CloudS: A Multi-cloud Storage System with Multi-level Security

Lu Shen, Shifang Feng, Jinjin Sun, Zhongwei Li, Gang Wang(✉), and Xiaoguang Liu(✉)

Nankai-Baidu Joint Lab, College of Computer and Control Engineering and College of Software, Nankai University, Tianjin 300350, China
{shenlu,fengsf,sunjj,lizhongwei,wgzwp,liuxg}@nbjl.nankai.edu.cn

Abstract. With the increase of data quantity, people have begun to attach importance to cloud storage, however, traditional single cloud can't ensure the privacy of users' data to a certain extent. To solve the security issue, we present a multi-cloud storage system called CloudS which spreads data over multiple cloud storage servers by using a new kind of XOR-based non-systematic erasure codes - Privacy Protecting Codes (PPC). For better user experiences and tradeoffs between security and performance, CloudS provides multiple levels of security by a variety of combinations of compression, encryption and coding schemes. In addition, we also put forward a novel Parallel Cyclic Encryption (PCE) scheme to achieve random secret key protection which attains high security and performance. We implement CloudS as a web application which doesn't require users to perform complicated operations on local.

Keywords: Multi-cloud · Multi-level security · Erasure code · Encryption · Key management

1 Introduction

Cloud storage has become the focus of public attention in recent years. Due to the ever-growing amount of data and local storage limits, more and more individual and enterprise users have begun to put data in cloud servers. According to iiMedia Research Group's report [2], only in China, the amount of individual cloud storage users will reach 450 million in 2015. With the popularity of cloud storage, many Internet companies have launched their own cloud services such as Microsoft OneDrive, Amazon S3 and Google Drive. However, cloud storage security issues are gradually exposed. In 2014, photo leakage occurs to icloud which is another security incident after Google Docs and Amazon S3 [7,8]. We summarize potential security hazards of cloud storage as the damage to data integrity and privacy. For data integrity, network transmission errors, hacker attacks and faulty operations from server administrators will cause data tampering and loss. For data privacy, both untrusted cloud providers and hacker attacks will leak out users' data. Besides, the availability of cloud services and vendor lock-in should be worth attention.

© Springer International Publishing Switzerland 2015
G. Wang et al. (Eds.): ICA3PP 2015, Part III, LNCS 9530, pp. 703–716, 2015.
DOI: 10.1007/978-3-319-27137-8_51

To solve the above problems, many multi-cloud systems whose main idea is data dispersal to multiple cloud servers have appeared such as DepSky [6], RACS [4], HAIL [7]. Existing multi-cloud systems typically use information dispersal algorithms to spread data across multiple storage sites. There are two kinds of common technologies: duplication and erasure codes. Compared by duplication, erasure codes have smaller storage overhead to achieve the same erasure-correcting. Hence, most multi-cloud systems use erasure codes like RAID-5 [13] and RS codes [15,18]. Traditional erasure codes are generally systematic codes which have no confusion and diffusion effects, therefore, they're usually used combined with encryption to reach high-level security, which brings out key management issues. Although non-systematic multi-erasure RS is proposed, it requires relatively complex finite field arithmetic and multi-erasure seems unnecessarily expensive for only a few cloud services. So in this paper, we put forward a new kind of XOR-based non-systematic erasure codes called PPC (Privacy Protecting Code). Although PPC provides relatively weaker protection on users' data compared with modern encryption algorithms, it has pretty high performance and can be combined with other mechanisms to provide multi-level security.

So-called multi-level security, that is, users can choose different security levels for their different data according to their needs. With various types of data uploaded to cloud servers, users' security demands for cloud systems have gradually diversified. Besides, processing a less-confidential file with an overly complex security mechanism will lead to server resource waste and bad user experiences, especially for those devices with a relatively weak computing capability. Therefore, multi-level security is needed to achieve trade-offs between security and performance. Unfortunately, existing multi-cloud systems don't involve this trade-off. So in this paper, we design and implement a multi-cloud system called CloudS which employs different combinations of compression, encryption and coding to achieve multi-level security. As a third party agent, CloudS is allowed to temporarily access user data only when it has got tokens. For better performance and user experiences, we adopt a security authorization mechanism proposed in [22]. In addition, we implement CloudS as a web application, therefore not requiring any complicated operation even installing a software on local, reducing troubles due to changing a computer.

