УДК 004.056.55

Разработка алгоритма хеширования информации на основе метода наименьших квадратов

А. А. Кобец, Н. В. Марковская
МОУ «Специализированная физико-математическая школа №17»
kobetc.rey8@gmail.com, marknata@mail.ru

Кобец А. А., Марковская Н. В. Разработка алгоритма хеширования информации на основе метода наименьших квадратов. *В статье описывается необходимость разработки новых принципов хеширования, а также формулируются требования к функциям, реализующим получение дайджестов сообщений. Выдвигается и обосновывается гипотеза о возможности использования метода наименьших квадратов в качестве метода получения дайджеста сообщений. Приводится разработанный алгоритм, построенный на базе данной гипотезы.*

Ключевые слова*: криптография, хеширование, дайджест, МНК.*

Постановка проблемы

В человеческом обществе все более актуальной становится проблема опасности перехвата конфиденциальной информации при работе с ней. Современная криптография для обеспечения защиты данных использует наборы преобразований, называемых криптографическими примитивами.

В частности, для обеспечения безопасного хранения и сравнения информации, используются дайджест-функции. Однако, возникает проблема кодировки, которая базируется на сравнительной простоте большого числа используемых дайджест-функций.

Стремительный рост производительности вычислительной техники позволяет говорить о реальных временных промежутках при применении brute-force атак (от 30 минут до 45 часов). К тому же продолжительное изучение криптоаналитиками данных алгоритмов, позволяет находить коллизии за минуты.

Таким образом, для решения данной проблемы необходима разработка совершенно нового способа хеширования, основанного на ранее не использовавшихся принципах.

Цель

Целью данной работы является создание оптимального алгоритма хеширования, основанного на методе наименьших квадратов.

Анализ существующих разработок

На данный момент существует большое количество алгоритмов хеширования. Многие из них реализуют, так называемую, блочную структуру. Наиболее показательными примерами таких алгоритмов являются дайджест-функции MD5 и SHA-1.

MD5 – это 128-битный алгоритм хеширования. Данный алгоритм был разработан Рональдом Л. Ривестом из Массачусетского технологического института как более надёжная версия алгоритма MD4. Впервые был описан в RFC – 1321 [1].

Основным достоинством алгоритма MD5 является наличие, так называемого, «лавинного эффекта». Это означает, что незначительное изменение в открытом тексте, передаваемое данному алгоритму, вызовет кардинальное изменение в результирующем хеше.

Однако, ещё в 2006 году чешский исследователь Властимил Клима опубликовал алгоритм, в последствии названный «туннелированние» [2]. Данный алгоритм позволял находить коллизии MD5 на домашнем компьютере за полминуты. В следствие чего, в конце 2008 года MD5 был признан небезопасным, и организация US-CERT призвала прекратить использовать MD5 в любых целях.

Не смотря на это, многие разработчики активно применяют его до сих пор.

SHA-1, описанный в RFC 3174 [3], – как и MD5, является усовершенствованной версией алгоритма MD4. Важным отличием SHA-1 от MD5 является то, что на выходе получаемая последовательность имеет длину 160 бит. Как и MD5, SHA-1 присущ лавинный эффект, более того, данный алгоритм гораздо более устойчив к поиску коллизий, нежели описанный выше.

Тем не менее, несмотря на то, что SHA-1 несколько более сложен по своей структуре, нежели MD5, подбор пароля при помощи brute-force – атаки, по заданному дайджесту SHA-1 не является проблемой. Семизначный пароль, в котором есть символы латинского алфавита, а также цифры 0-9 и спецсимволы подбирается примерно за 31 час[4].

Определение требований к надёжному алгоритму хеширования

В криптографии существует ряд требований к хеш-функциям:

1. Необратимость – для дайджеста, построенного заданной хеш-функцией должно быть вычислительно невозможно найти прообраз.
2. Стойкость к коллизиям первого рода – для заданного сообщения должно быть вычислительно невозможно подобрать такой аналог, чтобы в обоих случаях функция вернула одинаковый результат.
3. Стойкость к коллизиям второго рода - должно быть вычислительно невозможно найти пару сообщений, для которых заданная хеш-функция вернёт одинаковый результат.

Так же стоит отметить, что подавляющее большинство современных хеш-функций реализует «лавинный эффект» - свойство дайджеста значительно изменяться даже при незначительных отличиях исходных последовательностей.

