**毕业设计（论文）**

题目：**安全的加密文件去重存储系统开发**

**（英文）:** Secure encrypted file deduplication storage system development

学 院 计算机科学与教育软件学院

专 业 计算机科学与技术

班 级 计机117

学 号 1106102005

学生姓名 杨滨

指导教师 李进 教授

安全的加密文件去重存储系统开发

**摘要** 随着云计算技术的发展，云存储服务给予了我们社会活动的极大便利。因为数据量的大幅度增加，重复数据删除技术成为了大规模存储系统的核心技术要点，因为它是通过去除相似数据来提供系统的存储量；而数据加密之后的密文值是接近随机分布的，那么找到两个相同密文的概率就会很小，这个去重技术带来了很不便的因素，所以很多云存储服务商是以用户提交的数据来做去重处理，那么这些数据是裸露在云存储服务商的系统中。尽管去重处理后的数据将会加密存储，但用户已经不是真正意义上的唯一拥有者了，这就造成了数据的低保密性。本文将提供一种密钥管理和加密机制来实现加密文件的去重存储：在用户端进行对文件的对称加密，而密钥归用户管理，之后以哈希值作为密文标示符进行重复数据删除，密文存储在数据库模拟上传到云存储系统。最终达到在保证数据的高机密性的同时，提高重复数据删除率来保证数据存储空间拥有高利用率。

**关键词** 对称加密,哈希计算,去重,存储

Secure encrypted file deduplication storage system development

**ABSTRACT** With the development of cloud computing, cloud storage service has given us great convenience in social activities. Because a substantial increase in the amount of data deduplication technology has become the core techniques of large-scale storage system, as it is to provide the system's storage capacity by removing similar data; and data encryption ciphertext value after the close of randomly distributed , then the probability of finding two identical ciphertext will be very small, this deduplication technology has brought a very inconvenient factors, so a lot of cloud storage service providers based on data submitted by the user to re-do the deal, then these data are bare Cloud storage service providers in the system. Data will be stored encrypted Despite heavy to handle, but the user is not the sole owner of the real sense, and which resulted in low data confidentiality. This article will provide a key management and encryption mechanisms to achieve deduplication store encrypted files: the client files were symmetric encryption, and key management owned by the user, and then to the hash value as the ciphertext identifiers repeated data delete, upload ciphertext stored in the database to the cloud storage system simulation. Ultimately ensuring high confidentiality of data while improving deduplication ratio to ensure that the data storage space with high efficiency.

**Keywords** Symmetric encryption,Hashing,Deduplicat,Storage

目录

[1. 前 言 1](#_Toc418934839)

[1.1开发背景： 1](#_Toc418934840)

[1.2开发意义： 2](#_Toc418934841)

[2. 系统介绍 3](#_Toc418934842)

[2.1本系统的开发目的： 3](#_Toc418934843)

[2.1.1可行性分析及需求分析： 3](#_Toc418934844)

[2.2系统流程图及开发技术： 4](#_Toc418934845)

[2.2.1 MVC框架介绍： 4](#_Toc418934846)

[2.2.2 OpenSSL介绍及安装编译: 5](#_Toc418934847)

[2.2.3密码散列算法之SHA-1算法的介绍： 6](#_Toc418934848)

[2.2.4 加密算法介绍： 9](#_Toc418934849)

[2.2.5去重技术介绍： 18](#_Toc418934850)

[2.2.6 Qt4界面库开发介绍及系统界面介绍： 20](#_Toc418934851)

[2.2.7 Sql Server数据库技术的应用 23](#_Toc418934852)

[3.系统测试与分析 25](#_Toc418934853)

[3.1 测试方法： 25](#_Toc418934854)

[3.2 测试环境： 25](#_Toc418934855)

[3.3 测试用例： 26](#_Toc418934856)

[4.总结 29](#_Toc418934857)

[参考文献 30](#_Toc418934858)

[致谢 33](#_Toc418934859)

# 前 言

## 1.1开发背景：

在网络快速发展的这几年来，互联网思维被运用在我们的生活中，悄无声息地影响着我们的各种生活习惯，这当中发展最快也具有最大市场的云计算无疑是最大的焦点。各大互联网的巨头，还有权威的学术研究院，甚至连各国政府都参与到云计算的领域中来。

而对于云计算的研究，理论上还要基于美国标准与技术研究院（National Institute of Standards and Technology，NIST）对云计算做出的这样一个定义：云计算它是一个模型，而这个模型对用户提供一个网络接入，以此来实现用户快捷地按需接入一个可配置的计算资源共享池（如、网络、服务器、存储、应用和服务），这个资源的共享池能够以最小的管理代价、或者以最少的服务提供商交互来达到快速的提供跟释放。即使目前的云计算相关的学术研究还相当的欠缺，但是辅以完善理论的最好方式就是实践，而实际上各大企业都纷纷开始提供云计算的商业应用，而此来抢占云计算的先机。总所周知的最早提供云计算服务的是Amazon，而目前较完善的云计算服务器体系也当属Amazon。然而云计算的发展是迅速的，目前我国阿里巴巴集团创办的阿里云也已经开始了与Amazon的正面对抗。可以说，云计算就是当前互联网发展的热点。

目前来说，对云计算的商业运用最多的是云存储。本质上，云存储本身就是在云计算的概念上延伸和发展而来的一个全新的概念，是指通过集群应用、网络技术或分布式文件系统等功能，将网络中各种不同类型的存储设备通过应用软件集合起来协同工作，共同对外提供数据存储和业务访问功能的一个系统。当云计算系统运算和处理的核心是大量数据的存储和管理时，云计算系统中就需要配置大量的存储设备，那么云计算系统就转变成为一个云存储系统，所以云存储是一个以数据存储和管理为核心的云计算系统。

## 1.2开发意义：

通过对现有的云存储系统中的数据进行分析，不同用户间的数据重复现象以及同一个用户内部的数据重复现象都是比较常见的。同时，网络技术的高速发展进一步增加了不同用户之间分享数据的可能性。至此，对存储数据做去除重复处理是至关重要的，使存储系统中的相同数据标识出来，仅保存其中一个副本，这就大大降低了需要存储的数据总量，提供了系统的存储空间的利用率。同时，如果当用户准备要将数据上传到云存储系统时，可以在用户端先利用重复数据删除技术判断该数据是否重复，只有当云存储系统中没有该数据内容时候才会执行实际的数据上传过程，这种方式将大大地节约了用户上传操作的时间开销和占用的网络带宽。基于此，高效的数据去除重复技术是各大云存储服务商竞争云存储市场的一大利器。

