An Efficient Unicode encoded in UTF-16 text cryptography based on the AES algorithm

Sushmit Jahan Rose
Computer Science and Engineering
Varendra University, Bangladesh
sushmitrose@gmail.com

Umut Ozkaya
Electrical Electronics Engineering
Konya Technical University, Turkiye
uozkaya@ktun.edu.tr

Sabina Yasmin
Computer Science and Engineering
Varendra University, Bangladesh
sabina@vu.edu.bd

Suraiya Jabin
Computer Science and Engineering
Varendra University, Bangladesh
suraiyajabin765@gmail.com

Robiul Hasan
Computer Science and Engineering
Varendra University,Bangladesh
mdshantocse@gmail.com

Elias Kabir
Computer Science and Engineering
Varendra University, Bangladesh
anujkabir777@gmail.com

July 14, 2023

Abstract

Data security and secrecy from unwanted applications are the subjects of the science known as cryptography. The advanced encryption standard algorithm is the most used and secure algorithm to encrypt data. The AES algorithm is based on the symmetric algorithm and uses a single key to encrypt and decrypt data. The AES algorithm uses 128 bits length of plain text with 128 bits, 192 bits, or 256 bits key size to encrypt data. Latin script uses ASCII codes, and a single byte represents each alphabet. Therefore, in 128 bits AES encryption, 16 characters can be encrypted each time. The other language script used the Unicode standard to represent their alphabets. In Unicode, at least 2 bytes are required to represent a character. Therefore, eight characters can be encrypted at a time. Also, there is no available S-box for Unicode characters. Therefore, a modified algorithm is proposed for Unicode to encrypt data. To use the AES algorithm in Unicode data, we need to convert the Unicode into character encoding, such as UTF-16. Nevertheless, In UTF-16, some Unicode characters have similar recurrent values. This paper demonstrates a modified AES algorithm to encrypt the Unicode script to reduce time complexity.

***K*eywords** Cryptography · Unicode · AES algorithm · Galois Field

1 Introduction

Internet communication plays an enormous role in transferring data in various fields. Data might be transmitted through insecure channels and can be hacked by unauthorized persons. Securing the data by encrypting it with the best possible cryptography algorithm is essential. Advanced encryption standard is so far the most secure algorithm to encrypt data. The AES algorithm performs on 128 bits of plain text data, represented in the ASCII code or single byte Unicode of Latin script. However, other language scripts such as Bengali, Arabic, and Mandarin use their script. This script represents by the system in Unicode with at least 2 bytes. The Unicode is needed at least 2 bytes to represent a character. This paper uses the AES

algorithm to encrypt the Unicode script efficiently. In the AES, there can be 128 bits, 192 bits, or 256 bits blocks of key size to encrypt 128 bits of plain text [1]. In the AES, 16 characters can be encrypted as Latin characters are represented with ASCII code or single-byte Unicode, with one byte for each character. Nevertheless, to represent other scripts, Unicode encoded in UTF-16 needs at least 2 bytes to represent each character. To encrypt Unicode in AES, we need to encode the character in UTF-16. After the encoding, eight characters can be represented as 128 bits. Therefore, 8 Unicode characters can be encrypted each time using the AES algorithm. In the Unicode 2 bytes range, all the characters have one thing in common, and they start with an identical hexadecimal value. As in the AES algorithm, the plain text block is divided into one byte. So each character of Unicode will be represented in 2 blocks. The identical hexadecimal number will be repeated 8 times. The proposed algorithm reduces the repeated values and efficiently encrypts the text. The proposed algorithm reduced the first two hexadecimal values of the Unicode. So the total number of bits was reduced and made possible to encrypt a total of 16 alphabets of the Unicode (2 bytes range) script at a time in the AES algorithm. Finally, the identical byte is stored. After the decryption procedure, the stored identical bytes are added after each byte of the decrypted data.

