An Audio Shuffle-Encryption Algorithm

Abdelfatah A. Tamimi and Ayman M. Abdalla

***Abstract*—This novel algorithm employs a shuffling procedure to perform encryption of audio files, applying the stream cipher method. The algorithm uses a private key to perform encryption that is key dependent and data dependent. This algorithm was implemented and tested with different types of audio files of different sizes. Empirical analysis showed that this new algorithm is effective for encrypting audio files of medium or high quality.**

***Index Terms*— cryptography, stream cipher**

I. Introduction

THE Advanced Encryption Standard (AES) [1], is well-known for providing very secure encryption. Many encryption algorithms based on AES were developed. However, AES has limitations on some multimedia specific requirements, creating the need for other encryption algorithms to be developed [2], [3], [4], [5].

Some algorithms, such as [4], [6], [7], [8], applied different shuffling techniques to encrypt text files and images. Reference [9] used the RSA algorithm to encrypt speech files consisting of separate words. They extracted each word and converted it into text. This method cannot be easily generalized to apply to general audio contents. In [5], a chaotic system for audio encryption was presented. They obtained several encryption keys through a chaotic map where a different key is used in each iteration of the encryption process. Encryption of audio files in [10] was performed in five steps. These steps apply table substitution and different methods of shuffling to the audio data to perform lossless encryption.

Some research papers aimed to reduce audio encryption time by encrypting selected parts of the audio file [11], [12], [13]. Partial encryption of audio files can be done using the Discrete Fourier Transform to encrypt lower frequency bands [11]. Alternatively, an efficient encryption algorithm, such as AES, may be applied to selected parts of the audio file at the cost of a minor reduction in the security of encrypted files [12], [13].

In the next section, a novel algorithm is presented to perform encryption employing a shuffle stream cipher. The algorithm was implemented and tested, and analysis is performed to show its effectiveness.

Manuscript received July 10, 2014; revised July 26, 2014. This work was supported by a grant from Al-Zaytoonah University of Jordan.

A. A. Tamimi is the corresponding author (phone: +962-79-6559966; e-mail: drtamimi99@gmail.com). Both authors are with the Department of Computer Science, Faculty of Science & I.T., Al-Zaytoonah University of Jordan, Amman 11733, Jordan.

II. Performing Audio Shuffle Encryption

This algorithm takes an audio file and a key as input, and it performs byte shuffling of the audio data. The audio file is regarded as a stream, and the encryption performed is both key dependent and data dependent. The encryption algorithm is outlined below:

```
For i = 1 to k
    fixBit = Hash(Key,i)
    D  = Vector where D[j] is the value of bit (fixBit) of
         the j^th byte of the current stream
    S0 = Vector containing numbers of current stream
         bytes (j) that have (D[j] = = 0)
    S1 = Vector containing numbers of current stream
         bytes (j) that have (D[j] = = 1)
    Shuffle = Concatenation of S0 with S1
    Substitute the bytes of the current stream so that
      the new location of byte (j) is byte (Shuffle[j])
End For
```

This algorithm works as follows. A single bit, call it *fixBit*, is chosen by a hash function based on the key and iteration number. A shuffle vector (one-dimensional array) is constructed by listing the numbers of bytes that have the value of bit number *fixBit* equal to zero, followed by the numbers of bytes that have the value of bit number *fixBit* equal to one. This vector gives a mapping that specifies the new location of each byte in the partially encrypted stream. This step is repeated for several iterations. Each iteration uses a different *fixBit* and applies the same steps to the new partially encrypted stream that resulted from the preceding iteration. The number of iterations, *k*, is a small integer chosen by a key-dependent function.

The following simple example shows how this encryption is applied. Let the binary representation of the input be: 1111<u>0</u>101 0010<u>1</u>101 1010<u>0</u>011 1000<u>1</u>100 with *Key* = (3, 5). For simplicity, suppose two iterations are applied with the *fixBit* values chosen directly and sequentially from the key. In the first iteration, *fixBit* = 3, and this bit is underlined in the input above. Based on the values of this *fixBit*, *S0* = (1, 3) and *S1* = (2, 4), which make the shuffle vector (1, 3, 2, 4). Therefore, the input after the shuffle substitution will be 11110101 10100011 00101101 10001100. The second iteration is applied to this result in the same manner, but with *fixBit* = 5.

