Однонаправленное преобразование (вычисление хэш-функции)

Наряду с симметричными преобразованиями электронные ключи могут вычислять однонаправленные функции (хэш-функции): HASH64 с секретным ключом длиной 128 или 256 бит (только для Guardant Sign/ Time / Net) и SHA256 с ключом 256 бит.
Такие преобразования характеризуются тем, что по результату преобразования практически невозможно восстановить исходные данные.
Наличие секретного ключа обеспечивает уникальность преобразования, что часто используется при защите данных от подделки.
Эти свойства однонаправленных функций широко используются для проверки подлинности данных, например, паролей, а также для вычисления надежных контрольных сумм, аутентификации и других задач. Поскольку в память электронных ключей можно записывать одинаковые дескрипторы аппаратных алгоритмов, однонаправленное преобразование можно использовать и для проверки подлинности данных, которыми обмениваются разные защищенные приложения или разные части одного и того же приложения.

Алгоритм хэширования SHA256

Алгоритм хэширования SHA256 представляет собой разновидность алгоритма типа SHA-2 (Secure Hash Algorithm v.2). Число 256 в названии алгоритма говорит о том, что длина результирующего хэша составляет 256 бит (32 байта).
Минимальная длина блока данных, хэш которых может быть вычислен, составляет 0 байт.
Если длина массива входных данных составляет менее 32 байт, к блоку данных по специальным правилам, оговоренным в описании стандарта, автоматически добавляются биты-заполнители. Разработчику защиты не нужно реализовывать этот алгоритм самостоятельно.
Более подробно об алгоритме SHA256 можно прочитать здесь: http://www.csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf

Алгоритм хэширования HASH64

Важная информация
Аппаратный алгоритм HASH64 есть только в ключах Guardant Sign/Time и их разновидностях. Он не реализован в ключах с загружаемым кодом (Guardant Code и его модификации).
HASH64 – это алгоритм вычисления 64-битной хэш-функции на основе блочного симметричного алгоритма с секретным ключом. Длина секретного ключа алгоритма HASH64 может составлять 16 или 32 байта (128 или 256 бит). Минимальная длина блока данных, хэш которых должен быть вычислен, составляет 16 байт. Длина блока данных должна быть кратной 8 байтам.
Основное свойство алгоритма HASH64 состоит в его однонаправленности. Это означает, что, даже зная значение хэш-функции для какого-либо произвольного набора данных, нельзя восстановить сами данные.
Алгоритм HASH64 используется для вычисления надежных контрольных сумм, аутентификации и проверки подлинности данных.
Если длина массива данных не кратна 8 байтам или составляет менее 16 байт, к блоку данных нужно добавить недостающие байты. Сильно желательно, чтобы байты-заполнители не были постоянными. В качестве байтов-заполнителей можно использовать случайные числа. В этом случае весь блок данных требуется хранить полностью, вместе с байтами-заполнителями (а не отбрасывать эти байты).

Рекомендации по работе с вектором инициализации IV

Для корректного преобразования данных алгоритмом HASH64 необходимо принимать во внимание следующее:

- Чтобы результаты вычисления хэша одних и тех же данных алгоритмами с одинаковыми определителями совпали, вектор инициализации IV должен иметь одинаковые значения;
- Необходимо сохранять значение вектора инициализации IV между обращениями к алгоритму при продолжении вычисления хэша больших блоков (больше 248 байт);
- При вычислении, к примеру, разных записей БД или секторов диска, рекомендуется инициализировать IV этим номером записи/сектора для того, чтобы вычисление хэша каждой из них выполнялось всегда одинаково, а разные записи с одинаковыми значениями имели разные значения хэш-функции.

