Эссе по курсу "Защита информации"
кафедра радиотехники
Московский физико-технический институт (ГУ МФТИ)
http://www.re.mipt.ru/infsec

Алгоритм аутентификации RSA SecurID.

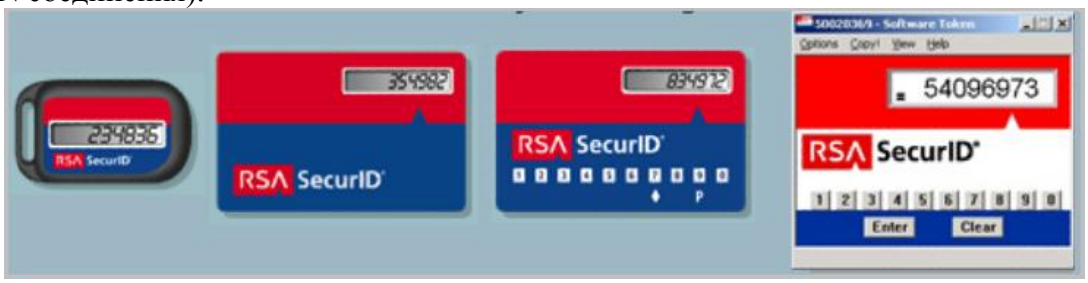
Кирющенко Вячеслав Сергеевич
vsk@land.ru

2005, г.Долгопрудный.

Содержание:

1. Введение

RSA securID token (ключ, жетон) в настоящий момент базируется на собственном алгоритме и предоставляет 6- или 8-значные последовательности цифр на выходе. Выходом является псевдослучайная последовательность, меняющаяся с интервалом 30 или 60 секунд. Ключ состоит из 4-битного микропроцессора работающего, обычно, на частоте 1 МГц. Принимая в расчёт эти вычислительные возможности, использование 64-битного значения времени и 64-битного секретного ключа оказалось хорошим выбором для защищённого проведения аутентификации(например, при использовании VPN соединения).

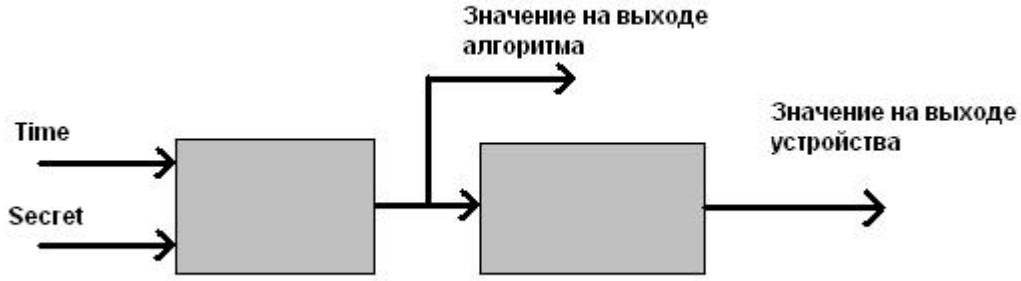


Если токенкоды предоставляются через среду, которая просматривается атакующим эту сеть или атакующий имеет доступ к токену или последовательности сгенерированных кодов, то возможность создания эмулятора, предсказывающего следующие коды, становится реальной.

Соответственно, для обеспечения безопасности доступа с использованием токенов достаточно обеспечить следующие условия: ограничить доступ к самим токенам или их софтовым аналогам на копьютере, использовать аутентификацию через шифрованные каналы.

2. Описание Алгоритма

Коды, генерируемые алгоритмом получаются из двух внутренних переменных: 64-битного значения времени и 64-битной секретной части. Результат действия алгоритма проходит через конечную конвертацию для более удобного представления на экране hardware или software токена.



2.1. Время в алгоритме

Значение времени используемое в алгоритме представляет текущее время в 64-битном виде. Однако известно, что это 64-битное значение получается из 32-битного представления текущего времени(GMT) в секундах, начиная с полночи 01/01/86.

```
INT64 time64;
INT32 time;
UCHAR byte;
time = gettimeofday(); // Кол-во секунд с  01/01/86, 00:00
// округление в меньшую сторону
time = time / 30;
time = time / 4;
time = time * 4;
// расширение времени до 64-битного значения дублированием битов.
byte = time | 0xFF;
time = time << 8;
time = time | byte;
time64 = time;
time64 = time64 << 32;
time64 = time64 | time;
```

Например, пусть число секунд, прошедшее от начала отсчёта будет 0x1C39B862, тогда время на входе алгоритма будет 0xF0DB7878F0DB7878. Это значение получено округлением количества секунд до достижения значения 0x00F0DB78. Затем значение сдвигается на 8 бит и младший значащий бит отображается дважды. Таким образом получается, что только 24 бита из прошедшего времени используются.

В функции округления значение сдвигается дважды, что эквивалентно умножению на 4. Следовательно, значение будет чётным и младшие значащие 2 бита всегда будут 00 и младший значащий байт может быть только 0x0, 0x4, 0x8 и 0xC. Это умножение убирает два последних бита от 24 битного значения времени достигающего 2^22 или 4,194,304 возможных значений времени. Также стоит отметить, что кол-во значений времени еще больше может быть уменьшено, учитывая то, что атакующий имеет представление о приблизительном времени, по которому сгенерирован код.