Before encrypting audio files using the above algorithm, data must be normalized into one or more streams of bytes. Depending on quality, the audio track may be using 8, 16, or more bits for representing audio values. If 8 bits (1 byte) is used for each value, the audio file is normalized into one audio stream. If higher quality audio is used, the audio file is normalized into more streams; where the number of streams

ISBN: 978-988-19252-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

is equal to the number of bytes required by the representation of audio values.

Consider, for example, the values in a typical audio format, which range from −1 to 1, where other audio formats may be normalized similarly. If 16 bits (2 bytes) are used for representing each value, add 1 to each value and then multiply it by 127. This converts each value into a rational number ranging from 0 to 254. We separate the integer part from the fraction part, where each part requires 8 bits (1 byte). When encryption is performed, the integer parts are regarded as one stream and the fraction parts are regarded as a different stream. Each stream is encrypted separately. Recall that a shuffle vector is constructed by listing the numbers of bytes that have the value of bit number *fixBit* equal to zero, followed by the numbers of bytes that have the value of bit number *fixBit* equal to one. Therefore, each of these two streams will generate a different shuffle vector, even when they use the same key. Consequently, when more streams are used, more mixing will take place, causing an increase of security.

The decryption algorithm is similar to the encryption algorithm. It is performed with the same number of iterations used in the encryption where each of the encryption steps is inverted. The decryption algorithm is outlined below.

For i = k to 1 step -1
  fixBit = Hash(Key,i)
  D = Vector where D[j] is the value of bit (fixBit) of the j^{th} byte of the current stream
  S0 = Vector containing numbers of current stream bytes (j) that have (D[j] = = 0)
  S1 = Vector containing numbers of current stream bytes (j) that have (D[j] = = 1)
  Shuffle = Concatenation of S0 with S1
  Substitute the bytes of the current stream so that the new location of byte number (Shuffle[j]) is byte number (j)
End For

When the encryption separates the input file into two or more streams, the decryption process will do that as well. Each stream is decrypted separately, and then the final decrypted streams are combined and restored into audio format.

III. Implementation and Analysis

The security of this algorithm comes from the complexity of the shuffle operation. If one or more bits in the key are changed, a different shuffle bit is chosen and the substitution is changed. For each data stream, there are $k \times 2^b$ different possible shuffle vectors for an input of size b bytes encrypted in k iterations. Since a typical audio file does not have a size less than a few kilobytes, a brute-force attack on the encrypted file is impossible.

The algorithm was applied to 25 audio files of various types and sizes, where their average size was 39 kilobytes. When different keys were used with the same file, they produced different encrypted files. In addition, analysis using histograms, peak signal-to-noise ratio (PSNR), correlation, and entropy show properties of the algorithm that resist statistical attacks.

When the input audio quality used two or more bytes for each value, the input file was separated into two or more streams, and each stream was encrypted separately. Then, the encrypted file was restored into audio format. For low-quality audio files requiring one byte per value, the file was regarded as one stream, and then encrypted and restored into audio format.

We plotted the audio values for different files to observe the effect of encryption. Fig. 1 and Fig. 2 show a plotted sample audio (Sample.wav) and its encrypted audio, respectively. The size of Sample.wav is 229 kilobytes. When comparing Fig. 1 to Fig. 2, it can be seen that the encrypted audio has no resemblance to the original and shows no features that may help an attack. The decryption algorithm recovered the original audio successfully, reproducing the audio in Fig 1. These observations were the same for all tested audio files.

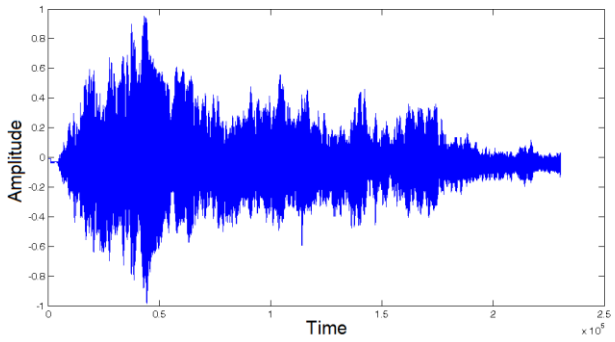

Fig. 1. Original audio for Sample.wav.