目前，在云存储模式下，数据托管存储于第三方云存储平台中，脱离了数据属主（Data Owner）的控制范围，其安全性高度依赖于云服务提供商。就像当前全球知名的云存储服务商（DropBox），该公司对用户的数据进行的加密处理后，其加密密钥便掌握在该公司手中，只要其密钥被泄露后，用户的数据同样会被泄露。那么有种解决方式便是在用户上传数据之前对数据进行加密处理，由用户持有加密数据的密钥，而上传到云存储系统的只是数据的密文格式。

由上述对云存储的前景，及对云存储当前的缺点进行分析可知，在重复数据删除的存储系统中如何保证数据保密性是一个非常重要的问题。

# 系统介绍

## 2.1本系统的开发目的：

### 2.1.1可行性分析及需求分析：

随着技术的快速发展，加密技术日趋成熟，加密算法的高效及可靠性致使加密数据被破解难上加难。目前情况分析得知，在全球应用程序中，相关的加密技术大部分都使用了目前运用最广泛的开源项目Openssl，其项目因为自身有着成熟的加密库而备受青睐。本系统也将会使用该库的3DES对称加密及sha1哈希计算，这不仅大大减小了本系统开发的难度，还提高了开发的可行性及可靠性。

因为主要是验证加密与去重的可行性，所以云存储方面并不会使用市面上的云服务提供商所提供的存储服务。本系统通过使用 Mircrosoft SQL Server 2008 R2数据库来模拟云存储的方式，使我们的数据上传到数据库来达到存储的效果。数据的去重算法可使用SQL的匹配查询做模拟，当数据库里有重复的密文时，就不上传。实际上这里主要解决的一个问题就是如何保证同样内容的密文是一致的。

当对文件进行Sha1计算时，该算法是根据文件内容做hash值计算的，即当我们的文件内容一样时，所得到的hash值时一样的。那么接下来，我们便使用该hash值作为文件的加密密钥key对文件进行对称加密。因为是对称加密，所以同样内容使用相同一样的加密密钥key来加密时，加密后的密文也将会是一致的。

## 2.2系统流程图及开发技术：

需加密文件

Hash值

使用hash值为key对文件进行3DES加密

得到文件密文

计算密文Hash值

将Hash值与系统已存储数据做去重处理

图2-2-1系统流程图

### 2.2.1 MVC框架介绍：

本系统至始至终以MVC框架做为开发的底层，一个清晰的开发框架有利于后期的对系统的扩展开发以及让系统保持高耦合性。

MVC全名是Model View Controller，是模型（model）-视图（view）-控制器（controller）的缩写，一种软件设计典范，用一种业务逻辑、数据、界面显示分离的方式组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑。MVC被独特的发展起来用于映射传统的输入、处理和输出功能在一个逻辑的图形化界面的结构中。

MVC框架的细节描述以及在本系统编写过程中的主要运用在于以下三个方面：

M是指与数据库做交互，如获取数据库的内容、查找数据库进行条件匹配以及将密文插入数据库。

V是指将数据库记录显示在界面。

C是指用户与本系统的交互，如选择文件加密以及对相应密文做解密。

### 2.2.2 OpenSSL介绍及安装编译:

#### 2.2.2.1 OpenSSL的简述：

目前，OpenSSL是较为开发者欢迎的免费的可用密码库。

1. 对称算法：

OpenSSL实现了一套标准的对称算法，这其中包括Blowfish、三重DES、RC4和RC5

1. 散列算法：

OpenSSL实现了一系列常见的散列函数，这其中包括SHA-1和MD5，因为新的SHA算法（SHA-256、SHA-512）还善未实现，而SHA-1又满足我们的HASH计算即去重运用，本系统将使用SHA-1来进行开发。

1. 消息验证码（MAC）：

OpenSSL为其支持的所有散列算法都实现了HMAC，即实现以一个密钥和一个消息作为输入，生成一个消息摘要来作为本次运算的输出。

1. 公共密钥算法：

OpenSSL实现了RSA、Diffie-Hellman和DSA签名算法

1. 质量和效率

鉴于OpenSSL函数的易调用跟开发环境的允许，使用OpenSSL进行本系统开发并没有多大的问题出现，以及效率并没有显著的差别。

#### 2,2.2.2 windows 7下安装编译OpenSSl：

1. 下载openssl源代码。
2. 下载安装perl，注意配置环境变量。
3. 运行Visual Studio 命令提示。
4. 使用cd 进入到openssl源代码的所在目录
5. 运行 perl Configure VC-WIN32-prefix=C：/opnessl\_lib ，prefix参数表示为生成库所存放的位置
6. 运行 “ms\do\_ms”
7. 运行”nmake-f ms\ntdll.mak”,执行make进行编译。该命令是将openssl编译成动态库，动态库即表示：在程序运行时才会去调用库内的函数。如果想要编译成静态库的话应该使用命令”nmake –f ms\nt.mak”，静态库即表示：将函数库内容嵌入到程序中。
8. 运行”nmake –f ms\ntdll.mak test”,检查上一步的编译是否成功完成。
9. 运行”nmake –f ms\ntdll.mak install”，该步骤表示将编译后的openssl安装到我们设定的C:/openssl\_lib目录下。
10. 查看C:/openssl\_lib目录下是否有bin\include\lib三个文件夹。
11. 编译安装完成。

### 2.2.3密码散列算法之SHA-1算法的介绍：

Hash算法也叫散列算法，散列算法是单向函数，美其名也就是说，函数接受一个明文字符串，然后将其转换成一个不能恢复的密文片段。

目前在技术应用方面比较常用的Hash算法有MD5（Message Digest 5）和SHA-1，相对与MD5来说，SHA-1算法比较成熟，且具有更高的安全性以及在编程方面的易实现性。实际上，很多文摘不建议因此，本系统使用了SHA-1算法来对文件进行hash计算，并以此为key来对文件进行加密，该算法同样运用于本系统的去重处理，关于去重处理将在下文讲述。

