ФОРМИРОВАНИЕ ФУНКЦИИ ХЭШИРОВАНИЯ ПАРОЛЕЙ, СТОЙКОЙ К УСКОРЕННОМУ ПЕРЕБОРУ ЗНАЧЕНИЙ

Ю. Б. Панфилов
ООО «БАЛТОРГЦЕНТР», 143900, Московская область, Балашиха, улица Крупской, 11

Статья поступила в редакцию 16 марта 2018 г.

Аннотация. В статье поднимается проблема поиска пароля по его хэшу с помощью специализированного оборудования, ускоряющего перебор. Предлагается функция хэширования с ключём на основе смешанной системы счисления, устойчивая к подобного рода оптимизациям. Демонстрируется её криптографическая стойкость к основным атакам, масштабируемость к различным моделям угроз и теоретический запас прочности.

Ключевые слова: аутентификация, смешанная система счисления, функция хэширования, хэширование паролей.

Abstract. Today, the problem of accelerated enumeration of the values of cryptographic functions with the help of optimized implementations of the corresponding algorithms for specially designed devices for this purpose is urgent. It is also not a secret that most users of different information systems neglect the requirements of complexity when choosing passwords, often choosing a password from a rather limited set of possible values. In this case, the potentially attacked system faces the task of nullifying the technical equipment of the attacker, making the dictionary search economically inefficient, based on the realities of the current level of computer technology development. Changing the parameters of the algorithm, and not the algorithm itself, it is possible to achieve a significant difficulty in booting at the hardware level, if in the process of computing there are guaranteed "inconvenient" numbers for the computing device, which makes it difficult to search even for a significant number of cores. The article proposes a hash function with a key based on a mixed number system, which is resistant to this kind of optimization. It demonstrates its cryptographic resistance to basic attacks, scalability to various threat models and theoretical margin of safety.

Key words: authentication, mixed-base system, hash function, password hashing.

На сегодняшний день достаточно актуальна проблема ускоренного перебора значений криптографических функций с помощью оптимизированных реализаций соответствующих алгоритмов для специально разработанных для данной цели устройств [1,2]. Подобные устройства, как правило, создаются на основе интегральных схем программируемой логики (сюда можно отнести и "заказные микросхемы") и мощных графических процессоров, позволяющих распараллеливать обработку данных по заданному алгоритму и вследствие этого выполнять вычисления более эффективно в сравнении с процессорами общего назначения. Известно, что отвечающие современным стандартам безопасности информационные системы, использующие пароль для аутентификации, не хранят у себя пароли и не требуют их пересылки в открытом виде, а используют хэши паролей [1,2]. Стандартную процедуру аутентификации в таком случае можно описать следующим образом. Рассмотрим некоторую систему, предоставляющую пользователям доступ к своим ресурсам по паролю. Система хранит хэш от пароля пользователя *Pass* и какого-нибудь входного вектора, не являющегося секретом, например случайного *R*, то есть *h(R,Pass)*. В процессе аутентификации пользователь подаёт запрос системе и получает от неё некоторый дополнительный входной вектор, также не являющейся секретом, это вполне может быть и текущая дата *Data*. После этого пользователь вычисляет хэш-значение *h(Data, h(R,Pass))* и пересылает системе, которая вычисляет то же самое и сравнивает полученные значения. Если значения равны, то аутентификация прошла успешно. Очевидно, что при таком подходе в случае компрометации системы (например, взломе сервера) пароли пользователей скомпрометированы не будут, также перехват сообщений между пользователем и системой не приведёт к раскрытию пароля, но хэши паролей могут оказаться в распоряжении злоумышленника.

Также не является секретом, что большинство пользователей различных информационных систем пренебрегают требованиями сложности при выборе паролей, зачастую выбирая пароль из довольно ограниченного набора

возможных значений (дата рождения, телефонный номер, имя домашнего животного и т.д.), а также используя один и тот же пароль для доступа к разным ресурсам [2]. В общем случае, имея на руках базу данных с хэшами паролей, злоумышленник для любого выбранного хэша H может осуществить "атаку по словарю", то есть последовательно выполнить проверку $h(R,Pass_i)=H$, где $Pass_i$ – очередной элемент множества M_1 наиболее вероятных парольных фраз, R хранится вместе с хэшем, $1<i<|M_1|$. При этом необходимо уточнить, что хотя множество M_1 многократно меньше мощности полного перебора M_0, тем не менее перебор по M_1 остаётся сложной задачей для злоумышленника, а используемая хэш-функция обладает свойствами криптографической стойкости. В этом случае перед потенциально атакуемой системой стоит задача свести на нет техническую оснащенность злоумышленника, сделав перебор по M_1 экономически неэффективным исходя из реалий текущего уровня развития вычислительной техники.

