Information Systems: Secure Access and Storage in the Age of Cloud Computing

RODRIGUES, Marcos A <http://orcid.org/0000-0002-6083-1303> and SIDDEQ, Mohammed M

Published version

RODRIGUES, Marcos A and SIDDEQ, Mohammed M (2016). Information Systems: Secure Access and Storage in the Age of Cloud Computing. Athens Journal of Sciences, 3 (4), 267-284.

Information Systems: Secure Access and Storage in the Age of Cloud Computing

Marcos A Rodrigues and Mohammed M Siddeq
GMPR-Geometric Modelling and Pattern Recognition Group
Sheffield Hallam University, Sheffield, UK
Email m.rodrigues@shu.ac.uk, mamadmmx76@yahoo.com

ABSTRACT

Given that cloud computing is a remotely accessed service, the connection between provider and customer needs to be adequately protected against all known security risks. In order to ensure this, an open and clear specification of all standards, algorithms and security protocols adopted by the cloud provider is required. In this paper, we review current issues concerned with security threats to cloud computing and present a solution based on our unique patented compression-encryption method. The method provides highly efficient data compression where a unique symmetric key is generated as part of the compression process and is dependent on the characteristics of the data. Without the key, the data cannot be decompressed. We focus on threat prevention by cryptography that, if properly implemented, is virtually impossible to break directly. Our security by design is based on two principles: first, defence in depth, where our proposed design is such that more than one subsystem needs to be violated to get both the data and their key. Second, the principle of least privilege, where the attacker may gain access to only part of a system. The paper highlights the benefits of the solution that include high compression ratios, less bandwidth requirements, faster data transmission and response times, less storage space, and less energy consumption among others.

Keywords
Cloud computing, data compression, encryption, security, privacy.

1. INTRODUCTION

Edward Snowden's revelations were a political fiasco and an economic threat to US based tech companies. The extent of NSA's data collection drove away overseas customers in large numbers over security and privacy concerns creating, at the same time, an opportunity for non-US tech companies. Despite security concerns over data breaches, the Cloud Computing paradigm (Buyya et al. 2009) in which servers, storage and applications are delivered to an organization's computers and devices through the Internet is here to stay. The benefits of this model is that it enables data centres to be accessed and shared as virtual resources in a scalable manner. For businesses, this is a very attractive model as services can expand or shrink as needs change.

Information systems stored in the cloud need to comply with EU data protection and privacy regulations, thus both the stored data and the connection between provider and customer need to be adequately protected against known security risks. Recent reports (Coles 2016) indicate that 82% of cloud providers encrypt data in transit, protecting against man-in-themiddle attacks as data are transmitted. However, only 9.4% of cloud providers encrypt data once stored in the cloud, for file sharing convenience. This is a serious issue leaving the cloud vulnerable to data breaches and unauthorized access. It is important to realise, however, that not all data in the cloud need to be protected by encryption, and not all data should be encrypted in the same way. For instance, images and video may be encrypted by a lossy method while text and other documents need to be lossless. Our algorithms cover both lossless and lossy requirements giving the user full control over what and where it is compressed-encrypted, either at the local machine or in the cloud.

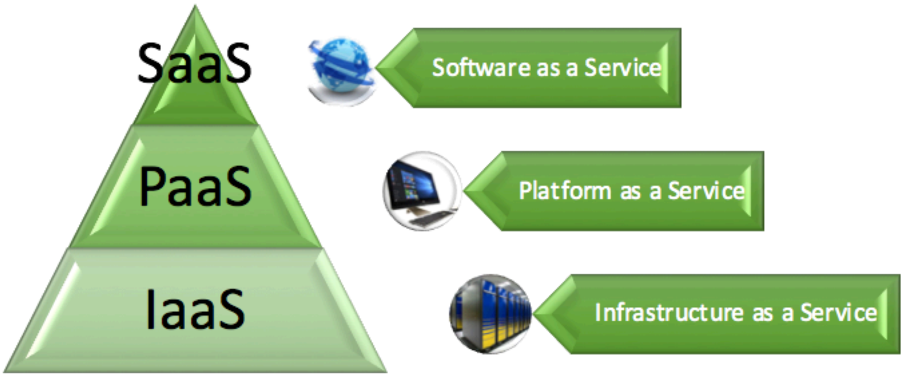
We present an algorithmic solution that has been demonstrated for compression of image and 3D data structures (Siddeq and Rodrigues 2016, 2015a, 2015b, 2014a, 2014b). A unique, data-dependent symmetric key is generated as a side effect to the compression method. Without the key, the data cannot be decompressed. The method allows us to tackle cloud security concerns through high compression ratios, to address data protection and privacy issues, cost of storage, reduced access time, and reduced bandwidth requirements when data in transit are in compressed format.

Section 2 reviews current security threats to Cloud Computing, Section 3 describes the compression-encryption method, and experimental results are reported in Section 4. Finally, a discussion and conclusion is presented in Section 5.

2. SECURITY THREATS TO CLOUD COMPUTING

Cloud Computing services are normally referred to as Infrastructure as a Service (IaaS), Platform as a service (PaaS), and Software as a service (SaaS) (Kepes 2016). IaaS are the hardware and software that powers the cloud such as operating systems, networks, servers, and storage. PaaS are a set of tools and services enabling coding and deployment of applications in a quick and efficient manner. Finally, SaaS are applications delivered over the web that are designed to satisfy end-user needs. These notions are illustrated in Figure 1 and, while blurred boundaries exist, it is assumed that IaaS would provide a secure environment for the other services of the Cloud Computing stack.

