Эссе по курсу «Защита информации», кафедра радиотехники,

Московский физико-технический институт (ГУ МФТИ),

http://re.mipt.ru/infsec

Схема TKS1 шифрования данных жесткого диска

Пешков Алексей Игоревич
30 марта 2006 г.

1. Иерархия паролей

Положим, шифрование данных жесткого диска происходит ключом, в качестве которого используется пароль пользователя. Помимо того, что обычные пароли обладают, по крайней мере, двумя недостатками (о которых речь пойдет в разделе 3) , возникает еще и так называемая проблема перешифрования данных. Другими словами, изменение пароля потребует следующих действий:

- Расшифрование данных с помощью старого пароля;
- Шифрование с использованием нового ключа-пароля;
- Запись полученной шифрованной информации на носитель.

Эта процедура кажется весьма затруднительной, если учесть то, что объемы современных носителей информации достигают значительных размеров, а сама процедура перешифрования – длительный процесс, вызывающий нежелательное замедление работы всей системы. Решением проблемы стало внедрение так называемой иерархии паролей или иерархии ключей. Предположим, что у нас есть следующая структура:

- Главный ключ
- Пользовательский ключ

Будем шифровать данные жесткого диска главным ключом, который остается все время неизменным. Сам же главный ключ будем шифровать пользовательским ключом-паролем, и полученный шифр запишем на тот же жесткий носитель. Эта процедура весьма проста, если учесть, что размер главного ключа составляет 16-32 байта. Таким образом, выявляется сразу несколько положительных моментов:

- Отсутствие необходимости перешифрования всех данных жесткого диска, так как главный ключ не меняется;
- Возможность доступа к шифрованным данным сразу нескольких пользователей, каждый – со своим паролем-ключом; в этом случае главный ключ шифруется разными пользовательскими ключами, и все его шифры записываются на жесткий диск.

2. AF-хранение данных

Хранение данных на жестком диске обладает особым свойством: даже если данные были удалены, то они легко могут быть восстановлены. Это свойство является существенным в отношении главного ключа, так как он хранится в зашифрованном виде на диске. Поэтому, учитывая его размеры (16-32 байта), весьма затруднительно уничтожение его данных. Рассмотрим одну из возможностей решения этой проблемы.

Положим, p - вероятность уничтожения некоторого блока данных, тогда $1-p$ - вероятность того, что он выживет. Если таких блоков n, то вероятность того, что все n блоков выживут: $(1-p)^n$, тогда $1-(1-p)^n$ - вероятность того, что блоки претерпят изменение. Вообще пользователь не может влиять на p, так как это – параметр самого жесткого диска. Однако есть возможность влиять на n, выбирая его сколь угодно большим. Это позволит событию изменения n блоков данных стать почти наверняка вероятным.

В связи со сказанным выше, введем операцию реконструкции:

$$\Re : \mathbb{S}^n \to \Bbbk ,$$

которая переводит данные расщепленного ключа в данные шифрованного ключа K. Подробнее это выглядит следующим образом: выберем произвольные случайные $S_i,\ i=[1,n-1]$ того же размера, что и шифрованный ключ K. Тогда определим S_n как:

$$S_n = S_1 \oplus S_2 \oplus ... \oplus S_{n-1} \oplus K$$

Здесь $\oplus$ - это операция XOR. Тогда операция реконструкции может быть представлена действием:

$$K = S_1 \oplus ... \oplus S_n$$

Очевидно, что для точной реконструкции шифрованного ключа необходимо, чтобы все его расщепленные данные $S_i,\ i=[1,n]$ были считаны с жесткого диска без ошибок. Выше было показано, что, управляя числом n, можно сделать вероятность изменения в наборе $\{S_1, S_2, ..., S_n\}$ почти 100%. В этом случае становится невозможным восстановление шифрованного ключа K. Таким образом, когда пользователь желает изменить шифрованные данные главного ключа, ему достаточно переписать значения $\{S_1, S_2, ..., S_n\}$, после чего K затирается с высокой долей вероятности.