Уже достаточно давно распространенным приёмом для повышения производительности процессоров (CPU) общего назначения стало использование векторизации посредством расширения системы команд и дополнительных векторных регистров [3]. В этом случае обычный процессор функционирует как специализированный векторный, обрабатывая одной командой составные векторные регистры, размер которых может достигать 512 бит. Более отчетливо подобный подход проявляется в использовании графических ускорителей (GPU) для выполнения векторизированных вычислений, когда количество процессоров может измеряться тысячами, так же как и разрядность общей шины данных может измеряться тысячами бит [4], однако в том и в другом случае вычисления фактически выполняются на множестве ядер, разрядность которых составляет те же 32-64 бит. Если значение не помещается в регистры процессора, то неизбежна потеря скорости вычислений за счёт роста числа вспомогательных операций ввода-вывода. В среднем для CPU операция чтения регистров занимает около 1 такта, чтение кэша первого уровня (L1) - около 2-3 тактов, обращение к оперативной памяти

(RAM) - около 100 тактов [5,6]. При этом всегда остаются варианты для использования программируемой логики или заказных микросхем, но здесь стоимость решения рискует гораздо быстрее оказаться на неприемлемом уровне, чем в случае решений на основе CPU или GPU. Изменяя параметры алгоритма, а не сам алгоритм, можно добиться существенного затруднения перебора на аппаратном уровне, если в процессе вычислений гарантированно появляются "неудобные" для вычисляющего устройства числа, что затруднит перебор даже при значительном количестве ядер. Далее предлагается алгоритм функции хэширования, позволяющий за счет изменения входных параметров управлять размером внутреннего состояния функции, адаптируя её к различным оптимизациям вычислений. Внутреннее состояние здесь представлено результатом преобразования на основе смешанной системы счисления, а именно соотношения

$$A=\sum_{i=0}^{k} z_i \prod_{j=0}^{i-1} n_j \ , \ \prod_{j=0}^{i-1} n_j \geq A, \ n_j \in \mathrm{N}, \ i=1,\ldots,k, \ n_1,\ldots,n_k \in \{1,\ldots,K\} \qquad (1)$$

за счет которого выполняется отображение вида $A \to (z_1,\ldots,z_k)$ и обратно [7,8]. Вообще любое натуральное число A может быть представлено в виде $A=\sum_{i=0}^{k} z_i \prod_{j=0}^{i-1} n_j = z_0 + z_1 n_1 + z_2 n_1 n_2 + \ldots + z_k n_1 n_2 \ldots n_k$, k-натуральное число, где набор $\{n_j\}$- основания системы счисления и $\{z_j\}$ - запись числа A по этим основаниям. В качестве оснований могут быть использованы любые натуральные числа, при этом должно выполнятся условие $\prod_{j=0}^{i-1} n_j \geq A.$ Набор $\{n_j\}$ может состоять из равных n_j , тогда можно говорить о позиционной записи числа по основанию, например 10 в случае n_j=10. Если все либо некоторые значения n_j различны, набор $\{z_j\}$ будет представлять собой цифры числа A, записанного по смешанным основаниям или в смешанной системе счисления. Если выполняется (1), то также справедливо:

$$A_1 = \left[\frac{A}{n_1}\right], \quad A = A_1 n_1 + z_0, \quad z_0 = A - A_1 n_1$$

$$A_2 = \left[\frac{A_1}{n_2}\right], \quad A_1 = A_2 n_2 + z_1, \quad z_1 = A_1 - A_2 n_2 \qquad (2)$$

$$A_3 = \left[\frac{A_2}{n_3}\right], \quad A_2 = A_3 n_3 + z_2, \quad z_2 = A_2 - A_3 n_3$$

.....

$$A_k = \left[\frac{A_{k-1}}{n_k}\right], \quad A_{k-1} = A_k n_k + z_{k-1}, \quad z_{k-1} = A_{k-1} - A_k n_k$$

