В статье дается исчерпывающее описание подхода автора к построению прототипа распределённых защищенных информа¬ционных систем для работы с конфиденциальной информацией SQL сервера, используя открытые каналы связи типа Интернет. Детально рассмотрены вопросы построения клиентских сессий, аутентификации и авторизации пользователей, ограничения доступа, сериализации типов, компрессии и шифрования данных, защиты клиентского приложения от несанкционированной модификации, управления производительностью

Основные цели построения прототипа (типовой многоуровневой модели)- научиться:

строить много проектные решения (много файловые сборки) в среде Visual Studio/C# (приложения, состоящие из одного .exe и нескольких .dll);

строить программные системы на основе нескольких программных приложений, взаимодействующих как в среде отдельного компьютера, так и в рамках локальной сети и Internet;

удаленно работать с конфиденциальной информацией SQL сервера, используя в качестве транспорта данных Интернет или незащищенный сегмент локальной сети, защищать клиентское приложение от несанкционированной модификации, а также, определить и осуществить реализацию минимального состава необходимых методов хеширования, сериализации типов, компрессии и шифрования передаваемой информации. В процессе построения прототипа решены следующие задачи:

аутентификации пользователя, инициирующего обмен и создание клиентской сессии;

авторизации пользователя в процессе работы;

ограничения доступа к функциям и записям таблиц базы данных информационной системы (управление доступом);

защиты трафика;

страничного представления клиенту информации первого плана и разовой подкачки недостающей информации второго плана;

компрессии информации;

защиты клиентского приложения от несанкционированной модификации;

адаптации структуры систем на базе прототипа к информационной нагрузке,управления модификациями кешированной информацией,

а также:

осуществлена реализация необходимых программных методов;

построено много проектное решение для пользовательского интерфейса,

построена шести уровневая программная система, состоящая из пяти взаимодействующих на основе .Net Remoting приложений (клиентское приложение, сервер взаимодействия с клиентом, сервер хранения запросов/ответов, КриптоСервер, сервер хранения модифицирующих предложений) и базы данных на SQL сервере,разработан и реализован общий подход к настройке .NET- сервисов и Windows приложений.

Особенности прототипа:

клиентское приложение прототипа свободно распространяется в Интернете, функционирует в незащищённой среде и защищено от несанкционированной модификации;

прототип не использует для клиентов учётные записи локальной сети Windows;

для построения и функционирования всех клиентских сессий достаточно одной ключевой пары, т.к. прототип - система с одним, требующим защиты, центром данных. Открытый ключ асимметричного алгоритма "зашивается" непосредственно в клиентское приложение.

Фундаментальная цель построения прототипа - разработка базовых элементов технологии и инструментария, необходимых для создания многоуровневых защищенных информационных систем на основе взаимодействующих .Net Remoting сервисов. В дальнейшем, опираясь на прототип, разработчик может создавать распределённую вычислительную среду для решения конкретной информационной задачи и адаптировать её структуру к информационным нагрузкам, распределяя функциональность по сетевым компьютерам. Принципиальный момент - в данной типовой модели аппаратно-программная структура вычислительной среды и расположение в ней .Net Remoting сервисов определяется на этапе проектирования. Связь одного сетевого сервиса к другим осуществляется по адресам URL формата (например: tcp://alpha:5000/ или http://localhost:80/). Поэтому можно разрабатывать и отлаживать все программные элементы системы, используя только один компьютер. И создать на основе этих элементов реальную систему такую, например, как показано на вышеприведенном рисунке.

Прототип содержит три базовых компонента (схемотехнических элемента, "кубика") - КриптоСервер (функциональность сервера бизнес-логики), Remote .Net сервер хранения запросов/ ответов на базе консольного приложения (функции управления системой динамических абонентских ящиков) и сервер управления взаимодействием с удаленными клиентами (функции подключения множества внешних клиентов к информационной системе). В качестве связующих стрелок используются адреса URL. К одному базовому схемотехническому элементу - серверу хранения запросов/ ответов - можно с одной стороны подключить несколько IIS элементов с Remote .Net сервисами, а с другой несколько серверов бизнес-логики, и получить вычислительный узел. На стадии проектирования системы, меняя число и состав её вычислительных узлов (при необходимости и динамически), решаем задачу адаптации структуры системы на базе прототипа к информационной нагрузке (масштабирование). Иными словами, собираем требуемую программно-техническую структуру информационной системы из "кубиков", комбинируя и распределяя их функциональные возможности.

Задачу "горячего" upgrate серверов бизнес-логики решаем так, для каждого сервера узла, последовательно - останов, upgrate и запуск.

Программный комплекс из двух элементов - сервера управления взаимодействием с удаленными клиентами и Remote .Net сервера хранения запросов/ответов - серьёзный инструмент для создания распределенных вычислительных сред. В абонентский ящик можно положить, как запрос клиента на выполнение работы сервером, так и сообщение клиента о готовности выполнить работу, которую закажет сервер.

Построение информационной системы на основе взаимодействующих .Net Remoting сервисов целесообразно также и экономически - клиентские лицензии для работы с базой данных (например, Oracle) требуется установить только на компьютеры с КриптоСервер-ами. Так для 1000 клиентов достаточно двух лицензий. И ещё один интересный момент - достаточно всего одной! дополнительной учетной записи клиента, от имени которого работаем с базой данных и запускаем КриптоСервер-а. Сетевые клиенты не работают напрямую с базой данных.

