A Secure Technique for Data Compression and Supercompression using Frequency Dependent Chaos

Sai Venkatesh Balasubramanian

Sree Sai Vidhya Mandhir, Mallasandra, Bengaluru-560109, Karnataka, India.
saivenkateshbalasubramanian@gmail.com

Abstract

A Chaos based compression technique offering high capacity and high security simultaneously is designed and implemented. A chaotic image, obtained by reshaping the signal representing a frequency dependant driven chaotic system is used as the data carrier in which data from the file to be compressed is embedded. Implementation of the algorithm is carried out in MATLAB and Python platforms for various filetypes such as txt, png, pdf, mp3, 3gp and rar formats. A comparative performance analysis reveals a high fidelity with a mean square errors of less than 0.0009 percent as well as a relatively high compression ratio value of 5-6. A very high level of security leading to up to 60 percent of mean square error values even for 1 percent misalignment in the decryption process is observed. The execution times for the implementations are obtained reasonably at around 5 seconds. A new compression technique, termed 'supercompression' consisting of repeated application of the compression technique is proposed. A proof-of-concept implementation achieved extremely high compression ratios of around 40000. The extreme simplicity of implementation coupled with the twin advantages of high compression ratios and high security forms the highlight of the present work.

Keywords: Data Compression, Chaos, Secure Encryption, Big Data, Supercompression

1. Introduction

The rise and growth of the internet and networking in recent years has led to an information explosion [1, 2, 3]. Buzzwords such as Big Data, Cloud Computing and Internet of Things are used extensively in industry and academia alike and the demands for higher data storage capacities are increasing by the day [4, 5, 6]. The urgent necessity of innovative techniques promising large compression ratios with minimal loss cannot be over-stated [7, 8, 9, 10].

Fortunately, advances made in the fields of nonlinear signal processing, wavelet transforms, chaos and fractal theory have enabled efficient techniques that achieve compression ratios of more than 100:1, where the compression ratio is defined as the ratio of the uncompressed data size to the compressed data size. [10, 11, 12, 13].

Specifically, fractal image compression, proposed by Barnsley et al. have exploited the self-similar and fractal pattern 'domains' in text and images and by eliminating the redundant and repetitive structures, have achieved compression ratios in the excess of 100:1 [14, 15, 16]. On the other hand, mainstream techniques such as JPEG and MPEG achieve maximum compression ratios of 30:1 with an acceptable loss factor [17, 18].

However, in the present era of Big Data, high storage capacity is not the only pressing need. The recent surge in cybercrimes such as hacking has highlighted a critical need for data security [19, 20]. While secure embedding techniques such as steganography and watermarking exist, they compromise heavily on the embedding capacity [21, 22].

The present work purports to the design and implementation of a chaos-based embedding process for textual data using images as the storage medium. Specifically, a frequency dependant chaotic system is proposed where the control parameter, the frequency ratio of the input signals forms a secure embedding 'key'. This chaotic system is used to generate a chaotic image. The textual data is embedded onto the image. This process thus forms a one-step process covering both compression and encryption, enabling the twin advantages of capacity and security simultaneously. The compression ratios obtained are greater than 150 and show a nonlinear dependence on the text size. The decryption of the text is observed to yield very low bit error rates of the order of 0.0009 percent. The simplicity of implementation of the present work coupled with extremely high compression ratios and secure encryption form the highlights of the present work.

The remainder of the work is structured as follows. 'Methodology', described in Sec. II elaborates on the two principal steps used in the embedding process - The Generation of a Chaotic 'Carrier' image, and the Embedding Process where the text to be compressed is converted into a pixel pattern, normalized and additively embedded in the image. The corresponding decryption process is also outlined. The 'Performance Assessment', described in Sec. III characterizes the

performance of the embedding process pertaining to three principal aspects - Capacity, Fidelity and Security. Standard parameters such as Compression Ratio and Mean Square Error are used for ascertaining the performance of the proposed embedding process. In the 'Inferences and Discussion' Sec. IV, the variation of such parameters with the nature of chaos generated, as described by the bifurcation diagram, as well as with the variations in the text size are discussed. Finally, based on the discussed results and inferences, a new variant of chaotic compression promising compression ratios in excess of 30000, termed 'Supercompression' is formulated and studied.

2. Methodology

The embedding process comprises of two functional units. The first pertains to the generation of a one dimensional chaotic signal, using a frequency-dependant iterative map and subsequently reshaping the signal into a two dimensional pattern (image). The second unit converts the text to be compressed into a pixel pattern, and after appropriate normalization, additively embeds it into the chaotic 'carrier' image generated in the first unit. This embedding process forms a one-step encryption-compression process. The de-compression then comprises of the corresponding inverse operations - subtractive decoding and denormalization. The compression-decompression process is illustrated as a block diagram in Fig (1). The implementation of this block diagram is carried out using a numerical computation software - MATLAB, and a system-interactive scripting language software - Python, and a comparative analysis of the corresponding performances is carried out [23, 24].

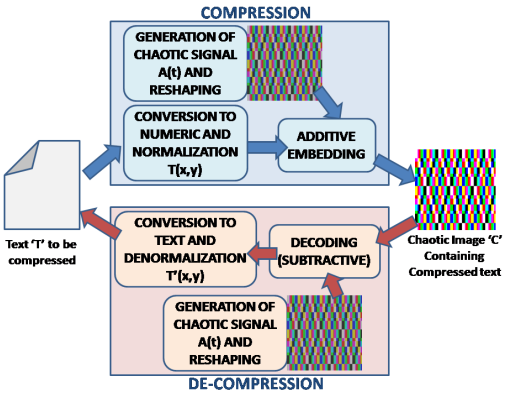

Figure 1: Block Diagram of the proposed Embedding Process

2.1. Generation of Chaotic 'Carrier' Image

The use of a chaotic system to facilitate a secure high capacity embedding process is the backbone of the present work.

The starting point in the definition of any chaotic system is the formulation of an iterative map, defining the evolution of a system using a control parameter r [25]. Conventionally, iterative map nonlinearities are defined on amplitudes by employing polynomial based functions [26]. However, the defining paradigm of the present work is the formulation of a novel kind of chaos - **frequency dependent chaos**, more similar in tone to the phase-based chaotic iterative functions used in standard circle maps [27]. This essentially consists of initially formulating an iterative map where the chaotic variable is frequency. The emphasis on the usage of frequency is to achieve two primary goals. The first is that frequency dependence encompasses a dependence on the driving signals, and this translates to easy tunability of the system. Arising from this is the second goal, which is that the frequency ratio of the driving signals, acting here as a control parameter, forms the ideal choice as a secure 'key', such that it is imperative to use the exact same key during compression and de-compression procedures.