Our Contribution:

- Design and implement a family of XOR-based non-systematic coding schemes called Privacy Protecting Code (PPC).
- Design and implement a variety of combinations of compression, encryption and coding schemes which provides multiple levels of trade-offs between security and performance.
- Implement the secure web-based multi-cloud storage system CloudS with the integration of the above methods.

The rest of this paper is organized as follows. In Sect. 2 we will discuss relevant work about multi-cloud systems and related techniques. Section 3 will introduce the design of CloudS. Section 4 describes PPC codes in detail. Section 5 focuses on the security levels. In Sect. 6 we evaluate the performance of PPC, different security combination schemes and CloudS system. And Sect. 7 concludes this paper and the future work is proposed.

2 Related Work

With the prevalence of cloud storage services and broadband Internet access, data confidentiality, integrity and availability problems in cloud storage raise concern gradually. To address these problems, many multi-cloud storage systems have been proposed to offer integrity and high availability using information dispersal, duplication and erasure codes like MDS [20]. For data confidentiality, Shamir's secret sharing [21] and Rabin's information dispersal [17] are widely used.

In the multi-cloud environment, many well-performance systems appear as well. Depsky, a virtual storage cloud system, addresses availability and confidentiality of data [6]. Similarly, RACS is a proxy that transparently spreads the storage load over many providers with RAID-like techniques to provide high-availability of data stored in the clouds and avoid the costs of vendor lock-in [4]. Aimed at guaranteeing data integrity and availability, HAIL is a distributed cryptographic multi-cloud system [7] which combines proof of retrievability (PORs) [11] and proofs of data possession (PDPs) [5]. ICStore [3] addresses CIRC (confidentiality, integrity, reliability and consistency) by using replication and Shamir's secret sharing. Nevertheless, the above systems don't consider about various security requirements for different data and/or different users. This paper fills that gap by presenting the systematic description of a multi-level security system called CloudS. Meanwhile, we come up with a new XOR-based non-systematic erasure code called PPC Code for data confidentiality and integrity with low storage and computational cost and apply it to CloudS. Compared with systematic erasure codes, PPC is trustworthier in cloud environments that we can still ensure data security even if one cloud server have been attacked.

3 The Framework Design

This section presents the basic model of our system and the architecture of CloudS.

3.1 System Model

Figure 1 shows our system model and there are 3 parts which are users, CloudS agent and public cloud servers provided by different cloud vendors. A user can be abstracted as a single web browser because no codes are executed locally outside the web browser. From another perspective, it doesn't require users to install any plug-ins locally, therefore avoiding inconvenience due to the failure or replacement of local computers. In this model, CloudS works as a third party proxy between users and cloud providers.

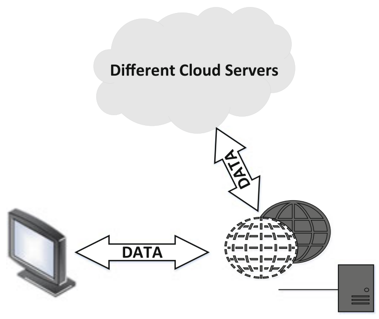


Fig. 1. System model

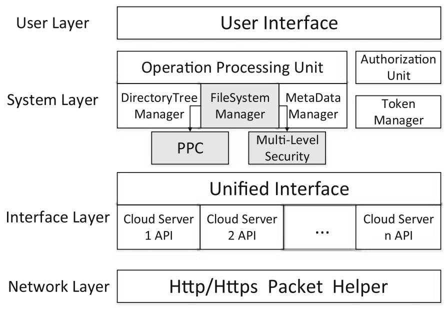


Fig. 2. CloudS architecture

3.2 CloudS Architecture

As presented in Fig. 2, CloudS architecture consists of four layers which are the user layer, the system layer, the interface layer and the network layer.

User Layer: It is used to accept users' requests and submit these requests to the system layer. We can regard the user layer as the "door" of interaction between users and CloudS.

Interface Layer: A multi-cloud system involves different cloud providers, and each provider has its own API. For good scalability, we've designed the interface layer to implement a unified interface.

Network Layer: Data transmission between CloudS and cloud servers is achieved by sending and receiving http/https packet, and the network layer is responsible for the work.