2 Literature Survey

Unicode is a standard method used to encode symbols and characters from diverse languages and scripts to be processed by computers. It conforms to ISO/IEC 10646-1:2000 and permits the definition of 65536 characters utilizing 16-bit encoding. This standard encompasses a wide variety of forms including letters, digits, symbols, and characters from various languages spoken worldwide, and it also includes technical and mathematical symbols, punctuation marks, arrows, and numerous other symbols. Moreover, Unicode has enough capacity to accommodate the characters required for high-quality typesetting [1]. The Advanced Encryption Standard (AES) is a widely used cryptographic algorithm that effectively secures digital information. It became the encryption standard in 2000 and has since gained global popularity. The AES's main objective is to provide a strong encryption algorithm that can protect US government documents for at least two decades. Furthermore, it is ideal for securely transmitting large amounts of data [2]. The US Federal Information Processing Standards (FIPS) approved the Advanced Encryption Standard (AES) as a cryptographic algorithm to protect digital data. AES uses a symmetric block cipher that allows for both encryption and decryption of data. When encrypting data, it is converted into ciphertext, while decryption reverses the process by converting the encrypted message back into plaintext. AES utilizes cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in 128-bit blocks. [1] Encryption algorithms can be broadly classified into two categories, symmetric and asymmetric. In symmetric algorithms, the same key is used for both encryption and decryption, and it can be shared between multiple parties to maintain secrecy. However, a significant drawback of this approach is that both parties have access to the same secret key. AES is a well-known example of a symmetric encryption algorithm. To encrypt Arabic words more effectively, multiple cipher methods are employed in combination. This approach generates more intricate outputs compared to those obtained from using individual cipher methods. The proposed encryption technique consists of five stages, including setup, controlling, encryption, moving, and decryption [3]. By combining Tamil alphabet letters with digits 1 to 9, a Galois field of 2^8 with an irreducible polynomial of degree 8 is formed. This encoding method can be used to implement the AES algorithm, replacing its pre-round transformation module, by converting Tamil texts into hexadecimal states. According to empirical evidence, this encoder enhances the cryptographic strength of the AES algorithm at every stage of the encryption process. To measure the strength of the AES ciphers, the run test scores of the bit sequences are compared to those of the English language [4]. To store a character, the amount of data required depends on its encoding, as not all languages can be represented using a straightforward one-to-one mapping. If a language requires more options, additional data is necessary to encode a character. In most encodings, the first 7 bits are reserved for ASCII characters, providing 128 values. The remaining 8th bit or an additional 128 values is reserved for representing other characters such as accented characters, Asian languages, Cyrillic, and more. Due to the vast array of characters that need to be represented, it becomes apparent that a single byte is insufficient to encode them all. The ASCII encoding used to represent Armenian letters with a single byte is no longer considered the best encoding method. Instead, UTF-8, a Unicode transfer encoding, is preferred as it uses a variable number of bytes to represent characters. For instance, the English letter "a" is represented by a single byte, whereas accented characters like "á" require two bytes. Similarly, the Armenian letter ayb " " requires two bytes in UTF-8. However, there are some characters, such as the Bengali letter " ", that need 4 bytes in UTF-16, which may cause issues in certain situations.

3 Methodology

3.1 Key Generation

The AES algorithm is a symmetric block cipher algorithm, 128 bits of data at a time using a 128, 192, or 256-bit key. The ASCII or single-byte Unicode is used to represent the Latin alphabet. The Latin alphabet is represented using 8 bits or two hexadecimal values. Therefore, 16 alphabets of Latin are encrypted at a time in AES. Bengali, Syriac, Oriya, Arabic, and other languages use their script as written form, and it is represented in computer codes of Unicode. In Unicode, at least 16 bits are used to represent an alphabet. The UTF-16 can define different script ranges from 0xD800 to 0xDBFF [5]. In Unicode, the alphabet is represented by 2 to 4 bytes or 16 bits to 32 bits. When a conventional AES algorithm is implemented to encrypt, Unicode's script is encoded in UTF-16 or UTF-16, and 8 or 4 alphabets can be encrypted at a time. The same message in other languages needs double or multiple times to be encrypted then the Latin script. As in AES encryption, 128-bit plain text is split into 16 bits to be represented in a 4-times-4 2-D matrix. The script is represented in Unicode with at least 2 bytes. Most have a common byte 0x or (0x=hexadecimal identical value) for each character. Such as the Unicode of Bengali script has 128 points [5], all starting with hexadecimal point 09. Also, the Unicode of the Arabic script has 256 points, and all of them are started with hexadecimal point 06 [5]. In the proposed algorithm, some values are identical in the hexadecimal code and can be subtracted to reduce the time to encrypt the message in AES. Later the values can be added to the encrypted text. Furthermore, the encrypted message may not be in the range of the expected languages Unicode. The algorithm proposed to create the encrypted message into the perimeter of the desired language, Unicode. It will give an extra layer of safety while encrypting a message in Unicode.

3.2 Key expansion in Unicode

In AES, the number of rounds *(Nr)* depends on the key size K; for every 128 bits, 192 bits, and 256 bits key, the *Nr* is 10, 12, or 14, respectively [6]. Therefore the key expansion must take place to generate *Nb(Nr+1)* words as the algorithm requires. The initial key w[i] contains an array of bytes or 32 bits words, and the value of i is the number of columns *Nk*. The array of bytes is represented by $a_0, a_1, a_2, ..., a_{15}$. The w[i] can be represented as $w[0] = a_0, a_1, a_2, a_3$
$w[1] = a_4, a_5, a_6, a_7$
$w[2] = a_8, a_9, a_{10}, a_{11}$
$w[3] = a_{12}, a_{13}, a_{14}, a_{15}$.