The presence of nonlinearity in the iterative map owing to some switching functions such as the differential (mod) function ensures that for some control parameter values, the system displays chaotic behavior [25, 26]. It has been well established in literature that circle maps are ideally suited to describing such systems based on switching and two competing driving frequencies [28, 29, 30, 31, 32]. Hence, the iterative function proposed in the present work is inspired from the standard circle map, whose iterative map is given as follows [28, 29]:

$$\theta(i+1) = mod(\theta(i) + \Omega + \frac{K}{2\pi} Sin(2\pi\theta(i)), 1) \tag{1}$$

with θ being taken as modulo 1, and K and Ω being the control parameters. The standard circle map represents an area preserving chaotic map physically seen in systems such as the kicked rotator.

By converting the phase terms from the above equation to frequency based terms, and with a remormalization to π, the frequency iterative function, christened as the '**frequency map**' is formed as follows:

$$f_o(i+1) = mod(f_o(i) + \frac{f_2}{f_1} - V(f_o(i)), \pi) \tag{2}$$

Here the f_o terms denote the output frequencies, whereas f_1 and f_2 denote the frequencies of the input signals. $V(f_o)$ denotes the input signal waveform employed in the chaotic system. The salient features of the above mentioned iterated function are as follows:

1. The nonlinearity, provided by the modulus function represents the switching operation, physically implemented using differential amplifiers or XOR gates [33].
2. The control parameter $r{=}f_2/f_1$ is an additive parameter and determines when the system transits from order to chaos and vice versa.
3. The $V(f_o)$ introduces a signal dependence, thus enabling the controlling of chaos by changing the waveform used as input.

The bifurcation plot for a sinusoidal input $V(f_o(i)){=}sin(f_o(i))$ is given in Fig.(2).

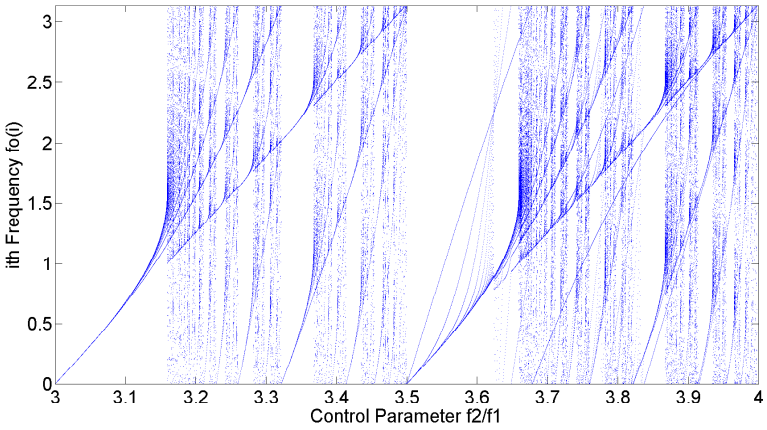

Figure 2: The Frequency Map Bifurcation Diagram of the chaotic system with sinusoidal inputs

From the diagram, it is seen that control values close to integers and half-integers give rise to order whereas non integral ratios such as 3.22 or 3.71 give rise to chaos. To understand the nature of the system in such chaotic regimes, the cobweb plot, a graphical visualization of the long term status of the system under repeated application of the proposed iterative map, is plotted for the frequency ratios of 3.22 and 3.71 in Fig. (3) [25, 26].

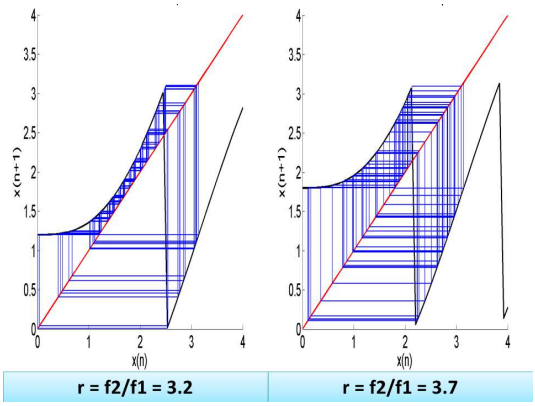

Figure 3: The Frequency Map Cobweb plots for frequency ratios of 3.22 and 3.71

It is noted that the iterative map elaborated above has three major components - first, a switching function based nonlinearity (the modulus function); second, an additive frequency dependance; third, a dependance on the input signal waveform (in this case, sinusoidal function). In order to form a chaotic signal from the defined frequency dependent chaotic system, another mapping is required where the chaotic parameter is amplitude. The three elements of the frequency map, namely modulus, ratio and signal dependence are stitched together to form an '**amplitude map**', without loss of principle and generality. This iterative map is given as follows:

$$v(i+1) = mod((v(i) + rcos(ri) + cos(i)), 1) \tag{3}$$

As with the frequency map, the nature of the amplitude map can be ascertained by plotting the bifurcation diagram, as shown in Fig. (4). The cobweb plots for the frequency ratio values of 3.22 and 3.71 are shown in Fig. (5). From the bifurcation diagram, it is seen that the amplitude map depicts a purely chaotic system, with no non-chaotic regions seen for any value of control parameter r. Significant amount of research exists in literature on the formulation and applications of purely chaotic systems [34, 35, 36]. However, the basis of frequency giving rise to the purely chaotic amplitude map is the novelty of the present work.

