INTRODUCTION

In the digital era, where vast amounts of data are transmitted over public and private networks every second, safeguarding this information from unauthorized access has become a paramount concern. As cyber threats continue to evolve, traditional cryptographic techniques often struggle to keep pace with the rising complexity of attacks. Cryptography, the science of encrypting and decrypting data, is essential for ensuring data security by preventing malicious actors from intercepting and exploiting sensitive information. However, with the increasing sophistication of attacks on conventional cryptographic methods, modern approaches are necessary to enhance data protection.

Recent advancements in artificial intelligence (AI) have opened new avenues for improving cryptographic techniques. Artificial Neural Networks (ANNs), inspired by the structure and function of the human brain, offer a dynamic and adaptable solution for encryption and decryption processes. In particular, the Tree Parity Machine (TPM) model has emerged as a promising tool in cryptography, enabling more robust key generation mechanisms. By leveraging ANNs and TPM, cryptographic systems can generate dynamic secret keys that are unique for each session, providing an additional layer of security that is resistant to conventional attacks such as brute force or key recovery.

This report delves into a modern cryptographic algorithm that utilizes ANN and the Tree Parity Machine model to create a symmetric key encryption system. The algorithm dynamically generates single-digit secret keys, ensuring that each key is distinct for every execution, thereby enhancing both the reliability and security of data transmission. The primary objective of this report is to analyze the key generation, encryption, and decryption processes of the proposed algorithm, evaluate its performance compared to traditional cryptographic methods, and highlight its potential applications in securing data over public networks.

OBJECTIVES

1. **To analyze the efficiency of the proposed ANN-based cryptographic algorithm**: Examine how the Tree Parity Machine (TPM) approach enhances the performance of symmetric key generation and encryption.
2. **To evaluate the security improvements**: Assess the dynamic nature of the secret key generation process and its impact on data protection, especially in comparison to traditional cryptographic methods.
3. **To compare execution time and complexity**: Investigate the algorithm's execution time and computational complexity in relation to existing algorithms, highlighting its advantages in real-time applications.
4. **To understand the role of Artificial Neural Networks (ANN) in cryptography**: Explore how ANN and the Tree Parity Machine model contribute to the robustness and reliability of modern cryptographic systems.
5. **To determine the practical applicability of the algorithm**: Assess whether the proposed cryptographic technique can be effectively implemented in real-world scenarios, particularly in public network communications.

METHODOLOGY

The encryption and decryption methodologies are broken down into specific steps, highlighting the role of each operation in securing and retrieving data.

Encryption Process

The encryption methodology consists of multiple transformation steps that alter the plain text, convert it to ciphertext, and make it unreadable without the correct key.

1. **Secret Key Generation**:
   - The Tree Parity Machine (TPM) model is used to generate a unique, single-digit secret key for each session. This key is dynamically generated and synchronized between the sender and receiver, ensuring a unique encryption-decryption pair for every communication session.
2. **Convert Text to ASCII**:
   - The input plain text is converted to ASCII values, which serve as a base for further transformations. ASCII conversion allows for mathematical manipulations at a numerical level.
3. **Swap Cases**:
   - All uppercase characters in the text are converted to lowercase, and vice versa. This step adds an additional layer of complexity to the ciphertext, making it more challenging for attackers to reverse-engineer the original text.
4. **Replace Spaces**:
   - Any spaces in the text are replaced by random ASCII values between 128 and 255. This substitution disguises word breaks, making it difficult for unauthorized parties to decipher words based on spacing patterns.
5. **Apply Secret Key to Even Indices**:
   - The single-digit secret key generated by the TPM is added to the ASCII values of characters at even indices in the text. By targeting only specific indices, the

encryption process retains randomness, making pattern recognition challenging for intruders.

6. **XOR Operation**:
   - Each ASCII value in the modified text undergoes an XOR operation with the secret key. This binary operation alters the values significantly, creating encoded text that appears as random characters.
7. **UTF-8 Encoding**:
   - The final transformed text is encoded in UTF-8, allowing it to be stored, transmitted, or displayed without loss of information. UTF-8 encoding is essential for compatibility across various systems and platforms.

Decryption Process

The decryption methodology reverses each transformation step from the encryption process, reconstructing the original plain text.

