Received February 13, 2020, accepted February 28, 2020, date of publication March 4, 2020, date of current version March 16, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2978297

Unified GPU Technique to Boost Confidentiality, Integrity and Trim Data Loss in Big Data Transmission

SHILADITYA BHATTACHARJEE[1], LUKMAN BIN AB. RAHIM[1], JUNZO WATADA[2,3], (Senior Member, IEEE), AND ARUNAVA ROY[2,3,4]

[1]High-Performance Cloud Computing Centre, Universiti Teknologi PETRONAS, Seri Iskandar 32610, Malaysia
[2]Computer and Information Sciences Department, Universiti Teknologi PETRONAS, Seri Iskandar 32610, Malaysia
[3]Shale Gas Research Group (SGRG), Universiti Teknologi PETRONAS, Seri Iskandar 32610, Malaysia
[4]Department of Computer Science and Engineering, SRM Institute of Science and Technology, Chennai 603203, India

Corresponding author: Shiladitya Bhattacharjee (shiladitya.b@utp.edu.my)

ABSTRACT Data integrity, confidentiality and data loss are the issues that arise during transmission because of the use of an inadequate security scheme. These issues become particularly critical for big data transmission due to its own individual overhead causes. Moreover, multiple executions of distinct security algorithms for maintaining confidentiality and integrity reduce throughput and add a large number of additional bits as security overhead that hampers the robustness against data loss. Conversely, an efficient compression technique minimizes data confidentiality, as it eliminates redundant data during compression. Contemporary studies shows the lack of security policies for solving the mentioned issues in a combinatorial manner. The current study proposes an innovative integrated technique to collectively addresses the above security issues. It increases confidentiality and offers a backup for accidental data loss by combining the simplified data encryption standard (SDES) and an advanced pattern generation technique that uses a unique pattern generation table. A novel dual round of error control technique has been introduced to maximize data integrity by considering an arbitrary number of transmission errors. A new compression technique is adopted to enhance the robustness against data loss along with high compression efficiency and resistance against transmission errors. Confidentiality and integrity are further enhanced by integrating advanced audio steganography that uses a distinctive sample selection for hiding bits. Additionally, the implementation of the proposed innovative integrated technique in the graphics processing unit (GPU) environment increases the execution speed and reduces time complexity with extended parallel processing power. Furthermore, the application of a GPU enhances the execution speed at least 28-fold compared to the CPU performance. Experiments are performed using the standard Calgary Corpuses, text files (sized up to 1 TB), and audio files to validate the objectives. The proposed method offers a higher signal-to-noise ratio (SNR), entropy, and avalanche effect (AE) and lower amplitude difference (AD), and uncorrectable error rate (UER) as well as a lower percentage of information loss (IL), which substantiates its potential to offer higher data confidentiality and integrity. The capacity to reduce the computational complexity is further measured with compression ratio (CR) and throughput. The results further depicts the method's superiority in offering confidentiality and integrity over contemporary approaches.

INDEX TERMS SDES, pattern string generation and the incorporation technique, dual round XOR operations to control transmission error, data compression, LSB-based audio steganography, GPU-oriented integrated technique, signal-to-noise ratio (SNR), entropy, avalanche effect (AE), amplitude difference (AD), uncorrectable error rate (UER), percentage of information loss (IL), compression ratio (CR) and throughput.

I. INTRODUCTION

Currently, 'big data' is a well-known term that depicts the exponential growth and availability of data in both structured and unstructured forms. According to [1], it commonly is a collection of similar or different datasets of very large scale and is furthermore too complex and difficult to process via traditional applications. In fact, countless efforts have been made to counter the adverse effects of various transmission hazards. For instance, a number of standard

The associate editor coordinating the review of this manuscript and approving it for publication was Tomás F. Pena.

public or private key cryptography methods have been used to resolve various confidentiality issues. Yet, such methods are subject to several limitations in terms of high time and space complexity due to involving many iterations as well as large initialization vector and key sizes during the encryption process [2], [3]. Alternatively, the applications of DNA structure-based pattern matching and hash functions are unable to stop data notching during transmission [4], [5]. Furthermore, efforts to resolve confidentiality issues by using a number of steganographic techniques in big data transmission have failed due to low hiding capacity and a complex hiding procedure [6]–[8]. The existing error control techniques for resolving data integrity issues are inadequate if the length of error bits is more than eight at the same time [9], [10]. Most importantly, the existing compression techniques fail to control the data size in big data transmission, as lossy compression compromises the data quality during compression by eliminating the redundant data bits permanently. Meanwhile, lossless compression methods also entail a number of issues. For instance, fixed-length coding (FLC), a type of lossless compression, is inefficient in not offering the desirable compression efficiency, while variable-length coding (VLC) is not robust against errors due to its dependency on prefix codes [11], [12].

In addition, we have observed that applications of very complex encryption, pattern matching or steganographic methods to enhancing data confidentiality entail high time and space complexity. As a consequence, data loss occurs, and data integrity in big data transmission suffers [13], [14]. There is commonly a high probability of data loss or corruption of a larger number of data bits in big data transmission. However, the existing methods do not adequately control larger (more than 8 bits) discrete or continuous errors and fail to offer a backup system for accidental data loss, which further jeopardizes data integrity [15], [16]. According to [17], [18], exiting data compression methods are inefficient in either data compression or maintaining data integrity due to a dependency on prefix codes. Moreover, the time complexity of addressing data confidentiality and integrity further reduces both efficiency aspects in big and complex data processing due to increased latency and channel congestion.

Hence, an innovative integrated technique is proposed in this study that can resolve all of these issues in a combinatorial way. This study also analyses the performance of all existing security techniques used to enhance the level of confidentiality in big data transmission. Additionally, this study helps identify the SDES encryption technique and design pattern generation and pattern matching techniques that perfectly fit big data transmission due to low time and space requirements. Their integration helps maintain data confidentiality and enhance robustness against data loss by providing a unique backup system for accidental data loss, which is a major problem in currently available big data transmission systems. Furthermore, the planned pattern generation technique uses a new reference table to generate pattern strings and offer a high confidentiality level; these features make it different from other exiting techniques.

Consequently, this research discusses various data integrity issues experienced in big data transmission and caused by various transmission errors. This study designs and builds a comprehensive error control technique that can also resolve the limitations of existing error control techniques by being applicable to any number of discrete or continuous error bits. The uniqueness of the proposed error control technique is in its capacity to control any number of discrete or continuous error bits, incorporated during transmission with a low space overhead, which makes it suitable for transferring big data files. Finally, this research explores various limitations discovered in the existing data compression methods used in big data transmission to enhance robustness against data loss. This exercise helps design and develop a highly efficient and low-complexity lossless compression method that suits big file transfers and removes all the limitations of current data compression methods. At the same time, the proposed compression technique offers a high level of robustness against data loss by reducing data overhead and enhancing resilience to various transmission errors, which makes it immediately usable.

As we know, a high compression efficiency poses a challenge to data confidentiality. Therefore, this research specifically identifies various existing security techniques that are less complex and can thus restore data confidentiality up to an adequate level. To this end, this study develops a less complex and highly confidential audio steganography technique that is a perfect fit to big data transmission. The innovation of this proposed technique is the use of an advanced sample selection method for hiding data, which helps enhance data confidentiality and data integrity at the same time. The implementation of the proposed novel integrated technique in the GPU environment enhances the execution speed at least 28-fold compared to the CPU implementation. The parallel processing power of a GPU further reduces the hardware and software latencies and data loss due to transmission delay and enhances data integrity by reducing data errors during execution.

The current work focuses on building an integrated technique that could enhance data confidentiality and integrity in addition to minimizing data loss and processing overhead in big data transmission. The specific objectives of this study are to do the following:

- Mitigate the present challenges to confidentiality related to big data transmission by applying the said low-complexity integrated technique and maximize the confidentiality level,
- Resolve the limitations of current data compression methods by controlling the extra data overhead and reduce channel congestion in big data transmission by designing a new lossless compression method as part of the said integrated technique,
- Reduce the processing time by reducing the hardware and software latencies, utilizing GPU-based accurate parallel

TABLE 1. Nomenclature.

$\sum$	A finite set of all numbers and characters
$\emptyset$	A null sequence
$\lfloor\rfloor$	The floor function
modulo	The remainder from division
$\langle\rangle$	The element sequence in a set
$\bigoplus$	The XOR operator
LSB	The least significant bit's position
MSB	The most significant bit's position

processing and an adequate resource allocation mechanism, and

✌ Explore the limitations of existing error control techniques to maximize data integrity by minimizing data loss in big data transmission with the use of a new error-checking technique applicable to any number of discrete or continuous bit errors.

The rest of paper is arranged as follows**:** Section II presents an effective review of the strengths and weaknesses of the contemporary security mechanisms used in big data transmission. Section III describes the current research gap and helps to resolve the current security issues, discovered in big data transmission. Section IV describes the design and the workings of SDES, as well as the said pattern generation, the error checking algorithm, the fixed length coding-based lossless compression, and LSB-based audio steganography. Complexity of the proposed approach has been analyzed in Section VI. Section V includes and describes the decoding process for retrieving the original data from the received stego files. The layout for performing experiments has been discussed in Section VII in the forms of Experimental Setup and Data Preparation. A set of assessment parameters have been defined in Section VIII and the performances of the projected integrated technique and its distinct parts have been done with them. In this section, the performances of this said integrated technique and its different parts are further compared with the performances displayed by corresponding existing techniques to show their superiorities over them. Finally, Section IX concludes and discusses the overall research work, as well as analyzes the strengths and weaknesses of the said integrated technique alongwith the possible future research.

II. RESEARCH BACKGROUND

Table 1 shows a list of notations that will be used throughout the remainder of the text.

According to the objectives of this research, the present section on the study's background has been divided into several subsections that review various security aspects of big data transmission. These subsections primarily discuss the strengths and limitations of distinct existing security techniques in enhancing data confidentiality and integrity in a large and complex file transmission.

A. CRYPTOGRAPHY AND PATTERNING

According to [1], [19], [20], data security is a technique that concerns protecting data from unauthorized intentional or unintentional tampering, destruction or disclosure of information and data loss. In accordance with [2], cryptography is the technique that converts or encrypts the input plaintext into an unreadable format called ciphertext. A technique to reduce the repetition of execution with a 128-bit key was inferred in [21], and the researchers also considered a table built on an S-box. This method is an embedded scheme designed to cover vital data and entails a low deployment cost. Nevertheless, it executes rather slowly and requires additional attention to protect the similar key pair. A GPU-enhanced structure was detailed in [22] for loading data in mobile devices by generating encrypted data using the XTS-AES encryption process. This encryption scheme approaches security through the loading of data of very large size and processes a very large quantity of data in parallel. The bit size, related to the input lump, is very minor. Therefore, the presented encryption is incapable of protecting from nonlinear attacks, such as rectangular and square attacks. A multipath-oriented routing protocol was proposed in [23] to improve confidentiality by using cryptography in ad hoc networks. The used protocol was rather simple and could be easily employed in diverse ad hoc devices. However, it resulted in similar data being transmitted redundantly by multipath routing and amplified channel overhead. A hash key-oriented encryption method for video transmission was devised in [24] to improve confidentiality. It precluded some unusual attacks, e.g., regrouping and erasure attacks, but data corruption in frames could occur during decryption and decompression. Pattern generation is the process of generating a sequence of tokens represented by character or binary strings, and pattern matching is the process of searching in string containing character or binary data. As reported in [4], [25], pattern generation and matching techniques further enhance data confidentiality in big data transmission. Here, the original information bits are converted into some predefined bit sequences to generate a particular bit pattern, and the pattern bits are then matched with the previously stored sample patterns from the database to retrieve the original information at the receiving end [26], [27]. Data patterns can also be generated by using a DNA structure and a hash function, and using such a hash function for security purposes is also useful for reducing the packet loss in any transmission system. However, DNA-based and hash function-based pattern generation techniques are too complex and cannot protect data from various attacks on their security. On a few occasions, cryptography has similarly been used to construct stronger message authentication codes and digital signatures; such processes involve many iterations and a great deal of space overhead during execution.

