Практический курс «Основы защиты приложений при помощи Guardant API»

Урок 3

Продвинутые механизмы и технологии работы с ключом (Delphi)

Содержание

1. Постановка задачи и планирование ее решения

Тестовое приложение

В качестве тестового приложения выбрана программа, представляющая собой графический интерфейс работы с БД MS Access, содержащей список клиентов и информации о них.

Постановка задачи

1. Зашифровать на аппаратном алгоритме часть данных таблицы «Клиенты».
2. Зашифровать на программно-реализованном алгоритме дополнительную таблицу «Список покупок».
3. Создать дополнительный поток и, используя генерацию случайных чисел, организовать:
    - проверку шифрования на аппаратном алгоритме по принципу «вопрос-ответ»;
    - генерацию электронной цифровой подписи (ЭЦП);
    - проверку ЭЦП;
    - проверку заведомо неверной ЭЦП.

Действия необходимые для решения задачи

Написать программу для шифрования таблиц «Клиенты» и «Список покупок»

1. Создать процедуры шифрования/расшифрования поля «Комментарий» в таблице «Клиенты» на аппаратном алгоритме GSII64 *(GrdCrypt)*, используя в качестве вектора инициализации ID клиента.
2. Создать процедуры шифрования/расшифрования поля «Название товара» в таблице «Список покупок» на программно-реализованном алгоритме AES256 *(GrdCrypt)*, используя в качестве ключа шифрования для функции AES256 поля «ID клиента» и «Имя клиента» из таблицы «Клиенты», зашифрованные на алгоритме GSII64.

При старте программы

1. Расшифровать поле «Комментарий» для каждого клиента на аппаратном алгоритме GSII64 и отобразить в расшифрованном виде *(GrdCrypt).*

При добавлении новой записи в таблицу «Клиенты»

1. Получить ID нового клиента в таблице «Клиенты».
2. Зашифровать комментарий на аппаратном алгоритме GSII64, используя в качестве вектора инициализации ID клиента *(GrdCrypt).*

При добавлении новой записи в таблицу «Список покупок»

1. Взять из таблицы «Клиенты» значения текущей строки «ID клиента» и «Имя клиента».
2. Зашифровать «ID клиента» и «Имя клиента» на аппаратном алгоритме GSII64 *(GrdCrypt).*
3. Зашифровать «Название товара» на программно-реализованном алгоритме AES256, используя в качестве секретного ключа шифрования значение, полученное в п.2 *(GrdCrypt).*

При генерации отчета «Список покупок клиента»

1. Взять из таблицы «Клиенты» значения текущей строки «ID клиента» и «Имя клиента».
2. Зашифровать «ID клиента» и «Имя клиента» на аппаратном алгоритме GSII64 *(GrdCrypt).*
3. Расшифровать набор полей «Название товара» на программно-реализованном алгоритме AES256, используя в качестве секретного ключа шифрования значение, полученное в п.2 *(GrdCrypt).*

Дополнительный поток

1. Запустить на бесконечное выполнение поток с паузой 5000 миллисекунд.
2. Вычислить случайное число, используя Microsoft CryptoAPI *(CryptGenRandom).*
3. Если значение попадает в диапазон (0 – 30%), то осуществить проверку таблицы вопрос-ответ на аппаратно-реализованном алгоритме AES *(GrdCrypt).*
4. Если значение попадает в диапазон (30 – 61%), то сгенерировать ЭЦП случайной строки на аппаратном алгоритме электронной цифровой подписи *(GrdSign).*
5. Если значение попадает в диапазон (61 – 92%), то проверить электронную цифровую подпись, сгенерированную в п.4 *(GrdVerifySign).*
6. Если значение попадает в диапазон (92 – 100%), то проверить электронную цифровую подпись, сгенерированную в п.4 заведомо преобразованную в неверную *(GrdVerifySign).*

2. Описание решения задачи

2.1 Введение

В предыдущем уроке были рассмотрены базовые способы защиты с использованием аппаратных алгоритмов. Они сводились к преобразованию данных в электронном ключе с последующей проверкой результатов. Стойкость защиты на основе электронных ключей напрямую зависит от степени интеграции защитных механизмов в логику программы. Глубокая интеграция позволит снизить до минимума возможность «вырезания» блоков защиты из программного кода. Грамотная организация логики проверок затруднит эмуляцию вызовов API злоумышленнику.

