Электронный ключ (также аппаратный ключ, иногда донгл от англ. dongle) — аппаратное средство, предназначенное для защиты программного обеспечения (ПО) и данных от копирования, нелегального использования и несанкционированного распространения.

**Современные электронные ключи**

Основой данной технологии является специализированная микросхема, либо защищённый от считывания микроконтроллер, имеющие уникальные для каждого ключа алгоритмы работы. Донглы также имеют защищённую энергонезависимую память небольшого объёма, более сложные устройства могут иметь встроенный криптопроцессор (для аппаратной реализации шифрующих алгоритмов), часы реального времени. Аппаратные ключи могут иметь различные форм-факторы, но чаще всего они подключаются к компьютеру через USB. Также встречаются с LPT- или PCMCIA-интерфейсами.

Принцип действия электронных ключей. Ключ присоединяется к определённому интерфейсу компьютера. Далее защищённая программа через специальный драйвер отправляет ему информацию, которая обрабатывается в соответствии с заданным алгоритмом и возвращается обратно. Если ответ ключа правильный, то программа продолжает свою работу. В противном случае она может выполнять определенные разработчиками действия, например, переключаться в демонстрационный режим, блокируя доступ к определённым функциям.

Существуют специальные ключи, способные осуществлять лицензирования (ограничения числа работающих в сети копий программы) защищенного приложения по сети. В этом случае достаточно одного ключа на всю локальную сеть. Ключ устанавливается на любой рабочей станции или сервере сети. Защищенные приложения обращаются к ключу по локальной сети. Преимущество в том, что для работы с приложением в пределах локальной сети им не нужно носить с собой электронный ключ.

**Предыстория**

Защита ПО от нелицензионного пользования увеличивает прибыль разработчика. На сегодняшний день существует несколько подходов к решению этой проблемы. Подавляющее большинство создателей ПО используют различные программные модули, контролирующие доступ пользователей с помощью ключей активации, серийных номеров и т. д. Такая защита является дешёвым решением и не может претендовать на надёжность. Интернет изобилует программами, позволяющими нелегально сгенерировать ключ активации (генераторы ключей) или заблокировать запрос на серийный номер/ключ активации (патчи, крэки). Кроме того, не стоит пренебрегать тем фактом, что сам легальный пользователь может обнародовать свой серийный номер.

Эти очевидные недостатки привели к созданию аппаратной защиты программного обеспечения в виде электронного ключа. Известно, что первые электронные ключи (то есть аппаратные устройства для защиты ПО от нелегального копирования) появились в начале 1980-х годов, однако первенство в идее и непосредственном создании устройства, по понятным причинам, установить очень сложно.

**Защита ПО с помощью электронного ключа**

**Комплект разработчика ПО**

Донгл относят к аппаратным методам защиты ПО, однако современные электронные ключи часто определяются как мультиплатформенные аппаратно-программные инструментальные системы для защиты ПО. Дело в том, что помимо самого ключа компании, выпускающие электронные ключи, предоставляют SDK (Software Developer Kit — комплект разработчика ПО). В SDK входит все необходимое для начала использования представляемой технологии в собственных программных продуктах — средства разработки, полная техническая документация, поддержка различных операционных систем, детальные примеры, фрагменты кода, инструменты для автоматической защиты. Также SDK может включать в себя демонстрационные ключи для построения тестовых проектов.

**Технология защиты**

Технология защиты от несанкционированного использования ПО построена на реализации запросов из исполняемого файла или динамической библиотеки к ключу с последующим получением и, если предусмотрено, анализом ответа. Вот некоторые характерные запросы:

* проверка наличия подключения ключа;
* считывание с ключа необходимых программе данных в качестве параметра запуска (используется, в основном, только при поиске подходящего ключа, но не для защиты);
* запрос на расшифрование данных или исполняемого кода, необходимых для работы программы, зашифрованных при защите программы (позволяет осуществлять "сравнение с эталоном"; в случае шифрования кода, выполнение нерасшифрованного кода приводит к ошибке);
* запрос на расшифрование данных, зашифрованных ранее самой программой (позволяет отправлять каждый раз разные запросы к ключу и, тем самым, защититься от эмуляции библиотек API / самого ключа)
* проверка целостности исполняемого кода путём сравнения его текущей контрольной суммы с оригинальной контрольной суммой, считываемой с ключа (к примеру, путём выполнения ЭЦП кода или других переданных данных алгоритмом ключа и проверки этой ЭЦП внутри приложения; т.к. ЭЦП всегда разная - особенность криптографического алгоритма - то это также помогает защититься от эмуляции API/ключа);
* запрос к встроенным в ключ часам реального времени (при их наличии; может осуществляться автоматически при ограничении времени работы аппаратных алгоритмов ключа по его внутреннему таймеру);

и т.д.