B. ERROR CONTROL MECHANISM

In any data transmission infrastructure, the conveyed information can be erroneous due to limitations of hardware or software, communication delay, bandwidth restrictions,

channel congestion, channel noise, and other reasons. Because of such erroneous information, the data retrieval process is interrupted at the receiving end, data loss occurs and the percentage of data transmitted correctly declines. The control mechanism for such errors can be divided into two phases: detection and correction. Error detection is the procedure of identifying data errors, which can be categorized into single or multiple bits and incorporated into data in a discrete or continuous form within the data during the transmission process [15], [16]. In contrast, error correction is the process of replacing erroneous bits with the original bits after the former have been detected [28], [29]. An error correction method of the burst error using cyclic codes has been proposed in [30]; it is simple and easy to implement, and it does not need any lookup table for decoding. However, the redundancy of the same code requires a longer execution time, and the method is too complex to implement by using logic gates. A forward error correction scheme for transmission in WSN has been designed in [28]. A flexible forward error correction code was proposed in [31] and used to recover the partial media. The cited study optimized the partial recovery of media, and the above code could be easily enhanced to protect against unequal errors. However, the complexity of error computation was very high, and the method additionally suffered from a high execution delay.

C. DATA COMPRESSION

With the rapid growth of internet use, transmissions of electronic media have become commonplace. Conversely, the transmission of large media files requires compressing the files to avoid extra overhead and channel congestion. The relevant compression techniques can be of two types: lossy and lossless. Lossy compression deletes redundant unimportant information permanently during the compression operation, which degrades data quality. JPEG, MPEG, TIFF, PING, and MP4 are some of the well-known examples of lossy compression methods. On the other hand, lossless compression methods, such as RLE, Shannon Fano, Huffman, adaptive Huffman, arithmetic, and LZ77, offer similar data quality as the original [17], [18]. Furthermore, [32] has recommended using a lossy data compression method for real-time data. This compression technique results in a high compression ratio and offers a low error rate and a higher execution speed. Lossless compression-based methods have been designed in [17], [33] for handling seismic floating point data. Such a method provides a high compression ratio and SNR in addition to offering a low data loss rate. Nevertheless, this technique offers very low processing speed, and transmission of compressed data is very expensive at the same time. Trimming the texture size via a lossless compression technique has been suggested in [34]. This approach improves the transmission rate but takes a very long time and degrades the texture quality during decompression.

D. STEGANOGRAPHY

Steganography is the process or tactic of hiding a message within media files (i.e., text, image, audio or video files). Based on the type of the cover file, it can be classified into several categories, such as plain text, still imagery, audio and video steganography. Text steganography offers a very weak confidentiality level, as the hidden text can be extracted very easily, whereas image steganography exploits a weakness of the human visual system (HVS) to maintain data confidentiality during incorporation of data within the image. However, image steganography is also considered weak, as it comprises a few simple steps for incorporating data and extracting hidden data. In steganography involving audio and video, secret messages are incorporated into digital audio or video signals. Among various audio steganography techniques, in parity bit embedding steganography, a parity bit is generated from the input cover sample string, and secret bits are embedded into the parity bit. The LSB of each binary sample is replaced by a secret message bit in an LSB-based steganography technique [35], [36].

Various studies have explored different steganographic techniques by using different media files and different types of input files for hiding data to resolve steganography's diverse shortcomings. Based on [35], the concept of multiple cryptographies where the data are encrypted into a cipher, and the cipher is hidden in a multimedia image file in an encrypted format has been proposed. This technique offers high data confidentiality and integrity but has a high time and space complexity and is unsuitable for big data applications. Additionally, a novel technique for image steganography based on the Huffman encoding has been proposed in [37]. It produces steganographic images of good quality to maximize the level of confidentiality and can resist a brute force attack. This algorithm also improves the security and the quality of the steganographic image and offers a better hiding capacity than do other existing techniques. However, its hiding capacity is nonetheless low, and it is not robust against errors.

E. TIME OPTIMIZATION

Time management is one of the greatest challenges in processing and transmission of large and complex files. The processing of a large and complex file on a CPU with any integrated security algorithm enhances the latency and transmission delay [38], [39]. As a consequence, data loss occurs during transmission of a large file over the internet. According to [40], [41], parallel processing in the GPU environment can reduce the time overhead by reducing the hardware and software latencies. Hence, several studies have explored the GPU environment, aiming to address the time-related issues in big data processing. However, the available literature could not address these issues related to time requirements in big data processing and transmission. According to [42], the processing of big graph data entails complex and numerous iterations that increase the task's difficulty; challenges include parallel memory bottlenecks, deadlocks, and resource inadequacy. Hence, the authors proposed an advanced mechanism for processing big graph data in the Cloud efficiently by analysing spatial data correlation and segregating a graph's dataset into clusters. In such a cluster, the execution overhead

can be aggregated by extrapolation depending on time series correspondence. The proposed technique further included temporal data compression that was implemented by utilizing temporal correlation for every individual time series or a single graph edge. An advanced data scheduling approach was further introduced to improve the throughput [43], [44]. However, this approach cannot be used for all types of correlations or data models to reduce the size of various large datasets, and the proposed technique cannot be used dynamically to produce a reasonable workload distribution and attain the desired throughput.

III. RESEARCH GAP ANALYSIS

The primary objective of this study is to ensure data confidentiality and integrity in big data transmission. The LSB-based steganography and the AES cryptographic techniques have been observed to lead to better efficiency in various aspects. However, various limitations, such as high time and space overhead and a weak ability to protect against data loss and ensure data integrity, make them unsuitable for big data applications. The literature shows that transmission errors pose another challenge to data confidentiality and integrity [1], [19], [20]. Consequently, studies show that using a cryptography-based hash function leads to better results, as it yields a higher SNR than do the other methods. However, these existing error control techniques entail several limitations, such as high time and space complexity and inability to cope with a large number of transmission errors [45]. At the same time, data overhead disrupts big data transmission very greatly. The background study suggests a number of data compression techniques to handle such issues. Nevertheless, those compression techniques either have poor efficiency of data compression or cannot efficiently provide the required robustness against various data errors [33]. A high compression efficiency also poses a challenge to data confidentiality. Hence, retaining data confidentiality without imposing a great deal of data and time overhead is another challenge in big data transmission. The issues of time overhead are important in executing any security mechanism for a large file transmission. The use of GPUs can solve these issues with the power of massive parallel processing. Yet, the subsequent issues that arise in GPU-based big data applications diminish its significance [39]. Therefore, there are still some research gaps that pertain to security aspects, such as data confidentiality and integrity in big data transmission.

IV. PROPOSED METHODOLOGY

The issues of confidentiality, integrity and data loss in big data transmission have been solved by implementing an integrated technique. The structure of the proposed technique is shown in Figure 1 using an unstructured input text file.

Figure 1 shows that the proposed integrated technique comprises several parts. The construction of each of them is further described in the following subsections.

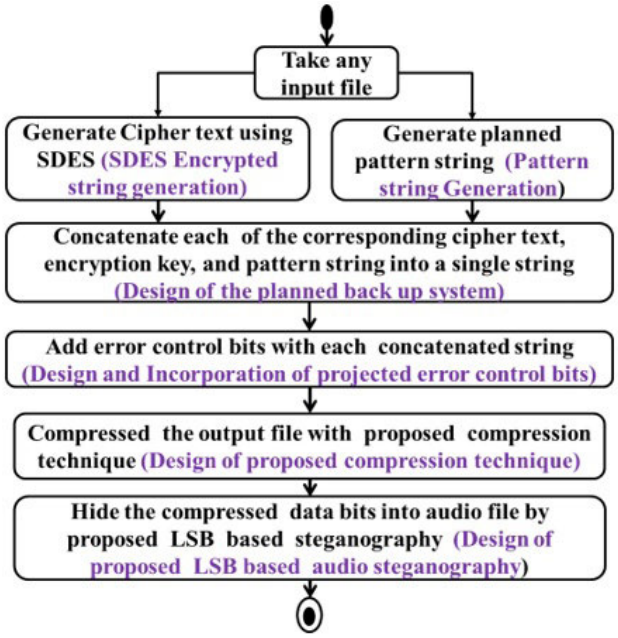

FIGURE 1. Flow of the proposed integrated technique.

A. SDES ENCRYPTION

The SDES encryption algorithm creates a cyphertext with an 8-bit input string and a 10-bit key. The 10-bit key can be generated by using any random 10-bit binary string generation algorithm. Studies [10], [46] present a detailed description of cyphertext formation from a parallel 8-bit input string using SDES and the corresponding key generation. Let n instances of 8-bit strings be generated from the input text file and store them in $\langle Pt_n \rangle$, where n is the total number of characters. The related 10-bit keys are stored in $\langle Key_n \rangle$, and the corresponding ciphertexts are in $\langle Ct_n \rangle$. A specific ciphertext and the related key are joined to create an array $\langle Et_n \rangle$ using the following Equation (4.1).

$$Et_n = (Ct_n \times (10)^{l(Key_n)} + Key_n \tag{4.1}$$

where $l(Key_n) = \lfloor \log_{10} Key_n \rfloor + 1$, and n varies from 0 to the total number of elements of input text. Similarly, the length of any element of array $\langle Et_n \rangle$ will be 18, as the length of each key is 10 bits and that of a cyphertext is 8 bits.

B. PATTERN STRING GENERATION

According to [25], a pattern string is a predefined bit sequence. Hence, a unique bit sequence is designed to generate the pattern string array in the proposed method. The uniqueness of this step is in the design of a novel pattern table. Input text $\langle Pt_n \rangle$ is used here to create pattern array $\langle Pat_n \rangle$ using Algorithm 4.1.

In Algorithm 4.1, line (a1) declares various variables and arrays for this operation, while lines (a2-a20) traverse the entire input text $\langle Pt_n \rangle$ to create the final pattern string array $\langle Pat_n \rangle$. The decimal value of an input character $\langle Pt_n \rangle$ is calculated by lines (a2-a5) and stored into $\langle Dec'_n \rangle$.

Algorithm 4.1: Pattern String Generation

(a1) Take, *Dec[n], Dec′*[n][3], *Pos[n][3]* as integer and *Sub[n][3][5]* as string as well as *x, y* as integer;
(a2) *for (int p = 0; p ≤ (n-1); p + +)*
(a3) *for (int i = 0; i ≤ 7; i + +)*
(a4) $Dec[p] = Dec[p] + (Pt[p][i] \times 2^i)$;
(a5) *End for*
(a6) *for (int k = 0; k ≤ 2; k + +)*
(a7) $Dec'[p][k] = \lfloor Dec[p]/10^j \rfloor$ *modulo 10;*
(a8) $Pos[p][k] = (k \times 10) + Dec'[p][k]$;
(a9) *for (int i = 0; i ≤ 4; i + +)*
(a10) $y = (Pos\ p][k]\ /2^i)$;
(a11) $x = y$ *modulo* 2^i;
(a12) $Sub[p][k][i] = x$;
(a13) *End for*
(a14) *End for*
(a15) *for (int j = 0; j ≤ 4; j + +)*
(a16) $Pat[p][j] = Sub[p][0][j]$;
(a17) $Pat[p][j + 5] = Sub[p][1][j]$;
(a18) $Pat[p][j + 10] = Sub[p][2][j]$;
(a19) *End For*
(a20) *End For*

TABLE 2. Pattern Table.

Position Value	Actual Value									
	0	1	2	3	4	5	6	7	8	9
0	**(0)**	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	**(6)**	0	0	0
2	0	0	0	0	0	**(5)**	0	0	0	0

Lines (a6-a7) split three consecutive decimal digits from each $\langle Dec'_n \rangle$. The position value of a particular separated decimal digit in the pattern table is calculated and stored into integer array $\langle Pos_{n\times 3} \rangle$ at step (a8). Each of the initial pattern values are subsequently converted into 5-bit binary strings and stored into $\langle Sub_{n\times 3\times 5} \rangle$ by steps (a10-a12). Each set of three resulting 5-bit binary strings is taken from string array $\langle Sub_{n\times 3\times 5} \rangle$ and concatenated to create the final 15-bit pattern string, stored in $\langle Pat_n \rangle$ in steps (a15-a19). An example is shown in Table 2 to explain the proposed pattern generation further. Let A be a character of the input text with the decimal value of 65. Initial pattern values generated from 65 are 0, 6, and 5. These initial values are placed in Table 2 according to position values of 0, 1, and 2.

Thus, the position values of the initial pattern such as 0, 6 and 5 are 1, 17 and 26, respectively. These three position values are subsequently converted into 5-bit strings and concatenated into a 15-bit final pattern string in each iteration. Apart from this, the SDES encryption is a symmetric key cryptographic method. Therefore, each key has to be transferred to the receiving end for decryption. Hence, if any accidental loss occurs, a backup system designed as shown in Figure 2 can be used to regenerate the lost data.

