基于再生码的拟态数据存储方案

陈越[1]，王龙江[1,2]，严新成[1]，张馨月[1]
（1. 解放军信息工程大学数据与目标工程学院，河南 郑州 450001；2. 解放军 61660 部队，北京 100089）

摘 要：针对云存储系统由于静态的存储架构和存储模式而带来的安全威胁，提出一种基于再生码的拟态化存储方案。该方案利用网络编码方案将数据存储在云端数据节点上，采用一种基于再生码的拟态变换机制，可根据随机时变因素动态地改变数据的存储状态，且能够保证数据完整性和数据持续可用性。拟态变换机制具有随机性、时变性和动态性，通过增加存储系统的不确定性，可阻断和干扰攻击链，增加了攻击者实施攻击的难度和成本，提高了系统的安全性和可靠性。

关键词：拟态；网络编码；再生码；云存储
中图分类号：TP309.2
文献标识码：A
doi: 10.11959/j.issn.1000-436x.2018066

Mimic storage scheme based on regenerated code

CHEN Yue[1],WANG Longjiang[1,2], YAN Xincheng[1], ZHANG Xinyue[1]
1. College of Target and Data Engineering, PLA Information Engineering University, Zhengzhou 450001, China
2. 61660 Unit of PLA, Beijing 100089, China

Abstract: Aiming to solve security threats in the cloud storage system due to static storage architecture and storage mode, a mimic storage scheme based on regenerated code was proposed. The scheme used network coding scheme to store the data in the cloud data node, and used mimicry transformation mechanism based on regeneration code to change data storage state dynamically according to the random time-varying factors, which could guarantee data integrity and data availability continuously. The mimicry transformation mechanism is a random, time-varying and dynamic scheme, which increases the uncertainty of storage system. It blocks and interferes with the attack chain, increases the difficulty and cost of the attack operation, and improves the security and reliability of the system.

Key words: mimic, network coding, regenerated code, cloud storage

1 引言

随着大数据和云计算技术的快速发展和日益普及，越来越多的用户选择将自己的数据托管在云存储平台（如 Amazon S3[1]、Windows Azure[2]）上，但是，现有的云存储平台面临着一系列固有的安全问题，严重威胁着用户的数据安全。静态的系统结构和存储模式容易暴露系统的脆弱性，信息系统内部的软硬件漏洞难以避免，传统的被动防御策略难以防范日益复杂的攻击手段。云存储平台的这些安全问题已经严重制约着云存储技术的发展和应用。

本文针对云存储系统确定性存储模式带来的安全威胁，提出了一种拟态数据存储方案，该机制在存储过程中引入了冗余性、随机性和时变性，支持数据的快速恢复和重构，提高了系统的容错性和抗毁性，可保证数据的完整性和持续可用性。通过再生码对编

收稿日期：2017-07-05；**修回日期**：2018-01-27
通信作者：王龙江，wlj2016@foxmail.com
基金项目：国家重点基础研究发展计划（“973”计划）基金资助项目（No.2012CB315901）；河南省科技攻关计划基金资助项目（No.17210221001）
Foundation Items: The National Basic Research Program of China (973 Program) (No.2012CB315901), The Key Technologies R&D Program of Henan Province (No.17210221001)

码数据进行动态、随机的变换，改变了传统云存储的静态存储模式，使系统的攻击表面难以预测，隐藏了系统的脆弱性和用户的操作信息，压缩了攻击者可利用的时间窗口，增加了攻击的难度和成本，可有效提高云存储系统的可靠性和安全性。

2 相关工作

目前，云存储平台多采用增加数据冗余的方式来保证数据安全，主要的方案有以下3类。1) 基于多副本的云存储方案，它将数据存储为多个副本来确保用户的数据安全，如谷歌文件系统[3]（GFS, Google file system）、Hadoop 分布式文件系统[4]（HDFS, Hadoop distribute file system），这类方案简单易行，但存储空间浪费严重，缺乏有效的安全机制。2) 基于纠删码的云存储方案[5]，该方案通过纠删码算法来存储和恢复数据，大大提高了空间利用率，但修复开销太大。3) 基于再生码的云存储方案，该方案基于网络编码理论，是一种改进的纠删码，可有效减少修复带宽，且具有更好的安全性，得到了广泛的研究和应用。

但是现有的云存储系统由于自身的静态存储模式和确定性的系统结构，往往采用被动式的防御策略，难以有效防范日益复杂多样的攻击手段，不能有效保护用户的数据安全。拟态安全防御技术[6]是近几年提出的一种主动防御技术，旨在通过增加系统的不确定性来阻断攻击链，为防范未知类型的攻击和潜在的安全风险开辟了新途径，在学术界和产业界引起广泛的关注。

拟态安全防御技术通过在信息系统中引入动态性、异构性、冗余性，构建一个动态可变的网络环境和程序运行环境，通过在不同的环境之间快速地跳变和迁移，使系统的攻击表面难以预测，从而增加了攻击者利用系统脆弱性实施攻击的难度和代价，可防范基于未知漏洞和后门的攻击手段。

理论研究方面，邬江兴[6]首先提出了拟态安全防御的思想并介绍了其技术特点。文献[7]介绍了拟态计算和拟态安全防御的原意和愿景，对其研究问题和应用背景做了阐述。文献[8]阐述了动态异构冗余架构以及如何利用不可信软硬构件组成高可靠、高安全等级信息系统的原理与方法，给出了拟态防御的基本概念。文献[9]通过分析对比拟态防御和入侵容忍、移动目标等技术的异同，介绍了拟态安全防御技术的特点和优势。文献[10]对拟态安全防御问题进行形式化建模，分析和阐述了拟态防御技术的构成、拟态防御的科学问题及其理论框架。文献[11]提出了将拟态防御技术与软件多样化技术相结合应用于软件安全产业的新思路。文献[12]提出了以多维重构函数化体系结构与动态多变体运行机制为核心的拟态计算和拟态安全防御技术体系。文献[13]介绍动态网络的主动变迁技术，提出了演进防御机制，该机制可以根据网络系统安全状态、网络系统安全需求等，选择最佳的网络配置变化元素组合来应对潜在的攻击，保证特定等级的安全要求。

工程实践方面，文献[14]提出了攻击链模型，针对已有安全技术的不足和问题，提出了基于“动态异构冗余”结构的拟态防御模型，设计实现了拟态防御 Web 服务器系统，并进行了安全性分析和性能测试。文献[15]提出了一种适用于拟态防御架构的 Web 服务器测试方法，基于让步灰盒测试，对拟态 Web 服务器原型系统进行性能、兼容性、功能实现、安全性等方面的测试。

目前，拟态安全防御技术的研究仅局限于计算机系统结构上，本文将拟态安全防御的思想应用到云存储领域，提出了一种拟态变换机制，利用再生码技术实现了数据存储状态的动态时变切换，支持数据的动态修复和快速自愈，增加了攻击的难度和成本，提高了系统的容错能力和安全性，可有效保证数据的完整性和可用性。

3 拟态存储原理

本节主要介绍拟态存储的基本原理和核心技术。

拟态化存储的基本思路为在数据存储和访问的过程中，加入时变和随机因素，实现数据存储状态的动态可变，从而增加攻击者获取数据的难度和成本，提高系统的可靠性和安全性。

拟态化存储机制主要包括数据的上传、下载和拟态变换 3 个部分。其中，数据的上传和下载使用现有的网络编码存储技术就能实现，网络编码存储方案具有天然的冗余性，相关方案已经比较成熟。

数据上传时，首先对文件进行加密处理，将文件对象切分成固定大小的数据块。然后采用随机线性编码对其进行编码，将编码后的数据块上传给云存储系统，存储在各个数据节点上。

数据读取时，从云存储系统中随机下载含有足够冗余信息的编码块，经过译码还原出原始数据块，再通过合并操作和解密操作即可得到原始的用户文件。

数据的拟态变换利用功能性最小存储再生[16]（FMSR, functional minimum storage regenerated）码来实现，主要依赖于 FMSR 码良好的修复性能，通过控制变换时机和编码系数的选取，可实现数据存储状态的随机时变切换。

用户和云端数据节点之间使用拟态变换协议来协商变换参数，每个数据节点根据拟态变换参数变换其存储内容，以达到数据存储状态动态可变的目的。数据节点的拟态变换过程类似于失效节点的修复过程，即将当前节点视为需要修复的数据节点，从其他节点上下载数据，利用随机编码技术重新构造新的数据块，覆盖现有内容，以达到拟态变换的目的。经过拟态变换后的数据节点，必须确保能够通过译码操作还原出原始数据，确保用户数据的完整性和持续可用性。

