METHOD AND SYSTEM TO PREVENT INFORMATION LEAKAGE IN INFORMATION DISPERSAL ALGORITHM

Field of the Invention

This invention relates to a method and system for preventing information leakage in information dispersal algorithm (IDA). Particularly, this invention relates to a method and system of processing data prior to dispersing the data using IDA.

Summary of the Prior Art

IDA uses non-systematic erasure coding to convert message from m bytes to n bytes where n > m. From the view of information-theoretic security, IDAs can provide only incremental confidentiality and thus have weaker confidentiality than secret sharing (with perfect confidentiality) and ramp schemes (with partially perfect confidentiality). However, IDAs can achieve optimal efficiency in data overhead. It has been proven that, as long the generator matrix C is designed properly, even if it is publicly known, any segment of the original file F cannot be reconstructed explicitly from fewer than m pieces. However, there are other types of information leakage such as coarse information leakages and partial information leakages.

For certain files such as picture and voice, there are strong correlation with adjacent data in a file. In this case, IDA may face coarse information leakage. For example, suppose an attacker knew the file is a picture and got about m-1 slices of it with the generator matrix C. He may just create a normalized new slice and do the reverse transformation based on this new slice plus (m-1) known slices. Although the result is not the same as the original picture, it may carry the skeleton or silhouette of the original picture. The reason of this kind of information leakage is that, as the original data is strongly correlated with each other, a human may be able to reconstruct the correlation even there is a gap.

For certain type of file with special data format such as text file, it is easy to distinguish whether (parts of) file data is valid. In this case, IDA will be vulnerable to brute force attack. For example, suppose a hacker knew the file is an email with pure Unicode text in English and he has gotten m-1 slices of the file with the generator matrix. In addition, assume that the block size is 1 byte. We know the common alpha-numeric symbol of Unicode is in the range of U+0020 to U+007E, which is 96 different values in total. However, the overall Unicode space is 65536. This means, if a user randomly guesses 65536 times, he can tell immediately that only the 96 of them are possible input,

and he can rules out the other >99.8% of the trial. This can help the hacker to guess out the text much faster. By guessing the value of one missing byte (256 times), he is able to get 256 different combinations of m bytes of the original file. Ideally, he is not able to know which one of this 256 is the actual one. However, he knows that all the value in the m bytes has to be in the range of U+0020 to U+007E.

Hence, those skilled in the art are constantly striving to come up with an improvement to increase the security of data via the use of IDA.

Summary of the Invention

The above and other problems are solved and an advance in the state of the art is made by systems and methods provided by embodiments in accordance with the invention. A first advantage of embodiments of systems and methods in accordance with the invention is that the systems and methods ensure a high level of security in that all slices of data are required in order to reconstruct the original data. A second advantage of embodiments of systems and methods in accordance with the invention is that systems and methods do not require any encryption key. Since encryption key is not required, this eliminates any risks in sharing of encryption keys over a network.

A first aspect of the invention relates to a system for storing data on a distributed storage network. The system comprises a processor, a memory and instructions store on the memory and executable by the processor to: receive a data; randomise the data; apply an Information Dispersal Algorithm (IDA) to the randomised data to obtain slices of data; and store the slices of data on the distributed storage network.

In accordance with an embodiment of the first aspect of the invention, the instruction to randomise the data comprises instructions to: divide the data into a plurality of blocks, $m_1, \ldots, m_n$, each block being k bit length;generate an Initialisation Vector (IV) comprising random number of k bit length; randomise a first block of the plurality of blocks, m_1, using the IV; randomise the rest of the plurality of blocks, $m_2, \ldots, m_n$, each of the rest of the plurality of blocks is randomised using the randomised data from the previous block; and generate an additional block for storing IV.

In accordance with an embodiment of the first aspect of the invention, the instruction to randomise the first block of the plurality of blocks, m_1, using the IV comprises instruction to: apply an XOR function to a hash of *IV* and the first block m_1, expressed in the following expression, $m_1' = h(IV) \oplus m_1$.

In accordance with an embodiment of the first aspect of the invention, the instruction to randomise the rest of the plurality of blocks, $m_2, \ldots, m_n$, each of the rest of the plurality of blocks is randomised using the randomised data from the previous block comprises instruction to: apply, for current block, an XOR function to hash of previous randomised block, m'_{i-1}, with the current block m_i, expressed in the following expression, $m'_i = h(m'_{i-1}) \oplus m_i$ where $i = 2..n$.

