Improving the SMS Security and Data Capacity Using Advanced Encryption Standard and Huffman Compression

LAURENTINUS¹, Harrizki Arie PRADANA², Dwi Yuny SYLFANIA³, Fransiskus Panca JUNIAWAN⁴

¹laurentinus@atmaluhur.ac.id, Department of Computer Science, STMIK Atma Luhur, Jl. Jend. Sudirman, Pangkalpinang, Indonesia
²harrizkiariep@atmaluhur.ac.id, Department of Computer Science, STMIK Atma Luhur, Jl. Jend. Sudirman, Pangkalpinang, Indonesia
³dysylfania@atmaluhur.ac.id, Department of Computer Science, STMIK Atma Luhur, Jl. Jend. Sudirman, Pangkalpinang, Indonesia
⁴fransiskus.pj@atmaluhur.ac.id, Department of Computer Science, STMIK Atma Luhur, Jl. Jend. Sudirman, Pangkalpinang, Indonesia

ABSTRACT
The development of technology that increasingly rapidly has a significant influence on telecommunications. One feature of cellular telephones is SMS (Short Message Service). Still, this facility in the form of SMS (Short Message Service) has a vulnerability in the way of information leakage. Therefore an encryption application is proposed using OOP (Object Oriented Programming) method with Waterfall’s model and (Unified Modeling Language) as tools. To the maintenance of the confidentiality of a message, it required AES (Advanced Encryption Standard) cryptography. Cryptography AES (Advanced Encryption Standard) had functions to encode messages into the form of ciphertext. The results of characters from encryption will usually have more than before because there is an addition to the cipher on each encrypted data. Therefore, Huffman algorithm compression is needed for text compression so that the results of encrypted messages become less. The purpose of this study is to maintain the confidentiality of messages. The results obtained in this study are differences in the number of text encryption characters AES (Advanced Encryption Standard) with the addition of Huffman compression, which is 17.35%. The implementation of this message application was successfully implemented using the AES algorithm and the Huffman algorithm. The message is not can be read by the service center, and the number of characters sent messages can compress into fewer.

Keywords: SMS, cryptography, compression, advances encryption standard, Huffman

1. INTRODUCTION

The mobile phone provides various features, one of the most popular feature is SMS (Short Message Service). SMS is a one of mobile phone service that allows users to communicate with each other by sending short messages in the form of text instantly and for a small fee. SMS is not securing the messages so we are not suppose to send a sensitive or confidential message. Therefore we need an application that can guarantee the security of sending SMS messages because there are millions of unprotected messages. The selection of the right algorithm is another important aspect, seen from the level of importance of the message. A good algorithm makes encryption unpredictable so cannot be disassembled using any method.

AES algorithm is an encryption method that uses asymmetry keys and licenses to use confidentiality [16]. SMS services have used in every field, such as banking for m-banking, e-commerce, and personal transactions.

Figure 1. GSM service structure [15]

In addition to discussing cryptography [8]-[10], this study also discusses data compression, how much encrypted data can be compressed using the huffman algorithm.

A research conducted with a comparative analysis of the performance and security of the most useful algorithms:
RC6 (Rivest Cipher) and RSA (Rivest Shamir Adleman). This study aims to determine the complexity of encryption and decryption algorithms better. For proof, it develops Android-based SMS encryption and decryption prototypes. The results show that all messages can be encrypted using the key generated before sending to the recipient. Encrypted messages also have better security and time performance [1]. Another research proposes an encrypted JPEG image capture scheme. Retrieval is based on the Huffman code in the JPEG bitstream. There are three parties involved, namely the content owner, cloud server, and authorized users. The testing results show that the proposed scheme can ensure confidentiality, integrity, and format compatibility of the JPEG image. Besides taking pictures in terms of various factors the quality of the image is still fairly effective [2]. Huffman is also applied for image compression based on Huffman coding [11]-[12]-[14]. Conducted a research by taking variables based on eye image compression and image compression methods. Compression is performed at the stage of image filtering by wavelet transforms to remove excessive information in the image. The proposed Huffman method is used in the image encoding process. From the test results obtained that the image format size JPEG images can be reduced in the same image effect [3].

