Guardant®

Система защиты от компьютерного пиратства

Эффективная
защита приложений

Урок 3.3: Guardant API
Использование асимметричных алгоритмов

Содержание

Общее описание метода защиты 3

Контрольный пример 5

Реализация защиты 6

Заключение 9

Дополнительные источники информации 10

WWW: http://www.guardant.ru 10

Служба технической поддержки:.....10

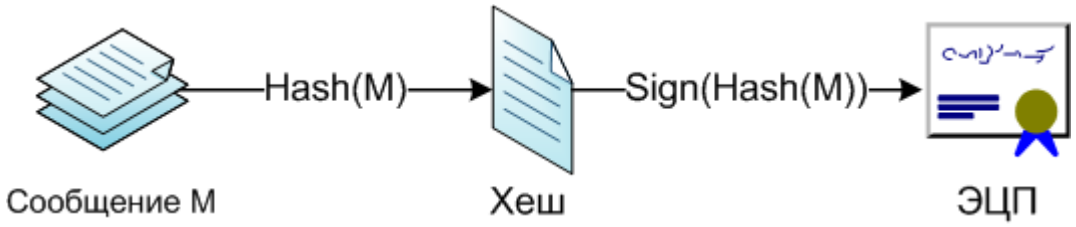
Общее описание метода защиты

Базовыми элементами маски ключа, с использованием которых могут быть организованы защитные механизмы приложения, являются:

- Симметричные алгоритмы шифрования
- Асимметричные алгоритмы шифрования
- Защищенные ячейки

В этом уроке речь пойдет об использовании **асимметричных алгоритмов**, а именно, алгоритма цифровой подписи ECC160, доступного во всех ключах поколения Guardant Sign/Time.

При помощи функции Guardant API **GrdSign()** можно подписать произвольные данные. Классическая схема, по которой происходит подпись, выглядит следующим образом:



Такой способ использования ЭЦП обусловлен **ограниченной длиной подписываемых данных**. Естественно, в случае небольшой длины сообщений и их высоких вероятностных характеристиках, использование алгоритма хеширования не является необходимым.

Обратимся к **классификации атак**, которые могут осуществляться на процедуры работы с алгоритмом цифровой подписи:

- Атаки на алгоритм генерации подписи
  - Модификация входных/выходных данных алгоритма подписи
  - Полная замена алгоритма генерации подписи
- Атаки на алгоритм проверки подписи
  - Изменение точки принятия решения о подлинности ЭЦП (т. н. бит-хак)
  - Модификация входных данных алгоритма проверки
  - Полная замена алгоритма проверки
- Атаки на функцию хеширования
  - Модификация входных/выходных данных алгоритма хеширования
  - Полная замена алгоритма подсчета хеш-суммы

Целью нарушителя является модификация процедур работы с алгоритмами ЭЦП электронного ключа так, чтобы обойти действие лицензионных ограничений.

Для защиты от **полной подмены** алгоритма подписи его статистическим аналогом, а также **модификации входных и выходных данных** алгоритмов генерации и проверки цифровой подписи предъявляются требования к **случайности** подписываемых данных, и **существенной зависимости** от них работы защищенного функционала в целом. При этом наиболее эффективной защитой от подмены/модификации алгоритма ЭЦП будет являться его прямое использование для подписи сообщений (хотя и может быть реализовано лишь в узком множестве приложений).

В качестве **источника энтропии** можно использовать:

- данные, поступающие с устройств ввода
- последовательность аппаратных/программных обращений к объектам
- текущее состояние таблиц баз данных или значения, получаемые по запросу пользователя
- значения системных счетчиков

Для **защиты от подмены данных** могут быть также реализованы простые проверки на отличие подписываемых данных, а также некоторых свойств, которым они неизбежно должны удовлетворять.