A cyclic permutation performs on the last column of w[i] by the function **RotWord()**, the function takes words $a_{12}, a_{13}, a_{14}, a_{15}$ and gives an output $a_{13}, a_{14}, a_{15}, a_{12}$. After **RotWord()**, the output is given to **SubWord()** function to apply S-box operations (g(w[i])). The round constant word array, **Rcon[i]**, contains the values given by $[x_{\texttt{\{i-1\}}}, \texttt{\{00\}}, \texttt{\{00\}}, \texttt{\{00\}}]$, with $x_{\texttt{\{i-1\}}}$ being powers of x (x is denoted as 02) in the field $GF(2^8)$(note that i starts at 1, not 0). To get the column of the first round key, an XOR operation performs on **Rcon[i]**, w[0], and the output of the S-box operations. Each subsequent word, w[i], is obtained by performing an XOR operation on the previous word, w[i-1], and the word that appears **Nk** positions earlier, w[i-**Nk**]. There are two ways to generate the initial key and expand the key by using Unicode values. Where **Nk** is 4, 128 bits of the key can be represented by 8 alphabets as each alphabet represent 2 bytes. The array of bytes $a_{[i]}$is the initial key where a_0 and $a_{[i\%2==0]}$ is identical values (Unicode 0x). In UTF-16, the script, which consists of 2 bytes for each character with the 0x identical value, can be written as $0xa_0, 0xa_1, 0xa_3, ..., 0xa_{[i]}$ (Algorithm 1). Such as, the Bengali alphabets consist of 2 bytes represented by Unicode can be written as $09a_0, 09a_1, 09a_3, ..., a_{[i]}$ ranges from 0980 to 09FF in hexadecimal. Also, Syriac alphabets consist of 2 bytes represented by Unicode and can be written as $07a_0, 07a_1, 07a_3, ..., 07a_{[i]}$ ranging from 0700 to 074F in hexadecimal. The other way is the identical value $a_{[i+1]}$ needs to remove and replace the value with a subsequent value (Algorithm 2). These two types of keys later work as the function AddRoundKey of the algorithm.

Algorithm 1
Step 1: Start.
Step 2: Insert Unicode with 2 bytes for each character (128 bits, 192 bits, or 256 bits.)
Step 3: Split 2-byte Unicode into 1 byte $a_{[i]}$

- $a_{[i]} = a_0, a_1, a_2, ..., a_{15}$(**Nk**=4)
- $a_{[i\%2==0]} = 0x$ (identical value)

Step 4: Cyclic permutation **RotWord()**

- **RotWord()**: For every initial 32 bit word w[i]
- If $w[i] = a_{12}, a_{13}, a_{14}, a_{15}$ then $w[i] = a_{13}, a_{14}, a_{15}, a_{12}$(**Nk**=4).

Step 5: w[i]input into **SubWord()** function of S-box ($g(w_{[i]})$).
Step 6: To generate the first column of the round key (w[i]). XOR on **Rcon[i]**, w[0] and (g(w[i])).

- **Rcon[i]**= [x i-1,00,00,00] of $GF(2^8)$

Step 7: The subsequence w[i], is equal to the XOR w[i-1], and the word Nk positions earlier, w[i-Nk].
Step 8: End.

Algorithm 2
Step 1: Start.
Step 2: Insert Unicode with 2 bytes for each character (256 bits, 384 bits, or 512 bits.)
Step 3: Remove identical value and replace every $a_{[i]} = a_{[i+1]}$.

- Round 256 bits, 384 bits, or 512 bits into 128 bits, 192 bits, or 256 bits, respectively.
- $a_{[i]} = a_0, a_1, a_3, ..., a_{15}$ (**Nk**=4)

Step 4: Cyclic permutation **RotWord()**

- **RotWord()**: For every initial 32 bit word w[i]
- If $w[i] = a_{12}, a_{13}, a_{14}, a_{15}$ then $w[i] = a_{13}, a_{12}, a_{14}, a_{15}$ (**Nk**=4).

Step 5: w[i] input into **SubWord()** function of S-box (g(w[i])).
Step 6: To generate the first column of the round key (w[i]). XOR on **Rcon[i]**, w[0] and (g(w[i])).

- **Rcon[i]**= [x i-1,00,00,00] of $GF(2^8)$

Step 7: The subsequence w[i], is equal to the XOR w[i-1], and the word **Nk** positions earlier, w[i- **Nk**].
Step 8: End.