System Layer: The core of CloudS is the system layer. It can be regarded as the "brain" of CloudS and undertakes all computing tasks. It's mainly composed of the operation processing unit, the authorization unit and the token manager unit. As a third party proxy, CloudS can temporarily access to user's data stored in cloud storage only if it has got the user's authorization and holds a string called token. Hence, how to get user's authorization and manage the token is crucial. The authorization unit and the token manager unit are designed for these jobs. The operation processing unit is a data manipulation module. The directory tree manager is designed for directory operation and metadata record directory information.

The filesystem manager is designed to process user data and ensures data security through a series of mechanisms. As mentioned before, security is a vital factor in storage systems but not the only one, so we must consider the tradeoff between security and performance. Here, we assume two different scenes: the user regards cloud services as normal file backup, the other is uploading confidential papers to cloud servers. For the former, users may not require high security and be more likely to pursue the performance of file transfer. For the latter, it's more likely that users pursue high security. Therefore, CloudS provides *different combinations of encryption, compression and coding schemes to satisfy different* *security levels.* With this mechanism, users can make their choices according to their needs instead of being restricted by system setting, which makes the system more flexible. This mechanism takes full account of user experiences and achieves the trade-offs between performance and security. The filesystem manager takes the security level selected by the user as an input parameter. With this parameter, the filesystem manager adopts the corresponding combination scheme. In addition, *PPC is implmented in this module as the coding scheme.*

4 PPC Design and Implementation

As presented, we propose PPC to meet the tradeoffs between security and performance. In this section, we explain PPC in detail. For convenience, we employ D_i to present the $(i+1)$-th original chunk and P_i for the $(i+1)$-th coding chunk.

4.1 Nomenclature

Before introducing PPC, we first define the terms used in this section.

Chunk [9]: A basic unit of storage holding data and/or coding ('parity') information. This is also referred to as a stripe unit or a strip [10] in traditional storage systems.

Stripe: In our cloud storage model, the user file is split into k chunks and then are encoded into a stripe composed of $n(= k + m)$ chunks. These chunks are spread across n nodes to provide high reliability and availability. From the theoretical view, a stripe is a codeword which is the minimum (and complete) collection of (data and parity) bits that encode and decode together [16].

Node: An independent storage container that stores data and parity chunks. This can be a cloud storage service, a node in peer-to-peer system, and so on.

4.2 Metrics of Privacy Protecting

Privacy Degree. *PPC code has a privacy degree of* t, *if it can resist any* t *breaches. That is, given any* t *out of* n *parity chunks, we cannot reconstruct any data chunk, and there exists* $(t+1)$ *parity chunks that can reconstruct some (not necessarily all) data chunks.*

As described before, PPC code is a kind of non-systematic erasure code, so no original data is retained. In other words, we cannot reconstruct any data if we only hold one chunk. Hence, the privacy degree of PPC must be greater than 1. In this paper, we define $PPC(k, n, t)$ which means k original data chunks generating n coding chunks and all original chunks can be recovered when holding at least k coding chunks, achieving $(n - k)$-erasure correcting and t presents privacy degree.

Safe and Unsafe Group. *Let* S *be a subset of chunks in a stripe. If we cannot reconstruct any data chunk from subset* S*, we call* S *a safe group, otherwise an unsafe group.*

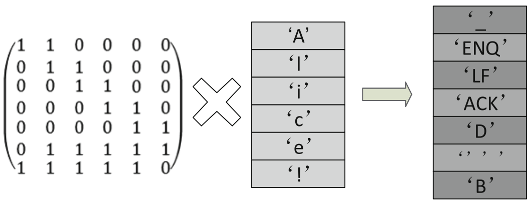


Fig. 3. PPC conversion

Best Case Privacy Degree. *A PPC code has a best case privacy degree of* t^{BC} *if the size of its biggest safe group in the PPC code is* t^{BC}.

To tell the difference between privacy degree and best case privacy degree, we consider a PPC conversion instance as Fig. 3. It's obvious that the chunk D_4 can be reconstructed by the XOR-sum of P_0, P_2 and P_6 and any data chunk cannot be computed by any pair of parity chunks, so the privacy degree is 2. However, not any three out of those seven parity chunk can reconstruct an original chunk. We can see that the first 5 chunks form a safe group, and any bigger group is unsafe, so its t^{BC} is 5.

