Электронные ключи. Применение и уязвимость.

Выполнил студент 4-ого курса ФРТК

Шостак Николай Петрович

Группа 912

Предисловие.

Ранние образцы защиты от компьютерного пиратства, появившиеся еще в начале 80-х годов, были откровенно примитивны. Их разработчики ограничивались использованием чисто программных приемов проверки легальности копии ПО и борьбы с нелегальным копированием (соответственно, и сама защита называлась программной). Как правило, репутация такой защиты была ужасна, пользователи защищенного софта были вынуждены мириться с постоянными “зависаниями” системы, серьезными проблемами при инсталляции защищенного ПО... Не дремали и пираты — они быстро научились “ломать” программную защиту. Разработчики начали усложнять ее принципы — так появились дискеты, царапанные иголкой или прожженные лазером, дискеты с программно записанной “некопируемой” меткой, защита, анализирующая временные характеристики компьютера или “привязывающая” программы к неким уникальным характеристикам ПК...

Однако упорный труд разработчиков не принес желаемого результата: программная защита оставалась ненадежной, капризной и очень неудобной в работе. А метания ее разработчиков в поисках удачных технологий привели к тому, что в персональном компьютере быстро не осталось ни одного “укромного уголка”, в который можно было бы спрятать ключевую информацию. Таким образом, век программной защиты подошел к концу. В мире созрели объективные предпосылки для создания принципиально новой защиты — надежной и удобной для пользователя и в то же время — стойкой ко взлому. И в середине 80-х годов она была создана. С появлением программно-аппаратной защиты на основе электронных ключей была открыта новая страница истории защиты.

Электронный ключ – это…

аппаратная часть системы защиты, представляющая собой плату с микросхемами памяти и, в некоторых случаях, микропроцессором, помещенную в корпус и предназначенную для установки в один из стандартных портов ПК (LPT, USB ...) или слот расширения материнской платы.

Электронные ключи по архитектуре можно подразделить на ключи с памятью (без микропроцессора) и ключи с микропроцессором (и памятью).

Наименее стойкими (в зависимости от типа программной части) являются системы с аппаратной частью первого типа. В таких системах критическая информация (ключ дешифрации, таблица переходов) хранится в памяти электронного ключа. Для дезактивации таких защит в большинстве случаев необходимо наличие у злоумышленника аппаратной части системы защиты (основная методика: перехват диалога между программной и аппаратной частями для доступа к критической информации).

Самыми стойкими являются системы с аппаратной частью второго типа. Такие комплексы содержат в аппаратной части не только ключ дешифрации, но и блоки шифрации/дешифрации данных, таким образом при работе защиты в электронный ключ передаются блоки зашифрованной информации, а принимаются оттуда расшифрованные данные. В системах этого типа достаточно сложно перехватить ключ дешифрации так как все процедуры выполняются аппаратной частью, но остается возможность принудительного сохранения защищенной программы в открытом виде после отработки системы защиты. Кроме того, к ним применимы методы криптоанализа.

Технологии и гарантии.

Ключи с энергонезависимой памятью – еще один шаг вперед.

Все известные производители электронных ключей оснащают свои изделия встроенной энергонезависимой памятью (как правило, EEPROM — ПЗУ с электрической записью и стиранием). Эта память может хранить данные десятки лет, не требуя при этом никаких источников питания. Ее объем может достигать 4 Мбит, хотя в электронных ключах обычно ограничиваются 512-ю байтами. Кроме того, такую память можно перепрограммировать буквально “на ходу”, для этого не требуется никаких дополнительных устройств.

Использование EEPROM-памяти в электронных ключах произвело эффект, которого их разработчики, наверное, и не ожидали. Ключи перестали быть только лишь средством защиты от пиратства. Они превратились в устройства, помогающие проводить в жизнь маркетинговую политику разработчиков ПО. С их помощью стало возможно создавать защищенные upgrade и демо-версии, организовывать прокат, аренду, лизинг ПО, решить проблему лицензирования сетевых приложений и т. д., и т. п. Для максимального удобства в работе с конечными пользователями была введена возможность “дистанционного” изменения записанных в ключ данных. Теперь стало возможно превращать демо-версии в рабочие, производить upgrade, продлевать ресурс использования защищенного софта буквально по телефону! Появились специальные ключи с микросхемой таймера, при помощи которых можно ограничить время использования защищенных программ.