3.3 AES Encryption on Unicode

The AES algorithm processes 128 bits of data blocks with a key length of 128 bits, 192 bits, or 256 bits. The AES algorithm is performed in 4-times-4 bytes arrays called states. This 4-times-4 array is added to the initial key, called the initial round. Depending on the key length, the round of encryption varies from 10, 12, and 14 rounds. After the initial round, each round contains four primary operations SubBytes, ShiftRows, MixColumns, and AddRoundKeys [6]. As in the Latin script, ASCII or single-byte Unicode has been used to interpret a character. Unicode represents other language scripts; it takes more than a byte to implement them in the computer system. Therefore Unicode uses 2 or 4 bytes to execute different language scripts worldwide. If Latin alphabets are encrypted using the AES algorithm at a time, 16 characters can be encrypted, as 16 characters can be interpreted as 128 bits or 4 words. Nevertheless, in Unicode, if the script is represented by 2 bytes, only 8 characters, and for the script which has been described by 4 bytes, only 4 characters can be encrypted at a time. Therefore processing time of encryption in Unicode is slower than in ASCII representation. It is needed double or multiple times to encrypt in Unicode than ASCII. Also, while encrypting a message in Unicode using the AES algorithm, the encrypted message sometimes crosses the boundary of the range given to a specific language's script in Unicode. In this paper, a modified approach is presented to encrypt Unicode efficiently.

3.4 The Main Algorithm

The AES algorithm process in bytes $(b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0)$,and these bits are interpreted as finite field elements of $GF(2^8)$ as $b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$ [6]. These bytes are represented

in an array of bytes $(a_0, a_1, a_2 ... a_{15})$. The AES algorithm can encrypt 128 bits at a time. Therefore 128 bits are represented in a 2-D array 4-times-4 array as the input array (Table. 1). This input array is then given to a state called state array (Table. 2) to be processed in the AES.

$\mathbf{in_0}$	$\mathbf{in_4}$	$\mathbf{in_8}$	$\mathbf{in_{12}}$
$\mathbf{in_1}$	$\mathbf{in_5}$	$\mathbf{in_9}$	$\mathbf{in_{13}}$
$\mathbf{in_2}$	$\mathbf{in_6}$	$\mathbf{in_0}$	$\mathbf{in_4}$
$\mathbf{in_3}$	$\mathbf{in_7}$	$\mathbf{in_1}$	$\mathbf{in_5}$

Table 1: Input bytes

$\mathbf{s_{0,0}}$	$\mathbf{s_{0,1}}$	$\mathbf{s_{0,2}}$	$\mathbf{s_{0,3}}$
$\mathbf{s_{1,0}}$	$\mathbf{s_{1,1}}$	$\mathbf{s_{1,2}}$	$\mathbf{s_{1,2}}$
$\mathbf{s_{2,0}}$	$\mathbf{s_{2,1}}$	$\mathbf{s_{2,2}}$	$\mathbf{s_{2,3}}$
$\mathbf{s_{3,0}}$	$\mathbf{s_{3,1}}$	$\mathbf{s_{3,2}}$	$\mathbf{s_{3,3}}$

Table 2: State array

$\mathbf{out_0}$	$\mathbf{out_4}$	$\mathbf{out_8}$	$\mathbf{out_{12}}$
$\mathbf{out_1}$	$\mathbf{out_5}$	$\mathbf{out_9}$	$\mathbf{out_{13}}$
$\mathbf{out_2}$	$\mathbf{out_6}$	$\mathbf{out_0}$	$\mathbf{out_4}$
$\mathbf{out_3}$	$\mathbf{out_7}$	$\mathbf{out_1}$	$\mathbf{out_5}$

Table 3: Output bytes

Each state array contains 4 words interpreted as,
$w_0 = s_{0,0}s_{1,0}s_{2,0}s_{3,0}$
$w_2 = s_{0,2}s_{1,2}\, s_{2,2}s_{3,2}$
$w_1 = s_{0,1}s_{1,1}s_{2,1}s_{3,1}$
$w_3 = s_{0,3}s_{1,3}s_{2,3}s_{3,3}$
In Unicode, if 128 bits are given as input bytes, the number of characters may vary from 4 to 8 characters. So each character given as input is needed to be split into a single byte. If the characters use 2 bytes in Unicode, then one character is divided into two boxes of the input array (Table. 1). In Unicode, given input bytes contain identical bytes in_0 and $in^{th}_{[i\%==0]}$ positions (Table. 4). Let these bytes be called τ. Therefore, the input array is in (Table. 4) (Table.5).