S-safe Partible and S-safe Partition. *A PPC code is s-safe partible if its stripe can be split into s safe groups. These groups form an s-safe partition of this PPC code.*

For a distributed storage system, we believe attacks are interrelated in some cases. In other words, attackers will breached relevant multiple nodes simultaneously. In the multi-cloud environment, servers which come from the same country may be regarded as relevant nodes if data security involves the issue of government intervention. In other case, natural disasters will cause all nodes within a region unavailable. The above definition offers a solution to these. For example, if we have four providers from one country and three from another country, we can employ the 2-safe partition $\{\{P_0, P_2, P_4, P_6\},\{P_1, P_3, P_5\}\}$ of $PPC(6,7,2)$ as shown in Fig. 3, so attackers cannot reconstruct original data even if he breaks all the cloud severs belonged to one country.

4.3 Structural Properties of $PPC(k, k+1, t)$

As described before, PPC is a kind of XOR-based (binary) codes, so its encoding and decoding can be achieved by multiplying the generator matrix and original data chunks. Hence, for a $PPC(k, k+1, t)$, we need a $(k+1) \times k$ generator matrix G, and this matrix must have the following properties:

1. *Any k of G's row vectors are linearly independent. In other words, every $k \times k$ square sub-matrix of G is invertible.*

   This property ensures the fault tolerance of PPC. If a coding chunk is unavailable, we delete the corresponding row in G so that the remaining k rows forms a $k \times k$ square matrix. If G satisfies the property, all such a sub-matrix is invertible. We can get the original data chunks by matrix multiplication using the inverse of the sub-matrix and the remaining k coding chunks. But if the property cannot be met, it must exists a chunk that cannot be reconstructed after failure.

2. *Casually choose a row v in G, then $ONES(v) \geq 2$.*

The above property ensures data privacy. We define the function $ONES(v)$ as the weight of the vector v, that is, the number of '1' entries in v. As discussed before, the privacy degree of PPC is greater than 1, we only take use of those generator matrices which are purely composed of row vectors whose weights are greater than or equal to 2.

4.4 Construction of PPC(k, k+1, t)

The key to realizing PPC is finding an optimal generator matrix. We design an algorithm to find optimal matrices which satisfy the structural properties by enumerating valid generator matrices. The algorithm needs an initial $k \times k$ matrix as input and we use r_i to represent the i-th row in the initial matrix. We set r_1 to $\overrightarrow{\mathbf{0}_{k-2}\mathbf{1}_2}$ (($k-2$) '0' followed by two '1'), and set r_m to $\overrightarrow{\mathbf{0}_{k-m-1}\mathbf{1}_2\mathbf{0}_{m-1}}$. If the first k rows are independent, let the $(k+1)$-th row be the XOR-sum of the first k row vectors. We calculate the privacy degree of the new matrix and update the optimal code if the new matrix is a better one. Although the structural properties of generator matrix prune the search space effectively, the enumerating algorithm is still exponential time, and we can find optimal PPC codes in an acceptable time only when $k \leq 9$.

In spite of the above size is appropriate for an end-user of multi-cloud storage systems, we still design and implement two infinite families of $PPC(k, k+1, t)$ codes for large k. The only difference between the two kinds of PPC is the generator matrix, so we only introduce one of them, ShrPPC.

4.5 ShrPPC

To construct a $ShrPPC(k, k+1, t)$ generator matrix, we start with constructing the first $(k-1)$ row vectors by shifting two consecutive set bits right as Formula 1. The i-th row vector $r_i = \overrightarrow{\mathbf{0}_{i-1}\mathbf{110}_{k-i-1}}$ satisfies $ONES(r_i) \geq 2$. Then, we construct the k-th row vector: If k is even, it is constructed as $\overrightarrow{\mathbf{01}_{k-1}}$, otherwise, we set r_k as $\overrightarrow{\mathbf{001}_{k-2}}$. Finally, we set the last row vector as the XOR-sum of the first k row vectors. When k is even, it is $\overrightarrow{\mathbf{1}_{k-1}\mathbf{0}}$. When k is odd, it is $\overrightarrow{\mathbf{101}_{k-3}\mathbf{0}}$. Formulas 2 and 3 describe the generator matrices of ShrPPC with even and odd k respectively. It's easy to see that the above two matrices satisfy the second property of generator matrix. Now we will prove they satisfy property 1. First, we transform the first k rows into an upper triangular matrix by performing some elementary matrix transformations. If k is even, we add the $2i$-th row to r_k for all $i \in \{1, 2, ..., \frac{k-2}{2}\}$. And if k is odd, add the $(2i+1)$-th row to r_k for all $i \in \{1, 2, ..., \frac{k-3}{2}\}$. Since the upper triangular matrix is full rank, the first k rows are linearly independent. The $(k+1)$-th row is the XOR-sum of first k rows, so any $k \times k$ sub-matrix of ShrPPC generator matrix is invertible.