Problem Identification and Literature Study

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Prototype</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES Cryptography &amp; Huffman Compression</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design and Implementation Cryptography &amp; Compression into SMS Application base on android</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Huffman Algorithm**

Huffman algorithm is a simple compression algorithm compiled by David Huffman in 1952 [6]. This algorithm included in the type of lossless data compression, which is data compression that does not eliminate or change the number of bytes and stored according to the original data [13].

**METHOD**

In making a message security application with cryptographic algorithm AES (Advanced Encryption Standard) and the Huffman algorithm, the prototype model is used, which is a looping system development model that provides a systematic and structured approach in developing a system. The following are the stages in the prototype model that the author uses in building applications.
AES Algorithm

AES algorithm that was socialized by the National Institute of Standards and Technology (NIST) in November 2001 created as a new encryption standard that developed from the DES (Data Encryption Standard) algorithm through several stages of comparison with other algorithms. Vincent Rijmen coined this algorithm, and Joan Daemen became the winners of the new algorithm replacement contest replacement DES [9]. AES algorithm uses substitution, permutation, and the number of cycles applied to each block to be encrypted and decrypted. For each spin, AES uses a different key. The key to each round is called the round key. AES works in the form of bytes or characters, the block size for the AES algorithm is 128 bits (16 bytes).

RESULT

This process discusses the application of AES cryptography with Huffman compression in the SMS messaging system.

AES Encryption Process

AddRoundKey
a. Perform XOR logic or initial round between the initial plaintext / state and the cipher key.
b. Round of Nr-1 times. The process carried out in each series are:
   1) SubBytes or substitution bytes using the substitution table (SBox).
   2) ShiftRows or through permutations of data bytes from different array columns.
   3) MixColumns or randomize data in each state array column.
   4) AddRoundKey or perform XOR operations between data and keys.
c. Final Round is the process for the last round of SubBytes, ShiftRows, and AddRoundKey
d. Key Expansion
   The AES algorithm executes a primary key and creates a key expansion to produce a key schedule. The key represented as a word (w[i]). Similar to the state, but the state element is the cipher key. The key expansion required is AES Nb (Nr + 1) word, so that in AES-128 requires 4 (10 + 1) word = 44 words. Some steps that are carried out to create a key schedule are rotword (), subWord, and Rcon (). Rotword () is if w[i] represented by an array of rows or columns into rows (transpose), then it can be described by sliding once to the left in the array position as shift rows () does in the second row. For example w[i] = (a0, a1, a2, a3), we get Rotword (w[i]) = (a1, a2, a3, a0), subword (), which
substitutes every byte converted to hexadecimal form with S-box table as well as what SubBytes () does, e.g. \( w[i] = \text{CF4F3C09} \), by substituting into the S-Box table get Subword \( (w[i]) = 8A84EB01 \), where \( \text{CF} \) becomes \( 8A \), \( \text{4F} \) becomes \( 84 \), \( \text{3C} \) becomes \( \text{EB} \), and \( 09 \) becomes \( 01 \). \( R\text{con}[i] \) is a fixed component (Constanta) word of the round during the calculation process of expansion into the key schedule. The value on the AES-128 that uses ten times the rotation.

```java
public static byte[] encryptSMS(String secretkey, String message) {
    try {
        byte[] returnArray;
        Key key = generateKey(secretkey);

        private static Key generateKey(String secretkey) throws Exception {
            Key key = new SecretKeySpec(secretkey.getBytes(), "AES");
            return key;
        }
        Cipher c = Cipher.getInstance("AES");
        returnArray = c.doFinal(message.getBytes());
        return returnArray;
    } catch (Exception e) { e.printStackTrace(); byte[] returnArray = null; return returnArray; }
}
```

