https://doi.org/10.65542/djei. v2i2.39
Dasinya Journal for Engineering and Informatics. **2026**, 2, 17.

Article

AI-Enhanced AES Encryption for Kurdish Unicode Texts: A Neural Network–Based Key Generation Approach Using Linguistic Statistical Features

Ziyad Hazim Abduljabbar [1,*]

[1] Information Technology Department, Technical College of Duhok, Duhok Polytechnic University, Duhok, Kurdistan Region, Iraq.

* Correspondence: ziyad.hazim@dpu.edu.krd

Abstract

This research presents an efficient and secure method for generating the AES master key, based on statistical features extracted from the linguistic structure of Middle Kurdish texts. These features include text length, word count, frequency of unique letters, bigram and trigram, and text entropy. The numerical feature vector is fed with a random salt value into a secure hashing algorithm (SHA256) to scale and encode it into a 32-bit intermediate key. This intermediate key is processed by a three-layer FFNN with random weights and bias values to output the AES master key. For optimal performance and security, the AES algorithm with a 256-bit key and the GCM operating mode were used. The encryption system was developed using Python. The initial test was performed on ten Kurdish texts. The measured entropy values for all generated master keys were high, approaching the maximum Central Kurdish alphabet entropy of 4.9542 bits/letter. To evaluate the system on a larger number of texts, a dataset named CKLTCD was created, consisting of 3000 Kurdish texts of varying lengths and domains. Keys were generated for all dataset texts. AES encryption and decryption were applied, yielding decrypted texts identical to the originals. The SHA function and FFNN significantly complicated and obscured the relationship between the original text and the generated key. The generated key became more independent and complex, making its analysis and prediction extremely difficult. The test results of cryptographic validation (NIST SP800-22, avalanche effect analysis, correlation analysis, and key sensitivity tests) and attack model evaluation (KPA, CPA, and brute-force attacks) reflected the success of the proposed encryption system's strength and security. Comparisons showed the method's similarity to standard functions (PBKDF2, Argon2, HKDF), validating this alternative method for generating dynamic keys based on Kurdish linguistic characteristics.

Keywords: Attack Model Evaluation; Central Kurdish Text; CKLTCD Dataset; FFNN; Feature Extraction; Key Generation.

1. Introduction

E-governance requires secure data transmission tools, especially for resource-limited languages like Kurdish, the focus of this research. The Advanced Encryption Standard (AES) algorithm is widely used for its high encryption efficiency, but key derivation functions still require significant resources like memory, processing time, and cloud

Received: January 24, 2026; Revised: April 24, 2026; Accepted: April 24, 2026; Published: May 08, 2026; Available online: May 08, 2026.

computing. Furthermore, given the continuous development of attackers' ability to launch brute-force attacks on master key prediction processes, it has become essential to consider new methods for generating a dynamic, unpredictable key. This research proposes a novel approach to master key generation based on the statistical features of natural language texts written in Central Kurdish. Several Kurdish language features were considered, such as text length, word count, letter count, bigram and trigram count, and text entropy. This means the goal of the search is to use the features of the text to generate the master key for encrypting the text itself. Statistical features of the text will be extracted and calculated, and a feature vector will be created, which acts as a unique digital signature for the text. Then, to conceal the relationship between the values of the feature vector and the text itself, the feature vector is passed through two stages of complexity. In the first stage, it enters the Secure Hash Algorithm (SHA256), a one-way hashing and encoding function whose purpose is to transform the feature vector into a unique 256-bit value that represents the intermediate key. In the second stage, this intermediate key is passed to a three-layer Feed-Forward Neural Network (FFNN) designed to generate a highly random final key. The relationship between the key and the text will become a complex differential relationship, making it difficult for an attacker to calculate and predict the master key. The encryption process will then begin based on the core of the AES algorithm standard, with reliance on the generated master key of length 256 bits, with the adoption of Galois/Counter Mode (GCM) operating mode to ensure the highest level of efficiency and security.

2. Related Works

The Advanced Encryption Standard (AES) is one of the most secure and widely used encryption algorithms in many information security systems [1][2][3]. Its security features and high reliability make it a fundamental component in many applications. However, despite its strength, most problems that arise in encryption systems do not stem from encrypting the data content [4] but rather from key generation and access management [5][6]. A major drawback of password-based systems is their ease of reuse, as users often forget their passwords [7][8],. The National Institute of Standards and Technology (NIST) guidelines recommend that encryption key generation, especially for sensitive applications, should use secure random or pseudo-random generation/encryption functions with sufficient entropy to resist brute-force and hacking attacks [6]. Numerous alternative key generation methods exist. Biometric fingerprinting and iris-based encryption key generation have been discussed in another paper, and there are also studies on generating cryptographic keys using DNA, voice, and image [9]. Although these factors are unique to each user, they can be easily copied or forged, requiring additional hardware [9]. Therefore, they are unsuitable for application in textual encryption systems. However, statistical features of text have recently become important in security applications. Researchers in computational linguistics and data mining have confirmed that common text properties, such as text length, word count, entropy, and character/binary/trilogy repetition, can be analyzed to better understand the text and applied in the field of security [10][11][12][13]. Hashing functions such as SHA have been found to produce fixed-length keys with high homogeneity with respect to the inputs[14][15][16][17]. Other studies have also attempted to apply neural networks to cryptography. Neural networks have been used to generate dynamic or pseudo-random sequences that can be used as encryption keys, including keys based on the Advanced Encryption Standard (AES) [18]. Other studies demonstrate the feasibility of using neural networks, particularly adversarial generative networks (GANs), for generating strong keys for secure communication systems [19][20][21,22].

Research on Kurdish text encryption has been limited. Some researchers have attempted to extract Kurdish language patterns for deep learning without using any encryption [23]. Others have combined the Advanced Encryption Standard (AES) with a one-time encryption key, but without any artificial intelligence techniques [24]. Similarly, some have attempted to find mathematical models for encrypting central Kurdish texts based on the Unicode system, but without applying AI-based key generation techniques, relying instead on traditional methods [25].

Regarding traditional standard key generation mechanisms, key generation functions such as PBKDF2, Argon2, and HKDF have been extensively studied in the scientific literature. PBKDF2 operates by increasing the expansion cost through its iterative nature before key generation. Thus, it mitigates brute-force attacks, but at the same time makes it more susceptible to longer processing loads [26][27]. Argon2 outperforms others due to its inclusion of additional functions that require more memory resources, making it more difficult to execute parallel attacks on GPUs [28]. In contrast, HKDF is a lightweight extraction and expansion function based on hashing functions. While it has proven highly efficient, it lacks sufficient security to withstand password-guessing attacks on its own [15]

This study proposed an alternative key generation method. This method uses dynamic key generation based on the core features of Kurdish texts. It uses random salt values with the hash function SHA and takes advantage of the FFNN network's complexity by using random parameter values (weights and bias). This makes a 256-bit key that serves as the master key for the AES encryption algorithm, which works in CGM mode. This mode provides efficiency in security, encryption, and reliability.

3. Research Contribution

This research represents a valuable contribution to the field of encryption key generation. The proposed method relies on extracting a statistical features vector from the Kurdish text to be encrypted in order to generate a strong dynamic key. The SHA function and FFNN neural network are used to increase the complexity of the key by concealing the relationship between the text and its generated key. This study aims to contribute to the field of cryptographic key generation by introducing a novel approach for dynamic key generation. The proposed method is presented as an additional alternative to standard key derivation functions (PBKDF2, Argon2, and HKDF). In addition to reducing reliance on traditional password-based encryption mechanisms. The strength and security of the proposed encryption system will be based on the efficiency of the AES algorithm and the strength of the master key generated from the features of the Kurdish text itself to be encrypted.

4. AES-GCM-256 Algorithm: An overview

The AES was adopted by the National Institute of Standards and Technology (NIST) in 2001. The AES is a symmetric encryption algorithm that provides strong performance and security against linear and differential analytical attacks due to its robust architecture [1][2][29]. The core of the AES algorithm consists of a series of calculations and switching operations—SubBytes, ShiftRows, MixColumns, and AddRoundKey—applied repeatedly over several rounds depending on the key size (10 rounds for a 128-bit key, 12 rounds for a 192-bit key, and 14 rounds for a 256-bit key). The purpose of these rounds is to increase the scramble and confusion of the ciphertext, making it impossible to reverse without knowing the master key.

Figure 1 illustrates the general standard steps for the process of encryption and decryption of the AES_GCM256 algorithm core.

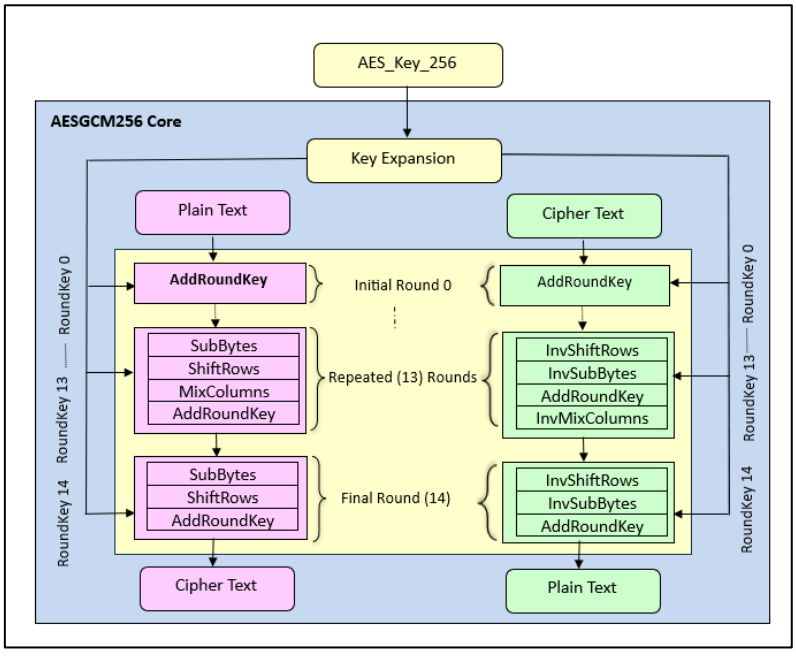

Figure 1. Encryption and decryption steps for AES_GCM256 core.

This algorithm can be implemented in several operating modes, the most important of which are ECB, CBC, CTR, and GCM. The GCM operating mode provides high encryption speed by adopting the parallelizable counter mode (CTR) for encrypting 128-bit stream blocks. This means that the encryption of the current block does not depend on the

encryption result of the previous block [30]. Using the GCM mode ensures the accuracy of received data based on a Galois Hash function (GHASH) that relies on additional authentication data (AAD). Furthermore, employing a 256-bit key ensures the application of the highest number of rounds (=14), which in turn guarantees the highest number of shuffling operations on the data block streams, thus achieving the highest level of confidentiality, integrity, and security against attackers [31][3][32][33][34].