$ShrPPC(k, k+1, t)$ provides a good privacy degree of $t = \lfloor \frac{k}{2} \rfloor - 1$, which is very close to the optimal privacy degree of $t = \lfloor \frac{k}{2} \rfloor$, a XOR-based $PPC(k, k+1, t)$ can achieve. For the lack of space, we omit the proof.

4.6 PPC with Higher Security

In the above parts, we introduce PPC whose generator matrix has a minimum row-weight of 2. In other words, each parity chunk is generated by at least two data chunks. It provides high-level performance in most cases, but not all cases. Suppose there are three data chunks, D_0, D_1, D_2, and an attacker has two parity chunks, one of which is $D_0 + D_1$ and the other of which is $D_1 + D_2$. Then he knows the value of $D_0 + D_2$ as well, and it's not complex to guess unencrypted D_0, D_1, and D_2 from this information, even if he's missing other data. Assuming another extreme case that D_0 is a full-zero chunk because of the special file format, it's obvious that the $D_0 + D_1 = D_1$ in this case. In view of these cases, we search optimal generator matrices with larger row-weight using the enumerating algorithm mentioned above. We put the initial matrix of $k \times k$, G, whose first row is $\overrightarrow{\mathbf{0}_{k-3}\mathbf{1}_3}$ and the k-th row is $\overrightarrow{\mathbf{0}_{k-m-2}\mathbf{1}_3\mathbf{0}_{m-1}}$ as input and we have found optimal generator matrices with minimum row-weight larger than or equal to 3 for $k < 9$ (Formula 4 is an example we found using the above method), which means that it becomes much harder for attackers to guess original chunks. In addition, we can resist the above chosen-plaintext attack by means of the combination of PPC and compression and/or encryption.

4.7 Optimal Schedule of PPC Encoding

PPC is a kind of XOR-based code that each coding chunk P is decided by the corresponding row v in the generator matrix G. The greater the weight of v is, the more XOR operations performed in generating P. Suppose G has two vectors of $v_i = \overrightarrow{\mathbf{001}_{k-2}}$ and $v_j = \overrightarrow{\mathbf{0001}_{k-3}}$ and therefore it takes $(2k-7)$ XOR operations to calculate the two parity chunks P_i and P_j. However, if we calculate P_j first, and then calculate P_i by XORing P_j and D_2, only $(k-3)$ XOR operations are needed. Since this paper focuses on small-scale multi-cloud systems, we adopt a breadth-first search algorithm [14] to determine the optimal schedule for PPC encoding.

$$\begin{pmatrix} 1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 1 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 1 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 1 \\ 0 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 1 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 1 \\ 0 & 0 & 1 & \cdots & 1 & 1 \\ 1 & 0 & 1 & \cdots & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

(1) (2) (3) (4)

5 Multi-level Security

For better user experiences, CloudS provides some security levels with different characteristics. From the system perspective, each level logically corresponds to a specific combination scheme of compression, encryption and coding. In CloudS, we implement not only combinations of some existing technologies such as RS, RAID-5, AES [12] and AONT [10], but some new technologies such as PPC and PCE. In this section, we list three new combination schemes and AONT−RS [19] by contrast to introduce different security levels in the system view.

5.1 PCE−PPC

Traditional encryption usually brings key management issues, AONT can be regarded as a better precept to avoid it. The principle of AONT has been amply demonstrated, we don't discuss it any more. Further, AONT−RS, an encryption and dispersal scheme, is proposed in [19] which achieves a pretty high level of security. AONT−RS has another property that it provides partial data the same protection degree as the whole data, e.g., if there're 7 servers used in CloudS, attackers must break 6 servers to retrieve (which is nearly impossible) whether the whole data or a single original byte. In terms of performance, parallel optimization and data extension is difficult in AONT, meanwhile, finite field arithmetic used in RS is relatively complex.