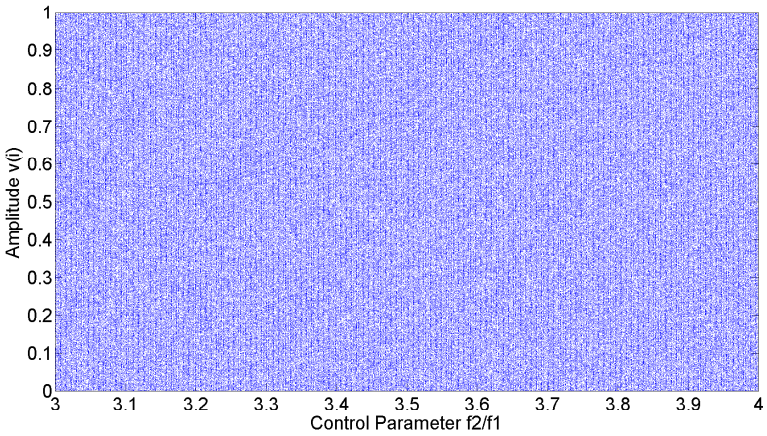

Figure 4: The Amplitude Map Bifurcation Diagram of the chaotic system with sinusoidal inputs

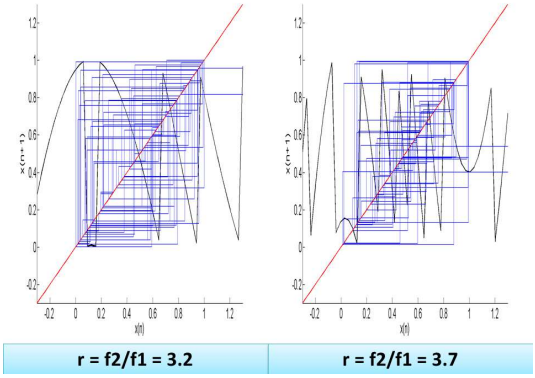

Figure 5: The Amplitude Map Cobweb plots for frequency ratios of 3.22 and 3.71

From the amplitude map, a signal is defined as follows, representing *ipso facto* the chaotic system of Fig. (2) and Fig. (4).

$$A(t) = mod(sin(2\pi f_1 t) + sin(2\pi f_2 t), 1) \tag{4}$$

where t is the discrete sample index and runs from 1 to N for a signal of N samples.

The waveform and 1D FFT spectrum corresponding to this equation are illustrated in Fig. (6) and Fig. (7) respectively.

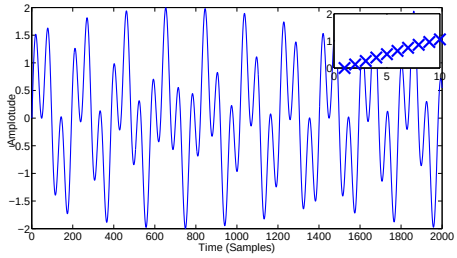

Figure 6: The waveform of the chaotic signal $A(t)$. Inset: The first ten samples of the signal

.

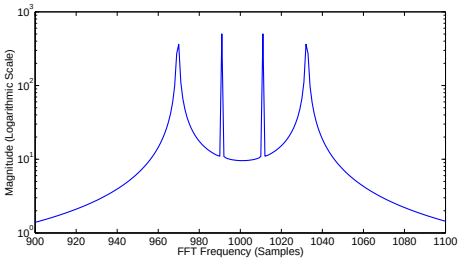

Figure 7: The 1D FFT Spectrum of the chaotic signal $A(t)$

The chaotic nature of the signal $A(t)$ is assertively established by calculating the largest Lyapunov Exponent, a measure of a system's sensitive dependence on initial conditions [37]. Rosenstein's algorithm is used to compute the Lyapunov Exponents λ_i, where the sensitive dependence is characterized by the divergence samples $d_j(i)$ between nearest trajectories represented by j given as follows, C_j being a normalization constant:

$$d_j(i) = C_j e^{\lambda_i (i\delta t)} \tag{5}$$

The Largest Lyapunov exponent thus obtained for the chaotic signal $A(t)$ is 2.29 [38].

The next step in the embedding process is to map the values of the generated chaotic signal into a pixel pattern (image). In order to achieve this, the 1D signal, represented as a row vector, is reshaped into an array of the dimensions $M \times M \times 3$, where M is an integer, and this reshaping is shown conceptually in Fig. (8). The resultant is that the N samples of the vector $A(t)$ is reshaped into a 3D array of the form $A(m, m, i)$, m and i representing spatial and color coordinates respectively [39]. The order of reshaping is essentially raster, going from left to right, then top to bottom for each color space, and this order is taken care of by the in built 'reshape' commands of MATLAB and Python [39]. Appropriate zero padding is done to ensure that M is an integer, such that the relation $N{=}M \times M \times 3$ holds. The base frequency f_1 is set as 0.01 and the frequency ratio r is set to the irrational number, π. The generated image for a M value of 440 is shown in Fig.(9). An M value of 440 implies that the image is capable of 'storing' $440 \times 440 \times 3$ characters/bytes. In Joint Photographic Experts Group (JPEG) format, this image takes up 31.2 kB storage space.

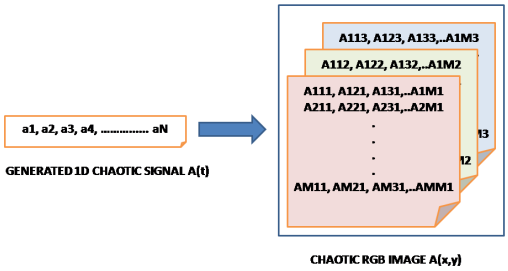

Figure 8: The mapping and reshaping of chaotic signal $A(t)$ into the RGB 'carrier' image $A(x, y)$

The chaotic/fractal nature of this image is understood by computing the fractal dimension, using the Minkowski Bouligand Box Counting Method [40]. In this method, various square 'boxes' of different sizes e are formed and for each

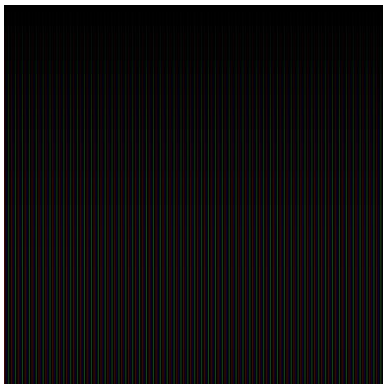

Figure 9: The chaotic 'carrier' image (440x440 png)

size e, the number of boxes $N(e)$ required to cover the entire set is computed. The fractal dimension D is then given by

$$D = \lim_{e \to 0} \frac{log(N(e))}{log(e)} \tag{6}$$

For the generated image shown in Fig.(9), the fractal dimension is obtained as 1.807, indicating the presence of self-similarity in the generated image.

2.2. The Embedding Process

The next step is the embedding process, which essentially involves reshaping a binary based file of any format (txt, png, mp3, 3gp, jpg, rar etc) into a character based pixel array and additively embedding into the image. As an illustration, a text based file is read and the text is converted into numeric signal using Unicode [23, 41]. A significant feature of the present work is that the embedding process proposed builds upon the compression techniques inherently present in the JPEG image format. [7, 8, 9].