В этом уроке мы попытаемся организовать частичное шифрование базы данных аппаратно и программно-реализованными алгоритмами. Делается это с целью более плотной привязки приложения к электронному ключу. В результате, электронный ключ будет дополнительно нести «полезную» нагрузку – с помощью него будет обеспечиваться защита используемых приложением данных от компрометации.

2.2 Основы шифрования баз данных

Вначале необходимо определить, какие данные могут быть зашифрованы без ущерба для общей функциональности и производительности приложения. В первую очередь, это поля, не являющиеся ключевыми, и по которым не осуществляется поиск, сортировка или фильтрация. В демонстрационном приложении такими полями являются «Комментарии» в таблице «Клиенты» и «Название товара» в таблице «Список покупок».

При шифровании полей базы данных необходимо учесть два важных момента. Первый состоит в том, что станет невозможным использование поиска, фильтрации и предварительного индексирования зашифрованных полей средствами СУБД. Второй – в результате шифрования с помощью алгоритмов, реализованных в Guardant API, на выходе получается массив байт, состоящий из печатаемых и непечатаемых символов. Для их хранения необходимо использовать поля специального типа или преобразовать данные в строку печатаемых символов, например, воспользовавшись кодировкой Base64. При ее использовании нельзя забывать, что длина полученной строки может превышать длину исходной, поэтому размер поля в описании структуры БД необходимо увеличить. Использование полей специального типа связано с дополнительной сложностью – необходимостью их отдельной обработки, поэтому воспользуемся преобразованием в Base64.

Шифрование данных усложняет логику работы программы и требует внимательного отношения ко всем функциям и методам, работающим с этими данными. Отсутствующие в каком-то блоке программы методы шифрования-расшифрования могут привести и к потере данных, и к неработоспособности самой программы.

2.3 Использование Guardant API для шифрования данных

Для шифрования с помощью ключей Guardant Sign и Guardant Time используется функция «GrdCrypt», ее формат подробно рассматривался на предыдущем уроке. Доступные алгоритмы шифрования можно условно разделить на 2 группы: аппаратные и программно-реализованные. У каждой из групп есть свои особенности, которые необходимо учитывать при выборе способа шифрования.

Аппаратные алгоритмы. Данная группа подразумевает преобразование данных непосредственно в ключе защиты. Это практически исключает возможность расшифрования данных без использования самого ключа. В ключах Guardant Sign (Guardant Time) реализованы два алгоритма – GSII64 с длиной ключа 128 или 256 бит, и AES с длиной ключа 128 бит. У аппаратных алгоритмов есть только один существенный недостаток – скорость выполнения. На ключах примерное время шифрования (расшифрования) буфера данных, состоящего из имени и фамилии, занимает порядка 20-30 миллисекунд. Поэтому, если типичный запрос из БД превышает 30-50 записей, использовать аппаратное шифрование не рекомендуется. Обойти данное ограничение можно двумя способами: используя частичное шифрование БД (например, шифровать каждую запись, номер которой нацело делится на 10), или воспользовавшись связкой программного и аппаратного алгоритмов.

Программно-реализованные алгоритмы. Их код располагается в библиотеках Guardant API. При шифровании данных ими обращений к электронному ключу не происходит. Основным достоинством программно-реализованных алгоритмов является высокая скорость шифрования, что позволяет использовать их для кодирования больших объемов данных. Недостаток – необходимость хранения в коде и передача в качестве параметра его секретного ключа; как следствие – потенциально менее высокая защищенность от взлома. Поэтому использовать такие алгоритмы рекомендуется в связке с аппаратными (как это делать – см. ниже). В настоящее время в состав Guardant API включена реализация симметричного алгоритма AES с длиной ключа 256 бит.

Таким образом, при шифровании больших объемов данных лучшим способом, сочетающим в себе высокую скорость шифрования-расшифрования и защищенность, является шифрование на программно-реализованном алгоритме с ключом, полученном на аппаратном алгоритме.