τ	τ	τ	τ
$\mathbf{in_1}$	in_5	in_9	in_{13}
τ	τ	τ	τ
$\mathbf{in_3}$	in_7	in_{11}	in_{15}

Table 4: Input bytes with τ

τ	τ	τ	τ
$\mathbf{s_{1,0}}$	$\mathrm{s}_{1,1}$	$\mathrm{s}_{1,2}$	$\mathrm{s}_{1,2}$
τ	τ	τ	τ
$\mathbf{s_{3,0}}$	$\mathrm{s}_{3,1}$	$\mathrm{s}_{3,2}$	$\mathrm{s}_{3,3}$

Table 5: State array with τ

This state array is given to four different functions of AES called SubBytes, ShiftRows, MixColumns, and AddRoundKeys, respectively. However, in this case, the time dependency is more significant than encrypting the message in ASCII code. Therefore in this modified algorithm, the input bytes are given to a function called ReduceByte() function before entering the main AES algorithm. The main AES functions encrypt the message, but the message is reduced in bit size. Therefore a function called RecoverBytes() is used to add the identical bytes τinto the encrypted message.

3.4.1 SpaceSubstitute function

Even though every language has its script but some of the UTF-16 values are the same in all languages. Such as, space has the UTF-16 value of "0020" for all languages. Also comma sign "002C" does not appear in the Bengali Unicode range [5]. To obtain identical bytes τ, we need to substitute the space, comma, and other bytes with a corresponding byte that is in the Unicode range of the language that we are working with. The substitute byte should not be a common Unicode to the language. Rather, the unused Unicode can be used to substitute the space and comma. For example, In Bengali "ঌ" UTF-16 encoded "09BC" and "ঽ" encoded "09F6" does not have any use in the Bengali language [5][7][8]. For Hindi script, the space also can be converted to "ঌ" UTF-16 encoded "09BC" as they are in a similar range. Also, the comma can be converted to "ঽ" encoded "09F6". To obtain τ value for Tamil and Oriya, Tamilian space can be substituted by Oriya "0B71", and Oriya space can be substituted by Tamilian month sign "0BF3" as these two languages' Unicode start with the same τ value "0B" [5]. Similarly, **SapceSubstitute()** can be used for Syriac and Thaana, Gurumukhi and Gujarati, Telegu and Kannada, Hiragana and Katakana, and other pairs.

3.4.2 SpaceSubstitute() algorithm (Bengali)

Algorithm
Step 1: Start
Step 2: Input a character (2 bytes in Unicode) into a variable (Sp)
Step 3: If Sp="0020" then

Step 4: Set Sp= "09BC" in hexadecimal
Step 5: Else if Sp="002C" then
Step 6: Set Sp= "09F6" in hexadecimal
Step 7: Else set Sp=Sp
Step 8: End.

3.4.3 ReduceByte function

In ASCII codes, 16 alphabets can be represented by 128 bits. Unicode is needed 2 or more bytes to represent an alphabet, so 16 alphabets in Unicode means at least 256 bits or more. The ReduceByte transformation function takes 16 alphabets of Unicode, regardless of bits, to represent those alphabets. The ReduceByte() identifies the identical byte τ of each character and subtracts the value from $\tau 00$ in hexadecimal. If any alphabet in Unicode has 2 bytes to represent, 256 bits are needed for 16 alphabets also represented as $a_0, a_1, a_2, a_3, \ldots\ldots a_{31}$ (Table. 6) where a_0 and $a_{[i\%2=0]} = \tau$ or identical bytes. So this can be represented as τ, $a_1, \tau, a_3, \ldots\ldots, \tau, a_{31}$ (Table. 7). Therefore, these 256 bits are given to ReduceByte(), which reduces τ (128 bits) and gives 128 bits (Table. 8) output as the input array (Table. 9).

a_0	a_8	a_{16}	a_{24}
a_1	a_9	a_{17}	a_{25}
a_2	a_{10}	a_{18}	a_{26}
a_3	a_{11}	a_{19}	a_{27}
a_4	a_{12}	a_{20}	a_{28}
a_5	a_{13}	a_{21}	a_{29}
a_6	a_{14}	a_{22}	a_{30}
a_7	a_{15}	a_{23}	a_{31}

Table 6: Initial array of bytes

τ	τ	τ	τ
a_1	a_9	a_{17}	a_{25}
τ	τ	τ	τ
a_3	a_{11}	a_{19}	a_{27}
τ	τ	τ	τ
a_5	a_{13}	a_{21}	a_{29}
τ	τ	τ	τ
a_7	a_{15}	a_{23}	a_{31}

Table 7: Common bytes a_0 and $a_{[i\%2==0]} = \tau$

a_0	a_4	a_8	a_{16}
a_1	a_5	a_9	a_{17}
a_2	a_6	a_{10}	a_{18}
a_3	a_7	a_{11}	a_{19}

Table 8: ReduceByte() applied

3.4.4 ReduceByte() algorithm

Algorithm
Step 1: Start
Step 2: Input a character (2 bytes in Unicode)
Step 3: Split the bytes into individual single-byte units ($a_{[i]}$ and $a_{[i+1]}$)
Step 4: Set $a_0 = \tau 00$ in hexadecimal
Step 5: Subtract the input with $\tau 00$ in hexadecimal
Step 6: Remove $a_{[i]}$ and set $a_{[i+1]}$ as $a_{[i]}$
Step 7: Store $a_{[i]}$ into the stack
Step 8: End.