To obtain better performance as well as approximate security, we design a new encryption method called PCE. Figure 4 shows an example of PCE and $ShrPPC(6, 7, 2)$, after that, we randomly spread the coding chunks to different cloud sites. In PCE, chunk D_i is encrypted into C_i with a unique random secret key k_i in parallel, then we append k_i at the end of $C_{(i+3)mod6}$. It's no doubt that PCE overmatches AONT in performance for its parallel optimization. Besides, PPC after PCE avoids the possible chosen-plaintext attack because we perform PPC on cipher not plaintext. Also, PPC, a kind of binary code, is faster than RS. Using PCE−PPC, attackers must hold $\{C_j, k_{(j-3)mod6}\}$ and $\{C_{(j+3)mod6}, k_j\}$

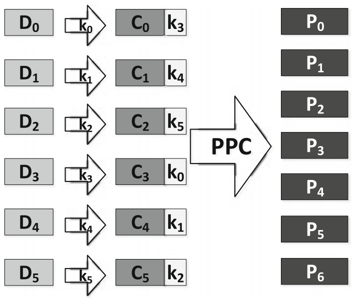


Fig. 4. PCE and PPC

which needs to break 5 cloud servers to decrypt a single chunk D_j, and 6 coding chunks are needed to retrieve all chunks. In a word, PCE−PPC is superior to AONT−RS in terms of performance and achieves the same security as AONT−RS for overall data. Compared with the following schemes containing compression, PCE−PPC provides stable performance without the influence of the repetition rate but more data will be transferred on Internet, thereby satisfying the small-size files with a lower repetition rate.

5.2 Compression−PPC and Compression−PCE−PPC

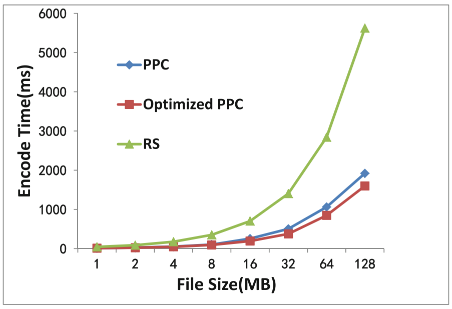
Considering the time-consuming network transmission process, GZip [1] compression is joint in these two schemes. Compared with AONT−RS, these two schemes have an overwhelming advantage in network transmission. As for security, non-systematic $PPC(k, k+1, t)$ itself has the effect of data protection, t (greater than 1) coding chunks (means at least two cloud servers) are needed for retrieving only one original chunk, which is very difficult. Meanwhile, chosen-plaintext attack will be remitted since original data patterns will be destroyed after compression. Nonetheless, decompression doesn't need any key, so Compression−PPC has relatively weak security but high speed. We suggest users to adopt Compression−PPC for less-important data. Compression−PCE−PPC is the synthesis of PCE−PPC and Compression−PPC which meets the demands for large confidential data.

6 Evaluation

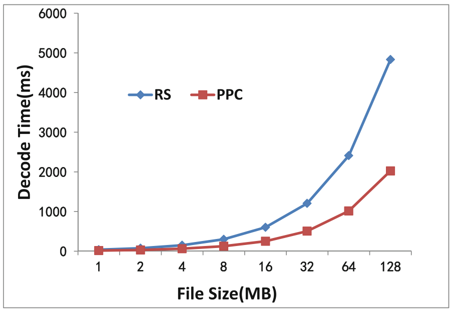
In this paper, we proposed a family of non-systematic erasure codes named PPC and CloudS with different security schemes to ensure data security in cloud storage environment. Meanwhile, we also concern about their performance. In this section, we will report some performance results. Our prototype is implemented in C++ and the experiments are conducted on Ubuntu 13.04 with Intel i3-350M@2.26 GHz.

6.1 PPC Performance

There're two important indicators to measure PPC, the encoding and decoding speed. To measure those performance, we implement encoders and decoders for PPC which generator matrix as shown in Formula 4 and the single fault-tolerant RS for contrast. Figure 5(a), (b) presents the encoding and decoding speeds of PPC and RS. For PPC, all the operations are over $GF(2)$, it's obvious that XOR operation has a higher-speed than operations over $\mathrm{GF}(2^8)$ used by RS. Besides, we compare encoding time between original PPC scheme and optimized PPC schedule. Original schedule needs 15 XOR operations and only 11 are needed in the optimized schedule. It's clear that the optimization becomes more obvious when encoding larger data. PPC enables cloud storage customers to explore trade-offs between a little extra storage cost and data confidentiality and integrity. Therefore, we believe that PPC is practical choice in the multiple clouds environment.