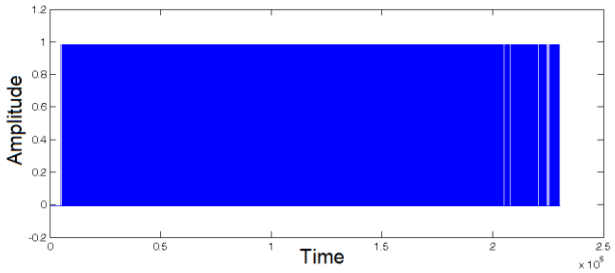

Fig. 2. Encrypted audio for Sample.wav.

When audio quality required two or more bytes for each value, the histograms of the encrypted files in these cases were different from the histograms of the original files. They gave no indication that may help statistical attacks. However, when the input audio quality required only one byte for each value, the histograms of the encrypted files were similar to the original-file histograms. This is due to the fact that shuffling the single stream keeps its values unchanged, unlike the cases that split values across two or more streams and shuffle them separately. Consequently, our algorithm is not suited for encrypting low-quality audio files since they will be vulnerable to statistical attacks. In that case, the audio file may be encrypted using a combination of our algorithm with another encryption algorithm that changes the audio values.

The mean squared error for two streams, stored in vectors A and B, is computed as follows:

ISBN: 978-988-19252-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(A[i]-B[i])^2 . \quad (1)$$

If streams *A* and *B* above represent the original audio file and its encryption, then PSNR is computed as:

$$PSNR = 10\log_{10}\left(\frac{MAX^2}{MSE}\right) \quad (2)$$

where *MAX* is the maximum value in the stream. A lower PSNR value is desired for encrypted files since it indicates more resistance to attacks. The PSNR obtained when encrypting Sample.wav was 4.373 decibels (dB). Figure 3 shows PSNR values computed for all encrypted audio files, where the average for these values was 4.536 dB. These low PSNR values indicate a high level of noise in the encrypted audio files, making them more resistant to attacks.

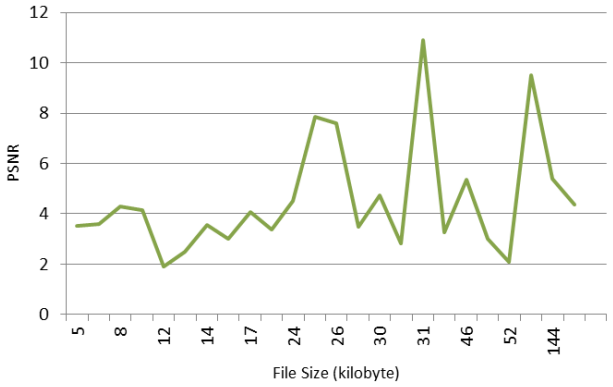

Fig. 3. PSNR for encrypted audio files.

The correlation, *r*, between two audio files, stored in vectors *A* and *B*, is computed as follows, where $\overline{A}$ and $\overline{B}$ are mean values for vectors *A* and *B*, respectively:

$$r = \frac{\sum_{i=1}^{n}(A[i]-\overline{A})(B[i]-\overline{B})}{\sqrt{\sum_{i=1}^{n}(A[i]-\overline{A})^2\sum_{i=1}^{n}(B[i]-\overline{B})^2}} \quad (3)$$

A lower correlation value between an audio file and its encryption indicates less resemblance between them, which provides more resistance to attacks. The correlation value for Sample.wav was -0.0274. The correlation values for all encrypted files are shown in Fig. 4. The average correlation value computed for the absolute values of correlation for the encrypted files related to their respective originals was 0.0263.

The randomness of a value can be measured with entropy. Entropy is computed as follows:

$$H = -\sum_{i=1}^{MAX}(P(i)\log_2(P(i))) \quad (4)$$

where *MAX* is the maximum value of the audio data and *P*(*i*) is the probability of the occurrence of value *i*. A higher entropy indicates higher randomness and, consequently, higher resistance to statistical attacks. The entropy for Sample.wav was 2.5932, and for its encrypted file, it was 6.2214. The average entropy value for the original audio files was 2.6498, where the average entropy value for their encrypted files was 5.2338. The entropy of all encrypted files and original files is illustrated in Fig. 5. As seen in the figure, the encryption caused a visible increase in entropy when the files were encrypted. This indicates an increase in randomness of values, which reduces the risk of attacks.

Fig. 4. Correlation between the original audio files and their corresponding encrypted files.

