algorithms

MDPI

Article

A New Way to Store Simple Text Files

Marcin Lawnik *, Artur Pełka and Adrian Kapczyński

Faculty of Applied Mathematics, Silesian University of Technology, Kaszubska 23, 44-100 Gliwice, Poland; artur260@op.pl (A.P.); adriank@polsl.pl (A.K.)

* Correspondence: marcin.lawnik@polsl.pl

Received: 24 March 2020; Accepted: 20 April 2020; Published: 22 April 2020

Abstract: In the era of ubiquitous digitization, the Internet of Things (IoT), information plays a vital role. All types of data are collected, and some of this data are stored as text files. An important aspect—regardless of the type of data—is related to file storage, especially the amount of disk space that is required. The less space is used on storing data sets, the lower is the cost of this service. Another important aspect of storing data warehouses in the form of files is the cost of data transmission needed for file transfer and its processing. Moreover, the data that are stored should be minimally protected against access and reading by other entities. The aspects mentioned above are particularly important for large data sets like Big Data. Considering the above criteria, i.e., minimizing storage space, data transfer, ensuring minimum security, the main goal of the article was to show the new way of storing text files. This article presents a method that converts data from text files like *txt*, *json*, *html*, *py* to images (image files) in *png* format. Taking into account such criteria as the output size of the file, the results obtained for the test files confirm that presented method enables to reduce the need for disk space, as well as to hide data in an image file. The described method can be used for texts saved in extended ASCII and UTF-8 coding.

Keywords: Big Data; text file; png image; compression; steganography

1. Introduction

The amount of information we deal with in the present world is enormous. Evidence of this is the growing interest in so-called Big Data, i.e., large data sets for which traditional methods of analysis and processing are ineffective [1]. Often, those data are of a different type: text, graphic, etc., and come from various sources, e.g., social media [2] or machine data [3]. Regardless of the nature of the data and its belonging to the Big Data, the storage method plays a critical role. Some of these data are stored on private disks or servers, and some use cloud services (e.g., Google Drive, Dropbox, or Microsoft OneDrive). In the case of cloud solutions or those where we do not have direct supervision over our data, there is a risk of loss or modification of stored data [4]. Of course, the service provider is responsible for these issues; however, relying solely on this may prove to be deceptive, which is confirmed by information from time to time about data leaks or internet portals defacement [5]. For this reason, it is worth hiding or encrypting data that are confidential. Confidentiality can be provided by the use of steganography as well as cryptography. Steganography allows to hide data in such a way that their presence cannot be detected by humans [6]. Unlike steganography, cryptography transforms the output in such a way that its content is not hidden [7,8].

Big Data, due to its features, i.e., volume, variety, and velocity, requires different storage than classical databases. These data require a set of servers working as a parallel system to store and to access the data for further processing. NoSQL is the type of database that is characterized as non-relational, distributed, and scalable, thus making them suitable for Big Data applications. There are more features of NoSQL databases, including easy replication, schema-free, and BASE, which are precisely described in [9].

Big Data storage can be placed within Big Data value chain [10] on the fourth level, after data acquisition, data analysis, and data curation and naturally before data usage. The review of the current state-of-the-art of data storage technologies brings the following types of storage systems:

- Distributed File Systems, e.g., [11],
- NoSQL Databases, e.g., [12],
- Big Data querying platforms, e.g., [13],
- NewSQL Databases, e.g., [14,15].

Big Data storage systems can be characterized by specifying both strengths and weaknesses, which were presented in [16]. Among the advantages, there is the support of heterogeneous structured data, simultaneous accessibility, and high fault tolerance. On the other hand, there are some weaknesses, for example, the lack of compliance with ACID set of properties that guarantee database transactions reliability (ACID stands for Atomicity, Consistency, Isolation, and Durability).

In case of large distributed file systems, it is essential to consider the size of the stored file, as well as its security attributes. One of the compression methods can be applied to reduce the size of the file. At the same time, one of the critical security attributes, namely confidentiality, can be set by the use of cryptography or steganography.

The main contribution of this article is to presents a method of storing text data to ensure their compression as well as to hide their content directly. It involves changing the data format from text to graphic data with the extension *png*. By appropriate transformation, the content of the text file is transformed into the file *png*, the content of which looks like a random string. To the best of our knowledge, no one has proposed such a solution before.

The article is divided into four parts. The first part consists of an introduction and an overview of related works. In the next section, the method of converting text files to the *png* graphic format was presented. Later, an example of the application of the specified method has been discussed. The last part of the article consists of a summary and a list of references.