SHA-1是SHA的一种。SHA是安全哈希算法(Secure Hash Algorithm)的英文缩写，主要适用于数字签名标准(Digital Signature Standard DSS)里面定义的数字签名算法(Digital Signature Algorithm DSA)。对于长度小于2^64位的消息，SHA-1会产生一个160位的消息摘要，因此SHA-1发生碰撞的概率为1/2^160。碰撞性即是：两个不同内容的输入串进行hash计算后的输出值一样的概率。

当接收到消息的时候，这个消息摘要可以用来验证数据的完整性。在传输的过程中，数据很可能会发生变化，那么这时候就会产生不同的消息摘要。SHA-1有如下特性：不可从消息摘要中恢复原来的信息；而且两个不同内容的消息产生同样的HASH值的概率极低，这也是SHA-1的完全性。对此，本系统使用该哈希算法来计算文件的加密密钥。相同文件内容进行SHA1计算后得到的消息摘要将会是一致的，这为接下来的对称加密提供了加密密钥。

SHA-1算法的运算步骤是，首先对信息报文填补信息位的长度：设*M*是原始的报文消息，即要将它转换为*M*1，如下：

|*M*1|| = 448 mod 512

If | *M| =* 448 mod 512 then | *M*1| = | *M*| + 512

而填补的内容是：100…00。这个内容将填补在末尾的64bit，这就要求原始的报文长度不可以超过2^64。

通常情况下，Hash函数的输入的长度都要大于输出的长度，这是一个压缩的过程。当处理512bit的报文分组序列时，分为四组。核心包含4个循环的模块，而这里的每个循环均由20个处理步骤组成。尽管这4个循环有着相似的地方，但每个循环所使用的逻辑函数是不同的。在循环中，SHA-1的压缩函数为：

A = (E +f(t,B,C,D) + S5(A) + Wt + Kt)

B = A,C = S30(B),D = C,E = D

函数其中的A,B,C,D,E表示为5个缓存器;t表示为步数，0 ≤ t ≤ 79;这里的f（t,B,C,D）是所执行t步的基本逻辑函数;Sk表示为循环左移k个位置给定的32位字;Wt表示为从当前的512个数据块中导出的32位字;Kt表示为用于加法的常量,这有4个不同的值。

#### 2.2.3.1本系统的sha-1核心代码：

#include "Sha1\_Func.h"

#include <cstdio>

#include <cstdlib>

#include <cassert>

#define File\_Buffer\_Size 1024

int shaFile(const char\* fileName,char \*des)

{

assert(fileName != NULL);

unsigned char buffIn[File\_Buffer\_Size];

unsigned char hash[SHA\_DIGEST\_LENGTH];

FILE\* fpIn = fopen(fileName,"rb");

if(fpIn == NULL)

{

printf("The %s can't be Sha1 processing\n",fileName);

return 0;

}

size\_t nRead;

SHA\_CTX ctx;

SHA1\_Init(&ctx);

while(1)

{

nRead = fread(buffIn,1,File\_Buffer\_Size,fpIn);

if(nRead)

{

SHA1\_Update(&ctx,buffIn,nRead);//Multiple Execution

}

else

break;

}

SHA1\_Final(hash,&ctx);//To get the final result and put it into hash

fclose(fpIn);

int i=0,j=0;

for(;i < SHA\_DIGEST\_LENGTH;j += 2, ++i)

{

sprintf(des + j,"%02x",hash[i]);

}

return 1;

}

### 2.2.4 加密算法介绍：

本系统使用了对称加密算法，其早先最主要用于保护数据的机密性，并且用于保护数据的完整性。

由于其速度快，对称性加密通常在消息发送方需要加密大量数据时使用。对称性加密也称为密钥加密。

所谓对称，就是采用这种加密方法的双方使用方式用同样的密钥进行加密和解密。密钥是控制加密及解密过程的指令。算法是一组规则，规定如何进行加密和解密。

因此加密的安全性不仅取决于加密算法本身，密钥管理的安全性更是重要。因为加密和解密都使用同一个密钥，如何把密钥安全地传递到解密者手上就成了必须要解决的问题。

本系统主要解决的问题是，当前很多云存储服务均是用户将所要存储的数据上传到云存储服务商，这样既使用户所存储的数据并不为用户独自拥有，如数据的主动权也上交给了云存储服务商。比如当前最受用户所欢迎的云存储服务商DropBox，尽管用户存储在其系统中的数据是加密的，但该密文的密钥并不为用户所持有，即DropBox也有权查看到用户的数据。

为此，本系统所要实现的是在客户端进行加密，而加密的密钥即是对文件进行SHA-1计算得到的Hash值，该密钥将由用户自行保管，这样也即保证了用户是数据的唯一拥有者，除非密钥被泄露。

对称加密算法的概况如下（图2-2-2 对称加密）：

加密算法

图2-2-2 对称加密

2.2.4.1 对称算法的安全性：

目前开发应用中，最常用的对称算法便是数据加密标准(Data Encryption Standard,DES)。目前，三重加密的DES（即3DES）是一种流行的现代对称分组算法，它是在DES的基础上进行三次加密，很大程度上保证了其保密性。然而，3DES也有一个问题就是速度，这问题并不影响本系统开发，所以对比其他算法之后，本系统选择使用3DES进行开发。

#### 2.2.4.2 Openssl库的3DES算法实现的细节：

本系统在实际开发中主要使用的还是openssl的EVP\_Encrypt支持的对称加密算法。

openssl对称加密算法的格式都以函数形式提供，其实该函数返回一个该算法的结构体，其形式一般如下：

**EVP\_CIPHER\*  EVP\_\*(void)**

返回的结构体类型为EVP\_CIPHER。

EVP\_Cipher\*...\*得以实现的一个基本结构是下面定义的一个算法结构，它定义了EVP\_Cipher系列函数应该采用什么算法进行数据处理，其定义如下（evp.h）：

typedef struct evp\_cipher\_st

 {

int nid;

int block\_size;

int key\_len;

int iv\_len;

unsigned long flags;

int (\*init)(EVP\_CIPHER\_CTX \*ctx, const unsigned char \*key, const unsigned char \*iv, int enc);

int (\*do\_cipher)(EVP\_CIPHER\_CTX \*ctx, unsigned char \*out, const unsigned char \*in, unsigned int inl);

int (\*cleanup)(EVP\_CIPHER\_CTX \*);

int ctx\_size;

int (\*set\_asn1\_parameters)(EVP\_CIPHER\_CTX \*, ASN1\_TYPE \*);

int (\*get\_asn1\_parameters)(EVP\_CIPHER\_CTX \*, ASN1\_TYPE \*);

int (\*ctrl)(EVP\_CIPHER\_CTX \*, int type, int arg, void \*ptr); /\* Miscellaneous operations \*/

void \*app\_data;