**Huffman Compression Process**

a. Count the Frequency

Tree is a concept that describes a directed graph that is connected and does not contain trajectories.

```java
int[] Frequency = new int[257];
Arrays.fill(Frequency, 1);
FrequencyTable freqtable = new FrequencyTable(Frequency);
HuffmanEncoder enc_cipher = new HuffmanEncoder(out);
enc_cipher.codeTree = freqtable.buildCodeTree();
int x = 0;
while (true) {
    int c = in.read();
    if (c == -1)
        break;
    enc_cipher.write(c);
    freqtable.increment(c);
    ++x;
    if (x < 262144 && isPowerOf2(x))
        enc_cipher.codeTree = freqtable.buildCodeTree();
    if (x % 262144 == 0)
        freqtable = new FrequencyTable(frequency);
}
enc_c.write(256);
```

b. Make a Huffman Tree for each character from root to leaf to 1 Huffman Tree

The Huffman code is a prefix code that consists of a collection of binary codes and has the characteristic that no binary code is the starting point for other binary codes. The prefix code draws as a binary tree that contains values or labels. On the left branch, the binary tree is labeled 0 while on the right branch,
and it is labeled 1. The series of bits formed at each line from root to leaf is a prefix code for the paired characters to produce a Huffman tree.

```java
public CodeTree buildCodeTree() {
    Queue<NodeWF> pqueue = new PriorityQueue<NodeWF>();
    for (int i=0; i < freq.length; i++) {
        if (freq[i] > 0)
            pqueue.add(new NodeWF(new Leaf(i), i, freq[i]));
    }
    for (int i = 0; i < freq.length && pqueue.size() < 2; i++) {
        if (i >= freq.length || freq[i] == 0)
            pqueue.add(new NodeWF(new Leaf(i), i, 0));
    }
    if (pqueue.size() < 2)
        throw new AssertionError();
    while (pqueue.size() > 1) {
        NodeWF nf_1 = pqueue.remove();
        NodeWF nf_2 = pqueue.remove();
        pqueue.add(new NodeWF(
            new InternalNode(nf_1.node, nf_2.node),
            Math.min(nf_1.lowestSymbol, nf_2.lowestSymbol),
            nf_1.freq + nf_2.freq));
    }
    return new CodeTree((InternalNode)pqueue.remove().node, freq.length);
}
```

**Huffman Decompression Process**

It is known that the code for a symbol / character cannot be the beginning of another symbol code to avoid ambiguity in the decompression process. Because each Huffman code generated is unique, the decoding process can be done easily.

- Do iterations to read binary string bits starting from the root.
- For each bit traverse the corresponding branch
- Code a series of bits that have been read with characters in the leaf.

```java
int[] Frequency = new int[257];
Arrays.fill(Frequency, 1);
FrequencyTable freqtable = new FrequencyTable(Frequency);
HuffmanDecoder dec = new HuffmanDecoder(in);
dec.codeTree = freqtable.buildCodeTree();
int count = 0;
while (true) {
    int symbol = dec.read();
    if (symbol == 256) // EOF symbol
        break;
    out.write(symbol);
    freqTable.increment(symbol);
    count++;
    if (count < 262144 && isPowerOf2(count) || count % 262144 == 0) // Update code tree
        dec.codeTree = freqtable.buildCodeTree();
    if (count % 262144 == 0) // Reset frequency table
        freqtable = new FrequencyTable(Frequency);
}
**AES Decryption Process**

```java
public static byte[] decryptSMS(String KunciRahasiaString, byte[] PesanTerenkripsi) throws Exception {
    generateKey(KunciRahasiaString);

    private static Key generateKey(String KunciRahasiaString) throws Exception {
        Key key = new SecretKeySpec(KunciRahasiaString.getBytes(), "AES");
        return key;
    }