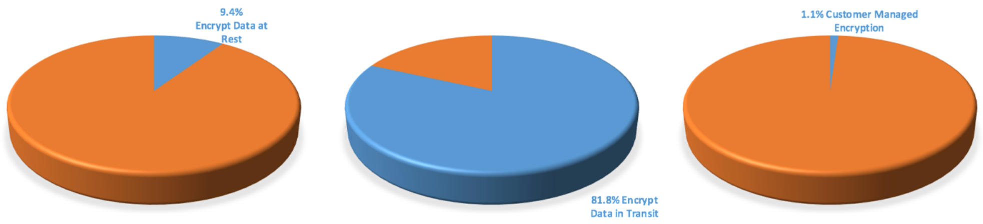
Figure 1: *A diagram depicting the Cloud Computing stack*



On behalf of the Cloud Security Alliance (CSA), the Top Threats Working Group periodically publishes and updates the most relevant perceived threats to Cloud Computing (CSA 2016). Their aim is to provide an informed understanding of cloud security risks and help management making informed decisions. There has been an observed shift in importance in the list of perceived top risks since 2010 from abuse and nefarious use, insecure interfaces and APIs, towards the problem of data breaches. A consistent pattern since 2012 has been observed towards data breaches being identified as the most important issue in each of those years. In 2016 the identified top three were data breaches, access control, and insecure interfaces and APIs (CSA 2016).

Clearly the purely technical issues such as performance, reliability, and availability have fully met or exceeded user's expectations. The focus has shifted from the IT department to the company's board as data and information are normally the most valuable assets a company has. However, it is reasonable to state that current data handling procedures as implemented by cloud providers are not fully meeting expectations. Focusing on data breaches, there are two situations in which a data breach can occur: when data are transferred to and from the cloud, or when data are at rest in the cloud servers. A recent report (Coles 2016) claims that only 9.4% of cloud providers encrypt data at rest; the main statistics in the report are reproduced in Figure 2 below. The report has also established that around 21% of the uploaded data contain sensitive information and that at least 34% of users have uploaded sensitive data to the cloud.

Figure 2: *Encryption practices vary among cloud providers*



Computer security professionals can make use of CASBs (Cloud Access Security Brokers) which are applications acting as a gatekeeper between the user and the cloud, and can enforce security policies beyond the company's structure. CASBs facilitate cloud access management and provide critical information on cloud services across multiple providers. In order to understand and assess the risks to the organization, enterprises need to have clear visibility of which cloud services are in use and by which people, what devices are accessing the data and from where, how sensitive are the data, and whether or not access control through encryption and other enterprise policies are being enforced.

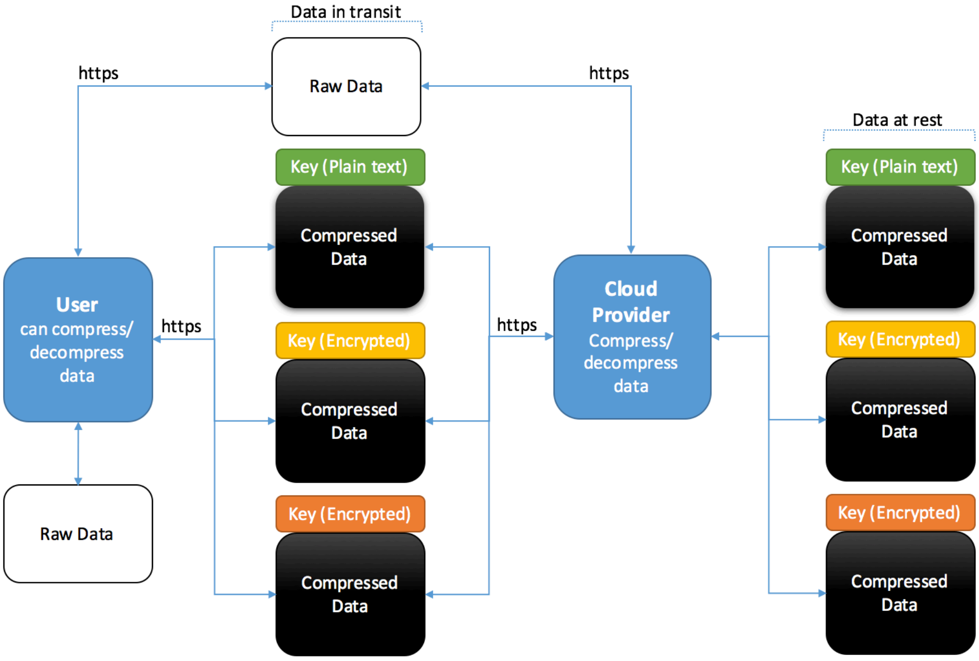
As pointed out above, it is clear that not all data in the cloud need to be encrypted. If that were the case, it would place heavy constraints on data sharing as the encryption key would need to be shared with the data. Solutions to this problem do exist; one of such solutions is proposed in the next sections. The main argument we put forward is for a data compression utility that generates a unique symmetric key as part of the compression step. The key can be kept together with the data in plain text for non-sensitive data. Otherwise, the key would be encrypted by a symmetric algorithm.

The diagram depicted in Figure 3 represents our proposal for the possible compression-encryption scenarios for storing data in the Cloud. The diagram can be used to cover all possibilities by defining whether compression and decompression are performed at the user machine or in the cloud. The possible use cases are as follows.

1) Uploading data to the Cloud:
   a) 🟩 *Non-sensitive data, compression in the cloud*: The Raw Data is transmitted over a secure connection, and compressed in the Cloud. Data are stored in the Cloud in compressed format with compression key in plain text.
   b) 🟩 *Non-sensitive data, compression at the user machine*: The Raw Data are compressed by the user and transmitted to the cloud with the compression key in plain text. Data are stored in the Cloud in compressed format, with compression key in plain text.
   c) 🟨 *Sensitive data, compression in the cloud*: The Raw Data is transmitted over a secure connection and compressed in the cloud. The compression key is encrypted using the symmetric key defined by the owner and known to the Cloud Provider. Data are stored in the Cloud in compressed format together with their encrypted key.
   d) 🟧 *Sensitive data, compression at the user machine*: The Raw Data are compressed by the user and the compression key is encrypted with a key only known to the user. Data are stored in the cloud in compressed format together with their encrypted key. Cloud Provider cannot decrypt the data.