Прототип, с одной стороны, минимизирует время занятости физического канала связи за счет "импульсного" (соединение - запрос/ответ - разрыв связи) режима работы с ним. А с другой, хранит параметры сессии (индекс сессии, криптоключ, индекс ожидаемого пакета) и в базе данных и на рабочей станции. Поэтому не требуется аутентификация клиента при повторной установки связи. Суммарно, такой подход дает экономически оправданную возможность использования мобильной связи, при её посекундной тарификации.

Построенный прототип реально представляет собой распределённую систему управления централизованным каталогом видео фильмов. Данная типовая модель может служить базой для построения распределённых информационных систем в банковском деле, построения общегородских распределённых информационных систем, построения информационных систем супермаркетов, товарных складов и т.п.

Аутентификация пользователя и построение сессии.

Каждый пользователь защищенной информационной системы идентифицируется уникальной учетной записью, которая физически является строкой соответствующей таблицы базы данных информационного SQL сервера. Структура учетной записи

N/N	Name	Data Type	Size
1	GUID	uniqueidentifier	16
2	Имя	nvarchar	64
3	hshLogin	binary	20
4	hshPassword	binary	20
5	Cmd	int	4
6	e_mail	nvarchar	64
7	hshPinCod	binary	20
8	kiPinCod	binary	32

3, 4 и 7 поле строки содержат hash представления Login, Password и PinCod-a пользователя.

5 поле (cmd) содержит битовый вектор доступа к функциям и объектам системы. Поле 8 содержит параметры шифрования файла на клиентской машине, содержащего hshLogin и hshPassword.

Учетные записи предопределённых пользователей защищенной информационной системы - Администратора и Гостя, генерируются при первоначальном запуске Windows .Net приложения "КриптоСистема" на любом из КриптоСерверов. Учетные записи других пользователей создаются Администратором.

Администратор системы, используя Windows .Net приложение "RSAKeyPairGen.exe", генерирует также и ключевую пару асимметричного криптографического алгоритма. Открытый ключ, вместе с клиентским Windows .Net приложением, передается пользователю (в текущей версии встроен в клиентское приложение).

Предопределённый пользователь может получить доступ к данным или как Гость, с минимальными возможностями (только просмотр имеющихся в наличии фильмов), или как Администратор, с почти неограниченными возможностями, но для этого ему надо знать Login и Password. Гость и Администратор не имеют PinCod-a. Кроме группы предопределённых пользователей существует группа пользователей, являющихся владельцами фильмов. Для своего фильма владелец может редактировать все параметры. Для совместного фильма - только ограниченное число своих параметров.

Запуская клиентское приложение на своей рабочей станции, пользователь любой группы попадает в состояние регистрации. Его просят ввести аутентификационную информацию - Login и Password или PinCod. Информация вводится стандартным защищенным способом с использованием DPAPI-функция CredUIPromptForCredentials. Реализованы два варианта аутентификации пользователя:

В дальнейшем, в байтовых последовательностях будем выделять фрагменты, - без шифрования (…), шифрованные открытым ключом {…}, шифрованные симметричным ключем […].

Вариант 1. Пользователь вводит информации в поля Login и Password. Клиентское приложение создаёт временные код сессии (КС), ключ (Key) и initialization vector (IV) симметричного алгоритма шифрования. КС, Key и IV величины случайные.

```
app.gdСессия = Guid.NewGuid(); //-- Код сессии (КС)

new RNGCryptoServiceProvider().GetBytes(bb);

//-- байтовый вектор bb содержит хорошо рандомизированную величину

for(int i=0;i<24;i++) app.keyСессия[i]=bb[i]; //-- TDES key сессии

new RNGCryptoServiceProvider().GetBytes(bb);

for(int i=0;i<8;i++) app.ivСессия[i]=bb[i]; //-- IV сессии
```

Последовательность запроса = (КС +IV+0x00)+{Key}+ [0x0003] передаётся КриптоСерверу для получения "временного параметра" ВП. По его получению программа генерирует постоянные КС и Key сессии, и временный IV. Для Login и Password строятся их hash представления, к ним добавляется "временной параметр", которые вместе шифруются симметричным ключом; симметричный ключ шифруется открытым ключом асимметричного алгоритма. Байтовая последовательность = (КС+IV+0x00)+ {Key}+ [0x0001+ ВП+ hashLogin+hashPassword] передается КриптоСерверу.

Вариант 2. Пользователь вводит информацию только в поле Password (PinCod). Клиентское приложение создаёт временные КС, Key, IV, строит байтовую последовательность запроса "временного параметра" = (КС +IV+0x00)+{Key}+[0x0003] и передаёт её КриптоСерверу. После получения ВП от КриптоСервера программа строит временные КС, Key, IV и hash представление PinCod-а, добавляет к ним "временной параметр", шифрует и делает запрос КриптоСерверу на предмет наличия зарегистрированного пользователя с таким PinCod-ом. Если сервер подтверждает наличие, передавая Key и IV шифрования, то программа считывает и дешифрирует hash представления Login и Password из файла hshLP.bin и далее аналогично первому варианту. Расположение файла hshLP.bin задается в файле настройки - или на съемном носителе (дискета, cd, флеш) или крипто защищённом личном разделе жесткого диска.