    Cipher c = Cipher.getInstance("AES");
    c.init(Cipher.DECRYPT_MODE, key);
    byte[] decValue = c.doFinal(PesanTerenkripsi);
    return decValue;
}
```

**3.5 Implementation**

![Diagram of System Design]

**Figure 7. System Design**

**Figure 8. Main Menu**

**Figure 9. Create Message Form**
TESTING

At this stage, measurement of system performance and functional testing.

Table 1. Encryption and Compression Results

<table>
<thead>
<tr>
<th>Character</th>
<th>AES algorithm without compression</th>
<th>AES Algorithm with Huffman Compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>B114E95845841ED3BF7C187 D2D02D93B</td>
<td>@^-^[bV'-H&quot;Y)?!. 1$S -€</td>
</tr>
<tr>
<td>halo, apa kabar?</td>
<td>608A8BB7C4A5548CCDF06F3</td>
<td>4,4=0xxzJni  ÚñAëë2/</td>
</tr>
<tr>
<td></td>
<td>271AF32586108C81B39D5F2</td>
<td>iHñ®îå çü Yoe_ñ9ç·r%p</td>
</tr>
<tr>
<td></td>
<td>366FA2C3AFDF047A5E2A</td>
<td></td>
</tr>
<tr>
<td>apa yang kamu lakukan</td>
<td>DC1FFA08711128A5D5E</td>
<td>B?+B@pL^RD BŠŒ,4=ÐN  ÕE0</td>
</tr>
<tr>
<td>akhir pekan ini?</td>
<td>DA7E6957D24E57AE750</td>
<td>osE è ëæÂ7</td>
</tr>
<tr>
<td></td>
<td>775BC9F50AD00180252</td>
<td>p7päl_yóÅŒûBg;IÝå3»ç[</td>
</tr>
<tr>
<td></td>
<td>48778E924F03B9C5D</td>
<td>ØYûößBYÖä</td>
</tr>
<tr>
<td>Kalo kamu tidak ada</td>
<td>3D7567EB9DCF2B62A2816E</td>
<td>1A20.0zvVpÜØÜNAVeGÆz</td>
</tr>
<tr>
<td>aktivitas pada waktu</td>
<td>1CB9026CB2B076F0CF859D</td>
<td>D’oRYô ¡`</td>
</tr>
<tr>
<td>akhir pekan, maukah kamu</td>
<td>91906F632F7BC1D49BB12E</td>
<td>3WuzuÅ$KIFûBu‘üz]/gx_” gw</td>
</tr>
<tr>
<td>kamu mengikuti kontes lari?</td>
<td>3E8C4F7126459DDR04A398E</td>
<td>Bä¹,È2c</td>
</tr>
<tr>
<td></td>
<td>EB39D69442708AB61C95B2</td>
<td>;Zi0®x]±Ve&gt;»JC®°nÌA,Y w x$\</td>
</tr>
<tr>
<td></td>
<td>229419CB0F485F6A50A588</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D2ED2437DE0636C1FA81335</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8CD3EF3A9C43330C929956</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A9573A6FE2</td>
<td></td>
</tr>
</tbody>
</table>
Figure 11. Graph Comparison of the Number of Character AES Encryption and AES Encryption with Huffman Compression

Table 2. Presentation Compression Efficiency

<table>
<thead>
<tr>
<th>Number of Characters</th>
<th>Encryption</th>
<th>Encryption with Compression</th>
<th>percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32</td>
<td>28</td>
<td>12.5%</td>
</tr>
<tr>
<td>16</td>
<td>64</td>
<td>52</td>
<td>18.8%</td>
</tr>
<tr>
<td>45</td>
<td>96</td>
<td>80</td>
<td>16.7%</td>
</tr>
<tr>
<td>88</td>
<td>192</td>
<td>151</td>
<td>21.4%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>17.35%</strong></td>
</tr>
</tbody>
</table>
REFERENCES