由于整个变换过程中不需要对数据进行译码，云端数据一直处于编码状态，编码参数始终由客户端控制，每次拟态变换的参数由客户端经过计算生成。攻击者在不掌握初始编码参数和变换规则的情况下，难以还原出原始文件。由于变换过程中引入了随机和时变因素，增加了穷举攻击的难度。拟态变换协议使用加密的通信链路来协商变换参数，减少了来自不可信网络环境的安全威胁。

FMSR 码是一种支持功能性修复的最小存储再生码，属于典型的 (n,k) 最大距离可分（MDS, maximum distance separable）码，保持了 MDS 码良好的容错能力和存储效率。对于一个大小为 M 的文件，(n,k) FMSR 码将其切分成 $k(n-k)$ 个固定大小的原始块，再将它们编码成 $n(n-k)$ 个编码块，上传给 n 个数据节点，每个节点存储 $n-k$ 个编码块。数据读取过程中，首先，随机挑选任意 k 个节点，下载 $k(n-k)$ 个编码块，然后，对其进行译码操作，还原出原始数据块，最后，将数据块合并成原始文件。

当某个数据节点因为意外情况失效了（例如，系统故障或外部原因导致无法对外提供服务），为了保证数据的安全性和服务的连续性，必须尽快对其上的数据进行修复。数据的重构过程需要在其他 $n-1$ 个数据节点上各取一个数据块，将这 $n-1$ 个数据块重新编码生成 $n-k$ 个编码块，替代失效节点的数据，(4,2)FMSR 码的修复过程如图 1 所示。

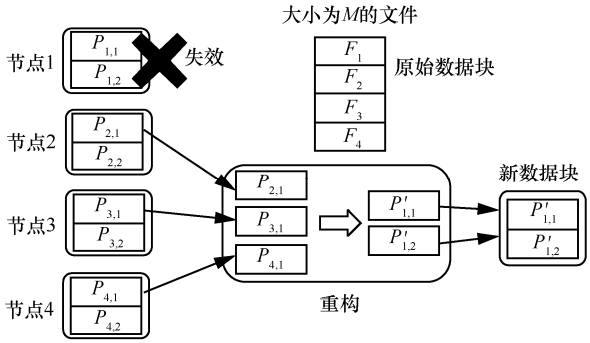

图 1　(4,2)FMSR 码的修复过程

4　数据拟态化存储机制

本节对数据的拟态存储机制进行深入的讨论，首先，给出了拟态存储系统所使用的存储框架和存储模型，然后，分别介绍数据的上传和下载过程，最后，详细阐述拟态变换机制。

4.1　存储框架

实际的云存储系统异常复杂，为了讨论问题的方便，本文对云存储系统进行简化和抽象。存储框架如图 2 所示，主要分客户端系统和云端存储系统 2 个部分。客户端系统通过互联网与各个数据节点相连。云端存储系统由一系列分布于网络中的数据节点组成，它们通过网络连接构成一个分布式存储系统。这里的数据节点是对下层存储系统进行抽象，可兼容多样化异构的存储设备，既可以是大型的数据中心，如 Amazon 的 S3、Microsoft 的 Azure 等，也可以是普通的 PC 机、服务器、NAS 设备等。

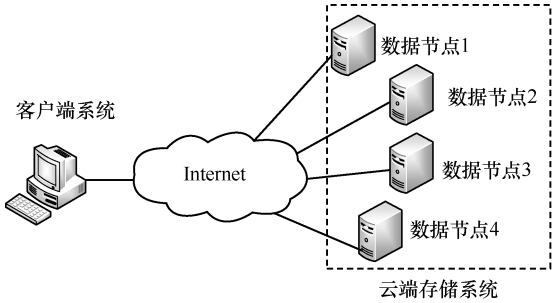

图 2　存储框架

用户端通过运行客户端程序与云端数据节点进行交互，客户端负责数据的上传和下载以及必要

的身份鉴别和访问控制。客户端与数据节点通过拟态变换协议，完成拟态变换参数的协商。客户端还负责维护编码参数和加密密钥等信息。本节只讨论对单个文件的拟态存储过程，多个文件的存储原理相同。

4.2 数据的上传

数据的上传需要经过以下 2 个步骤。

1) 客户端首先对文件进行预处理，对文件进行加密后，切分成块。假设文件大小为 M，将其切分成 $k(n-k)$ 个固定大小的原始数据块，记作 $(F_i)_{i=1,2,\cdots,k(n-k)}$，每个数据块的大小为 $\frac{M}{k(n-k)}$。

2) 客户端对原始数据块进行编码，得到 $n(n-k)$ 个编码数据块，记作 $(P_i)_{i=1,2,\cdots,n(n-k)}$，每个编码块都是 $k(n-k)$ 个原始数据块的线性组合。

具体的编码过程如下。

Step1 构造一个 $n(n-k)\times k(n-k)$ 的编码矩阵 $\boldsymbol{EM}=[\boldsymbol{\alpha}_{i,j}]$，其中，元素 $\boldsymbol{\alpha}_{i,j}$ 均是从有限域 $GF(2^w)$（一般取 $w=8$）中随机产生，为了保证能够正确译码，要求 $\boldsymbol{EM}$ 必须满足 MDS 性质。

Step2 使用编码矩阵与原始块相乘，即可得到 $n(n-k)$ 个编码块。$\boldsymbol{EM}$ 中每个行向量称为一个编码向量（ECV, encoding coefficient vector），对应于一个编码块，包含了 $n(n-k)$ 个编码系数。第 i 个编码块的编码过程可表示为

$$\boldsymbol{P}_i=\boldsymbol{ECV}_i\boldsymbol{F}=\sum_{j=1}^{k(n-k)}\boldsymbol{\alpha}_{i,j}F_j \tag{1}$$

其中，$i=1,2,\cdots,n(n-k)$，编码中涉及加法和乘法运算遵循有限域 $GF(2^w)$ 上的运算规则。

Step3 客户端将 $n(n-k)$ 个编码数据块上传给 n 个数据节点，每个节点存储相邻的 $n-k$ 个数据块，编码矩阵 $\boldsymbol{EM}$ 由客户端存储和维护。

当 $n=4$、$k=2$ 时，数据的上传过程如图 3 所示。

4.3 数据的下载

1) 客户端。从 n 个数据节点中任取 k 个下载其所有的编码块（一般取负载较小的节点），共计 $k(n-k)$ 个编码块，从编码矩阵 $\boldsymbol{EM}$ 中取出这些数据块对应的编码向量，组成一个 $k(n-k)\times k(n-k)$ 阶的方阵，记作 $\boldsymbol{EM}'$。由于 $\boldsymbol{EM}'$ 是从 $\boldsymbol{EM}$ 中产生的，它的各个行向量线性无关，其逆矩阵必然存在。

2) 客户端。将 $\boldsymbol{EM}'^{-1}$ 乘以下载的编码块即可得到 $k(n-k)$ 个原始块，将其合并组装，再经过解密即可得到原始文件。

当 $n=4$、$k=2$ 时，数据的下载过程如图 4 所示。

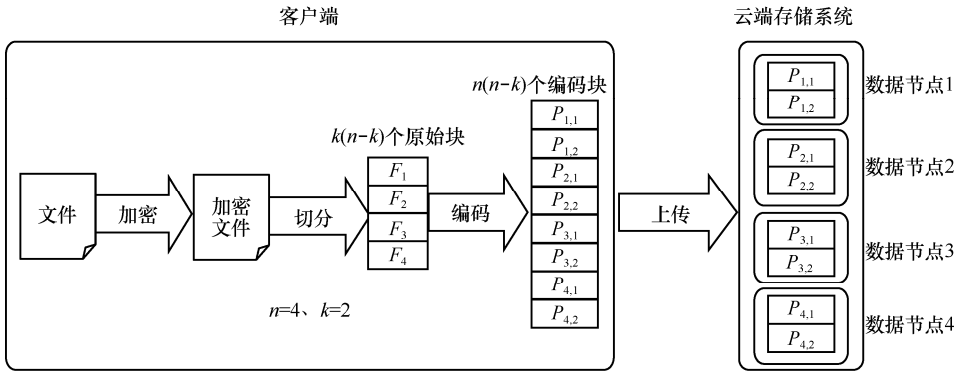

图 3 n=4、k=2 时的数据上传过程

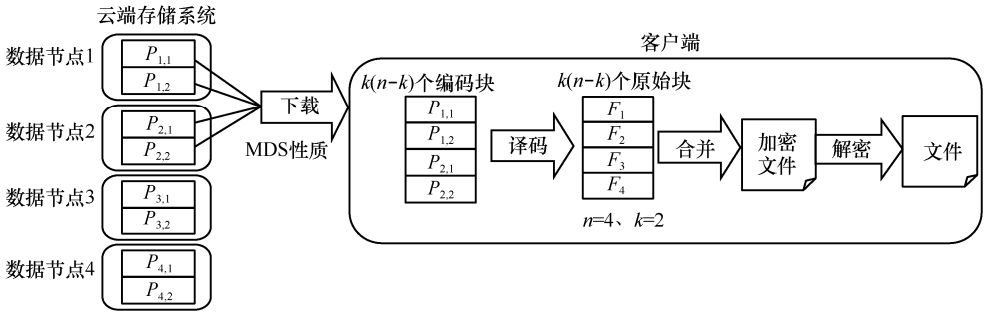

图 4 n=4、k=2 时的数据下载过程

4.4　数据的拟态变换

4.4.1　拟态变换

在数据节点 $N_1,N_2,\cdots,N_n$ 中，假设对数据节点 N_t 上的数据进行拟态变换，单轮拟态变换需经过以下 7 个步骤。

Step1　客户端在除了 N_t 以外的每个节点上，随机挑选一个编码块（共有 $(n-k)^{n-1}$ 种不同的可能），在编码矩阵中取出这 $n-1$ 个编码块所对应的编码向量，记作 $\boldsymbol{ECV}_{i_1},\boldsymbol{ECV}_{i_2},\cdots,\boldsymbol{ECV}_{i_{n-1}}$，这一操作实际上在客户端完成，只需要对编码矩阵进行操作，不涉及数据块的读取。