2) Downloading data from the Cloud:
   a) 🟩 *Non-sensitive data, decompression in the cloud*: Decompression is performed in the cloud and the Raw Data are transmitted to the user through a secure connection.
   b) 🟩 *Non-sensitive data, decompression at the user machine*: The compressed data with their key in plain text are transmitted to the user through a secure connection. Decompression is performed at the user local machine.
   c) 🟨 *Sensitive data, decompression in the cloud*: Compression key is decrypted by a symmetric key known to the Cloud Provider and to the owner of the file. Data are decompressed and transmitted as Raw Data over a secure connection.
   d) 🟧 *Sensitive data, decompression at the user machine*: Compressed data and their encrypted key are transmitted to the user. The compression key is decrypted by a symmetric key only known to the user followed by data decompression.

Figure 3: *Compression-encryption scenarios for storing data in the Cloud*



The most secure form of communication and storage are the (d) cases above in which compression and decompression can only be performed by the user as only they are in the possession of the key. However, an enterprise might be comfortable with the cloud provider having access to the decryption keys and the data being decompressed on the fly on the cloud providers' servers. In this situation, options (c) provide adequate level of security. For sharing non-sensitive data, options (a) and (b) are appropriate where the compression key is kept in plain text along with the compressed data. In this situation, data can be decompressed at the user machine or in the cloud and transmitted through a secure connection, either in plain text or compressed-encrypted.

3. THE GMPR COMPRESSION-ENCRYPTION METHODS

There are two main categories of encryption algorithms namely symmetric ciphers (also known as private or symmetric key algorithms) and public key ciphers (public or asymmetric key algorithms). Symmetric ciphers use the same key for encryption and decryption and all the security is in the key, none in the algorithm. Symmetric key algorithms are very fast and the primary problem is in communicating the key securely. Normally it would be easier for an attacker to intercept the key rather than spending resources to crack the message. A secondary issue is that for each pair of users wishing to communicate privately, one separate key is needed, so for n users $n(n-1)/2$ private keys are required.

The strengths of symmetric key algorithms are in the secrecy of the key and on the difficulty of guessing the key by trying out all possible combinations in a brute force attack. There is no way of reversing the encryption without knowing the key, and there are no back doors or alternative ways to decrypt the file without knowing the key. Therefore, the length of symmetric key algorithms is the most important factor to protect against brute force attack. The length of the key is expressed in bits 2^b where b is the number of bits. Currently some of the most popular and used symmetric key algorithms are Triple DES (3x56 bits), Blowfish (up to 448 bits), Twofish (256 bits) and AES (128, 192 and 256 bits).

On the other hand, public ciphers have one private and one public key. Algorithms are based on number theory and mathematical equations with particular properties. The two different keys are used for encryption and decryption: a *public key* is used by anyone wishing to encrypt messages to be sent to a specific user, and a *private key* which is used by the user (receiver) to decrypt the messages. It solves the key-exchange problem of symmetric cryptography as the same public key can be used by anyone, and only the user in possession of the private key is able to decrypt such messages. For n users we need only $2n$ keys. The most common used public key algorithms are the RSA (Rivest, Shamir, and Adleman 1978) and DSA—Digital Signature Algorithm (DSA 2013). Because factorization which is the basis of such algorithms is a very slow process, public key cryptography is only used to exchange symmetric keys and all messages are then encrypted by symmetric algorithms which are orders of magnitude faster than public ciphers.

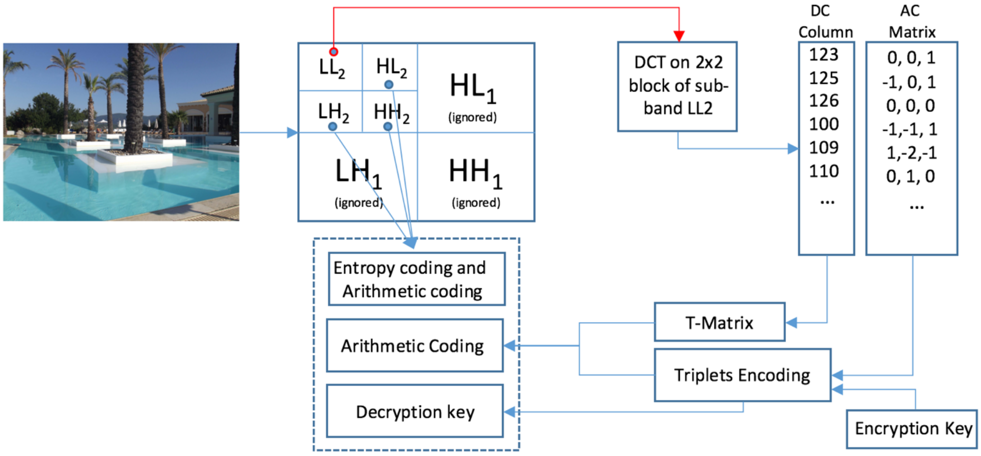
An analysis of 2D image compression and 3D data compression algorithms developed within the GMPR group is presented here from a compression-encryption perspective. We show that the proposed methods provide simultaneous efficient compression-encryption of data for both 2D images and 3D data structures (Siddeq and Rodrigues 2016, 2015a, 2015b, 2014a, 2014b). In Siddeq and Rodrigues (2014b) we proposed a novel 2D image compression method based on high-frequency sub-bands. The complexity of the algorithm and the sequential nature of the solution meant long execution times at decompression stage. New methods using JPEG were proposed in (Siddeq and Rodrigues 2014a) where data decompression was achieved by a number of parallel threads speeding up the process. In (Siddeq and Rodrigues 2015a), further algorithms were developed and tested on frequency sub-bands of DWT followed by DCT. Fast data decompression was achieved through multiple threads.