The text to be encrypted is entered. A 256-bit encryption key is generated using the Key Derivation Function (KDF) with a salt value to enhance the key's security. The text is divided into fixed blocks of 128 bits. The expansion function generates the corresponding 128-bit round key. An Initialization Vector (IV) (or Number Used Once (Nonce)) is generated to ensure different encryption for similar text segments. The corresponding counter block (CTRi) is encrypted using the round key block, and the resulting block is used to encrypt the text block (Pi) and generate the corresponding ciphertext block (Ci). At the same time, the authentication Tag (T) is computed from the GHASH process, which links together the integrity of all the (AAD) and (C) blocks [30]. Figure 2 shows the inputs (plain text, Key) and outputs (Cipher Text, Tag) of the AES-GCM-256 algorithm, based on the traditional method of generating the master key. The general equations of the AES-GCM-256 algorithm can be illustrated as follows:

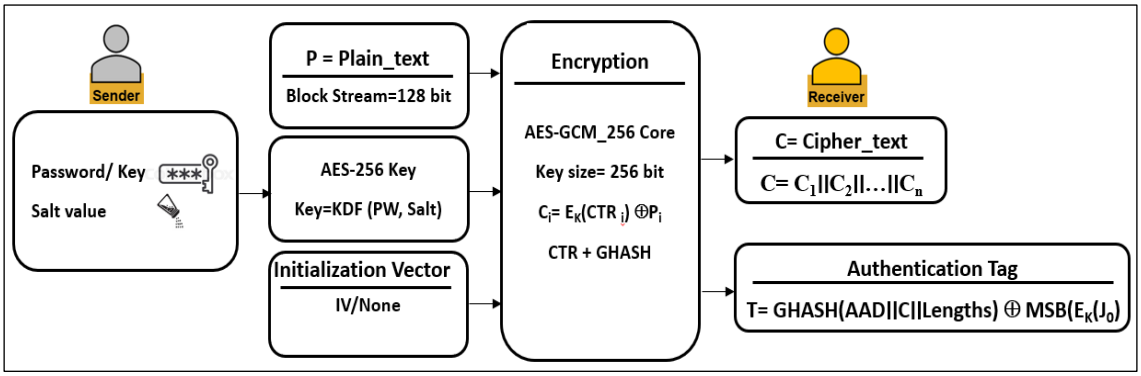

Figure 2. AES-GCM-256 algorithm based on the normal method of generating the master key.

1- Inputs and initialization:
- Plain Text (P)
- AAD
- Password (Pw), and Salt value.
- IV/Nonce.
- J_0=IV; where J_0= Initial counter block derived from IV

2- Key and hash key Formation:

$$K = KDF(Pw, Salt) \quad (1)$$

$$H = E_k(0^{128}) \quad (2)$$

Where: KDF: Key Derivation Function, K= 256-bit encryption key, H=128-bit hash key.

3- Encryption (CTR Mode)

$$C_i = E_K(\text{Counter Stream}) \oplus P_i;$$

$$C_i = E_K(J_0+i) \oplus P_i \quad (3)$$

Where:
- EK: AES-256 block encryption function using key K,
- C_i, P_i = block number (i) for cipher and plain text.

$$\text{The complete cipher text (C)= C1||C2||}\ldots\text{||Cn} \quad (4)$$

4- Generating the Authentication Tag

$$S = GHASH_H(\text{AAD} \,||\, \text{C} \,||\, \text{Lengths}) \quad (5)$$

$$T = S \oplus E_K(J_0) \quad (6)$$

Where:
- T: Authentication Tag, GHASH = Galois Hash function based on hash key (H).
- Lengths: Len (AAD) || Len (C); Block of size 128 bits containing the lengths of (AAD) and (C)
- $E_K(J_0)$: Represents the encryption function of the Initial Counter Block J_0.

5. Methodology of the Proposed Technique

5.1. Features of the Central Kurdish language:

After studying and analyzing several specialized studies on the structure and orthography of the Kurdish language, it became clear that the Kurdish language has several characteristics that support relying on its textual features, the most important of which are:

5.1.1. The Kurdish language has a large alphabet (31 letters), compared to Arabic (28 letters) and English (24 letters). Accordingly, the Kurdish alphabet will have a high probability distribution, meaning that the generated keys will have a high entropy value and high randomness [35][36].

5.1.2. The Kurdish language is characterized by its frequent use of vowels, such as (ه، ی، و), which affects the statistical distribution of the Kurdish text and gives it a distinct statistical distribution [37].

5.1.3. The Kurdish language is characterized by its numerous suffixes, such as (ـەکان, ـان), which make its words longer compared to the length of Arabic and English words. Furthermore, these suffixes can be relied upon to form bigrams and trigrams of letters, which are considered one of the statistical features of the text [35][38].

It is important to mention an issue with some letters of the Kurdish alphabet that share the same appearance, despite being different Unicode letters, such as (ک for ك) and (ی for ي), when fairly non-standard keyboards are being used. This is important to consider when normalization of the text is undertaken in the preprocessing stage [39].

5.2. Steps for generating an AES algorithm key based on Kurdish text features

The original plaintext Kurdish text to be encrypted is fed into two pathways: the first involves inputting this plaintext into the core of the AES GCM encryption algorithm. The second pathway, which is the focus of this proposed research, involves inputting a copy of the plaintext into the following steps, see Figure 3.

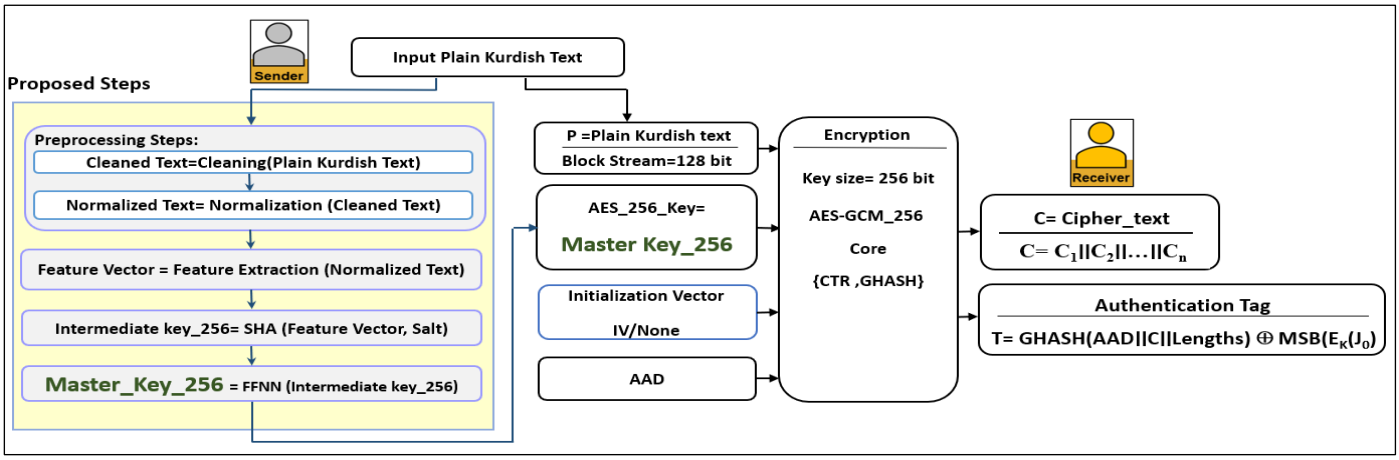

Figure 3. Proposed steps for generating the master key for the AESGCM-256 algorithm.

5.2.1. Preprocessing steps:

The Kurdish text is prepared through two preprocessing steps before feature extraction begins: text cleaning and normalization. The text cleaning steps remove all non-literal characters such as numbers, punctuation marks, and other symbols. The normalization step unifies visually similar characters that differ in their Unicode representation. These processes result in a uniform and homogeneous text, facilitating accurate statistical calculations that can be reliably used to generate reliable keys.

$$T_c = Text_{Clean} = Clean\ (Text_{plain}) \quad (7)$$

$$T_n = Text_{Normalize} = Normalize\ (T_c) \quad (8)$$

5.2.2. Feature Extraction:

Based on the normalized text, the text's statistical features are extracted. For each input text, a text feature vector is calculated. This vector includes text length, word count, frequency distribution (letters, bigrams, and trigrams), and entropy. The equations for the feature extraction can be illustrated as follows:

$$Feature\ Vector\ (FV) = Feature\_Extract\ (Text\_Normalized) \quad (9)$$

FV= [L, W, Fchr, Fbig, Ftri, Top10chr, Top10big, Top10tri, H] (10)

Where: L: The number of letters (Nch) in the normalized text excluding white spaces

$$L = \sum_{i=1}^{|T_n|} f(T_{n_i}) \ \ldots . where\ f(T_{n_i}) = \begin{cases} 1 \ \ldots \ \ if\ (T_{n_i}) \neq\ space \\ 0 \ \ldots \ \ if\ (T_{n_i}) =\ space \end{cases} \quad (11)$$

Where: -W: The number of words (Nw) in the normalized text separated by space(s)

$$W = \sum_{i=1}^{|T_n|} g(T_{n_i}, T_{n_{i-1}}) \ \ldots . where\ g(T_{n_i}) = \begin{cases} 1 \ \ldots \ \ if\ (T_{n_i}) \neq\ space\ \ and\ (T_{n_{i-1}}) =\ space \\ 0 \ \ldots \ \ otherwise \end{cases} \quad (12)$$

Where: -Fchr = The frequency of distinct (unique) characters present in the normalized text.

$$F_{chr} = \sum_{k=1}^{L} Match(T_n[k], C_i) \quad (13)$$

Where: -Fbig = Frequency of distinct (unique and non-repeated) bigrams present in the text.

$$F_{big} = \sum_{k=1}^{L-1} Match\big(T_n[k{:}\,k+1], B_{ij}\big) \quad (14)$$

Where: -Ftri = Frequency of distinct (unique and non-repeated) trigrams present in the text.

$$F_{tri} = \sum_{k=1}^{L-2} Match\big(T_n[k{:}\,k+2], R_{ijk}\big) \quad (15)$$

Where: -Top10chr, Top10big, and Top10tri are the lists of the top ten frequencies for letters, bigrams, and trigrams, respectively.

-H ($Text_{Normalized}$)= the Entropy of the normalized text (bits per character).

Here, it should be mentioned that the optimal entropy value of the central Kurdish alphabet will be used to evaluate the entropy for both input Kurdish texts and generated master keys. This optimal entropy value is actually based on the central Kurdish alphabet (31 letters) and is calculated as follows:

$$H_{Max}(\mathrm{A}) = log_2(31) = \frac{log(31)}{log(2)} \approx 4.9542\ bit/letter$$

As for the entropy of the normalized text, it is calculated as follows:

$$\therefore \mathrm{H}(Text_{Normalized}) = -\sum_{i=1}^{n} P_i Log_2(P_i) \quad (16)$$

Where: - n represents the number of distinct letters in the normalized text.

- Pi: the probability of occurrence of letter I, which will be calculated as:

$$P_i = \frac{f_i}{N} \quad (17)$$

Where: - fi: the frequency of the letter i in the normalized text.

- N: Total number of letters in the normalized text.

5.2.3. Generating the Intermediate Key:

The feature vector created in the previous step, along with a random salt value, is added to the SHA256 function to generate the intermediate key that is compatible with the 32-byte (256-bit) input layer of the FFNN network. SHA256 is used as a one-way encoding function that converts the statistical features of the Kurdish text into a 32-byte intermediate key. The SHA256 function constitutes a first security and complexity layer. Applying the SHA256 function ensures that the Kurdish characters included in the feature vector are encoded into a one-byte ASCII code, independent of the two-byte Kurdish Unicode characters from which they were extracted. This adds complexity that ensures the relationship between the key and the original text is difficult to predict.

$$\text{Intermediate\_Key} = \text{SHA256}(\text{encode}(\text{FV} \,||\, \text{salt})) \quad (18)$$

Where:

- FV = feature vector;
- salt = random string or fixed; used for key diversification;
- encode () = Text to bytes conversion function (UTF-8);

5.2.4. Final Key Generation:

A 32-byte SHA-256 intermediate key is used as the input to a Feedforward Neural Network (FFNN). The network consists of three layers (input → hidden → output). Each layer contains 32 nodes. The values of the weights (w1(32×32), w2(32×32)) and biases (b1(32), b2(32)) are chosen randomly and then fixed during the execution of activation functions in the hidden and output layers. The FFNN, with its randomized w and b values, acts as a second layer of key randomness and complexity. This reduces the correlation between the original Kurdish text and the generated key, making key analysis and prediction more difficult. Figure 4 illustrates the FFNN layers used to generate the final key.