Разработка алгоритма

Метод Наименьших Квадратов (МНК англ. ***Ordinary Least Squares, OLS***) – это метод регрессионного анализа, позволяющий сгладить результаты наблюдений, содержащих случайные ошибки.

Именно на нём построен основной принцип работы алгоритма.

Выбор данного метода обоснован следующими его свойствами:

1. Для трёх точек, не принадлежащих одной прямой, шанс вхождения одной отдельно взятой точки во множество решений аппроксимирующей линейной функции, сравнительно мал, а также невозможно сказать в какую сторону происходит отклонение. По этим причинам, истинное положение точки не может быть воспроизведено аналитически.
2. Изменение положения любой из искомых точек отражается на конечном результате в достаточно большой степени, чтобы вызвать «лавинный эффект».

Для реализации алгоритма был выбран язык программирования Java, так как его среда исполнения поддерживается большинством современных устройств, а так же в состав его стандартной библиотеки входят классы, реализующие возможность использования вещественных чисел с неограниченной разрядностью, что весьма важно для увеличения эффективности результата.

Вычисление дайджеста сообщения происходит в три условных этапа:

1. Вычисление первичной числовой комбинации.
2. Вычисление вторичной числовой комбинации.
3. Слияние распределение комбинаций по массиву байт требуемой длины.

Вспомогательные функции

В ходе вычисления дайджест последовательности, алгоритм неоднократно обращается к следующим вспомогательным функциям:

1. **pseudoRandom(seed, mix)** - Данная функция предназначена для генерации псевдослучайных чисел, вычисляемых на основе двух аргументов по следующему алгоритму:
   1. Инициализируется переменная $a = \sin(\cos(seed)) * 100$;
   2. Выполняется присваивание $a = a \cdot sin((int)(a * 100))$. В данном случае (int) означает приведение к целочисленному типу с отбрасыванием всех разрядов меньше единицы.
   3. Значение a переводится из радианов в градусы.
   4. Выполняется присваивание $a = a \cdot \sin(mix)$;
   5. Значение a приводится к целочисленному типу с отбрасыванием всех разрядов меньше единицы
   6. Значению a присваивается сумма первых двух и последних двух разрядов.
   7. На выходе данная функция возвращает **|*a*|**.

   При условии, что параметры данной функции лежат в интервале [0;65536], её результат будет лежать в промежутке [0-146].

2. **arrayShuffle(array, rndSeed);**

   Данная функция предназначена для перестановки элементов массива на основе вышеописанного ГПСЧ. Работает по следующему алгоритму:
   1. Для каждого i-го элемента массива:
   2. Вычисляется число j, такое, что:

$$j = (int)\left(\left(\frac{pseudoRandom(rndSeed,i)}{144}\right) * i\right) \quad (1)$$

   Исходя из формулы, $j \in [0; i]$;
   3. Меняются элементы массива с номерами i, j местами;
   4. По завершении цикла, функция возвращает преобразованный массив.

3. arrayXOR(bytes, bytes1)

Данная функция выполняет сложение по модулю 2 всех элементов принимаемых массивов, а так же – корректирует их длины. Данная функция реализует следующий алгоритм:

1) Если длины массивов совпадают, действия 2-3 пропускаются.
2) Избыточные элементы поочерёдно суммируются по модулю два со всеми элементами массива, имеющего лишнюю длину.
3) Выполняется уравнивание длин массивов.
4) Для *i*-го элемента массива *bytes* выполняется действие:

$$bytes1[i] = bytes[i] \oplus bytes1[i];$$

5) Результат данной функции – массив *bytes1*.

На вход данного алгоритма поступает два значения:

1. *mdLenBytes* – число байт, ожидаемое пользователем, должно входить в натуральные числа.
2. *sequence* – входная последовательность байт.

Входное сообщение разбивается на массив *numbers*, хранящий в себе числовое представление символов исходной строки. Дальнейшие операции производятся уже на нём.

3. В данный массив дописывается число, зависящее исключительно от дины массива и вычисляемого по формуле:

$$len = preLength \oplus pseudoRandom(preLength, preLength);$$

где *preLen* – длина массива до включения в него искомого числа.