Несколько алгоритмов в одном ключе.

Основную логику в ключе, как правило, реализует *микроконтроллер*, либо специализированная заказная микросхема - *ASIC (Application Specific Integrated Circuit)*

С помощью ASIC-чипа в ранних ключах аппаратно реализовывалась функция Y=F(X), с помощью которой защищенная программа получает в ответ нечто более внятное, чем “да / нет”. Но производители, в угоду растущим запросам рынка ПО, стремятся повысить стойкость своих изделий, реализую не одну, а несколько функций. Это очень усложняет ASIC-чипы, и некоторые производители переходят на микроконтроллеры. Вообще говоря, микроконтроллер является более гибким устройством, позволяя реализовывать более сложную логику.

Он, например, является основным конструктивным элементом, так называемых, Stealth-ключей. На этапе изготовления ключа в память его микроконтроллера записывается микропрограмма. В сочетании с дескрипторами, она реализует множество различных функций вида Y=F(X), обладающих высокой сложностью и нелинейностью. Эти функции называются также аппаратными алгоритмами. Микроконтроллер представляет собой «черный ящик», полностью скрывающий все происходящие в нем процессы. Поэтому микропрограмму невозможно ни считать, ни модифицировать.

При помощи аппаратных алгоритмов можно производить кодирование любой информации, жизненно важной для защищенного приложения. При правильной организации системы защиты, использование аппаратного алгоритма делает бессмысленным удаление из тела приложения вызовов функций API: в этом случае попросту некому будет декодировать нужные приложению данные. Кроме того, сам факт

наличия аппаратных алгоритмов настолько усложняет логику работы электронных ключей, что создание их эмулятора превращается в весьма нетривиальную задачу.

В одном Stealth-ключе при желании можно организовать до 18 различных аппаратных алгоритмов. Естественно, для защиты одной и той же программы можно использовать все созданные в ключе алгоритмы: например, какая-то часть данных кодируется 1-м алгоритмом, какая-то – 2-м и т.д. При прочих равных условиях это еще более усложняет задачу взлома защиты.

Сложность аппаратных алгоритмов

Каждый аппаратный алгоритм, созданный в Stealth-ключе, полностью описывается своим дескриптором. Дескрипторы записываются в энергонезависимую память Stealth-ключа и защищаются от чтения и модификации. Часть дескриптора описывает свойства алгоритма (к их описанию мы еще вернемся). Вторая его часть представляет собой определитель аппаратного алгоритма. Он играет важнейшую роль в реализации конкретного алгоритма.

Определитель – это числовая последовательность, которая в сочетании с микропрограммой участвует в образовании аппаратного алгоритма ключа.

Длина определителя может достигать 200 байт или 1600 бит. Нетрудно подсчитать, что для подбора такого алгоритма хакеру придется перебрать до 2^{1600} различных комбинаций. Взяв минимальное время работы такого алгоритма, получим, что для его подбора хакеру потребуется 10475... тысячелетий. И даже если алгоритм будет иметь определитель с минимально возможной длиной (4 байта или 32 бита), для его подбора потребуется 1051 год.

Если сравнивать Stealth-ключи с любыми другими, то на первый взгляд может показаться, что такая стойкость алгоритмов не нужна – она явно избыточна. Но так ли это на самом деле?

Во-первых, злоумышленник может и не обращаться всякий раз к электронному ключу. Если он будет иметь некий массив данных и в кодированном, и в исходном виде, то попытается подобрать алгоритм, по которому был закодирован этот массив, целиком опираясь на вычислительные возможности своего компьютера. Тогда время подбора алгоритма станет обратно пропорциональным быстродействию компьютера, и будет на несколько порядков меньше, рассчитанного нами. Более того. Согласно закону Мура, вычислительная мощность процессоров удваивается каждые 18 месяцев. Получается, что каждые полтора года как бы «съедается» один разряд любого ключа шифрования (сегодня на подбор N+1 разряда кода нужно столько же времени, сколько полтора года назад – на подбор N разрядов). Поэтому алгоритмы, вполне надежные сегодня, уже через несколько лет могут превратиться в ничто. Следовательно, для того чтобы алгоритм оставался надежным и спустя годы, сегодня он должен быть заведомо избыточным – то есть иметь большой запас стойкости.