В криптографии не является хорошим стилем, если тестовые значения PinCod, Lofin, Password и их hash представления долго находятся в памяти компьютера. Эти значения должны быть стерты сразу же, как только отпадет в них надобность. Для этой цели используются две функции:

```
//-- Очистка строк (забивание символом)

unsafe private void strClear(string str){

fixed(char* pStr=str){//-- Фиксируем память в куче, распределённую под str

for(int i=0;i<str.Length;i++) pStr[i]='w';

}

}

//-- Очистка байтовых последовательностей

private void btClear(byte[] bt)

{for(int i=0;i<bt.Length;i++) bt[i]=0x00;}
```

Схема передачи запроса КриптоСерверу: клиентское приложение -> сервер взаимодействия с клиентом -> сервер хранения запросов/ответов ->КриптоСервер

- сервер взаимодействия с клиентом это Remote .Net сервер на базе консольного приложения - для клиентов локальной сети, на базе IIS - для клиентов Интернета. Сервер управления взаимодействием с клиентами обеспечивает функционирование Remote .Net singlecall сервисов управления взаимодействием с клиентами. Информационные запросы клиентов динамически порождают создание копий singlecall сервиса в среде Remote .Net сервера или IIS. Для каждого клиента - своя копия. Как только ответ будет передан клиенту, его копия singlecall сервиса уничтожается.

- сервер хранения запросов/ответов это Remote .Net сервер на базе консольного приложения. Сервер обеспечивает функционирование Remote .Net singleton сервиса управления системой динамических абонентских ящиков.

Сервисы взаимодействия с клиентом и управления системой динамических абонентских ящиков никаких крипто преобразований над байтовыми последовательностями запроса/ответа не осуществляют. Они создают виртуальный канал в Интернете (сегменте локальной подсети), по которому передаётся шифрованная информация.

Взаимодействие Remote .Net сервисов с другими приложениями реализовано на основе интерфейсов ими представляемых, например, для сервиса управления системой динамических абонентских ящиков

```
public delegate void dlgEventHandlerОтвет(short j);

public interface IКлиент {

short БронированиеАбонентскогоЯщика(byte[] gd);

void ПоложитьЗапросВЯщик(short idx,byte[] bv,dlgEventHandlerОтвет vtHandler);

void ВзятьОтветИзЯщика(short idx,byte[] gd);

}
```

Реализация ожидания и обработки события ответа на запрос в сервисе взаимодействия с клиентом

```
public class crServer_KatalogVideo_Impl: MarshalByRefObject,IОбработкаЗапроса,IКлиент

{

static AutoResetEvent[] evtОжидание = new AutoResetEvent[1000];

...

//-- Функция обработки ответа на запрос клиента

//-- вызывается, как только ответ помещен

//-- в абонентский ящик клиента

[OneWay]

public void fОтвет(short j) {

if(j>=0){

//-- Продолжаем обработку запроса данным сервисом.

if(!(evtОжидание[j]==null)) ((AutoResetEvent)evtОжидание[j]).Set();

}
```

```
}
...
//-- Пытаемся бронировать динамический абонентский ящик
for(i=10;i>=0;i--) { //-- Ждем освобождения абонентского ящика
idx=iКлиент.БронированиеАбонентскогоЯщика(gd);
if(idx>=0) break;
Thread.Sleep(2000);
}
if(idx<0) return new byte[2]{1,2}; //-- Ошибка записи строки запроса

//-- создаём объект для ожидания
evtОжидание[idx]=new AutoResetEvent(false);

//-- В метод удаленного сервиса передается делегат обработчика события
iКлиент.ПоложитьЗапросВЯщик(idx,bv,new dlgEventHandlerОтвет(fОтвет));
//-- Ожидаем байтовую строку ответа
if(evtОжидание[idx].WaitOne(iTimeOut,false)) //-- не допустим вечный цикл
{//-- Не time-out. Запрос обработан
evtОжидание[idx].Reset();
evtОжидание[idx]=null;
return(iКлиент.ВзятьОтветИзЯщика(idx,gd)); //-- Сериализованный ответ
}
else {//-- Time-out
return new byte[2]{1,2}; //-- Ошибка записи строки запроса
}
...
```

Сервис управления системой динамических абонентских ящиков обрабатывает запрос сервиса взаимодействия с клиентом на бронирование абонентского ящика, получает байтовую последовательность запроса от клиентского приложения, проводит её частичный анализ и помещает её в выделенный абонентский ящик.

Сервис хранит информацию о запросах/ответах в следующих структурах:

byte[] даяКодСостояния значения компоненты: 0xff-бронировано, 0x00-свободно, 0x01-запрос, 0x02-запрос обрабатывается; 0x03-запрос обработан.

Guid[] даяСессия компонента хранит Guid сессии

object[] даяЗапросОтвет компонента хранит сериализованное, сжатое и шифрованное представление запроса/ответа

DateTime[] даяДлительность компонента хранит максимально допустимое время обработки запроса (длительность бронирования сессией динамического абонентского ящика)

dlgEventHandlerОтвет[] даяОтвет компонента хранит информацию, необходимую для обратного вызова функции обработки ответа по завершению обработки запроса.

Совокупность компонент разных векторов с одним индексом образуют динамический абонентский ящик (префикс "дая").

