived from normal distribution.
- **Cumulative Sums Test**
  Analyzes the cumulative sum of bits from the mean.
  Statistic: $S_n = \max|\sum_{i=1}^{n} X_i|$, where $X_i \in \{-1, +1\}$
  P-value: Based on a normal approximation.

- **Discrete Fourier Transform (DFT) Test**
  Detects periodic features by analyzing the frequency spectrum.
  Statistic: Number of peaks below a threshold.
  P-value: From normal approximation.
- **Rank Test**
  Measures rank of sub-matrices to check linear dependency.
  Statistic: Chi-square based on observed vs expected matrix ranks.
  P-value: From chi-square distribution.
- **Entropy Test**
  Computes Shannon entropy; higher entropy indicates randomness.
  Formula: $H = -\sum p(x) \log_2 p(x)$
  P-value: From empirical distribution or simulations.
- **Chi-Square Uniformity Test**
  Checks uniformity in frequency distribution.
  Statistic: $\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$
  P-value: From chi-square distribution.
- **Autocorrelation Test**
  Measures similarity between sequence and shifted version.
  Statistic: $r_k = \frac{1}{n-k} \sum_{i=1}^{n-k} x_i x_{i+k}$
  P-value: Based on normal distribution.
- **Jarque-Bera Test**
  Tests for normality using skewness and kurtosis.
  Statistic: $JB = \frac{n}{6}(S^2 + \frac{(K-3)^2}{4})$
  P-value: From chi-square distribution with 2 degrees of freedom.
- **Shapiro-Wilk Test**
  Assesses normality based on order statistics.
  Statistic: $W = \frac{(\sum a_i x_{(i)})^2}{\sum (x_i - \bar{x})^2}$
  P-value: From tabulated reference values.
- **Kolmogorov-Smirnov (KS) Test**
  Compares empirical with theoretical cumulative distribution.
  Statistic: $D = \sup_x |F_n(x) - F(x)|$
  P-value: From KS distribution.
- **Lilliefors Test**
  Modified KS test for unknown mean and variance.
  Statistic: Same as KS.
  P-value: From empirical distributions.
- **Spearman Rank Correlation Test**
  Tests for monotonic relationships.
  Statistic: $\rho = 1 - \frac{6 \sum d_i^2}{n(n^2-1)}$
  P-value: From t-distribution.
- **Mann-Whitney U Test**
  Non-parametric comparison of two independent distributions.
  Statistic: $U = \min(U_1, U_2)$
  P-value: Based on normal approximation.

- **T-Test**
  Checks whether the means of two samples differ significantly.
  Statistic: $t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$
  P-value: From t-distribution.
- **Anderson-Darling Test**
  Emphasizes tail differences to test for normality.
  Statistic:
  $A^2 = -n - \frac{1}{n} \sum_{i=1}^{n} (2i-1) \left[\ln F(x_i) + \ln(1 - F(x_{n+1-i})) \right]$
  P-value: From empirical approximation.
- **F-Test**
  Compares the variances of two samples.
  Statistic: $F = \frac{s_1^2}{s_2^2}$
  P-value: From F-distribution.
- **Random Runs Test**
  Counts runs of increasing/decreasing values.
  Statistic: Based on number of runs relative to expected.
  P-value: From normal approximation.
- **Autoregressive Test**
  Tests for linear dependence on prior values.
  Model: $X_t = \alpha X_{t-1} + \epsilon_t$
  P-value: Based on regression t-statistic.
- **Peak Test**
  Counts how many transformed values exceed a set threshold.
  Statistic: Count of peaks.
  P-value: From normal approximation.
- **Hamming Weight Test**
  Measures the number of 1s per block.
  Statistic: $HW = \sum x_i$
  P-value: From binomial or normal distribution.
- **Gap Test**
  Measures spacing between repeated occurrences (e.g., of 1s).
  Statistic: Histogram of gaps compared to expected.
  P-value: From geometric or empirical distribution.
- **Chi-Square Distribution Test**
  General goodness-of-fit test.
  Statistic: $\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$
  P-value: From chi-square distribution.
- **Runs Test with Blocks**
  Applies the runs test to aggregated data blocks.
  Statistic: Modified chi-square based on grouped runs.
  P-value: From chi-square or normal distribution.