In accordance with an embodiment of the first aspect of the invention, the instruction to generate the additional block for storing IV comprises instruction to: apply an XOR function to a hash of one of the randomised blocks, m'_j, and the *IV*, expressed with the following expression, $m'_0 = h(m'_j) \oplus IV$ where j is selected from $1 \ldots n$.

In accordance with an embodiment of the first aspect of the invention, the system further comprises instructions to: retrieve slices of data; apply the IDA to reconstruct the slices of data to obtain the randomised data; and de-randomise the randomised data to obtain the data.

In accordance with an embodiment of the first aspect of the invention, the instruction to de-randomise the randomised data to obtain the data comprises instructions to: divide the randomised data into a plurality of randomised blocks, $\ldots m'_n$, each randomised block being k bit length; determine the *IV* based on the additional randomised block, m'_0; de-randomise a first block of the plurality of randomised blocks, m'_1, using the *IV*; de-randomise the rest of the plurality of randomised blocks, $m'_2, \ldots, m'_n$, each of the rest of the plurality of randomised blocks is de-randomised using the previous randomised block; and arrange the de-randomised block to form the data.

In accordance with an embodiment of the first aspect of the invention, the instruction to determine the *IV* based on the additional randomised block, m'_0, comprises instruction to: apply an XOR function to a hash of a selected randomised block with the additional randomised block, m'_0., expressed in the following expression, $IV = h(m'_j) \oplus m'_0$ where j is selected from $1 \ldots n$.

In accordance with an embodiment of the first aspect of the invention, the instruction to de-randomise the first block of the plurality of randomised blocks, m'_1, using the *IV* comprises instruction to: apply an XOR function to the hash of *IV* and the first randomised block m'_1, expressed in the following expression, $m_1 = h(IV) \oplus m'_1$.

In accordance with an embodiment of the first aspect of the invention, the instruction to de-randomise the rest of the plurality of randomised blocks, $m'_2, \ldots, m'_n$, each of the rest of the plurality of randomised blocks is de-randomised using the previous randomised block comprises instruction to: apply, for current randomised block, an XOR function to hash of previous randomised block, m'_{i-1}, with the current randomised block m'_i, expressed in the following expression, $m_i = h(m'_{i-1}) \oplus m'_i$ where *i* = *2..n*.

In accordance with an embodiment of the first aspect of the invention, the de-randomised blocks are arranged with the following expression, $F = m_1, \ldots, m_n$.

A second aspect of the invention relates to a method for storing data on a distributed storage network. The method comprises: receiving a data; randomising the data; applying an Information Dispersal Algorithm (IDA) to the randomised data to obtain slices of data; and storing the slices of data on the distributed storage network.

In accordance with an embodiment of the second aspect of the invention, the step of randomising the data comprises: dividing the data into a plurality of blocks, $m_1, \ldots, m_n$, each block being k bit length; generating an Initialisation Vector (IV) comprising random number of k bit length; randomising a first block of the plurality of blocks, m_1, using the IV; randomising the rest of the plurality of blocks, $m_2, \ldots, m_n$, each of the rest of the plurality of blocks is randomised using the randomised data from the previous block; and generating an additional block for storing IV.

In accordance with an embodiment of the second aspect of the invention, the step of randomising the first block of the plurality of blocks, m_1, using the IV comprises: applying an XOR function to a hash of *IV* and the first block m_1, expressed in the following expression, $m'_1 = h(IV) \oplus m_1$.

In accordance with an embodiment of the second aspect of the invention, the step of randomising the rest of the plurality of blocks, $m_2, \ldots, m_n$, comprises: applying, for current block, an XOR function to hash of previous randomised block, m'_{i-1}, with the current block m_i, expressed in the following expression, $m'_i = h(m'_{i-1}) \oplus m_i$ where *i* = *2..n*.

In accordance with an embodiment of the second aspect of the invention, the step of generating the additional block for storing IV comprises: applying an XOR function to a hash of one of the randomised blocks, m'_j, and the *IV*, expressed with the following expression, $m'_0 = h(m'_j) \oplus IV$ where *j* is selected from *1…n*.