Во-вторых, при подборе любого кода есть вероятность, что злоумышленнику попросту повезет, и он сумеет подобрать его неожиданно быстро.

В-третьих, в последнее время становятся популярными т.н. «Internet-взломы». Множество «заинтересованных лиц» (от сотен до многих тысяч) занимаются подбором одновременно, синхронизируя свои действия при помощи Internet. В этом случае время подбора уменьшается еще соответственно в сотни – тысячи раз. Только алгоритм с избыточной стойкостью способен выдержать подобные атаки. Таким образом, избыточность аппаратных алгоритмов, заложенная в технологии Stealth, не только оправдана, но и совершенно необходима.

Уникальность аппаратных алгоритмов для каждого пользователя.

Каждый пользователь защиты получает электронные ключи с уникальными кодами доступа, которые никогда не повторяются, и с несколькими созданными по умолчанию аппаратными алгоритмами, вид определителя которых также уникален. Следовательно, аппаратные алгоритмы в ключах, предназначенных для одного пользователя защиты, будут кодировать одни и те же данные по-иному, нежели у любого другого пользователя той же системы защиты. Это гарантия того, что злоумышленнику не удастся создать универсальный эмулятор работы всех Stealth-ключей.

Кроме того, пользователи имеют возможность конструировать свои собственные аппаратные алгоритмы. При помощи специальной утилиты, поставляемой производителем, они могут задавать свойства своего аппаратного алгоритма, конструировать его определитель и т.д. В результате электронный ключ станет кодировать данные по алгоритму, конкретный вид и свойства которого будут известны только пользователю защиты. Если некая компания производит несколько программных продуктов, можно каждый из них защищать с использованием своего уникального аппаратного алгоритма. Или вообще в каждом электронном ключе создавать свой уникальный алгоритм – и тогда каждая копия вашей программы будет защищена уникальным методом. Таким образом, используя свои уникальные аппаратные алгоритмы, вы полностью исключите вероятность создания универсальных эмуляторов для ваших программных продуктов (или для разных версий одного вашего продукта).

Свойства аппаратных алгоритмов.

Разработчики Stealth-технологии анонсируют следующие свойства аппаратных алгоритмов, позволяющие увеличить стойкость защиты.

Зависимость аппаратного алгоритма от идентификационного номера ключа. При активизации этого свойства соответствующий аппаратный алгоритм в каждом из ваших ключей будет преобразовывать данные уникальным образом, даже если определители таких алгоритмов будут одинаковыми во всех ваших ключах. Помимо всего прочего, вид такого алгоритма будет зависеть от идентификационного номера (ID) ключа – уникального 8-байтового значения, которое записывается в каждый электронный ключ на этапе его производства.

Зависимость аппаратного алгоритма от значения его счетчика. Воспользовавшись этим свойством, характер преобразования данных алгоритмом можно сделать зависящим от значения, присвоенного счетчику алгоритма (оно заносится в специальное 4-байтовое поле, входящее в состав дескриптора). Поэтому, если счетчики таких алгоритмов будут иметь разные значения, то и данные они будут преобразовывать по-разному – при прочих равных условиях.

Ограничение количества запусков аппаратного алгоритма. При активизации этого свойства аппаратный алгоритм может быть запущен на выполнение только заданное вами число раз. Счетчик алгоритма записывается в специальное 4-байтовое поле (32 бит), входящее в состав дескриптора. При каждом запуске такого аппаратного алгоритма значение счетчика будет уменьшаться на 1, а по достижении им нулевого значения алгоритм перестанет запускаться. Алгоритмы с этим свойством – прекрасный способ получения защищенных программ с ограниченным сроком «жизни» (например, демо-версий).