Step2　客户端构造一个 $(n-k)\times(n-1)$ 大小的变换矩阵 $\boldsymbol{TM}=[\boldsymbol{\gamma}_{i,j}]$，其中，$i=1,2,\cdots,n-k$，$j=1,2,\cdots,n-1$，元素 $\boldsymbol{\gamma}_{i,j}$ 均从有限域 $GF(2^w)$（一般取 $w=8$）中随机产生。

Step3　客户端为新的编码块计算编码向量，$\boldsymbol{TM}$ 乘以 $[\boldsymbol{ECV}_{i_1},\boldsymbol{ECV}_{i_2},\cdots,\boldsymbol{ECV}_{i_{n-1}}]^{\mathrm{T}}$ 得到 $n-k$ 个新的编码向量 $\boldsymbol{ECV}'_1,\boldsymbol{ECV}'_2,\cdots,\boldsymbol{ECV}'_{n-k}$，该过程可表示为

$$\begin{aligned}&[\boldsymbol{ECV}'_1,\boldsymbol{ECV}'_2,\cdots,\boldsymbol{ECV}'_{n-k}]^{\mathrm{T}}\\&=\boldsymbol{TM}[\boldsymbol{ECV}_{i_1},\boldsymbol{ECV}_{i_2},\cdots,\boldsymbol{ECV}_{i_{n-1}}]^{\mathrm{T}}\end{aligned}\tag{2}$$

其中，每个新的编码向量可表示为

$$\boldsymbol{ECV}'_i=\sum_{j=1}^{n-1}\boldsymbol{\gamma}_{i,j}\boldsymbol{ECV}_{i_j},i=1,2,\cdots,n-k\tag{3}$$

Step4　客户端用 $[\boldsymbol{ECV}'_1,\boldsymbol{ECV}'_2,\cdots,\boldsymbol{ECV}'_{n-k}]$ 替换原来编码矩阵 $\boldsymbol{EM}$ 中的数据节点 N_t 所占部分，形成一个新的编码矩阵 $\boldsymbol{EM}'$。

Step5　客户端判断 $\boldsymbol{EM}'$ 是否满足 MDS 性质，即判断 $\boldsymbol{EM}'$ 中任意 $k(n-k)$ 个行向量组成的方阵是否满秩。若满足 MDS 性质，执行 Step6，否则，跳到 Step1，进入下一轮迭代，重新挑选新的编码向量和变换系数。

Step6　客户端将 $\boldsymbol{TM}$ 以及从 Step1 中挑选出的编码块序号发送给数据节点 N_t。

Step7　数据节点 N_t 从相应的数据节点上下载 Step1 中的挑选出的编码块，并使用 $\boldsymbol{TM}$ 对这些编码块进行编码得到 $n-k$ 个数据块，覆盖自身原有的数据，完成一次拟态变换。

当 $n=4$、$k=2$ 时，对数据节点 1 进行拟态变换的过程如图 5 所示。

4.4.2　拟态变换协议

拟态变换协议用于用户端系统与数据节点间变换参数的协商。为了保证交互信息的安全性，拟态变换协议使用了 TSL 协议来构建安全信道，对交换的控制信息进行了加密处理。用户端系统和数据节点间的交互过程如图 6 所示。

首先，客户端上传数据到数据节点，当该节点上的数据完成存储时，会向客户端发送一个存储完成消息（SCM, storage completed message）。

当客户端收到所有数据节点发来的存储完成消息后，说明所有的数据块已经存储完成，向每个数据节点发送存储完成消息确认消息。

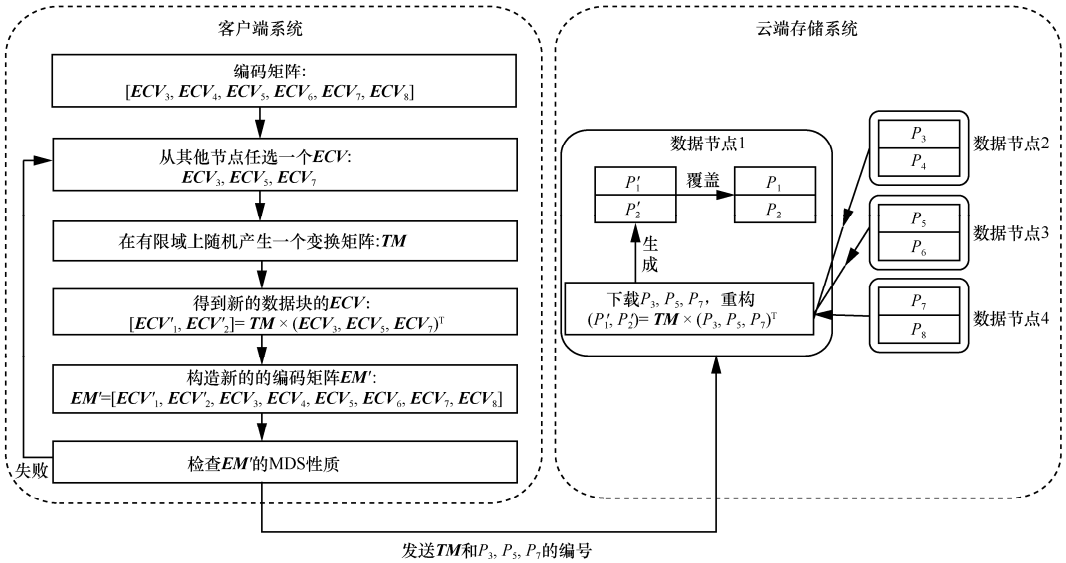

图 5　n=4、k=2 时对数据节点 1 的拟态变换过程

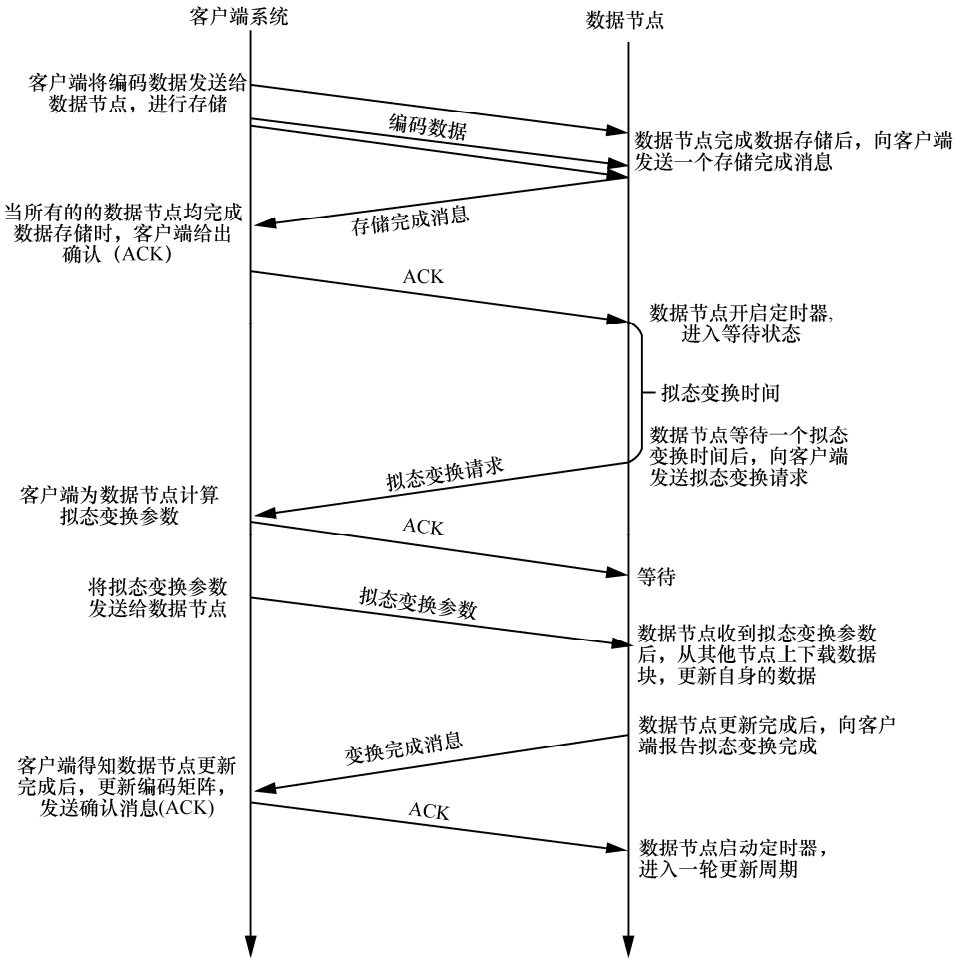

图 6　拟态变换协议示意

每个数据节点设置一个定时器，当收到客户端给出的确认后，启动定时器，定时器的初始时间设置为一个随机值。

当定时器时间到时，数据节点向客户端发送一个拟态变换请求（MTR, mimic transform request），MTR 消息中包含该数据节点的编号、编码块序列号区间等信息。