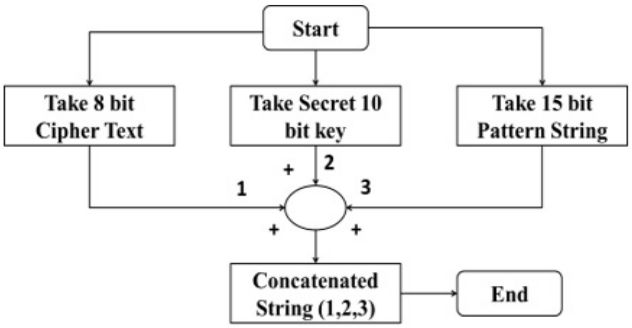

FIGURE 2. Concatenation operation.

Figure 2 shows that each concatenated string array is generated from each particular set of 8-bit ciphertexts $\langle Ct_n \rangle$, 10-bit keys $\langle Key_n \rangle$, and the corresponding 15-bit pattern strings $\langle Pat_n \rangle$. Here, the "+" sign represents the concatenation operation. This procedure is further illustrated by Equation (4.2).

$$Concat_n = (Et_n \times (10)^{l(Pat_n)}) + Pat_n \tag{4.2}$$

where $l(Pat_n) = \lfloor \log_{10} Pat_n \rfloor + 1$, and the length of a specific element in string array $\langle Concat_n \rangle$ becomes 33 bits after the concatenation operation has been performed.

C. CONTROL OF TRANSMISSION ERRORS

The proposed error control technique includes two phases used to identify and correct errors. The first is designed to facilitate an additional backup system to be used in the event of accidental loss. The second phase examines each bit of the custom string to determine the particular error location and to correct the respective error bit.

1) FIRST STEP OF ERROR CONTROL BIT FORMATION AND FUSION

In the first phase of error control bit generation, XOR operations are performed between each individual set of encrypted strings $\langle Et_n \rangle$ and the corresponding pattern string $\langle Pat_n \rangle$ to generate $\langle XorStr_n \rangle$.

$$XorStr_n = (Et_n \oplus Pat_n) \tag{4.3}$$

In this first phase of error control bit generation and incorporation, the resulting strings are stored in an another new string array $\langle Concat'_n \rangle$ by equation (4.4).

$$Concat'_n = (Concat_n \times 10^{l(XorStr_n)}) + XorStr_n \tag{4.4}$$

where $l(XorStr_n) = \lfloor \log_{10} XorStr_n \rfloor + 1$. A specific concatenated string from array $\langle Concat_n \rangle$ and a XOR string from array $\langle XorStr_n \rangle$ are 33 bits and 18 bits long, respectively. Hence, after the completion of the second phase, each of the strings in $\langle Concat'_n \rangle$ becomes $33 + 18 = 51$ bits long.

Algorithm 4.2: Error Control Bit Fusion in Phase II

(b1) Declare *len, num* as integer and *Str* as string;
(b2) $len = 51$; //*len* is the length of each item in $\langle Concat'_n \rangle$
(b3) *for (int* $i = 0$; $i \leq n$; $i++$*)*
(b4) $ErrCon[i][0] = Concat'[i][0]$;
(b5) $ErrCon[i][1] = Concat'[i][1]$;
(b6) $num = 0$;
(b7) *for (int* $j = 2$; $j < len$; $j = j + 2$*)*
(b8) $Str = Concat'[i][num] \oplus Concat'[i][num+1]$;
(b9) $ErrCon[i][j] = Str$;
(b10) $ErrCon[i][j+1] = Concat'[i][num+2]$;
(b11) $num = num + 1$;
(b12) *End for*
(b13) *End for*

2) SECOND STEP OF ERROR CONTROL BIT FORMATION AND FUSION

The second phase of error control bit generation and incorporation is initiated and performed with the input string array $\langle Concat'_n \rangle$. Algorithm 4.2 illustrates the operations in this phase. The final results are stored in a string array $\langle ErrCon_n \rangle$.

In Algorithm 4.2, line (b1) declares an integer variable len and a string variable Str, while line (a2) initializes len to 51. Lines (b3-b13) traverse the array $\langle Concat'_n \rangle$, though lines (b4-b5) assign the first and second elements of string array $\langle Concat'_n \rangle$ to the first two places in string array $\langle ErrCon_n \rangle$. Lines (b7-b12) traverse the array $\langle Concat'_n \rangle$ from its third to last elements. Line (b8) generates the reference error control bits by performing the XOR operation on two specific consecutive bits of a particular string of array $\langle Concat'_n \rangle$. Each of the resultant error control reference bits is included in the third place after the corresponding input bit pair in the same string of array $\langle ErrCon_n \rangle$ with the use of lines (b9-b11). All of these operations are continued for each of the string elements of $\langle Concat'_n \rangle$ to construct the string array $\langle ErrCon_n \rangle$ for controlling errors at the second stage.

D. DESIGN OF THE PROPOSED COMPRESSION TECHNIQUE

The application of SDES, pattern string generation and incorporation of error control bits causes the converted input file to exceed its original size. Hence, a new fixed length-based (FLC) lossless compression is introduced to reduce data overhead and address the limitations of existing FLCs and VLCs. After the dual rounds of error control bits' generation and incorporation operations are performed on string array $\langle Concat'_n \rangle$, the newly generated string array $\langle ErrCon_n \rangle$ is used as input here to execute the proposed data compression routine. In addition, the proposed compression routine has three parts, and each of them is described in detail in the following subsection.

1) GENERATION OF CHARACTER TABLE

In first phase of compression, the redundant elements are eliminated from the string array $\langle ErrCon_n \rangle$ and copied into the string array $\langle Chstr_m \rangle$, where $0 \leq m \leq n$. Here, m is the total number of non-redundant elements in the input text. The formation of $\langle Chstr_m \rangle$ is further represented by Equation (4.5).

$$Chstr_m = \binom{ErrCon_n}{m} \tag{4.5}$$

Once all the non-redundant elements have been copied, the string arrays $\langle ErrCon_n \rangle$ and $\langle Chstr_m \rangle$ are used as inputs to create the frequency table. Such a table $\langle Freq_m \rangle$ is created by putting the frequencies (the number of occurrences) of each non-redundant element in array $\langle ErrCon_n \rangle$. The string array $\langle Chstr_m \rangle$ is subsequently sorted in the order of descending frequency. The creation of frequency table $\langle Freq_m \rangle$ and sorting of character table $\langle Chstr_m \rangle$ are shown in Algorithm 4.3.

Algorithm 4.3: Creation of the Sorted Character Table

(c1) Declare *temInt* as an integer and *temStr* as string;
(c2) *for (int* $j = 0$; $j < m$; $j++$*)* //Frequency table creation
(c3) *for (int* $i = 0$; $i < n$; $i++$*)*
(c4) *if (Chstr[j] == ErrCon[i])*
(c5) *Freq[j] = Freq[j] + 1;*
(c6) *End if*
(c7) *End for*
(c8) *End for*
(c9) *for (int* $i = 0$; $i < m$; $i++$*)* //Sorting
(c10) *for (int* $j = 0$; $j < m$; $j++$*)*
(c11) *if (Freq[j-1] < Freq[j])*
(c12) *temInt = Freq[j-1];*
(c13) *temStr = Chstr[j-1];*
(c14) *Freq[j-1] = Freq[j];*
(c15) *Chstr[j-1] = Chstr[j];*
(c16) *Freq[j] = temInt;*
(c17) *Chstr[j] = temStr;*
(c18) *End if*
(c19) *End For*
(c20) *End For*

In Algorithm 4.3, line (c1) declares a temporary integer and a string variable used to create the sorted character tables whilst the array $\langle Chstr_m \rangle$ and $\langle ErrCon_n \rangle$ are traversed in lines (c1-c8) and (c2-c7). During such traversal, steps (c4-c6) compare each pair of elements of string arrays $\langle Chstr_m \rangle$ and $\langle ErrCon_n \rangle$ to calculate the frequency of each element of string array $\langle Chstr_m \rangle$ appearing in array $\langle ErrCon_n \rangle$ using step (c5). The corresponding calculated frequency values are subsequently stored in integer array $\langle Freq_m \rangle$ in step (c5). The character table $\langle Chstr_m \rangle$ is traversed further twice: initially from the first to the last element in lines (c9-c20) and from the second to the last element in steps (c10-c19), while lines (c11-c18) sort the character array $\langle Chstr_m \rangle$. Sorting is performed in the descending order of frequency of elements of array $\langle Chstr_m \rangle$ depending upon the corresponding frequency value in integer array $\langle Freq_m \rangle$.

2) GENERATION OF COMPRESSED CODE

The compressed codes corresponding to each non-redundant character of the character table are generated in this phase. In any input text, there can be at most 128 distinct characters according to the American Standard Code for Information Interchange (ASCII) Table. The compressed code generation process is split into several parts. Each of them is described in the following subsections.

Case 4.1 (Compressed Code Generation If $0 \leq m \leq 31$): The compressed codes are created by converting the position value of each element in the string array $\langle Chstr_m \rangle$ into a 5-bit string and storing the newly generated string into the string array $\langle CodeStr_m \rangle$. If *x, i, j* are integer variables, the code string is created using Equation (4.6).

$$\left.\begin{array}{l} x = \ i \ modulo \ 2^j \\ CodeStr_{ij} = x \end{array}\right\} for \ 0 \leq i \leq m \ \& \ 0 \leq j \leq 4 \qquad (4.6)$$

Here, *i* represents the position value of a specific code string, and *j* represents the bit sequences of each 5-bit compressed code.

Case 4.2 (Compressed Code Generation If $32 \leq m \leq 61$): In the second phase, the compressed codes are created if the position value of any character string in string array $\langle ChStr_m \rangle$ is varied between 31 and 61. The 5-bit code strings are created from the corresponding position values when $m \leq 30$, and the analogous compressed codes are put into string array $\langle CodeStr_m \rangle$ by repeatedly applying equation (4.6). The prefix code is generated and concatenated with the compressed code to create the final compressed code when the corresponding position value is greater than 30. In this phase, the decimal value 31 is converted into a 5-bit binary string using Equation (4.6), and the result is assigned to a string variable Pref. If *i, j* are two integers, $31 \leq i \leq m$ and $0 \leq j \leq 4$, then,

$$\left.\begin{array}{l} x = (i - 31) \ modulo \ 2^j \\ TemStr_j = x \\ TemStr' = (Pref \times 10^{l(TemStr)}) + TemStr \\ CodeStr_i = TempStr' \end{array}\right\} \qquad (4.7)$$

Here, x is an integer, TemStr and TemStr' are string variables, and $l(TemStr) = \lfloor \log_{10} TemStr \rfloor + 1$. The numeric value of 31 is subtracted on every occasion from a particular position value. Each value resulting from the subtraction is further converted into a 5-bit pre-compressed code and assigned to TemStr. The prefix code (*Pref*) and an individual pre-compressed code (TempStr) are subsequently concatenated and assigned to TemStr'. Finally, all the concatenated compressed codes are stored in a string array $\langle CodeStr_m \rangle$ using Equation (4.7).

Case 4.3 (Compressed Code Generation If $62 \leq m \leq 89$): The compressed codes are created and stored into the string array $\langle CodeStr_m \rangle$ in a way similar to that of Case 4.2 when the position value varies from 0 to 29. Using equation (4.6), decimal values 30 and 31 are converted into 5-bit strings to create prefix codes *Pref1* and *Pref2*. Repeating Case 4.2, the decimal value of 30 is subtracted from a particular position value of a character string in $\langle ChStr_m \rangle$ when the particular position value is between 30 and 59. Similarly, when the position value is between 60 and 89, the decimal value of 60 is subtracted from the respective value. The 5-bit pre-compressed codes are created from the corresponding position value that resulted from subtraction and concatenated with *Pref1* if $30 \leq m \leq 59$ and with *Pref2* if $60 \leq m \leq 89$ using equation (4.7).

Case 4.4 (Compressed Code Generation If $90 \leq m \leq 115$): In this case, the compressed codes in string array $\langle CodeStr_m \rangle$ are determined by repeating Case 4.3 when a particular position value varies from 0 to 28. Similarly to Case 4.3, prefix codes *Pref1, Pref2* and *Pref3* are generated by converting the decimal values 29, 30 and 31 into 5-bit binary strings, and decimal values 29, 58 and 87 are subtracted from the respective position values when a particular position value is in the range of either 29 to 57, 58 to 86 or 87 to 115. The pre-compressed codes are subsequently generated from each respective value that resulted from subtraction by repeating Case 4.3. Equation (4.7) determines the final compressed codes by concatenating the pre-compressed codes with their respective prefix codes and stores them into code array $\langle CodeStr_m \rangle$ in parallel.