Свойства аппаратного алгоритма можно комбинировать. Например, при активизации 2-го и 3-го свойств получится «плавающий» алгоритм, при каждом запуске преобразующий данные по-разному. Исходя из размерности счетчика алгоритма, общее количество различных вариантов преобразования для данного конкретного алгоритма равно 2^{32}. Использование аппаратного алгоритма с таким набором свойств позволяет достичь

максимально возможного уровня защищенности от эмуляции электронного ключа. Такой алгоритм можно использовать и как генератор псевдослучайных чисел разрядностью до 255 байт (в этом случае ответы «плавающего» алгоритма рассматриваются как случайные числа).

Кодирование массива данных при помощи аппаратных алгоритмов.

До сих пор в основном рассматривались свойства Stealth-ключей, блокирующие попытки злоумышленника изучить внутреннюю структуру электронного ключа, «отгадать» его аппаратные алгоритмы и построить на этой базе некий универсальный эмулятор. Однако при создании эмулятора можно пойти и другим путем. Напомним, что ключ работает по следующей схеме: от защищенной программы к нему поступает некий блок данных (вопрос к ключу), и он при помощи аппаратного алгоритма преобразует (кодирует или декодирует) эти данные. Так получается ответ ключа, посылаемый защищенной программе. Так как длина блока данных, который может преобразовать алгоритм ключа за один сеанс работы, в общем случае не бесконечна, то понятно, что существует и конечное количество комбинаций. То есть, вопросов к ключу, так же как и его ответов, не может быть бесконечно много. Например, если алгоритм некоего электронного ключа может преобразовать только 2 байта (или 16 бит) данных одновременно, то общее число вопросов к такому ключу (равно как и его ответов) равно 2^{16} или 65536.

Этим часто и пользуются злоумышленники. «Обстреливая» электронный ключ различными значениями, они получают верные ответы на все возможные вопросы к ключу. Затем составляется таблица, в которой каждому из возможных вопросов соответствует верный ответ ключа. Эта таблица используется в эмуляторе для генерации ответов псевдоключа. В нашем примере достаточно будет массива, состоящего из 65536 двухбайтовых элементов (это массив всех возможных ответов ключа). При этом уже не нужно определять конкретный вид аппаратного алгоритма – достаточно знать, какой ответ Y соответствует вопросу X.

Любой из аппаратных алгоритмов Stealth-ключа за один сеанс способен обработать до 255 байт (или 2040 бит) данных. Причем алгоритмы устроены таким образом, что длина шифрующей последовательности равна длине шифруемой. Иными словами, на шифруемую последовательность длиной N байт накладывается нециклическая шифрующая последовательность длиной N байт.

Помимо аппаратных алгоритмов, можно использовать для защиты алгоритм быстрого преобразования данных. Эта возможность позволяет производить кодирование сверхбольших объемов данных с такой высокой скоростью, что сам процесс будет совершенно незаметен для пользователя защищенного приложения.

Методы снятия защиты.

Создание аппаратной копии электронного ключа.

С помощью специальных программных или аппаратных средств считывают содержимое микросхемы памяти электронного ключа и переносят эту информацию в аналогичные микросхемы памяти других электронных ключей. Таким образом, создаются аппаратные копии электронного ключа, пригодные для работы с защищенной программой.

Создание аппаратных копий ключа применимо только в тех случаях, когда известен конкретный тип микросхемы памяти, когда имеется программатор для записи данных в микросхему памяти ключа-копии и, самое главное,— при условии, что память электронного ключа никак не защищена от ее нелегального считывания. Именно это третье условие и стало слабым местом некоторых марок электронных ключей, именно слабость (а то и полное отсутствие) защиты данных в памяти ряда ключей обусловило распространение и успешное применение этого метода взлома.

Методу имеет несколько недостатков. Главный из них в том, что, даже получив нелегальную копию программы, не удастся решить задачу ее незаконного распространения — ведь копия так и осталась защищенной, просто теперь она “привязана” к “левому” электронному ключу. Кроме этого, достаточно велика стоимость такого копирования — зачастую дешевле просто купить легальную копию продукта. Поэтому такой метод не получил широкого распространения.