客户端收到这个 MTR 消息后，查看所有的数据节点的状态信息，如果其他的数据节点均处于存储完成的状态，就允许这个更新请求，并发送确认消息，并根据编码矩阵为该数据节点构造拟态变换参数。拟态变换参数包括 2 个部分：需要读取的编码块序号列表和一个用于产生新的编码块的拟态变换矩阵 ***TM*** 。

数据节点收到确认消息后，就处于等待状态，等待客户端发来的拟态变化参数。当收到客户端发来的拟态变化参数时，从其他节点下载指定序号的编码数据块，并利用 ***TM*** 对它们进行编码，用新的编码块覆盖当前的数据。当变换操作完成后，数据节点向客户端发送一个变换完成消息（TCM, transform completed message），客户端收到这个消息后，得知数据节点已经完成了数据更新，于是，根据 ***TM*** 更新编码矩阵，并向数据节点端发送确认消息。

数据节点收到确认消息后，启动定时器，进入下一个拟态变换周期。

4.4.3　拟态变换周期的确定

拟态变换周期通常以小时为单位，初始化为一个[1,10]之间的随机值，并随着数据节点的运行状态和网络的安全态势的变化而变化。

每个数据节点需要维持一个资源值 r ，用来描述当前系统的资源使用情况，该值由系统当前的运行状态（包括 CPU 利用率、内存利用率、I/O 速率、出口带宽等）计算出来，可表示为

$$r = resource(cpu\_usage, ram\_usage, I/O\_rate, outbound\_bandwidth, \cdots) \quad (4)$$

r 越大，表示当前节点的系统资源越充足；r 越小，表示当前节点的系统资源越紧缺。

每个数据节点还要维持一个安全值 s，用来反映当前系统的安全状况。s 越大，表示当前系统越安全；s 越小，表示当前系统面临的安全风险越大。安全值通过对系统进行一系列的安全性评估而得出。使用基于知识库的安全评估方法，对系统面临的安全风险进行评估，进而计算出一个安全值来刻画系统当前的安全状况和防御能力。

每次拟态变换完成后计算一次资源值和安全值，新的拟态变换周期 T_{new} 可表示为

$$T_{\text{new}} = T_{\text{old}} \frac{\alpha w_{\text{new}} + \beta s_{\text{new}}}{\alpha w_{\text{old}} + \beta s_{\text{old}}} \tag{5}$$

其中，$\alpha,\beta \in [0,1]$，$\alpha+\beta=1$，α 和 β 分别表示资源值和安全值的权重。

5　拟态变换的建模与分析

拟态变换过程必须保证数据经过变换后依然能够正确译码，变换过程不能破坏数据的完整性和可用性。本节对 4.4 节讨论的拟态变换过程进行了深入的分析，给出了拟态变换后数据持续可用的严格证明。本节讨论 $k=n-2$，即双节点冗余的情形（目前的云存储系统中，普遍使用双节点冗余的设置），如下的讨论均基于这个前提条件。

定义 1　MDS 性质。在再生码中，对于一个 nx 个编码块的数据集，若其中任意 kx 个编码块，即可解码出 kx 个原始块，则该数据集满足 MDS 性质。其中 $n>k$，x 为一个存储单元的数据块个数。

对于一个 $nx\times kx$ 的矩阵来说，若其中任意的 kx 个行向量是线性无关的，则该矩阵满足 MDS 性质，反之亦然。其中 $n>k$，x 为任意正整数。

因此，不难得出若一个编码数据集满足 MDS 性质，则其编码矩阵也满足 MDS 性质。

定义 2　可译码性。如果 kx 个编码块的数据集可以通过译码操作还原出 kx 个原始数据块，则说明该数据集满足可译码性。其中，x 为一个存储单元的数据块个数。

因此，若一个 nx 个元素的数据集满足 MDS 性质，则它的任意一个大小为 kx 的子集均满足可译码性，x 为任意正整数。

引理 1　$n(n-k)$ 个编码块组成的数据集 S，S' 是 S 的大小为 $k(n-k)$ 的任意子集。S' 所对应的编码系数组成的方阵为 $\boldsymbol{A}$，若 S 满足 MDS 性质，则有以下结论。

1)　$\boldsymbol{A}$ 是满秩的。

2)　$\boldsymbol{A}$ 的行列式不为 0，即 $\det(\boldsymbol{A})\neq 0$。

3)　S' 可解码出原始数据。

证明　S 满足 MDS 性质，则其编码矩阵也满足 MDS 性质，即任意 $k(n-k)$ 个行向量是线性无关的，根据线性无关组的性质，这些向量组成的矩阵必然是满秩的，结论 1)得证。由于 $\boldsymbol{A}$ 是满秩矩阵，因此，其行列式的值不为 0，结论 2)得证。S 满足 MDS 性质，由定义 2 可知，任意 $k(n-k)$ 个编码块满足可译码性，结论 3)得证。

引理 2　Schwartz-Zippel 定理[17]。设 $h(x_1,\cdots,x_t)$ 是有限域 F 上的多变元 ρ 次非零多项式，S 是 F 的一个子集，则对 S 中随机选取的元素 $x_1,\cdots,x_t$，有 $\Pr[h(x_1,\cdots,x_t)=0]\leqslant\dfrac{\rho}{|S|}$。

证明　略，可参考文献[18]。

定理 1　数据持续可用性。假设一个文件存储在 $k=n-2$ 的拟态存储系统中，在第 r 轮对节点 j 的拟态变换中，从其他节点各取一个编码块，共计 $n-1$ 块编码块进行编码重新构造出新的编码块，那么通过增加有限域 $GF(q)$ 的大小 q，变换后的数据集总能满足 MDS 性质。

证明　使用归纳法证明。

初始化。首先使用 Reed-Solomon 码将原始数据编码为 $n(n-k)=2n$ 个编码块，满足 MDS 性质。

归纳假设。假设在第 r 轮变换后，MDS 性质满足，接下来，证明在第 $r+1$ 轮变换后，MDS 性质依然满足。

用 $U_r=\{P_{1,1},P_{1,2};\cdots;P_{k+2,1},P_{k+2,2}\}$ 表示第 r 轮变换后 n 个节点上的数据集，U_r 满足 MDS 性质。在第 $r+1$ 轮变换中，假设要对节点 1 实施变换，其他节点的情况与此同理。从节点 $2,\cdots,$ 节点 n 中任意选出 $n-1$ 个编码块，记作 $F=\{P_{2,f(2)},\cdots,P_{k+2,f(k+2)}\}$，其中 $f(*)\in\{1,2\}$。用 F 来对节点 1 进行变换，假设为节点 1 生成新的编码块为 $\{P'_{1,1},P'_{1,2}\}$，变换过程可表示为 $P'_{1,j}=\sum_{i=2}^{k+2}\gamma_{i,j}P_{i,f(i)}$，其中，$j=1,2$，$\gamma_{i,j}$ 来自有限域 $GF(q)$。

接下来，将证明通过调整域 $GF(q)$ 中 $\gamma_{i,j}$，总能够使在 $r+1$ 轮中变换后数据集 $U_{r+1}=\{P'_{1,1},P'_{1,2};\cdots;$

$P_{k+2,1}, P_{k+2,2}\}$ 依然满足 MDS 性质。

由于 U_r 满足 MDS 性质，所以只需要证明剩下的 $n-1$ 个节点 $\{2,\cdots,n\}$ 中任意 $k-1$ 个节点子集 $\{s_1, s_2, \cdots, s_{k-1}\}$ 和变换后节点 1 组成的数据集满足 MDS 性质即可。