Стоит отметить, что некоторые современные ключи (Guardant Code от Компании "Актив", LOCK от Astroma Ltd., Rockey6 Smart от Feitian, Senselock от Seculab) позволяют разработчику хранить собственные алгоритмы или даже отдельные части кода приложения (например, специфические алгоритмы разработчика, получающие на вход большое число параметров) и исполнять их в самом ключе на его собственном микропроцессоре. Помимо защиты ПО от нелегального использования такой подход позволяет защитить используемый в программе алгоритм от изучения, клонирования и использования в своих приложениях конкурентами. Однако для простого алгоритма (а разработчики часто совершают ошибку, выбирая для загрузки недостаточно сложный алгоритм) может быть проведен криптоанализ по методу анализа "черного ящика".

Как следует из вышесказанного, «сердцем» электронного ключа является алгоритм преобразования (криптографический или другой). В современных ключах он реализован аппаратно — это практически исключает создание полного эмулятора ключа, так как ключ шифрования никогда не передается на выход донгла, что исключает возможность его перехвата.

Алгоритм шифрования может быть секретным или публичным. Секретные алгоритмы разрабатываются самим производителем средств защиты, в том числе и индивидуально для каждого заказчика. Главным недостатком использования таких алгоритмов является невозможность оценки криптографической стойкости. С уверенностью сказать, насколько надёжен алгоритм, можно было лишь постфактум: взломали или нет. Публичный алгоритм, или «открытый исходник», обладает криптостойкостью несравнимо большей. Такие алгоритмы проверяются не случайными людьми, а рядом экспертов, специализирующихся на анализе криптографии. Примерами таких алгоритмов могут служить широко используемые ГОСТ 28147—89, AES, RSA, Elgamal и др.

**Реализация защиты с помощью функций API**

Помимо использования автоматической защиты, разработчику ПО предоставляется возможность самостоятельно разработать защиту, интегрируя систему защиты в приложение на уровне исходного кода. Для этого в SDK включены библиотеки для различных языков программирования, содержащие описание функциональности API для данного ключа. API представляет собой набор функций, предназначенных для обмена данными между приложением, системным драйвером (и сервером в случае сетевых ключей) и самим ключом. Функции API обеспечивают выполнение различных операций с ключом: поиска, чтения и записи памяти, шифрования и расшифрования данных при помощи аппаратных алгоритмов, лицензирования сетевого ПО и т. д.

Умелое применение данного метода обеспечивает высокий уровень защищённости приложений. Нейтрализовать защиту, встроенную в приложение, достаточно трудно вследствие её уникальности и «размытости» в теле программы. Сама по себе необходимость изучения и модификации исполняемого кода защищенного приложения для обхода защиты является серьёзным препятствием к её взлому. Поэтому задачей разработчика защиты, в первую очередь, является защита от возможных автоматизированных методов взлома путём реализации собственной защиты с использованием API работы с ключами.

**Обмен данными между программой и электронным ключом**

Основным способом построения надёжной защиты является использование библиотеки API для работы с электронным ключом. Как правило, API поставляется в виде статической и динамической библиотеки.

Статические библиотеки, в отличие от динамических, присоединяются (линкуются) к исполняемой программе в процессе сборки. Их использование наиболее предпочтительно, т.к. исключает возможность простой подмены файла. Далее будем рассматривать защиту приложений, использующих именно статическую библиотеку.

Программа обменивается данными с электронным ключом через библиотеку API, которая напрямую взаимодействует с драйвером электронного ключа. Типовая схема обмена между защищённой программой и электронным ключом изображена на рис.1.

Атаки в такой схеме нацелены на взаимодействие между различными модулями защиты. Перехват запросов к электронному ключу на уровне драйвера электронного ключа (1) или драйвера USB-шины (2) не требует повторной модификации каждой новой версии приложения, в отличие от варианта с перехватом вызовов статической библиотеки API (3).

**Обход защиты**

Задача злоумышленника — заставить защищённую программу работать в условиях отсутствия легального ключа, подсоединённого к компьютеру. Не вдаваясь очень глубоко в технические подробности, будем исходить из предположения, что у злоумышленника есть следующие возможности:

* перехватывать все обращения к ключу;
* протоколировать и анализировать эти обращения;
* посылать запросы к ключу и получать на них ответы;
* протоколировать и анализировать эти ответы;
* посылать ответы от имени ключа и др.