In accordance with an embodiment of the second aspect of the invention, the method further comprises: retrieving the slices of data from the distributed storage network; applying the IDA to reconstruct the slices of data to obtain the randomised data; and de-randomising the randomised data to obtain the data.

In accordance with an embodiment of the second aspect of the invention, the step of de-randomising the randomised data to obtain the data comprises: dividing the randomised data into a plurality of randomised blocks, $\ldots m'_n$, each randomised block being k bit length; determining the *IV* based on the additional randomised block, m'_0; de-randomising a first block of the plurality of randomised blocks, m'_1, using the *IV*; de-randomising the rest of the plurality of randomised blocks, $m'_2, \ldots, m'_n$, each of the rest of the plurality of randomised blocks is de-randomised using the previous randomised block; and arranging the de-randomised block to form the data.

In accordance with an embodiment of the second aspect of the invention, the step of determining the *IV* based on the additional randomised block, m'_0, comprises: applying an XOR function to a hash of a selected randomised block with the additional randomised block, m'_0., expressed in the following expression, $IV = h(m'_j) \oplus m'_0$ where *j* is selected from *1…n*.

In accordance with an embodiment of the second aspect of the invention, the step of de-randomising the first block of the plurality of randomised blocks, m'_1, using the *IV* comprises: applying an XOR function to the hash of *IV* and the first randomised block m'_1, expressed in the following expression, $m_1 = h(IV) \oplus m'_1$.

In accordance with an embodiment of the second aspect of the invention, the step de-randomising the rest of the plurality of randomised blocks, $m'_2, \ldots, m'_n$, comprises: applying, for current randomised block, an XOR function to hash of previous randomised block, m'_{i-1}, with the current randomised block m'_i, expressed in the following expression, $m_i = h(m'_{i-1}) \oplus m'_i$ where *i* = *2..n*.

In accordance with an embodiment of the second aspect of the invention, the de-randomised blocks are arranged with the following expression, $F = m_1, \ldots, m_n$.

Brief description of drawings

The above and other features and advantages of a method and a system in accordance with this invention are described in the following detailed description and are shown in the following drawings:

Figure 1 illustrating an overview of the process of dispersing data in accordance with an embodiment of this invention;

Figure 2 illustrating an overview of the process of retrieving data in accordance with an embodiment of this invention;

Figure 3 illustrating a representative processing system in computers, mobile devices, cloud storage servers and any servers that execute instructions to perform the processes in accordance with this invention;

Figure 4.1 illustrating a flow diagram of a process for randomising data upon receiving a data in accordance with an embodiment of this invention;

Figure 4.2 illustrating a flow diagram of a process for slicing and distributing data upon receiving randomised data in accordance with an embodiment of this invention;

Figure 5.1 illustrating a flow diagram of a process for retrieving and reconstructing data to obtain the randomised data in accordance with an embodiment of this invention; and

Figure 5.2 illustrating a flow diagram of a process for de-randomising data upon receiving a randomised data in accordance with an embodiment of this invention.

Detailed Description

This invention relates to a method and system for preventing information leakage in information dispersal algorithm (IDA). Particularly, this invention relates to a method and system of processing data prior to dispersing the data using IDA.

This disclosure relates to pre-processing data by randomizing the data before dispersing in an efficient manner, without requiring any encryption process, and allows restoration of data only if an attacker has in procession all the data of the threshold of slices. Any missing or partial slices will render the data totally useless.

It is proposed a solution which can help to address the above two weaknesses identified in the summary of the prior art. That is to employ an All or Nothing (AONT) module prior to dispersing the data using IDA. The AONT combines a special (hashed) Cipher Block Chaining (CBC) with IDA. The CBC step is not "encryption", since it makes no use of any secret key information. It is merely an invertible pre-processing step with some good properties we are looking for.