Case 4.5 (Compressed Code Generation If $m \geq 116$): The compressed codes are created in a way similar to that of Case-4.3 when the position value of a particular character in string array $\langle ChStr_m \rangle$ diverges from 0 to 27. Similarly to Case 4.2 to Case 4.4, the prefix codes *Pref1, Pref2, Pref3*, and *Pref4* are created by converting the decimal values 28, 29, 30 and 31 into 5-bit binary strings using equation (4.6), and the decimal values 28, 56, 84 and 112 are subtracted if the position value of a particular character is between 28 to 55, 56 to 83, 84 to 111 or $112 - 127$, respectively. Each of the values resulting from subtraction is then converted into a 5-bit string to create the corresponding pre-compressed code. The final compressed codes are created by concatenating the pre-compressed codes with the respective prefix codes and are stored in string array $\langle CodeStr_m \rangle$ at the same time by iterating Equation (4.7).

3) GENERATION OF THE FINAL COMPRESSED STRING

String arrays $\langle ChStr_m \rangle$, $\langle CodeStr_m \rangle$ and $\langle ErrCon_n \rangle$ are used as inputs in this stage. The string array $\langle ComStr_n \rangle$ is created by replacing all elements of array $\langle ErrCon_n \rangle$ with the resulting compressed codes determined using Equation (4.8).

$$ErrCon_n = ChStr_m \rightarrow ComStr_n = CodeStr_m \qquad (4.8)$$

Equation (4.8) compares each pair of elements $\langle ErrCon_n \rangle$ and $\langle ChStr_m \rangle$. It further stores the equivalent compressed code from $\langle CodeStr_m \rangle$ into $\langle ComStr_n \rangle$ if a particular string of array $\langle ErrCon_n \rangle$ matches a string of $\langle ChStr_n \rangle$. After the creation of $\langle ComStr_n \rangle$ is complete, a separator string (*Sep*) is created by concatenating the value one (1) 101 times. Finally, the final compressed string (*FinCom*) is created using Equations (4.9-4.11) by concatenating $\langle ChStr_n \rangle$, *Sep*

and $\langle ComStr_n \rangle$.

$$FinCom = FinCom \times 10^{l(Chstr_m)} + Chstr_m \quad (4.9)$$

$$FinCom = FinCom \times 10^{l(Sep)} + Sep \quad (4.10)$$

$$FinCom = FinCom \times 10^{l(ComStr_n)} + ComStr_n \quad (4.11)$$

Here, $l(ChStr_m) = \lfloor \log_{10} ChStr_m \rfloor + 1$, $l(Sep) = \lfloor \log_{10} Sep \rfloor + 1$ and $l(ComStr_n) = \lfloor \log_{10} ComStr_n \rfloor + 1$. After concatenation, the final compressed string (*FinCom*) is ready for inclusion in the audio file.

E. PROPOSED AUDIO STEGANOGRAPHY

According to [47], [48], a high compression efficiency resulting from any compression technique threatens data confidentiality. As the proposed compression is also very efficient, it also jeopardises data confidentiality. Hence, this study considers a unique LSB-based audio steganography technique that helps to resolve this issue and offers a larger amount of hiding space along with a new hiding scheme. The development of this technique has been divided into several phases and is discussed in the following subsections.

1) BINARIZATION OF THE COVER AUDIO FILE

In the initial phase, an input audio file is converted into several minor samples using the sampling theory. In this step, samples are extracted from the cover file using a predefined threshold value. Normalization is further performed on these extracted samples to obtain their actual amplitude values. Such normalized amplitude values are numbers with a decimal point and lie in the range of -255 to 254. Hence, these normalized values are multiplied by 10 to convert them into integers. After being converted into an integer, each sample value is transformed into an 8-bit string using Equation (4.7). Finally, with a different use of Equation (4.7), these converted strings are stored into the string array $\langle A_s \rangle$, where s is the number of sample strings.

2) GENERATION OF REFERENCE ARRAY FOR SELECTING SAMPLE

According to [6], if no audible difference is detected between a steganographic audio file and the original file, the functional steganography is called efficient or vice versa. Hence, A few minor changes are made in the original cover file to perform the proposed embedding operation. The sample selections from the string array $\langle A_s \rangle$ for hiding compressed bits are initiated by marking the prime position values of string elements in array $\langle A_s \rangle$. The prime position values are then put into another integer array $\langle A'_p \rangle$, where p is the total number of prime positions in the string array $\langle A_s \rangle$. The position values of each element in $\langle A'_p \rangle$ is furthermore stored into an integer array $\langle Pos_p \rangle$. The resulting values from $\langle A'_p \rangle$ and $\langle Pos_p \rangle$ are added to generate the actual position values. The resulting position values are stored at that time into a separate integer array $\langle A''_p \rangle$ using equation (4.12).

$$A''_p = A'_p + Pos_p \quad (4.12)$$

3) EMBEDDING A COMPRESSED STRING INTO AN AUDIO SAMPLE

Embedding of compressed bits starts with the collection of 8-bit samples from $\langle A_s \rangle$ according to the position values of $\langle A''_p \rangle$, and the collected strings are stored into a string array $\langle Sam_p \rangle$. The final compressed string (*FinCom*) and the newly generated string array $\langle Sam_p \rangle$ are used as inputs here. The embedding operation picks two consecutive compressed bits form string *FinCom* in each iteration and incorporates them at the first and fourth bit positions respectively. The fourth bit of any selected sample string in $\langle Sam_p \rangle$ can be changed either from 0 to 1 or from 1 to 0. At the same time, the inclusion of a compressed bit in the first bit position does not require any change of other bit positions. Pass 4.1 and Pass 4.2 further describe the required changes in the other bit positions.

Pass 4.1: The Fourth Bit Is Changed From 0 to 1

```
(P1.1)  if (Samp2 = 1 and Samp4 = 1 )
(P1.2)    for (int i = 0; i ≤ 2; i++)
(P1.3)      Sampi = 0;
(P1.4)    End for
(P1.5)  End if
(P1.6)  if (Samp2 = 1 and Samp4 = 0 )
(P1.7)    for (int i = 0; i ≤ 2; i++)
(P1.8)      Sampi = 0;
(P1.9)    End for
(P1.10) End if
(P1.11) if (Samp2 = 0 and Samp4 = 1 )
(P1.12)   Samp4 = 0;
(P1.13)   for (int i = 0; i ≤ 2; i++)
(P1.14)     Sampi = 1;
(P1.15)   End for
(P1.16) End if
(P1.17) if (Samp2 = 0 and Samp4 = 0 )
(P1.18)   for (int i = 0; i ≤ 2; i++)
(P1.19)     Sampi = 1;
(P1.20)   End for
(P1.21)   for (int j = 4; j ≤ 7; j++)
(P1.22)     if (Sampj = 1)
(P1.23)       Sampj = 0;
(P1.24)       Break;
(P1.25)     End if
(P1.26)     else Sampj = 1;
(P1.27)   End for
(P1.28) End if
```

In Pass 4.1, steps (P1.1-P1.5) consider the changes made if the third and fifth bits of Sam_p are both 1, and the fourth bit changes from 0 to 1 during the inclusion of a compressed bit at the fourth position. Steps (P1.2-P1.4) set all bits from the first to the third to 0 at the same time. If the third bit is 1 and fifth bit is 0, steps (P1.6-P1.10) describe the changes in Sam_p, while steps (P1.7-P1.9) set all bits from the first to the third bit of Sam_p to 0. Steps (P1.11-P1.16) further describe the alterations made if the third bit of Sam_p is 0 and fifth bit

is 1. The fifth bit of Sam_p is set to 0 in step (P1.12), and steps (P1.13-P1.15) set all bits from the first to the third to 1. Steps (P1.17-P1.28) further describe the changes of Sam_p made if the fifth and the third bits of Sam_p are both 0. Consequently, steps (P1.18-P1.20) set all bits from the first to the third bit of Sam_p to 1. If any bit of Sam_p between the fifth and the eighth bit positions is 1, steps (P1.21-P1.27) set that bit to 0. Step (P1.24) stops such changes when 1 is encountered between the fifth to the eighth bit; otherwise, step (P1.26) sets all bits to 1.

Pass 4.2: The Fourth Bit Is Changed From 1 to 0

(P2.1) *if* ($Sam_{p2} = 0$ *and* $Sam_{p4} = 0$)
(P2.2) *for* (*int* $i = 0$; $i \leq 2$; $i++$)
(P2.3) $Sam_{pi} = 1$;
(P2.4) *End for*
(P2.5) *End if*
(P2.6) *if* ($Sam_{p2} = 0$ *and* $Sam_{p4} = 1$)
(P2.7) *for* (*int* $i = 0$; $i \leq 2$; $i++$)
(P2.8) $Sam_{pi} = 1$;
(P2.9) *End for*
(P2.10) *End if*
(P2.11) *if* ($Sam_{p2} = 1$ *and* $Sam_{p4} = 0$)
(P2.12) $Sam_{p4} = 1$;
(P2.13) *for* (*int* $i = 0$; $i \leq 2$; $i++$)
(P2.14) $Sam_{pi} = 0$;
(P2.15) *End for*
(P2.16) *End if*
(P2.17) *if* ($Sam_{p2} = 1$ *and* $Sam_{p4} = 1$)
(P2.18) *for* (*int* $i = 0$; $i \leq 2$; $i++$)
(P2.19) $Sam_{pi} = 0$;
(P2.20) *End for*
(P2.21) *for* (*int* $j = 4$; $j \leq 7$; $j++$)
(P2.22) *if* ($Sam_{pj} = 0$)
(P2.23) $Sam_{pj} = 1$;
(P2.24) *Break*;
(P2.25) *End if*
(P2.26) *else* $Sam_{pj} = 0$;
(P2.27) *End for*
(P2.28) *End if*

In Pass 4.2, steps (P2.1-P2.5) show the essential modifications made in Sam_p if the third and the fifth bits of Sam_p are both 0, and the fourth bit of Sam_p is changed from 1 to 0 during integration. Initially, Steps (P2.2-P2.4) set all bits from the first to the third to 1. Similarly, steps (P2.6-P2.10) perform the changes of Sam_p if its third bit is 0 and the fifth bit is 1. Steps (P2.7-P2.9) set all bits from the first to the third to 1. If the third bit of Sam_p is 1 and the fifth bit is 0, the adjustments of Sam_p are shown by steps (P2.11-P2.16). Consequently, step (P2.12) sets the fifth bit of Sam_p to 1, and steps (P2.13-P2.15) set all bits from first to the third bit of Sam_p to 0. The changes of Sam_p shown specifically by steps (P2.17-P2.28) are made if the third and the fifth bits of Sam_p are both 1. In particular, steps (P2.18-P2.20) set all bits from the first to the third to 0, while if any bit between the fifth and the eighth bit is 0, steps (P2.21-P2.27) change that particular bit to 1. Step (P2.24) prevents this conversion from continuing further if 0 is encountered between the fifth and the eighth bit; otherwise, step (P2.26) sets all bits from the fifth to the eighth to 0. In contrast, the first bit of Sam_p is changed according to the compressed bit during the embedding process, and no further changes are required for other bits of Sam_p. At the same time, if the forth bit or the first bit of Sam_p are the same as the corresponding compressed bits, Sam_p does not require any changes. As a result of repeating this overall set of steps, the entire compressed string is embedded within the respective string elements of array $\langle Sam_p \rangle$ to create the final array $\langle Ste_s \rangle$ of steganographic samples from the secret input text file.

4) FORMATION OF A STEGANOGRAPHIC AUDIO FILE

After the embedding of dual compressed bits at the first and the fourth *LSB* positions of each selected sample, the steganographic array $\langle Ste_s \rangle$ is used as input for generating the steganographic audio file of same format as that of the cover file. In the beginning, each element of $\langle A_s \rangle$ is converted into a decimal value using Algorithm 4.1 and stored into an integer array $\langle SteInt_s \rangle$. The decimal values of $\langle SteInt_s \rangle$ are dissimilar to the amplitude values extracted from original file. Hence, all of these values of $\langle SteInt_s \rangle$ are divided by 10 and stored in another double-precision array $\langle SteDb_s \rangle$. The denormalization operations are performed further with all elements of $\langle SteDb_s \rangle$ to obtain the corresponding steganographic samples using a predefined threshold value. Finally, the de-sampling operation is performed with all denormalized samples to concatenate them into a single steganographic audio file in the same format as that of the cover file for transmittal to the target location.