 }**EVP\_CIPHER**;

下面对这个结构的部分成员的含义作一些解释：

nid——是算法类型的nid识别号，openssl里面每个对象都有一个内部唯一的识别ID

block\_size——是每次加密的数据块的长度，以字节为单位

key\_len——各种不同算法缺省的密钥长度

iv\_len——初始化向量的长度

init——算法结构初始化函数，可以设置为加密模式还是解密模式

do\_cipher——进行数据加密或解密的函数

cleanup——释放EVP\_CIPHER\_CTX结构里面的数据和设置。

ctx\_size——设定ctx->cipher\_data数据的长度

set\_asn1\_parameters——在EVP\_CIPHER\_CTX结构中通过参数设置一个ASN1\_TYPE

get\_asn1\_parameters——从一个ASN1\_TYPE中取得参数

ctrl——其它各种操作函数

app\_data——应用数据

通过定义这样一个指向这个结构的指针，你就可以在连接程序的时候只连接自己使用的算法；而如果你是通过一个整数来指明应该使用什么算法的话，会导致所有算法的代码都被连接到代码中。通过这样一个结构，还可以自己增加新的算法。

在这个基础上，每个EVP\_Cipher\*...\*函数都维护着一个指向一个EVP\_CIPHER\_CTX结构的指针。

 typedef struct evp\_cipher\_ctx\_st

{

const EVP\_CIPHER \*cipher;

ENGINE \*engine;

int encrypt;

int buf\_len;

unsigned char oiv[EVP\_MAX\_IV\_LENGTH];

unsigned char iv[EVP\_MAX\_IV\_LENGTH];

unsigned char buf[EVP\_MAX\_BLOCK\_LENGTH];

int num;

void \*app\_data;

int key\_len;

unsigned long flags;

void \*cipher\_data;

int final\_used;

int block\_mask;

unsigned char final[EVP\_MAX\_BLOCK\_LENGTH];

} **EVP\_CIPHER\_CTX**;

下面对这个结构部分成员做简单的解释：

cipher——是该结构相关的一个EVP\_CIPHER算法结构

engine——如果加密算法是ENGINE提供的，那么该成员保存了相关的函数接口

encrypt——加密或解密的标志

buf\_len——该结构缓冲区里面当前的数据长度

oiv——初始的初始化向量

iv——工作时候使用的初始化向量

buf——保存下来的部分需要数据

num——在cfb/ofb模式的时候指定块长度

app\_data——应用程序要处理数据

key\_len——密钥长度，算法不一样长度也不一样

cipher\_data——加密后的数据

上述两个结构是EVP\_Cipher(EVP\_Encrypt)系列的两个基本结构，它们的其它一些列函数都是以这两个结构为基础实现了。文件evp/evp\_enc.c是最高层的封装实现，各种加密的算法的封装在p\_enc.c里面实现，解密算法的封装在p\_dec.c里面实现，而各个e\_\*.c文件则是真正实现了各种算法的加解密功能，当然它们其实也是一些封装函数，真正的算法实现在各个算法同名目录里面的文件实现。

在openssl中，所有提供的对称加密算法长度都是固定的，有特别说明的除外。下面对这些算法进行分类的介绍，首先介绍一下算法中使用的通用标志的含义。

#### 2.2.4.3加密流程介绍：

结束

开始

初始化key和iv

初始化密码算法结构体，设置密码算法、密钥、初始化向量

调用加密结束函数，输出最后的密文

循环调用加密update，直到处理完所有的待加密数据

一般来说，EVP\_Encrypt\*...\*系列函数的应用架构如下所描述(假设加密算法为3DES):

1. **定义一些必须的变量**

char key[EVP\_MAX\_KEY\_LENGTH];     //密钥，EVP\_MAX\_KEY\_LENGTH = 64

 char iv[EVP\_MAX\_IV\_LENGTH];   //初始化向量，EVP\_MAX\_IV\_LENGTH  = 16

EVP\_CIPHER\_CTX ctx; //evp算法上下文

 unsigned char out[512+8]; //输出密文缓冲区

1. **给变量key和iv赋值**

EVP\_BytesToKey(EVP\_des\_ede3\_cbc,EVP\_md5,NULL,passwd,strlen(passwd),key,iv)

该函数从自动产生了我们所需要的密钥key和初始化向量值iv。该函数的调用不是必须的，因为我们也能自己设定key跟iv值。在本系统中，我们是通过从文件的Hash摘要中提取24节作为加密的key，而iv的8个字节只是使用了简单的1-8.

1. **初始加密算法结构EVP\_CIPHER\_CTX**

 EVP\_EncryptInit\_ex(&ctx, EVP\_des\_ede3\_cbc(), NULL, key, iv);

该函数对密码算法结构体ctx进行初始化，而这里使用的是3DES的算法。

1. **进行数据的加密操作。**

 while (....)

 {

EVP\_EncryptUpdate(ctx,out,&outl,in,512)

}

对于不定长的数据或者大数据的话，一般来说都采用循环的方式来对其进行相应处理，每次循环加密数据均为512字节，密文输出到out，out和int应该是指向不相同的内存的，避免覆盖，而本系统使用不同的文件来存储。

1. **结束加密，输出最后的一段512字节的数据。**

EVP\_EncryptFinal\_ex(&ctx, out, &outl)

该函数会进行加密的检测，如果加密过程有误，一般会检查出来。

说明：解密跟上述过程是一样的，只不过要使用EVP\_Decrypt\*...\*系列函数。

#### 2.2.4.4加密核心代码：

#include "Encrypt\_Func.h"

#include "Sha1\_Func.h"

#include <cstdio>

#include <cstdlib>

#include <cstring>

#include "openssl/evp.h"

#include "openssl/x509.h"

#include <openssl/sha.h>

#include <cassert>

#define File\_Buffer\_Size 1024

int Encrypt\_File(const char \*EncryptF,char \*CipherF,char store\_key[])