- КриптоСервер это Windows .Net приложение, взаимодействующее с сервисом управления системой динамических абонентских ящиков и сервисом хранения модифицирующих предложений посредством Remote .Net технологии. Все решения принимаются и выполняются только на уровне КриптоСерверов. КриптоСервера являются "мозгом" всей шести уровневой системы. Каждый КриптоСервер постоянно получает кольцевой индекс последнего записанного представления модифицирующего предложения и сканирует (точнее, обращается в цикле к сервису хранения запросов/ответов) вектор даяКодСостояния[] сервиса хранения запросов/ответов и при наличии запроса переходит к его детальному анализу и выполнению. Если запрос может быть обработан данным КриптоСервером, то анализ продолжается, иначе работа над запросом завершается без уничтожения запроса. Запросы разделены на группы по признаку максимальной длительности выполнения и есть КриптоСерверы, выполняющие только короткие запросы.

- Сервер хранения модифицирующих предложений это Remote .Net сервер на базе консольного приложения. Он активизирует remote singleton сервис хранения и работает с ним.

На практике встречаются ситуации когда либо SQL сервер имеет недостаточную производительность, либо каналы связи. Проектировщик вынужден кешировать редко модифицируемую информацию SQL сервера на серверах приложений или рабочих станциях (создавать её реплики). Обычно это информация справочников. Модель прототипа допускает изящное решение задачи управления кешированной информацией. В основе решения лежит тот факт, что вся работа с базой данных идет через КриптоСервера, включая и выдачу модифицирующих SQL предложений (INSERT, DELETE, UPDATE) серверу для изменения кешируемой информации. КриптоСервера получают сериализованные представления модифицирующих SQL предложений, формируют истинные предложения и передают их на SQL сервер. Если обработка предложения SQL сервером прошла штатно, то представления модифицирующих SQL предложений передаются серверу хранения модифицирующих предложений. Его сервис хранения реализует кольцевую структуру для хранения информации - только представления модифицирующих SQL предложений в случае кеширования на КриптоСерверах или параметров симметричного криптоалгоритма + сжатого и шифрованного представления. Каждый кеширующий субъект хранит у себя кольцевой индекс последнего им считанного представления и всегда в ответном пакете получает кольцевой индекс последнего записанного кем-то представления. Размер кольцевого индекса - 32 бит. Если индексы имеют разные значения, то субъект обращается к сервису хранения за представлениями модифицирующих SQL предложений. Интерфейс взаимодействия с сервером хранения модифицирующих предложений:

```
public interface IМодифицирующиеПредложения {

int ПолучитьКольцевойИндекс();

void ПоложитьМодифицирующееПредложение(byte[] bv);

byte[]bv ВзятьМодифицирующееПредложение (int idx);

}
```

По завершению обработки запроса КриптоСервер записывает ответ в компоненту вектора даяЗапросОтвет[] и устанавливает компоненту вектора даяКодСостояния[] =0x03. Сервер хранения запросов/ответов, используя переданный делегат, активизирует функцию fОтвет обработки события в сервисе взаимодействия с клиентом. Функция толкает "застывший" клиентский singlecall сервис, который забирает ответ из динамического абонентского ящика и отправляет его байтовую строку через Интернет (сегмент локальной подсети) клиенту. На этом singlecall сервис прекращает своё существование.

Если пользователь с данными Login и Password создан (имеется соответствующая запись в таблице Владельцы информационного SQL сервера), то генерируется запись в таблицу сессий информационного SQL сервера, запись ответа в абонентский ящик и КриптоСервер продолжает сканирование других абонентских ящиков. Группа функций с кодом 0x00 отвечает за регистрацию клиента. Любой пользователь может активизировать несколько самостоятельных сессий.

Структура записи таблицы сессий

N/N	Name	Data Type	Size	Примечание
1	gdСессия	uniqueidentifier	16	Код (guid) сессии
2	gdКлиент	uniqueidentifier	16	Код (guid) клиента (владельца), связь с таблицей Владельцы
3	dtСессия	datetime	8	Time-out
4	tdsKey	binary	24	Ключ симметричного шифрования
5	cmd	int	4	Битовый вектор доступа
6	ivKey	binary	8	IV симметричного алгоритма шифрования

7	strЗапрос	ntext	16	Храним SELECT запроса фильмов
8	hs	image	16	Храним сериализованный список Guid
9	idxПакет	int	4	Ожидаемый индекс приходящего пакета

Структура байтовой последовательности ответа в случае регистрации клиента

Ошибка - (двухбайтовый код ошибки)

Штатно - (0x00+0x00+IV)+[симметрично зашифрованное Info]

Структура Info ответа для регистрации:

[cmd клиента]+[guid аутентифицированного клиента]+[размер страницы]+[имя клиента]

Структура байтовой последовательности для общего случая информационного запроса/ответа:

Запрос:

(КС+IV+индекс группы команд)+[Info]

Структура Info запроса:

[idxПакет+индекс команды в группе+параметры команды]

Структура ответа:

Ошибка - (двухбайтовый код ошибки)

Штатно - (0x00+0x00+IV)+[Info]

В общем случае Info ответа содержит результирующий набор данных в виде отсортированного массива строк. Размер реального массива может существенно превышать допустимые границы, например, в случае запроса показать все фильмы. Для работы с подобными наборами данных используем страницы. Идея проста - результирующий набор данных делится на страницы и за единичный акт обмена на рабочую станцию пользователя передаётся одна страница отсортированного результирующего набора данных. С целью уменьшения трафика, предложение SELECT для построения страничного набора данных и его параметры передаются по сети единожды и хранятся в полях строки таблицы Сессий (strЗапрос, hs). Для получения следующей страницы достаточно передачи её номера. Размер страницы в записях - параметр настройки сервиса управления системой динамических абонентских ящиков. Эту величину КриптоСервер-ы получают по отдельному запросу. Основную часть задачи страничного представления информации клиенту решает SQL сервер двумя динамическими запросами - Sel_1 и sel_2:

CREATE PROCEDURE au_getPageФильмыDel

@Sel_1 ntext,

@RowNum int,

@Sel_2 ntext

```
AS

-- очистим таблицы GUID Фильмов

TRUNCATE TABLE #gdФильмы

TRUNCATE TABLE #gdTmp

-- Запись отсортированных ключей в #gdTmp

EXECUTE(@Sel_1)

-- Перепись суррогатных ключей страницы

INSERT INTO #gdФильмы(GUID)

SELECT GUID FROM #gdTmp WHERE RowNum > @RowNum ORDER BY RowNum"

TRUNCATE TABLE #gdTmp

-- Вот эти данные уходят на клиента

EXECUTE(@Sel_2)

GO
```

Каждый "КриптоСервер" имеет один Connect с базой данных. В рамках Connect-а строятся две временные таблицы - #gdФильмы и #gdTmp

```
CREATE TABLE #gdФильмы (RowNum int IDENTITY,[GUID] [uniqueidentifier]);

CREATE TABLE #gdTmp (RowNum int IDENTITY,[GUID] [uniqueidentifier]);
```

Первым динамическим запросом "режем" таблицу по длине и заполняем временную таблицу #gdTmp значениями суррогатных ключей выбранных отсортированных строк исходной таблицы

```
INSERT INTO #gdTmp(GUID) SELECT TOP ... GUID FROM Фильмы WHERE ... ORDER BY ...
```

Вторым динамическим запросом "режем" выбранные строки таблицы по ширине и возвращаем выборку

```
SELECT f.GUID, f.Преф, f.Название_рус ...

FROM Фильмы f INNER JOIN #gdФильмы g ON f.GUID=g.GUID ORDER BY g.RowNum
```

Фрагмент программы получения страницы, сериализации и сжатия её строк:

```
...

MemoryStream ams=new MemoryStream(40000);

int i=0; //-- подсчет числа строк страницы

object[] objArray=new object[10];
```

```
try {
//-- Запрос к базе данных на построение выборки
cmmКаталогВидео.Parameters.Clear();
cmmКаталогВидео.Parameters.Add("@Sel_1", SqlDbType.NText).Value = Sel_1;
cmmКаталогВидео.Parameters.Add("@RowNum", SqlDbType.Int).Value = iRecords;
cmmКаталогВидео.Parameters.Add("@Sel_2", SqlDbType.NText).Value = Sel_2;
cmmКаталогВидео.CommandText = "au_getPageФильмыDel";
cmmКаталогВидео.CommandType = CommandType.StoredProcedure;
sdr=cmmКаталогВидео.ExecuteReader();
//-- Сериализуем строки выборки таблицы Фильмы
//-- Цикл чтения строк выборки из таблицы Фильмы
while(sdr.Read()){
//-- Построим строку страницы (мой вариант сериализации)
xms=new MemoryStream(1024);
BitRowClear(); //-- Очистим битовое поле признаков DBNull
if(sdr.IsDBNull(0)){SetBit(0);} else{GuidToXMS(sdr.GetGuid(0));} //-- GUID
if(sdr.IsDBNull(1)){SetBit(1);} else{StringToXMS(sdr.GetString(1));} //-- Преф
//-- Читаем Название_рус
if(sdr.IsDBNull(2)){SetBit(2);} else{StringToXMS(sdr.GetString(2));}
//-- Читаем Название_анг
if(sdr.IsDBNull(3)){SetBit(3);} else{StringToXMS(sdr.GetString(3));}
if(sdr.IsDBNull(4)){SetBit(4);} else{StringToXMS(sdr.GetString(4));} //-- Год
if(sdr.IsDBNull(5)){SetBit(5);} else{StringToXMS(sdr.GetString(5));} //-- Жанр
if(sdr.IsDBNull(6)){SetBit(6);} else{StringToXMS(sdr.GetString(6));} //-- Режисёр
if(sdr.IsDBNull(7)){SetBit(7);} else{StringToXMS(sdr.GetString(7));} //-- Актёр
//-- Читаем Дата
if(sdr.IsDBNull(8)){SetBit(8);} else{DateTimeToXMS(sdr.GetDateTime(8));}
//-- Читаем ts
if(sdr.IsDBNull(9)){SetBit(9);} else{xms.Write((byte[])sdr.GetValue(9),0,8);}
```

```
ams.Write(BitDBNullRow,0,BitDBNullRow.Length); //-- Признаки DBNull
/* //-- вариант сериализации Microsoft
objArray[0] =sdr.GetValue(0); //-- GUID
objArray[1] =sdr.GetValue(1); //-- Преф
objArray[2] =sdr.GetValue(2); //-- Название_рус
objArray[3] =sdr.GetValue(3); //-- Название_анг
objArray[4] =sdr.GetValue(4); //-- Год
objArray[5] =sdr.GetValue(5); //-- Жанр
objArray[6] =sdr.GetValue(6); //-- Режисёр
objArray[7] =sdr.GetValue(7); //-- Актёр
objArray[8] =sdr.GetValue(8); //-- Дата
objArray[9] =(byte[])sdr.GetValue(9); //-- ts (timestamp!)


if(!binSerObject(objArray)) { //-- сериализуем в xms
xms.Close(); //-- закроем Reader
sdr.Close();
xbb=new byte[2]{2,10}; //-- Ошибка сериализации objArray
return false;
} //-- end if
/*
xms.WriteTo(ams); //-- добавим в рабочий MemoryStream
xms.Close();
i++;
} //-- end while
} //-- end try
catch {
if(sdr!=null) sdr.Close();
xbb=new byte[2]{4,10}; //-- Ошибка сериализации страницы
```

```
return false;
}
sdr.Close();
//-- Передаем значение полной выборки?
byte[]abb;
if(chf<0) {//-- Нет
xbb=new byte[4+ams.Length];
IntToXBB(i,0); //-- число строк страницы
abb=ams.ToArray();
abb.CopyTo(xbb,4);
}
else {//-- Да
xbb=new byte[8+ams.Length];
IntToXBB(chf,0);
IntToXBB(i,4);
abb=ams.ToArray();
abb.CopyTo(xbb,8);
}
ams.Close();
xms.Close();
//-- сжимаем информацию
if(!Компрессор()) {
xbb=new byte[2]{2,10}; //-- Ошибка сжатия
return false;
}
. . .
```