3.4.5 SubBytes function

The input bytes from ReduceByte() are given to SubByte() of the AES algorithm based on a non-linear S-box to substitute a byte in the state for another byte. This stage fulfilled Shannon's principle of confusion and diffusion. The S-box has been calculated over Galois Field (2^8).

in_0	in_4	in_8	in_{12}
in_1	in_5	in_9	in_{13}
in_2	in_6	in_{10}	in_{14}
in_3	in_7	in_{11}	in_{15}

Table 9: Array of bytes as $in_{[i]}$

3.4.6 ShiftRows function

During the ShiftRows() transformation, the bytes in the last three rows of the State are shifted cyclically by varying numbers of bytes (or offsets). The first row, r = 0, remains unshifted. The procedure for this step is as follows:
$S'_{r,c} = S_{r,(c+shift(r,Nb))modNb}$ for $0 < r < 4$ and $0 <= c < Nb$

3.4.7 MixColumns function

In the MixColumns() transformation, the state array is considered as a matrix of columns, with each column represented as a four-term polynomial that is multiplied by a fixed polynomial, denoted as a(x). The 9^{th} bit of the output is reduced by using the irreducible polynomial of the $GF(2^8)$, which is $x^8 + x^4 + x^3 + x + 1$.

3.4.8 MixColumns function

During the AddRoundKeys() transformation, a round key that's generated by the key generation algorithm is added to the State using a bitwise XOR operation. The round keys generated using Unicode contain two types of data, and as such, these two data types have been used separately as round keys. The AddRoundKeys() can be driven as,
$[s_{0,c}', s_{1,c}', s_{2,c}', s_{3,c}'] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}]\ [w_{round*Nb+c}]$ for $0 <= c < Nb$

3.4.9 Methodology 1: RecoverBytes function

The RecoverBytes function took the encrypted message after AES and added the τ before each output byte $out_{[i]}$. The function takes each of the$out[i]$ bytes one by one and add $\tau 00$ with the given byte.

3.4.10 RecoverBytes() algorithm

Algorithm
Step 1: Start
Step 2: Input an output byte $out_{[i]}$
Step 3: Add $\tau 00$ with each $out_{[i]}$
Step 5: Store $out_{[i]}$ into the stack
Step 6: End.

3.4.11 Methodology 2: StoreidenticalByte() function

The **StoreidenticalByte** function took the identical value τ and stored it. The function takes the identical value and adds the value only one-time in front of the whole encrypted message.

3.4.12 StoreidenticalByte algorithm

Algorithm
Step 1: Start
Step 2: Input an output byte $out_{[i]}$
Step 3: Add 00 from each $out_{[i]} = \tau 00$
Step 5: Store $out_{[i]}$ into the stack
Step 6: End.

3.5 AES decryption on Unicode

The decryption process for the encrypted data of Unicode is followed by the ReduceByte(), Inverse of AES, and the RecoverBytes() functions. The encrypted message of Unicode has 256 bits if the Unicode has been represented by 2 bytes for each character. Therefore, ReduceByte() is applied to the encrypted message and round the encrypted message into 128 bits by removing the τ value. The state then given to InvShiftRows(), InvSubBytes(),InvMixColumns(), AddRoundKey(), and RecoverBytes() for decryption [1]. After the decryption procedure, RecoverSpace() is used to add the spaces and other symbols which has been substituted.

3.5.1 InvShiftRows() function

In decryption, InvShiftRows() is the inverse of the ShiftRows() method. The bytes in the last three rows of the State are shifted cyclically over a range of bytes (offsets). The equation is as follows: $S'_{r,(c+shift(r,Nb))modNb} = S_{r,c}$ for $0 < r < 4$ and $0 <= c < Nb$

3.5.2 InvSubBytes() function

The inverse S-box is applied to each of the bytes of the state to substitute the bytes by the value of the inverse S-box.