{

unsigned char out[File\_Buffer\_Size];

int outl;

unsigned char in[File\_Buffer\_Size];

int inl;

int rv;

FILE \*fpIn;

FILE \*fpOut;

unsigned char key[EVP\_MAX\_KEY\_LENGTH];

unsigned char iv[EVP\_MAX\_KEY\_LENGTH];

assert(EncryptF != NULL);

printf("The %s will be saved to be encrypted\n",EncryptF);

fpIn = fopen(EncryptF,"rb");

if(fpIn == NULL)

{

printf("fault:can't open the %s\n",EncryptF);

return 0;

}

fpOut = fopen(CipherF,"wt");

printf("The ciphertext will be saved in %s\n",CipherF);

if(fpOut == NULL)

{

fclose(fpIn);

printf("Fault:save the ciphertext files\n");

return 0;

}

char\* des = (char \*)malloc(SHA\_DIGEST\_LENGTH\*2 +1);

if( !shaFile(EncryptF,des) )

{

printf("Fault:can't Obtain the encryption-key\n");

return 0;

}

else

{

int i=0;

for(;i < 24;++i)

{

key[i] = des[i]; // encrypt key

store\_key[i] = des[i];

}

for(i = 0;i < 8;++i)

iv[i] = i;

}

free(des); //free memory

EVP\_CIPHER\_CTX ctx;

EVP\_CIPHER\_CTX\_init(&ctx);

rv = EVP\_EncryptInit\_ex(&ctx,EVP\_des\_ede3\_cbc(),NULL,key,iv);

if(rv != 1)

{

printf("fault:cannot initialize ctx");

return 0;

}

for(;;)

{

inl = fread(in,1,1024,fpIn);

if(inl <= 0)

break;

rv = EVP\_EncryptUpdate(&ctx,out,&outl,in,inl);

if(rv != 1)

{

fclose(fpIn);

fclose(fpOut);

EVP\_CIPHER\_CTX\_cleanup(&ctx);

printf("fault:EncryptUpdate");

return 0;

}

fwrite(out,1,outl,fpOut);

}

rv = EVP\_EncryptFinal\_ex(&ctx,out,&outl);

if(rv != 1)

{

fclose(fpIn);

fclose(fpOut);

EVP\_CIPHER\_CTX\_cleanup(&ctx);

printf("fault:EncryptFinal");

return 0;

}

fwrite(out,1,outl,fpOut);

fclose(fpIn);

fclose(fpOut);

EVP\_CIPHER\_CTX\_cleanup(&ctx);

printf("Encrypt success\n");

return 1;

}

### 2.2.5去重技术介绍：

De-duplication,即重复数据删除，它是当前存储技术中主流且是非常热门的应用，它可通过对数据集中重复的数据进行处理，而只保留其中一份，从而把冗余的数据删除而达到存储容量的提高。这种技术因为其在很大程度上减少了物理存储空间的使用，从而满足当前数据大爆发的存储需求。

本系统将采用的数据去重算法为SHA-1（160位的散列）。

#### 2.2.5.1 “数据去重”的几种方法：

1. 文件级的去重

通过在文件系统中检查是否两个文件完全相同，来实现文件级的去重。如果相同，即其中的一个文件会被指向另一个文件的指针所替换。这项技术并不会影响到数据的读取性能。因为它只是将对多份数据的访问请求指向同一份数据，所以它的效率较低。

1. 数据块级的去重

该方式是将所有的文件分解成较小的数据块，然后再通过散列算法，为每块数据创建一个哈希值，再与其他数据块的哈希值做匹配，如果已存在相同哈希值的数据块，那么系统将用指向已存在数据块的指针来代替。

1. 元数据级的去重

元数据级去重实现的基本原理：基于对相同文件内又或者是阵列空间的数据对象的元数据对象进行相关算法处理，并且将使用其结果作为算法的列表码或者是算法的KEY环，来比对写入流的元数据对象，如果比对法相相同的元数据，那么将以相应的指针代替后删除相同元数据。

#### 2.2.5.2 系统所使用的去重方法

因为加密后的密文只有128位，所以将使用文件级的去重方式，只需对密文进行一次SHA-1哈希值的计算而得到20字节的哈希值后，我们再进行系统内哈希值的匹配，如果有相同的哈希值存在，即以指向已存在哈希值的指针代替，从而达到删除重复数据，提高系统存储容量的目的。

关于SHA-1算法的技术在上文已经介绍，在这里不做累述。

### 2.2.6 Qt4界面库开发介绍及系统界面介绍：

软件的界面相当于人的整体外表，完整的软件界面设计可以让软件更加生动，可使使用者操作便捷，易上手。本系统将采用Qt4的界面库来进行GUI的开发。

#### 2.2.6.1 Qt4开发流程及主要技术介绍：

本次开发使用Qt4库进行GUI设计及开发，GUI的设计借助了Qt旗下的一款软件：Qt Designer，而相应的模块开发将在visual studio 2010进行。在visual studio 2010调用Qt4库需要安装Qt by Digia v4.8.6 (VS2010 OpenSource)以及Qt Visual Studio Add-in。

在visual studio 2010开发Qt软件的主要流程如下：

* 1. 新建Qt项目

安装好Qt by Digia v4.8.6 (VS2010 OpenSource)以及Qt Visual Studio Add-in后，我们重启vs2010，然后选择文件新建项目，将出现（图2-6-1 新建Qt项目），选择Qt Application后输入项目名称后点击确定。