The converted numeric signal is normalized to 128 and reshaped and is shown as a $440 \times 440 \times 3$ image in Fig.(10). The normalization to 128 ensures that an additive embedding will not result in an overflow. Each character of the text is mapped to a pixel in the image file created earlier, using a simple raster embedding pattern [42]. Thus, three characters are embedded into the Red, Green and Blue values of a single pixel.

In the present work, the text-only version of the entire story of Gullivers travels, obtained from the Gutenberg Archiving Project is used as the text to be embedded. The size of the txt file is 566kB. The text file is read, converted into numeric using ASCII, and appropriate zero padding is done to ensure size match. The data is then additively embedded into a 440x440 image, and the embedded image is shown in Fig.(11).

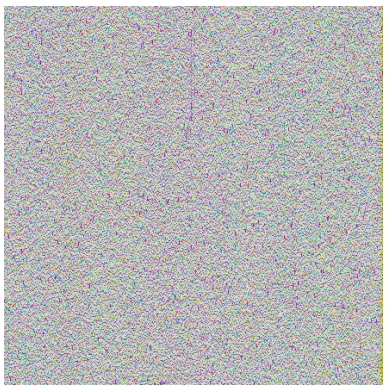

Figure 10: The text of Gulliver's Travels shown as an image

The image contains the text version of Gullivers Travels embedded onto the image shown in Fig. (11). The total size of this image in JPG format is 57.68 kB.

The decryption is performed by just subtracting the 'carrier' image shown in Fig. (9) from the embedded image of Fig.(11), followed by denormalizing and converting the signal obtained back to ASCII/Unicode text.

In the present work, the decryption performed yielded back the text only version of Gullivers travels with a size of 566kB.

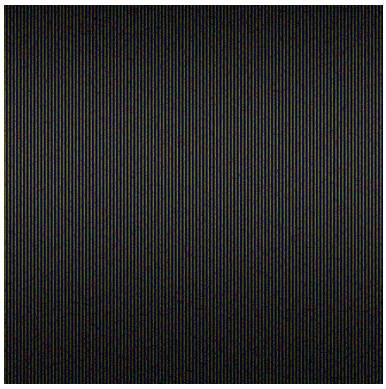

Figure 11: The chaotic image containing the text of Gulliver's Travels (440x440 png)

3. Performance Assessment

3.1. Definition of Performance Assessment Parameters

In order to assess and characterize the performance of the embedding process elaborated above, the performance is characterized with respect to three parameters, namely capacity, fidelity and security, where the performance analysis parameters are defined as follows.

3.1.1. Capacity

The Compression Ratio CR is typically defined as follows [14, 15, 16]:

$$CR = \frac{\text{Total Size of Uncompressed Data (SU)}}{\text{Total Size of Compressed Data (SC)}} \tag{7}$$

In the present work, the total size of uncompressed data SU is the size of the file to be compressed. The total size of compressed data SC is the size of the compressed image as shown in Fig.(11). The original 'carrier' image shown in Fig.(9) is considered part of the embedding process and its size is not taken into consideration for calculating CR.

3.1.2. Fidelity

The fidelity of the embedding process is characterized by the Mean Square Error between the numeric values of the original and decrypted data, normalized as a percentage of the maximum value (256 for one byte). Thus, if the original file converted to a Numeric Signal is N, and the decrypted Numeric Signal is N_o, the MSE is given as follows.

$$MSE(Percentage) = \frac{(N - N_o)^2}{128} \tag{8}$$

Though the algorithm proposed in the present work is by itself error-free, thanks to the perfectly invertible operations during encryption (addition) and decryption (subtraction), errors and fidelity errors may arise in the manner in which the platform of implementation (MATLAB/Python) handles the image and file input/output. Such errors are essentially errors due to rounding-off of data/signal values prior to normalization. In fact, this is the motivation behind a comparative study of MATLAB/Python implementations in the present work.

3.1.3. Security

In order to characterize the security, a decryption of the compressed image of the original data file is attempted, but with the frequency ratios not aligned to the ones used during encryption.

Specifically, the frequency ratio r is misaligned by a factor of 1 percent yielding the nre frequency ratio as follows:

$$rnew = r(1 + 0.01) \tag{9}$$

The factors leading to high security are enumerated as follows.

1. In the context of the proposed frequency dependent chaos, it is seen that the use of a 1 percent mismatched ratio creates a drastic difference between the original and mismatched signals, as plotted in Fig. (12).
2. This results in a drastic change in the pixel patterns of the chaotic images generated. The pixel differences between images generated using r and $rnew$ are plotted as an image in Fig. (13).
3. The MSE value obtained for a mismatched ratio ($rnew$) is noted as the percent of Mismatched MSE (MMSE). These values are compared to the MSE value obtained during correct decryption using the ratio r.

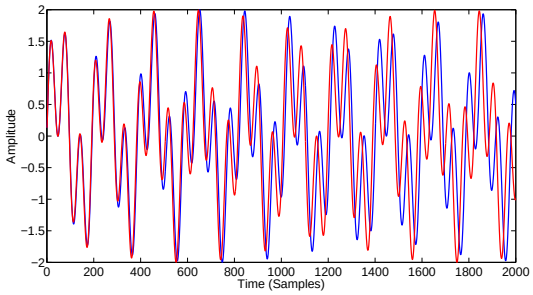

Figure 12: Difference between chaotic signals generated using ratios r (blue) and $rnew$ (red)

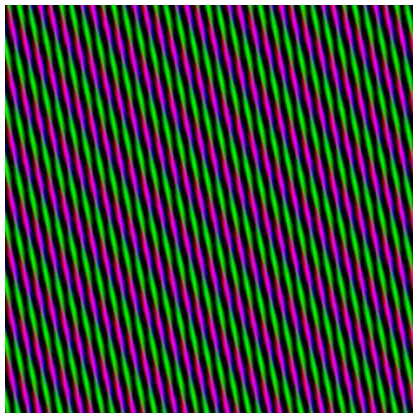

Figure 13: Difference between carrier images generated using ratios r and $rnew$

3.2. Performance Analysis for different filetypes

In order to analyse the performance of the embedding process in terms of capacity, fidelity and security, files of different types are implemented using the above mentioned embedding scheme.