Appendix B - Comparison of Statistical Values

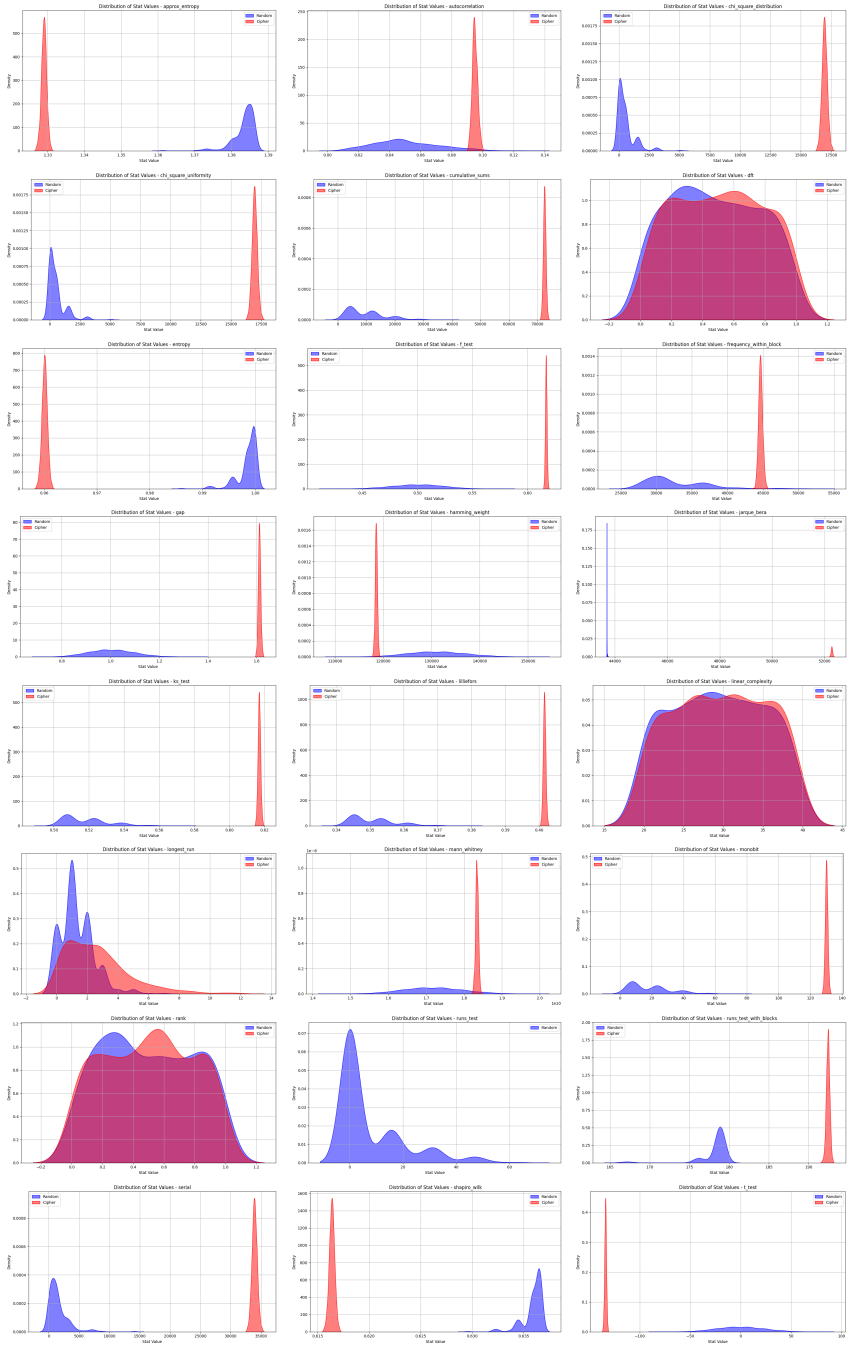


Fig. 8: Comparison of Statistical Values for Ciphertext and Random Data

Appendix C - Shortlisted Statistical Tests

1. Anderson–Darling Test
2. Approximate Entropy Test
3. Chi-Square Goodness-of-Fit Test (for distribution fit)
4. Chi-Square Uniformity Test (for uniformity of bits)
5. Cumulative Sums (Cusum) Test
6. Shannon Entropy Calculation
7. F-Test for Equality of Variances
8. Gap Test (for gaps between occurrences of a symbol)
9. Jarque–Bera Test (for normality)
10. Kolmogorov–Smirnov Test
11. Lilliefors Test (variant of KS for normality)
12. Monobit (Frequency) Test
13. Runs Test with Blocked Bit Groups
14. Serial Test (for pattern frequency)
15. Shapiro–Wilk Test (for normality)
16. Student's t-Test (for means)
17. Hamming Weight Test
18. Frequency Test within a Block
19. Autocorrelation Test
20. Mann–Whitney U Test (non-parametric test for medians)

References

1. J. Daemen and V. Rijmen, *The Design of Rijndael: AES—The Advanced Encryption Standard*, Springer, 2002.
2. N. Ferguson, R. Schroeppel, and D. Whiting, "Improved Cryptanalysis of Rijndael," in *Fast Software Encryption*, Springer, 2000, pp. 213–230.
3. H. Gilbert and M. Minier, "A Collision Attack on AES," in *International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, 2000, pp. 230–241.
4. M. Matsui, "Linear Cryptanalysis Method for DES Cipher," in *Workshop on the Theory and Application of Cryptographic Techniques*, Springer, 1993, pp. 386–397.
5. A. Khovratovich and I. Nikolić, "Rotational Cryptanalysis of ARX," in *International Conference on Fast Software Encryption*, Springer, 2010, pp. 333–346.
6. A. Rukhin *et al.*, *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*, NIST Special Publication 800-22, 2001.
7. U. M. Maurer, "A Universal Statistical Test for Random Bit Generators," *Journal of Cryptology*, vol. 5, no. 2, pp. 89–105, 1992.
8. A. Gohr, "Deep Learning for Distinguishing Ciphertexts from Random," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2019, pp. 75–102.
9. M. Moini *et al.*, "Deep Learning-Based Output Prediction Attacks on Block Ciphers," *IACR Cryptology ePrint Archive*, 2023, Paper 2023/1101.
10. L. E. Bassham III, *The Advanced Encryption Standard Algorithm Validation Suite (AESAVS)*, National Institute of Standards and Technology (NIST), Information Technology Laboratory, Computer Security Division, November 15, 2002.