Результатом такого уменьшения временного пространства является возможность генерации полного цикла прохождением по всем значениям времени для заданного секретного значения. Период цикла подбирается таким образом, чтобы быть больше времени предполагаемого использования токена. Если токен работает в real-time режиме (8-битное значение выводится один раз в минуту, то полный цикл занимает около 16 лет). Однако, используя модифицированный software токен можно пройти весь цикл за несколько минут.

2.2. Секретная часть

RSA не разглашает каким образом происходит генерация секретной 64 битной части алгоритма, которая подаётся на один из входов алгоритма. Но т.к. эти значения предопределены, можно предположить, что даже если алгоритм генерации был бы предсказуем, то его всё равно можно было бы прозрачно менять на более случайный алгоритм генерации. Однако даже с сильным случайным или псевдослучайным механизмом генерации секретной части, ограниченное число возможных значений времени делает алгоритм уязвимым в некоторых случаях.

Предположим, что секретная часть извлечена из hardware или software токена, тогда всевозможные значения выхода алгоритма могут быть “прокручены” и записаны используя соответствующий SecureID алгоритм. Это может быть сделано без вмешательства в оригинальный токен, так что атака не будет замечена немедленно. Я не предусматриваю рассмотрение этих методов в данном эссе.

2.3. Конвертация.

Значение кода показываемое пользователю представляет собой 6 или восьмизначное число в десятичном виде. Это десятичное значение получается из 64 битного шестнадцатиричного значения, называемого “pre-converted value”, с помощью простого преобразования приведённого ниже. Эта функция приводит шестнадцатиричное число к десятичному не общепринятым способом:

```
//выполняется для каждого байта.
{
        if (nibble > 9)
        {
        nibble = nibble - 2;
        nibble = nibble - {0, 2, 4, 6, 8};
        nibble = nibble % 10;
        }
}
```

Выполняется для байта в шестнадцатиричного числа, если этот байт больше 9. Если байт меньше 9ти, то это число и выводится. В идельном случае значения от 0xA до 0xF должны быть преобразованы в десятичные числа от 0 до 9. Однако, т.к. 0xA, 0xC и 0xE могут быть поставлены в соответствие числам 0, 2, 4, 6 или 8, а 0xB, 0xD и 0xF только 1, 3, 5, 7 или 9, то число возможных pre-converted значений значительно уменьшается. В таблице соответствие отображаемых чисел и соответствующих им значений до прохождения конвертации:

0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
A	B	A	B	A	B	A	B	A	B
C	D	C	D	C	D	C	D	C	D
E	F	E	F	E	F	E	F	E	F

Пусть *tlen* – это длина кода отображаемого токеном. Для любого заданного значения токенкода можно найти только *4^tlen* возможных значений до прохождения конвертации, которые могут быть проанализированы для определения определённой битовой конфигурации настоящей секретной части. В идельном случае возможно *7^tlen.* Например, для обычнго токена отображающего 6 цифр существует *4^6=4096* таких значений. Идеальный же случай даёт *7^6 или 117649* значений.

Вероятность, что отображаемое на самом деле число на экране токена является натуральным(т.е. отображаемый 0 имеет соответствие с 0 до конвертации) равна 62.5%, а вероятность того, что число получено из {A, B, C} или {D, E, F} – 12.5%. Более подробный статистический анализ проводить не будем.

2.4. Коллизии в цикле алгоритма.

Анализ цикла алгоритма токена для небольшого количества псевдослучайных 64-bit секретных чисел даёт интересный результат. Из 8388608 возможных значений кода токена(по 2 на каждое знаение времени), приблизительно 8 миллионов из этих кодов уникальны и проявляются только один раз за весь цикл. Оставшиеся 300000 – это повторяющиеся значения.

Процент коллизий = (1 – (кол-во уникальных токенов)/(кол-во возможных токенов))*100.

Таким образом, 4% цикла токена содержит значения, хотя бы один раз повторяющиеся за цикл. Возможно это происходит из-за недостатков конвертации.

3. Заключение.

Для того, чтобы максимально обезопасить аутентификацию с использование SecureID алгоритма, компаниям достаточно придерживаться достаточно придерживаться некоторых правил. Во первых – шифровать все соеденения. Во вторых необходимо ограничить доступ к самим токенам, будь они реализованы в виде отдельных устройств или лишь эмуляторми на PC. Тогда вероятность получения злоумышленниками секретной части алгоритма значительно уменьшается. Также могут быть использованы механизмы шифрования SSH, SSL, DESTelnet.

4. Литература

1. Sample SecurID Token Emulator with Token Secret Import [I.C. Wiener]
   http://www.securityfocus.com/archive/1/152525

2. RSASecurity.com
   http://www.rsasecurity.com

3. Carlo U. Nicola, Daniel Mall, SGI FH Aargau - Authentication protocols
   http://lis.fh-aargau.ch/LISLectures/lectures/Protocol.ppt