3.5.3 InvMixColumns() function

The InvMixColumns() considers the state array as a column-by-column matrix that is treated as four-term polynomials multiplied with a fixed inverse polynomial $a^{-1}(x)$

3.5.4 RecoverSpace() function

RecoverSpace() is the opposite of the SpaceSubstitute() function. As the SpaceSubstitute() function is called to substitute space and other characters which are not in the range of the given language, therefore these should be recovered to obtain the desired output. Hence the RecoverSapce() function is called to obtain the space and other symbols.

3.5.5 RecoverSpace() Algorithm (Bengali)

Algorithm
Step 1: Start
Step 2: Input a character (2 bytes in Unicode) into a variable (Sp)
Step 3: If Sp="09BC" then
Step 4: Set Sp="0020" in hexadecimal
Step 5: Else if Sp="09F6" then
Step 6: Set Sp="002C" in hexadecimal
Step 7: Else set Sp=Sp
Step 8: End.

4 Results

4.1 Key Generation Algorithm 1 and Method 1

To encrypt Unicode encoded in UTF-16, we need a 16-byte long key. If we use a key expansion algorithm where we split the single character to form a 4-by-4 matrix then we need to choose a word with 8 characters. The key used was "ধূমকেতু।" . To remove space "0020" we need to call SpaceSubstitute(). As the chosen key does not have any space it forms a 4-by-4 matrix shown in Table 10.

09	09	09	09
A7	AE	C7	C1
09	09	09	09
C2	95	A4	64

Table 10: 4-by-4 matrix key as $in_{[i]}$

a8	c7	ad	8c
c0	8c	c3	a8
b0	a8	a4	c0
ac	bf	c7	b0

Table 11: The first 16 characters of the plain text
after applying ReduceByte() as $in_{[i]}$

aa	cd	a4	a8
cd	9c	8c	8c
b0	b2	a6	64
9c	bf	bf	64

Table 12: Example Matrix of the first 16 bytes of the
key after applying ReduceByte() as $in_{[i]}$

These split bytes form expanded keys to perform the AES to the text. In the main algorithm, a Space-Substitute() function is called to remove space and substitute the byte with the correspondence byte of the algorithm so that τ value can be obtained. Then a function called ReduceByte() is applied to remove the identical bytes from the plaintext so that 16 characters can be encrypted at a time. The plaintext (Table. 12)

has 112 characters with space. Therefore, total of 7 rounds will require to encrypt the text. After running the SpaceSubstitute() function the space will be replaced (Table.12). Now all of the characters are in the same range of Unicode so that we can easily calculate the τ value. To calculate the τ value ReduceByte() function has been called and it forms the following 4-by-4 matrix for the first 16 characters (Table.11).

To generate the cipher text, convert the key's Unicode character to a hex value and then store the first byte in the τ variable. In the following step, encrypt the hexadecimal value using the remaining byte of each character. Finally, **RecoverBytes()** function has been called to add τ value to the cipher data to obtain the desired output. For the decryption procedure, ReduceByte() function has been called to obtain τ value and the AES decryption procedure has been used. After the decryption procedure RecoverSpace() function is used to recover the spaces and the other symbols (Table 13).

Key	ধূমকেতু।
Plaintext	নীরবে নিভৃতে নীরব স্বপ্নে।আমি বসে থাকি আঁখি বন্ধ করে।।পাখির চিৎকার প্রজ্জলিত দিন। প্রকৃতির ওপার সৌন্দর্য ঘিরে ।।।
Space Substitute	নীরবে৯নিভৃতে৯নীরব৯স্বপ্নে।আমি৯বসে৯থাকি৯আঁখি৯বন্ধ৯করে।।পাখির৯চিৎকার৯প্রজ্জলিত৯দিন। প্রকৃতির৯ওপার৯সৌন্দর্য৯ঘিরে৯।।।
Encrypted text	́ো□□□ ্□□□ক়□ঝে৭চ□ঢ়□ঈব□□ऴ□ত□ব□□□২ऑऱॅছ□ळु□ঝে ্□□□ ू□□□য়□□□□ক়ধ०ॉ□শৌত॥□□ ्□□ ्□□□□□ ्□ ণি□□□ऽ□□ ु□ি৩□□ফঁ□□দ□□८□□৭□□□□□□ ॅब□□□□্
Decrypted text	নীরবে নিভৃতে নীরব স্বপ্নে।আমি বসে থাকি আঁখি বন্ধ করে।।পাখির চিৎকার প্রজ্জলিত দিন। প্রকৃতির ওপার সৌন্দর্য ঘিরে ।।।

Table 13: Result of Key generation algorithm 1 and Method 1

4.2 The key Generation Algorithm 2 and Method 1

The key generation algorithm 2 generated round keys for the encryption process using 128 bits of text. We have not removed the τ value in the key in the earlier algorithm, whereas in this algorithm, we converted the Unicode character of the key to a hex value and removed the first-byte τ of each character. After removing the first identical byte τ, we seemed to have our desired key. Using that key on the plaintext, we obtained the secure cipher text. For this example, we need a 16-byte word "প্রজ্জলিত দিন ।।". To remove the spaces from the key the SpaceSubstitute() function has been called and the key will be "প্রজ্জলিত৯দিন৯।।". Due to SpaceSubstitute() function, all of the characters are in the same range, therefore the identical value τ can be easily calculated. The τ value has been reduced by the ReduceByte() and it forms a 16-byte key (Table. 12). Now the same plaintext will be used for the encryption procedure (Table 14).