$\{s_1, s_2, \cdots, s_{k-1}\}$ 共有 $\mathrm{C}_{n-1}^{k-1} = \dfrac{(n-1)(n-2)}{2}$ 种可能的组合，为了不失一般性，本文假设 $(s_1, \cdots, s_{k-1}) = (2, \cdots, k)$，其他情况的证明与此同理。

令 $V = \{P_{2,1}, P_{2,2}; \cdots; P_{k,1}, P_{k,2}; P'_{1,1}, P'_{1,2}\}$ 是节点 1 到 k 上的编码块组成的数据集，令 $R = \{P_{2,1}, P_{2,2}; \cdots; P_{k,1}, P_{k,2}; P_{k+1,f(k+1)}, P_{k+2,f(k+2)}\}$，于是有

$$\begin{bmatrix} P_{2,1} \\ P_{2,2} \\ \vdots \\ P_{k,1} \\ P_{k,2} \\ P'_{1,1} \\ P'_{1,2} \end{bmatrix} = \boldsymbol{A} \begin{bmatrix} P_{2,1} \\ P_{2,2} \\ \vdots \\ P_{k,1} \\ P_{k,2} \\ P_{k+1,f(k+1)} \\ P_{k+2,f(k+2)} \end{bmatrix} \tag{6}$$

其中，$\boldsymbol{A}$ 是一个 $k(n-k)\times k(n-k)$ 的编码矩阵，为

$$\boldsymbol{A} = \begin{bmatrix} 1,0, & \cdots, & 0,0, & 0,0 \\ 0,1, & \cdots, & 0,0, & 0,0 \\ \vdots & \ddots & \vdots & \vdots \\ 0,0, & \cdots, & 1,0, & 0,0 \\ 0,0, & \cdots, & 0,1, & 0,0 \\ \delta_{2,1}\gamma_{2,1}, \delta_{2,2}\gamma_{2,1}, & \cdots, & \delta_{k,1}\gamma_{k,1}, \delta_{k,2}\gamma_{k,1}, & \gamma_{k+1,1}, \gamma_{k+2,1} \\ \delta_{2,1}\gamma_{2,2}, \delta_{2,2}\gamma_{2,2}, & \cdots, & \delta_{k,1}\gamma_{k,2}, \delta_{k,2}\gamma_{k,2}, & \gamma_{k+1,2}, \gamma_{k+2,2} \end{bmatrix} \tag{7}$$

其中，δ 为区分每个节点上选择的是哪个编码块，当 $f(i)=1$ 时，$\delta_{i,1}=1$，$\delta_{i,2}=0$；当 $f(i)=2$ 时，$\delta_{i,1}=0$，$\delta_{i,2}=1$。

由于 U_r 满足 MDS 性质，R 是 U_r 的一个 $k(n-k)$ 子集，由引理 1 可知，R 可译码出原始文件。

根据引理 2，$\boldsymbol{A}$ 的行列式 $\det(\boldsymbol{A})$ 是以任意概率不为 0。由于 R 是可译码的，且 $\boldsymbol{A}$ 是满秩的，所以 V 是可译码的。

因此，U_{r+1} 满足 MDS 属性，证毕。综上所述，只要有限域的规模足够大，总能找到合适的编码参数使数据集始终满足 MDS 性质，从而确保数据在任何时刻都能够被正确译码，保证了数据的持续可用性。

6　仿真实验

本节对拟态变换的过程进行了仿真实验，验证了拟态变换机制的正确性和可行性，并根据仿真实验的结果，对其变换性能进行了估计和分析。本文使用 C++语言，借助于 Jerasure 库[19]提供的有限域操作接口以及 Eigen3 矩阵运算库提供的 API，设计和实现了拟态变换过程的仿真程序，仿真程序的流程如图 7 所示。仿真实验环境的配置为 Intel Core i3 @ 2.10 GHz 4 核处理器，8 GB 内存，Ubuntu12.04 LTS 64 位操作系统。

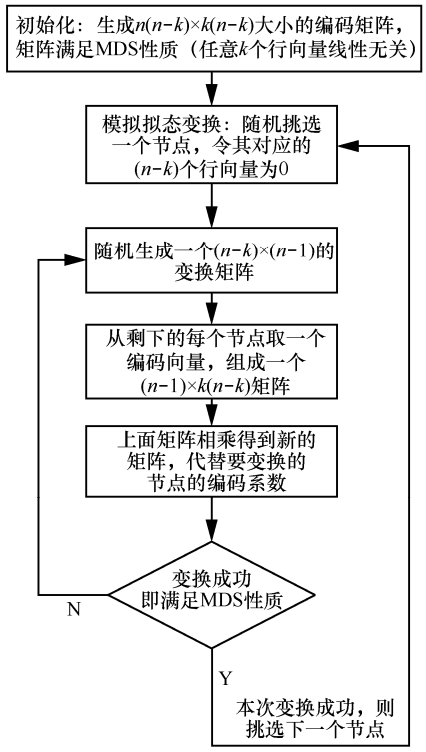

图 7　仿真程序流程

接下来，通过一个实例来说明具体的仿真过程。本文取 $n=4$、$k=2$，每个节点有 2 个数据块，所有的编码系数都是在 $GF(2^8)$ 上随机选取的。

1) 初始状态的编码矩阵为

67	c6	69	73
51	ff	4a	ec
29	cd	ba	ab
f2	fb	e3	46
7c	c2	54	f8
1b	e8	e7	8d
76	5a	2e	63
33	9f	c9	9a

2) 当对节点 3 进行变换时，其对应的编码系数标记为 0，对应的矩阵为

67　c6　69　73
51　ff　4a　ec
29　cd　ba　ab
f2　fb　e3　46
00　00　00　00
00　00　00　00
76　5a　2e　63
33　9f　c9　9a

3) 随机生成 $(n-k)\times(n-1)=2\times3$ 大小的矩阵为

32　0d　b7
31　58　a3

4) 从 $n-1=3$ 个节点分别随机选择一行系数为

67　c6　69　73
f2　fb　e3　46
33　9f　c9　9a

5) 上面 2 个矩阵相乘，得到新的节点系数矩阵为

4a　ee　9f　86

6) 用新节点系数矩阵的系数替换节点 3 的编码系数，检查新的编码矩阵是否满足 MDS 性质。

67　c6　69　73
51　ff　4a　ec
29　cd　ba　ab
f2　fb　e3　46
4a　ee　9f　86
e8　ed　97　13
76　5a　2e　63
33　9f　c9　9a

7) 若满足，拟态变换成功（本例中满足）。否则，跳到步骤 2)，进入下一次迭代。

本文通过统计变换成功所需要的迭代次数对拟态变换的性能进行估计，分别使用几组不同的编码参数和不同大小的有限域进行仿真实验，测量和评估编码参数及有限域大小对变换过程的影响。

6.1　编码参数对拟态变换性能的影响

取 $k=n-2$ 的存储设置，本实验分别选择了(4,2)、(6,4)、(8,6)这 3 个编码参数，在有限域 $GF(2^8)$ 上进行了仿真实验，分别统计了 10 000 次拟态变换操作中变换成功需要的迭代次数和平均时间，分别如表 1 和表 2 所示。

表 1　不同参数设置下拟态变换需迭代的次数

参数设置	迭代次数/次			迭代总次数
	1 次变换	2 次变换	3 次及其以上变换	
$n=4$、$k=2$	7 303	1 975	722	13 279 以上
$n=6$、$k=4$	5 303	1 523	3 174	17 868 以上
$n=8$、$k=6$	1 501	1 319	7 180	25 679 以上

表 2　不同参数设置下拟态变换的平均时间

参数设置	平均时间/ms		
	1 次变换	2 次变换	3 次及其以上变换
$n=4$、$k=2$	122	219	396
$n=6$、$k=4$	127	233	428
$n=8$、$k=6$	135	245	457

从实验结果可以看出，当 n 和 k 逐渐增大时，生成具有 MDS 性质的编码矩阵所需要的迭代总次数逐渐增加。当 n=4、k=2 时，测试 10 000 次，1 次变换成功为 7 303 次，2 次随机生成矩阵变换成功 1 975 次，3 次及其以上变换成功为 722 次；而当 n=8、k=6 时，10 000 次中 1 次变换成功为 1 501 次，2 次随机生成矩阵变换成功为 1 319 次，3 次及其以上变换成功为 7 180 次。生成可用的变换矩阵所用时间也随着参数 n 的变大而增加，但均不超过 1 s。

实验结果表明了随着数据节点数的增加，产生满足 MDS 性质的拟态变换参数将需要更多的迭代总次数和时间开销，由于该算法是一种启发式算法，难以完全保证多项式时间。但是在目前的情况下已经接近最优，算法的时间复杂度接近 $O(n^2)$，该算法原理简单、易于实现，适合分布式计算环境。由于拟态变换属于离散事件，拟态变换周期通常以小时计算，拟态变换参数构造算法只涉及小规模的矩阵计算，占用的资源较少，对客户端系统的影响可以忽略不计。