We stress that the main novel aspect of the GMPR methods concerning security is the automatic generation of a unique symmetric compression-encryption key that is data dependent. The data are divided into blocks and, within each block and after a differential operation, triplets of data are converted into a single value through a weighting factor. This single step reduces the data by 2/3 and, together with the differential process, are the main factors driving the high compression ratios that can be achieved. The array representing the triplets may have repeated values and only one instance of each is kept resulting in a further reduction of the array. This array is the actual compression-encryption key enabling data to be decoded.

3.1 Image Compression-Encryption via DWT and DCT Transforms

There are many possible ways to compress-encrypt 2D images using the GMPR method. The method is characterized by triplet encoding and unique generation of a compression-encryption key. Many transformations before and after these main steps are possible including quantization, entropy coding, and arithmetic coding among others. The example described here uses a double DWT followed by DCT whose parameters are then encoded by the method. The example is of lossy compression, as we apply a DWT over the image and focus on the LL band only, ignoring all high frequency bands. We then apply a second level DWT over the LL band followed by DCT. Figure 4 below illustrates the process.

Figure 4: *An example of the GMPR compression-encryption method applied to a 2D image*



The DWT transform separates a signal into two classes namely approximation and detail coefficients. The signal is decomposed into various frequency bands and scales (Al-Haj 2007; Khashman and Dimililer 2008) by two function sets: scaling and wavelet which are associated with low and high-pass filters. Some of the important properties of the DWT are that many of the coefficients for the high-frequency components (LH_1, HL_1 and HH_1) are zero or insignificant (Grigorios et al. 2008; Sadashivappa and Ananda Babu 2008; Antonini et al. 1992). Most of the important information in the signal is contained in the LL_1 sub-band. In particular, the Daubechies wavelet transform has the ability to reconstruct with high degree of accuracy the original signal through second level sub-bands (LL_2, HL_2, LH_2 and HH_2) while others first level high frequency sub-bands can be ignored leading to high compression ratios (Gonzales and Woods 2001; Acharya and Tsai 2005).

Following a two-level DWT transform, a DCT is applied to each 2x2 block of pixels from the low frequency LL2 sub-band as shown in Figure 4. The transformed coefficients concentrate energy on the low frequency coefficients (top left) which rapidly decreases for higher frequency coefficients at the bottom right of the matrix (Richardson 2002; Rao and Yip 1990). It is safe to discard small value coefficients of the DCT without significantly affecting the quality of the image since they are de-correlated. Compression works more efficiently on a compact matrix of de-correlated coefficients than on a highly correlated matrix (Sayood 2000; Ahmed, Natarajan and Rao 1974).

Without affecting image quality, the high frequency sub-bands in the first level DWT are set to zero (i.e. discard or ignore all HL_1, LH_1 and HH_1). However, the high-frequency sub-bands in the second level DWT (HL_2, LH_2 and HH_2) cannot be discarded without significantly affecting image quality. A quantization $\boldsymbol{Q}$ can be applied at this stage which depends on the maximum value in each sub-band as follows:

$$\boldsymbol{Q} = q\boldsymbol{H}_{max} \tag{1}$$

where the matrix $\boldsymbol{H}$ refers to the high-frequency coefficients in HL_2, LH_2 and HH_2, the factor q refers to the quality affecting the matrix $\boldsymbol{H}$. Thus, image details are reduced in case quality $q \geq 0.01$. The limit range for this factor is specified by the user in a similar way to other image compression methods such as JPEG. The sub-bands HL_2, LH_2 and HH_2 are lossless compressed by arithmetic coding.

3.2 Triplets Encoding, Encryption and Decryption Keys

The purpose of this step is to encode an arbitrary matrix of data with dimension rc where r is the number of rows and c is the number of columns. For computational efficiency at decompression stage, the AC matrix of Figure 4 is divided into blocks where each block is made out of a certain number of rows by exactly 3 columns which are padded with zeros if required. Each block is then encoded and afterwards decoded separately by concurrent threads.

The three columns of each row are encoded into a single value by a generated encryption key K_i which can be generated randomly between $\{0 \ldots 1\}$. Here, the 3-valued key is directly generated from the data which is the preferred solution. Let us assume that the three values to be encoded is the triplet (x, y, z) representing a single row from a block of data. If the data is represented by floating point numbers, it is convenient to convert to integer by multiplying each value by a shift value S; after decompression, the recovered data are divided by the same shift value. Thus,

$$(x, y, z)_{integer} = \mathrm{floor}\big(S(x, y, z)\big) \tag{2}$$

In order to reduce the number of bits needed to represent each triplet a delta or differential process is defined such that only differences are kept after the first values:

$$D_i = D_i - D_{(i+1)} \tag{3}$$

where $i = 1,2,\dots,m-1$ and m is the number of rows of each block. Assuming an integer multiplier factor $F \geq 1$ the parameters required by triplet encoding are defined as follows:

$$M = 1.5\max(x, y, z) \quad (4)$$

$$K_1 = \text{rand}(0,1) \quad (5)$$

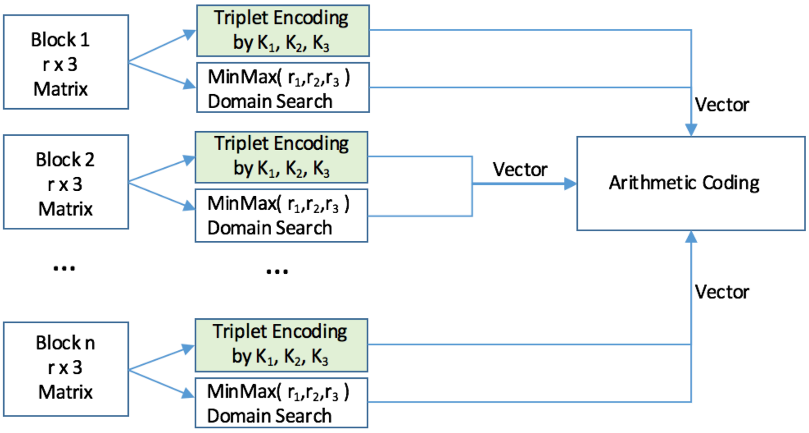
$$K_2 = K_1 + M + F \quad (6)$$

$$K_3 = FM(K_1 + K_2) \quad (7)$$

$$C = K_1 x + K_2 y + K_3 z \quad (8)$$

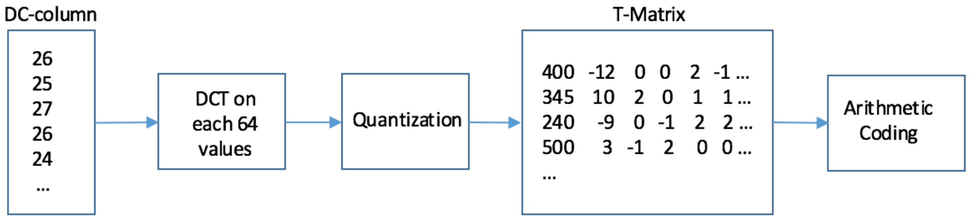
Where C is the coded triplet. Given that the original data are organized into a number of b blocks, each block contains a number of rows r with exactly 3 columns and, to allow reconstruction after compression, we also encode the minimum and maximum values for each of the 3 columns of data for each block as illustrated in Figure 5 below. The MinMax operation extracts both the minimum and maximum values of each column of data for each block.

Figure 5: *Each block of data is uniquely coded by a new set of K_1,K_2,K_3 encryption keys.*



The decryption key as earlier illustrated in Figure 4 is obtained by entropy coding the values of the AC matrix. Such values can also be seen as frequency data and are used at decompression stage: the set of weights are valid if the error is zero and the triplets are all in the domain search or are members of the frequency data (details in Section 3.3 below). The remaining operation is to encode the DC column depicted in Figure 4. It contains the DC values of the DCT partitioned into n-arrays (e.g. $n = 64$). Each of these arrays are transformed by a one-dimensional DCT, quantized and stored in a temporary T-matrix. This matrix contains de-correlated values yielding good compression ratios. Each column of the T-matrix is concatenated into a one-dimensional array which is then coded by Arithmetic coding (Sayood 2000). The coding process of the DC values is illustrated in Figure 6 below.

Figure 6: *Encoding the DC columns from the DCT transforms*



3.3 Decoding the Data

In order to decode the data, a number of reverse steps are necessary. First we reverse the arithmetic coding to recover the T-Matrix of Figure 6. This is followed by inverse quantization and inverse DCT leading to the recovery of of the DC-column data – the inverse path of Figure 6. The AC-matrix has been coded as depicted in Figure 5. Applying inverse arithmetic coding, the sum value C defined in Equation 8 for each triplet is recovered together with minimum and maximum values for each column. The issue here is to recover the generated key values K_1, K_2, and K_3 – note that for each block we have a new set of generated keys. We have developed a number of algorithms and this is currently an active area of research. Here we describe a method based mostly on a simple search. The value of C is constrained by the minimum and maximum values that are known at this stage, and also F is known as it is saved in the file header. Knowledge of the range minimum and maximum allow the development of a number of optimized search algorithms where the goal is zero error:

$$E = C - \sum_{i=1}^{3} S_i K_i \tag{9}$$

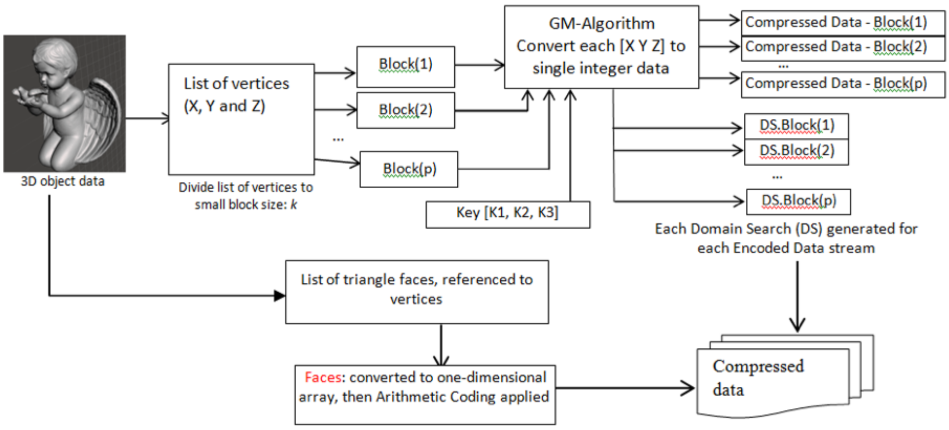
If the error E is zero, then the estimated values S_i correspond to the original values in the AC-Matrix. At this stage then, we have recovered the DC-column and AC-matrix values depicted in Figure 4. An inverse DCT is performed on each block recovering the LL_2 sub-band. The other sub-bands HL_2, LH_2 and HH_2 are recovered by reversing the arithmetic code and the original data are reclaimed.

3.4 Compression-Encryption of 3D Data Structures

The approach to 3D data compression is similar to 2D images in the sense that we look at compressing triplets of data through randomly generated keys and keeping the minimal information that is required to be able to successfully reconstruct the data. This minimal information makes part of the compression-encryption key. We show the steps in the methodology by providing an example of compressing a 3D structure defined in Wavefront's OBJ file format. The OBJ format is very structured with a list of vertices, faces, vertex normal directions, and texture mapping information. In the description that follows, we focus on the compression of a list of vertices; the rest of the file is compressed by the differential process of Equation 3 followed by arithmetic coding.