3.2.1. Text and Image based Files

For text and image based filetypes, three file formats namely txt, png and pdf are evaluated. For text formats, ten different text files varying in file size and number of characters are taken, and the compression ratio and error rate are tabulated in Table 1 for MATLAB and Python implementations. Similarly, for image based files, nine PNG files of varying sizes are used and the performance is tabulated in Table 2. Mixed content of text and images are also evaluated using the PDF format and the performance is shown in Table 3.

Table 1: Performance Assessment for TXT files

Image Dimension (Pixels)	File Size (Bytes)	CR (Py)	CR (MATLAB)	MSE (Py)	MSE (MATLAB)	MMSE (Py)	MMSE (MATLAB)
120	43186	6.9520	5.0345	0	0.0006	59	61
197	116282	9.3007	6.6831	0	0.0007	64	60
294	257681	7.9705	4.9337	0	0.0009	62	63
440	580329	9.8240	5.2041	0	0.0004	58	65
550	905730	8.4801	5.1820	0	0.0010	65	67
801	1923723	8.7776	7.1864	0	0.0008	60	63
830	2066355	11.5520	6.3223	0	0.0009	61	66
1422	6063802	8.1486	4.8243	0	0.0011	62	69
1724	8915249	13.9119	7.1820	0	0.0005	67	63
2245	15118416	8.6951	5.3461	0	0.0004	65	62

Table 2: Performance Assessment for PNG files

Image Dimension (Pixels)	File Size (Bytes)	CR (Py)	CR (MATLAB)	MSE (Py)	MSE (MATLAB)	MMSE (Py)	MMSE (MATLAB)
129	49781	5.4127	5.0719	0	0.0008	58	60
249	184587	5.8941	5.7339	0	0.0005	64	61
411	505232	5.9536	5.7107	0	0.0004	63	66
548	899696	6.4139	5.7397	0	0.0006	59	65
758	1723221	6.5258	4.8606	0	0.0010	62	67
896	2407513	6.5192	4.9133	0	0.0003	60	63
1436	6181252	6.5014	4.8229	0	0.0010	61	64
1852	10286142	6.5085	4.8689	0	0.0002	62	65
2100	13228375	6.5010	4.8973	0	0.0007	67	63

From Tables 1, 2 and 3, one clearly observes a fairly consistent compression ratio of around 6 for Python implementation whereas MATLAB shows a lesser compression ratio of around 5. Also Python invariably shows an absolute zero error whereas MATLAB shows an error of 0.0005 percent. The proposed algorithm in the above section is by itself error-free since the encryption operation (addition) and decryption operation (subtraction) are perfectly reversible. However, the error in MATLAB is attributed to the manner in which MATLAB handles file input/output, where rounding off of values prior to normalization may result in quantization based errors.

3.2.2. Multimedia files

In order to evaluate the performance of the embedding process for multimedia files, two classes of files namely audio (MP3) and video (3GP) files are considered. The performance in terms of compression ratio and error rate for MATLAB and Python implementations are shown in Tables 4 and 5.

As with the text/image case, here too one observes a consistent compression ratio of around 6 for Python and 5 for MATLAB, with the corresponding error rates 0 and 0.0007 percent.

Table 3: Performance Assessment for PDF files

Image Dimension (Pixels)	File Size (Bytes)	CR (Py)	CR (MATLAB)	MSE (Py)	MSE (MATLAB)	MMSE (Py)	MMSE (MATLAB)
198	117219	6.5020	6.7405	0	0.0008	58	69
298	264967	6.4520	6.0507	0	0.0004	67	62
1747	9151604	6.1214	4.8386	0	0.0009	66	58
825	2038564	6.2170	5.0354	0	0.0004	64	62
440	580322	6.4567	5.0048	0	0.0002	62	58
150	66671	6.2578	5.7193	0	0.0003	61	63
1322	5235367	6.5902	4.8079	0	0.0009	63	59
2148	13833605	6.5447	4.8958	0	0.0010	65	67
913	2498348	6.2873	4.8870	0	0.0007	62	68
554	918160	7.1646	4.9214	0	0.0005	59	66

Table 4: Performance Assessment for MP3 files

Image Dimension (Pixels)	File Size (Bytes)	CR (Py)	CR (MATLAB)	MSE (Py)	MSE (MATLAB)	MMSE (Py)	MMSE (MATLAB)
1680	8461525	6.5759	4.8249	0	0.0003	69	58
806	1947692	6.6271	5.2112	0	0.0005	62	67
429	549703	6.2328	4.9888	0	0.0009	58	66
929	2588173	6.3504	4.9320	0	0.0004	62	64
547	897612	6.0966	5.5981	0	0.0010	65	68
1902	10849836	6.7460	4.6564	0	0.0008	63	61
135	54158	5.4821	6.0081	0	0.0009	59	63
412	506808	6.4200	6.2856	0	0.0002	67	65
1254	4714496	6.5650	4.9070	0	0.0007	68	62
248	184353	6.3521	6.4993	0	0.0004	66	59

Table 5: Performance Assessment for 3GP files

Image Dimension (Pixels)	File Size (Bytes)	CR (Py)	CR (MATLAB)	MSE (Py)	MSE (MATLAB)	MMSE (Py)	MMSE (MATLAB)
176	92647	6.2739	6.8343	0	0.0004	58	69
340	346487	6.4098	4.7551	0	0.0006	67	62
427	544734	6.0073	5.0451	0	0.0007	66	58
805	1939487	6.2092	4.7698	0	0.0005	63	67
851	2167902	15.6890	5.3318	0	0.0008	68	65
1405	5915191	6.2630	4.7890	0	0.0010	57	63
1660	8260076	8.9756	4.8993	0	0.0003	63	59
1874	10525678	8.2235	4.8760	0	0.0009	65	67

3.2.3. Mixed Files

Finally, to assess the performance of embedding on mixed content combining text, images and multimedia, the RAR compressed format is chosen. As in the previous cases, the performance in MATLAB and Python is tabulated in Table 6 and the corresponding compression ratios are obtained at around 5 and 6 respectively, with the error rates 0.0007 percent and 0.