6.2　有限域大小对拟态变换性能的影响

本实验取 $n=4$、$k=2$，分别选取了 $GF(2^4)$、$GF(2^8)$、$GF(2^{16})$、$GF(2^{32})$ 4 个不同大小的有限域进行了仿真实验，分别统计了 10 000 次拟态变换操作中变换成功所需要的迭代次数和平均时间分别如表 3 和表 4 所示。可以看出，随着有限域大小的增加，1 次迭代成功的比例逐渐增加，当有限域的规模达到 2^{32} 时，1 次迭代的成功率达到了 99%以上。因此，通过选择较大的有限域可以有效地提高拟态变

换的效率，一般取 $GF(2^8)$ 就能够满足实际需求，时间开销随着有限域的大小大致呈线性增长。

表 3 不同大小有限域下拟态变换需迭代的次数

有限域	迭代次数/次		
	1 次变换	2 次变换	3 次及其以上变换
$GF(2^4)$	3 647	3 822	2 531
$GF(2^8)$	7 303	1 975	722
$GF(2^{16})$	9 080	875	45
$GF(2^{32})$	9 937	63	0

表 4 不同大小有限域下拟态变换需迭代的时间

有限域	迭代时间/ms		
	1 次变换	2 次变换	3 次及其以上变换
$GF(2^4)$	99	178	301
$GF(2^8)$	122	219	396
$GF(2^{16})$	148	250	423
$GF(2^{32})$	272	378	694

7 安全性分析

本节从以下 5 个方面对拟态变换机制的安全增益进行了深入的分析。

1) 冗余性增加了系统的可靠性，可保护数据的完整性和服务的连续性

基于网络编码的存储方案具有天然的冗余性，整个存储系统满足 MDS 性质，用户只需要访问部分数据节点，即可还原出原始数据。当出现失效节点时，可以通过其他节点的数据进行快速有效的恢复和重构，对于 (n,k) MDS 码，可以容忍 $n-k$ 个节点的失效，具有较好的容错能力。当攻击行为导致数据节点受损时，只要存储系统内可用的数据节点数保持在一个安全的阈值内，通过再生码修复方案可以快速有效地修复这些受损的节点，可避免数据的丢失。黑客的攻击行为通常会在系统内部产生许多污染数据，通过再生码技术可以快速地对这些被污染的数据节点进行清洗和修复，防止污染数据的进一步扩散。

冗余性能够避免节点失效造成的数据损失，保证了用户数据的完整性和可用性，可有效应对攻击者对系统的蓄意破坏和污染攻击行为，增强了系统的可靠性，确保了数据的安全性和服务的连续性。

2) 随机性使系统的攻击表面难以预测，增加了暴力破解编码矩阵的难度和成本

基于网络编码的拟态存储系统中的随机性主要体现在以下 2 个方面。

①数据的编码过程采用了随机线性网络编码，其编码参数是在有限域中随机选取的，仅要求编码矩阵满足 MDS 性质即可。利用随机算法来构造编码矩阵，可以防止攻击者对编码矩阵进行碰撞攻击，当有限域的规模足够大时，对编码参数的穷举攻击在现有的计算条件下几乎是不可能的。

②数据拟态变换过程中，当一个节点需要变换时，需要从其他的每个节点上随机地挑选一个编码块，增加了变换空间。由于每一次拟态变换过程中都加入了随机因素，使拟态变换过程具有很强的随机性，数据块的重构过程攻击者事先无法预测，事后无法重现，攻击者即使拿到了原始的编码矩阵，也无法构造出完全一致的拟态变换参数。

综上所述，随机性的引入使系统的攻击表面难以预测，极大地增加了攻击者暴力破解的难度和成本，有效地增强了系统的顽健性和安全性。

3) 时变性压缩了攻击者实施攻击的时间窗口

时变性主要体现在变换时机的选取上，在本文方案中，每个数据节点会维持一个定时器，变换周期初始化为一个随机的时间，随后会随着系统资源的使用情况和安全状况进行动态的调整。当定时器到时，就启动数据的变换过程，对当前节点上的数据进行重构。

拟态化存储机制将用户的文件切分成 $k(n-k)$ 份，经过编码生成 $n(n-k)$ 个编码块，分散地存储在 n 个数据节点上，从 n 个数据节点中任意的 k 个节点上下载数据，通过使用先前的编码矩阵进行译码即可还原出原始文件。对于攻击者要想拿到用户的原始文件，必须同时满足 2 个条件，一是集齐 $k(n-k)$ 个编码块，二是获得该文件的编码矩阵。如果破坏这个 2 个条件，攻击就难以奏效。

假设 t_0 时刻的数据块集合为 $data_0$，编码矩阵为 $\boldsymbol{EM}_0$，此时，攻击者只有同时拿到数据集 $data_0$ 的 $\frac{k}{n}$ 和 $\boldsymbol{EM}_0$，才能还原出原始文件，如果在攻击者得到这些信息之前，将这些信息进行变换，就能够阻断攻击过程，使之前针对 $data_0$ 和 $\boldsymbol{EM}_0$ 的攻击行为变得无效，增加了攻击者获取数据的成本和代价。

假设所有数据节点的拟态变换周期的最大值为 T，经过时间 T，所有的数据节点都至少经过了一轮拟态变换，设变换后新的数据集版本为 $data_1$，

相应地，客户端的编码矩阵也发生了变化，设变换后的编码矩阵为 $\boldsymbol{EM}_1$，攻击者如果之前已经获得 $data_0$ 中的少于 $k(n-k)$ 个编码块，t_0+T 时，由于所有的编码块至少被刷新过一次，那么攻击者获得的这部分数据就没有任何意义了，就算攻击者拿到先前的编码矩阵也无法还原出原始数据。所以只要设置合理的拟态变换周期就能够有效地阻断攻击链，保护数据安全。

综上所述，时变性改变了数据的静态存储状态，实现了云端数据的动态变化，压缩攻击者可利用的时间窗口，增加攻击者获取数据的难度和成本，提高系统的安全性。

4) 拟态变换机制可隐藏用户的操作信息和行为特征

在传统的静态存储系统中，用户和系统行为的确定性会给攻击者提供许多有用的信息，无意中暴露了系统的脆弱性，攻击者通过分析用户的行为特征和存储方式，寻找和利用系统防护的薄弱点实施渗透和攻击，进而窃取和破坏用户的数据。在拟态变换机制中，变换的时机和参数都是随机选取的，变换参数以加密的形式传输，可有效隐藏和保护用户的操作信息，干扰攻击者的信息探测手段，可有效防止社会工程学攻击。

5) 拟态变换机制可减小系统的攻击面

拟态变换机制使存储系统具有动态可变属性，可隐藏系统的脆弱性，降低存储系统漏洞暴露在网络中的概率，通过减小系统的攻击面来保护系统安全。

综上所述，拟态变换机制可阻断攻击链，增加攻击者实施攻击的难度和成本，有效保护存储系统的安全性。

8　系统实现与性能分析

本节设计和实现了一个拟态存储验证系统，该系统采用了分层的松耦合系统结构，实现了上文提出的拟态变换机制，并在真实的网络和存储环境下，进行了相关的功能验证和性能测试。

8.1　系统设计与实现

本方案的系统架构如图 8 所示，整个存储系统可分为客户端系统和数据节点 2 部分。客户端系统负责用户文件的存取，与数据节点进行交互，主要分成 6 个功能层次，其中，文件系统层、编码层和存储层是必需的，而访问控制层、加密层和拟态变换层是可选的，对其他层是透明的。数据节点分布于网络中，通过安全信道与客户端相连。数据节点支持不同类型的存储设备和存储系统，本节讨论的数据节点是广义的数据节点，既可以是普通的 PC 机、服务器，也可以是数据中心、云服务提供商（如百度云、DropBox）等大型存储设施。