4. Длина исходного массива увеличивается до ближайшего числа кратного 9. При этом новые элементы заполняются элементами из старого (по «кольцевой схеме». Например, при исходном массиве = [5, 4, 3], новый массив выглядит следующим образом: [5, 4, 3, 5, 4, 3, 5, 4, 3]).
5. Затем выполняется цикл, в котором все значения исходного массива заменяются по следующему принципу:

$$numbers[i] = 1 + \left(\frac{pseudoRandom(numbers[i],i)}{144}\right) \cdot 65536; \quad (2)$$

6. Первый этап – получение первичной хеш-последовательности:

Из созданного массива вычисляется трёхмерная матрица *matrix*, размером [N][3][3]. Данная матрица при дальнейших операциях интерпретируется как N избыточных систем линейных уравнений вида:

$$\begin{cases} matrix[i][0][0] \cdot x + matrix[i][0][1] \cdot y = matrix[i][0][2]; \\ matrix[i][1][0] \cdot x + matrix[i][1][1] \cdot y = matrix[i][1][2]; \\ matrix[i][2][0] \cdot x + matrix[i][2][1] \cdot y = matrix[i][2][2]; \end{cases} \quad (3)$$

где i – номер данной системы.

Для простоты обозначения, введены специальные обозначения:

$$A_{i_j} = matrix[i][j][0]; B_{i_j} = matrix[i][j][1]; \quad (4)$$
$$C_{i_j} = matrix[i][j][2];$$

С учётом введённых обозначений, система 3 будет выглядеть следующим образом:

$$\begin{cases} A_{i_1} \cdot x + B_{i_1} \cdot y = C_{i_1}; \\ A_{i_2} \cdot x + B_{i_2} \cdot y = C_{i_2}; \\ A_{i_3} \cdot x + B_{i_3} \cdot y = C_{i_3}; \end{cases} \quad (5)$$

Данная избыточная система легко решается при помощи метода наименьших квадратов. После всех преобразований, выводится формула, являющаяся явным представлением пары чисел (X, Y):

$$\begin{cases} X = \frac{\sum_{j=0}^{n} A_{i\,j}^2 \cdot \sum_{j=0}^{n} A_{i\,j} C_{i\,j} - \sum_{j=0}^{n} A_{i\,j} B_{i\,j} \cdot \sum_{j=0}^{n} B_{i\,j} C_{i\,j}}{\sum_{j=0}^{n} B_{i\,j}^2 \cdot \sum_{j=0}^{n} A_{i\,j}^2 - \left(\sum_{i=0}^{n} A_{i\,j} B_{i\,j}\right)^2}; \\ Y = \frac{\sum_{j=0}^{n} A_{i\,j} C_{i\,j} - \sum_{j=0}^{n} A_{i\,j}^2 \cdot X}{\sum_{i=0}^{n} A_{i\,j} B_{i\,j}}; \end{cases} \quad (6)$$

где n – это количество рассматриваемых уравнений – 1 (в данном случае: 3-1= 2).

Результаты вычислений записываются в двумерную матрицу results.

Первичная хеш-последовательность представляет собой множество, состоящее из двух чисел и являющееся произведением полученных результатов.

Очень важно сохранить максимальную точность данного числа, т. к. от этого будет зависеть насколько сложен будет дайджест в результате работы алгоритма поэтому рекомендуется использовать типы данных, позволяющие работать с неограниченным количеством символов после запятой, например BigDecimal в языке Java.

Второй этап – получение вторичной хеш-последовательности.

Данный этап, как и предыдущий основывается на методе наименьших квадратов. Однако, в данном случае, вместо использования его в качестве метода решения избыточных систем, здесь реализуется ещё одно применение данного метода. При получении вторичного хеша, данный метод используется для получения коэффициентов линейного уравнения, наиболее точно описывающего последовательность точек:

$$\left(i;\ pseudoRandom(sequence[i], i)\right) \quad (7)$$

где i – номер позиции символа в передаваемой последовательности.

Т. к. первое число является координатой *i*-ой точки на оси абсцисс, обозначим его как X_i, а второе, соответственно Y_i.

Явный вид параметров линейного уравнения:

$$\begin{cases} A = \frac{\sum_{i=0}^{n-1} X_i \cdot \sum_{i=0}^{n-1} Y_i - n \cdot \sum_{i=0}^{n-1} X_i Y_i}{(\sum_{i=0}^{n-1} x_i)^2 - n * \sum_{i=0}^{n-1} X_i Y_i} \\ B = \frac{1}{n} \cdot (\sum_{i=0}^{n-1} Y_i - A \cdot \sum_{i=0}^{n-1} x_i \end{cases} \quad (8)$$

Также следует указать, для достижения наилучшего эффекта, требуется использовать то же количество символов после запятой, что и у первичной хеш-последовательности.

Множество {A, B} является вторичной хеш-последовательностью.