Такие широкие возможности противника можно объяснить тем, что он имеет доступ ко всем открытым интерфейсам, документации, драйверам и может их анализировать на практике с привлечением любых средств.

Для того чтобы заставить программу работать так, как она работала бы с ключом, можно или внести исправления в программу (взломать её программный модуль), или эмулировать наличие ключа путём перехвата вызовов библиотеки API обмена с ключом.

Стоит отметить, что современные электронные ключи (к примеру, ключи Guardant поколения Sign и современные ключи HASP HL) обеспечивают стойкое шифрование протокола обмена электронный ключ -- библиотека API работы с ключом. В результате наиболее уязвимыми местами остаются точки вызовов функций этого API в приложении и логика обработки их результата.

**Атака на драйвер электронного ключа**

Данный вид атаки является наиболее простым. Оригинальный драйвер электронного ключа заменяется на драйвер-эмулятор. Данные, передаваемые в ключ и возвращаемые обратно, перехватываются и сохраняются в файле на диске. Затем оригинальный ключ извлекается из компьютера и программа начинает взаимодействовать с эмулятором, продолжая искренне верить в то, что общается с ключом.

Переходя к частностям, хотелось бы остановиться подробнее на защите от программных эмуляторов в электронных ключах Guardant. Драйверы электронных ключей Guardant содержат электронную подпись (ЭП). При вызове функций Guardant API защищенное приложение автоматически проверяет подпись драйвера в оперативной памяти. Поскольку закрытый ключ не известен, создать драйвер-эмулятор с правильной подписью невозможно. Убрать проверку мешает виртуальная машина (псевдокод), с помощью которой защищены исполняемые файлы драйверов и библиотеки Guardant API. Схема проверки изображена на рис.2.

**Атака на драйвер USB-шины**

Создание такого эмулятора потребовало изучение протокола обмена между драйвером электронного ключа и драйвером USB-шины. Как это ни парадоксально звучит, но в такой ситуации более надёжными оказались LPT-ключи, т.к. драйвер электронного ключа напрямую взаимодействует с LPT-ключом через порты ввода/вывода компьютера, минуя промежуточный драйвер.

К сожалению, полностью избавиться от программных эмуляторов на уровне USB-шины, используя ключи с симметричной криптографией, невозможно. Тем не менее, хорошая защита, построенная на постоянном обмене с электронным ключом, может потребовать не один день для записи всех возможных посылок и ответов к ключу и сработать у нелегального пользователя в самый неподходящий момент. Взломанные программы, перестающие работать по непонятным причинам, лишь тому подтверждение.

Отдельно хочется упомянуть о защитах, когда разработчики ограничиваются простой проверкой наличия электронного ключа. Это грубая ошибка. Используя дизассемблер, “независимость” такой программе можно подарить за 15 минут.

В электронных ключах с асимметричной криптографией обмен данными между защищённой программой и электронным ключом шифруется на сеансовых ключах. По этой причине единственно возможным является третий вариант атаки.

**Эмуляция ключа**

При эмуляции никакого воздействия на код программы не происходит, и эмулятор, если его удается построить, просто повторяет все поведение реального ключа. Эмуляторы строятся на основе анализа перехваченных запросов приложения и ответов ключа на них. Они могут быть как табличными (содержать в себе все необходимые для работы программы ответы на запросы к электронному ключу), так и полными (полностью эмулируют работу ключа, так как взломщикам стал известен внутренний алгоритм работы).

Построить полный эмулятор современного электронного ключа — это достаточно трудоёмкий процесс, требующий большого количества времени и существенных инвестиций. Ранее злоумышленникам это удавалось: например, компания Aladdin признаёт, что в 1999 году злоумышленникам удалось разработать довольно корректно работающий эмулятор ключа HASP3 и HASP4. Это стало возможным благодаря тому, что ключ использовал проприетарный алгоритм кодирования, который был взломан. Сейчас большинство ключей используют публичные криптоалгоритмы, поэтому злоумышленники предпочитают атаковать какой-то конкретный защищённый продукт, а не защитный механизм в общем виде. Для современных систем защиты HASP и Guardant эмуляторов в свободном доступе нет, так как используется криптосистема с открытым ключом.

Информации о полной эмуляции современных ключей Guardant не встречалось. Существующие табличные эмуляторы реализованы только для конкретных приложений. Возможность их создания была обусловлена неиспользованием (или неграмотным использованием) основного функционала электронных ключей разработчиками защит.

Так же отсутствует какая-либо информация о полной или хотя бы частичной эмуляции ключей LOCK, либо о каких-либо других способах обхода этой защиты.