1.1. Related Work

1.1.1. Compression

Another problem related to storing data on external platforms is the volume that they occupy. It is especially important in the case of huge data volumes, which we deal with within the context of Big Data [16]. The smaller the data set is, the faster it will be processed, and the cost of the storage service provided is lower. This cost includes not only the cost of disk space but also the cost associated with the data transfer process. For this reason, stored data should be compressed, which will optimize costs. The compression used should be lossless, i.e., the data after decompression should have the same content as the data before compression [17]. Examples of file types that accomplish this task include *png* [18] (for image files) or *zip* [19], which is a solution for all file types.

1.1.2. Steganography of the Text

Discussed method concerns (among other issues) steganography of the text. In the literature, one can find many works on this subject, e.g., [20–27]. Most of them use large amounts of non-confidential information (media) to be able to hide in a piece of the confidential message.

In [20], a technique that uses reflection symmetry of the English alphabets was used for this purpose. Thanks to horizontal or vertical symmetry of the alphabet characters, the corresponding bits of the confidential message are hidden, creating appropriately modified text. In turn, Reference [21] developed a method of hiding text using the omega network. This method consists of generating a word from a dictionary that contains two letters coupled using an omega network with one letter of messages to hide. Yet another method of hiding text is in [22]. It uses Huffman coding in such a

way that a secret message is attached to the received codes. In [23], a multi-keywords carrier-free text steganography method based on the part of speech tagging is discussed. The work [24], in turn, presents coverless plain text steganography based on the parity of Chinese characters' stroke number. On the other hand, in [25], the authors showed how to hide a secret message in the formatting (e.g., color) of workbook files *xls*. In [26] a method of hiding text in a properly constructed number system is presented. Besides, email-based text steganography is presented in [27]. The method relies on hiding the secret data within several email addresses.

Another approach is to hide text in a format other than text, e.g., in image files. The basic way that uses images to hide a confidential message is called LSB (Least Significant Bit) method [28]. It uses the least significant bits of the pixel RGB color components to enter the bits of a confidential message.

1.2. Graphic Format png

The PNG (Portable Network Graphics) file format is a raster image file format that also provides lossless compression of image data. It allows saving data in two variants: RGB (24-bit palette) and RGBA (32-bit palette). Compression was provided by the LZ77 algorithm [29] and Huffman coding [30]. Besides, just before compression, there is a new stage called filtering. It consists of preparing data for the best compression. This is done by applying a combination of filters that simplifies data for faster data writing. An example of pixel structure of the file *png* is presented on Figure 1. The *png* format itself is not patented and is free. More about it can be found, among others, in the specification [18].

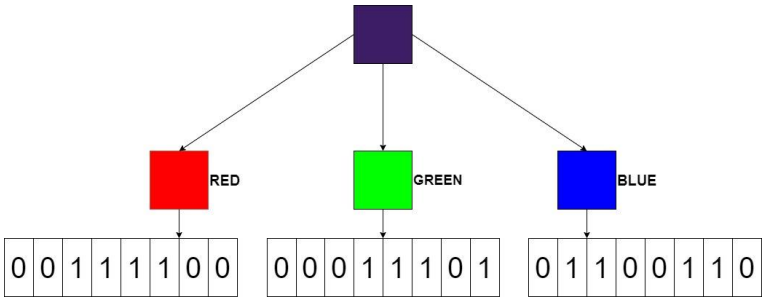

Figure 1. Representation of structure of pixel in *png* image.

2. Method

2.1. Variant 1: Text Encoded in Extended Ascii

The method of replacing a text file with a *png* file requires that each character in the text file be saved using 8-bit encoding. This means that some content (encoding other than 8 bits) cannot be directly converted to the format *png*. This is because the RGB color components of the *png* file are values in the 0–255 range, i.e., saved in 8 bits. However, the occurrence of an inappropriate character (outside the range: 0–255) should not affect the further processing of the file. This character will be—of course—incorrectly encoded, and one will not be able to regain its correct value.

The method of converting a text file saved with extended ASCII encoding into an image file *png* can be described by the following steps:

1. The text file is loaded into the buffer.
2. The following are added to its content: *ETX*, filename with the extension, *ETX*, where *ETX* means *end of text* and is encoded with an ASCII code of 3.
3. The image size (height and width) is calculated using the formula

$$size = \left\lceil \sqrt{\frac{len}{3}} \right\rceil, \tag{1}$$

where *len* is the length of text from step 2 and $\lceil \cdot \rceil$ means the ceiling function.

4. If the length of the text is less than
$$3size^2, \tag{2}$$
the content is appended with random values, so that it is exactly (2).
5. A two-dimensional array $T_{size \times size}$ is created, with elements consisting of 3-element tuples.
6. Text characters are converted to 8-bit numbers (according to Extended ASCII encoding) and divided into 3-fold tuples. The next tuples are saved as the next elements of the two-dimension array T.
7. The two-dimensional array of tuples T is treated as a pixel array and saved to the format *png*. Writing to *png* is possible using ready–made programming tools, like Python library called Pillow [31].
8. (optional) The filename is the abbreviation obtained from the filename using the selected hash algorithm, e.g., SHA3 [32].