Прототип разрабатывался для работы в Интернет, а это далеко не самая быстродействующая среда. Поэтому для увеличения скорости реакции системы на запрос реализована идея динамической разовой подкачки недостающей информации. Суть идеи в следующем - информация, нужная клиенту, разбита на два плана. Информация первого плана (основная) содержится в строках страницы выборки по фильмам. Она передается клиенту по запросу страницы, записывается в таблицу локальной DataSet и отображается на главном DataGrid информационной панели (UserControl). С каждой строкой страницы ассоциируется информация второго плана (частная). По запросу страницы с основной информацией её частная информация автоматически в локальный DataSet не передается и соответствующие его таблицы в начальный момент пусты.

Клиент сканирует информацию первого плана ScroolBar - ом грида. Для получения информации второго плана он выбирает требуемую строку страницы (щелчок мышки на соответствующей строке грида). Клиентское приложение делает попытку отобразить требуемую информации из локального DataSet. Если её там нет, то следует разовый запрос подкачки недостающей информации с серверной базы данных.

Повторяя операцию подкачки для других строк страницы, клиент может динамически, по запросам, построить полный локальный информационный фрагмент серверной базы данных. А может просто перейти к другой странице. В этом случае таблицы клиентского DataSet, содержащие информацию второго плана, будут очищены.

Авторизация пользователя при дальнейших запросах производится по коду сессии, параметрам симметричного алгоритма шифрования (ключу и IV шифра) и сессионному индексу запроса.

Ограничения доступа пользователя к функциям и записям таблиц базы данных информационной защищенной системы задаются битовым вектором "cmd". Решение по ограничению доступа принимаются КриптоСервером (уровень бизнес-логики).

Защита трафика осуществляется путем шифрования симметричным ключом сессии передаваемой байтовой последовательности. Для защиты передачи симметричного ключа сессии используется открытый ключ. Для зашиты процесса аутентификации клиента и построения сессии применяется "временной параметр". Защита сессии от возможной повторной передачи серверу перехваченного запроса а обслуживание реализована так:

Во время обработки запроса данной сессии вновь приходящие от неё запросы будет отклонены;

КриптоСервер-ы формируют и анализируют сессионные индексы запросов обслуживания и, если запрос имеет неожидаемый сессионный индекс, то будет отклонен.

Защита клиентского приложения от несанкционированной модификации реализована так:

клиентское приложение создано в форме много проектного решения и компилируется с использованием строгих имен,

его узловая сборка (файл clsStartCrypto.dll) зашифрована в файл clsStartCrypto.cry,

если клиент идентифицирован, то КриптоСервер передаёт клиентскому приложению параметры дешифрования файла clsStartCrypto.cry,

клиентское приложение дешифрирует файл, загружает сборку, создает и активизирует объект заданного класса (допустим и вариант пересылки сборки КриптоСервером на клиентский компьютер):

. . .

```
//-- Дешифруем clStartSer.cry (крипто clStartSer.dll)

//-- ivKey - параметры дешифрования, переданные КриптоСервер-ом

//-- xbb <-- clStartSer.dll (byte[]xbb - переменная уровня класса)

if(!TDESDecryptStartCrypto(ivKey)) return false;

Assembly asStartSer=null;

object obj=null;

try {

//-- Загружаем сборку

asStartSer=Assembly.Load(xbb);

Type[] mytypes = asStartSer.GetTypes();

obj = Activator.CreateInstance(mytypes[0],null,null);

return true;

}

catch {return false;}
```

Сборка clsStartCrypto.dll состоит из одного класса. Для доступа к его методам используются делегаты. Сборка должна компилироваться в процессе компиляции всего приложения, но не один элемент приложения не должен напрямую ссылаться на сборку clsStartCrypto.dll (не должен, используя new, создавать объект из класса, содержащегося в сборке).

Производительность информационной системы в широком диапазоне регулируется изменением числа "КриптоСерверов" на вычислительном узле и изменением числа вычислительных узлов. Ручная сериализация результирующего набора строк таблицы фильмов не позволила кардинально повысить производительность системы. Внутренние методы сериализации .Net Framework работают прекрасно и видимо разумно использовать именно их, плюс сжатие информации.