Вычисление электронной цифровой подписи: алгоритм ECC160

ECC160 – алгоритм электронной цифровой подписи, в основе которого лежит математический аппарат эллиптических кривых. Этот алгоритм был специально разработан для использования в электронных ключах. Длина секретного ключа ECC160 составляет 20 байт (160 бит). Секретный ключ этого алгоритма хранится в определителе аппаратного алгоритма соответствующего типа. Минимальная длина блока данных, преобразуемых алгоритмом за один цикл, составляет 20 байт, при этом на выходе алгоритма получается блок данных, содержащий подпись, длиной 40 байт. Поскольку алгоритм является асимметричным, для проверки подписи используется открытый ключ длиной 40 байт (320 бит). Этот ключ должен храниться в защищенном приложении. Во избежание подмены не следует хранить открытый ключ в чистом виде (см. **Рекомендации программисту**).

Асимметричные функции типа ECC160 позволяют осуществлять электронную цифровую подпись данных на секретном ключе, который хранится в дескрипторе аппаратного алгоритма, и последующую проверку подписи на открытом ключе чисто программными методами. Проверка подписи позволяет убедиться в том, что подпись выполнена именно электронным ключом, поскольку к секретному ключу доступа нет. Применение асимметричных алгоритмов сильно затрудняет построение эмуляторов за счет того, что для проверки можно использовать не заранее известные данные, а любые, сгенерированные случайным образом в процессе работы приложения.

Таким образом, мы избавляемся от «классической» схемы «вопрос-ответ», свойственной симметричным и однонаправленным функциям, и подразумевающей использование полученных заранее пар исходных и преобразованных данных, на которых, собственно, и основаны табличные эмуляторы.

О том, как правильно использовать асимметричные алгоритмы, см. раздел Приложение А. **Рекомендации программисту**.

Генерация случайных чисел:алгоритм RND64

Важная информация

Аппаратный алгоритм RND64 есть только в ключах Guardant Sign/Time и их разновидностях. Он не реализован в ключах с загружаемым кодом (Guardant Code и его модификации).

RND64 представляет собой алгоритм генерации 64-битных случайных чисел на основе блочного симметричного алгоритма с секретным ключом. Длина секретного ключа алгоритма RND64 может составлять 16 или 32 байта (128 или 256 бит). Минимальная длина блока данных, инициализирующих генератор, составляет 8 байтов. Длина блока данных должна быть кратной 8 байтам.

Для получения случайных чисел при помощи алгоритма RND64 используется функция **GrdTransform** (или **GrdTransformEx**), при вызове которой необходимо указать в параметре **dwAlgoNum** числовое имя соответствующего алгоритма. Содержимое буфера **pData** на результат влияния не оказывает.

Загружаемый код(только Guardant Code / Code Time)

Электронный ключ с возможностью загрузки кода приложения представляет собой своего рода доверенную платформу, позволяющую производить значимые вычисления вне центрального процессора и оперативной памяти компьютера, на котором работает защищенное приложение.

Традиционно электронные ключи «умели делать» две основные вещи: а) хранить какие-либо данные, критичные для работы защищаемого приложения, б) выполнять внутри ключа алгоритмы, преобразующие данные по неким алгоритмам.

Эти алгоритмы, как правило, представляют собой либо симметричные алгоритмы шифрования, либо однонаправленные функции с секретным ключом преобразования.

Разработчики систем защиты, основанных на «традиционных» электронных ключах, зачастую испытывают сложности с выбором данных, которые можно было бы подвергать преобразованиям. Как правило, такие данные приходится генерировать искусственно, а не использовать поток реальных данных, на которых производятся вычисления.

В связи с этим существуют проблемы, связанные с тем, что преобразованные аппаратным алгоритмом данные, во-первых, далеко не всегда можно использовать напрямую в приложении. Во-вторых, нужно очень постараться, чтобы поток этих данных был разнообразным на протяжении достаточно длительного времени.

Защитный механизм загружаемого кода основан на том, что алгоритм, запрограммированный в ключ самим разработчиком, обрабатывает натуральные (естественные) данные, получаемые в процессе работы приложения. Обработанные данные можно использовать в приложении напрямую, исключая проверки валидности, которые, как правило, сводятся к одной-двум ассемблерным командам. Поток натуральных данных не постоянен и разнообразен. Поэтому алгоритм загружаемого кода с большой вероятностью будет производить вычисления на постоянно меняющихся данных, если алгоритм этот верно выбран и корректно реализован.