Однако также очевидно, что изменение лишь части S_i повлечет за собой изменение той же части K, тогда как вся остальная информация будет по-прежнему доступна. Только полное изменение S_i сделает данные K невосстановимыми. Поэтому описанная выше схема усиливается так называемыми диффузионными элементами, которые позволяют устранить эту проблему. В качестве диффузионных элементов используются H - блоки, которые возвращают хэш-значение от входных данных:

$$h_1 = S_1$$
$$h_i = H(h_{i-1}) \oplus S_i$$

Блок, осуществляющий полную диффузию указанным способом, называется AF-сплиттер (от английского Anti-Forensic splitter).

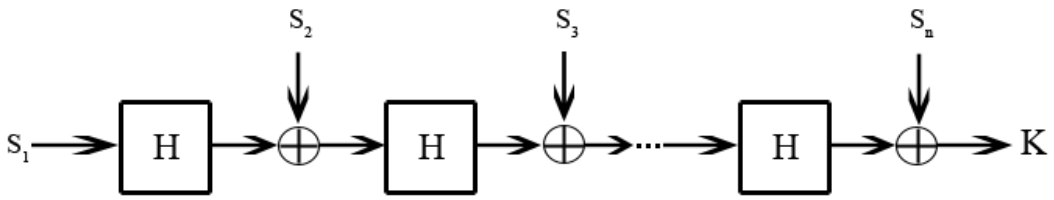

Рис. AF-сплиттер

Интересно также указать способ выбора n. Пусть на этот раз p - вероятность того, что один бит информации уничтожается. Тогда для количества записей n длины k вероятность выживания равна $(1-p)^{kn}$. Пусть эта нежелательная вероятность будет определяться вероятностью подбора главного ключа с первого раза: $\left(\frac{1}{2}\right)^k$. Имеем:

$$\left(\frac{1}{2}\right)^k = (1-p)^{kn}$$

Отсюда получаем желаемое значение n:

$$n = \frac{\ln(2)}{\ln(1-p)}$$

То есть n выбирается лишь на основе данных о свойствах самого носителя и не зависит от длины главного ключа. Значение вероятности p зависит от характеристик самого жесткого диска, от его времени службы и прочего, и принимает значения порядка 0,01-0,0001

3. PBKDF2 (процессорно-интенсивные функции)

Теперь подробнее коснемся проблем, связанных с пользовательским паролем-ключом. Как правило, такие пароли обладают двумя недостатками:

- Относительно небольшая длина;
- Пароли основаны на фразах и их сочетаниях из словаря.

Эти качества, с одной стороны, позволяют легко запоминать их, но с другой стороны, делают возможным атаку перебором. Как говорят криптоаналитики в этом случае, пароль обладает недостатком энтропии.

Ниже предлагается вариант решения описанной проблемы с учетом возможности пользователя работать с легко запоминающимися паролями.

Не существует легкого способа увеличения энтропии, который подошел бы для наших целей. Поэтому предлагается иной вариант защиты против атаки перебором. Подобная атака, как правило, представляет собой три последовательных шага, повторяющихся до тех пор, пока не будет достигнут желаемый результат:

- Выбор блока шифра и ключа;
- Расшифрование выбранного блока;
- Проверка, был ли достигнут положительный результат.

Существует возможность сделать процесс расшифрования медленным по времени, но тогда это будет тормозить работу всей системы. Было предложено ввести так называемую операцию процессорно-интенсивной трансформации. Математически это означает следующее: мы переводим область значений пользовательских фраз в область значений пользовательского ключа в его первичном понимании, то есть вместо вычисления $K_k(c)$ предлагается вычислять $K_{f(k)}(c)$, где функция $f(k)$ требует значительного процессорного времени для ее подсчета. Вся хитрость в том, что пользователь имеет возможность настраивать время подсчета $f(k)$ (в диапазоне от 1 до 5 секунд, например), что делает невозможным перебор пользовательских фраз.