Реализация программных методов хеширования, сериализации типов, компрессии информации и шифрования

```
//-- Переменные уровня класса

byte[] xbb;

MemoryStream xms;

byte[] TDESKey=byte[24];

byte[] TDESIV=byte[8];
```

```
int xbbN;

byte[] BitDBNullRow=byte[2]; //-- флаги DBNull в полях записи

//-- Установка бита

//===========================================================

private void SetBit(int i)

{BitDBNullRow[i/8]|=(byte)(1 << (i%8));}


//-- Формируем hash код строки

//=====================================

private byte[] hКод(string s){

UnicodeEncoding UE = new UnicodeEncoding();

//-- Преобразование строки в вектор байтов.

byte[] bv = UE.GetBytes(s);

//-- Создадим объект SHA1Managed для формирования hash значения.

SHA1Managed SHhash = new SHA1Managed();

//-- Построим hash значение для байтового вектора.

return SHhash.ComputeHash(bv);

}

//-- =======================

//-- == Сериализация типов ==

//-- =======================

//-- Бинарная сериализация известного объекта

//-- В нашем случае зто вектор строк результирующего набора данных

//-- xms <-- байтовое представление объекта

//===========================================================

private bool binSerObject(object obj){

try {

xms=new MemoryStream();

BinaryFormatter binForm=new BinaryFormatter();
```

```
binForm.Serialize(xms,obj,null);

xms.Position=0;

return true;

}

catch {return false;}

}

//-- Бинарная десериализация известного объекта из памяти

//-- xms <-- байтовое представление объекта

//=========================================================

private object binDeSerObject(){

try {

BinaryFormatter binForm=new BinaryFormatter();

return binForm.Deserialize(xms,null);

}

catch {return null;}

}

//-- Формируем 4-х байтовое представление int

//-- ii целое

//-- k индекс в xbb

//-- xbb целое

//=================================================

private void IntToXBB(int ii,int k) {

byte[] bt=BitConverter.GetBytes(ii);

bt.CopyTo(xbb,k);

}


//-- Формируем 4-х байтовое представление int

//-- xms целое

//=================================================
```

```
private void IntToXms(int ii) {

byte[] bt=BitConverter.GetBytes(ii);

xms.Write(bt,0,bt.Length);

}

//------------ Восстановление int ------------

//-- int байтовое представление xbb

//============================================================

private int XBBToInt()

{return BitConverter.ToInt32(xbb,0);}

//------------ Восстановление int ------------

//-- int байтовое представление xms

//============================================================

private int xmsToInt() {

byte[]bt=new Byte[4];

xms.Read(bt,0,4);

return BitConverter.ToInt32(bt,0);

}

//-- Формируем 8-и байтовое представление long

//-- xms long

//==================================================

private void LongToXms(long ii) {

byte[] bt=BitConverter.GetBytes(ii);

xms.Write(bt,0,bt.Length);

}

//------------ Восстановление long ------------

//-- long xms(байтовое представление long)

//============================================================

private long xmsToLong() {

byte[]bt=new Byte[8];
```

```
xms.Read(bt,0,8);

return BitConverter.ToInt64(bt,0);

}

//-- Формируем 8-и байтовое представление DateTime в xms

//-- xms DateTime

//=========================================================

private static void DateTimeToXMS(DateTime dt) {

LongToXMS(dt.Ticks);

}

//-- Восстановление DateTime из xms

//-- DateTime xms

//===========================================================

private static DateTime xmsToDateTime() {

long n=xmsToLong();

return new DateTime(n); //-- DateTime <-- long

}

//-- Работа с байтовым представлением строкового типа

//-- s - Строковый тип; bt - байтовое представление

//===========================================================

byte[]bt=UnicodeEncoding.UTF8.GetBytes(s); //-- byte[] s

string s=UnicodeEncoding.UTF8.GetString(bt); //-- s byte[]


//-- ==========================

//-- == Компрессия информации ==

//-- ==========================

//-- Компресор

//-- вход: xbb (вектор для компресии)

//-- выход: xbb (длина исходного(4)+сжатый вектор)

//==========================================================
```

```
private bool Компрессор(){

if(xbb.Length==0) return true;

byte[]bb;

try {

bb=ZipBase.Compress(xbb);

int m=xbb.Length;

xbb=new byte[4+bb.Length];

IntToXBB(m,0);

bb.CopyTo(xbb,4);

return true;

}

catch {return false;}

}

//-- Декомпрессор

//-- вход:

//-- фрагмент xms содержит компресированный байтовый вектор,

//-- указатель стоит на начале фрагмента

//-- выход;

//-- xms <-- декомпресированное Info

//===========================================================================

private bool Декомпрессор(){

byte[] bb=new byte[((int)xms.Length)-4];

try {

int n=xmsToInt(); //-- длина исходного Info

byte[] bb1=new byte[n]; //-- место для декомпрессированного Info

xms.Read(bb,0,bb.Length); //-- bb <-- компрессированное Info

ZipBase.Uncompress(bb1,bb);

xms.Close();

xms=new MemoryStream(bb1,0,bb1.Length);
```

```
return true;
}
catch {
xms.Close();
return false;
}
}
//-- ================
//-- == Шифрование ==
//-- ================
//-- TDES крипто
//-- вход: xbb (вектор для шифрования)
//-- выход: xbb (шифрованный вектор)
//=========================================================
private bool TDESEncrypt(){
if(xbb.Length==0) return true;
CryptoStream cs=null;
MemoryStream ms=null;
try {
byte[]bb=new Byte[32];
byte[]key=new byte[24];
byte[]iv=new byte[8];
new RNGCryptoServiceProvider().GetBytes(bb);
//-- байтовый вектор bb содержит хорошо рандомизированную величину
for(int i=0;i<8;i++) TDESIV[i]=bb[i];
TDESKey.CopyTo(key,0);
TDESIV.CopyTo(iv,0);
ms=new MemoryStream();
cs=new CryptoStream(ms,tdes.CreateEncryptor(key,iv),CryptoStreamMode.Write);
```

```
cs.Write(xbb,0,xbb.Length);

cs.FlushFinalBlock();

xbb=ms.ToArray();

cs.Close();

ms.Close();

return true;

}

catch {

if(cs!=null) cs.Clear();

if(ms!=null) ms.Close();

return false;

}

}

//-- TDES декрипто

//-- фрагмент MemoryStream xms содержит шифрованную байтовую последовательность,

//-- указатель стоит на начале фрагмента

//-- ln <-- длина фрагмента

//==============================================================================

private bool TDESDecrypt(int ln){

CryptoStream cs=null;

try {

byte[]key=new Byte[24];

byte[]iv=new Byte[8];

TDESKey.CopyTo(key,0);

TDESIV.CopyTo(iv,0);

cs=new CryptoStream(xms,tdes.CreateDecryptor(key,iv),CryptoStreamMode.Read);

xbbN=(int)cs.Read(xbb,0,ln);

cs.Close();

return true;
```

```
}
catch {
if(cs!=null) cs.Clear();
return false;
}
}
```