The above method of converting a text file into a graphic *png* is reversible. Replacement of the file with *png* extension with text file can be described by the following steps:

1. The following pixels of the graphic file are loaded to get values representing the data of the text file until a pixel with an RGB color component equal to 3 is found.
2. The RGB components of the loaded pixels are perceived as consecutive ASCII characters and saved in a text file.
3. The next pixels are read to get the values of ASCII characters representing the name and file extension, to come across another RGB component value of 3.

2.2. Variant 2: Utf8 Coded Text

If the characters in the file fall outside the extended ASCII encoding range, the UTF-8 encoding can be used. This means that significant changes must be made to the algorithm of writing and reading from Section 2.1. Besides, as in the variant with ASCII coding, if a character occurs outside of the UTF-8 range, one will get the same situation, i.e., the character will be incorrectly processed, and its original value cannot be restored. In this variant, the compression of the resulting file, which we had in the case of extended ASCII encoding, is not always preserved.

The method of converting a text file saved with UTF-8 encoding into an image file *png* can be described by the following steps:

1. The text file is loaded into the buffer.
2. The following are added to its content: *ETX*, filename with the extension, *ETX*, where *ETX* means *end of text* and is encoded with an ASCII code of 3.
3. The image size (height and width) is calculated using the formula
$$size = \left\lceil \sqrt{\frac{2}{3} len} \right\rceil, \tag{3}$$
where *len* is the length of text from step 2 and $\lceil \cdot \rceil$ means the ceiling function.
4. If the length of the text is less than
$$\frac{3}{2} size^2, \tag{4}$$
the content is appended with random values to be exactly (4).
5. A two-dimensional array $T_{size \times size}$ is created, with elements consisting of 3-element tuples.
6. Text characters t_i are converted to numbers from 0–65,535 (according to UTF-8 encoding).
7. Each of the values of t_i is stored in a positional system with a basis of 256 according to the equation:
$$t_i = a_i \cdot 256 + b_i, \tag{5}$$
where a_i and b_i are the coefficients for writing the number t_i in a system based on 256.

8. The values a_i and b_i are written in tuples of length 3. The next tuples are saved as the next elements of the two-dimension array T.
9. The two-dimensional array of tuples T is treated as a pixel array and saved to the format *png*. Writing to *png* is possible using ready-made programming tools such as Python library called Pillow [31].
10. (optional) The filename is the abbreviation obtained from the filename using the selected hash algorithm, e.g., SHA3 [32].

The above method of converting a text file into a graphic *png* is reversible. The following steps can describe the replacement of the file with *png* extension with a text file:

1. The following pixels of the graphic file are loaded in order to get values representing the data of the text file until a pixel with an RGB color component equal to 3 is found.
2. From the RGB components of the pixels every two values marked as a_i and b_i are consecutive taken. These values are the coefficients of the number t_i written in a numerical system based on 256, i.e.,
$$t_i = a_i \cdot 256 + b_i. \tag{6}$$
3. The values of t_i are perceived as subsequent UTF-8 characters and saved in a text file.
4. The next pixels are read to come across another RGB component value of 3, and as in step 2, the next every two values marked as a_i and b_i are used to compute t_i values with (6). The resulting string is the name and extension of the text file.

3. Analysis And Discussions

3.1. Limitations

The method of replacing text with the graphic file *png* consists of saving the text characters to the RGB values of individual pixels. Therefore, some file formats, such as the *doc* extension, cannot be directly processed with it. This is because the addition to the text content itself also includes the formatting of individual elements. Among others, the following formats can be directly used with this method: *.txt*, *.tex*, *.js*, *.html*, *.json* (also *.ipynb*), *.py*, *.css*. Note, that there are also some problems one may encounter while trying to read files with *doc* file format. They require specialized libraries for this purpose, e.g., *textract* [33].

Besides, the conversion of a text file into a graphic file is possible for text files encoded with extended ASCII and UTF-8 characters. In these cases, there are some differences in the way of saving and reading the pixels of the output image file.

3.2. Case Study

The proposed method converts any text file encoded using extended ASCII or UTF-8 format to the *png* file format. Thus, different formats are reduced to one. In fact, this effect can be seen as desirable when viewed from the point of large volume data sets.