Table 6: Performance Assessment for RAR files

Image Dimension (Pixels)	File Size (Bytes)	CR (Py)	CR (MATLAB)	MSE (Py)	MSE (MATLAB)	MMSE (Py)	MMSE (MATLAB)
119	41824	5.2266	4.7478	0	0.0007	61	59
244	178302	6.2110	6.8220	0	0.0006	60	64
294	259234	6.3506	5.0456	0	0.0004	63	62
445	592515	6.1093	5.3576	0	0.0009	65	58
521	812246	6.1229	5.3857	0	0.0008	67	65
677	1373307	6.1462	5.0127	0	0.0010	63	60
958	2750180	6.5497	4.8910	0	0.0012	66	61
1657	8233656	6.2213	4.9141	0	0.0009	69	62
2059	12717494	6.0916	4.8824	0	0.0004	63	67

4. Inferences and Discussion

The performance assessment of the proposed embedding process for different filetypes elaborated in Table 1 - 6 and the consistent performance obtained therein in terms of compression ratio and error rates testify to the fact that the proposed algorithm is independent of filetype, language, size etc. The only implementation related differences arising in the performance occur due to the differences in the file handling capabilities of MATLAB and Python as mentioned earlier. However, it is also evident that of the two implementations, the Python implementation stands out as more efficient, in terms of both higher compression ratio and lower error rate.

4.1. Compression Ratio

To ascertain the efficiency of the embedding process in terms of the compression ratio, the typical values of compression ratios achieved and reported in literature for standard compression and encoding schemes are compared platform agnostically [9, 2, 18, 22, 23].The state-of-art compression/encoding schemes are broadly classified under three headings:

1. Lossless Compression Techniques which provide very high fidelity with low to moderate amount of security. Among such techniques portable network graphics (png) and the Netapp Snapmirror report compression ratios of 3.5.
2. Lossy Compression where with moderate amount of loss and security, high compression ratios are achieved. The Joint Photographic Experts Group (JPEG) report compression ratios of around 3.2 with a tolerable MSE of 5 to 10 percent whereas Moving Picture Experts Group (MPEG) report compression ratios of around 10 to 20 for the same MSE tolerance.
3. Encryption where security is of paramount importance, and very rarely good compression ratios are achieved. The most popular encryption format, namely Cyclic Redundancy Check (CRC32) reports no compression owing to redundancies created due to extra parity bits.

As can be seen, the compression ratio of around 6 obtained for the proposed algorithm is in par, if not higher than most state-of-the-art compression techniques combining the best features of encryption, lossless and lossy compression techniques. The relatively high compression ratio can be attributed to the following factors.

1. The JPEG (jpg) format, has been shown to achieve an inherent compression ratio of around 3 for most typical data[9].
2. The chaotic 'carrier' image generated in Fig.(9) effectively exploits the enormous storage offered by the color space. Since each of the Red, Green and Blue components of a pixel can store a 8 bit value (maximum capacity of 255 each), the total combination of colors possible is 16.5 million. In order to validate this statement, the embedding process elaborated earlier has been carried out by replacing the 'carrier' image in Fig. (9) by a pure black image, with absolutely no color distribution. It is found that the compression ratio is this case is 3.2 which is close to the inherent compression ratio of the jpg format itself.

3. The frequency ratio of π selected for embedding plays a major role in boosting the compression ratio. The concept can be intuitively understood as follows. In a purely amplitude dependant system consisting of two signals V_1 and V_2, there are two variables V_1 and V_2 for storage of data. However, in a frequency dependant chaotic system consisting of V_1 and V_2, the storage variables are the frequencies f_1 and f_2 as well as the mixing products $2f_1$, $2f_2$, $3f_1$, $3f_2$, $2f_1 - f_2$, $2f_2 - f_1$, $2f_1 + f_2$, $2f_2 + f_1$ and so on, thus offering a steep improvement in storage space compared to the amplitude case [43].

A typical data file, when represented as a numerical signal has multiple frequency components, as suggested by successful application of frequency dependant distributions such as Zipf's Law [44], and it is easy to visualize how certain frequency components of the text easily match with the corresponding counterparts in the image, such matches creating redundancies which will be compressed greatly due the Lempel-Ziv LZ77 coding used in the jpg format [9]. Thus, the more chaotic the frequency ratio f_2/f_1, the more the harmonics produced by $A(t)$, the more the matches and redundancies, and the more the compression ratio CR.

In order to assess the dependance of compression ratio on the nature of chaos generated by the frequency ratio, a graph is plotted in Fig.(14) showing the variation of compression ratio CR with frequency ratios R between the integer values 3 and 4, all results obtained from the Python implementation. It is noted that the compression ratios are in agreement with the chaotic regimes at 0.2 and 0.7 mantissa values, as predicted earlier by the bifurcation and cobweb plots.

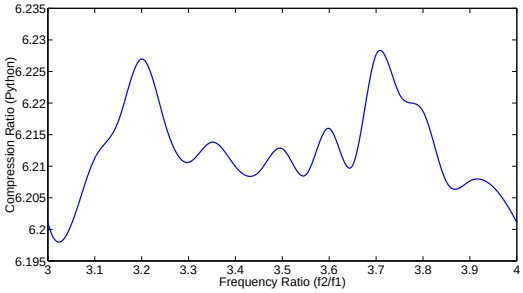

Figure 14: Variation of the compression ratio with the frequency ratio

An intuitive implication is then that the compression ratio might have a direct dependance on the Lyapunov Exponent of the generated 'carrier' signal $A(t)$, since the Lyapunov exponent numerically characterizes the 'richness' of chaos. A plot confirming this hypothesis is shown in Fig.(15). The implication is then that the rich frequency components in $A(t)$ generated by certain frequency ratio values increase the number of redundancies and thus provide an extremely conducive environment for the Lempel Ziv Coding inherent in JPEG to reduce the wordlength required to code the resultant image, and hence the compressed file size.

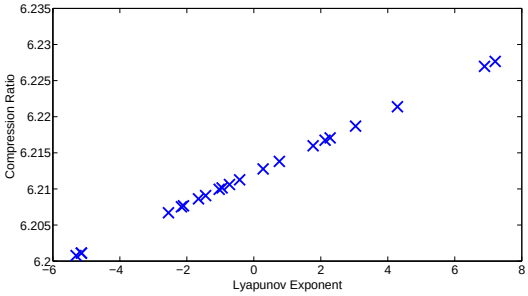

Figure 15: Variation of the compression ratio with Lyapunov exponent value of the $A(t)$ used

4.2. Security and Mismatched Mean Square Error

With regards to security, the consistently high error rates (around 60 percent) obtained for all filetypes and sizes in both Python and MATLAB implementations for 1 percent mismatch of the decompression frequency ratio R clearly highlights the extremely sensitive dependence on this control parameter, which guarantees a very high level of security regardless of the implementation platform. The key factors responsible for such a high value of MMSE are twofold:

1. The drastic mismatches in the chaotic signal and image generated using a correct ratio value r and a 1 percent misaligned ratio value $rnew$ as seen in Fig. (12) and Fig. (13) ensure that the decompression of the files using the mismatched ratios will certainly result in high MMSE values.
2. The frequencies generated using different ratios such as r and $rnew$ result in differences in the frequencies in the generated image according to the bifurcation analysis shown in Fig. (2) leading to differences in the redundant components created and eventually differences in the compressed image.