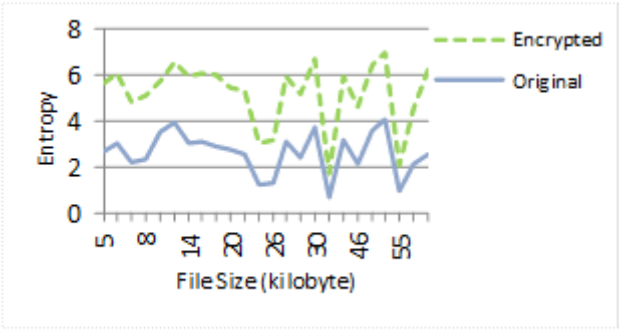

Fig. 5. Entropy of original and encrypted audio files.

IV. Conclusion

A new encryption algorithm for audio files was presented. This new algorithm performs encryption using a shuffling procedure. Statistical analysis using histograms, PSNR, correlation, and entropy showed that the algorithm is not vulnerable to statistical attacks unless it is used for encrypting low-quality audio files. In addition, the huge number of possible keys makes a brute-force attack on the algorithm impossible.

References

[1] The Advanced Encryption Standard, *Federal Information Processing Standards Publication (FIPS 197)*, pp. 92-96, 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[2] T. McDevitt and T. Leap. "Multimedia cryptology," *Cryptologia* (Taylor & Francis), vol. 33, no. 2, 142-150, 2009. DOI: 10.1080/01611190802300408

[3] D. Socek, S. Magliveras, D. C'ulibrk, O. Marques, H. Kalva, and B. Furht. "Digital video encryption algorithms based on correlation-preserving permutations," *EURASIP J. Inform. Secur.*, 2007. DOI: 10.1155/2007/52965

[4] J. W. Yoon and H. Kim. "An image encryption scheme with a pseudorandom permutation based on chaotic maps," *Commun. Nonlinear Sci. Numer. Simulat.*, vol. 15, no. 12, pp. 3998-4006, 2010. DOI: 10.1016/j.cnsns.2010.01.041

[5] R. Gnanajeyaraman, K. Prasadh, and Ramar "Audio encryption using higher dimensional chaotic map," *Int. J. Recent Trends Eng.*, no. Academy Publisher), vol. 1, no. 2, 103-107, 2009.

ISBN: 978-988-19252-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

[6] A. Tamimi and A. Abdalla, "A double-shuffle image-encryption algorithm," in *The 2012 Int. Conf. Image Processing, Computer Vision, and Pattern Recognition (IPCV '12)*, Las Vegas, NV, USA. 16-19 July 2012. Printed in *Image Processing, Computer Vision, and Pattern Recognition* [eds. H.R. Arabnia and L. Deligiannidis], pp. 496-499, CSREA Press, 2012.

[7] A. Yahya and A. Abdalla. "A shuffle encryption algorithm using S-box," *J. Comp. Sci.* (Science Publications), vol. 4, no. 12, 999-1002, 2008.

[8] A. Yahya and A. Abdalla. "An AES-based encryption algorithm with shuffling," in *The 2009 Int. Conf. Security & Management (SAM '09)*, Las Vegas, NV, USA. 13-16 July 2009. Printed in *Security and Management* [eds. H.R. Arabnia and K. Daimi], pp. 113-116, CSREA Press, 2009.

[9] Md. M. Rahman, T. K. Saha, and Md. A.-A. Bhuiyan. "Implementation of RSA algorithm for speech data encryption and decryption," *Int. J. Comput. Sc. & Netw. Secur.*, vol. 12, no. 3, 74-82, 2012.

[10] D. Sharma. "Five level cryptography in speech processing using multi hash and repositioning of speech elements," *Int. J. Emerging Technol. and Adv. Eng.*, vol. 2, no. 5, 21-26, 2012.

[11] S. Sharma, L. Kumar, and H. Sharma. "Encryption of an audio file on lower frequency band for secure communication," *Int. J. Adv. Res. Comput. Sc. & Software Eng.*, 3, no. 7, 79-84, 2013.

[12] B. Gadanayak, C. Pradhan, and U. C. Dey. "Comparative study of different encryption techniques on MP3 compression," *Int. J. Comput. Appl.*, vol. 26, no. 3, 28-31, 2011.

[13] B. Gadanayak, C. Pradhan, and N. Baranwal. "Secured partial MP3 encryption technique," *Int. J. Comput. Sci. & Inform. Technol.*, vol. 2, no. 4, 1584-1587, 2011.

ISBN: 978-988-19252-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)