V. DECODING FOR RETRIEVAL OF THE ORIGINAL DATA

The reverse operations are performed at receiving end to extract the original data from the received steganographic file. This phase further comprises the following steps.

A. RETRIEVAL OF THE COMPRESSED STRING

In the foundation step of this stage, the received steganographic audio file is sampled based on the sampling theory to extract the audio samples, and normalization is performed using each extracted sample. The normalization process helps extract the amplitude values depending upon a predefined threshold value. After normalization, the newly generated decimal amplitude values are transformed to be within the range of -255 to 254. These normalized decimal values are converted to integers by being multiplied by 10. Each of the resulting integer values is further converted into the corresponding 8-bit binary string. The compressed bits embedded in the sample strings are selected using the method of Subsection 4.5.2 and Equation (4.12). Such bits are collected from the selected steganographic sample strings and concatenated into a single compressed string.

B. DECOMPRESSION OF THE EXTRACTED STRING

In the beginning of the decompression operation, the character string and the code string are separated from the extracted compressed string using the 101-bit separator and stored into a character array. The separator can be corrupted during transmission or extraction of the compressed string. Hence, if a 101-bit string comprises at least 70 discrete or contiguous bits with the value of one (1), it is regarded as a separator. The code array is prepared further based on the character array. By referencing the code array, all the elements of the compressed array are replaced with the corresponding character strings of the character array to form the extended string array $\langle ErrCon'_n \rangle$, where n is the number of elements.

C. REGENERATION OF THE ORIGINAL FILE

In this stage, the dual rounds of error control operations are performed with each element of array $\langle ErrCon'_n \rangle$. The decimal 0 is concatenated at the MSB position of each input string of string array $\langle ErrCon'_n \rangle$ to initiate the first round of the error control operation. In the first round, each third bit of the input string is regarded as a reference bit. Additionally, an XOR operation is performed for each two consecutive bits preceding the third reference bit. The resulting bit is subsequently compared with the third reference bit. If it fails to match, then the previous and the next bit sequences are checked in a similar way to detect the corrupted bit, are changed accordingly, and all reference bits are eliminated. Similar operations are performed cyclically with all elements of array $\langle ErrCon'_n \rangle$, and the output strings are stored in $\langle ErrCon''_n \rangle$.

In the second round, the pattern, encrypted and *XorStr* strings are separated from each element of $\langle ErrCon''_n \rangle$. A second pattern string is generated by performing the *XOR* operation for *XorStr* and encrypted strings; a second encrypted string is generated by performing the *XOR* operation for *XorStr* and pattern strings. The 10-bit secret key is subsequently separated from both encrypted strings, and two pairs of 8-bit keys are generated from them. The encrypted string pair is decrypted using the corresponding dual 8-bit key pairs, and the pattern string pair is de-patterned further. Two pairs of 8-bit strings are generated at this stage. They are subsequently converted into characters by reference to the *ASCII* table. Afterwards, such dual pairs of characters are compared, and the best matched character is regarded as original. All of these steps are repeated for all elements of $\langle ErrCon''_n \rangle$ to extract all characters, which are then concatenated into a file.

VI. COMPLEXITY ANALYSIS OF THE PROPOSED APPROACH

As per the working principle of SDES encryption algorithm, it executes with a fixed size of data block. Hence, with the increasing file size, the time and space complexities will not change with. As per the convention, the time and space complexity of SDES Encryption and Decryption are $\mathcal{O}(1)$. Similarly, the proposed pattern string generation, incorporation, retrieval of original symbol or character from pattern string as well as error control techniques require fixed size of input string for execution. They are independent of input size. Hence, the complexities of projected pattern string generation, incorporation and error control techniques are also $\mathcal{O}(1)$ accordingly. Again, the complexity of projected data compression depends on the input size. It has three parts called, character and code tables generation and final compressed string. As per the Algorithm-4.3, the complexity of character table creation in planned compression technique is $\mathcal{O}(m^2)$, where m is the total number of character in character table. For Case-4.1 to Case-4.5, the complexity of final compressed code table creation is $\mathcal{O}(1)$, which is dependent of fixed size of final character table. Furthermore, as per the Equation-4.7 to Equation-4.11, the complexity of final compressed string generation is $\mathcal{O}(n)$, where n is the total number of symbols or characters in the input file. As the proposed technique is developed for very large input files, hence, $n \gg m$ and the cumulative complexity of the proposed compression becomes $\mathcal{O}(n)$. The complexity of decompression technique is $\mathcal{O}(n)$. Conclusively, the projected steganography has four different stages such as binarization of cover file, generation of reference array for selecting sample, embedding a compressed string into an audio sample, and formation of stego audio file. Among the four stages, the embedding of compressed string into audio file does not rely on input file size. Therefore, the complexity of it is $\mathcal{O}(1)$. The other three stages are linear and dependent of input file size; specifically, number of compressed bits. Thereupon, the complexities of these three stages are $\mathcal{O}(n)$ accordingly. It follows that the unified complexities of the outlined steganography technique is $\mathcal{O}(n)$. In the same manner the complexity of compressed data retrieval from the stego file is also $\mathcal{O}(n)$. Hereinafter, the final collective complexities of the proposed integrated technique will be $\mathcal{O}(n)$ in both encoding and decoding junctures.

VII. LAYOUT FOR PERFORMING EXPERIMENTS

This section primarily includes the description of the experimental setup for implementing the proposed integrated technique and its distinct parts in a diverse environment. It further explains data preparation for testing various aspects of performance of the proposed technique.

A. EXPERIMENTAL SETUP

The proposed integrated technique was implemented to be executed on a CPU and a GPU to examine various aspects of its performance in both environments. During implementation, the PC being used was configured with 32 GB of DDR3 RAM and an *Intel® Core™ i*8 CPU. A custom CPU-based implementation was developed using Java (specifically, JDK 7.0) as the programming language due to its parallel processing feature (Java threads). The GPU-based implementation uses an NVIDIA GTX 570 GPU as the hardware and CUDA 8.0 as the programming language for utilizing the GPU's massive parallel processing capabilities. The open-source CUDA 8.0 was downloaded from NVIDIA's website, and

JDK 7.0 was downloaded from Oracle's website. Both CPU and GPU implementations used UBUNTU 16.04 LTS as the operating system. The High Performance Computing Centre (HPC^3) of Universiti Teknologi PETRONAS (Perak, Malaysia) provided the cloud storage space to store input and output files. Local area and wireless networks were used during the experimental large data transfer.

B. DATA PREPARATION

Performance measures of distinct parts related to the proposed integrated technique are evaluated using various types and sizes of input files, such as files of the Calgary Corpus, text files, and WAV sound files. Text files of various sizes are generated by a Java-based program designed to generate and merge such files. The sizes of input text files vary up to approximately 1 TB during the experiments. Due to several resource limitations, a 1 TB input text file is regarded as a large data file during performance analyses. The WAV files and Calgary Corpus files are downloaded from distinct benchmark databases. The large cover files (of the WAV format) used during the experiment are prepared by merging distinct small WAV audio files using the "Audio Convert and Merge Free" software. The names of all important software and input files used in the experiments as well as their sources are further tabulated in Table 8 in the Appendix.

VIII. RESULT ANALYSES

The performance of the proposed integrated technique and its various parts is evaluated in this section by computing their efficiency in minimizing the basic security issues related to confidentiality and integrity in big data transmission. This section is divided into three parts according to the basic objectives of this study, and they are discussed in detail in the following subsections.

A. ABILITY TO PROTECT DATA CONFIDENTIALITY

According to [37], [47], [48], the confidentiality level of our integrated technique can be measured by determining its capacity to retain perceptual similarities between the original and steganographic files and resist unauthorized tampering of illicit third parties.

1) PERCEPTUAL SIMILARITY

Following the approach of other studies, perceptual difference can be assessed by measuring the signal-to-noise ratio (*SNR*) and the amplitude difference (*AD*) between any original and steganographic samples. *SNR* and *AD* are defined in detail as follows.

a: SIGNAL-TO-NOISE RATIO (SNR)

The SNR is the amplitude or length of the output analogue or digital data sample relative to the background noise [11], [29], [34]. It is used to measure the difference in strength between the output and input samples. In audio steganography, it is used to calculate a measure of the level

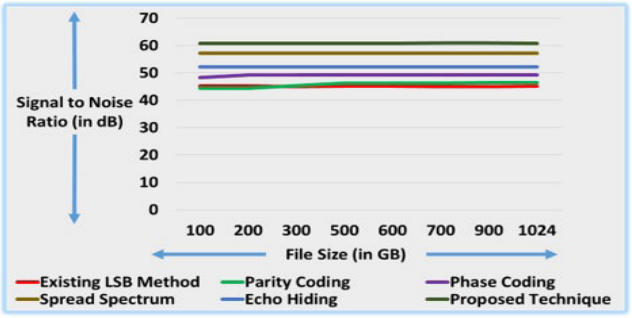

FIGURE 3. SNR_{dB} measures of various steganographic methods.

of confidentiality. It can be expressed in decibels (*dB*) as

$$SNR_{dB} = 10 \times \log_{10} \sum_n x^2(n) / \sum_n (x^2(n) - y^2(n)) \quad (7.1)$$

In Equation (7.1), $x(n)$ denotes the mean amplitude of the cover file, whereas $y(n)$ represents the mean amplitude of the steganographic sample, and n is any finite number. Accordingly, if *SNR* measured in dB is high, the perceptual difference between the cover file and the steganographic sample will be very insignificant, and vice versa. According to [16], [36], [47], various steganographic method, such as the *existing LSB method, parity coding, phase coding, spread spectrum* and *echo hiding*, can be used in big data applications. The *SNR* measures of the above-mentioned techniques are calculated using Equation (7.1) and plotted in Figure 3.

The steganographic file size has been varied from 100 GB to 1 TB in calculating SNR_{dB}. Moreover, Figure 3 shows that the proposed technique resulted in a greater *SNR* than did other existing methods in all cases. Th*SNR* values of our said technique vary from 60.79 dB to 60.91 dB. Figure 3 further shows that *parity coding* and the *existing LSB method* offer lower *SNR* values. As the proposed technique results in a higher *SNR* than those of the other existing methods, it offers a lower perceptual difference and higher confidentiality.

b: AMPLITUDE DIFFERENCE (AD)

The amplitude of any audio sample determines the power of the sound it represents. If a sound wave of frequency *F (cycles/sec)* travels *D* meters, the amplitude *(A)* of a sample is

$$A = \left(\frac{D}{F}\right) \frac{meter}{cycles/second} \quad (7.2)$$

The perceptual difference between the cover and steganographic audio files can be measured by considering their *amplitude difference (AD)* [36], [37], [49]. *AD* can be formulated as

$$AD = |Stego \text{ Amplitude-Actual } Amplitude| \quad (7.3)$$

According to the definition, *AD* represents the audible difference between the input and output samples and is used to measure the quality of any steganographic method. The *AD* values of the planned technique are shown in Figure 4.

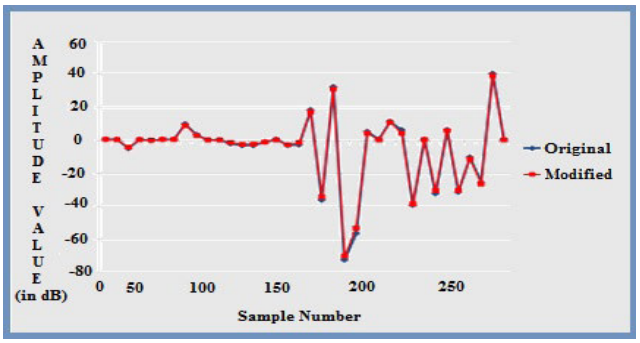

FIGURE 4. *Amplitude differences (ADs)* **between original and steganographic audio samples.**

In Figure 4, the peak amplitude values of steganographic samples and the corresponding original samples are almost identical. This signifies that there are minor perceptual differences between the original and steganographic samples. Hence, Figure 3 and Figure 4 show that the proposed steganographic technique offers high data confidentiality.