The above discussion, coupled with the almost consistent value of 60-62 percent mismatched MSE assertively establish the high degree of security offered by the proposed compression process.

4.3. Execution Time

Finally, in addition to the high capacity, security and fidelity as discussed above, another crucial factor determines the usefulness of the proposed code in real-time data handling applications - execution time. All the implementations mentioned in this work are carried out in a Core 2 Quad HP Workstation with an 8GB RAM. Among all the filetypes and sizes the maximum time for execution are seen for file sizes of 13-15 MB. In Python, this maximum execution time is 6 seconds for compression and 6 seconds for decompression, whereas in MATLAB, the corresponding maximum execution times are 8 seconds for compression and 7 seconds for decompression. The low execution times is a testimony to the simplicity of the algorithm developed, involving basic addition and subtraction operations. It is opined that in systems with more powerful data handling capabilities, the execution will be even more faster.

5. Super-Compression

From the inferences in the above section, it is understood that chaos based compression schemes achieve relatively high compression ratios with negligible error and high security taking very less times to execute. Also, the compression technique proposed converts a given data file to an image file, which can technically act as the input file for another compression iteration. It is intuitively suggested that the low execution time can be exploited to achieve higher compression ratios by repeatedly applying the compression algorithm to the compressed image. In order to validate this hypothesis, a 28.8MB video file in 3gp format is taken and is compressed. The resultant is a $3102 \times 3102 \times 3$ JPEG image of around 4MB. This image is then taken as the input for the next iteration of compression, and results in a compressed image of around 650kB, which then acts as the input file for the next iteration. This process is repeated for 10 iterations. It is noteworthy that the size of the final compressed $16 \times 16 \times 3$ JPEG image is a mere 691 bytes, yielding a compression ratio of 28.8 MegaBytes /691 bytes which is a staggering 41750 in the Python implementation. Such a repeated application of the 1D chaotic compression to a text file yielding extremely high compression ratios of the order of 40000 is termed as 'Super Compression'. The overall error rate between the original video file and final decompressed video file after 10 iterations of compression and 10 iterations of decompression is obtained as 0 in Python and 0.0006 percent in MATLAB.

A natural doubt arising over the supercompression algorithm mentioned above is the execution time it takes to run multiple iterations of compression. It is observed that for the 8GB RAM HP workstation system specifications mentioned earlier, the compression of the 28.8MB video over 10 iterations takes 45 seconds of execution time in Python and 87 seconds in MATLAB. The corresponding decompression execution times are 43 seconds in Python and 88 seconds in MATLAB.

The variation of compression ratio over multiple iterations in supercompression is plotted as a graph in Fig. (16). It is evident that in subsequent iterations the compression ratio decreases gradually. This is due to the fact that at every iteration the resultant image size is reduced, leading to a lesser number of pixels for the next iteration. Lesser number of pixels implies lesser number of spatial frequencies and hence a lesser amount of storage space in the frequency domain. Finally, at the ninth and tenth iteration it is seen that the file size attains a minimum saturation point of 691 bytes, beyond which it cannot be compressed any more.

6. Conclusion

A frequency dependant chaotic system is proposed and characterized and a chaotic image generated from such a system is used as a storage 'carrier' to securely embed textual data. The performance assessment of the embedding process revealed the following vital points:

1. High fidelity characterized by MSE of around 0.0009 percent for MATLAB and absolute zero for Python implementations are obtained regardless of data size or file type.
2. The Compression ratio is obtained consistently at around 6 for Python and around 5 for MATLAB regardless of filetype or size. Also, repeated application of the compression algorithm yields extremely high compression ratios of the order of 40000.

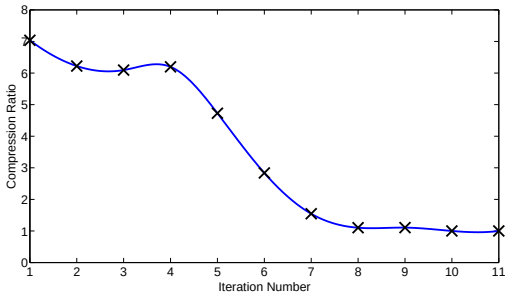

Figure 16: Variation of the compression ratio for subsequent iterations in supercompression

3. The execution times for the compression and decompression are obtained reasonably at around 6-7s for filesizes of around 15MB and around 50s for supercompression for a file size of 28.8MB in an 8GB workstation. These times depend on the platform of implementation, with the algorithm itself taking minimal time to execute, thanks to the simplicity of the proposed embedding technique.
4. Extreme sensitivity to frequency ratios is seen, with a 1 percent misalignment in the frequency ratio causing an MSE as high as 63 percent, and this assertively establishes the security offered by the proposed technique.

Though a detailed and thorough investigation of the mechanism of chaos generation and the embedding process awaits, the results discussed in the present work establish the fact that this embedding process serves as a one-step compression plus encryption process, guaranteeing the twin advantages of high capacity and high security. These advantages, coupled with the extreme simplicity of implementation form the crux of the present work. The embedding process used in the present work thus opens the doors for a golden new era, the era of 'Affordable Big Data'.The hallmarks of such an era would be the usage of extremely simple yet effective techniques for securely compressing and supercompressing Big Data such as DNA Genome Sequences, Automated Sensor Data, Financial Records and other Multimedia based Signals. Another immediate application of the supercompression is to circumvent the maximum file size limit posed in email attachments and other cloud computing services.