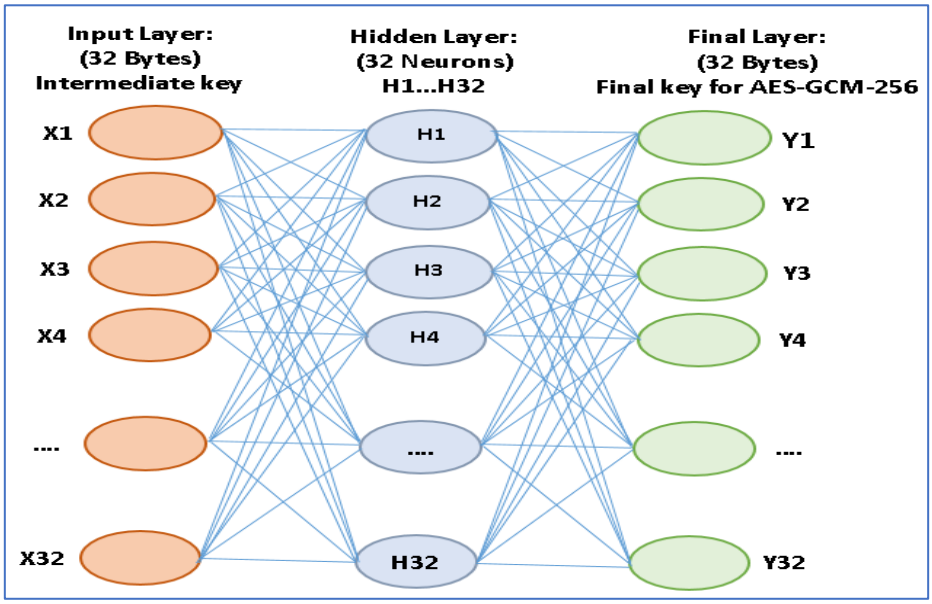

Figure 4. FFNN network layers that generate the final key generation.

The following are the steps and mathematical equations for an FFNN network to generate the final key.

1. Input Layer 0: Scaling the intermediate key: To initialize the input layer of a fully connected FFNN network. The 32-byte intermediate key must first be scaled by dividing by 255. This results in Xi values being in the range of 0-1. Prior to being passed through the sigmoid function, this step is necessary to ensure that the internal calculations of both internal layers of the FFNN are stable.

$$X_i = \frac{Input\ Byte_i}{255} \quad (19)$$

Where:

- (InputByte_i): The $_i$th byte value of the Intermediate Key, ranging in [0, 255].
- (X_i): The i_{th} element of the scaled input vector (X), ranging in [0, 1].

2. Hidden layer 1: Applying complexity function: After the scaling stage, the inputs pass into the hidden layer. The weights $W^{(1)}$ and biases $b^{(1)}$ were randomly initialized and fixed in the range [-0.5, +0.5]. This provides a balance of positive and negative effects from the weights and biases, getting the maximum complexity in the analytical inversion of the function, again enhancing the security of the key. For each encrypted text, its own w's and b's random values are generated.

A. Weighted sum hidden layer $Z^{(1)}$: Linear transformation equation of the input vector

$$Z^{(1)} = XW^{(1)} + b^{(1)} \quad (20)$$

Where:

- W(1): The fixed random weight matrix of the input-hidden layer connection, with a range of (32×32).
- b(1): The fixed random bias vector of the input-hidden layer connection, with a range of (32).

- Z(1): The resulting weighted sum vector.

B. Activation Hidden Layer $H^{(1)}$

The ReLU function is applied element-wise, introducing non-linearity.

$$A^{(1)}= h^{(1)}= \text{ReLU}(Z^{(1)})= \text{MAX}(0,Z^{(1)}) \quad (21)$$

Where:

- ReLU: The Rectified Linear Unit function.
- $A^{(1)}$: The output activation vector of the hidden layer H^1.

3. Output Layer 2: Transformation layer: The activation from the hidden layer is mapped to the final key space, employing the Sigmoid function for range clamping.

A. Weighted Sum output layer $Z^{(2)}$

$$Z^{(2)} = A^{(1)}.W^{(2)} + b^{(2)} \quad (22)$$

B. Activated output layer

The Sigmoid function is applied to constrain the output values to the [0, 1] range.

$$Key_{normalized} = O^{(2)} = \frac{1}{1 + e^{-Z^{(2)}}} \quad (23)$$

Where:

- $W^{(2)}$: The fixed random weight matrix of the hidden-output layer connection, with a range of (32×32).
- $b^{(2)}$: The fixed random bias vector of the hidden-output layer connection, with a range of (32).
- $Z^{(2)}$: The resulting weighted sum vector
- $K_{normalized} = O^{(2)} =$: The normalized key vector, ready for conversion.

C. Derived the final key from the normalized key

The normalized key values are converted back into standard 8-bit bytes (0-255) to generate the final key.

$$Key_{Final} = \lfloor Key_{normalized} \times 255 \rfloor (mod\ 255) \quad (24)$$

Where:

- $\lfloor . \rfloor$ The Floor function returns the greatest integer less than or equal to the argument.
- mod 256: The modulo operation, ensuring the resulting byte is within the [0,255] range.
- Key_{Final}: The final 32-byte cryptographic key.

5.2.5. AES encryption based on the generated key:

The final key generated by the FFNN serves as the master 256-bit key for the AESGCM256 algorithm. This master key is used to produce 15 round keys by the key expansion function. The first-round key (0) is generated for the initial round. An additional 14 round keys (1-14) are generated for other rounds. To create the ciphertext, the standard transformation methods SubBytes, ShiftRows, MixColumns, and AddRoundKey are applied. AES encrypts data in 128-bit blocks. The CTR mode produces the keystream that is used to encrypt data on a block-by-block basis. The authentication function generates the authentication tag to indicate that the data being processed has not been changed.

5.3. Conceptual Comparison of the Proposed Method and the Standard Key Derivation Functions

This section will present three widely used standard key derivation functions, which are PBKDF2, Argon2, and HKDF. This overview will assess the conceptual basis of each function, allowing for a comparison with the key derivation mechanism used in the proposed encryption system. The comparison will focus on the differences in inputs, mechanisms, and outputs for each function. Whereas the comparison of practical evaluation results will be included in the results and discussion section. The conceptual foundations of the presented key derivation functions are based on the formal definitions as specified in the official RFC documents. [40][41][42], see Table 1.

Table 1. Conceptual comparison of Standard Key derivation methods with the proposed approach

No.	Method	Input Type	Mechanism	Output
1.	**PBKDF2**	**Password + Salt + Iterations**	**Iterative cryptographic hashing with adjustable computational cost.**	**Fixed-length key**
2.	**Argon2**	**Password + Salt + Memory + Iterations**	**Memory-hard iterative processing with high memory usage**	**Fixed-length key**
3.	**HKDF**	**Initial key material + Salt + Context information**	**Two-stage extraction and expansion process for key derivation**	**One or multiple keys**
4.	**Proposed Method**	**Kurdish text features + Random Salt**	**Feature extraction followed by hashing SHA-256 and FFNN nonlinear transformation [This study].**	**Dynamic 256-bit key**

5.4. Cryptographic verification

Since the proposed encryption system is based on the Kurdish text, it is well known that the UTF-8 encoding system is used for representing, storing, and transmitting data and files over the internet. This encoding is dynamic and variable in length; it represents English letters with one byte, which fall within the range (U+0000 to U+007F), while it represents Kurdish letters with two bytes, which fall within the range (U+0600 to U+06FF). Therefore, a text consisting of 10 letters requires 10 bytes to represent it in English and 20 bytes to represent it in Kurdish. This difference in the length of the Kurdish text representation will positively affect the text's entropy, as well as the size, distribution, and properties of the data after encryption [43][44].

Verifying the security and stability of any new encryption system is crucial, and cryptographic verification is a fundamental element of this process. Cryptographic verification is defined as the evaluation of the randomness of a system and its bit distribution, as well as its sensitivity to minute changes in the plaintexts and keys. This is achieved through a series of standardized tests designed to evaluate encryption systems. Some of the most important of these tests are described below:

5.4.1. NIST SP800-22 Randomness Tests:

These tests are among the most rigorous standards for encryption systems and consist of 15 separate measures of a binary flow within the encryption system, shown in Table 2. Each measure is designed to evaluate the statistical properties of the flow, including the balance of bits and binary blocks, or the repetition of specific patterns within the flow. The importance of these tests lies in their ability to determine the level of randomness of the strings and how close they are to true randomness. One of the most important characteristics of encryption systems is their ability to protect against attacks that rely on inferences or the analysis of the statistics of the encrypted string. Therefore, if the proposed encryption system can produce a cipher string or key that passes the NIST SP800-22 tests, this indicates that the system has met one of the most important criteria for evaluating the encryption of the strings it produces. The test results are represented by the probability value (P-value), which is acceptable in the case of P-value ≥ 0.01 [45].

5.4.2. Avalanche effect analysis (AE):

This test measures the sensitivity of an encryption algorithm to a single bit change in the source text. An ideal value is approximately 50%. This means that changing just one bit in the source text results in a significant change, or at least close to half of the total binary values in the ciphertext. This indicates that the algorithm has high dispersion and is unbreakable, as there is no discernible relationship between the source text and the ciphertext. Equation (25) illustrates how this test is calculated: [46][47][48]

$$AE = \frac{1}{n}\sum_{i=1}^{n}(C_i \oplus C_i') \times 100\% \quad (25)$$

Where:

- n: Number of bits in the ciphertext

- C_i , C_i': The i[th] bit number in the ciphertext before and after changing one bit in the plaintext.

Table 2. NIST SP 800-22 Randomization Tests with the function of each Test

Seq.	Test	Function
1.	**Frequency (Monobits)**	– **Measures the homogeneity of the number of zeros and ones at the binary string level.**
2.	**Frequency within a block**	– **Measures the homogeneity of the number of zeros and ones at the block level within the binary string.**
3.	**Runs Test**	– **Tests the number of sequences of ones and zeros.**
4.	**Longest Run of Ones in a Block**	– **Tests the number of sequences of ones at the block level within the binary string.**
5.	**Binary Matrix Rank**	– **Tests the simple linear independence between parts of the binary string.**
6.	**Discrete Fourier Transform**	– **Tests based on spectral analysis to detect regular patterns or repetitions.**
7.	**Non-Overlapping Template Matching**	– **Tests to ensure that the text does not contain repetitions of abnormal patterns.**
8.	**Overlapping Template Matching**	– **To ensure that the text contains normally distributed binary patterns.**
9.	**Maurer's "Universal Statistical"**	– **A test to ensure that the text cannot be statistically compressed due to its high randomness.**
10.	**Linear Complexity**	– **To confirm the level of linear complexity of the text, which, if high, makes it difficult to predict and generate the text.**
11.	**Serial Test**	– **To measure the level of balance of short binary patterns contained in the text**
12.	**Approximate Entropy**	– **To measure the level of complexity and unpredictability by comparing the frequencies of binary patterns of different lengths within the text**
13.	**Cumulative Sums**	– **To analyze the cumulative deviation of binary patterns in a random path to confirm the absence of a statistical trend in the text string.**
14.	**Random Excursions**	– **To analyze the text string to confirm that its behaviour closely resembles a random path in the transition of statistical states.**
15.	**Random Excursions Variant**	– **To examine the frequency of different states to confirm the level of statistical equilibrium of the text in the transition of states.**

5.4.3. Correlation analysis: [49][50]

Correlation analysis provides analysts with an important means of assessing the degree of association between the plaintext and the ciphertext. A correlation coefficient close to ≈ zero indicates that the statistical method being employed to attack the encryption system has a low likelihood of success. Correlation analysis is represented by equation (26).

$$r_{Pc} = \frac{\sum_{i=1}^{n}(P_i - \bar{P})(C_i - \bar{C})}{\sqrt{\sum_{i=1}^{n}(P_i - \bar{P})^2 \ \sum_{i=1}^{n}(C_i - C)^2}} \quad (26)$$

Where:

- r_{Pc}: Correlation coefficient between the plain text and the ciphertext.
- n: Number of bits in the plain text and the ciphertext.
- P_i , C_i: The i[th] bit number in the plain text and the ciphertext.
- $\bar{P}, \bar{C}$: Average values of the plain text and the ciphertext.