First it is very convenient to note that a vertex in 3D is defined by its triplet (x, y, z) coordinates, and that our technique is based on converting each triplet into a single value, so it is most appropriate for vertex encoding. Figure 7 depicts the method, starting by breaking the data into a number of blocks. The rationale for doing so is to allow fast decompression by running parallel threads each operating on a single block. Triplet encoding amounts to a geometry minimization process and is indicated in the figure by converting each triplet (x, y, z) into a single integer data. The required variables (M, K_1, K_2, K_3, F) are determined by Equations 4 through 8. The domain search (DS) or frequency data makes part of the decryption key which represents the frequency at which data occurs. It is used in conjunction with the coded data C of equations 8 and 9 to decide whether or not the reconstructed triplet is accepted (at decompression stage).

Figure 7: *Compression-encryption of 3D data*


3D object data
List of vertices (X, Y and Z)
Divide list of vertices to small block size: k
Block(1)
Block(2)
...
Block(p)
GM-Algorithm Convert each [X Y Z] to single integer data
Key [K1, K2, K3]
Compressed Data - Block(1)
Compressed Data - Block(2)
...
Compressed Data - Block(p)
DS.Block(1)
DS.Block(2)
...
DS.Block(p)
Each Domain Search (DS) generated for each Encoded Data stream
List of triangle faces, referenced to vertices
Faces: converted to one-dimensional array, then Arithmetic Coding applied
Compressed data

The vertex texture mapping represented by the (u, v) coordinates and the triangle face indices (V_1, V_2, V_3) are subject to the differential process of Equation 3 applied to each row from left to right and then compressed by arithmetic coding. Data decompression is achieved by reversing the compression method. Each block is decompressed independently by a concurrent thread and a number of search algorithms can be implemented to recover the differential data followed by recovery of vertex data using Equation 9. The normal directions, triangulated face indices, and vertex texture coordinates are recovered by reversing both arithmetic coding and the differential process.

4. EXPERIMENTAL RESULTS

4.1 Compression-Encryption of Images

Here we use four images as examples. We apply the GMPR compression and decompression method and compare with the standard image compression methods JPG and JPEG2000. In all the compared methods one can control the quality of the image which will result in a larger or smaller compressed file. It is therefore, necessary that we compress all images to equivalent sizes such that the perceived quality of the image together with the root mean square errors can be directly compared. Figure 8 depicts the four images used and provides information on original and compressed file sizes, and the achieved compression ratios using the GMPR method. It is noted that compression ratios around 99% are achieved with good perceived quality of the reconstructed image comparable to JPEG2000.

Figure 8: *Images showing their original and compressed sizes*

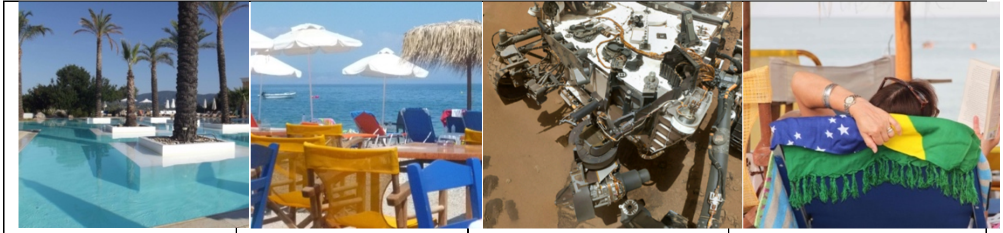


Image1: 39MB to 300KB Compression: 99.2%	Image2: 9MB to 92KB Compression: 98.9%	Image3: 10MB to 120KB Compression: 98.8%	Image4: 19.3MB to 193KB Compression: 99.0%

Table 1 provides a comparison between the GMPR method, JPEG and JPEG2000. We observe that both JPEG2000 and the GMPR method have an equivalent, superior perceptual quality when compared with JPEG for the same high compression ratio. When we consider an objective measure of quality such as 2D RMSE, then the GMPR method is superior to both JPEG and JPEG2000 methods.

Table 1: *2D compression-encryption of sample image files and comparative analysis with JPEG and JPEG2000*

Image	Original size (MB)	GMPR Method		JPEG		JPEG2000	
		Compressed size (MB)	2D RMSE	Compressed size (MB)	2D RMSE	Compressed size (MB)	2D RMSE
Image1	39.4	0.300	2.85	0.300	8.33	0.300	5.32
Image2	9.0	0.092	3.14	0.096	7.39	0.092	6.33
Image3	10.0	0.120	4.68	0.122	11.20	0.120	11.38
Image4	19.3	0.193	2.83	0.197	5.80	0.193	4.45

4.2 Compression-Encryption of 3D Data Structures

We demonstrate 3D compression-encryption methods through two examples, with and without texture mapping. Both models are publicly available from (David 2016) in several file formats. Here we use the OBJ file format as the original file and the purpose is to compress, decompress and evaluate both the perceived quality of the reconstruction and calculate an objective measure of 2D RMSE for texture mapping and 3D RMSE for vertex locations. Two experiments were carried out. The first experiment involved lossy compression, in which the GMPR methods were applied using shift values of 2 and 10. In the second experiment, all floating point vertices were defined as integers and a lossless compression was applied.

Figure 9: *3D models Angel (left) and Face (right)*

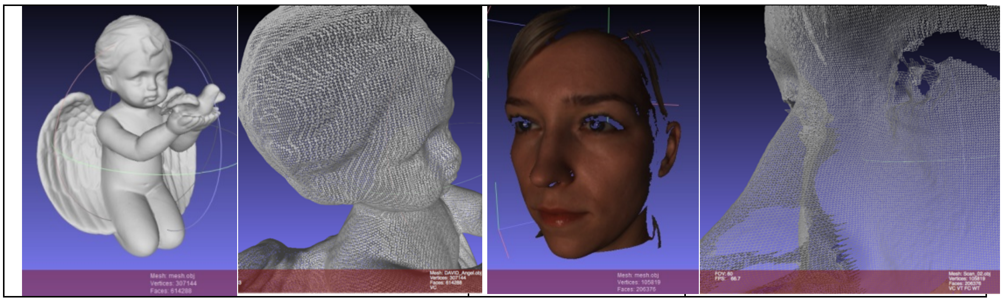


Table 2 illustrates typical achieved compression rates for lossy compression: there are two observed peaks, for a large number of tested files compression rates are around 90% while most of the remainders are around 98%. The quality value used to convert from floating point to integer according to Equation 2 allows the user to control data loss after the decimal point and thus, the overall quality of the mesh in terms of 3D RMSE. If the model is defined in millimetres it may be safe to round off vertex coordinates to the nearest integer as, at this level of detail, humans may not perceive such small differences.