2.4 Шифрование на аппаратном алгоритме

В тестовом приложении на аппаратном алгоритме будет зашифровано поле «Комментарий» таблицы «Клиенты», т.к. данное поле не участвует в поиске и фильтрации данных и не подвергается предварительному индексированию. В нашем случае поле «Комментарий» вынесено на главную форму. Это допустимо ввиду малого количества записей в таблице «Клиенты». В реальном приложении с большим объемом данных это может негативно сказаться на производительности. Возможным решением может быть вынос данного поля на отдельную форму с подробной информацией о клиенте.

Для шифрования поля «Комментарий» используем алгоритм GSII64 в режиме OFB, позволяющем шифровать блоки данных произвольной длины. В качестве начального значения вектора инициализации будем задавать модифицированное значение поля «ID клиента». В противном случае (при использовании одинакового вектора инициализации), несколько записей, имеющих одинаковое поле «Комментарий», были бы зашифрованы идентично. Так как все клиенты имеют уникальный «ID клиента», то результат шифрования всех записей будет различным. Побочным эффектом является дополнительное усложнение логики защиты, что хорошо.

Вызов функции шифрования на алгоритме GSII64 выглядит следующим образом:

```
ClientID := ClientID xor 769390234;  // Преобразуем ID клиента
szInitVectorGS2:=IntToHex(ClientID,8);      // Вектор инициализации для алгоритма GSII64, 8
                                            // символов
// Шифруем комментарий на алгоритме GSII64
nRet:=GrdCrypt(hGrd,                        { Хэндл контейнера Guardant }
       04,                                  { Номер аппаратного алгоритма GSII64}
       Length(CommCrypt),                   { Длина строки для шифрования }
       @CommCrypt[1],                       { Указатель на строку для шифрования }
       GrdAM_OFB + GrdAM_Encode,            { Шифрование в режиме OFB }
       @szInitVectorGS2[1],                 { 64-битный вектор инициализации }
       nil,                                 { Не используется }
       nil );                               { Не используется }
ErrorHandling(hGrd,nRet);                   // Проверяем на возможные ошибки
CommCrypt64 := EncodeString(CommCrypt);     // Преобразуем в  Base64
```

В конце происходит преобразование строки в Base64, после этого результат можно заносить в обычное текстовое поле БД.

2.5 Шифрование на программно-реализованном алгоритме

В качестве примера шифрования на программно реализованном алгоритме в тестовом приложении выбрана таблица «Список покупок», в которую заносятся все покупки, сделанные каждым клиентом. В программе доступна опция просмотра отчета о покупках каждого клиента. Выборка для данного отчета почти всегда будет содержать много записей, поэтому целесообразно использовать шифрование на программно-реализованном алгоритме. Ранее упоминалось об особенностях аппаратных и программных алгоритмов. Исходя из них, реализуем следующую схему: в качестве секретного ключа шифрования программного алгоритма AES256 (32 байта) будем использовать комбинацию из ID и имени клиента, зашифрованную на аппаратном алгоритме GSII64.

```
{ Создаем строку из ID и имени клиента}
CryptKeyAES :=  IntToStr(ClientID) + ClientName;
{ Если ее длина меньше 32, дополняем значениями ID клиента}
while (Length(CryptKeyAES)<33) do CryptKeyAES := CryptKeyAES + IntToStr(ClientID);
CryptKeyAES := Copy(CryptKeyAES,1,32);          // Обрезаем до 32 символов
{ Шифруем на аппаратном алгоритме GSII64 в режиме ECB }
nRet:=GrdCrypt(hGrd,                        { Хэндл контейнера Guardant }
        04,                                 { Номер аппаратного алгоритма }
        Length(CryptKeyAES,                 { Длина строки для шифрования }
        @CryptKeyAES[1],                    { Указатель на строку для шифрования }
         GrdAM_ECB + GrdAM_Encode,          { Шифрование в режиме ECB }
        nil,                                { в режиме ECB не используется }
        nil,                                { Не используется }
        nil );                              { Не используется }
```

Ключ шифрования AES256 получен, теперь можно переходить к шифрованию покупок клиента на программно-реализованном алгоритме AES256. Шифрование осуществляется с помощью той же функции «GrdCrypt», где в качестве параметра «номер алгоритма» выступает константа с номером программно-реализованного алгоритма AES256. Так же к параметрам добавляются полученный ключ шифрования (32 байта) и буфер, необходимый для работы алгоритма.