The following files were selected for processing using the proposed method-as well as for further analysis:

1. data1.txt: a text file containing some English text encoded in extended ASCII
2. data2.txt: a text file containing some Polish text encoded in UTF-8
3. data3.txt: a text file containing 10^6 random digits
4. data4.json: a text file in the format *json* containing the code of the *.ipynb* version of *python.py* file
5. python.py: Python file which content is the source code published on Github platform
6. latex.tex: the file containing the latex source code of this article

All of the presented algorithms have been implemented in the Python programming language. The source code responsible for the conversion (file with text content into a graphic format) has been shown on the Github platform. The URL to the source code repository can be found in Appendix A. The analysis of the time of changing the text format into an image in the *png* format was made on a personal computer with:

A Microsoft Windows 10 equipped with an Intel(R) Core(TM) i7-8565U CPU and 16 GB of RAM;
B Linux Ubuntu 18.04.4 LTS equipped with Intel(R) Core(TM) 2 Duo T7200 CPU and 2 GB of RAM;
C MacOS Catalina 10.15.3 equipped with Intel(R) i7-3667U CPU and 8 GB of RAM.

Figures 2 and 3 show the results of application of proposed method in variant 1 (extended ASCII coding) and variant 2 (UTF-8 coding) for test file *data1.txt*.

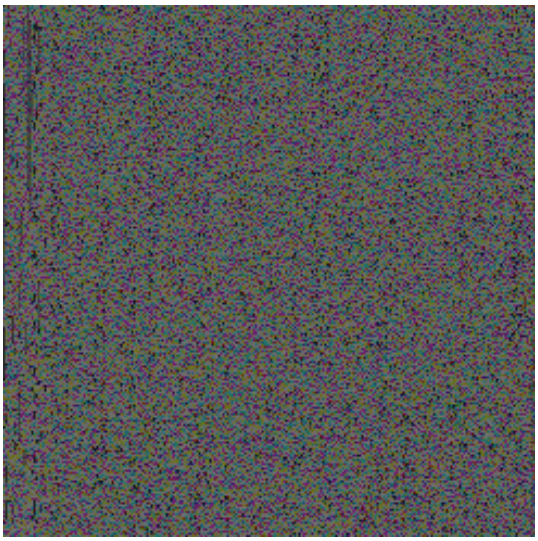

Figure 2. Result for file *data1.txt* with variant 1 (extended ASCII coding).

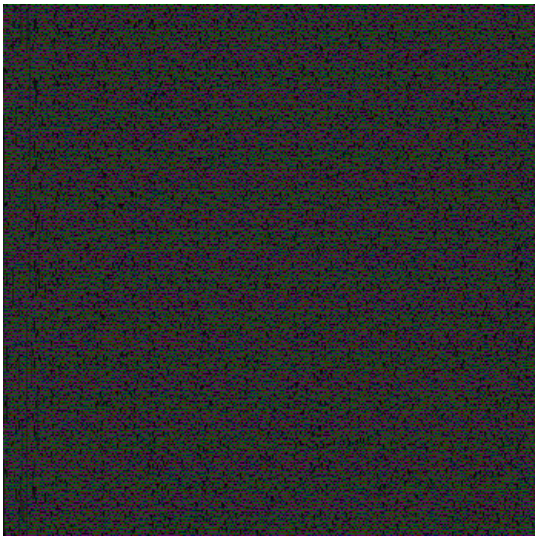

Figure 3. Result for file *data1.txt* with variant 2 (UTF-8 coding).

3.3. Compression

The basic measure determining the degree of compression is the **Compression Ratio**, which is described by the following equation [17]:

$$\text{Compression Ratio (CR)} = \frac{\text{size of the output stream}}{\text{size of the input stream}}. \quad (7)$$

The smaller the *CR* value, the less disk space the compressed file takes. The results for selected files with different extensions are provided in the Tables 1 and 2.

Table 1. Results for selected test files using variant 1 of the method (extended ASCII coding). The list of columns includes filename, size of the text file, PC specification identifier, size of the *png* file, CR, average time needed to create the *png* file and standard deviation of the time. The script has been run 100 times.

File	Text File Size (B)	PC	*png* Size (B)	CR	Avg Time (s)	std Time (s)
data1.txt	184,401	A	156,621	0.8493	0.0836	0.0014
		B	156,630	0.8493	0.2687	0.0052
		C	156,625	0.8493	0.1749	0.0127
data3.txt	1,091,008	A	569,894	0.5223	0.5565	0.0083
		B	569,898	0.5223	1.6488	0.0183
		C	569,858	0.5223	1.6279	0.7378
data4.json	12,563	A	9614	0.7652	0.0065	0.0010
		B	9615	0.7653	0.0196	0.0016
		C	9618	0.7655	0.0130	0.0017
python.py	8740	A	7197	0.8234	0.0044	0.0001
		B	7199	0.8236	0.0126	0.0012
		C	7197	0.8234	0.0077	0.0009
latex.tex	38,153	A	32,006	0.8388	0.0194	0.0012
		B	31,986	0.8383	0.0674	0.0023
		C	31,985	0.8383	0.0467	0.0044

Table 2. Results for selected test files using variant 2 of the method (UTF-8 coding). The list of columns includes filename, size of the text file, PC specification identifier, size of the *png* file, CR, average time needed to create the *png* file and standard deviation of the time. The script has been run 100 times.