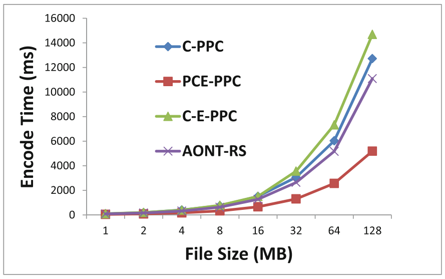


(a) Encoding speed

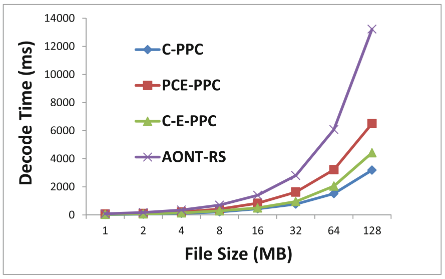


(b) Decoding speed

Fig. 5. PPC performance



(a) Encoding speed between different security schemes



(b) Decoding speed between different security schemes

Fig. 6. Security schemes performance

6.2 Performance of Different Security Schemes

In this part, we report scheme performance without network transmission process. We use C−PPC to represent Compression−PPC and C−E−PPC to symbolize the Compression−PCE−PPC. Figure 6(a) presents the encoding speed, two schemes without compression do better because the compression speed is much lower than encryption. PCE−PPC has the greatest performance due to PPC's better capability and parallel optimization using PCE. Figure 6(b) distinguishes the decoding speed between different schemes. Different from the lower compression speed, decompression speed is much faster, schemes with compression perform better. Certainly, PCE−PPC has decent decoding performance after parallel optimization. We can clearly figure that the performance gap of schemes is widening when the data amount increases. Above all, whether for encoding or decoding performance, PCE−PPC indeed excels AONT−RS.

Table 1. System performance

(a) Uploading Time(s).

File Size(MB)	1	8	16	64	128
C−PPC	0.87	3.90	7.29	21.01	45.28
PCE−PPC	1.51	6.23	13.49	43.45	87.71
C−E−PPC	1.08	4.18	9.07	25.80	51.86
AONT−RS	1.78	8.96	16.61	47.15	96.42

(b) Downloading Time(s).

File Size(MB)	1	8	16	64	128
C−PPC	0.98	4.25	8.40	36.38	76.83
PCE−PPC	1.51	8.73	18.69	71.73	168.44
C−E−PPC	1.09	6.04	10.20	39.00	80.83
AONT−RS	1.91	9.44	19.94	75.39	177.70

6.3 System Performance

In CloudS, there're 3 steps in uploading and downloading process: the transmission between client and CloudS agent, operations on CloudS and the transmission between CloudS and cloud servers. The performance of the first step is up to network condition which is out of our research range, here we only test the last two performance. It's notable that some uncontrolled exterior factors such as network condition and the repetition rate of files will also influence our results. We create accounts on Vdisk to simulate multiple clouds environment and adopt randomly generated test files. We run Apache on PC for test which means a real web server will do better.

Table 1(a), (b) respectively presents the uploading and downloading performance. It's obvious that the schemes containing compression do much better due to less data being delivered. C−PPC does the best on account of less data to be transferred and less mechanism being used. PCE−PPC has better performance than AONT−RS on account of PPC's more optimal speeds and parallel optimization using PCE. Certainly, all those schemes are in the range users can accept.

7 Conclusion and Future Work

With the development of cloud storage, the security issues become more and more serious, data integrity and availability is being threatened. We design a new family of erasure code called PPC to guarantee data security. PPC is XOR-based code that results in a low storage computational cost. Simultaneously, for a single-fault-tolerance code, PPC$(k, k+1, t)$ only need $\frac{(k+1)}{k}$ times extra storage overhead to achieve the trade-offs between storage and fault tolerance. It is important to note that PPC, as a kind of non-systematic code, has a better function of confusion.

With PPC code, we put forward a multi-cloud system called CloudS. Though many multi-cloud systems have appeared in recent years such as RACS and Depsky, they cannot fully adapt to users' different requirements for different files in cloud environment. The single-level security scheme will give rise to a worse user experience and system performance. Specially for the devices with weak computing ability and limited computing resources, redundant and complex security mechanisms will become the bottleneck of the system. In CloudS, we provide several different combination schemes of compression, encryption and coding to achieve multi-level security for better performance and user experiences. At the same time, we implement CloudS as a web application which doesn't require users to do more operation locally.