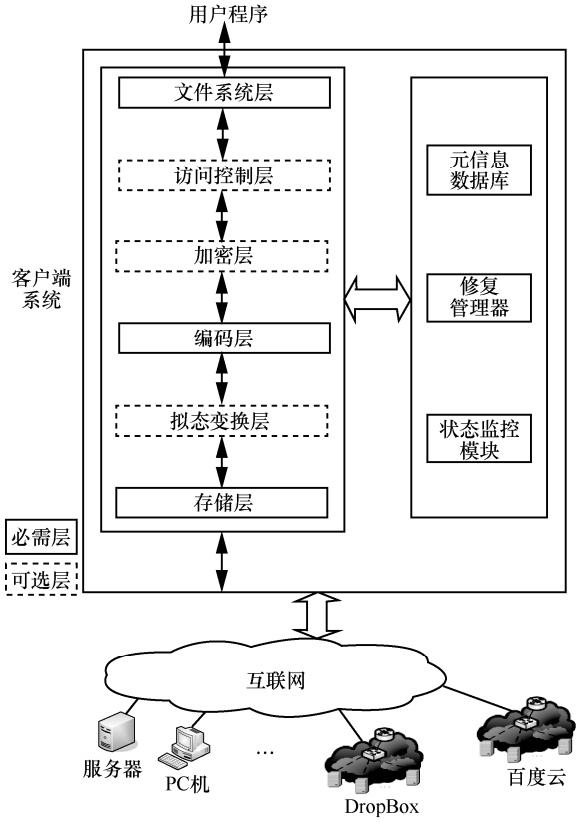

图 8　系统的整体架构

客户端系统采用基于代理的结构，用户通过在终端运行代理程序与数据节点进行交互。代理程序对上向用户和应用程序提供标准统一的数据访问接口，屏蔽下层的实现细节；对下兼容各种不同类型的存储系统和存储介质，能够适应不同的存储和网络设置，可根据不同的业务和安全性需求，添加和调整功能构件。客户端系统采用松耦合的分层设计，具有良好的可扩展性，每个功能层次相互独立，各层之间约定了数据交换格式和调用接口，各层的功能模块可根据用户需求进行动态重构，每一层可以根据需要动态替换和增加其他的功能和组件，而不影响其他层的功能和系统整体的逻辑结构。

1) 文件系统层

文件系统层负责与用户和上层应用进行交互，对上提供标准的数据访问接口（包括读、写、删除和更新等操作），处理由用户发起的文件操作请求，

还负责文件元数据的创建和维护。文件系统层建立在开源的应用层文件系统框架 FUSE 之上，实现了标准的文件操作接口。它支持以逻辑驱动器的形式挂载在主机的文件系统下，这样既可以简化设计，又允许应用程序像访问本地文件一样透明地访问云端数据，而不需要关心底层的实现细节。

2) 访问控制层

访问控制层负责文件内容的访问控制，根据用户要求和文件属性定制和执行不同的访问控制策略，并将这些策略存储在数据库中。通过审查用户的文件读取请求，来验证用户的身份，对照用户文件预设的访问权限执行访问控制，可防止数据的非授权访问，保护用户数据安全。

3) 加密层

加密层负责对数据进行加解密操作，对下行数据进行加密处理，对上行数据进行解密处理，该层功能相对独立，对文件系统层和编码层完全透明。加密层包括加密器、解密器和密钥管理模块 3 部分，支持不同类型的加密方案。为了提高加密的效率，一般采用对称加密算法，如 AES128、AES256、3DES、RC4 等。

4) 编码层

编码层负责对数据进行切分和编码，包括编码器、解码器、修复模块、编码管理和配置模块。编码层针对基本的编码和译码操作，设计了相应的编码和译码原语，向用户开放了编程接口，用户可以根据需要实现自定义的编码方案，也可建立自己的编码库；编码管理和配置模块用来读取编码参数和使用伪随机数发生器而产生编码矩阵。编码层各个模块之间相互独立，通过规范的接口相连，具有较好的扩展性。

5) 拟态变换层

拟态变换层负责为各个数据节点计算拟态变换参数。该层是一个可选的层，只参与数据的拟态变换操作，在文件的上传和下载过程中，拟态变换层被跳过，不执行任何操作。拟态变换层中运行拟态变换协议，建立和维持客户端主机与数据节点间的安全链路，响应拟态变换请求，产生和分发拟态变换参数。

当客户端系统收到某个数据节点的拟态变换请求时，具体的处理过程如下。

Step1 拟态变换层从拟态变换请求包中解析出目标数据节点的标识和其上数据块的序列号范围。

Step2 拟态变换层从本地元信息数据库中读取编码矩阵，运行拟态变换算法（详细可参考 4.4 节），为其计算拟态变换参数。拟态变换参数包括该目标节点需要下载的数据块列表和拟态变换矩阵。

Step3 用户端将拟态变换参数发送给目标数据节点，数据节点进行拟态变换。

6) 存储层

存储层负责数据块的存取访问，将编码数据块分发给网络上的数据节点进行存储，可根据编码层的数据块访问请求，从云端数据节点上读取相应的编码块，对下可兼容不同类型的存储介质和存储系统。存储层对下层的存储介质进行了抽象，屏蔽了下层存储系统的实现细节，对上提供统一的访问接口。

存储层为上层提供访问不同的存储节点的通用接口，负责数据块粒度的读写访问。存储层会根据编码层所使用的编码方式，综合考虑下层不同的数据节点受损概率、I/O 性能来确定相应的放置策略，并根据数据节点的运行状况进行实时的调整。

7) 数据节点

数据节点可以是普通的 PC 机、廉价的商用服务器、NAS 设备，也可以是云服务提供商或大型的数据仓库等。数据节点需要具备一定的存储和计算能力，每个数据节点上需要运行一个守护进程，负责采集系统运行信息，与客户端程序配合完成文件的写入、读取、更新操作和拟态变换等操作。

本文使用 C++语言实现一个拟态存储实验系统。其中，文件系统层是基于开源的应用层文件系统框架 FUSE 来实现，提供了最基本的文件读写操作接口，在此基础上实现了拟态变换功能。访问控制层使用访问控制列表（ACL, access control list）来实现强制访问控制，ACL 存储在共享数据库中。加密层使用开源的加密库 Crypto++ 5.6.5 提供的 API 接口，Crypto++库支持多种不同密钥长度的对称加密算法，如 DES、3DES、AES128、AES256 等，满足对不同安全等级文件的加密需求。编码层和拟态变换层使用 Jersure 库提供的 Galois 域操作接口和矩阵运算库 Eigen3，实现了 Reed-Solomon 和 FMSR 这 2 种编码方案。客户端与云端存储系统间使用 OpenSSL 建立安全信道。数据节点采用了普通的服务器，每个数据节点上运行一个守护进程，专门用于数据的读取、写入和拟态变换。

8.2 性能测试与分析

本文利用局域网环境和 OpenStack 来模拟真实

的云存储环境，部署了拟态存储系统。实验拓扑和相关配置如图 9 所示，客户机运行客户端程序，6 个存储节点由 OpenStack 虚拟而来，每个节点上运行一个守护进程，负责用户数据的读取、写入和拟态变换。存储节点和客户机通过局域网连接，为了简化客户端的存储层，存储节点通过网络文件系统（NFS）挂载在客户机目录下，这样客户机可以通过访问本地目录来访问存储节点。

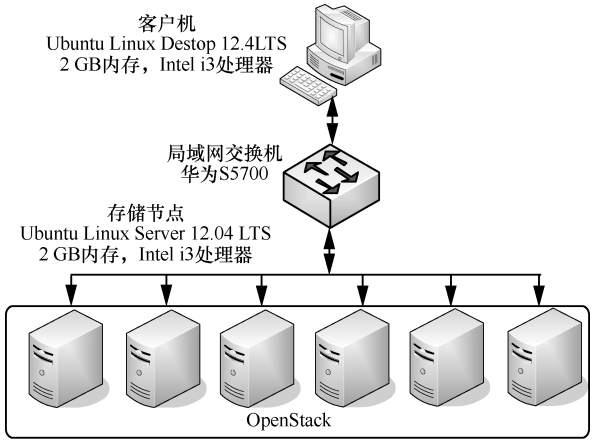

图 9　实验拓扑与配置

本节对拟态存储机制进行功能验证和性能测试，主要从文件的上传、下载、拟态变换 3 个方面进行实验。实验采用的编码方案是(6,4)FMSR 码，采用的有限域为 $GF(2^8)$。

8.2.1　文件的上传

本文分别选取不同大小的文件上传到数据节点上，测量上传过程所需要的时间，实验结果如图 10 所示。文件的上传过程需要经过 4 个阶段：文件的加密、数据切分编码、数据的传输、数据节点接收并写入数据。从图 10 中可以看出，文件上传的大部分时间都消耗在网络传输上面，而加密和编码环节所用的时间相对比较少。随着文件大小的增加，数据上传所用的时间大致呈线性增长趋势，说明存储机制具有良好的稳定性和适应性。

8.2.2　文件的下载

本节将前面上传到数据节点上的文件分别下载到本地，测量下载过程所需要的时间，结果如图 11 所示。文件的下载过程分为 4 个子过程：数据读取、数据传输、数据译码、数据块合并和解密。从图 11 中可以看出，数据的读取和传输时间占总下载时间的主要部分，数据译码和解密时间相对较少，下载所用时间与文件大小大致呈线性关系。