Защита от подмены и модификации **алгоритма проверки подписи** может быть реализована с использованием исходного кода алгоритма проверки (предоставляется в составе комплекта разработчика), при его объединении с кодом защищаемого функционала. С той же целью могут производиться проверки подлинности заведомо ложной ЭЦП.

Наиболее проблематичной является **защита функционала хеширования**, так как атака только на него потенциально компрометирует всю схему работы с ЭЦП. Рекомендуется по возможности использовать несколько реализаций алгоритма подсчета хеш-функций. Также возможна реализация простых проверок случайности получаемых хеш-сумм (выходом качественной хеш-функции обычно является последовательность с высокими вероятностными характеристиками).

Разработчиком могут быть созданы и реализованы любые другие проверки корректности работы процедур ЭЦП. Целесообразно также использование механизмов контроля целостности кода защищенного функционала (которые, отчасти, реализуются при последующем наложении автозащиты с соответствующими опциями).

Так как использование алгоритма ЭЦП по прямому назначению (к примеру, в составе инфраструктуры открытых ключей), в общем случае, не представляется возможным, будем рассматривать ситуацию с реализацией фиктивных обращений к алгоритму ЭЦП и проверок подписи, т. е. **подписывать и проверять подпись в рамках одного и того же приложения**.

Использование аппаратной подписи и программной проверки представляет собой мощный инструмент, который при грамотном применении позволяет построить защиту беспрецедентно высокого уровня. Для этого во время работы защищенной программы необходимо многократно подписать (**GrdSign()**) и программно проверить подпись (программная реализация **GrdVerifySign()**) от неких, отличающихся от раза к разу, данных. В этом случае нарушителю будет сложно связать факты генерации и проверки цифровой подписи особенно, если логика работы защитных механизмов подразумевает проверку как реальных, так и заведомо ложных подписей, а также генерацию вовсе неиспользуемых подписей.

В рамках данного урока будет подробно рассмотрен способ внедрения защиты на основе асимметричного алгоритма ECC160. Реализация механизмов защиты, в соответствии с указанными принципами, должна происходить в контексте конкретного приложения (в силу специфики защищаемого функционала).

Контрольный пример

На примере простого консольного приложения, реализующего выбор напитка, рассмотрим использование алгоритма цифровой подписи для защиты алгоритма принятия решения и реализации лицензионных ограничений. Выбор напитка происходит из вектора значений, сориентированного случайным образом. Ниже приведен листинг программы на языке C++:

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <ctime>

using namespace std;
ptrdiff_t random (ptrdiff_t i) { return (rand()% i);}
ptrdiff_t (*p_random)(ptrdiff_t) = random;