Очевидно, что $z_k = A_k$. Также очевидно, что если $A, z_0, A - z_0 \in \mathrm{N}$, тогда натуральные решения уравнения

$$z_0 = A - x\left[\frac{A}{x}\right],$$

это все делители числа $A - z_0$, большие z_0. Для полного перебора всех возможных значений A преобразование (1) должно быть выполнено K^k раз. Обратное выполнение преобразования (1) соответствует решению системы уравнений (2) относительно $\{n_j\}$ при известных $(A, \bar{z})$. Если каждое из этих уравнений имеет не более одного решения, то есть решается перебором за K шагов, то вся система (2) решается за Kk шагов. Если же каждое из A_i имеет хотя бы по 2 делителя, то получим уже $\sum_{i=1}^{k} 2^i$ шагов, что вполне достаточно для стойкости относительно атаки перебором. В общем случае число решений системы (2) растёт экспоненциально от длины ключа [9], поэтому преобразование (1) можно рассматривать как одностороннюю (труднообратимую) функцию с секретом, где секретом будет набор оснований $\{n_j\}$. Из всего вышесказанного следует, что на основе смешанной системы

счисления может быть создан алгоритм хэширования, схема одной итерации которого показана на Рис.1.

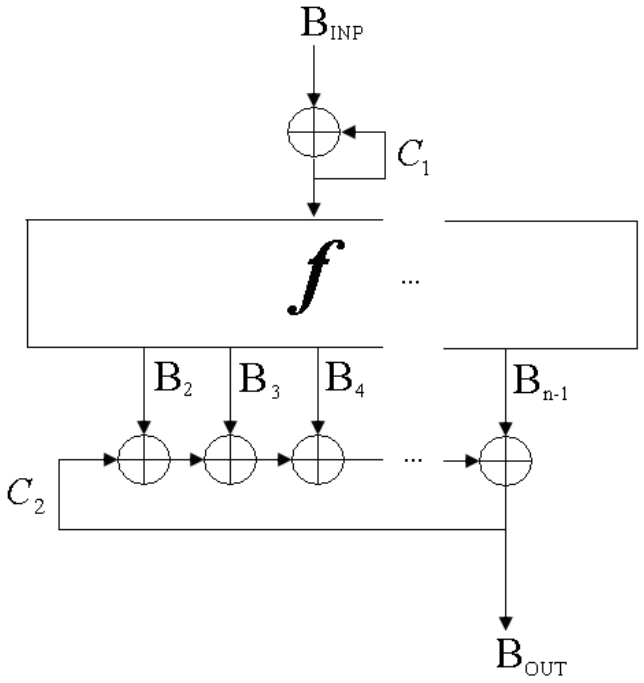

Рис.1. Схема одной итерации алгоритма функции хэширования

Каждая такая итерация состоит из двух преобразований – основного и дополнительного. В качестве основного используется запись в смешанной системе счисления, в качестве дополнительного – преобразование, маскирующее теоретически возможную утечку статистической информации о внутреннем состоянии алгоритма. Статистическая независимость между входным и выходным текстами алгоритма демонстрируется в Приложении 1. Алгоритм работает следующим образом:

1. Задаётся ключевой набор $\{n_j\}$. Также задаются значения C1 и C2, не являющиеся секретом.

2. Функция хэширования принимает на вход блок открытого текста B_{INP} и выполняет $C1' = B_{INP} \oplus C1$, где $\oplus$ - операция сложения по модулю 2. Размер B_{INP} может быть любой, но далее для определённости будет рассматриваться блок длиной в 128 бит.

3. Последовательность ASCII-кодов, соответствующих символьной записи вектора $C1'$, интерпретируется как натуральное A, дополняемое нулями справа

до заданной длины с целью соответствия условию $\prod_{j=0}^{i-1} n_j \geq A,\ i=1,\ldots,k$.

Выполняется функция ***f***, когда на основании (2) находятся все элементы $\{z_i\}$.

4. Последовательно записанные числа набора $\{z_i\}$ интерпретируются как натуральное Z длиной k десятичных знаков, которое затем переводится в двоичный восьмибитный код, после чего эта битовая строка делится на n 128-битных блоков B_q, q=1,…,n, причём последний блок может быть неполным.