Figures 1 and 2 illustrate an overview of a process of pre-processing a data prior to applying an IDA until a recipient reconstruct and re-arrange the slices to obtain the data. Particularly, figure 1 illustrates an overview of the process of dispersing data 101. The data 101 is transmitted to an AONT module 110 where the data would be randomised in a certain process. The randomised data is then transmitted to the IDA module 120 where the randomised data would be divided according to the IDA. The slices are then transmitted to various storages. Figure 2 illustrates an overview of the process of retrieving data 101. Particularly, all the slices 102 pertaining to the data would be retrieved and transmitted to the IDA module 120. The IDA module 120 would combine the slices to form the randomised data and transmit the randomised data to the AONT module 110. In response, the AONT would rearrange the randomised data accordingly and generate the original data. All slices are required in order to reverse the randomised data. If any slices are missing, it will mean certain blocks of data are missing or incomplete, and hence the original data cannot be recovered.

In brief, the AONT module 110 includes a processing having the steps of: 1) randomly generating an Initialization Vector (IV); 2) randomising every block of data by XOR with the hash of the last block; and 3) generating the first block by hashing the last data block and XOR with the IV. Further details of the processes performed by the AONT module 110 would be described below.

Figure 3 illustrates exemplary processing system 300 that represents the processing systems in computers, mobile devices, cloud storage servers and any servers that execute instructions to perform the processes described below in accordance with this invention. One skilled in the art will recognize that the instructions may be stored and/or performed as hardware, firmware, or software without departing from this invention. One skilled in the art will recognize that the exact configuration of each processing system may be different and the exact configuration executing processes in accordance with this invention may vary and processing system 300 shown in figure 3 is provided by way of example only.

Processing system 300 includes Central Processing Unit (CPU) 305. CPU 305 is a processor, microprocessor, or any combination of processors and microprocessors that execute instructions to perform the processes in accordance with the present invention. CPU 305 connects to memory bus 310 and Input/Output (I/O) bus 315. Memory bus 310 connects CPU 305 to memories 320 and 325 to transmit data and instructions between the memories and CPU 305. I/O bus 315 connects CPU 305 to peripheral devices to

transmit and receive data between CPU 305 and the peripheral devices. One skilled in the art will recognize that I/O bus 315 and memory bus 310 may be combined into one bus or subdivided into many other busses and the exact configuration is left to those skilled in the art.

A non-volatile memory 320, such as a Read Only Memory (ROM), is connected to memory bus 310. Non-volatile memory 320 stores instructions and data needed to operate various sub-systems of processing system 300 and to boot the system at start-up. One skilled in the art will recognize that any number of types of memory may be used to perform this function.

A volatile memory 325, such as Random Access Memory (RAM), is also connected to memory bus 310. Volatile memory 325 stores the instructions and data needed by CPU 305 to perform software instructions for processes such as the processes for providing a system in accordance with this invention. One skilled in the art will recognize that any number of types of memory may be used to provide volatile memory and the exact type used is left as a design choice to those skilled in the art.

I/O device 330, keyboard 335, display 340, memory 345, network device 350 and any number of other peripheral devices connect to I/O bus 315 to exchange data with CPU 305 for use in applications being executed by CPU 305. I/O device 330 is any device that transmits and/or receives data from CPU 305. Keyboard 335 is a specific type of I/O that receives user input and transmits the input to CPU 305. Display 340 receives display data from CPU 305 and display images on a screen for a user to see. Memory 345 is a device that transmits and receives data to and from CPU 305 for storing data to a media. Network device 350 connects CPU 305 to a network for transmission of data to and from other processing systems such as the storages that store the slices of data.

The AONT module 110 and IDA module 120 are provided by way of a software application executable by the processor of the computers, mobile devices, cloud storage servers and any servers that execute instructions to perform the processes described below in accordance with this invention.

Storing data on distributed storages

Figure 4.1 illustrates a flow diagram of a process 400 for randomising data upon receiving the data. Process 400 is performed by the AONT module 110 in accordance with an embodiment of this invention. Process 400 begins with step 405 where the data,

F, is divided into fixed size blocks, $m_1, \ldots, m_n$. Each block is *k* bit length. One skilled in the art will recognise that the size of each block is dependent on the total size of the data, parameters of the IDA used in the IDA module 120, or other factors. Hence, the exact size of each block is left as a design choice to those skilled in the art.

In step 410, process 400 generates a random Initialization Vector (*IV*) comprising random number, of *k* bits. *IV* is essentially a matrix of random number with *k* bit length.

In step 415, the first block, m_1, is randomized using the random *IV*. Particularly, the hash of *IV* is XOR with the first block m_1. In other words, process 400 computes the first randomised block, m_1', with the following expression, $m_1' = h(IV) \oplus m_1$.