Создание эмулятора электронного ключа.

Изучив внутреннюю логику работы электронного ключа, создается некий программный модуль — резидентная программа, DLL, драйвер и т.п., — который эмулирует (то есть воспроизводит) работу электронного ключа. Качественно написанный эмулятор способен воспроизвести работу ключа во всех тонкостях, включая особенности протокола обмена между защищенной программой и ключом, генерацию им верных ответов и т.д. В результате защищенная программа будет спокойно работать, даже не подозревая о подмене.

По принципу организации эмуляторы можно разделить на эмуляторы структуры ключа (как правило, это универсальные эмуляторы ключей без аппаратных алгоритмов либо ключей с примитивно простыми аппаратными алгоритмами) и эмуляторы ответов электронных ключей. При этом первые воспроизводят структуру электронного ключа во всех ее тонкостях, что и обусловливает их универсальность, а вторые используют специально созданную таблицу верных ответов ключа.

Но в любом случае эмуляторы должны верно воспроизводить не только посылаемые ключом данные, но и протокол обмена с ним (в противном случае защищенная программа не сможет "понять" передаваемые эмулятором данные).

Эмуляторы электронных ключей — это мощный и эффективный метод взлома программно-аппаратной защиты, получивший достаточно широкое распространение. В настоящее время разработчики электронных ключей используют следующие методы борьбы с программной эмуляцией.

Отделение модуля автоматической защиты.

Метод автоматической защиты готовых программ (его еще называют методом envelope или "конвертом" защиты) доступен практически во всех системах программно-аппаратной защиты. Его преимущества: при помощи специальной утилиты в считанные секунды можно "повесить" защиту на любую готовую программу. Экономится масса времени, которого всегда не хватает, не требуется никаких навыков в области построения защиты, а порой этот метод вообще является единственно возможным (например, если по каким-либо причинам недоступны исходные тексты защищаемых программ). Однако у этого метода есть слабое место. Так как защита устанавливается на готовую программу, она не может образовать с программной неразрывного целого. Модуль автоматической защиты как бы "приклеивается" к программе, а это значит, что есть теоретическая возможность отделить, "отклеить" его — причем без ущерба для самой программы. Да и опыт в этом плане накоплен большой, есть даже специальные утилиты автоматического взлома программ, защищенных таким методом.

Абсолютно надежного способа противодействия взлому автоматической защиты не существует, и ее взлом — это лишь вопрос времени. А время взлома такой защиты, в свою очередь, зависит от ее качества. В наибольшей степени страдает автоматическая защита тех систем, разработчики которых не учли уже используемых специфических приемов

взлома. Но существует и весьма надежная автоматическая защита, которая не поддается "стандартным" утилитам автоматического взлома и вынуждает выполнять большую часть работы вручную. Соответственно возрастает стоимость взлома, и в ряде случаев оказывается более выгодным приобрести легальную копию программы.

Другой действенный метод спасти защиту от быстрого взлома — это комбинирование автоматической защиты с защитой при помощи функций API. В этом случае, даже "отсоединив" модуль автоматической защиты, еще не получим незащищенной программы — во многих местах ее тела останутся нетронутыми вызовы функций API защиты.

Удаление вызовов функций API.

Помимо автоматической защиты, в распоряжение пользователя предоставлен набор функций API защиты. Как правило, API существуют для всех популярных языков программирования и содержат самые разнообразные функции. Пользователь защиты должен вписать вызовы нужных функций в исходный текст защищаемой программы, скомпилировать и слинковать его с соответствующим объектным модулем API. В результате защита будет внедрена глубоко в тело программы, и при правильной организации логики своей работы будет составлять с программой неразрывное целое. Взломать защиту такого типа гораздо сложнее, чем автоматическую — хотя бы потому, что невозможно создать универсальное средство автоматического взлома. При помощи дисассемблеров и отладчиков изучается логика работы программы, находятся места, в которых происходят вызовы функций API (либо точки входа в сами эти функции — смотря что оказывается проще), и нужным образом исправляется программный код. Для борьбы с этим методом взлома есть несколько испытанных способов.