1. **Decode UTF-8**:
   - The encrypted text is converted back from UTF-8 encoding to the ASCII-based format used during encryption. This step prepares the text for reversal of previous transformations.
2. **XOR Operation**:
   - Each ASCII value is subjected to an XOR operation with the secret key. This step reverses the initial XOR operation in encryption, returning each character closer to its original ASCII form.
3. **Subtract Key from Even Indices**:
   - The secret key value is subtracted from characters at even indices, reversing the addition step in the encryption process. This step helps realign the ASCII values with the original text structure.

4. **Replace ASCII with Spaces**:
    - ASCII values corresponding to random characters that were originally spaces are reverted to actual space characters (ASCII 32). This restores the word boundaries in the text.
5. **Swap Cases**:
    - The final decryption step involves switching the cases of characters back to their original forms, completing the process and restoring the original plain text.

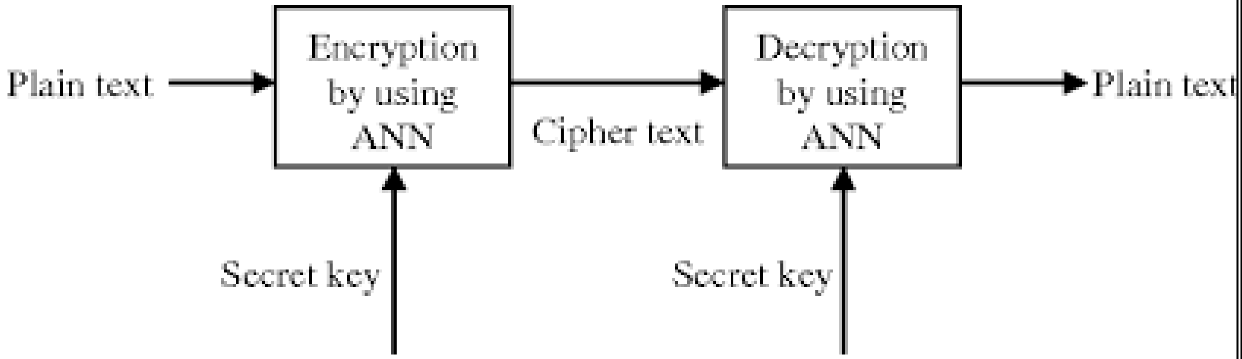

Fig. 1 Block diagram representation

ALGORITHM

The following sections present the stepwise structures for both encryption and decryption algorithms.

Encryption Algorithm Structure

1. **Input plain text** and convert each character to its ASCII equivalent.
2. **Swap cases** of characters (uppercase to lowercase, and vice versa).
3. **Replace spaces** with random ASCII values between 128-255 and store the substituted values.
4. **Add secret key** to characters at even indices in the ASCII sequence.
5. **Perform XOR operation** between each ASCII value and the secret key.
6. **Encode** the modified text using UTF-8 encoding to produce the final ciphertext.

Decryption Algorithm Structure

1. **Decode the ciphertext** from UTF-8 encoding back to ASCII values.
2. **Apply XOR** between each ASCII value and the secret key to reverse the XOR operation.
3. **Subtract the secret key** from characters at even indices to restore their original values.
4. **Replace ASCII placeholders** with spaces to recover word boundaries.
5. **Swap cases** of characters to complete the reconstruction of the original plain text

Manual Example

Suppose we want to encrypt the text **"Hi"** with a secret key of **3**.

1. **Encryption**:
   - Convert **"Hi"** to ASCII: H = 72, i = 105.
   - Swap cases: h = 104, I = 73.
   - Apply the secret key of 3 to even indices: [107, 73].
   - Perform XOR with the key 3: [104 ^ 3 = 107, 73 ^ 3 = 74], yielding [107, 74].
   - Convert to UTF-8 encoded ciphertext: **"kJ"**.
2. **Decryption**:
   - Decode the ciphertext **"kJ"** back to ASCII values: [107, 74].
   - XOR each value with 3: 107 ^ 3 = 104, 74 ^ 3 = 73, yielding [104, 73].
   - Subtract the key from even indices: [104 - 3 = 72, 73].
   - Swap cases to get back the original text: **"Hi"**.

This example demonstrates how the transformations obscure the original text and how reversing each step restores it.