2) UNAUTHORISED INTERFERENCE

According to the literature, the strength of any security technique can be determined by measuring its ability to resist attacks. The *avalanche effect (AE)* and entropy values are important measures of such ability [50], [51]. This subsection defines both of them as follows.

a: AVALANCHE EFFECT

A corruption of any single bit or multiple bits during the encryption, decryption or transmission processes may cause very large changes in the output string. As a result, it may be impossible to decrypt the encrypted data properly. In cryptography, the *avalanche effect (AE)* measure calculates how a slight change during encryption can affect the output string [4]. *AE* can be calculated as

$$AE = \left(\frac{Total\ flipped\ bits\ in\ ciphertext}{Total\ bits\ in\ ciphertext} \right) \times 100 \qquad (7.4)$$

According to the definition, *AE* is used to measure the efficiency of any encryption technique. Hence, if a technique is characterized by a high *AE*, it is efficient in offering higher data confidentiality and integrity.

b: ENTROPY

According to information theory, *entropy* is used to measure the uncertainty of information associated with a random variable. As a result, *entropy* is the opposite of the amount of structural data present in any message [32][50-51]. According to Shannon's theory, the *entropy (H(S))* of any source *(S)* is

$$H(S) = \sum_{i=0}^{2n-1} \left(P(S_i) \times \log_2 \frac{1}{P(S_i)} \right) \qquad (7.5)$$

TABLE 3. Values of *AE* and *entropy* of diverse cryptographic methods (s).

Cryptographic Technique	Avalanche Effect (Percentage)	Entropy Value
AES	63.2	7.73
DES	64.3	7.53
3-DES	62.2	7.67
Blowfish	59.2	7.52
RSA	61.6	7.51
Proposed Integrated Technique	76.2	7.77

Here, $P(S_i)$ represents the probability of any character S_i, and n is the total number of characters. According to [31], [32], [37], [47], [50], *AES, DES, 3-DES, Blowfish* and *RSA* offer strong protection against diverse attacks. The efficiency of various techniques used to increase *AE* and *entropy* is calculated using Equations (7.4) and (7.5) and is shown in Table 3.

Table 3 shows that the proposed technique outperforms the other existing methods according to *AE* and *entropy*. Hence, the table shows that the proposed technique offers a better potential for protecting confidential data than do existing methods. Figure 3, Figure 4 and Table 3 therefore support our first objective.

B. ABILITY TO REDUCE PROCESSING OVERHEAD

A method's ability to reduce the file size can be measured by its compression ability in terms of producing a lower number of *bits per code (BPC)* [51], [52]. According to [13], the time efficiency of any process can be measured by calculating the *throughput*.

1) BITS PER CODE (BPC)

In compression methods based on *variable-length coding (VLC)*, the code length is determined by the frequency of each element. Hence, in such methods each code's length is varied, whereas it remains the same in *fixed-length coding (FLC)*. The average number of bits required to define a compressed code is called *bits per code (BPC)* [52]–[54]. *BPC* can be calculated as

$$BPC = \left(\frac{Compressed\ File\ Size}{Uncompressed\ File\ Size} \times 8 \right) \qquad (7.6)$$

By convention, if a compression method results in a high *BPC*, it is said to be efficient, and vice versa. According to the literature, arithmetic coding, Huffman, *BWT, LZSS BWT, and dictionary-based* methods are some of the well-known VLCs [51]–[54], whereas *ASCII coding, EBCDIC, and run length coding* are some well-known FLCs [11], [17], [18]. The values of *BPC* of various techniques are calculated using Equation (7.6) and plotted in Figure 5 and Figure 6.

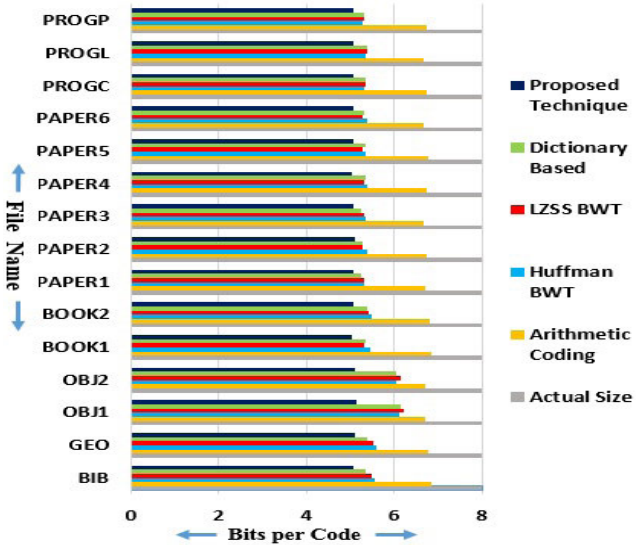

FIGURE 5. Comparison of the proposed technique and various *VLCs* with respect to *BPC*.

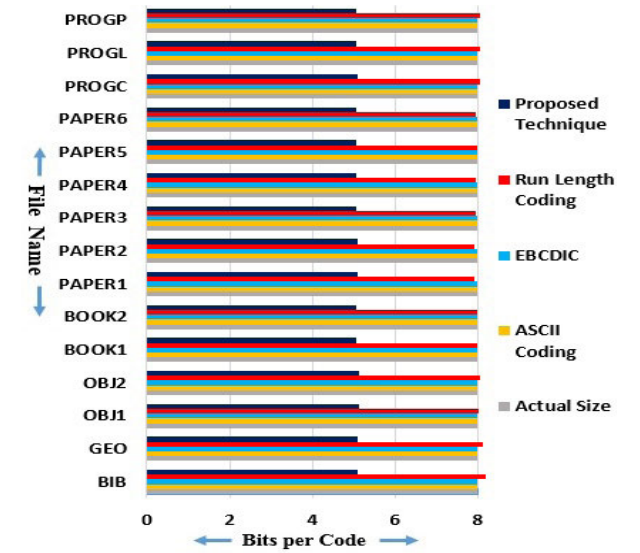

FIGURE 6. Comparison of the proposed technique and various *FLCs* with respect to *BPC*.

Figure 5 shows that the proposed technique offers the smallest *BPC* in all cases. Hence, its compression ability is better than that of other existing *VLCs*. Figure 6 further compares the proposed technique with existing *FLCs* according to the ability to produce lower *BPC* values.

Figure 6 shows that the existing *FLC* compression methods produce *BPCs* inferior to those of the proposed method in all cases. This finding thus supports our second research objective.

2) THROUGHPUT (TP)

In a computing system, any specific task should be completed within a certain timeframe. The efficiency of any data transmission can also be measured using throughput. According to [13], [21], *TP* represents the amount of work performed within a given time. *TP* can be measured as

$$TP = \left(\frac{Output\ file\ size}{Total\ execution\ time\ to\ hatch\ output} \right) \quad (7.7)$$

According to [16], [32], the required time is inversely proportional to processing speed. The *TP* measures related to all individual parts of the proposed technique and other existing methods are calculated using Equation (7.7) and plotted in Tables 4-7.

For execution on a CPU, Tables 4-7 show that each of the individual parts of the proposed integrated technique offers a higher execution speed in terms of offering a higher *throughput*. Table 6 shows that parity checking offered a

TABLE 4. Values of TP of various cryptographic techniques.

Cryptographic Technique	Creation (MB/Sec)	Retrieval (MB/Sec)
DES	2.64	2.62
RSA	1.67	1.68
3-DES	2.08	2.05
AES	4.17	4.16
Blowfish	6.71	6.72
Combination of SDES and the proposed pattern matching technique	8.61	8.63

TABLE 5. Values of TP of various compression techniques.

Compression Technique	Compression (MB/Sec)	Decompression (MB/Sec)
Arithmetic Coding	1.59	1.57
Huffman BWT	0.82	0.80
LZSS BWT	0.93	0.90
RLE	2.05	2.08
Dictionary-based	0.97	0.96
Proposed Technique	3.54	3.51

TABLE 6. Values of TP of various error control techniques.

Error control technique	Generation and Embedding of error control bits (MB/Sec)	Detection and Correction (MB/Sec)
CRC	4.24	4.22
FEC	4.59	4.57
Hamming Distance	5.21	5.23
ARQ	3.21	3.20
Parity Checking	6.21	6.23
Proposed Technique	6.15	6.15

TABLE 7. Values of TP of various steganographic methods.

Audio Steganography	Hiding Data (MB/Sec)	Data Retrieval (MB/Sec)
Existing LSB Technique	2.54	2.51
Parity Coding	2.76	2.77
Phase Coding	1.76	1.75
Spread Spectrum	1.54	1.51
Echo Hiding	1.87	1.85
Proposed Technique	3.45	3.47

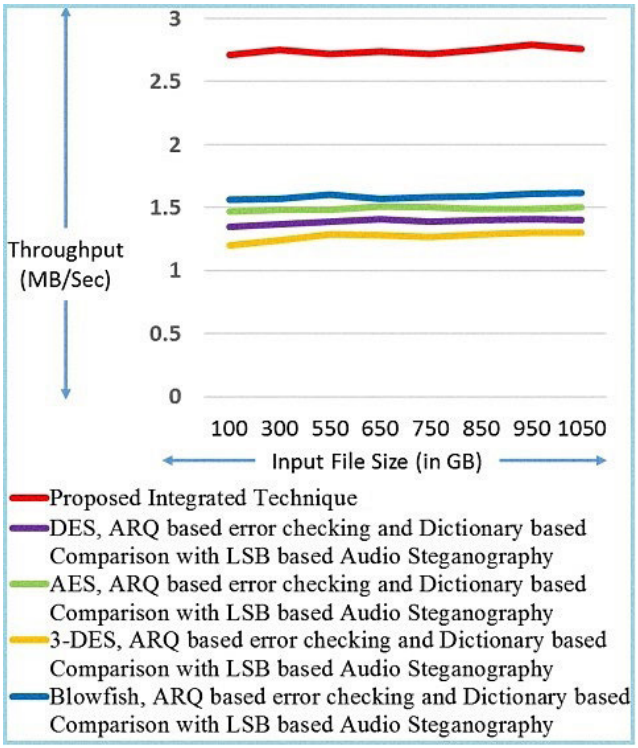

FIGURE 7. Comparison of *TP* of the proposed technique and several other security combinations (during execution on a CPU).

higher processing speed in terms of *TP*. However, it failed to detect more than one error bit and to identify the error's location. Considering a CPU implementation, the *throughput* of the proposed integrated technique and some state-of-the-art security combinations available in the literature are further compared in Figure 7.

Figure 7 clearly shows the superiority of the proposed integrated technique with regard to offering a higher *throughput* than that of other security combinations for executions on a CPU. The execution speed in the GPU environment is further calculated using Equation (7.7), and the speed differences between various methods in different conditions are plotted in Figure 8.

Figure 8 further shows that the execution speed of the GPU implementation is much higher than that of the CPU

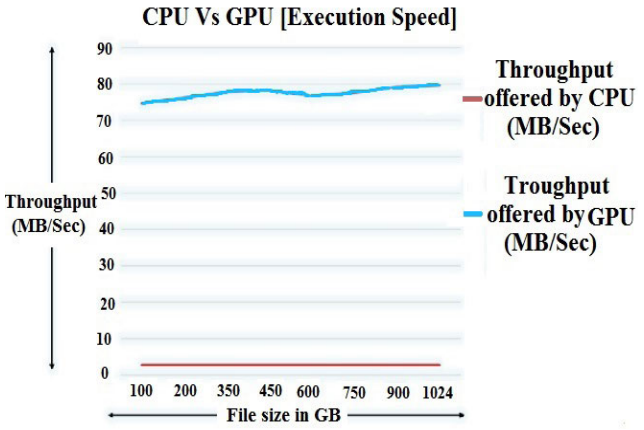

FIGURE 8. Values of *TP* observed during execution on CPU and GPU.

implementation. Hence, Tables 4-7 and Figure 7 prove that the individual parts as well as the integrated form of the proposed technique outperform other related individual security techniques or security combinations in terms of the execution speed. Thus, based on Tables 4-7, Figure 7 and Figure 8, we can claim that the proposed integrated technique fulfils our third research objective.

C. ABILITY TO OFFER INTEGRITY

Data integrity of any transmission system depends on its ability to protect from data loss or corruption. According to [11], [24], [32], the metrics required for measuring the integrity level are the *uncorrectable error rate* and the *percentage of information loss (IL)*. Both metrics of integrity are further described in the following subsections.