int main(int argc, char* argv[])
{
      vector<string> ans;
      string q="";
      srand ( unsigned ( time(NULL) ) );
      ans.push_back("Coffee");
      ans.push_back("Tea");
      random_shuffle(ans.begin(),ans.end(),p_random);
      cout << "============ Tea or Coffee ...============"<< endl;
      cout << "Heads or tails ? Enter you choice ( \"h\" or \"t\" ):";
      while ((q!="t") & (q!="h"))       cin >>q;
      cout << "Today your drink is ";
      if (q=="t")
            cout<< ans[0];
      else
            cout<< ans[1];
      cout << " !!!" << endl;
      cout << "=========================================="<< endl;
      return 0;
}
```

Реализация защиты

Будем защищать алгоритм принятия решения нашего тестового приложения. Для этого организуем подпись данных, вводимых пользователем.

При работе с алгоритмом ECC160 необходимо помнить, что он принимает на вход два массива, 20 и 40 байт: подписываемое сообщение и массив под подпись соответственно.

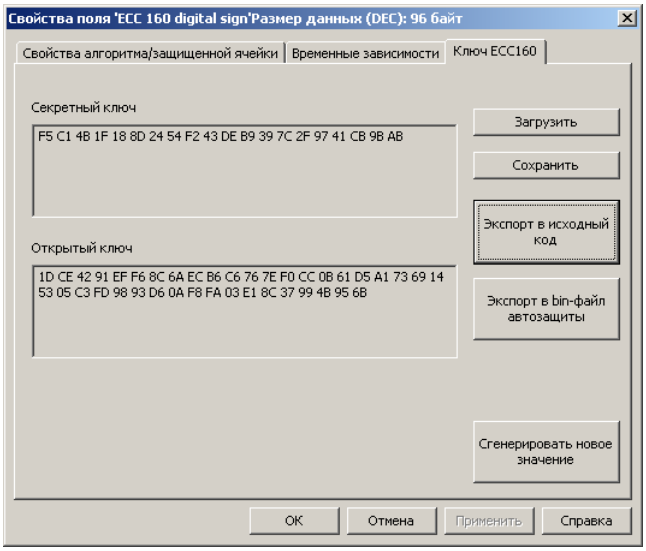
- Если при этом размер реальных данных более 20 байт, то необходимо подписывать хеш от блока данных.
- При использовании SHA256 (длина хеш-суммы равна 32 байтам), к примеру, в случае **Guardant Code**, подпись осуществляется первых 20 байт выхода хеш-функции.

Прежде всего, необходимо определиться с номером алгоритма ECC160 в маске ключа. Сделать это можно при помощи утилиты программирования ключей, работа с которой описана в **уроке 1.2**. Если алгоритм ECC160 в маске отсутствует, его необходимо добавить.

0716	0094	r w	Таблица лицензий 07	LMS table	5
0810	0096	r w	Алгоритм 08 (ECC160)	ECC 160 digital sign	F5 C
0906	0092	r w	Алгоритм 09 (AES128)	AES 128 Demo	AC 5

Этот номер будет передаваться функции **GrdSign()** для генерации ЭЦП. После внесения всех изменений, маску можно записать в ключ.

Далее необходимо получить открытый ключ алгоритма выбранного алгоритма для последующего его использования в процедуре проверки подписи. Для этого служит кнопка **Экспорт в исходный код** в свойствах определителя созданного алгоритма ECC160 маски ключа:



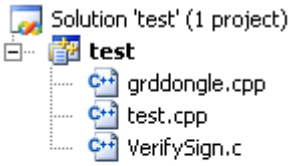
В результате будет получен следующий код, который необходимо добавить в защищаемое приложение любым из доступных способов:

```
BYTE      abyPublicKey[]=  "\x1D\xCE\x42\x91\xEF\xF6\x8C\x6A\xEC\xB6\xC6\x76\x7E\xF0
\xCC\x0B\x61\xD5\xA1\x73\x69\x14\x53\x05\xC3\xFD\x98\x93\xD6\x0A\xF\xFA\x03\xE1\x8C\
x37\x99\x4B\x95\x6B";
```

Ниже приведем код тестового приложения с добавленным функционалом генерации и проверки цифровой подписи:

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <ctime>
#include "grddongle.h"
#include "verifysign.h"

using namespace std;
ptrdiff_t random (ptrdiff_t i) { return (rand()% i);}
ptrdiff t (*p random)(ptrdiff t) = random;

BYTE
abyPublicKey[]="\x1D\xCE\x42\x91\xEF\xF6\x8C\x6A\xEC\xB6\xC6\x76\x7E\xF0\xCC\x0B\x61\xD5\xA
1\x73\x69\x14\x53\x05\xC3\xFD\x98\x93\xD6\x0A\xF8\xFA\x03\xE1\x8C\x37\x99\x4B\x95\x6B";

char    szMessage[GrdECC160 MESSAGE SIZE];        // Данные для подписи
char    szDigitalSign[GrdECC160 DIGEST SIZE];     // Цифровая подпись
string       q="";

int main(int argc, char* argv[])
{
      vector<string> ans;
      srand ( unsigned ( time(NULL) ) );

      int nRet = 0;

      CGrdDongle GrdDongle( GrdFMR Local );
      nRet = GrdDongle.Create(GrdDC DEMONVK, GrdDC DEMORDO, 0, 0);
      nRet = GrdDongle.Login( -1l );
      if(nRet)
            cout << "GrdLogin() failed!" << endl;
      else
      {
            ans.push back("Coffee");
            ans.push back("Tea");
            cout << "============ Tea or Coffee ...=============="<< endl;
            cout << "Heads or tails ? Enter you choice ( \"h\" or \"t\" ):";
            while ((q!="t") & (q!="h"))
            {
                  cin >>q;
                  // Подписываем значение пользовательского ввода
                  // Значение помешаем в серередину массива для подписи
                  szMessage[GrdECC160_MESSAGE_SIZE / 2] = *q.begin();
                  nRet = GrdDongle.Sign(0x0008,        // Номер алгоритма ECC160
                                        sizeof(szMessage), szMessage,
                                        sizeof(szDigitalSign), szDigitalSign, NULL );
                  if(nRet) cout << "GrdSign() failed!" << endl;
            }

            // Проверяем подлинность подписи
            nRet = GrdVerifySignSource( GrdECC160 PUBLIC KEY SIZE, abyPublicKey ,
                                        sizeof(szMessage), szMessage,
                                        sizeof(szDigitalSign), szDigitalSign, NULL );
            if(nRet)
                  cout << "GrdVerifySign() failed!" << endl;
            else
            {
                  // Алгоритм принятия решения о выборе напитка
                  random shuffle(ans.begin(),ans.end(),p random);
                  cout << "Today your drink is ";
                  if (q=="t")
                        cout<< ans[0];
                  else
                        cout<< ans[1];
                  cout << " !!!" << endl;
            }
            cout << "=========================================="<< endl;

      }
      return 0;
}
```

Для компиляции проекта необходимо добавить в состав проекта файлы с реализацией класса CGrdDongle и алгоритма проверки подписи:



Также необходимо указать пути к используемым заголовочным и объектным файлам компилятору и линкеру, соответственно (об этом говорилось в **уроке 3.1**).

Так как код процедуры проверки цифровой подписи доступен, рекомендуется объединить его с кодом алгоритма принятия решения о выборе напитка (но не с кодом вывода информации о результате выбора, так как работа с текстовыми строками наиболее уязвима к анализу защищенного приложения). Для этого, например, можно поместить часть функционала защищаемого приложения в состав алгоритма проверки ЭЦП так, чтобы без нее работа приложения была либо бессмысленна, либо некорректна.

Заключение

В ходе данного урока был рассмотрен один из возможных способов защиты функционала приложения при помощи аппаратного алгоритма ЭЦП. Представленный метод защиты основан на выполнении генерации и проверки цифровой подписи в рамках одного приложения.

При защите реального приложения предполагается создание большого числа изощренных механизмов контроля корректности схемы работы с ЭЦП, в том числе:

- Объединение кода функционала проверки подписи с защищаемым алгоритмом принятия решений,
- Использование элементарных проверок данных для защиты от модификации данных и/или подмены алгоритмов в схеме работы с ЭЦП
- А также любых других механизмов в соответствии с принципами, перечисленными в начале урока.

Использование алгоритма ЭЦП по прямому назначению значительно сокращает множество потенциальных атак на схему его использования. Также не стоит забывать, что на основе алгоритма ЭЦП могут быть эффективно реализованы, к примеру, процедуры получения платных обновлений защищенного приложения конечным пользователем.

Дополнительные источники информации

При возникновении вопросов, на которые вам не удалось найти ответа в этом пособии, рекомендуем обратиться к следующим дополнительным источникам информации:

WWW: http://www.guardant.ru

Web-сайт разработчика содержит большой объем справочной информации об электронных ключах Guardant.

Служба технической поддержки:

e-mail: hotline@guardant.ru

тел. +7(495)925-77-90