5.4.4. Key sensitivity (KS):

This test adds an important dimension to the sensitivity of the encryption system. It requires that a change of even a single bit in the generated key causes significant scattering and alteration of the ciphertext. Its ideal value is 50%. This

test reflects the high reliability and sensitivity of the encryption system to the generated key. It also indicates that the encryption system does not exhibit any correlation between the ciphertexts if even a slight change is made to the bit(s) of the key used. Equation (27) illustrates how this test is calculated.[51] [52]

$$KS = \frac{1}{n}\sum_{i=1}^{n}(C_i \oplus C_i') \times 100\% \quad (27)$$

Where:

- n: Number of bits in the ciphertext
- C_i , C_i': The i^{th} bit number in the ciphertext before and after changing one bit in the key used.

Table 3 briefly presents the validation tests used to evaluate the encryption system. For each test, a positive property of the encryption system is indicated if it successfully passes on that test.

Table 3. Cryptographic validation tests and their derived properties for evaluating the proposed system

NIST SP800-22 Metrics	Pass	Cryptographic Properties
➢ **Frequency (Monobit)**	✓	
➢ **Frequency Within Block**	✓	
➢ **Cumulative Sums**	✓	**Randomness property**
➢ **Discrete Fourier Transform (FFT)**	✓	**The ciphertext bitstream exhibits random behavior**
➢ **Runs Test**	✓	
➢ **Approximate Entropy**	✓	
➢ **Maurer's Universal Statistical Test**	✓	
➢ **Discrete Fourier Transform (FFT)**	✓	
➢ **Serial Test**	✓	**Unpredictability**
➢ **Approximate Entropy**	✓	**No statistically exploitable patterns are present**
➢ **Cumulative Sums**	✓	
➢ **Frequency (Monobit)**	✓	
➢ **Frequency Within Block**	✓	**Uniformity**
➢ **Runs Test**	✓	**Balanced distribution of bits**
➢ **Longest Run of Ones**	✓	
➢ **Linear Complexity Test**	✓	
➢ **Serial Test**	✓	**Complexity**
➢ **Binary Matrix Rank Test**	✓	**High structural complexity of the bitstream**
➢ **Maurer's Universal Statistical Test**	✓	
Other Security Metrics and Cryptographic Properties		
Metrics		**Cryptographic Properties**
➢ **Avalanche (key flip)** **Effect of a 1-bit change in the key**	✓	**Diffusion**
➢ **Average Avalanche (plain flip)** **Effect of a 1-bit change in the plaintext**	✓	**A small change in the input (key/plain) leads to a significant change in the ciphertext**
➢ **Average (Correlation Plain–Cipher|)** **Correlation between plaintext and ciphertext**	✓	
➢ **Average (Correlation Key–Plaintext)** **Correlation between key and plaintext**	✓	**Statistical Independence** **No dependency between inputs and outputs**
➢ **Average Key Sensitivityy** **Sensitivity of ciphertext to key variations**	✓	**Key Sensitivity** **A small change in the key produces a completely different ciphertext**

5.5. Attack Model Evaluation

To verify the success of any new encryption system, the number of attack models (scenarios) are analyzed. These models assume the attacker's capabilities and the type of attacks they would carry out. The efficiency of the encryption

system is then verified based on its behavior, characteristics, and the verification test results. Research studies have confirmed the number of standard attack scenarios. This research focuses on the most important of these:

5.5.1. Known-plaintext attacks: In this model, the attacker has obtained the plaintext and its corresponding ciphertext and is trying to extract the relationship between them in order to extract the key. The success of the encryption system in the correlation coefficient test indicates the attacker's certain failure. [53]

5.5.2. Chosen-plaintext attacks: In this model, the attacker attempts to obtain two plaintexts with very slight differences and analyzes the differences in the resulting ciphertexts. The success of the encryption system in the avalanche test indicates the attacker's failure to discover any relationship in the resulting ciphertexts.[54]

5.5.3. Brute-force attacks: In this type of attack, the attacker attempts to try all possible keys to obtain a text similar to the desired ciphertext or part of it. If the proposed encryption system successfully tests the sensitivity of the generated key, making key prediction impossible. [55]

Table 4 briefly presents the validation tests and the positive impact of their successful implementation on attack resistance. In other words, the evaluation of attack models depends on the success of the encryption system's validation tests.

Table 4. Role of cryptographic validation tests in supporting the proposed system's resistance to attack scenarios

Tests (Pass)	Impact	Attack Model Evaluation
✓ **NIST SP800-22 (Frequency, Runs, Serial, Approximate Entropy)** ✓ **Correlation Analysis (≈ 0)**	➢ **Confirm the absence of statistical patterns.** ➢ **Indicates no dependency between plaintext and ciphertext.**	○ **Known-Plaintext Attack (KPA)**
✓ **NIST tests (FFT, Approximate Entropy, Cumulative Sums)** ✓ **Avalanche Effect (~50%)**	➢ **Confirm no predictable structure even with controlled inputs.** ➢ **Ensures small input changes produce significant output differences.**	○ **Chosen-Plaintext Attack (CPA)**
✓ **NIST tests (Linear Complexity, Maurer's Universal, Rank Test)** ✓ **Key Sensitivity (~50%)**	➢ **Indicate high structural complexity and randomness.** ➢ **Ensures incorrect keys produce completely different outputs.**	○ **Brute-Force Attack**

5.6 Central Kurdish Linguistic Text Cryptography Dataset (CKLTCD)

For the verification and evaluation of the proposed coding system, a dataset CKLTCD (.xlsx) was created and made publicly available on Zenodo [56]. The dataset includes 3,000 Kurdish texts of varying lengths. These texts cover multiple fields, including cultural, educational, literary, news, scientific, and technological topics. These texts were extracted from multiple electronic sources, including several news websites and digital knowledge platforms, most notably Kurdistan24, Rudaw, NRT, Kurdish Wikipedia, and Kurdipedia.

The dataset was collected and organized in a structured format using an excel file with 18 total worksheets, one or more for each stage of the new proposed model. There were worksheets for the Kurdish text, text length, text feature vectors, and final keys generated from every text. There were also worksheets for the test results and a comparison of the test results with results obtained from official key generators (PBKDF2, Argon2, and HKDF), which will be discussed later. Table 5 illustrates the structure of the main contents of the CKLTCD dataset file.

6. Practical Implementation:

Python was chosen for developing the proposed encryption system because it offers library functions that effectively support all stages of the proposed system. Figure 5 illustrates the graphical user interface (GUI) for inputting Kurdish text and performing the preprocessing operations (cleaning and normalization) to obtain the pure normalized Kurdish text.

Table 5: Structure of CKLTCD (.xlsx) dataset: worksheet titles with the main associated field names

Sheet No.	Sheet Name	Field Names					
01	Texts_Info_3000	Seq	Domain	No_of_letteres	Plan_Text		
02	FFNN_Random_Params	FFNN_Seed	W1 (32x32)	b1 (1x32)	W2 (32x32)	b2 (1x32)	
03	Final_Keys_256bit	PBKDF2_Final_Key	Argon2_Final_Key	HKDF_Final_Key_hex	Proposed_Final_Key		
04	Plain_Ciphertexts	Domain	Text	PBKDF2_Cipher_hex	Argon2_Cipher_hex	HKDF_Cipher_hex	Proposed_Cipher_hex
05	PBKDF2_Test_Details	Frequency (Monobit)	Frequency Within Block	…….	……..	……	Key_Sensitivity
06	PBKDF2_NIST15	Test No.	Test Name	Aggregated p-value	Threshold	Decision	Note
07	PBKDF2_Summary	Metric	Value	Interpretation	Target		
08	Argon2_Test_Details	Frequency (Monobit)	Frequency Within Block	…….	……..	……	Key_Sensitivity
09	Argon2_NIST15	Test No.	Test Name	Aggregated p-value	Threshold	Decision	Note
10	Argon2_Summary	Metric	Value	Interpretation	Target		
11	HKDF_Test_Details	Frequency (Monobit)	Frequency Within Block	…….	……..	……	Key_Sensitivity
12	HKDF_NIST15	Test No.	Test Name	Aggregated p-value	Threshold	Decision	Note
13	HKDF_Summary	Metric	Value	Interpretation	Target		
14	Proposed_Test_Details	Frequency (Monobit)	Frequency Within Block	…….	……..	……	Key_Sensitivity
15	Proposed_NIST15	Test No.	Test Name	Aggregated p-value	Threshold	Decision	Note
16	Proposed_Summary	Metric	Value	Interpretation	Target		
17	Final_Comparison	Metric	PBKDF2(Test Value, Result)	Argon2(Test Value, Result)	HKDF (Test Value, Result)	Proposed (Test Value, Result)	
18	Method_Notes	Item	Details				

The normalized text now serves as the basis for extracting and calculating the feature matrix. For the example input text, length of the text is (93 char), the number of words is (24), number of unique letters in this text is (26), the number of unique and distinctive bigrams is (76), the number of unique and distinctive trigrams is (87), and the entropy value of this Kurdish text is (4.3186 bits/char), See the GUI in Figure 6.

Of course, as is clear from Figure 6, the number of occurrences of letters, bigrams, and trigrams is large, and these numbers increase with the length of the text. Therefore, these features were summarized, and only the top (10) occurrences of letters, bigrams, and trigrams. were used. Accordingly, the final summarized feature vector formed from this text was as follows: See the GUI in Figure 7.

[93,24,4.32,26,76,87,13,11,7,5,5,5,5,4,4,4,3,2,2,2,2,2,2,2,2,2,2,2,2,2,1,1,1,1,1,1]

Figure 5. Input of the Kurdish text interface with the pre-process operations.