1) UNCORRECTABLE ERROR RATE (UER)

The *uncorrectable error rate (UER)* is used to calculate the total number of uncorrected errors that exist in a data file after an error correction algorithm has been applied [16], [32], [55]. The *UER* metric can be calculated as

$$UER = \frac{Total\ errors - Corrected\ errors}{Total\ Errors} \times 100 \qquad (7.8)$$

By convention, an error control technique with a high percentage of *UER* is regarded as less efficient, and vice versa. Values of *UER* of the proposed technique and other existing methods are calculated using Equation (7.8) and shown in Figure 9.

Figure 9 shows that *UER* obtained after applying the proposed integrated technique is less than that of other methods. Hence, the proposed integrated technique is more efficient in reducing data loss and enhancing data integrity in big data transmission.

2) PERCENTAGE OF INFORMATION LOSS

According to [24], [42], [56], a part or the entirety of data can be modified or corrupted during transmission by noise or various unwanted causes; this phenomenon is known as

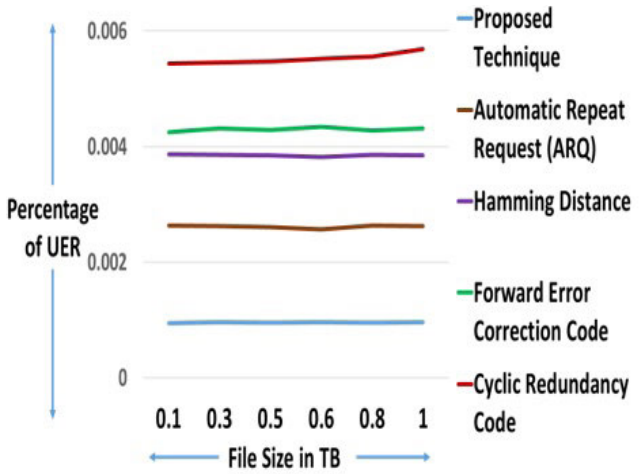

FIGURE 9. Values of *UER* of various error control techniques.

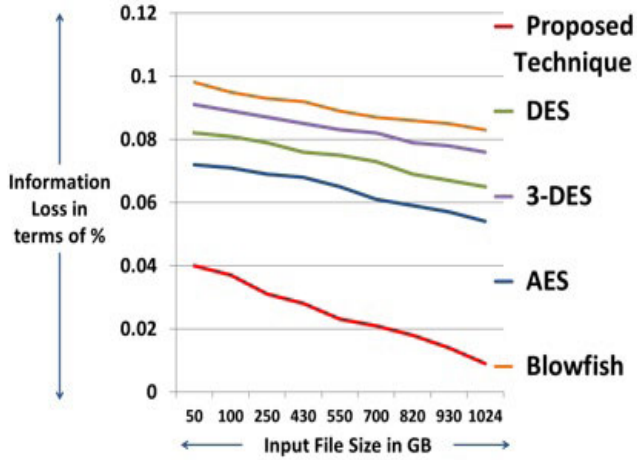

FIGURE 10. Percentage of *information loss* observed after the application of various security techniques.

information loss (IL). The ability to offer integrity can be measured by calculating *IL* as

$$IL = \frac{Actual\ file\ size - Restored\ file\ size}{Actual\ file\ size} \times 100 \qquad (7.9)$$

Equation (7.9) shows that the percentage of *IL* is the amount of the difference between the original and retrieved file sizes. The *percentage of IL* measured for the planned and other techniques during data retrieval is shown in Figure 10.

Figure 10 shows that all security techniques, including the proposed one, offer such a small *percentage of information loss* that all techniques are comparable to each other. According to Figure 10, the proposed integrated technique results in *IL* being at least 0.0009% and at most 0.004% of the input size. Under the same conditions, *Blowfish* produced the maximum *percentage of IL* among the existing methods. Hence, the proposed technique has been proven to be useful in protecting the transmission system in terms of having a lower *percentage of information loss*. So, Figure 9 and Figure 10 show that the proposed integrated technique is capable of enhancing data integrity, which fulfils the final objective.

IX. CONCLUSIONS AND DIRECTIONS FOR FUTURE RESEARCH

Present work shows the inability of the contemporary security techniques in offering an integrated solution for resolving confidentiality, integrity and data loss issues in big data transmission. Hence, the current study proposes a novel integrated technique to address the above issues of transporting big files in combinatorial way. The proposed technique includes simplified data encryption standard (SDES) and a new and less complex pattern generation technique to enhance the confidentiality of data and to offer a backup in case of accidental data loss. It adopts a new pattern generation table that distinguishes it from the other methods. Proposed approach further combines a novel dual round error control technique to combat various transmission errors. This technique removes the limitations of the existing error control techniques by accommodating any number of error bits. A distinct lossless compression technique is proposed in the current work for reducing data size. Moreover, this technique can resolve the current issues related to both lossy and lossless data compression methods. Usually, a high compression efficiency reduces data confidentiality; hence, this study introduces advanced LSB-based audio steganography as an integral part. It uses a new sample selection technique during data hiding, which resolves the current limitations of audio steganography. With the use of massive parallel processing, the implementation of the entire method on a GPU enhances the execution speed 28-fold compared to that on a CPU.

The result analysis shows that, compared to other existing steganographic techniques, the proposed integrated technique provides greater SNRs, avalanche effects, and entropy values and lower amplitude differences between the steganographic file and the cover audio file. The proposed integrated technique has been compared with several exiting audio steganography techniques in terms of complexity and the ability to resist various security attacks. Additionally, the ability to offer data integrity has been measured by estimating the percentage of information loss after the extraction of original information from the received steganographic audio files, and the percentage of uncorrectable error rate was calculated by using the extracted data's size. The results show that the proposed integrated technique produced lower percentage of information loss and uncorrectable error rate than did the existing error control techniques. Hence, the error control part of the proposed integrated technique outperformed in terms of integrity by resisting various transmission errors and data loss more effectively than did the other existing techniques. Moreover, the efficiency of the compression part of the proposed integrated technique has been estimated by computing the number of bits per code. The compression part of the proposed integrated technique offers a greater space efficiency than do the other existing approaches. The results show that the proposed integrated technique and its

TABLE 8. Sources of downloaded files and software.

Data files /Software	Downloaded from
Sound files (.wav)	http://www.wavsource.com/movie_stars/ -mstars.htm http://www.pacdv.com/sounds/voices-5.html http://www.jokes.thefunnybone.com/waves/ http://quiss.tripod.com/sounds.html http://www.thewavsite.com/birthday.htm
Audio Convert and Merge Free	http://download.cnet.com/Audio- -Convert-Merge-Free/3000-2140_4-75738774.html
Standard Calgary Corpus	http://www.data-compression.info/ -Corpora/CalgaryCorpus/

individual parts are more time-efficient in a CPU implementation in terms of throughput (2.71 to 2.76 MB/s) than the corresponding existing techniques. The analysis of results additionally shows that use of a GPU further enhances the throughput (76.25 to 79.14 MB/s) of the proposed integrated technique considerably by utilizing the power of immense parallel processing, and reduction of hardware and software latencies.

The experimental results show that despite the efficiency of the proposed integrated technique, it neither suppresses transmission errors absolutely nor protects against complete data loss. Additionally, the experimental results show that it produced a very low percentage of information loss (0.004-0.0009%), given the input file size. Furthermore, it displayed inefficiency in resisting all types of security attacks and data errors, as it produced up to 71% of *avalanche effect* when subject to various security attacks and the *uncorrectable error rate* of at least 0.00094%. The following points will be investigated in our forthcoming research works**:**

✌ With the further advancement of the projected integrated technique the percentage of data loss can be sensibly minimized, and the transmission error can be controlled more accurately.

✌ The execution time can further be reduced by employing parallel approaches and high-quality GPUs, which will be investigated in our future works.

✌ The user and packets authentication will be added in our future research works to further improve the security of the current proposal.

✌ The space complexity can also be reduced with the enhancement of projected data compression, which will be addressed in our subsequent works.

APPENDIX

See Table 8.

ACKNOWLEDGMENT

The authors would like to express their special gratitude to NVIDIA for supporting them in their research by offering the integrated GPU boards for testing various aspects of performance of the proposed integrated technique, important research directions and advice.

REFERENCES

[1] J. Whitworth and S. Suthaharan, "Security problems and challenges in a machine learning-based hybrid big data processing network systems," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 41, no. 4, pp. 82–85, Apr. 2014.

[2] C.-C. Lee, H.-H. Chen, H.-T. Liu, G.-W. Chen, and C.-S. Tsai, "A new visual cryptography with multi-level encoding," *J. Vis. Lang. Comput.*, vol. 25, no. 3, pp. 243–250, Jun. 2014.

[3] C. Li, D. Lin, B. Feng, J. Lü, and F. Hao, "Cryptanalysis of a chaotic image encryption algorithm based on information entropy," *IEEE Access*, vol. 6, pp. 75834–75842, 2018.

[4] O. Mujahid, Z. Ullah, H. Mahmood, and A. Hafeez, "Fast pattern recognition through an LBP driven CAM on FPGA," *IEEE Access*, vol. 6, pp. 39525–39531, 2018.

[5] A. K. Shukla, A. K. Shukla, B. Singh, and A. Kumar, "A secure and high-capacity data-hiding method using compression, encryption and optimized pixel value differencing," *IEEE Access*, vol. 6, pp. 51130–51139, 2018.

[6] A. Majumder and S. Changder, "A novel approach for text steganography: Generating text summary using reflection symmetry," *Procedia Technol.*, vol. 10, pp. 112–120, Jan. 2013.

[7] G. Blinowski, P. Januszewski, G. Stepniak, and K. Szczypiorski, "LuxSteg: First practical implementation of steganography in VLC," *IEEE Access*, vol. 6, pp. 74366–74375, 2018.

[8] N. Meghanathan and L. Nayak, "A review of the audio and video steganalysis algorithms," in *Proc. 48th Annu. Southeast Regional Conf. (ACM SE)*, 2010, pp. 1–5.

[9] B. Kwon, M. Gong, and S. Lee, "Novel error detection algorithm for LZSS compressed data," *IEEE Access*, vol. 5, pp. 8940–8947, 2017.

[10] X. Liu, S. Wu, X. Xu, J. Jiao, and Q. Zhang, "Improved polar SCL decoding by exploiting the error correction capability of CRC," *IEEE Access*, vol. 7, pp. 7032–7040, 2019.

[11] L. Guo, D. Zhou, J. Zhou, S. Kimura, and S. Goto, "Lossy compression for embedded computer vision systems," *IEEE Access*, vol. 6, pp. 39385–39397, 2018.

[12] S.-H. Kim, N.-U. Kim, and T.-M. Chung, "Attribute relationship evaluation methodology for big data security," in *Proc. Int. Conf. IT Converg. Secur. (ICITCS)*, Dec. 2013, pp. 1–4.

[13] S. Bajaj and R. Sion, "TrustedDB: A trusted hardware-based database with privacy and data confidentiality," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 3, pp. 752–765, Mar. 2014.

[14] G. Russello, C. Dong, N. Dulay, M. Chaudron, and M. van Steen, "Providing data confidentiality against malicious hosts in shared data spaces," *Sci. Comput. Program.*, vol. 75, no. 6, pp. 426–439, Jun. 2010.

[15] T. Hong, Y. Li, S.-B. Park, D. Mui, D. Lin, Z. A. Kaleq, N. Hakim, H. Naeimi, D. S. Gardner, and S. Mitra, "QED: Quick error detection tests for effective post-silicon validation," in *Proc. IEEE Int. Test Conf.*, Nov. 2010, pp. 1–10.

[16] J. Kang, D. Nyang, and K. Lee, "Two-factor face authentication using matrix permutation transformation and a user password," *Inf. Sci.*, vol. 269, pp. 1–20, Jun. 2014.

[17] X. Xie and Q. Qin, "Fast lossless compression of seismic floating-point data," in *Proc. Int. Forum Inf. Technol. Appl.*, vol. 1, May 2009, pp. 235–238.

[18] J. Shukla, M. Alwani, and A. K. Tiwari, "A survey on lossless image compression methods," in *Proc. 2nd Int. Conf. Comput. Eng. Technol.*, vol. 6, 2010, pp. V6-136–V6-141.

[19] A. S. Elmaghraby and M. M. Losavio, "Cyber security challenges in smart cities: Safety, security and privacy," *J. Adv. Res.*, vol. 5, no. 4, pp. 491–497, Jul. 2014.

[20] C. Tankard, "Big data security," *Netw. Secur.*, vol. 2012, no. 7, pp. 5–8, 2012.