Размер области значений пользовательского ключа, преобразованной функцией $f(k)$, может быть сделан сколь угодно большим, что, опять-таки, препятствует перебору по словарю, составленному по всем пользовательским фразам, преобразованным той же $f(k)$. Однако следует учесть и экспоненциальный рост технологий, что позволит однажды решить эту проблему. Поэтому целесообразно функцию $f(k)$ сделать зависимой не только от пароля пользователя, но и от произвольной константы s: $f(k,s)$. Очевидно, что каждый бит s удваивает размер словаря для атаки перебором. Поэтому разумно взять s по возможности большим. Следует также отметить, что нет необходимости в конфиденциальности s; его можно хранить на жестком диске в открытом виде.

В качестве PBKDF (Password-Based Key Derivation Function) функции, то есть фактически функции $f(k)$ наиболее подходящими являются хэш-функции, поскольку область их значений определена для входной строки переменного размера. Для того, чтобы сделать такие функции процессорно-интенсивными, достаточно выполнять их итеративно столько раз, сколько покажется нужным. PBKDF2 (Password-Based Key Derivation Function, revision 2) как раз воплощают описанный прием.

В самом упрощенном варианте подсчет подобных функций сводится к вычислению значения

$$f(p,d,s,i) = h^1(s \| i) \oplus h^2(s \| i) \oplus ... \oplus h^d(s \| i)$$

Где $h^j()$ означает взятие хэш-значения от аргумента в j-ой итерации, а d - глубина общей итерации. Результат предоставляется в виде конкатенации строк:

$$f(p,d,s,1) \| f(p,d,s,2) \| ... \| f(p,d,s,k)$$

Подробнее с PBKDF можно ознакомиться в RFC 2898

4. Обзор TKS1

Итак, подведем итог. Мы ознакомились со следующими технологиями:

- *Иерархия паролей* для предоставления возможности многопользовательского доступа к данным;
- *AF-сплиттинг* для возможности уничтожения данных, хранимых на жестких носителях, без возможности их восстановления;
- *PBKDF2* (*процессорно-интенсивные функции*) для усиления защиты паролей.

Все эти технологии с успехом применяются в общей схеме хранения/доступа к данным жесткого диска TKS1 (Template Key Setup 1). Рисунок, приведенный ниже, поясняет интеграцию описанных технологий в общую структуру защиты информации жесткого диска.

Рис. Структура TKS1

Как видно из рисунка, основную часть всего процесса занимает операция расшифрования главного ключа. Краткое описание этого алгоритма следующее:

1. считывание константы s и глубины общей итерации из области жесткого диска, отведенной под хранение главного ключа;
2. считывание $\{S_1, S_2, ..., S_n\}$ - шифрованных данных главного ключа из той же области жесткого диска и AF-восстановление этих данных;
3. ввод пользователем фразы-пароля;
4. PBKDF2-обработка введенной пользователем фразы;
5. расшифрование главного ключа на основе полученных данных;
6. работа над шифрованными данными жесткого диска с использованием главного ключа;

7. удаление копий главного ключа из памяти.

В заключении хотелось бы сказать пару слов об использовании TKS. Данная технология нашла применение в LUKS – стандарте шифрования данных жесткого диска под Linux. Подробную информацию о стандарте можно найти на официальном сайте LUKS (http://luks.endorphin.org/)

5. Материалы

1. New Methods in Hard Disk Encryption, Cl. Fruhwirth, 2005

http://clemens.endorphin.org/nmihde/nmihde-A4-ds.pdf

2. RFC 2898 Password-Based Cryptography Specification

http://rfc.net/rfc2898.html

3. LUKS – Linux Unified Key Setup website

http://luks.endorphin.org/

4. Secure Deletion of Data from Magnetic and Solid-State Memory, P. Gutmann, 1996

http://www.cs.auckland.ac.nz/%7Epgut001/pubs/secure_del.html