File	Text File Size (B)	PC	*png* Size (B)	CR	Avg Time (s)	std Time (s)
data1.txt	184,401	A	194,749	1.0561	0.2804	0.0047
		B	194,761	1.0561	0.5984	0.0088
		C	194,734	1.0560	0.3804	0.0504
data2.txt	252,640	A	196,413	0.7774	0.3885	0.0057
		B	196,400	0.7773	0.8292	0.0158
		C	196,394	0.7773	0.5325	0.0696
data3.txt	1,091,008	A	588,732	0.5396	1.6277	0.1736
		B	588,751	0.5396	3.7014	0.0328
		C	588,741	0.5396	2.2655	0.1097
data4.json	12,563	A	9990	0.7951	0.0169	0.0024
		B	10,008	0.7966	0.0469	0.0031
		C	10,015	0.7971	0.0281	0.0028
python.py	8740	A	7162	0.8194	0.0112	0.0007
		B	7163	0.8195	0.0282	0.0023
		C	7159	0.8191	0.0178	0.0015
latex.tex	38,153	A	30,668	0.8038	0.0507	0.0044
		B	30,684	0.8042	0.1345	0.0081
		C	30,676	0.8040	0.0786	0.0053

The resulting values show that the corresponding *png* images have significantly smaller sizes (in the case of extended ASCII files and most test files in UTF-8 coding), which means savings in the necessary disk space. This is due to the way the *png* format works, i.e., the content is lossless compressed. As it can be noticed, the results obtained are worse than for other compression methods (see Table 3). However, in the case of the *data3.txt* test file, which consists of digits only, the CR compression value does not reflect that much. This case of the test file is similar in construction to Big Data. Nevertheless, those compression methods do not allow to hide the content of the file, which is discussed in Section 3.5.

Table 3. Results for selected test files using different file compressing methods. The list of columns includes filename, size of the text file, compression method, size of the compressed file and CR.

File	Text File Size (B)	Compression Method	Compressed File Size (B)	CR
data1.txt	184,401	zip	70,990	0.3849
		bz2	60,433	0.3277
		gz	74,393	0.4034
data2.txt	252,640	zip	99,364	0.3933
		bz2	81,066	0.2298
		gz	103,412	0.4093
data3.txt	1,091,008	zip	479,335	0.4393
		bz2	458,533	0.4202
		gz	499,474	0.4578
data4.json	12,563	zip	2567	0.2043
		bz2	2518	0.2004
		gz	2512	0.1999
python.py	8740	zip	2202	0.2519
		bz2	2221	0.2541
		gz	2128	0.2434
latex.tex	38,153	zip	12,684	0.3324
		bz2	12,609	0.3304
		gz	13,184	0.3455

3.4. Application in Data Transfer

This method can be used to reduce the amount of data transfer needed to perform operations. Using a *png* file format instead of a text file format, will allow to limit the amount of data needed to be sent and downloaded from the server. When storing files on the Internet, one can use various pricing plans that have a specific amount of storage space. Using this method will save space needed to store this file, by resulting compression, which will have consequences: saving money or saving disk space that can be used for subsequent files.

3.5. Application in Steganography

The presented method, apart from compression, also allows hiding the contents of a text file. The following is because the content of processed files in the format *png* looks random at first glance. This is confirmed by the sample Figures 2 and 3. A secondary observer is not able to directly extract the content contained in the image. Of course, if the graphic file is created as described in the article, then it may try to read it using the reversed method. It should be noted that the reverse method, i.e., the transition from image to text, is fully reversible. Thus, its effectiveness is 100%. However, the proper determination of the method of image hiding or encryption (in the case when the file was found to be an encrypted image) does not have to be so simple, which suggests a large number of algorithms addressing this issue, e.g., [34–39]. Table 4 shows the transition times from the graphic file to the text file. At every turn, the script has been executed 100 times, and the results show the average time and standard deviation. The results show that the transition from image to text is not a time-consuming process.

Compared to files compressed and stored in the format *zip*, the advantage of hiding the file's content is undoubted. A computer system can easily open the *zip* file. It means that we have direct access to the content of the text file. However, to make a change to a compressed file, it must be firstly decompressed. Though, the proposed method allows reading a specific pixel *png* file and its modification (only those particular pixel).