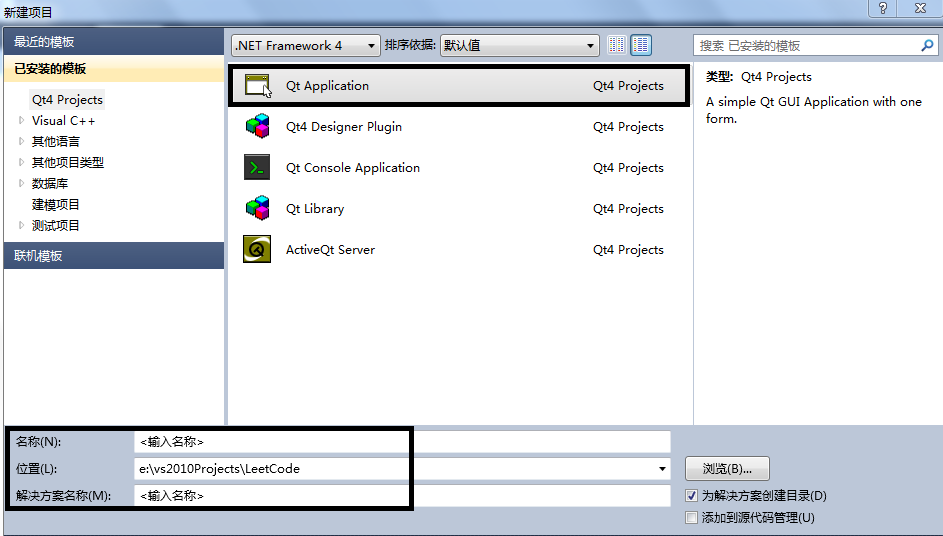


图2-6-1 新建Qt项目

* 1. 选择GUI开发所需要库

点击确定后，将会出现Qt4 GUI Project Wizard的配置框（图2-6-2 选择开发包），选择Project Settings中我们所需要的包后（本系统需要使用到SQL，所以选择了SQL library），点击Finish，即完成了我们项目的初步配置。

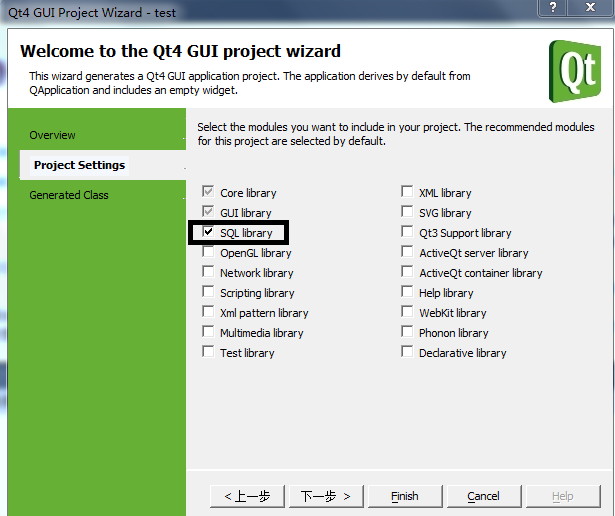


图2-6-2 选择开发包

* 1. 本系统各目录略解

本系统的代码目录（图2-6-3 系统目录）中，后缀为ui的file\_encrypt\_store.ui即是使用Qt designer 设计的GUI界面，之后通过实现界面类来将其显示。文件EncryptAndStore存放的是系统开发所使用的加密及Hash算法库。因为本系统使用了MVC框架，所以Control存放的便是C层文件，View存放的是V层的文件，Sql及Source File文件提供函数接口，也为M层。

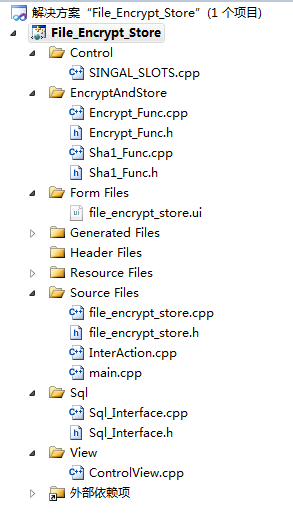


图2-6-3 系统目录

* 1. Qt开发核心技术之信号槽机制

Qt框架最核心的技术就是引入了信号槽机制。当理解和熟练掌握信号槽时，才能够设计出解耦的非常漂亮的程序，这有利于增强我们的技术设计能力。

而所谓信号槽机制，实际上就是观察者模式。即当某个事件发生之后，比如，按钮检测到自己被点击了一下，它就会发出一个信号（signal）。这种发出是没有目的的，类似广播。而如果有对象对这个信号感兴趣，它就会使用连接（connect）函数，意思是，用自己的一个函数（成为槽（slot））来处理这个信号。也就是说，当信号发出时，被连接的槽函数会自动被回调。这就类似观察者模式：当发生了感兴趣的事件，某一个操作就会被自动触发。

#### 2.2.6.2当系统启动时主界面以及各组件的介绍：

打开系统后，直接显示系统的操作界面图（图2-6-1 为系统界面）。

浏览按钮：选择要加密的文件，文本文件、图像文件等。

加密文件显示框：显示加密文件的路径。

密钥输入框：输入密钥名，输入完后系统自动匹配该密钥名是否已经存在。

加密按钮：开始加密后，将密文的hash值存入数据库。

密钥列表：显示密钥存储库所存储的密钥即对应的内容。

密文列表：显示密文的hash值即对应的密钥。

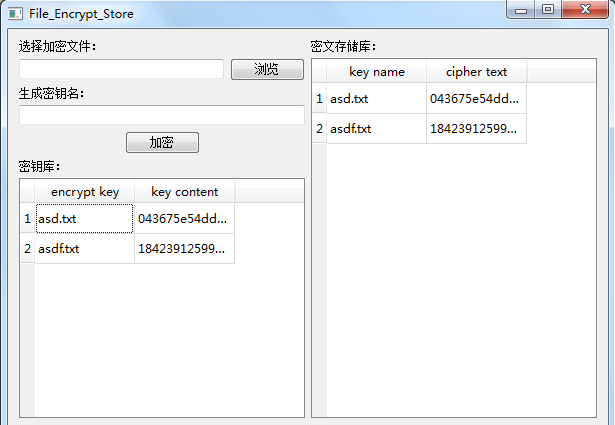


图2-6-1 系统界面

### 2.2.7 Sql Server数据库技术的应用

本系统使用Sql Server 2008 R2来存储用户密钥及模拟云存储的数据存储过程，而Qt框架为我们提供了QtSql模块，即Qt4 GUI Project Wizard配置时，所选择的SQL library。当使用了Qt所提供的SQL模块时，Qt提供了QSqlDatabase类，我们可通过调用其addDatabase（）函数来建立一个QSqlDatabase的实例，之后再进行相应配置。

以下为配置一次数据库连接时所使用的函数接口。Db\_name为此次连接的名字，我们可以通过名字来进行相对于某次连接的操作。

void MySql::configure\_database( QString db\_name)