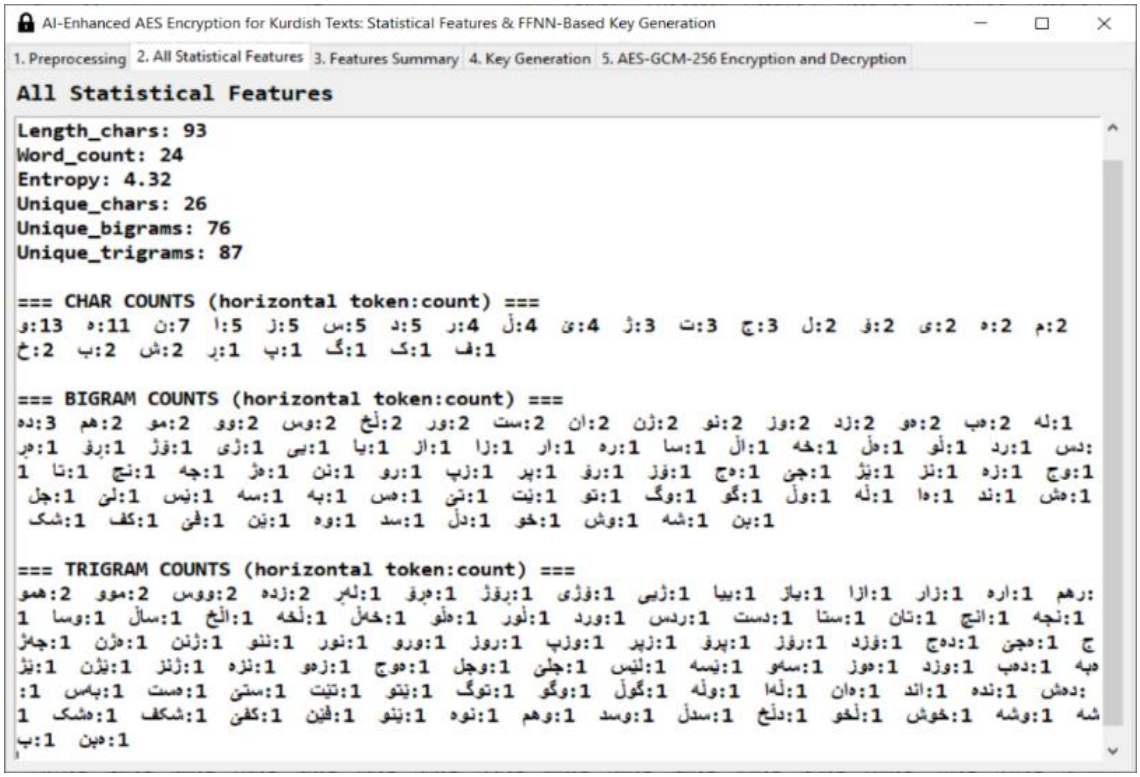

Figure 6. GUI for feature extraction of the central Kurdish text.

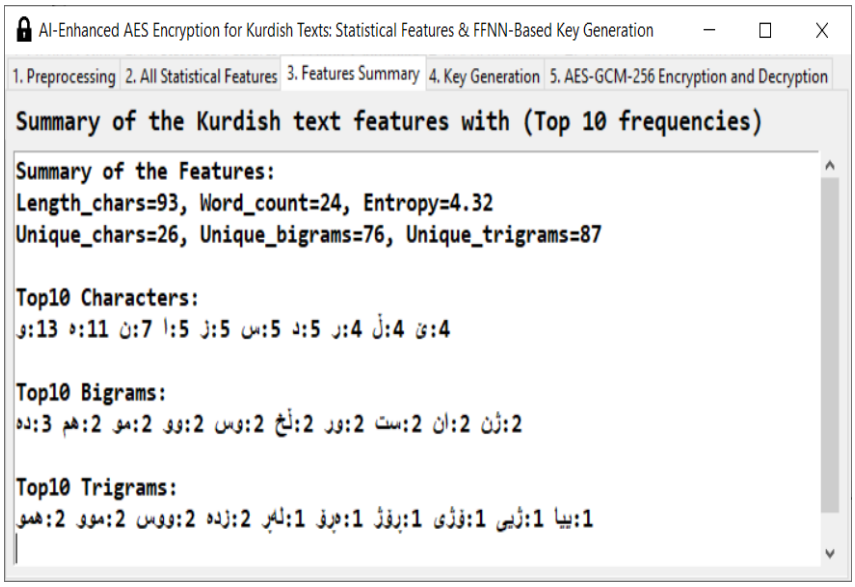

Figure 7. Summarized feature extraction of the central Kurdish text.

The above summarized feature vector is fed into the hash function (SHA256) to generate the intermediate key. A salt value (chosen manually or randomly) is added to increase the complexity and randomness of the intermediate key and to prevent duplicate keys from being created in cases where the texts to be encrypted are identical. The intermediate key acts as an input to the FFNN network, which, in turn, acts as a nonlinear complexity function, partitioning the key into bytes. Each byte occupies one of the 32 inputs of the FFNN input layer. Random values of weights (W1 and W2) and bias values (b1 and b2) are generated for each byte to be used in the required equations in the hidden and output layers. The Kurdish alphabet, as previously mentioned, consists of 31 letters. Therefore, the maximum entropy value for the Central Kurdish alphabet is 4.9542 bits/char. The entropy value of this tested Kurdish text, H(TextNormalized)=4.3186 bits/char, is high and close to the maximum entropy value for the Kurdish alphabet, at 87.17%. And the entropy value of the final generated key for this Kurdish text, H(Keyfinal)=3.6508 bits/char, which is a good entropy value, close to the maximum entropy value for the Kurdish alphabet, at 73.69%. See the GUI in Figure 8.

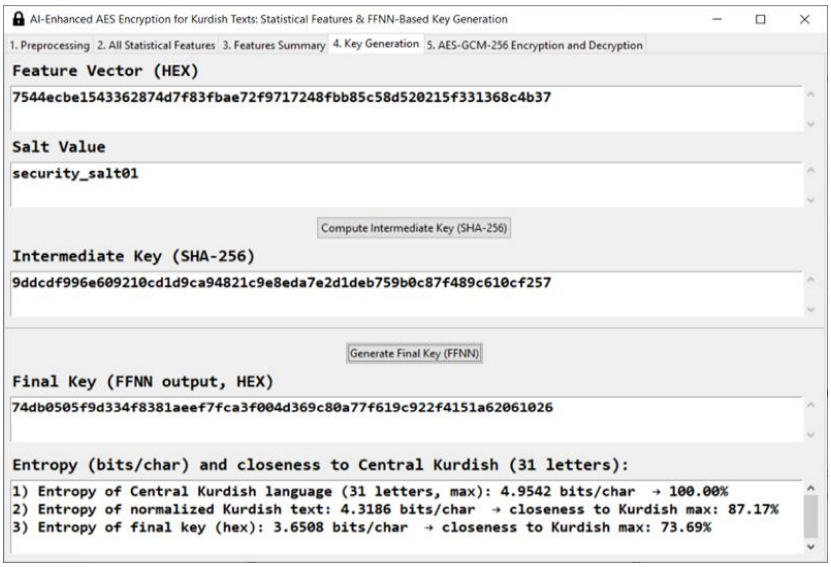

Figure 8. The intermediate and final (master) keys generation.

The final key generated by the FFNN is used to encrypt the text itself. The original text (before preprocessing) and its generated key were fed into the AESGCM256 algorithm. The ciphertext was generated and then verified through decryption. The decrypted text was 100% identical to the original text. See the GUI in Figure 9.

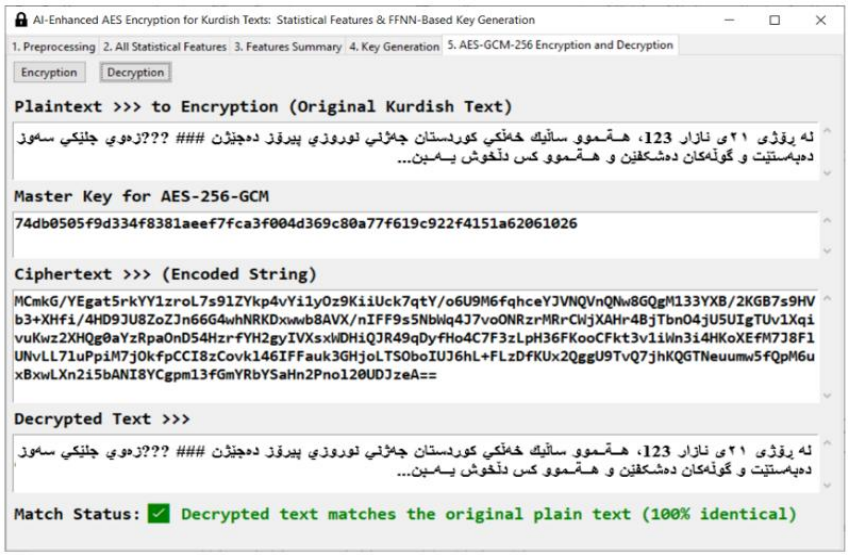

Figure 9. Encryption and Decryption process based on the generated master key.

7. Result and Discussion:

7.1 Preliminary tests of a small sample of central Kurdish texts:

Initially, some preliminary tests were carried out on a small sample of central Kurdish texts, and their success formed the starting point for conducting standard and approved tests on a dataset that included a large number of texts.

7.1.1. Generating a unique linguistic signature for each text:

Figure (10-a) shows the normalized text that was tested, along with a summary of the extracted and calculated features. Figures (10-c, 10-b, and 10-d) show the line charts for the distribution of letters, bigrams, and trigrams, respectively. These results reflect the statistical structure of the Kurdish text. Furthermore, these results indicate that the cleaning and normalization processes did not distort the text. Thus, these results constitute a unique linguistic signature for each text. Here, the role of the random salt value, along with the weights and biases of the FFNN network, which are also random, becomes apparent. These random values add a level of randomness and ensure the generation of different random keys even if the plain texts are similar. This provides a reliable basis for creating strong and dynamic encryption keys.

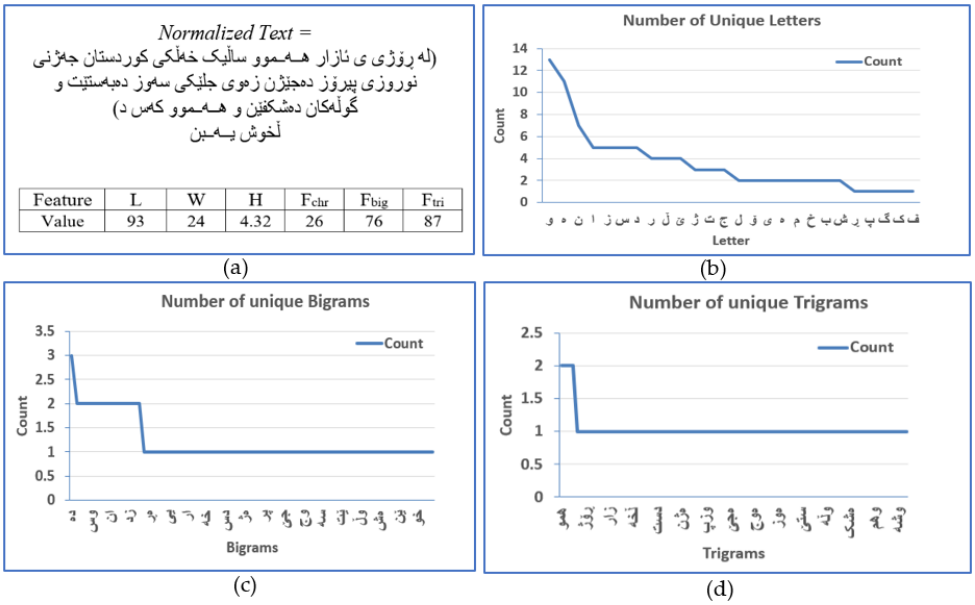

Figure 10. Statistical features for: (a) Specific tested Kurdish text with distribution of; (b) Letters; (c) Bigrams; (d) Trigrams.

7.1.2. Entropy measurement for a specific number for both generated keys and plain texts:

Table 6 shows the master keys in two formats (hexadecimal and textual), generated from the textual features of a number of tested Kurdish texts. As mentioned previously, the generated key is 32 bytes (256 bits) in size. These keys were tested for encrypting their corresponding Kurdish text. The decrypted texts were matched to the originals.

The randomness (entropy) of both the Kurdish texts and the keys was measured. As shown in Table 7, the average entropy of the texts was 3.852 bits/character, with an approximation of 77.75% to the optimal value for Kurdish alphabetic entropy. The average entropy of the keys generated from these texts was 3.6222 bits/character, with an approximation of 73.11% to the optimal value for Kurdish alphabetic entropy. It should be noted that the optimal value represents the upper limit of Kurdish alphabetic entropy, which was previously calculated to be 4.9542 bits/character. The above confirms that the Kurdish language was suitable for key generation due to its richness, as its texts and generated keys are characterized by high randomness. The mean of the standard deviation of the Kurdish text samples was 0.118439764, and the mean of the standard deviation of the generated keys was 0.189239907. These small values reflect the homogeneity and stability of the Kurdish texts and their generated keys, despite the differences in the tested texts; see Table 7.