Table 4. The results show the transition times from the image *png* to the text file. The list of columns includes filename, the time needed to create the text file, PC A, PC B, and PC C. The script has been run 100 times. Avg and std are respectively: mean time and standard deviation.

File	Encoding	Time	A	B	C
data1.txt	ASCII	avg	0.1840	0.4015	0.2518
		std	0.0025	0.0052	0.0118
	UTF-8	avg	0.0718	0.1547	0.1104
		std	0.0013	0.0059	0.0068
data2.txt	ASCII	avg	-	-	-
		std	-	-	-
	UTF-8	avg	0.0955	0.2026	0.1375
		std	0.0016	0.0058	0.0047
data3.txt	ASCII	avg	2.4929	5.2552	3.5691
		std	0.0076	0.0373	0.9982
	UTF-8	avg	0.4629	0.9612	0.6622
		std	0.0076	0.0127	0.0381
data4.json	ASCII	avg	0.0058	0.0127	0.0085
		std	0.0012	0.0002	0.0021
	UTF-8	avg	0.0058	0.0115	0.0078
		std	0.0001	0.0025	0.0021
python.py	ASCII	avg	0.0039	0.0084	0.0059
		std	0.0009	0.0011	0.0015
	UTF-8	avg	0.0044	0.0075	0.0055
		std	0.0001	0.00147	0.0008
latex.tex	ASCII	avg	0.0240	0.0507	0.0319
		std	0.0011	0.0008	0.0026
	UTF-8	avg	0.0165	0.0332	0.0230
		std	0.0007	0.0008	0.0019

As described above, during the last step of the algorithm, the filename can be renamed by the use of the hash function. The original filename and its extension are hidden inside the image. Identifying the right file requires reading and processing many images *png* as it finds the right one. For the owner of the text files (or for the server on which they are stored), it is possible to create an encrypted list with information about the name under which the file is stored (after converting it to the *png* format). However, if one does not need to protect its data in this way, then the step of naming the image with its hash can be skipped.

Another problem related to data, in particular large data sets, is their processing [40], including sorting [41–43], or processing to get random values from the collected data [44]. The proposed method of data storage allows for further processing, even though they are hidden to the human eye.

3.6. Application in Cryptography

Images obtained as a result of the method can also be encrypted at the stage of creating the *png* file. Before saving the text character to the appropriate pixel, it can be transformed using any encryption algorithm. Without additional encryption of the pixel components, it can be seen that the histograms for the colors red, green and blue are not flat, but resemble the structure of histograms for the natural language in which the original text file was. Using even the simplest encryption methods that flatten histograms can give entirely satisfactory results when trying to read it.

Furthermore, encryption of *png* files can be provided by use of any of the algorithms intended for this purpose, e.g., [34–39].

4. Conclusions

The article describes a method for storing text data in the file stored on the server or in the cloud, which allows simultaneously performing the compression and steganography operations of its content. Described method copes well with most text file formats (including *txt*, *json*, *tex*, *html*,

css). Files stored with *doc* extension can proceed without formatting. Its limitation is the method of coding the original text—extended ASCII or UTF-8 coding is required. In the first case, the output file is compressed, and image content looks like it is randomly generated. In contrast, UTF-8 encoding allows the processing of a much broader range of text files, and in most cases (as demonstrated by the results obtained), it provides compression of the output file, which content looks like randomly generated. The method has been implemented in Python programming language, and the source code has been placed in the Github publicly accessible repository. The method can be used in the area of Big Data, where we deal with a large number of files (usually with text filetype), which content can be compressed (in case of extended ASCII encoding) as well as protected against a simple attempt to read their content. It should be emphasized that the proposed steganographic method extends the classic approach based on a graphic file as a medium in which bits of plain text are introduced at a bit-level of individual RGB components. The article fills a gap in the area of theory and has documented application potential, which will be the subject of further research.

Author Contributions: conceptualization, M.L.; methodology, M.L., A.P. and A.K.; software, M.L. and A.P.; formal analysis, M.L., and A.P.; writing—original draft preparation, M.L. and A.K.; writing—review and editing, M.L. and A.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Appendix A

Address to the Python script, which is the implementation of both versions (extended ASCII and UTF-8 coding) of the presented method can be found on platform Github under the following link: https://github.com/MMI-research-team/mmi/tree/master/T2PT?fbclid=IwAR29u_aEXnlyaW083y6LtZWeZspv_10ENDBhyXBTaUupMj-ba_4XFpEMD1g.

This script has been prepared to be run under the following operating systems: Microsoft Windows, Linux and Apple MacOS. The Python programming language environment (*Python 3*) and the libraries: *numpy* and *Pillow* are required to run this script.