{

db = &QSqlDatabase::addDatabase("QODBC",db\_name);

db->setHostName("127.0.0.1");

db->setDatabaseName("DRIVER={SQL SERVER};SERVER=127.0.0.1;DATABASE=" + db\_name);

db->setUserName("sa");

db->setPassword("123456");

}

各函数讲解：

SetHostName()函数表示我们所连接的数据库位置，此次为本机地址（因为使用了本地的Sql Server）。

SetDatabaseName()函数表示使用了SQL SERVER的驱动，以及此次数据库连接的名称。

SetUserName()函数设置数据库的用户名。

SetPassword()函数设置数据库用户名的密码。

# 3.系统测试与分析

在系统发布之前，我们需要对系统进行一系列的测试来确保系统能够正常的运行，减少Bug的出现。以下将分三部分：测试方法、环境以及用例来做本章内容。

## 3.1 测试方法：

当我们完成一个系统的基本开发时，还需要一个重要的收尾步骤：软件测试。软件测试是对系统质量进行保证的一种手段：通过在系统交付之前，对系统编码进行测试以及对功能需求的审核。通过模拟客户可能出现的情况，在我们不断设定的环境中对系统进行测试，是发现系统错误的有力手段。

站在客户的角度，客户希望通过软件测试这一环节来测试出系统中的错误、缺陷以及用户体验，从而决定是否接受此系统。站在开发者的角度，则是希望通过测试来证明自己所开发的软件产品并没有存在Bug，从而确定该产品的规格已经符合客户的需求，以此来确立客户对开发者开发工作的信心。软件测试是软件制作周期中不可缺少的重要一环，是保证此产品不易被市场淘汰，这也保证了产品的可维护性以及可用性。

目前，对程序的测试大致可分为两种：第一种方法可称为黑盒测试，通过对系统的每个功能进行测试，看软件是否能正常运行。第二种方法称为白盒测试，这是通过对程序内部结构编写测试用例来测试代码的一种方法，该方法检测程序中的每条通道是否进行着预定要求的工作。

本系统将主要采用黑盒测试来做说明。

## 3.2 测试环境：

处理器：Intel(R) Core(TM)i3-2330M CPU @ 2.20GHz 2.20GHz

硬盘：500G

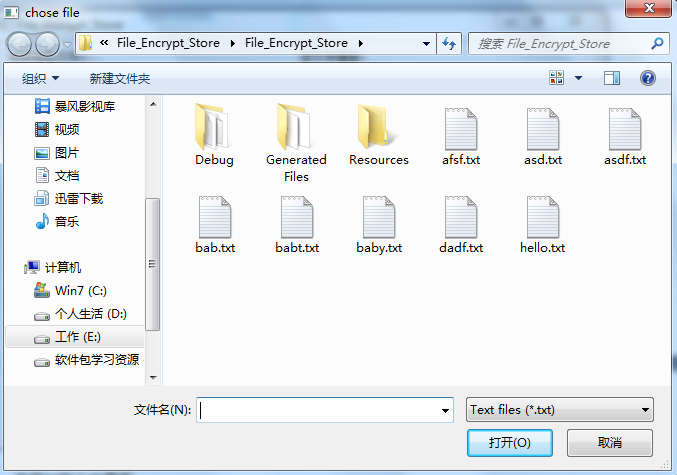
操作系统：Windows 7

数据库：Sql Server 2008 R2

## 3.3 测试用例：

(1)选择加密文件界面

通过点击主界面的浏览按钮，进入到选择加密文件的界面（图3-3-1 选择文件界面），默认在程序本目录打开。



（图 3-3-1 选择文件界面）

（2）输入密钥名及检测密钥名是否重复

在生成密钥名的提示文本下方为密钥名的输入框，可通过自定义密钥名来方便用户的管理，当密钥名已在密钥管理库存在时，系统将会提示（图 3-3-2 密钥已存在提示框）

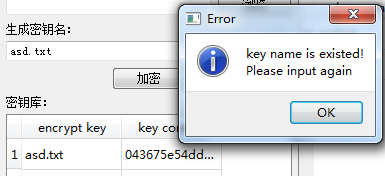


图3-3-2 密钥已存在提示框

（3）加密文件及密钥名不可为空

当未选择加密文件而点击加密按钮时，系统提示用户输入加密文件（图3-3-3 加密文件不能为空提示框）

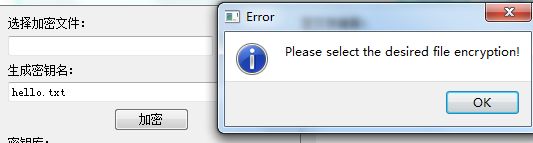


图3-3-3 加密文件不能为空提示框

当未写入密钥而点击加密按钮时，系统提示用户输入密钥名（图3-3-4 密钥不可为空提示框）。

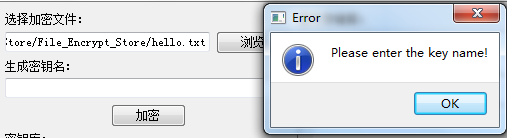


图3-3-4 密钥不可为空提示框

(4) 加密存储成功

点击加密按钮，加密存储成功时，将提示加密存储成功，并同时刷新密钥库以及密文存储库列表（图3-3-5 密文存储成功界面）



图3-3-5 密文存储成功界面

（5）去重处理

当密文的去重操作表示密文已经存在时，系统对用户进行提示（图3-3-5 密文已存在提示框）

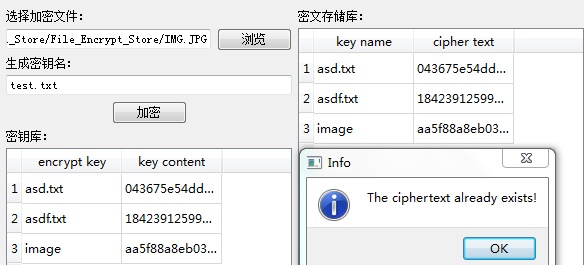


图3-3-5 密文已存在提示框

当将与图片IMG\_0636同样内容但不同名字的图片IMG存入系统时，系统提示密文已经存在，即达到了只要内容重复就不进行存储的效果，从而减少了数据对内存的占有，提高了系统的容量。