ADVANTAGES

- **Dynamic Secret Key**: The single-digit secret key is generated dynamically, providing a unique key for each session and enhancing security against brute-force attacks.
- **Linear Time Complexity**: With a time complexity of O(n)O(n)O(n), this algorithm ensures efficient performance, making it suitable for real-time encryption needs.
- **Lightweight and Flexible**: Due to its simplicity and minimal processing requirements, this algorithm can be implemented on low-power devices like IoT hardware.

DISADVANTAGES

- **Limited Key Range**: The single-digit key limits the algorithm's complexity, potentially reducing its security in scenarios requiring highly complex encryption.
- **Symmetric Key Dependency**: Both the sender and receiver must share the same key, which requires secure transmission of the key, adding a potential vulnerability point.
- **Space Complexity**: Intermediate steps in encryption may require additional memory for storing characters, especially for large data sets.

APPLICATIONS

The proposed cryptographic algorithm has several practical applications due to its balance between security and efficiency:

1. **Secure Messaging**: Used in messaging apps to protect text messages exchanged over unsecured channels, ensuring only the intended recipient can decrypt them.
2. **Cloud Data Security**: Encrypts files before uploading them to cloud storage, protecting data from unauthorized access in case of breaches.
3. **IoT and Edge Devices**: Provides lightweight encryption suitable for resource-constrained environments, protecting sensitive data exchanged between IoT devices and networks.
4. **Embedded Systems**: Used in embedded systems where computational resources are limited, offering a practical solution for encrypting sensitive information.

CONCLUSION

The rapid growth of internet-based communication has increased the need for robust and efficient cryptographic techniques to secure sensitive information. The cryptographic algorithm discussed in this report leverages Artificial Neural Networks (ANN) through the Tree Parity Machine (TPM) model to introduce a dynamic symmetric key generation process. This innovation enhances the reliability and security of data transmission by ensuring that a unique secret key is generated for each session, making it highly resistant to attacks.

Through a detailed analysis, it is clear that the proposed algorithm significantly improves upon traditional methods in terms of execution time and complexity. The algorithm's use of XOR operations, dynamic key generation, and its integration with modern cryptographic principles ensures a high level of security, especially for applications involving public networks. The experimental comparisons demonstrate that the TPM-based approach outperforms existing algorithms in efficiency, while its dynamic nature further strengthens the system's defense against unauthorized access.

In conclusion, the use of ANN in cryptography, as demonstrated by this algorithm, represents a promising direction for future research and development in secure communication. Further optimizations could reduce complexity to O(log n), making this approach even more applicable in real-time scenarios. The algorithm stands as an innovative solution to the challenges of modern data security, offering both robustness and adaptability in a rapidly evolving digital landscape.

REFERENCE

1. "An Artificial Neural Network Technique of Modern Cryptography" S. K. Pal1 , B. Datta , A. Karmakar JOURNAL OF SCIENTIFIC RESEARCH Publications, www.banglajol.info/index.php/JSR
2. B. F. Cruz, K. Domingo, F. D. Guzman, and J. Cotiangco, Int. J. Comp. Sci. Mobile Comput. 6, 133 (2017). https://doi.org/10.13140/RG.2.2.36392.72969
3. S. Kumari, Int. J. Engg. Comp. Sci. 6, 20915 (2017). https://doi.org/10.18535/ijecs/v6i4.20
4. P. Patil, P. Narayankar, D. G. Narayan, and S. M. Meena, Proc. Comput. Sci. 78, 617 (2016). https://doi.org/10.1016/j.procs.2016.02.108
5. A. D. Dwivedi, Sensors 21, 1 (2021). https://doi.org/10.3390/s21175744
6. F. Thabit, S. Alhomdy, H. A. Al-Ahdal, and S. Jagpat, Global Transitions Proc. 2, 91 (2021) https://doi.org/10.1016/j.gltp.2021.01.013
7. K. Balaji, and S. S. Manikandasaran, J. Sci. Res. 14, 153 (2022).
8. S. K. Pal, B. Datta, and A. Karmakar - Proc. Emerging Technologies in Data Mining and Inform. Security (Springer Nature, Singapore) 3 (2021) pp. 47-56. https://doi.org/10.1007/978- 981-15-9774-9_5
9. J. M. Padilla, U. M. Baese, and S. Foo, EURASIP J. Informat. Security 3 (2018). https://doi.org/10.1186/s13635-018-0073-z
10. S. A. Dar, Int. J. Modern Trends Engg. Res. 5, 73 (2018). https://doi.org/10.21884/IJMTER.2018.5013.DAYGS