Table 6. Tested Kurdish texts and the keys generated for each of them

Text No.	Plain Kurdish Texts	Generated Master Key	
		Hexadecimal Format	Text Format
T1	پێویستە تا کاتژمێر 8:30 سەرلەبەیانی، تەواوی ئامادەکارییەکان بۆ کۆبوونەوەکە تەواو بکەین، ئینجا بڕۆین.	**0194c752fef502f336d6f3a1d226ecfaf9 fdcde91c03f80200f209fdea06e0fe**	**” ÇRþõó6Öó¡Ò&ìúùýÍ éøò ýêàþ**
T2	تەنیا دوو (2) هەفتە ماوە تا پڕۆژەکە تەواو بکرێت؛ ئایا دەتوانین لەم ماوە کەمەدا کاری باش بکەین؟	**a4e7e203f639fcbefd83fd1f0808c5f1f182 fd6e96ee0685e6dfd2fd14ce0aff**	**¤çâö9ü¾ý✾ý- Åññ✾ýn– î✾æßÒýÎÿ**
T3	ئەگەر سێ (3) خولەک چاوەڕێ بکەیت، من بە دڵنیاییەوە دێم و هەموو شتێکت بۆ ڕوون دەکەمەوە!	**63f801027f037f99fc940022c131fc0008fd e8a2df30df1201fce80003ecc1d3**	**cø□□✾ü” "Á1üýè ¢ß0ßüèìÁÓ**
T4	ئەم کتێبە، کە نرخی تەنها 15,000 دینارە، لەسەر چوار (4) بەشی سەرەکی نووسراوە.	**9ae9bd0f8ef8fee2b6fdeed306f24249db 2a3bfe48a7570de8e1bf0314f4eb3a**	**✾é½✾øþâ¶ýîÓòB IÛ*;þH§Wèá¿ôë:**
T5	تەواوی ڕێگاکە، کە نزیکەی 1,200 کیلۆمەترە، بە باشترین شێوە کۆتایی هات و هیچی لێ ڕووی نەدا.	**061d29020004fefff13c00f18342fdf903 fe040d001409fbaa03e21386fbf8f3**	**)þÿñ<ñ✾Býùþ ûªâ✾ûøó**
T6	لە ساڵی 2024 دا، من پێنج (5) کتێبم خوێندەوە، بەڵام هیچ کامیان بە قەد ئەو یەکە بەسوود نەبوون.	**fafef6ff020a04ff55f300e6fe0ffb117 bebfee1fa0691ce1d000b4309fffdff**	**úþöÿÿUóæþû{ ëþáú✾ÎC ÿýÿ**
T7	ئێمە شەش (6) جار سەردانی ئەو شوێنەمان کردووە، ئایا ئێستا کاتی ئەوە نەهاتووە شوێنێکی تر ببینین؟!	**9501003afeeeeefb07038f009da20acaf d1ac080b905da4001f808f0fdf9031a**	**✾:þîîû ✾✾¢ ÊýÀ✾¹Ú@øðýù**
T8	چوونکە من لە دەرەوە بووم، ناتوانم وەڵامت بدەمەوە؛ تکایە دوای 15 خولەک جارێکی تر پەیوەندی بکە.	**2b09c083de0332e124f0d8e8b0f88cc9 4d265a100c749d2ea63b147e1f69f8f5**	**+À✾Þ2á$ðØè°ø✾É M&Zt✾.¦;~iøõ**
T9	ئەوان هەوڵ دەدەن لە ماوەی 90 ڕۆژدا، ئەم کێشە گەورەیە چارەسەر بکەن؛ بەڕاستی کارێکی قورسە.	**c4326808fc5d78f835fe9c00fdf5d3fe 0afe8cb6ef51e7120d6f2442e2f804cf**	**Ä2hü]xø5þ✾ýõÓþ þ✾¶ïQçº$BâøÏ**
T10	بەڵێ، کاتێک دەچیتە بازار، هەندێک شتم بۆ بێنە، وەك: نان، شەکر، و زەیت؛ سوپاس دەکەم.	**09ea00b1e84912f9eefbf6d9fba5e36 500bd000adc105b0807fb042bffb200a2**	**ê±èIùîûöÙû¥ãe½ Ü[û+ÿ²¢**

Figure 11 clearly shows a significant difference in the high entropy values of both the pain text and the generated key. This difference arises from the random salt values and complexity layers incorporated by the proposed encryption system, which are represented with “SHA” and “FFNN". This difference establishes a complex, non-linear relationship

between the text and the key. This also provides support for the proposed encryption system's resistance to statistical analysis attacks.

Table 7. Text entropy and key entropy for some Kurdish texts

Text No.	No. of letters	Entropy bits/char		Closeness to Max Entropy	
		Text	Master Key	Text Entropy	Master Key Entropy
T1	79	3.7909	3.6137	76.52%	72.94%
T2	72	3.7803	3.8649	76.31%	78.01%
T3	64	3.9429	3.4643	79.59%	69.93%
T4	51	3.7298	3.7078	75.29%	74.84%
T5	66	3.9308	3.4303	79.34%	69.24%
T6	64	3.9071	3.3143	78.86%	66.90%
T7	69	3.7209	3.571	75.11%	72.08%
T8	73	3.7129	3.9141	74.94%	79.01%
T9	67	3.9617	3.7295	79.97%	75.28%
T10	59	4.0427	3.6121	81.60%	72.91%
Average:		3.852	3.6222	77.75%	73.11%
VAR.S:		0.014028	0.035811742	0.000571116	0.001458943
STD.S:		0.1184398	0.189239907	0.023898026	0.038196108
Central Kurdish Alphabet Entropy (Max Entropy) = (4.9542 bits/char)					

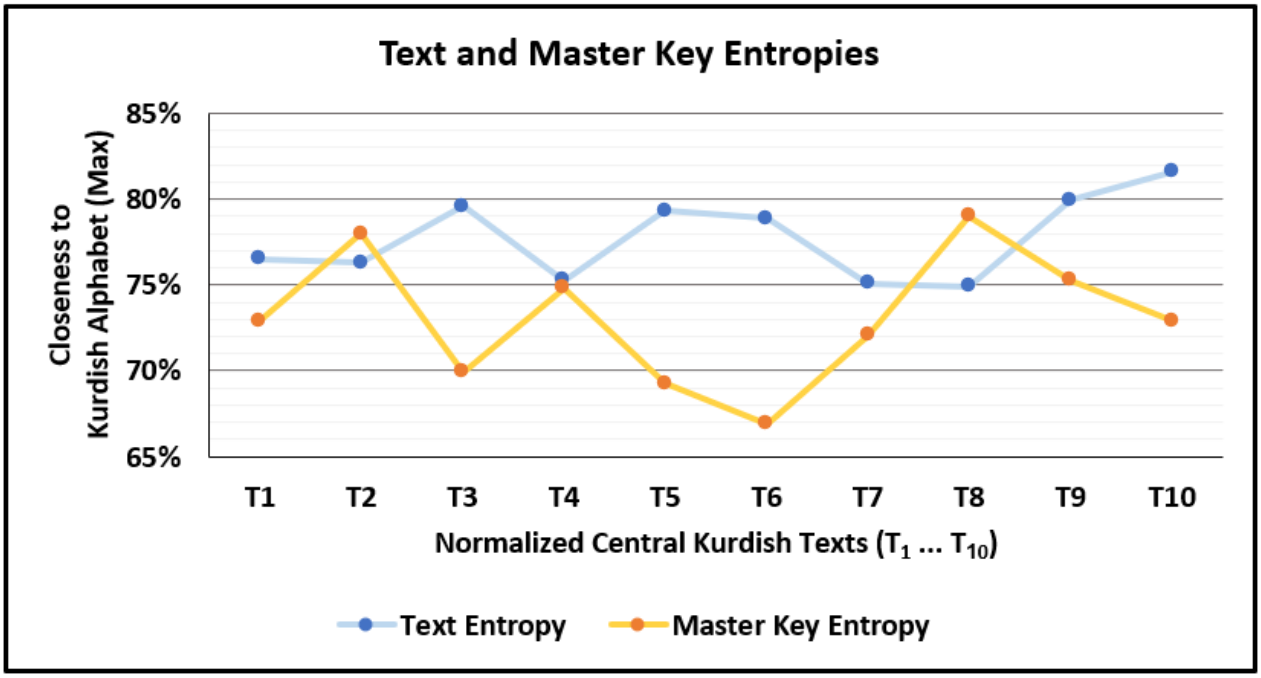

Figure 11. The relationship between text entropy and the generated master key entropy.

7.2. Implementation of cryptographic validation:

The proposed encryption system was applied to a CKLTCD of 3000 Kurdish texts; see Table 8. The following is a discussion of these results.

1. The proposed encryption system passed 13 out of 15 NIST SP800-22 tests, achieving a success rate of 13/15. This demonstrates a high degree of randomness in the binary strings of the ciphertexts. The two template matching tests

were not performed because they require the tested data to have very long binary strings. The following is an interpretation of the results from the most important NIST SP800-22 tests.

a. The successful frequency and runs tests indicate a good balance within the binary string.
b. The successful FFT, serial, and approximate entropy tests indicate that there are no repeating patterns in the data being tested.
c. The linear complexity and Maurer's universal tests were successful and indicate that the binary strings have structural complexity.

2. The avalanche effect testing was conducted and provided results that are close to the ideal (≈ 50%). This indicates that the proposed encryption system is very sensitive to change. In other words, if the original text and/or key changes, the ciphertext will also change significantly, resulting in a strong resistance to CPA attacks on the encryption system.
3. Correlation analysis tests showed results very close to zero, indicating that the encryption system is statistically independent. This confirms the absence of a linear relationship between the source text and the ciphertext. This gives the proposed encryption system strong resistance to penetration analysis (KPA) attacks.
4. Key sensitivity tests also showed results very close to the optimal value (≈ 50%). This means that if the key changes slightly, the ciphertext will change significantly. This result supports the encryption system's resistance to brute-force attacks.

Table 8. Cryptographic validation results of the proposed system based on the ciphertext

Proposed_Text Feature_SHA_FFNN - NIST SP800-22 15-Test Summary (Ciphertext-Based)				
Test No.	**Test Name**	**P-value**	**Threshold**	**Decision**
1	**Frequency (Monobit)**	**0.104**	**≥ 0.01**	**Pass**
2	**Frequency Within Block**	**0.940**	**≥ 0.01**	**Pass**
3	**Cumulative Sums**	**0.173**	**≥ 0.01**	**Pass**
4	**Runs**	**0.354**	**≥ 0.01**	**Pass**
5	**Longest Run of Ones**	**0.065**	**≥ 0.01**	**Pass**
6	**Binary Matrix Rank**	**0.703**	**≥ 0.01**	**Pass**
7	**Discrete Fourier Transform (FFT)**	**0.114**	**≥ 0.01**	**Pass**
8	**Non-overlapping Template Matching**	**-**	**≥ 0.01**	**N/A**
9	**Overlapping Template Matching**	**-**	**≥ 0.01**	**N/A**
10	**Maurer's Universal**	**1.000**	**≥ 0.01**	**Pass**
11	**Approximate Entropy**	**0.080**	**≥ 0.01**	**Pass**
12	**Random Excursions**	**0.500**	**≥ 0.01**	**Pass**
13	**Random Excursions Variant**	**0.500**	**≥ 0.01**	**Pass**
14	**Serial**	**0.130**	**≥ 0.01**	**Pass**
15	**Linear Complexity**	**0.979**	**≥ 0.01**	**Pass**
	Average	**0.434**		**13/15 Passed**
N/A	**Not Applicable: Insufficient condition on aggregated ciphertext bitstream.**			
	Other Security Metrics			
Test No.	**Test Name**	**Aggregated Value**	**Threshold**	**Decision**
1	**Average Avalanche (key flip) %**	**50.011**	**~50%**	**Pass**
2	**Average Avalanche (plain flip) %**	**49.993**	**~50%**	**Pass**
3	**Average (Correlation Plain-Cipher)**	**0.048**	**0**	**Pass**
4	**Average (Correlation Key-Plain)**	**0.147**	**0**	**Pass**
5	**Average Key Sensitivity %**	**50.011**	**~50%**	**Pass**
Average (Avalanche+ Key Sensitivity)		**50.005**		
Average (Correlation)		**0.097**		

7.3. Comparison with Standard Key Generators.

To verify the efficiency of the proposed encryption system, it was compared with standard key generators: PBKDF2, Argon2, and HKDF. To achieve this, keys of the same size (256 bits) were generated using these generators. These keys were then used to encrypt the same Kurdish texts in the dataset (CKLTCD) by using the AES algorithm. As shown in Table 9 and based on Table 3, the validation results for the proposed encryption system were comparable to those of the standard generation methods. The results confirm that the proposed key generation method possesses similar characteristics of standard encryption generators in terms of randomness, statistical independence, dispersion, diffusion, complexity, and key sensitivity to achieve the desired outcome of unpredictability.