В разделе объявления переменных буфер объявляется следующим кодом:

```
abyGrdAES_Context:   array[0..GrdAES_CONTEXT_SIZE] of Byte;
```

Здесь GrdAES.CONTEXT_SIZE – это константа, содержащая длину буфера для алгоритма AES256.

```
{ Шифрование на программно-реализованном алгоритме AES256 }
nRet:= GrdCrypt(hGrd,               { Хэндл контейнера Guardant }
        GrdSC_AES256,               { Номер программно-реализованного алгоритма AES256}
        Length(PurchaseName),       { Длина данных для шифрования }
        @PurchaseName[1],           { Данные для шифрования }
        GrdAM_Encrypt + GrdAM_CFB + GrdSC_All,  { Шифровать в режиме CFB,
                                                  единственный блок данных}
        @szInitVectorAES[1],        { Вектор инициализации }
        @CryptKeyAES[1],                { Секретный ключ для шифрования }
        @abyGrdAES_Context[0] );        { Буфер для преобразования }
{ Обновляем значение поля, преобразуя данные в  base64 }
ADODataSet2.FieldByName('PurchaseName').Text := EncodeString(PurchaseName);
```

Расшифрование данных происходит аналогичным образом.

При проектировании защиты базы данных важно помнить о существующих зависимостях между данными. В нашем случае, поле название покупки зависит от полей «ID» клиента и «Имя клиента», если ID клиента на протяжении всей жизни БД остается неизменным, то поле «Имя клиента» по каким-то причинам может быть изменено. В случае его модификации необходимо предусмотреть процедуру перешифрования всех покупок на измененном значении поля «Имя клиента».

2.6 Дополнительный способ усиления защиты

Эффективным способом защиты является вынесение части вызовов Guardant API в дополнительные потоки. Это позволяет сделать защиту более гибкой, а также значительно затруднить анализ приложения взломщиком, ввиду небольшого числа отладчиков, способных работать с многопоточными приложениями.

В тестовом приложении в качестве демонстрации использования подхода создается один дополнительный поток. В нем создается хендл и с периодичностью в 5000 миллисекунд выполняется одно из нескольких действий: проверка вопрос-ответ, генерация, либо проверка электронной цифровой подписи, а также проверка заведомо неверной ЭЦП. При этом ошибки от попыток взлома не отображаются явным образом, а изменяют параметры главной формы, делая невозможной нормальную работу с приложением. Такое поведение затрудняет выявление методов Guardant API и анализ логики защиты, а также дает злоумышленнику ложную уверенность в том, что программа успешно взломана.

В реальном приложении для усиления защиты может создаваться множество дополнительных потоков, запускаемых и завершающихся в процессе работы приложения. Это затруднит анализ программы, а если данные из дополнительного потока будут использоваться в процессе работы основной программы, то также исключит возможность битхака. Например, создается поток, в котором какая-то переменная получает свое значение, позже эта переменная используется в вычислениях или функциях. В этом же потоке происходят вызовы Guardant API. Если злоумышленник попытается остановить/закрыть созданный поток, то получит некорректно работающую программу.

2.7 Заключение

Возможности аппаратных ключей защиты Guardant позволяют превратить программное обеспечение в неприступную крепость для взломщиков. Но строить данную «крепость» внутри программы разработчикам необходимо самостоятельно с помощью вызовов Guardant API. Чем глубже будет интеграция защиты в логику работы программы и чем более непредсказуемыми ее вызовы, тем тяжелее будет взломщику найти все защитные механизмы и уничтожить их. В большинстве случаев, после очередного раунда борьбы, он сдастся и бросит эту затею.

При усилении защиты не стоит забывать о самой программе – ее корректной работе, удобстве использования и производительности. Пользователь покупает, в первую очередь, качественную и надежную программу, которую легко использовать и получать стабильные результаты. При применении сложных методов защиты необходимо предусмотреть и корректно обработать максимальное количество возможных вариантов, при которых логика работы программы может быть нарушена или произойдет потеря важных данных.