References

1. Özköse, H.; Arı, E.S.; Gencer, C. Yesterday, Today and Tomorrow of Big Data. *Procedia Soc. Behav. Sci.* **2015**, *195*, 1042–1050. [CrossRef]
2. Bello-Orgaz, G.; Jung, J.J.; Camacho, D. Social big data: Recent achievements and new challenges. *Inf. Fusion* **2016**, *28*, 45–59. [CrossRef] [PubMed]
3. Plageras, A.P.; Psannis, K.E.; Stergiou, C.; Wang, H.; Gupta, B. Efficient IoT-based sensor BIG Data collection—Processing and analysis in smart buildings. *Future Gener. Comput. Syst.* **2018**, *82*, 349–357. [CrossRef]
4. Pottier, R.; Menaud, J. TrustyDrive, a Multi-cloud Storage Service That Protects Your Privacy. In Proceedings of the 2016 IEEE 9th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 27 June–2 July 2016; pp. 937–940. [CrossRef]
5. ECB Says One of Its Websites Was Hacked, Data Possibly Captured. 2019. Available online: https://news.bloomberglaw.com/banking-law/ecb-says-one-of-its-websites-was-hacked-data-possibly-captured (accessed on 24 March 2020).
6. Kapczyński, A.; Banasik, A. Biometric logical access control enhanced by use of steganography over secured transmission channel. In Proceedings of the 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS'2011, Prague, Czech Republic, 15–17 September 2011; Volume 2, pp. 696–699. [CrossRef]
7. Lawnik, M. Generalized logistic map and its application in chaos based cryptography. *J. Phys. Conf. Ser.* **2017**, *936*. [CrossRef]

8. Lawnik, M.; Kapczyński, A. Application of modified Chebyshev polynomials in asymmetric cryptography. *Comput. Sci.* **2019**, *20*, 367–381. [CrossRef]
9. Chen, J.K.; Lee, W.Z. An Introduction of NoSQL Databases Based on Their Categories and Application Industries. *Algorithms* **2019**, *12*, 106. [CrossRef]
10. Strohbach, M.; Daubert, J.; Ravkin, H.; Lischka, M. Big Data Storage. In *New Horizons for a Data-Driven Economy: A Roadmap for Usage and Exploitation of Big Data in Europe*; Cavanillas, J.M., Curry, E., Wahlster, W., Eds.; Springer International Publishing: Cham, Switzerland, 2016; pp. 119–141._7. [CrossRef]
11. Almansouri, H.T.; Masmoudi, Y. Hadoop Distributed File System for Big data analysis. In Proceedings of the 2019 4th World Conference on Complex Systems (WCCS), Ouarzazate, Morocco, 22–25 April 2019; pp. 1–5. [CrossRef]
12. Meier, A.; Kaufmann, M. *SQL & NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management*; Springer Vieweg: Berlin, Germany, 2019.
13. Bisong, E. Google BigQuery. In *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*; Apress: Berkeley, CA, USA, 2019; pp. 485–517.
14. Kaur, K.; Sachdeva, M. Performance evaluation of NewSQL databases. In Proceedings of the 2017 International Conference on Inventive Systems and Control (ICISC), Coimbatore, India, 19–20 January 2017; pp. 1–5. [CrossRef]
15. Murazzo, M.; Gómez, P.; Rodríguez, N.; Medel, D. Database NewSQL Performance Evaluation for Big Data in the Public Cloud. In *Cloud Computing and Big Data*; Naiouf, M., Chichizola, F., Rucci, E., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 110–121.
16. Siddiqa, A.; Karim, A.; Gani, A. Big data storage technologies: A survey. *Front. Inf. Technol. & Electron. Eng.* **2017**, *18*, 1040–1070. [CrossRef]
17. Salomon, D. Introduction. In *Data Compression: The Complete Reference*; Springer: Berlin/Heidelberg, Germany, 2000; pp. 1–12._1. [CrossRef]
18. Portable Network Graphics (PNG) Specification (Second Edition), 2003. Available online: https://www.w3.org/TR/2003/REC-PNG-20031110/#F-Relationship (accessed on 24 March 2020).
19. ZIP File Format Specification, 2019. Available online: https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT (accessed on 24 March 2020).
20. Majumder, A.; Changder, S. A Novel Approach for Text Steganography: Generating Text Summary Using Reflection Symmetry. *Procedia Technol.* **2013**, *10*, 112–120. [CrossRef]
21. Hamdan, A.M.; Hamarsheh, A. AH4S: An algorithm of text in text steganography using the structure of omega network. *Secur. Commun. Netw.* **2016**, *9*, 6004–6016. [CrossRef]
22. Lee, C.F.; Chen, H.L. Lossless Text Steganography in Compression Coding. In *Recent Advances in Information Hiding and Applications*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 155–179._8. [CrossRef]
23. Liu, Y.; Wu, J.; Xin, G. Multi-keywords carrier-free text steganography based on part of speech tagging. In Proceedings of the 2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), Guilin, China, 29–31 July 2017; pp. 2102–2107. [CrossRef]
24. Wang, K.; Gao, Q. A Coverless Plain Text Steganography Based on Character Features. *IEEE Access* **2019**, *7*, 95665–95676. [CrossRef]
25. Alsaadi, H.I.; Al-Anni, M.K.; Almuttairi, R.M.; Bayat, O.; Ucan, O.N. Text Steganography in Font Color of MS Excel Sheet. In *DATA '18: Proceedings of the First International Conference on Data Science, E-Learning and Information Systems*; ACM: New York, NY, USA, 2018; pp. 10:1–10:7. [CrossRef]
26. Mandal, K.K.; Singh, P.K. Information Hiding in Text Steganography: A Different Approach. In Proceedings of 2nd International Conference on Advanced Computing and Software Engineering (ICACSE), Sultanpur, India, 8–9 February 2019.
27. Fateh, M.; Rezvani, M. An email-based high capacity text steganography using repeating characters. *Int. J. Comput. Appl.* **2018**, 1–7, doi:10.1080/1206212X.2018.1517713. [CrossRef]
28. Bharti, J.; Solanki, S.; Beliya, A. Comparison of LSB methods and pattern. In Proceedings of the 2017 International Conference on Recent Innovations in Signal processing and Embedded Systems (RISE), Bhopal, India, 27–29 October 2017; pp. 250–256. [CrossRef]
29. Ziv, J.; Lempel, A. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory* **1977**, *23*, 337–343. [CrossRef]