In step 420, process 400 randomises each of the blocks from the second to last block. Particularly, each block would be randomised using the randomised data from the previous block. More particularly, the current randomised block, m_i', is determined by hash of previous randomised block being XOR with the current block m_i. In other words, process 400 computes the second to last randomised blocks, $m_2', \ldots, m_n'$, with the following expression, $m_i' = h(m_{i-1}') \oplus m_i$ where *i* = *2..n*.

In step 425, process 400 generates an additional randomised block, m_0', to store the *IV*. Particularly, process 400 generates the additional randomised block with the following expression, $m_0' = h(m_j') \oplus IV$ where *j* is a number selected from *1* to *n*. For purposes of this description, *j=n* is selected. The choice of *j* is pre-determined as the selected randomised block is required to recover *IV* during reconstruction of the data. Further, while it is described that the additional randomised block is arranged before the first block, one skilled in the art will recognise that the additional randomised block may also be arranged as the last block, m_{n+1}', without departing from the invention.

In step 430, process 400 arrange the randomized blocks starting with the additional block, m_0', followed by the rest of the randomised blocks in ascending order to form a randomised data. Particular, the arrangement of the randomised data can be shown in the following expression, $F' = m_0', m_1', m_2', \ldots, m_n'$. The randomised data, F', is then transmitted to the IDA module 120.

Process 400 ends after step 430.

Figure 4.2 illustrates a flow diagram of a process 450 for distributing the randomised data to different storages. Process 450 is performed by the IDA module 120 in accordance with an embodiment of this invention. Process 450 relates to the process

where the IDA module 120 in response to receiving the randomised data, F', would apply IDA to obtain and distribute the slices data in the following manner. Particularly, process 450 begins with step 455 where randomised data, F' would be split into n slices as follows:$F' = (b_1, b_2, b_3, \dots, b_m), (b_{m+1}, b_{m+2}, b_{m+3}, \dots, b_{2m}) \dots (b_{n-m+1}, \dots, b_n)$

In step 460, process 450 applies a transform matrix to the n slices as follows.

Let A be the transform matrix $\begin{bmatrix} a_{11} & a_{12} & \cdot\cdot & a_{1m} \\ a_{21} & a_{22} & \cdot\cdot & a_{2m} \\ \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot \\ a_{n1} & a_{n2} & \cdot\cdot & a_{nm} \end{bmatrix}$and B is the input matrix of F and the C is the output matrix (output slices). Thus, we can write the following matrix equation:

$$\begin{bmatrix} a_{11} & a_{12} & \cdot\cdot & a_{1m} \\ a_{21} & a_{22} & \cdot\cdot & a_{2m} \\ \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot \\ a_{n1} & a_{n2} & \cdot\cdot & a_{nm} \end{bmatrix} . \begin{bmatrix} b_1 & b_{m+1} & \cdot\cdot & b_{N-m+1} \\ b_2 & b_{m+2} & \cdot\cdot & b_{N-m+2} \\ \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot \\ b_m & b_{2m} & \cdot\cdot & b_n \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \cdot\cdot & c_{1N/m} \\ c_{21} & c_{22} & \cdot\cdot & c_{2N/m} \\ \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot \\ c_{n1} & c_{n2} & \cdot\cdot & c_{nN/m} \end{bmatrix}$$

In step 465, the output slices are distributed to the various storages accordingly. Particularly, each row of the output matrix C is the slice of data which will be stored and secured separately on different storages. Process 450 ends after step 465.

The IDA is a widely known algorithm for dividing a data to small pieces (known as slices) and scattering the slices to different storages. For brevity, a detailed description of the IDA is omitted. Importantly, this disclosure is directed to pre-processing of a data prior to the applying an IDA to distribute the slices to various storages, and the post-processing of data after applying an IDA to retrieve the original data.

<u>Retrieving data from distributed storages</u>

In response to receiving a request for retrieving data, the system would retrieve all the slices of data and transmit to the IDA module. The IDA module would reconstruct the slices to form the randomised data in the following manner using *m* slices (where *m* < *k*). Figure 5.1 illustrates a flow diagram of a process 550 for retrieving all the slices of data and reconstructing the data to form the randomised data. Process 550 is performed by the IDA module 120 in accordance with an embodiment of this invention. Process 550 begins with step 555 where all n slices of data are retrieved from the various storages.