Table 9. Comparison of Cryptographic Validation Results Between the Proposed Method and Standard KDFs

Test No.	Metric	PBKDF2		Argon2		HKDF		Proposed_Text Feature_SHA_FFNN	
		P-value	Result	P-value	Result	P-value	Result	P-value	Result
1	Frequency (Monobit)	0.455	Pass	0.768	Pass	0.648	Pass	0.104	Pass
2	Frequency Within Block	0.849	Pass	0.961	Pass	0.501	Pass	0.940	Pass
3	Cumulative Sums	0.744	Pass	0.857	Pass	0.087	Pass	0.173	Pass
4	Runs	0.457	Pass	0.221	Pass	0.549	Pass	0.354	Pass
5	Longest Run of Ones	0.467	Pass	0.915	Pass	0.943	Pass	0.065	Pass
6	Binary Matrix Rank	0.599	Pass	0.255	Pass	0.333	Pass	0.703	Pass
7	Discrete Fourier Transform (FFT)	0.485	Pass	0.295	Pass	0.325	Pass	0.114	Pass
8	Non-overlapping Template Matching	-	N/A	-	N/A	-	N/A	-	N/A
9	Overlapping Template Matching	-	N/A	-	N/A	-	N/A	-	N/A
10	Maurer's Universal	1.000	Pass	1.000	Pass	1.000	Pass	1.000	Pass
11	Approximate Entropy	0.235	Pass	0.720	Pass	0.197	Pass	0.080	Pass
12	Random Excursions	0.500	Pass	-	N/A	-	N/A	0.500	Pass
13	Random Excursions Variant	0.500	Pass	-	N/A	-	N/A	0.500	Pass
14	Serial	0.405	Pass	0.604	Pass	0.667	Pass	0.130	Pass
15	Linear Complexity	0.957	Pass	0.985	Pass	0.968	Pass	0.979	Pass
	NIST 15 summary	13/15 passed		11/15 passed		11/15 passed		13/15 passed	
Test No.	Metric	PBKDF2		Argon2		HKDF		Proposed_Text Feature_SHA_FFNN	
		Value	Result	Value	Result	Value	Result	Value	Result
1	Avg. Avalanche (key flip) %	50.011	Pass	49.985	Pass	49.997	Pass	50.011	Pass
2	Avg. Avalanche (plain flip) %	50.002	Pass	49.993	Pass	50.009	Pass	49.993	Pass
3	Avg. |Correlation Plain-Cipher|	0.048	Pass	0.048	Pass	0.049	Pass	0.048	Pass
4	Avg. |Correlation Key-PlainHash|	0.144	Pass	0.143	Pass	0.145	Pass	0.147	Pass
5	Avg. Key Sensitivity %	50.011	Pass	49.985	Pass	49.997	Pass	50.011	Pass
	Average (Avalanche+ Key Sensitivity)	50.008		49.988		50.001		50.005	
	Average (Correlation)	0.096		0.096		0.097		0.097	

- **Dataset size: This workbook uses the first 3000 Kurdish texts from kurdish_texts.xlsx.**
- **Ciphertext basis: All NIST SP800-22 results in this workbook were computed on concatenated ciphertext bitstreams for each method, not on generated keys.**
- **Salt usage: Random salt values were generated for every text in all four methods.**
- **FFNN parameters: A single global random FFNN (W1, b1, W2, b2) was generated and fixed for the whole experiment; all parameter values are stored in sheet 03_FFNN_Random_Params.**
- **Key sensitivity basis: Key Sensitivity was computed on the ciphertext level, using the ciphertext change after a 1-bit key modification, instead of measuring the bit change inside the key itself.**

7.4. Attack model evaluation based on cryptographic validation results:

The resistance of the proposed encryption system to some basic attack patterns was evaluated. This evaluation was based on the validation tests described previously in Tables 3 and 4, as well as their results shown in Table 8. Thus, the evaluation can be summarized as follows:

1. Various test results reported (frequency, runs, FFT, approximate entropy, and correlation) indicate that the linear relationship between the plain text and ciphertext is either weak or nonexistent; therefore, it can be concluded that the proposed encryption system supports resistance to KPA attacks.
2. Various test results reported (cumulative sums, entropy, serial, and avalanche effect) indicate that any time there is a change in one input parameter (i.e., key/plaintext), the ciphertext will exhibit a significant change, thus making the structure/behavior of the ciphertext next to impossible to predict. Therefore, the results indicate that the proposed encryption system supports resistance to CPA attacks.
3. The results of various reported tests (binary matrix rank, Maurer's universal test, linear complexity, and key sensitivity). In addition, the proposed encryption system can generate a 256-bit key. This would make predicting 2^{256} keys impossible in terms of time. Therefore, the proposed encryption system is highly resistant to brute-force attacks.

8. Conclusions:

The proposed encryption system integrates the confidentiality, efficiency, and trustworthiness of the AES-GCM-256 algorithm with an effectively generated 256-bit dynamic key (via the features of the central Kurdish text). The features of the Kurdish language, in terms of the number of letters, vowels, and suffixes, provide an appropriate basis for the diversity and distribution characteristics of the proposed encryption system. The hash function (SHA) with random salt provides an additional layer of protection by obscuring the relationship between the plaintext and the key, and it also ensures that the same input will never yield the same key, thereby preventing reuse and precomputation attacks. The FFNN network contributes to the security of the proposed encryption system in two primary ways. First, the randomly generated parameters (weight and bias values) of the one-way neural network increase randomness. Therefore, similar inputs will not produce similar keys. Second, the nonlinear transformations produced by the one-way neural network greatly increase the complexity of the relationship between input and output of the one-way neural network and therefore provide an additional layer of security against attempts to predict or use analytical methods targeted at the generated key. The generated key of the proposed encryption system is secure because it is not generated directly from the plaintext's characteristics; rather, it is generated from intermediate values that are produced through a series of processing steps, i.e., encoding, cryptographic hashing (SHA), and nonlinear transformations (FFNN). The overall performance of the proposed encryption system is similar to that of standard key derivation functions: PBKDF2, Argon2, and HKDF. This supports the contribution of the study by providing an alternative approach for dynamic key generation based on linguistic features of the text, particularly for Kurdish texts.

9. Limitations

The study faced limitations during the research development phase, primarily due to the limited availability of scientific and linguistic resources related to leveraging Kurdish text features for cryptographic key generation. Although the proposed encryption system demonstrates strong performance in key generation efficiency and security enhancement, its methodological scope is restricted to the Kurdish language. The selected statistical and structural features, while effective within this context, confine the model to a specific linguistic framework, thereby limiting the direct generalizability of the results to languages with different structural and statistical properties.

Extending the proposed system to other languages, such as English or Arabic, presents challenges due to linguistic variability, as each language possesses distinct structural and statistical characteristics. These variations result in substantial differences in the features extracted from textual data. Consequently, adapting the proposed methodology requires a comprehensive redesign of the feature extraction process, including redefining the feature vector and selecting language-specific features that accurately reflect the linguistic structure of each target language. Furthermore, deploying the system in multilingual environments necessitates a thorough re-evaluation of its performance in terms of both efficiency and security to ensure robustness across diverse linguistic contexts.

Author Contributions: Conceptualization, Z.H.A.; methodology, Z.H.A.; software, Z.H.A.; validation, Z.H.A.; formal analysis, Z.H.A.; investigation, Z.H.A.; resources, Z.H.A.; data curation, Z.H.A.; writing—original draft preparation, Z.H.A.; writing—review

and editing, Z.H.A.; visualization, Z.H.A.; supervision, Z.H.A.; project administration, Z.H.A. The author has read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The dataset supporting the reported results in this study, named the Central Kurdish Linguistic Text Cryptography Dataset (CKLTCD), has been made publicly available and can be found on Zenodo at https://doi.org/10.5281/zenodo.19742666.

Acknowledgments: Not applicable.

Conflicts of Interest: The author declares no conflicts of interest.

Abbreviations

The following are the abbreviations used in this research:

AES	**Advanced Encryption Standard**
GCM	**Galois/Counter Mode**
CTR	**Counter Mode**
IV	**Initialization Vector**
Nonce	**Number Used Once**
AAD	**Additional Authentication Data**
GHASH	**Galois Hash function**
SHA-256	**Secure Hash Algorithm- 256-bit**
FFNN	**Feed-Forward Neural Network**
ReLU	**Rectified Linear Unit**
HMAC	**Hash-based Message Authentication Code**
KPA	**Known-plaintext attacks**
CPA	**Chosen-plaintext attacks**
PBKDF2	**Password-Based Key Derivation Function 2**
Argon2	**A memory-hard password hashing function (not an acronym)**
HKDF	**HMAC-based Key Derivation Function**

References

1. Mohammed N. Alenezi, Haneen Alabdulrazzaq, H.M.A. and F.A.A. On the Performance of AES Algorithm Variants. *International Journal of Information and Computer Security* **2024**, *Vol. 23, N*, doi:https://doi.org/10.1504/IJICS.2024.138494.
2. Mu, C. Application of Optimizing Advanced Encryption Standard Encryption Algorithm in Secure Communication of Vehicle Controller Area Network Bus. **2024**, 1–11, doi:10.3389/fmech.2024.1407665.
3. Chen, A.C.H. Performance Comparison of Various Modes of Advanced Encryption Standard. **2024**, doi:10.1109/ICSSES62373.2024.10561385.
4. Ganesh, R.; Khan, B.U.I.; Khan, A.R.; Kamsin, A. Bin A Panoramic Survey of the Advanced Encryption Standard: From Architecture to Security Analysis, Key Management, Real-World Applications, and Post-Quantum Challenges. *Int. J. Inf. Secur.* **2025**, *24*, doi:10.1007/s10207-025-01116-x.
5. Soni, A.; Sahay, S.K.; Mehta, P. AESHA3: Efficient and Secure Sub-Key Generation for AES Using SHA-3. **2025**, doi:10.1007/978-3-031-81168-5_5.
6. Barker, E.; Roginsky, A.; Davis, R. *Recommendation for Cryptographic Key Generation*; Gaithersburg, MD, 2020;
7. Bonneau, J.; Herley, C.; Van Oorschot, P.C.; Stajano, F. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. *Proc. IEEE Symp. Secur. Priv.* **2012**, 553–567, doi:10.1109/SP.2012.44.
8. Kim, J.; Song, M.; Seo, M.; Jin, Y.; Shin, S.; Kim, J. PassREfinder-FL: Privacy-Preserving Credential Stuffing Risk Prediction via Graph-Based Federated Learning for Representing Password Reuse between Websites. **2025**.
9. Sandhya Venu, V.; Rithwik, D.; Prasoon, M.M.; Praneeth, D.S. A Bio-Cryptographic Approach to Aes Key Generation Using Randomized DNA Genes and Binary Encoding. **2025**, *14*, 3–6.
10. Setiawan, Y.; Maulidevi, N.U.; Surendro, K. The Optimization of N-Gram Feature Extraction Based on Term Occurrence for Cyberbullying Classification. *Data Sci. J.* **2024**, *23*, 1–21, doi:10.5334/dsj-2024-031.
11. Hassani, H.; Beneki, C.; Unger, S.; Mazinani, M.T.; Yeganegi, M.R. Text Mining in Big Data Analytics. *Big Data and Cognitive Computing* **2020**, *4*, 1–34, doi:10.3390/bdcc4010001.
12. Kalimeri, M.; Constantoudis, V.; Papadimitriou, C.; Karamanos, K.; Diakonos, F.K.; Papageorgiou, H. Entropy Analysis of Word-Length Series of Natural Language Texts: Effects of Text Language and Genre. *International Journal of Bifurcation and Chaos* **2012**, *22*, doi:10.1142/S0218127412502239.