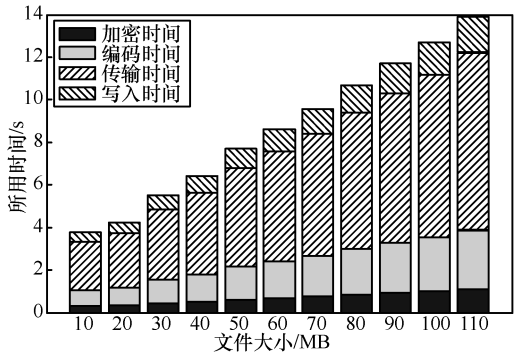

图 10　文件大小与上传时间关系

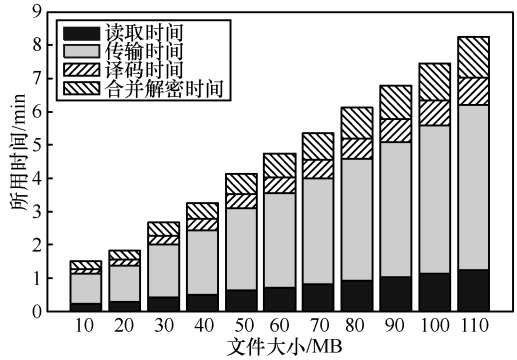

图 11　文件大小与下载时间关系

8.2.3　拟态变换

本节任选一个数据节点，对于不同大小的文件进行拟态变换，测量一次拟态变换所需的时间如图 12 所示。拟态变换过程主要分为 4 部分：客户端计算变换参数、节点端下载编码块、数据重构和节点端写入编码块。从图 12 中可以看出，随着文件大小的增加，拟态变换的时间呈线性增长。数据的拟态变换操作的时间开销主要集中于文件的网络传输和 I/O 操作上，变换参数的计算和编码过程占用的时间较少，说明拟态变换算法的计算开销较小，不会对系统性能造成明显的影响。

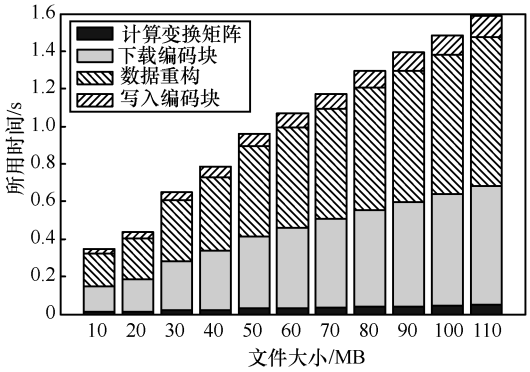

图 12　文件大小与拟态变换时间关系

8.2.4 数据的完整性和可用性验证

文件上传时计算其散列值，对其进行若干次的拟态变换后，将其下载到本地，还原出原始文件，再计算其散列值，比较 2 个散列值是否一致，以此来检验数据的完整性。使用 SHA-1 算法来计算文件的散列值，对 100 个不同的文件，每个数据节点进行 1 000 次拟态变换，对变换前后的散列值进行比对，实验结果表明，文件的完整性并未遭到破坏，数据依然保持持续可用性。

9 结束语

本文针对存储系统因静态存储模式而易被攻击的问题，提出了一种基于再生码的拟态存储机制，通过对数据进行编码存储，并在云端进行拟态变换，增加了攻击者获得数据的难度和成本；对拟态变换问题进行了建模与分析，给出了拟态变换过程中数据完整性和持续可用性的形式化证明；对拟态变换机制进行了仿真实验，验证了拟态变换的正确性和有效性，分析了不同参数和有限域大小对变换性能的影响；对拟态化存储机制的拟态特征进行归纳，并分析了其安全性；设计和实现了一个拟态存储实验系统，并进行了性能测试和分析，实验结果表明，拟态变换方案切实可行，能够保证数据的持续可用性和完整性。

参考文献：

[1] PALANKAR M R, IAMNITCHI A, RIPEANU M, et al. Amazon S3 for science grids: a viable solution?[C]// The 2008 International Workshop on Data-aware Distributed Computing. 2008: 55-64.

[2] CALDER B, WANG J, OGUS A, et al. Windows azure storage: a highly available cloud storage service with strong consistency[C]// The Twenty-Third ACM Symposium on Operating Systems Principles. 2011: 143-157.

[3] MCKUSICK K, QUINLAN S. GFS: evolution on fast-forward[J]. Communications of the ACM, 2010, 53(3):42-49.

[4] SHVACHKO K, KUANG H, RADIA S, et al. The Hadoop distributed file system[C]//MASS Storage Systems and Technologies. 2010:1-10.

[5] 罗象宏, 舒继武. 存储系统中的纠删码研究综述[J]. 计算机研究与发展, 2012, 49(1):1-11.
LUO X H, SHU J W. Summary of research for erasure code in storage system[J]. Journal of Computer Research and Development, 2012, 49(1):1-11.

[6] 邬江兴. 网络空间拟态安全防御[J]. 保密科学技术, 2014(10):4-10.
WU J X. Cyber minic security defense[J]. Secrecy Science and Technology, 2014(10):4-10.

[7] 邬江兴, 张帆, 罗兴国, 等. 拟态计算与拟态安全防御[J]. 计算机学会通讯, 2015, 11(1): 8-14.
WU J X, ZHANG F, LUO X G, et al. The meaning and vision of minic computing and minic security defense[J].Communications of CCF, 2015, 11(1): 8-14.

[8] 邬江兴. 网络空间拟态防御研究[J]. 信息安全学报, 2016, 1(4):1-10.
WU J X. Research on cyber mimic defense[J]. Journal of Cyber Security, 2016, 1(4):1-10.

[9] 罗兴国, 仝青, 张铮,等. 拟态防御技术[J]. 中国工程科学, 2016, 18(6):69-73.
LUO X G, TONG Q, ZHANG Z, et al. Mimic defense technology[J].Engineering Sciences,2016,18(6):69-73.

[10] 斯雪明, 王伟, 曾俊杰,等. 拟态防御基础理论研究综述[J]. 中国工程科学, 2016, 18(6):62-68.
SI X M, WANG W, ZENG J J, et al. A review of the basic theory of mimic defense[J]. Engineering Sciences, 2016, 18(6):62-68.

[11] 庞建民, 张宇嘉, 张铮,等. 拟态防御技术结合软件多样化在软件安全产业中的应用[J]. 中国工程科学, 2016, 18(6):74-78.
PANG J M, ZHANG Y J, ZHANG Z,et al. Applying a combination of mimic defense and software diversity in the software security industry[J]. Engineering Sciences,2016, 18(6):74-78.

[12] 邬江兴. 专题导读—拟态计算与拟态安全防御的原意和愿景[J]. 电信科学, 2014, 30(7):1-7.
WU J X. Meaning and vision of mimic computing and mimic security defense[J]. Telecommunications Science, 2014, 30(7):1-7.

[13] 吴春明. 动态网络主动安全防御的若干思考[J]. 中兴通讯技术, 2016(1):34-37.
WU C M. Proactive security defense of dynamic network[J].ZTE Technology Journal, 2016(1):34-37.

[14] 仝青,张铮,张为华,等.拟态防御 Web 服务器设计与实现[J]. 软件学报,2017,28(4):883-897.
TONG Q, ZHANG Z, ZHANG W H, et al. Design and implementation of mimic defense Web server[J]. Journal of Software, 2017,28(4): 883-897.

[15] 张铮, 马博林, 邬江兴. Web 服务器拟态防御原理验证系统测试与分析[J]. 信息安全学报, 2017, 2(1):13-28.
ZHANG Z, MA B L, WU J X. The test and analysis of prototype of mimic defense in web servers[J]. Journal of Cyber Security, 2017, 2(1):13-28.

[16] HU Y, LEE P P C, SHUM K W. Analysis and construction of functional regenerating codes with uncoded repair for distributed storage systems[C]// IEEE INFOCOM. 2012:2355-2363.

[17] TRACEY H, MURIEL M. On randomized network coding[C]//The Annual Allerton Conference on Communication Control and Computing. 2003(3):11-20.

[18] MOSHKOVITZ D. An alternative proof of the schwartz-zippel lemma[J]. Electronic Colloquium on Computational Complexity, 2010, 17.

[19] PLANK J S. Jerasure: a libraryin C/C++ facilitating erasure coding for storage applications-version 1.2, UT-CS-08-627[R]. Department of Computer Science, University of Tennessee, 2008.

[作者简介]

陈越（1965-），男，河南开封人，博士，解放军信息工程大学教授、博士生导师，主要研究方向为网络与信息安全。

王龙江（1991-），男，陕西商洛人，解放军信息工程大学硕士生，解放军 61660 部队助理工程师，主要研究方向为网络信息安全、云存储安全等。

严新成（1991-），男，河南信阳人，解放军信息工程大学博士生，主要研究方向为网络信息安全、云存储安全、云数据访问控制。

张馨月（1994-），女，满族，吉林通化人，解放军信息工程大学硕士生，主要研究方向为云数据访问控制。