[21] C.-H. Liu, J.-S. Ji, and Z.-L. Liu, "Implementation of DES encryption arithmetic based on FPGA," *AASRI Procedia*, vol. 5, pp. 209–213, Jan. 2013.

[22] M. A. Alomari and K. Samsudin, "A framework for GPU-accelerated AES-XTS encryption in mobile devices," in *Proc. IEEE Region Conf. (TENCON)*, Nov. 2011, pp. 144–148.
[23] J. Ben-Othman and B. Yahya, "Energy efficient and QoS based routing protocol for wireless sensor networks," *J. Parallel Distrib. Comput.*, vol. 70, no. 8, pp. 849–857, Aug. 2010.
[24] X. Wang, P. Golle, M. Jakobsson, and A. Tsow, "Deterring voluntary trace disclosure in re-encryption mix-networks," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 2, pp. 1–24, Feb. 2010.
[25] W. Bian, Y. Luo, D. Xu, and Q. Yu, "Fingerprint ridge orientation field reconstruction using the best quadratic approximation by orthogonal polynomials in two discrete variables," *Pattern Recognit.*, vol. 47, no. 10, pp. 3304–3313, Oct. 2014.
[26] F. Benhammadi and K. B. Bey, "Password hardened fuzzy vault for fingerprint authentication system," *Image Vis. Comput.*, vol. 32, no. 8, pp. 487–496, Aug. 2014.
[27] K.-K.-R. Choo, J. Nam, and D. Won, "A mechanical approach to derive identity-based protocols from Diffie–Hellman-based protocols," *Inf. Sci.*, vol. 281, pp. 182–200, Oct. 2014.
[28] M. P. Singh and P. Kumar, "An efficient forward error correction scheme for wireless sensor network," *Procedia Technol.*, vol. 4, pp. 737–742, Jan. 2012.
[29] Z. Zhao, G. Kim, D. Y. Suh, and J. Ostermann, "Comparison between multiple description coding and forward error correction for scalable video coding with different burst lengths," in *Proc. IEEE 14th Int. Workshop Multimedia Signal Process. (MMSP)*, Sep. 2012, pp. 37–42.
[30] V. P. Semerenko, "Burst-error correction for cyclic codes," in *Proc. IEEE EUROCON*, May 2009, pp. 1650–1655.
[31] J. Korhonen and P. Frossard, "Flexible forward error correction codes with application to partial media data recovery," *Signal Process., Image Commun.*, vol. 24, no. 3, pp. 229–242, Mar. 2009.
[32] L. M. Varlakshmi, G. F. Sudha, and G. Jaikishan, "An efficient scalable video encryption scheme for real time applications," *Procedia Eng.*, vol. 30, pp. 852–860, Jan. 2012.
[33] S. Bhattacharjee, L. B. A. Rahim, and I. B. A. Aziz, "A secure transmission scheme for textual data with least overhead," in *Proc. 20th Nat. Conf. Commun. (NCC)*, Feb. 2014, pp. 1–6.
[34] J. Strom and P. Wennersten, "Lossless compression of already compressed textures," in *Proc. ACM SIGGRAPH Symp. High Perform. Graph. (HPG)*, 2011, pp. 177–182.
[35] L. Fan, W. Tiejun, and W. Liuyi, "Research and implementation on model for high availability of enterprise information system," *IERI Procedia*, vol. 3, pp. 181–185, Jan. 2012.
[36] R. Das and T. Tuithung, "A novel steganography method for image based on Huffman encoding," in *Proc. 3rd Nat. Conf. Emerg. Trends Appl. Comput. Sci.*, Mar. 2012, pp. 14–18.
[37] P. Marwaha and P. Marwaha, "Visual cryptographic steganography in images," in *Proc. 2nd Int. Conf. Comput., Commun. Netw. Technol.*, Jul. 2010, pp. 1–6.
[38] S. I. Hakak, A. Kamsin, P. Shivakumara, G. A. Gilkar, W. Z. Khan, and M. Imran, "Exact string matching algorithms: Survey, issues, and future research directions," *IEEE Access*, vol. 7, pp. 69614–69637, 2019.
[39] M. Lastra, J. Carabaño, P. D. Gutiérrez, J. M. Benítez, and F. Herrera, "Fast fingerprint identification using GPUs," *Inf. Sci.*, vol. 301, pp. 195–214, Apr. 2015.
[40] M. Tang, Y. Yu, Q. M. Malluhi, M. Ouzzani, and W. G. Aref, "LocationSpark: A distributed in-memory data management system for big spatial data," *Proc. VLDB Endowment*, vol. 9, no. 13, pp. 1565–1568, Sep. 2016.
[41] G. Zhou, X. Zhang, Y. Lang, R. Bo, Y. Jia, J. Lin, and Y. Feng, "A novel GPU-accelerated strategy for contingency screening of static security analysis," *Int. J. Elect. Power Energy Syst.*, vol. 83, pp. 33–39, Dec. 2016.
[42] C. Yang, X. Zhang, C. Zhong, C. Liu, J. Pei, K. Ramamohanarao, and J. Chen, "A spatiotemporal compression based approach for efficient big data processing on cloud," *J. Comput. Syst. Sci.*, vol. 80, no. 8, pp. 1563–1583, Dec. 2014.
[43] K. Shirahata, H. Sato, and S. Matsuoka, "Out-of-core GPU memory management for MapReduce-based large-scale graph processing," in *Proc. IEEE Int. Conf. Cluster Comput. (CLUSTER)*, Sep. 2014, pp. 221–229.
[44] C. Chen, K. Li, A. Ouyang, Z. Zeng, and K. Li, "GFlink: An in-memory computing architecture on heterogeneous CPU-GPU clusters for big data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 6, pp. 1275–1288, Jun. 2018.
[45] P. Reviriego, M. F. Flanagan, S.-F. Liu, and J. A. Maestro, "Error-detection enhanced decoding of difference set codes for memory applications," *IEEE Trans. Device Mater. Rel.*, vol. 12, no. 2, pp. 335–340, Jun. 2012.
[46] S. Puangpronpitag, P. Kasabai, and D. Pansa, "An enhancement of the SDP security description (SDES) for key protection," in *Proc. 9th Int. Conf. Electr. Eng./Electron., Comput., Telecommun. Inf. Technol.*, May 2012, pp. 1–4.
[47] K. Qazanfari and R. Safabakhsh, "A new steganography method which preserves histogram: Generalization of LSB++," *Inf. Sci.*, vol. 277, pp. 90–101, Sep. 2014.
[48] L. B. A. Rahim, S. Bhattacharjee, and I. B. Aziz, "An audio steganography technique to maximize data hiding capacity along with least modification of host," in *Proc. 1st Int. Conf. Adv. Data Inf. Eng. (DaEng)*. Singapore: Springer, 2014, pp. 277–289.
[49] E. Satir and H. Isik, "A compression-based text steganography method," *J. Syst. Softw.*, vol. 85, no. 10, pp. 2385–2394, Oct. 2012.
[50] P. E. D. Pinto, F. Protti, and J. L. Szwarcfiter, "Exact and approximation algorithms for error-detecting even codes," *Theor. Comput. Sci.*, vols. 440–441, pp. 60–72, Jul. 2012.
[51] Y. Liang and Y. Li, "An efficient and robust data compression algorithm in wireless sensor networks," *IEEE Commun. Lett.*, vol. 18, no. 3, pp. 439–442, Mar. 2014.
[52] V. Sathish, M. J. Schulte, and N. S. Kim, "Lossless and lossy memory I/O link compression for improving performance of GPGPU workloads," in *Proc. 21st Int. Conf. Parallel Archit. Compilation Techn. (PACT)*, 2012, pp. 325–334.
[53] W. Zhan and A. El-Maleh, "A new scheme of test data compression based on equal-run-length coding (ERLC)," *Integration*, vol. 45, no. 1, pp. 91–98, Jan. 2012.
[54] S. T. Klein and D. Shapira, "Practical fixed length Lempel–Ziv coding," *Discrete Appl. Math.*, vol. 163, pp. 326–333, Jan. 2014.
[55] K. Wang, X. Ding, R. Lee, S. Kato, and X. Zhang, "GDM: Device memory management for GPGPU computing," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 1, pp. 533–545, Jun. 2014.
[56] Y. Yuan, M. F. Salmi, Y. Huai, K. Wang, R. Lee, and X. Zhang, "Spark-GPU: An accelerated in-memory data processing engine on clusters," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2016, pp. 273–283.

SHILADITYA BHATTACHARJEE received the B.Tech. degree in information technology from the Techno India University, India, and the M.Tech. degree in information technology from Jadavpur University, India, and the Ph.D. degree from Universiti Teknologi PETRONAS (UTP), Malaysia, with the development of an integrated technique for achieving confidentiality, integrity, and robustness of big data transmission. He is currently a Postdoctoral Researcher at the High-Performance Cloud Computing Centre, UTP. His current research interests are in data security, network security, big data security applications, the IoT, clustering, and cloud security. His current research is focused, in particular, on the IoT-based road safety and cloud migration. Some of the projects he is presently working on are: 1) a method for validating software architecture for efficient software modernization of legacy system; 2) seismic big data handling with Hadoop; and 3) various security aspects of big data processing using GPU. Apart from these, he is also involved in research projects related to software engineering, big data analytics, detecting faults in turbines and predicting missing data, and mission override control systems based on video processing.

LUKMAN BIN AB. RAHIM received the Ph.D. degree from Lancaster University, with a project verifying model transformations using model checking. He is currently a Core Researcher at the High-Performance Cloud Computing Centre and a Senior Lecturer in computer and information sciences at Universiti Teknologi PETRONAS (UTP). His current research interests are in formal verification, software and system modelling, and software architecture. His current research is focused, in particular, on adopting model-driven engineering and formal verification in cloud computing. Some of the projects he is presently working on are: 1) using architecture-driven modernization and model-driven engineering in deploying engineering simulation software as a cloud service; 2) formal verification of cloud security mechanisms using model checking; and 3) domain-specific modeling languages for educational games. Apart from these projects, he is also involved in research projects related to system engineering and big data, i.e., real-time cloud platforms, the correlation between scheduling and job workload and energy consumption, and secure data transmission for big data applications.

JUNZO WATADA (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees in electrical engineering from Osaka City University, Osaka, Japan, and the Ph.D. degree from Osaka Prefecture University, Sakai, Japan. Until March 2016, he was a Professor of management engineering, knowledge engineering, and soft computing at the Graduate School of Information, Production, and Systems, Waseda University, Kitakyushu, Japan. He is currently a Professor with Universiti Teknologi PETRONAS, a Research Professor at the Research Institute of Quantitative Economics, Zhejiang Gaongshang University, China, and a Professor Emeritus at Waseda University. He is the President of the Forum for Interdisciplinary Mathematics, India, from 2019 to 2021, after serving as the Vice President for six years, and the President of the International Society of Management Engineers. He received the Henri Coanda Medal Award from Inventico in Romania, in 2002, and the GH Asachi Medal from the Universitatea Tehnica GH Asachi, IASI, Romania, in 2006. He is the Principal Editor, Co-Chief Editor, and an Associate Editor of various international journals, including the *ICIC Express Letters*, the *Information Sciences*, the *Journal of Systems and Control Engineering* (Proc. IMechE), the *International Journal of Innovative Computing*, the *Information and Control*, and the *Fuzzy Optimization and Decision Making*.

ARUNAVA ROY received the Ph.D. degree from the Department of Applied Mathematics, Indian Institute of Technology Dhanbad, in August 2014. He worked as a Postdoctoral Researcher at the Department of Computer Science, The University of Memphis, Memphis, TN, USA. He has also worked as a Research Fellow at the Department of Industrial and Systems Engineering, National University of Singapore (NUS), and the Corporate Laboratory, Singapore University of Technology and Design (SUTD). He is currently working as a Research Fellow of the Computer and Information Sciences (CIS), Universiti Technologi Petronas (UTP), Malaysia, and a Research Assistant Professor (on one-year leave) with the Department of Computer Science and Engineering, SRM Institute of Science and Technology, India. He is also a part of the Shale Gas Research Group (SGRG), UTP. He has published a number of articles in various reputed journals. He has two U.S. utility patents on multifactor authentication (US9912657B2) and insider threat mitigation (US15/949,111). He has also authored a book *Advances in User Authentication*. His current research interests are machine learning, deep learning, statistical modeling, and information security.