30. Huffman, D.A. A Method for the Construction of Minimum-Redundancy Codes. *Proc. IRE* **1952**, *40*, 1098–1101. [CrossRef]
31. Pillow, 2019. Available online: https://python-pillow.org/ (accessed on 24 March 2020).
32. Dworkin, M. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*; NIST: Gaithersburg, MD, USA, 2015. [CrossRef]
33. Textract, 2019. Available online: https://textract.readthedocs.io/en/stable/ (accessed on 24 March 2020).
34. Kumari, M.; Gupta, S.; Sardana, P. A Survey of Image Encryption Algorithms. *3D Res.* **2017**, *8*, 37. [CrossRef]
35. Uhl, A.; Pommer, A. Image and Video Encryption. In *Image and Video Encryption: From Digital Rights Management to Secured Personal Communication*; Springer US: Boston, MA, USA, 2005; pp. 45–134._5. [CrossRef]
36. Guan, Z.H.; Huang, F.; Guan, W. Chaos-based image encryption algorithm. *Phys. Lett. A* **2005**, *346*, 153–157. [CrossRef]
37. Yavuz, E.; Yazıcı, R.; Kasapbaşı, M.C.; Yamaç, E. A chaos-based image encryption algorithm with simple logical functions. *Comput. Electr. Eng.* **2016**, *54*, 471 –483. [CrossRef]
38. Arab, A.; Rostami, M.J.; Ghavami, B. An image encryption method based on chaos system and AES algorithm. *J. Supercomput.* **2019**, *75*, 6663–6682. [CrossRef]
39. Hua, Z.; Zhou, Y.; Huang, H. Cosine-transform-based chaotic system for image encryption. *Inf. Sci.* **2019**, *480*, 403–419. [CrossRef]
40. Duda, O.; Kochan, V.; Kunanets, N.; Matsiuk, O.; Pasichnyk, V.; Sachenko, A.; Pytlenko, T. Data processing in IoT for smart city systems. In Proceedings of the 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), Metz, France, 18–21 September 2019; Volume 1, pp. 96–99. [CrossRef]
41. Marszałek, Z. Performance tests on merge sort and recursive merge sort for big data processing. *Tech. Sci.* **2018**, *21*, 19–35. [CrossRef]
42. Shatnawi, A.; AlZahouri, Y.; Shehab, M.A.; Jararweh, Y.; Al-Ayyoub, M. Toward a new approach for sorting extremely large data files in the big data era. *Clust. Comput.* **2019**, *22*, 819–828. [CrossRef]
43. Chen, H.; Wan, J.; Li, X. Research and implementation of database high performance sorting algorithm with big data. In Proceedings of the 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA), Beijing, China, 10–12 March 2017, pp. 94–99. [CrossRef]
44. Lawnik, M. Generation of numbers with the distribution close to uniform with the use of chaotic maps. In Proceedings of the 2014 4th International Conference On Simulation And Modeling Methodologies, Technologies And Applications (SIMULTECH), Berlin, Germany, 5–7 September 2014; pp. 451–455. [CrossRef]