1. Далее все составляющие элементы, составляющие первичную и вторичную хеш-последовательности переводятся в массивы байт.
2. Объявляется массив fusionArray, который, в последствии, будет хранить объединённые значения первичного и вторичного хеша.
3. Выполняются следующие действия:

$fusionArray[0] = arrayXOR(primaryHash[0], secondaryHash[0]);$
$fusionArray[1] = arrayXOR(primaryHash[1], secondaryHash[1]);$

где primaryHash[0], primaryHash[1], secondaryHash[0], secondaryHash[1] – первые и вторые цифры первичной и вторичной хеш последовательности соответственно.

4. Создаётся переменная *hash*, в которую записывается следующее значение:

$hash = arrayXOR(fusionArray[0], fusionArray[1]);$

5. Каждый элемент массива подвергается изменению при помощи формулы:

$hash[i] = hash[i] \oplus (pseudoRandom(i, hash[i]) \oplus i;$

где i – номер элемента в массиве.

6. Положение значений в массива *hash* меняется в псевдослучайном порядке:

$hash = arrayShuffle(hash, (slength \oplus hashLength));$

где slength и hashLength длина исходной строки и длина хеша соответственно.

7. Повторение действия 11.
8. Создаётся returnHash изаполняется по следующему алгоритму: для i, не превышающего длины массива элемента массива:

$returnHash[i] = pseudoRandom(i, hash.length \oplus mdLenBytes);$

9. Выполняется слияние сгенерированных и полученных раннее данных.

В цикле с переменной-счётчиком i, не превышающем длину массива hash, проверяется условие. Существует ли i-й элемент в массиве returnHash. Если условие верно, то выполняется простейшее преобразование:

$returnHash[i] = returnHash[i] \oplus hash[i];$

Иначе, выполняется вход в другой цикл, в котором изменяются все элементы массива returnHash номера которых кратны остатку от деления i на размер массива returnHash. Изменение происходит по следующщему примеру:

$returnHash[j] = returnHash[j] \oplus hash[i];$

где j – это номер элемента массива, кратный остатку от деления.

10. Если размер массива hash меньше размера массива returnHash, выполняется преобразование:

$returnHash = arrayShuffle(returnHash, pseudoRandom(hash.length, returnHash.length));$

11. Массив returnHash и является искомым дайджестом.

Выводы

В рамках данного исследования был проведён анализ основных уязвимостей криптографических функций. В качестве альтернативы к уже устаревшим алгоритмам был разработан совершенно новый, основанный на ранее не применявшихся принципах алгоритм хеширования.

Литература

1. RFC 1321 [Электронный ресурс] // IETF Tools - Режим доступа: https://tools.ietf.org/html/rfc1321

2. Tunnels in Hash Functions: MD5 Collisions Within a Minute. Vlastimil Klima Prague, Czech Republic [Электронный ресурс] // Режим доступа: http://cryptography.hyperlink.cz/2006/tunnels.pdf

3. RFC 3174 [Электронный ресурс] // IETF Tools Режим доступа: https://tools.ietf.org/html/rfc3174

4. NIST COMMENTS ON CRYPTANALYTIC ATTACKS ON SHA-1 [Электронный ресурс] // National Institute of Standards and Technology - Режим доступа: http://csrc.nist.gov/groups/ST/hash/statement.html

5. Дринфельд Г.И. Интерполирование и способ наименьших квадратов // Г.И. Дринфельд К.: «Вища школа», 1984. - 103 с.

Кобец А. А., Марковская Н. В. Разработка алгоритма хеширования информации на основе метода наименьших квадратов. *В статье описывается необходимость разработки новых принципов хеширования, а также формулируются требования к функциям, реализующим получение дайджестов сообщений. Выдвигается и обосновывается гипотеза о возможности использования метода наименьших квадратов в качестве метода получения дайджеста сообщений. Приводится разработанный алгоритм, построенный на базе данной гипотезы.*

Ключевые слова*: криптография, хеширование, дайджест, МНК.*

Kobets A. A., Markovskaya N. V. Developing hashing information algorithm based on ordinary least squares method. *The article describes the necessity of developing new principles of hashing and formulates the requirements to the functions that realize getting message digests. A new hypothesis which says that ordinary least squares method is suitable as the method of getting message digests had been put forward and justified. Developed algorithm based on this hypothesis is given.*

Keywords*: cryptography, hashing, digest, OLS.*

Статья поступила в редакцию 24 апреля 2018 г.
Рекомендована к публикации профессором Павлышом В. Н.