Table 2: *3D lossy compression-encryption of files from OBJ format*

Image or Model	OBJ Original size (MB)	Quality value, S	Compressed size (MB)	Compression Ratio	No. of Vertices	No. of triangles	3D RMSE	2D RMSE
Angel	24.7	10	2.670	89%	307,144	614,287	2.022	N/A
		50	3.090	87%	307,144	614,287	2.023	N/A
Face	14.4	2	0.290	98%	105,819	206,376	1.283	5.7E-4
		10	0.378	97%	105,819	206,376	1.285	5.7E-4

A comparison in terms of compression ratios was made with standard compression algorithms available on Unix/Linux environments, namely *Lempel-Ziv-Welch*, *xz*, *gzip* and *bzip2*. The Lempel-Ziv-Welch (LZW) algorithm is a widely used lossless Unix compression utility based on creating a dictionary for the sequences existing in the data as it is encoded. When the next character is added to the current sequence and it makes a new sequence that it is not in the dictionary, it is added. The algorithm provides fast compression and fast decompression but it is not very efficient in terms of compression ratios.

xz is also a lossless compression algorithm that incorporates the LZMA-Lempel-Ziv-Markov chain algorithm. It shares the same compression format as 7-Zip, a popular compression algorithm on Windows. It provides a relatively very slow compression and fast decompression.

gzip is both a file format and a compression algorithm used in Unix systems. It was developed to replace LZW and it contains a combination of the LZ77 algorithm and Huffman coding. The algorithm allows the concatenation of multiple files although its normal use is for compression of single files. The algorithm provides slow compression and fast decompression.

bzip2 is an open source file compression utility. It is more efficient than LZW and there are parallel implementations with multiple threads but these are not available on the standard version of the algorithm. Similar to gzip, bzip2 is a file compressor and provides no means to compress multiple files. It provides relatively slow compression and fast decompression.

Table 3: *Lossless and lossy compression: comparative analysis with standard Unix compression utilities*

File	Original size (MB)	GMPR Method (MB)	Lempel-Ziv-Welch (MB)	xz (MB)	gzip (MB)	bzip2 (MB)
Angel (floating point)	24.7	**2.670**	7.3	3.1	5.5	5.3
Face (floating point)	14.0	**0.290**	4.7	1.2	3.3	2.6
Average compression ratio		**94% (lossy)**	69%	90%	78%	81%
Angel (integer)	19.1	**3.35**	6.3	2.7	4.6	4.8
Face (integer)	12.0	**0.556**	4.1	0.723	2.7	2.1
Average compression ratio		**89% (lossless)**	66%	90%	77%	79%

Note that the GMPR method operates most efficiently on integers, so normally floating point data are converted to integers by shifting the decimal point to the right and then shifting it back after decompressing. A small shift to the right means that numbers after the decimal point may be truncated and this will result in information loss. For most applications a small shift is acceptable as it would not be possible to discern small decrease in quality when data are visualized.

Results are depicted in Table 3 for both lossy and lossless compression providing a comparative analysis with standard Unix compression utilities. It is noted that the GMPR method compares favourably against all major algorithms in terms of file size without perceived degradation of quality which was verified through careful visual inspection and RMSE measures depicted in Table 2. For lossless compression, the xz algorithm shows a slightly higher compression ratio than the GMPR method.

5. DISCUSSION AND CONCLUSION

In this paper, we review recent security threats to cloud computing and focus on threat prevention through cryptographic methods that, when properly implemented, are virtually impossible to break directly. We pointed out that consistent reports indicate data breaches as the most significant threat as perceived by end users. While most cloud providers encrypt data in transit, data at rest are not encrypted for convenience of data sharing. While it is accepted that most cloud providers implement adequate access control the danger remains that an attacker can gain access to sensitive data stored in raw format.

The best solution from a security and privacy perspective would be that sensitive data be compressed at the user machine and their compression key be encrypted with a key only known to the user before depositing the file in the cloud. The cloud provider would not be able to decompress the data. This is the principle of least privilege. Note that the compressed data would not need to be encrypted, only the compression key. This situation however, can create issues when data need to be shared as users would also have to share their key. Therefore, this is only a solution for non-shareable sensitive data.

In many instances, the user might be comfortable with the cloud provider having access to the key to facilitate data sharing. In this case, we propose that the compression key be encrypted by a symmetric key of the user choice, which is exchanged with the cloud provider through public key infrastructure. The cloud provider then would be able to decrypt the key and decompress the data before allowing access to authorized users over a secure connection. This solution would protect the data against data breaches as, if data are stolen, their compression key is encrypted, so there is no way the actual data can be accessed. The solution however, does not protect the data in the case of a rogue employee having access to users' keys. In any case, the solution provides protection in depth as for an attacker to succeed, first it would need access to the data, then access to the symmetric key to finally enable data decompression.

The GMPR method yields a per-file compression-encryption as the generated key from triplets is entirely data-dependent. All data in the cloud are stored in compressed format, where the compression key is encrypted for sensitive data and kept in plain text for non-sensitive data. Furthermore, the size of the compression-encryption key also depends on the data. To ensure strong encryption, the key can be tested for a minimum of 128 bits, padding if necessary.