Комбинирование защиты при помощи функций API с автоматической. Хорошая автоматическая защита умеет защищать программу и от дисассемблера, и от отладчика. Следовательно, для того чтобы использовать весь инструментарий в полную силу, придется сначала "снять" модуль автоматической защиты. Кстати, именно поэтому все разработчики защиты рекомендуют комбинировать оба метода. Модуль автоматической защиты должен образовывать "внешний уровень обороны", защищая программу от дилетантов и скрывая от любопытных глаз ядро защиты, реализованное функциями API.

Усложнение логики работы самих модулей защиты. Профессиональные системы защиты имеют в своем арсенале много внутренних методов борьбы с изучением логики их работы. Кодирование тела модулей автоматической защиты и библиотек API, "зашумление" фрагментов их кода, подсчет контрольных сумм важных участков кода — эти и многие другие меры способны существенно усложнить процесс исследования логики работы самих модулей защиты.

Разработка нетривиальной логики взаимодействия программы и защиты. Не приходится рассчитывать на приемлемый уровень защищенности программы, если информация, возвращаемая функциями API, используется программой для банального сравнения: "верно / неверно". Есть множество эффективных приемов усложнения логики работы программы с модулями защиты. Откладывание реакции программы на информацию, полученную от ключа, использование ответных данных ключа в качестве индексов каких-либо массивов, подсчет контрольных сумм данных с использованием значений, полученных от ключа и т.д.— это лишь наиболее простые из таких приемов. Воспользовавшись ими и им подобными, можно создать защищенную программу, разобраться в логике работы которой будет уже сложнее. В результате шансы удалить из программы все вызовы функций API без последствий для самой программы станут минимальными. "Остатки" защиты постоянно будут проявляться во взломанной

программе, ограничивая ее возможности и нарушая нормальный ход ее работы. Такую программу можно рассматривать лишь как демонстрационную — ведь пользователь не сможет использовать все ее возможности и рано или поздно будет вынужден приобрести легальную копию такой программы.

Использование ответов аппаратных алгоритмов в работе программы. На этапе установки защиты важные для программы данные можно закодировать при помощи аппаратного алгоритма ключа и в таком виде записать их в тело программы. В процессе работы программа в нужные моменты будет запускать аппаратный алгоритм, декодирующий эти данные, а затем использовать их по назначению. При такой схеме организации защиты даже полное удаление из программы всех обращений к функциям API ничего не дает. Ведь защищенная программа содержит свои важные данные в кодированном виде, и раскодировать их, не имея электронного ключа, нельзя.

Выводы.

Плюсы и перспективы очевидны:

1. Значительное затруднение нелегального распространения и использования ПО;
2. Избавление производителя ПО от разработки собственной системы защиты;
3. Высокая автоматизация процесса защиты ПО;
4. Наличие API системы для более глубокой защиты;
5. Возможность просто проводить маркетинговую политику;
6. Достаточно большой выбор таких систем на рынке;

Минусы могут показаться надуманными и несущественными, но могут доставить неприятности:

1. Затруднение разработки и отладки ПО из-за ограничений со стороны средств защиты;
2. Дополнительные затраты на приобретение системы защиты и обучение персонала;
3. Замедление продаж из-за необходимости физической передачи аппаратной части;
4. Повышение системных требований из-за защиты (совместимость, драйверы);
5. Снижение отказоустойчивости ПО;
6. Несовместимость систем защиты и системного или прикладного ПО пользователя;
7. Несовместимость защиты и аппаратуры пользователя;
8. Снижение расширяемости компьютерной системы;
9. Угроза кражи аппаратного ключа

Список использованных материалов

1. Электронные ключи Guardant

http://www.novex.ru/products/stealth.htm

2. Защита программных продуктов

http://isoft.com.ru/secy/26.shtml

3. Aladdin Software Security R.D.

http://www.aladdin.ru/

4. Энергонезависимая память EEPROM

http://www.cec-mc.ru/comp/memory/eeprom.shtml