In step 560, process 550 reconstructs the slices to form the randomised data in the following manner.

Let A^{-1} be the inverse matrix of A', matrix $\begin{bmatrix} a_{11} & a_{12} & \cdot\cdot & a_{1m} \\ a_{21} & a_{22} & \cdot\cdot & a_{2m} \\ \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot \\ a_{n1} & a_{n2} & \cdot\cdot & a_{nm} \end{bmatrix}$, C is the input matrix (slices retrieved), and B is the output matrix of F. Thus, we can write the following matrix equation:

$$\begin{bmatrix} a_{11} & a_{12} & \cdot\cdot & a_{1m} \\ a_{21} & a_{22} & \cdot\cdot & a_{2m} \\ \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot \\ a_{m1} & a_{m2} & \cdot\cdot & a_{mm} \end{bmatrix} . \begin{bmatrix} c_{11} & c_{12} & \cdot\cdot & c_{1N/m} \\ c_{21} & c_{22} & \cdot\cdot & c_{2N/m} \\ \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot \\ c_{m1} & c_{n2} & \cdot\cdot & c_{mN/m} \end{bmatrix} = \begin{bmatrix} b_1 & b_{m+1} & \cdot\cdot & b_{N-m+1} \\ b_2 & b_{m+2} & \cdot\cdot & b_{N-m+2} \\ \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot \\ b_m & b_{2m} & \cdot\cdot & b_N \end{bmatrix}$$

In step 565, process 550 arranges the output matrix to form the randomised data in the following manner.

$F' = (b_1, b_2, b_3, \dots, b_m), (b_{m+1}, b_{m+2}, b_{m+3}, \dots, b_{2m}) \dots (b_{n-m+1}, \dots, b_n)$ where $m'_0 = (b_1, b_2, b_3, \dots, b_m)$, $m'_1 = (b_{m+1}, b_{m+2}, b_{m+3}, \dots, b_{2m})$ and $m'_n = (b_{n-m+1}, \dots, b_n)$.

In other words, $F' = m'_0, m'_1, m'_2, \dots, m'_n$.

Process 550 ends after step 565.

With the AONT module, all slices are required in order to reconstruct the slices to obtain the randomised data. Hence, *m* cannot be less than *k* while applying the IDA since all it takes is a missing slice to render the data retrieved as garbage after applying the AONT module.

The randomised data, F', is then transmitted to the AONT module to de-randomise the randomised data. Figure 5.2 illustrates a flow diagram of a process 500 performed by the AONT module 110 for re-randomising data upon receiving the randomised data, F', from the IDA module in accordance with an embodiment of this invention. Process 500 begins with step 505 by receiving randomised data, F', and dividing the randomised data into the fixed block size, $m'_0 \dots m'_n$, where each randomised block is *k* bit length.

In step 510, process 500 determines the *IV* based on the additional randomised block, m'_0, and the selected randomised block m'_j where *j*=*n*. Particularly, the hash of the selected randomised block is XOR with the additional randomised block, m'_0. In other words, process 500 computes *IV* with the following expression, $IV = h(m'_j) \oplus m'_0$.

In step 515, process 500 determines the first block, m_1, based on the *IV* and the first randomised block, m'_1. Particularly, the hash of the *IV* is XOR with the first

randomised block, m_1'. In other words, process 500 computes the first block with the following expression, $m_1 = h(IV) \oplus m_1'$.

In step 520, process 500 determines each of the blocks from the second to last randomised blocks. Particularly, each randomised block would be de-randomised using the randomised data from the previous block. More particularly, current block, m_i, is determined by the hash of the previous randomised block, m_{i-1}', being XOR with the randomised current block, m_i'. In other words, process 400 computes the second to last blocks, $m_2, \ldots, m_n$, with the following expression, $m_i = h(m_{i-1}') \oplus m_i'$ where *i* = *2..n*.

In step 525, process 500 arranges the blocks to obtain the data. Particularly, the data is arranged with the following expression, $F = m_1, \ldots, m_n$. Essentially, the additional block is removed to form the original data.

Process 500 ends after step 525.