References

[1] M. Hilbert, *How much of the global information and communication explosion is driven by more, and how much by better technology?*, Wiley Journal of the Association for Information Science and Technology, **65**, 856-861 (2014).
[2] G. B. Giannakis, F. Bach, R. Cendrillon, M. Mahoney,J. Neville, *Signal Processing for Big Data*, IEEE Signal Processing Magazine, **31**, 15-16 (2014).
[3] T. White, Hadoop: *The Definitive Guide*, (O"Reilly, 2010).
[4] X.Wu, X.Zhu, G.Q.Wu and W.Ding, *Data mining with big data*, IEEE Trans. on Knowledge and Data Engineering **26**,97-107 (2014).
[5] E.Schouten, *IBM SmartCloud Essentials*, (Packt, 2013).
[6] A.McEwan and H.Cassimally, *Designing the Internet of Things*, (Wiley, 2013).
[7] F. Wu, *Advances in Visual Data Compression and Communication: Meeting the Requirements of New Applications*, (CRC Press, US, 2014).
[8] D. Salomon, D. Bryant and Giovanni Motta, *Handbook of Data Compression*, (Springer, California, 2010).
[9] D.S. Cruz, C. Ebrahimi, J. Askelof, M. Laarsson, C. A. Chritsopoulos, *JPEG 2000 still image coding versus other standards*, SPIE Digital Image Processing XXIII, **4115**, 446-454 (2000).
[10] K. E. Barner and G. R. Arce, *Nonlinear Signal and Image Processing: Theory, Methods, and Applications*, (CRC Press, U.S, 2003).
[11] A. M. Rufai, G. Anbarjafari, H. Demiral, *Lossy image compression using singular value decomposition and wavelet difference reduction*, Elsevier Digital Signal Processing, **24**, 117-123 (2014).
[12] E.Bilotta and P.Pantano, *A gallery of Chua attractors*, (World Scientific, Singapore, 2008).
[13] S. T. Welstead, *Fractal and Wavelet Image Compression Techniques*, (SPIE, US, 1999).
[14] M. F. Barnsley, A. D. Sloan, *Chaotic Compression*, Computer Graphics World, **3** (1987).
[15] Y. Fisher, *Fractal Image Compression*, Fractals, **2** (1994).
[16] M. F. Barnsley, *Fractals Everywhere*, (Courier, Dover, 2008).
[17] Z. Fan, R. L. D. Queiroz, *Identification of bitmap compression history: JPEG detection and quantizer estimation*, IEEE Trans. Image Processing, **12**, 230-235 (2003).
[18] C. A. Poynton, *Digital Video and HDTV: Algorithms and Interfaces*, (Morgan Kauffman, US, 2003).
[19] K. E. Himma, *Internet Security: Hacking, Counterhacking, and Society*, (Jones and Bartlett, UK, 2007).
[20] R. Broadhurst, P. Grabosky, M. Alazab, S. Chon, *Organizations and Cyber crime: An Analysis of the Nature of Groups engaged in Cyber Crime*, Int. J. Cybercriminology, **8** (2014).
[21] N.Hopper, L.VonAhn and J.Langford, *Provably secure steganography*, IEEE Trans. on Computers **58**,662-676(2009).
[22] P. H. Mahajan, P. B. Bhalerao, *A Review of Digital Watermarking Strategies*, International Journal of Advanced Research in Computer Science And Management Studies, **7** (2014).
[23] D. Salomon, *Data Compression: The Complete Reference*, (Springer, US, 2004).

[24] M. H. Y. Fritz, R. Leinonen, G. Cochrane, E. Birney, *Efficient storage of high throughput DNA sequencing data using reference-based compression*, Genome Research, **21** 734-740, (2011).
[25] S. H. Strogatz, *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering* ,(Westview Press, Cambridge, 2008).
[26] M. Ausloos, M. Dirickx, *The Logistic Map and the Route to Chaos: From the Beginnings to Modern Applications*, (Springer, US, 2006).
[27] M. H. Jensen, P. Bak, T. Bohr, *Complete Devil's Staircase, Fractal Dimension, and Universality of Mode- Locking Structure in the Circle Map*, Phys. Rev. Lett. **50**, 1637 (1983).
[28] R. Gilmore and M. Lefranc, *The Topology of Chaos*, (Wiley,US, [2002]).
[29] J. M. T. Thompson and H. B. Stewart, *Nonlinear Dynamics and Chaos* (Wiley,UK, [2002]).
[30] V. I. Arnold, *Small Denominators I. Mappings of the circumference onto itself*, Am. Math. Soc. Transl, **46**, 213-284 (1965).
[31] M. R. Hermann, *Mesure de Lebesgue et nobre de rotation (Geometry and Topology)*, (Springer,Germany, [1977]).
[32] R. S. Mackay and C. Tresser, *Some flesh on the skeleton: the bifurcation structure of bimodal maps*, J. Phys. Lett., **45**, L741-L746 (1984).
[33] B. Razavi, *RF Microelectronics*, (Prentice Hall, US, 2011).
[34] R. M. May, *Regulation of populations with nonoverlapping generations by microparasites: a purely chaotic system*, American Naturalist, **85**, 573-584, (1985).
[35] U. S. Freitas, C. Letellier, L. A. Aguirre, *Failure in distinguishing colored noise from chaos using the noise titration technique*, Physical Review E, **79**, 035201, (2009).
[36] F. Doveri, M. Scheffer, S Rinaldi, S. Muratori, Y. Kuznetsov, *Seasonality and chaos in a plankton fish model*, Theoretical Population Biology, **43**, 159-183, (1993).
[37] R. G. James, K. Burke, J. P. Crutchfield, *Chaos forgets and remembers: Measuring information creation, destruction, and storage*, Int. J Bifurcation Chaos, **378**, 2124-2127, (2014).
[38] M. T. Rosenstein, J. J. Collins, C. J. De Luca, *A practical method for calculating largest Lyapunov exponents from small data sets*, Physica D, **65**, 117-134, (1993).
[39] P. C. Hansen, *Regularization Tools Version 4.0 for MATLAB 7.3*, Numer. Algor., **46**, 189-194, (2007).
[40] P. Maragos, F. K. Sun, *Measuring the fractal dimension of signals: morphological covers and iterative optimization*, IEEE Trans. Signal Processing, 41, 108-121 (1993).
[41] J. Korpela, *Unicode Explained*, (O'Reilly, US, 2006).
[42] T. H. Lan, M. F. Mansour, A. H. Tewfik, *Robust high capacity data embedding*, Image Processing 2000, **1**, 581-584 (2000).
[43] J. C. Pedro, N. B. Carvalho, *Intermodulation Distortion in Microwave and Wireless Circuits*, (Artech House, US, 2002).
[44] W. Li, *Random texts exhibit Zipf's-law-like word frequency distribution*, IEEE. Trans. Information Theory, **38**, 1842-1845 (1992).