В качестве Web сервера используется IIS, с максимально отключенными возможностями.

В качестве Remote .Net сервера используется консольное приложение.

В качестве ассимметричного алгоритма шифрования используется 1024 битный RSA, в качестве симметричного - TrippleDES.

Сервисы взаимодействия с клиентом и хранения запросов/ответов ни под каким предлогом не могут войти во внутреннюю локальную сеть и получить доступ к КриптоСерверам и данным информационного SQL сервера!!!!

Все приложения системы (клиентские, Remote сервис под Web сервер, Remote сервис под Remote сервер, КриптоСервер и RSAKeyPairGen) написаны на C#. Клиентское Windows.Net приложение для работы с Remote .Net сервисом на IIS сервере или с Remote .Net сервисом на Remote .Net сервере написано в формате много проектного решения (много файловой сборки).

Что должно быть в промышленной версии, но отсутствует в прототипе:

- проработка вопросов сопровождения ПО, включая upgrate клиентской части программной системы

- сбор и накопление статистики для будущих эвристических алгоритмов фильтрации запросов

- счетчики загрузки и производительности информационной системы

В настоящее время дл фильтрации запросов применяется шифрование, "временной параметр" и отклонение сессией текущего запроса, если не выполнен предыдущий. Возможно ошибочное завершение выполнение запроса по time-out.

Графический интерфейс пользователя для работы с данными, как в Интернете, так и локальной сети абсолютно идентичен и реализован в одном приложении. Выбор канала обмена (Tcp или Http) задается в файле "НастройкаRemoteServiceКлиента.txt" настройки приложения. Разница в работе ощущается лишь по длительности операции загрузки страницы данных с информационного SQL сервера в локальную базу данных. В случае локальной сети этого не наблюдается.

Клиентское приложение построено в форме много проектного Visual Studio (VS-) решения (много файловой сборки), основу которого составляют четыре проекта - clsApp, dbДоступ, clsStartCrypto и wcКаталогВидео. Решение компилируется с использованием строгих имён.

Первый проект - clsApp, реализует построение класса общих (глобальных) параметров решения, включая описание и определение делегатов общих функций всего проекта. Он не ссылается ни на один другой проект решения, но на него могут ссылаться все. Иными словами, его видят все, он не видит никого. В последовательности компиляции проект clsApp стоит первым номером. Особенностью данного класса является определение здесь общих параметров как статических переменных:

static public dgtРегистрацияПользователя РегистрацияПользователя=null;

static public dgtАнализPinCod АнализPinCod=null;

Второй проект - dbДоступ, содержит реализацию единой локальной базы данных (DataSet) всего клинтского приложения. В последовательности компиляции он стоит вторым номером.

Третий проект - clsStartCrypto, является реализацией интерфейса клиентского приложения с КриптоСервером. Это узловой проект всего решения, отвечающий за реализацию обмена информацией клиентского приложения с КриптоСеревер-ами. Его сборка поставляется клиенту в зашифрованном виде.

Четвёртый проект - wcКаталогВидео, содержит ссылки на все другие проекты решения. Он видит всех, его не видит никто. В последовательности компиляции он последний. Содержит описание единого окна решения и реализацию общих функций пользовательского интерфейса, включая static void Main();. Первое, что создает Main, это объект глобальных параметров на основе проекта clsApp - app xApp=new app(); и заполняет его значениями, включая доступ к локальному DataSet - app.dbДоступ=new nsКаталогВидео.dbДоступ();. В дальнейшем, доступ к DataSet из любого другого объекта реализуется так: private nsКаталогВидео.dbДоступ pp=(app.dbДоступ as dbДоступ);.

Остальные проекты или содержат описания оконных панелей графического интерфейса и реализуют соответствующую функциональность, или реализуют функциональность вывода отчетов (реализован вывод отчетов в форматах HTML и CrystalReport).

Всё хорошо, но удручает отсутствие возможности выгрузки из памяти не нужной сборки.

Также, для построения VS-решений применяется внешняя Zlib.dll, реализующая функциональность компрессора данных.