Key	প্রজ্জলিত দিন ।।
Key without space	প্রজ্জলিত৯দিন৯।।
Plaintext	নীরবে নিভৃতে নীরব স্বপ্নে।আমি বসে থাকি আঁখি বন্ধ করে।।পাখির চিৎকার প্রজ্জলিত দিন। প্রকৃতির ওপার সৌন্দর্য ঘিরে ।।।
Space Substitute	নীরবে৯নিভৃতে৯নীরব৯স্বপ্নে।আমি৯বসে৯থাকি৯আঁখি৯বন্ধ৯করে।।পাখির৯চিৎকার৯প্রজ্জলিত৯দিন। প্রকৃতির৯ওপার৯সৌন্দর্য৯ঘিরে৯।।।
Encrypted text	ीड़७ः□झ□ ो□ऑ□ग़न□ट□ढ□□□५□ळ□क़□ॡ□□□□□ा□ऑ़□धऩ□□□३ऍ□□□□□ओ□□□ज़□□□□□□ढ□□□ड़छद□ ्□ ढ़□□□ ॄ□□ध□□४ःओढ□□डु□□जऩ□यि□ऊयप□□□छ□□□□७स□ख
Decrypted text	নীরবে নিভৃতে নীরব স্বপ্নে।আমি বসে থাকি আঁখি বন্ধ করে।।পাখির চিৎকার প্রজ্জলিত দিন। প্রকৃতির ওপার সৌন্দর্য ঘিরে ।।।

Table 14: Result of Key generation algorithm 1 and Method 1

5 Conclusion

In this research, we introduced an effective method for implementing advanced encryption standards for long-byte Unicode data encoded in UTF-16. There are different Unicode code points for different language scripts.

Most of the Unicode code points require a minimum of 2 bytes to represent a character. In most languages, the first byte of all the characters has an identical value. As the AES algorithm has 10, 12, or 14 rounds depending on the key length, it is herculean to encrypt extra bytes and is a tedious job. We have introduced ReduceByte(), SpaceSubstitute(), StoreIdenticalByte(), RecoverSpace(), and RecoverBytes() functions to reduce the identical byte and run AES without the identical byte. After the AES procedure, the identical byte is added to the encrypted text. For the decryption procedure, again the identical byte was removed from the encrypted message and decrypted, and again the identical byte was added to get the message. Hence, some of the characters are universal, such as the Unicode of space, so these values were substituted with the value of the desired Unicode range. In conclusion, the proposed algorithm provides a simpler and more efficient approach for implementing the AES algorithm for Unicode code points.

References

[1] Aliea Sabir and Wid Akeel. A new text steganography method by using non-printing unicode characters and unicode system characteristics in english/arabic documents . 3, 08 2012.

[2] Chunxia Tu. Design of an improved method of rijndael s-box. In Minli Dai, editor, *Innovative Computing and Information*, pages 46–51, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[3] A.S.H. Altamimi and A.M. Kaittan. A proposed arabic text encryption method using multiple ciphers. management. *Computing Technology and Information Management*, pages 319–326, 2021.

[4] Shan. Suthaharan. Scientific tamil lexicon: The revelation of cryptographic connection between tamil language and galois field. `https://doi.org/10.31219/osf.io/vb843`, 2022.

[5] The Unicode Consortium. *The Unicode Standard, Version 13.0*. Unicode Consortium, Mountain View, CA, USA, 15 edition, 2022.

[6] James R. Nechvatal James Foti Lawrence E. Bassham E. Roback James F. Dray Jr. Morris J. Dworkin, Elaine B. Barker. Advanced encryption standard (aes), federal inf. process. stds. (nist fips). *National Institute of Standards and Technology, Gaithersburg, MD*, 2001.

[7] Md Khairullah. A novel steganography method using transliteration of bengali text. *Journal of King Saud University - Computer and Information Sciences.*, 31:348–366, 2019.

[8] Mohammad Shahidur Rahman Md. Mamun Hossain, Ahsan Habib. Performance improvement of bengali text compression using transliteration and huffman principle. *International Journal of Engineering Research and Application*, 6:88–97, 2016.