13. Liu, K.; Ye, R.; Zhongzhu, L.; Ye, R. Entropy-Based Discrimination between Translated Chinese and Original Chinese Using Data Mining Techniques. *PLoS One* **2022**, *17*, doi:10.1371/journal.pone.0265633.
14. Kodwani, G.; Arora, S.; Atrey, P.K. On Security of Key Derivation Functions in Password-Based Cryptography. *Proceedings of the 2021 IEEE International Conference on Cyber Security and Resilience, CSR 2021* **2021**, 109–114, doi:10.1109/CSR51186.2021.9527961.
15. Dang, Q.H. Secure Hash Standard. *FIBS 180-4 Publication* **2015**, *4*, 36.
16. Barker, E.; Roginsky, A. Transitioning the Use of Cryptographic Algorithms and Key Lengths. *NIST Special Publication 800-131A Revision 2* **2019**, 17–18.
17. Liu, Y.A.; Chen, L.; Li, X.W.; Liu, Y.L.; Hu, S.G.; Yu, Q.; Chen, T.P.; Liu, Y. A Dynamic AES Cryptosystem Based on Memristive Neural Network. *Sci. Rep.* **2022**, *12*, doi:10.1038/s41598-022-13286-y.
18. Zied, G.; Zrigui, M.; Guitouni, Z.; Zairi, A. Secure Transmission of Medical Images in IoMT Systems Using Feedforward Neural Networks. **2023**.
19. Wu, X.; Han, Y.; Zhang, M.; Zhu, S.S.; Cui, S.; Wang, Y.; Peng, Y. Pseudorandom Number Generators Based on Neural Networks: A Review. *Journal of King Saud University - Computer and Information Sciences* **2025**, *37*, doi:10.1007/s44443-025-00007-4.
20. Guitouni, Z.; Zairi, A.; Zrigui, M. Implementation of Neural Key Generation Algorithm For IoT Devices. *Journal of Computer Science Advancements* **2024**, *1*, 276–290, doi:10.70177/jsca.v1i5.637.
21. Nitaj, A.; Rachidi, T. Applications of Neural Network-Based AI in Cryptography. *Cryptography* **2023**, *7*, 1–26, doi:10.3390/cryptography7030039.
22. Wu, X.; Han, Y.; Zhang, M.; Li, Y.; Cui, S. GAN-Based Pseudo Random Number Generation Optimized through Genetic Algorithms. *Complex and Intelligent Systems* **2025**, *11*, doi:10.1007/s40747-024-01606-w.
23. Badawi, S. Deep Learning-Based Cyberbullying Detection in Kurdish Language. *Comput. J.* **2024**, *67*, 2548–2558, doi:10.1093/comjnl/bxae024.
24. Abdulrazaq, N.N. A Novel Approach For Safeguarding Kurdish Text Files Via Modified AES-OTP And Enhanced RSA Cryptosystem On Unreliable Networks. *Eurasian Journal of Science and Engineering* **2024**, *10*, 102–119, doi:10.23918/eajse.v10i2p8.
25. Abid Al Jabbar, Z.H.; Ali, Z.A.; Taher, H.A. Design and Implementation of a Mathematical Encryption Model for the Central Kurdish Font Based on Unicode. *Science Journal of University of Zakho* **2023**, *11*, 273–279, doi:10.25271/sjuoz.2023.11.2.1126.
26. Kathleen Moriarty Password-Based Cryptography Specification. **2017**.
27. *Understanding Optimizations and Measuring Performances of PBKDF2*; Woungang, I., Dhurandher, S.K., Eds.; Lecture Notes on Data Engineering and Communications Technologies; Springer International Publishing: Cham, 2019; Vol. 27; ISBN 978-3-030-11436-7.
28. Biryukov, A.; Dinu, D.; Khovratovich, D.; Josefsson, S. RFC 9106: Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications. *Internet Research Task Force (IRTF)* **2021**, 1–21.
29. Aljuffri, A.; Huang, R.; Muntenaar, L.; Gaydadjiev, G.; Hamdioui, S.; Taouil, M. The Security Evaluation of an Efficient Lightweight. **2024**, 1–20.
30. Dworkin, M.J. NIST Special Publication 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. *Computer Security, National Institute of Standards and Technology (NIST), published by NIST* **2007**.
31. Medetov, B.; Serikov, T.; Tolegenova, A.; Dauren, Z. Comparative Analysis of the Performance of Generating Cryptographic Ciphers on the Cpu and Fpga. *J. Theor. Appl. Inf. Technol.* **2022**, *100*, 4813–4824.
32. Mouha, N.; Dworkin, M. *NIST Internal Report NIST IR 8459 Ipd Report on the Block Cipher Modes of Operation in the NIST SP 800-38 Series Initial Public Draft*; 2023; ISBN 0000000327459.
33. Lee, J.S.; Kim, D.C.; Seo, S.C. Parallel Implementation of GCM on GPUs. *ICT Express* **2025**, *11*, 310–316, doi:10.1016/j.icte.2025.01.006.
34. Kampanakis, P.; Campagna, M.; Crocket, E.; Petcher, A.; Gueron, S. Practical Challenges with AES-GCM and the Need for a New Cipher. *Third NIST Workshop on Block Cipher Modes of Operation* **2024**, 3.
35. Abdulrahman, R.O.; Hassani, H.; Ahmadi, S. Developing a Fine-Grained Corpus for a Less-Resourced Language: The Case of Kurdish. *arxiv.org* **2019**, 106–109.
36. Montemurro, M.A.; Zanette, D.H. Universal Entropy of Word Ordering across Linguistic Families. *PLoS One* **2011**, *6*, doi:10.1371/journal.pone.0019875.
37. Abdulrahman, R.O.; Hassani, H. A Language Model for Spell Checking of Educational Texts in Kurdish (Sorani). *1st Annual Meeting of the ELRA/ISCA Special Interest Group on Under-Resourced Languages, SIGUL 2022 - held in conjunction with the International Conference on Language Resources and Evaluation, LREC 2022 - Proceedings* **2022**, 189–198.
38. Hamarashid, H.K.; Saeed, S.A.; Rashid, T.A. Next Word Prediction Based on the N-Gram Model for Kurdish Sorani and Kurmanji. *Neural Comput. Appl.* **2021**, *33*, 4547–4566, doi:10.1007/s00521-020-05245-3.
39. A. Rashid, T.; M. Mustafa, A.; Saeed, A.M. A Robust Categorization System for Kurdish Sorani Text Documents. *Information Technology Journal* **2016**, *16*, 27–34, doi:10.3923/itj.2017.27.34.
40. Kathleen, M.; Burt, K.; Adam, R. *PKCS #5: Password-Based Cryptography Specification Version 2.1*; 2017;
41. Alex, B.; Daniel, D.; Dmitry, K.; Simon, J. *Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications*; 2021;

42. Hugo, K.; Pasi, E. *HMAC-Based Extract-and-Expand Key Derivation Function (HKDF)*; 2010;
43. The Unicode Consortium Unicode 17.0 Character Code Charts Available online: https://unicode.org/charts/.
44. François Yergeau *UTF-8, a Transformation Format of ISO 10646*; 2003;
45. Bassham, L.E.; Rukhin, A.L.; Soto, J.; Nechvatal, J.R.; Smid, M.E.; Barker, E.B.; Leigh, S.D.; Levenson, M.; Vangel, M.; Banks, D.L.; et al. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*; Gaithersburg, MD, 2010;
46. Gáll, J.; Gürgez, P.; Horváth, G. Adding an Avalanche Effect to a Stream Cipher Suitable for IoT Devices. **2025**, doi:10.3390/electronics.
47. Mohamed, K.; Pauzi, M.N.M.; Ali, F.H.H.M.; Ariffin, S. Analyse On Avalanche Effect In Cryptography Algorithm. In Proceedings of the Proceedings of the International Conference on Sustainable Practices, Development and Urbanisation (IConsPADU 2021), 16 November 2021, Universiti Selangor (UNISEL), Malaysia; European Publisher, October 31 2022; Vol. 3, pp. 610–618.
48. Upadhyay, D.; Gaikwad, N.; Zaman, M.; Sampalli, S. Investigating the Avalanche Effect of Various Cryptographically Secure Hash Functions and Hash-Based Applications. *IEEE Access* **2022**, *10*, 112472–112486, doi:10.1109/ACCESS.2022.3215778.
49. Shamsa, K.; Saba, I.; Muqaddas, B.; Leila, J. An Efficient and Robust Image Encryption Model Using Hybrid Technique, SHA-224 Hashing, and Shift AES. **2026**.
50. Koulouh, F.; Amine, S.; Es-Sabry, M.; AKKAD, N. EL; Shahin, A.I.; El-Shafai, W. Optimizing Color Image Security Using Hybrid Cryptographic Techniques Based on Sine and Logistic Maps. *Sci. Rep.* **2026**, doi:10.1038/s41598-025-33319-6.
51. Zhang, X.; Chai, Y.; Xiang, S.; Li, S. Research on Image Encryption with Multi-Level Keys Based on a Six-Dimensional Memristive Chaotic System. *Entropy* **2025**, *27*, doi:10.3390/e27111152.
52. Zhiqiang, H.; Rauf, A.; Nazir, A.; Tchier, F.; Aslam, A.; Tola, K.A. Design and Analysis of a Secure Image Encryption Algorithm Using Proposed Non-Linear RN Chaotic System and ECC/HKDF Key Derivation with Authentication Support. *Sci. Rep.* **2025**, *15*, doi:10.1038/s41598-025-23592-w.
53. Wang, F.; Sang, J.; Liu, Q.; Huang, C.; Tan, J. A Deep Learning Based Known Plaintext Attack Method for Chaotic Cryptosystem. **2021**.
54. Hazzazi, M.M.; Shah, D.; Alghamdi, M.; Alkhazi, I.S.; Riaz, N. Chosen-Plaintext Attacks on Image Ciphers Based on Nonlinear Dynamical Systems. *Array* **2026**, 100642, doi:10.1016/j.array.2025.100642.
55. Babu, P.A.; Thomas, J.J. Freestyle, a Randomized Version of ChaCha for Resisting Offline Brute-Force and Dictionary Attacks. **2018**.
56. Ziyad, Hazim.A. *Central Kurdish Linguistic Text Cryptography Dataset (CKLTCD)*; 2026.