Заключительные преобразования для получения результирующего блока:

$$B_{OUT}=C2\oplus B_2\oplus B_3\oplus\ldots\oplus B_{n-1}, \qquad (3)$$

$C2= B_{OUT}$.

Для усиления криптографической стойкости алгоритм использует расширение ключевого набора. Решение предложено М.А. Черепнёвым и ранее не публиковалось. В этом случае каждому n_j ставится в соответствие новое значение по следующему правилу:

$$n_{i,j} = p + jp^{k+(k-1)+\ldots+(i+1)},\ i=1,\ldots,k-1,\ n_{k,j} = p + j,\ j=1,\ldots,K, \qquad (4)$$

где p - наименьшее простое число, для которого p>K.

Пример. Пусть K=3, k=7, p=5, $\{l_i\}=\{1,3,2,2,1,3,2\}$, $i=1,\ldots,K$. Тогда получаем:

$l_1 \to n_{1,1},\ n_{1,1}=5+1*5^{27}$;

$l_2 \to n_{2,3},\ n_{2,3}=5+3*5^{25}$;

$l_3 \to n_{3,2},\ n_{3,2}=5+2*5^{22}$;

.....

$l_7 \to n_{7,2},\ n_{7,2}=5+2$;

Отсюда $\{n_{i,l_i}\}=\{5+3*5^{27},\ 5+3*5^{25},\ 5+2*5^{22},\ \ldots,\ 5+2\}$.

Доказательство однозначности для (1) при таком подходе приведено в Приложении 2.

Поскольку размер внутреннего состояния является основным защитным механизмом алгоритма, далее в качестве демонстрации работы функции ***f*** приведены два примера, при этом значения векторов B_{INP}, C1 и C2 для обоих случаев неизменны и составляют (в шестнадцатиричной записи):

B_{INP}=33 0A 35 0A 37 0A 31 31 0A 31 33 0A 31 37 0A 31

C1= 3F 6B 66 FE AF 5A EC D7 32 07 F3 10 D0 06 40 60

C2= C8 14 3F 53 E8 F5 4E 70 49 DF 78 3F AF E8 F7 0F

Пример 1. K=255, k=5, $\{l_i\}=\{255,255,255,255,255\}$, $i=1,\ldots,K$.

Z_{10}=11971606609289005793550296526149775369295555819131933962796358121
69856891866531937095867404410169293099189313

Z_2 = 10000010011111101011101000100001001111111101101000010000011000010101001010011110100101110001101011111010101111111101001011100111001111010010011111100010101100000010001010000000011010000011111100100100000101101101100010100111000101101111000111010011000000100110100011101000011001000001100011100000000101111010010111001011111101000100110110100011010001000001

Длина итогового десятичного значения Z_{10} составляет 51 десятичный знак, длина битовой строки Z_2 составляет 360 бит.

Пример 2. K=255, k=7, $\{l_i\}=\{255,255,255,255,255,255,255\}$, $i=1,\ldots,K$.

Z_{10}= 27682823244450796043324732768144821246172826797471008548489140283392273236500046440783000221325447932451391522298354119299922413845126343784500383049433572062301064007399253823472916312248401000059249568392427910754642263080244391343875668420833031203062016127204281358323267694156564162

Z_2 = 10111010001010011101100101100001110011110001010110000010010110111011111111110010101011000100000110100011001000100111000101100111011001101110000111000101110010010100010110101011101001011111001010011111111100000001100111000101110101101100111101011011100011101110001100001000001100000011101110101010111110000111001100111111011100100010010011111101000111011110000001001111110010110111111010001110000101

1110011000111111001100011010011110010100110100100011110001010101011010
11011101000011101110111000111010100100010010000000111011010011000000
11111100100111101001000010101100101010011100111011100011011100100111
10011101101011011101110101011001000001100011001110000001101011001010
10110011011011000000001110101011001000011001111110110001111101100110
10111001101010000000011011001100010001110001100010011011011111101011
00110010001100010110111110100001111010010110000110100010001110010010
10100101000101110011010111100011011000001000100101000001111100011011
000010

Длина итогового десятичного значения Z_{10} составляет 287 десятичных знаков, длина итоговой битовой строки Z_2 составляет 952 бит.