In our future work, we would like to extend our PPC codes to resist double. In addition, we can still optimize CloudS in terms of performance.

Acknowledgments. This work is partially supported by NSF of China (grant numbers: 61373018, 11301288), Program for New Century Excellent Talents in University (grant number: NCET130301) and the Fundamental Research Funds for the Central Universities (grant number: 65141021).

References

1. Gzip. http://en.wikipedia.org/wiki/Gzip
2. China personal cloud storage industry and users' behavior research. http://www.iimedia.cn/38351.html
3. Dependable storage in the Intercloud. http://domino.research.ibm.com/library/cyberdig.nsf/papers/630549C46339936C852577C200291E78
4. Abu-Libdeh, H., Princehouse, L., Weatherspoon, H.: RACS: a case for cloud storage diversity. In: Proceedings of the 1st ACM Symposium on Cloud Computing, pp. 229–240. ACM, Indianapolis (2010)
5. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable data possession at untrusted stores. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, pp. 229–240. ACM, Alexandria (2007)
6. Bessani, A., Correia, M., Quaresma, B., André, F., Sousa, P.: DepSky: dependable and secure storage in a cloud-of-clouds. ACM Trans. Storage (TOS) **9**, 12 (2013)
7. Bowers, K.D., Juels, A., Oprea, A.: HAIL: a high-availability and integrity layer for cloud storage. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, pp. 187–198. ACM, Chicago (2009)
8. Cachin, C., Keidar, I., Shraer, A.: Trusting the cloud. ACM SIGACT News **40**, 81–86 (2009)
9. Ghemawat, S., Gobioff, H., Leung, S.T.: The google file system. In: ACM SIGOPS operating systems review, pp. 29–43. ACM, New York (2003)
10. Hafner, J.L.: WEAVER codes: highly fault tolerant erasure codes for storage systems. In: 4th Conference on File and Storage Technologies, pp. 16–16. USENIX, San Francisco (2005)
11. Juels, A., Kaliski Jr., B.S.: PORs: proofs of retrievability for large files. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, pp. 584–597. ACM, Alexandria (2007)
12. Daemen, J., Rijmen, V.: The Design of Rijndael: AES-The Advanced Encryption Standard. Springer Science and Business Media, Heidelberg (2013)
13. Patterson, D.A., Gibson, G., Katz, R.H.: A case for redundant arrays of inexpensive disks (RAID). ACM (1988)

14. Plank, J.S., Schuman, C.D., Robison, B.D.: Heuristics for optimizing matrix-based erasure codes for fault-tolerant storage systems. In: Proceedings of the 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 1–12. IEEE/IFIP, Boston (2012)
15. Plank, J.S., et al.: A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. Softw. Prac. Exp. **27**, 995–1012 (1997)
16. Plank, JS, Huang C: Tutorial: erasure coding for storage applications. In: Slides presented at FAST-2013: 11th Usenix Conference on File and Storage Technologies. USENIX, San Jose (2013)
17. Rabin, M.O.: Efficient dispersal of information for security, load balancing, and fault tolerance. J. ACM (JACM) **36**, 335–348 (1989)
18. Rashmi, K., Nakkiran, P., Wang, J., Shah, N.B., Ramchandran, K.: Having your cake and eating it too: jointly optimal erasure codes for I/O, storage, and network-bandwidth. In: Proceedings of the 13th USENIX Conference on File and Storage Technologies, pp. 81–94. USENIX, Santa Clara (2015)
19. Resch, J.K., Plank, J.S.: AONT-RS: blending security and performance in dispersed storage systems. In: Proceedings of the 9th USENIX Conference on File and Storage Technologies. USENIX, San Jose (2011)
20. Singleton, R.: Maximum distance-nary codes. In: IEEE Transactions on Information Theory, pp. 116–118. IEEE (1964)
21. Shamir, A.: How to share a secret. Communications of the ACM. **22**, 612–613 (1979)
22. Sun, J., Xu, M., Feng, S., Li, Z., Wang, G., Liu, X.: Secure store of user authentication tokens in multi-cloud storage system. J. Comput. Inf. Syst. **11**, 1013–1020 (2015)