Compression of 2D images and 3D structures were reported in the experimental results for both lossy and lossless compression. For 2D data, the method provides compression ratios up to 99% outperforming JPG and being of equivalent perceptual quality of JPEG2000. For 3D data structures, the method yields average compression ratios of 94% for lossy compression and 89% for lossless compression, comparing very favourably to a number of popular data compression utilities available on Unix/Linux environments.

The main advantage of the GMPR methods as presented here are to ensure security and privacy of sensitive data. Given the superior compression ratios of the techniques, a number of further advantages can be listed for deployment in a cloud environment. First, the method requires less storage space than current techniques and this can be even more significant when we consider that cloud providers have to implement file redundancy to guarantee integrity and accessibility of user data. Redundancy imposes hard limits: if the data are to be duplicated, storage space needs to be duplicated there is no alternative way. Second, sensitive data whose compression key is encrypted do not need to be further encrypted for transmission and this can save significant bandwidth obtained from the high compression ratios. This will lead to faster transmission and faster response times. Third, because of less physical space and less bandwidth requirements there will also be corresponding energy savings to be made so it is a green solution to cloud access and storage.

Finally, data protection and privacy legislations are not similar across the globe. It is demonstrated that our solution addresses security and privacy concerns to the highest standards according to current European legislation on data protection whether the servers are located or not in the EU. Future work is focused on implementing a set of Linux utilities as system calls for compression encryption that automatically recognise all types of data (image, 3D formats, video, text, audio, etc.) applying the algorithms accordingly. Also, work is under way on increasing the speed of decompression through high performance computing techniques and will be reported in the near future.

6. REFERENCES

Acharya, T. and Tsai, P.S. 2005. *JPEG2000 Standard for Image Compression: Concepts, Algorithms and VLSI Architecture*. New York: John Wiley & Sons.

Ahmed, N., Natarajan, T. and Rao, K.R. 1974. Discrete cosine transforms*, IEEE Transactions Computer*, Vol. C-23, pp. 90-93.

Al-Haj, A. 2007. Combined DWT-DCT Digital Image Watermarking, *Science Publications, Journal of Computer Science* 3 (9): 740-746.

Antonini, M., Barlaud, M., Mathieu, P. and Daubechies, I. 1992. Image coding using wavelet transform, *IEEE Trans. on Image Processing*, Vol. 1, No. 2, pp. 205–220.

Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J. and Brandic, I. 2009. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility, *Future Generation Computer Systems*, 599—616.

Coles, C. 2016. Only 9.4% of Cloud Providers Are encrypting Data at Rest, Skyhigh Report, [online] https://www.skyhighnetworks.com/cloud-security-blog/only-9-4-of-cloud-providers-are-encrypting-data-at-rest/

CSA 2016. The Treacherous 12 CSA's Cloud Computing Top Threats in 2016. *CSA Top Threats Working Group*. [online] https://cloudsecurityalliance.org/group/top-threats/

David 2016. David 3D Scanner, [online] document available for download from http://www.david-3d.com/en/support/downloads

DSA 2013. *FIPS PUB 186-4: Digital Signature Standard (DSS*), [online] July 2013. csrc.nist.gov.

Gonzalez, R.C. and Woods, R.E. 2001. *Digital Image Processing*, Addison Wesley publishing company.

Grigorios, D., Zervas, N.D., Sklavos, N. and Goutis, C.E. 2008. Design Techniques and Implementation of Low Power High-Throughput Discrete Wavelet Transform Tilters for JPEG 2000 Standard, *WASET , International Journal of Signal Processing,* Vo. 4, No.1.

Kepes, B. 2016. Understanding the Cloud Computing Stack: SaaS, PaaS, IaaS. *Rackspace US Inc*. [online] https://support.rackspace.com/white-paper/understanding-the-cloud-computing-stack-saas-paas-iaas/

Khashman, A., Dimililer, K. 2008. Image Compression using Neural Networks and Haar Wavelet, *WSEAS TRANSACTIONS on SIGNAL PROCESSING*, Vol. 4, No.5.

Rao, K.R. and Yip, P. 1990. *Discrete cosine transform: Algorithms, advantages, applications*, Academic Press, San Diego, CA.

Richardson, I.E.G. 2002. *Video Codec Design*, John Wiley & Sons.

Rivest, R.L., Shamir, A., and Adleman, L. 1978. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, *Communications of the ACM*, Vol 21, No. 2, February 1978, p. 120-26.

Sadashivappa, G. and Ananda Babu K.V.S. 2008. Performance Analysis of Image Coding using Wavelets, *IJCSNS International Journal of Computer Science and Network Security*, VOL. 8 No.10.

Sayood, K. 2000. *Introduction to Data Compression*, 2nd edition, Academic Press, Morgan Kaufman Publishers.

Siddeq, M.M. and Rodrigues, M.A. 2016. Novel 3D Compression Methods for Geometry, Connectivity and Texture, *3DR Express, 3D Research*, June 2016 7:13.

Siddeq, M.M. and Rodrigues, M.A. 2015a. A novel 2D image compression algorithm based on two levels DWT and DCT transforms with enhanced minimize-matrix-size algorithm for high resolution structured light 3D surface reconstruction. *3D Research*, **6** (3), p. 26.

Siddeq, M.M. and Rodrigues, M.A. 2015b. Applied sequential-search algorithm for compression-encryption of high-resolution structured light 3D data. In: BLASHKI, Katherine and XIAO, Yingcai, (eds.) *MCCSIS: Multiconference on Computer Science and Information Systems 2015.* IADIS Press, 195-202.

Siddeq, M.M. and Rodrigues, M.A. 2014a. A new 2D image compression technique for 3D surface reconstruction. *Advances in information sciences and application: Proceedings of 18th International Conference on Computers (part of CSCC'14). Recent advances in computer engineering series*, 1 (22). 379-386.

Siddeq, M.M. and Rodrigues, M.A. 2014b. A novel image compression algorithm for high resolution 3D reconstruction. *3D research*, **5** (7), 17 pages.