# 4.总结

本系统是针对于目前软件开发行业最热门的两个热点：云计算跟安全存储进行相关的学习跟开发。通过对前沿技术的了解及发掘，有利于我自己在软件开发这个新技术不断涌现的行业中站稳脚步。本系统通过对称加密进行密钥管理，再通过哈希计算来进行数据去重处理，在此过程中，我不仅掌握了安全加密的技术，甚至了解到了当前云存储服务商所运用的技术，还有界面开发的流程以及对Sql Server的实质运用。从实用性来说，系统还没能达到市场投放的程度，但是对相应理论的验证以及应用可说明，在保证用户数据保密性的同时也能达到提高系统存储效率的作用。

加密文件去重存储系统的开发，是我自己在广州大学学习的这两年里所学到知识的一个灵活运用，也是一把打开我社会工作之门的钥匙。从一开始收集资料，已经对各个模块，各个组件的学习与实现，都是对我发现问题、思考问题、解决问题等能力的一个质的提升，可以提高我从零到有开发一个基础系统的能力，这些都是我在本系统开发过程中所积累到的宝贵财富，这也为我今后的软件开发工作积累了宝贵的经验。

# 参考文献

[1] BOBBARJUNG D R, JAGANNATHAN S, DUBNICKI C. Improving Duplicate Elimination in Storage Systems [J]. ACM Transactions on Storage, 2006, 2(4):424-448.  
[2] GUNAWI H S, AGRAWAL N, ARPACI-DUSSEAN A C, et al. Deconstructing Commodity Storage Clusters [C]//Proceedings of the 32nd International Symposium on Computer Architecture (ISCA’05), Jun 4-8, 2005, Madison, WI, USA. Los Alamitos, CA, USA: IEEE Computer Society, 2005: 60-71.  
[3] BOLOSKY W J, CORBIN S, GOEBEL D, et al. Single Instance Storage in Windows 2000 [C]//Proceedings of the 4th USENIX Conference on File and Storage Technologies(FAST’05),Dec 13-16,2005, San Francisco, CA,USA. Berkeley, CA, USA: USENIX Association, 2005:13-24.  
[4] QUINLAN S, DORWARD S. Venti: A New Approach to Archival Storage [C]//Proceedings of the 2nd USENIX Conference on File and Storage Technologies(FAST’02), Jan 28-30,2002, Monterey, CA,USA. Berkeley, CA, USA: USENIX Association, 2002:89-101.  
[5] POLICRONIADES C, PRATT I. Alternatives for Detecting Redundancy in Storage Systems Data [C]//Proceedings of the 2004 USENIX Annual Technical Conference (USENIX’04), Jun 27-Jul 2, 2004, Boston, MA,USA. Berkeley, CA, USA: USENIX Association, 2004:73-86.

[6] [美]维格(Viega,j.),麦格劳(McGraw,G.).安全软件开发之道：构筑软件安全的本质方法[M].北京 机械工业出版社. 2014年

[7] YOU L, POLLACK K, LONG D. Deep Store: An Archival Storage System Architecture [C]//Proceedings of the 21st IEEE International Conference on Data Engineering(ICDE’05), Apr 5-8, 2005, Tokyo, Japan. Los Alamitos, CA, USA: IEEE Computer Society, 2005: 804-815.  
[8] BRODER A Z. Identifying and Filtering Near-duplicate Documents [C]//Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching(CPM’00), Jun 21-23, 2000, Montreal, Canada. Berlin, Germany: Springer-Verlag, 2000:1-10.  
[9] DOUGLIS F, IYENGAR A. Application-specific Delta encoding Via Resemblance Detection [C]//Proceedings of the 2003 USENIX Annual Technical Conference (USENIX’03), Jun 9-14, 2003, San Antonio, TX,USA. Berkeley, CA, USA: USENIX Association, 2003:113-126.  
[10] BRODER A Z, MITZENMACHER M. Network Applications of Bloom Filters: A Survey [J]. Internet Mathematics, 2003, 1(4):485-509.  
[11] ZHU B, LI K, PATTERSON H. Avoiding the Disk Bottleneck in the Data Domain Deduplication File System [C]//Proceedings of the 6th USENIX Conference on File and Storage Technologies(FAST’08), Feb 26-29, 2008, San Jose, CA, USA. Berkeley, CA, USA: USENIX Association, 2008: 1-14.  
[12] YANG T, JIANG H, FENG D, et al. DEBAR: A Scalable High-performance De-duplication Storage System for Backup and Archiving [R]. TR-UNL-CSE-2009-0004.St Lincoln, NE, USA: University of Nebraska-Lincoln.2009.  
[13] BHAGWAT D, ESHGHI K, LONG D E, et al. Extreme Binning: Scalable, Parallel Deduplication for Chunk-based File Backup [C]//Proceedings of the 17th IEEE International on Modeling, Analysis & Simulation of Computer and Telecommunication Systems(MASCOTS’09), Sep 21-23,2009, London, UK. Piscataway, NJ, USA: IEEE, 2009.

[14] [美] Mark Allen Weiss.数据结构与算法分析：c++描述[M].北京.人民邮电出版社出版发行.2007年.

[15] 姜斌,吕秋云.信息安全与应用编程实验教程[M].杭州.浙江大学出版社.2014年.

# 致谢

作为即将毕业的大学生，尽管来到广州大学只有短短的两年，而我也非常怀念这两年来在大学校园里愉快的学习时光，在这里有辛勤为我们教授知识的老师和对我们生活无比关怀的学校领导。在此，我想特别说上一句祝福，感谢你们对我们的栽培，感谢你们在了解到我们插本生基础比较薄弱时，特别有针对性的讲授知识，甚至一改以往的教学风格，可以说是对我们尽心尽责，呕心沥血。在老师们的教授下，我们不仅吸收了知识，还成为了能为梦想付诸行动的实干派。

在这毕业论文完成的尾声，我要特别感谢我的指导老师李进教授对我的悉心指导，使我能够如此顺利地完成了本次的系统开发及毕业设计。因为李进教授的研究方向是信息安全，所以老师在我的开发过程中，给了我许多的权威帮助跟教导。在撰写论文的过程中，李进教授多次询问了我的研究进度，并为我指点迷津，帮助了我开拓思路去研究及攻克开发过程中所遇到的问题，这给予了我莫大的鼓励。再次表达我对老师的由衷感谢。

在此向广州大学，计算机科学与软件教育学院的所有老师与领导致以诚挚的谢意！