Essentially, all blocks, $m_0, \ldots, m_n$, are required in order to reverse the randomised data. If any slices are missing, it will mean certain blocks of data are missing or incomplete, and hence the original data cannot be recovered.

In any good hash function, like SHA2, the hash value generated has the property of uniformity. It means every hash value in the output range should be generated with roughly the same probability. At the same time, the hash value has almost zero correlation with the original value. Hence, it is a good way to generate a "random" value and also retrievable in decoding process. By using an XOR operator on this "random" number with the actual data block, it helps to randomize the data itself, and hence 1) break any correlation of adjacent data; and 2) break any existing data format if there is any. As a result, it prevents the problems discussed above.

The system and method described above can be implemented as part of the secure cloud storage solutions. This can be incorporated as part of the public cloud storage solutions by cloud consumer (enterprises), cloud broker, or application developer of cloud security solution/application.

The above is a description of embodiments of a system and method of pre-processing data prior to applying IDA in accordance with the present disclosure. It is foreseeable that those skilled in the art can and will design alternative system and method based on this disclosure that infringe upon this disclosure.

Claims

1. A system for storing data on a distributed storage network comprising:

a processor, a memory and instructions store on the memory and executable by the processor to:

receive a data;

randomise the data;

apply an Information Dispersal Algorithm (IDA) to the randomised data to obtain slices of data; and

store the slices of data on the distributed storage network, wherein the instruction to randomise the data comprises instructions to:

divide the data into a plurality of blocks, $m_1, \ldots, m_n$, each block being k bit length;

generate an Initialisation Vector (IV) comprising random number of k bit length;

randomise a first block of the plurality of blocks, m_1, using the IV;

randomise the rest of the plurality of blocks, $m_2, \ldots, m_n$, each of the rest of the plurality of blocks is randomised using the randomised data from the previous block; and

generate an additional block for storing IV.

2. The system according to claim 1 wherein the instruction to randomise the first block of the plurality of blocks, m_1, using the IV comprises instruction to:

apply an XOR function to a hash of *IV* and the first block m_1, expressed in the following expression, $m_1' = h(IV) \oplus m_1$.

3. The system according to anyone of claims 1-2 wherein the instruction to randomise the rest of the plurality of blocks, $m_2, \ldots, m_n$, each of the rest of the plurality of blocks is randomised using the randomised data from the previous block comprises instruction to:

apply, for current block, an XOR function to hash of previous randomised block, m_{i-1}', with the current block m_i, expressed in the following expression, $m_i' = h(m_{i-1}') \oplus m_i$ where $i = 2..n$.

4. The system according to anyone of claims 1-3 wherein the instruction to generate the additional block for storing IV comprises instruction to:

apply an XOR function to a hash of one of the randomised blocks, m'_j, and the *IV*, expressed with the following expression, $m'_0 = h(m'_j) \oplus IV$ where *j* is selected from *1…n*.

5. The system according to anyone of claims 1-4 further comprising instructions to:

retrieve slices of data;

apply the IDA to reconstruct the slices of data to obtain the randomised data; and

de-randomise the randomised data to obtain the data.

6. The system according to claim 5 wherein the instruction to de-randomise the randomised data to obtain the data comprises instructions to:

divide the randomised data into a plurality of randomised blocks, $\ldots m'_n$, each randomised block being k bit length;

determine the *IV* based on the additional randomised block, m'_0;

de-randomise a first block of the plurality of randomised blocks, m'_1, using the *IV*;

de-randomise the rest of the plurality of randomised blocks, $m'_2, \ldots, m'_n$, each of the rest of the plurality of randomised blocks is de-randomised using the previous randomised block; and

arrange the de-randomised block to form the data.

7. The system according to claim 6 wherein the instruction to determine the *IV* based on the additional randomised block, m'_0, comprises instruction to:

apply an XOR function to a hash of a selected randomised block with the additional randomised block, m'_0., expressed in the following expression, $IV = h(m'_j) \oplus m'_0$ where *j* is selected from *1…n*.

8. The system according to anyone of claims 6-7 wherein the instruction to de-randomise the first block of the plurality of randomised blocks, m'_1, using the *IV* comprises instruction to:

apply an XOR function to the hash of *IV* and the first randomised block m'_1, expressed in the following expression, $m_1 = h(IV) \oplus m'_1$.

9. The system according to anyone of claims 6-8 wherein the instruction to de-randomise the rest of the plurality of randomised blocks, $m'_2, \ldots, m'_n$, each of the rest of the plurality of randomised blocks is de-randomised using the previous randomised block comprises instruction to:

apply, for current randomised block, an XOR function to hash of previous randomised block, m'_{i-1}, with the current randomised block m'_i, expressed in the following expression, $m_i = h(m'_{i-1}) \oplus m'_i$ where *i = 2..n*.

10. The system according to anyone of claims 6-9 wherein the de-randomised blocks are arranged with the following expression, $F = m_1, \ldots, m_n$.

11. A method for storing data on a distributed storage network comprising:

receiving a data;

randomising the data;

applying an Information Dispersal Algorithm (IDA) to the randomised data to obtain slices of data; and

storing the slices of data on the distributed storage network, wherein the step of randomising the data comprises:

dividing the data into a plurality of blocks, $m_1, \ldots, m_n$, each block being k bit length;

generating an Initialisation Vector (IV) comprising random number of k bit length;

randomising a first block of the plurality of blocks, m_1, using the IV;

randomising the rest of the plurality of blocks, $m_2, \ldots, m_n$, each of the rest of the plurality of blocks is randomised using the randomised data from the previous block; and

generating an additional block for storing IV.

12. The method according to claim 11 wherein the step of randomising the first block of the plurality of blocks, m_1, using the IV comprises:

applying an XOR function to a hash of *IV* and the first block m_1, expressed in the following expression, $m'_1 = h(IV) \oplus m_1$.

13. The method according to anyone of claims 11-12 wherein the step of randomising the rest of the plurality of blocks, $m_2, \ldots, m_n$, comprises:

applying, for current block, an XOR function to hash of previous randomised block, m'_{i-1}, with the current block m_i, expressed in the following expression, $m'_i = h(m'_{i-1}) \oplus m_i$ where *i = 2..n*.

14. The method according to anyone of claims 11-13 wherein the step of generating the additional block for storing IV comprises:

applying an XOR function to a hash of one of the randomised blocks, m'_j, and the *IV*, expressed with the following expression, $m'_0 = h(m'_j) \oplus IV$ where *j* is selected from *1…n*.

15. The method according to anyone of claims 11-14 further comprising:

retrieving the slices of data from the distributed storage network;

applying the IDA to reconstruct the slices of data to obtain the randomised data; and

de-randomising the randomised data to obtain the data.

16. The method according to claim 15 wherein the step of de-randomising the randomised data to obtain the data comprises:

dividing the randomised data into a plurality of randomised blocks, $\ldots m'_n$, each randomised block being k bit length;

determining the *IV* based on the additional randomised block, m'_0;

de-randomising a first block of the plurality of randomised blocks, m'_1, using the *IV*;

de-randomising the rest of the plurality of randomised blocks, $m'_2, \ldots, m'_n$, each of the rest of the plurality of randomised blocks is de-randomised using the previous randomised block; and

arranging the de-randomised block to form the data.

17. The method according to claim 16 wherein the step of determining the *IV* based on the additional randomised block, m'_0, comprises:

applying an XOR function to a hash of a selected randomised block with the additional randomised block, m'_0., expressed in the following expression, $IV = h(m'_j) \oplus m'_0$ where *j* is selected from *1…n*.

18. The method according to anyone of claims 16-17 wherein the step of de-randomising the first block of the plurality of randomised blocks, m'_1, using the *IV* comprises:

applying an XOR function to the hash of *IV* and the first randomised block m'_1, expressed in the following expression, $m_1 = h(IV) \oplus m'_1$.

19. The method according to anyone of claims 16-18 wherein the step de-randomising the rest of the plurality of randomised blocks, $m'_2, \ldots, m'_n$, comprises:

applying, for current randomised block, an XOR function to hash of previous randomised block, m'_{i-1}, with the current randomised block m'_i, expressed in the following expression, $m_i = h(m'_{i-1}) \oplus m'_i$ where *i = 2..n*.

20. The method according to anyone of claims 16-19 wherein the de-randomised blocks are arranged with the following expression, $F = m_1, \ldots, m_n$.