Кроме ускоренного перебора за счет использования специальных технических средств известен ряд практических методов криптографического анализа, целью которых в данном случае будет понижение мощности полного перебора для извлечения пароля либо обход аутентификации. Далее будут рассмотрены некоторые из таких методов и механизмы защиты от них [10].

Устойчивость к столкновениям. Столкновением принято называть ситуацию, когда криптоаналитик пытается обнаружить два отличающихся сообщения, дающие одинаковое значение хэш-функции, то есть когда существуют m и m' такие, что $h(m) = h(m')$. В рассматриваемом алгоритме реализована защита от столкновений, поскольку расширяющее ключевой набор правило (4) обеспечивает однозначность прямого и обратного преобразования (1).

Иммунитет к удлинениям. Поскольку практически все известные функции хэширования выполняют $h(m)$ над m поблочно, существует специфическая для хэш-функций атака удлинением сообщения, когда возможно подобрать такое сообщение m', являющееся по сути дополненным (удлиненным) сообщением, при котором будет выполняться $h(m') = h'(h(m), m_{k+1})$, где m_{k+1} определяется перебором. Поскольку рассматриваемая хэш-функция представляет собой одностороннюю функцию с

секретом, незнание ключа (пароля) не позволит злоумышленнику успешно выполнить подобную атаку.

Анализ преднамеренно ослабленного алгоритма. Поскольку прямой доступ криптоаналитика к внутреннему состоянию не облегчит ему задачу нахождения $h(m) = h(m')$, он может попытаться восстановить ключевой набор $\{n_j\}$, что сводится к решению системы (2), однако при использовании расширения ключа (4) такое решение будет единственное, а верхняя оценка числа шагов решения этой системы, как было отмечено выше, при известных $(A, \bar{z})$ составляет не менее K^k. Кроме того, чтобы "обойти" дополнительное преобразование (3) и восстановить набор $\{z_i\}$ необходимо как минимум подобрать два отброшенных блока – B_1 и B_n. При размере блока в 128 бит подобный поиск приведет к 2^{256} попыток подбора $\{z_i\}$ для последующего обращения $A \to (z_1, \ldots, z_k)$, что практически нереализуемо.

В контексте анализа стойкости алгоритма также имеет смысл обратить внимание на параметры K и k, которые также могут быть частью ключа, подобно набору $\{l_i\}$, поскольку несут довольно много информации. Очевидно, что хранение в секрете K и k создает дополнительные проблемы потенциальному криптоаналитику, которому перед тем как пытаться как-либо определить внутреннее состояние придется попытаться найти его размер. При массовом использовании хэш-функции, например для аутентификации на сетевых ресурсах, засекречивание K и k не имеет смысла, хотя в случае каких-либо особо ответственных применений это вполне может быть оправдано. Выбор параметров K и k необходимо осуществлять исходя из принятой модели угрозы, в которой должны быть отражены технические возможности предполагаемого противника, поскольку основное предназначение алгоритма - хэширование паролей для целей аутентификации и противодействие их ускоренному перебору. Любое увеличение количества вычислительных ядер может быть скомпенсировано изменением значений K и k, поскольку не

существует каких-либо принципиальных ограничений для выбора значений K и k, кроме указанных в соотношении (1).

Литература

1. Colin Percival. Stronger key derivation via sequental memory-hard functions. [Электронный ресурс] BSDCan 2009 - The Technical BSD Conference, 09.05.2009 URL: http://www.bsdcan.org/2009/schedule/attachments/87_scrypt.pdf (дата обращения: 10.03.2014)

2. Jean-Phillippe Aumasson. Kudelski Security, Switzerland. Password Hashing: the Future is Now [Электронный ресурс] BLACK HAT USA 2013. 11.07.2013 URL: https://media.blackhat.com/us-13/US-13-Aumasson-Password-Hashing-the-Future-is-Now-WP.pdf (дата обращения: 10.03.2014)

3. Intel Advanced Vector Extensions (Intel AVX) [Электронный ресурс] Корпорация Intel [Офиц. сайт] URL: https://software.intel.com/en-us/isa-extensions/intel-avx?_ga=2.140062833.391382544.1520326774-162056414.1520326735 (дата обращения: 01.03.2018)

4. Видеокарта AMD Radeon R9 серии [Электронный ресурс] 2015 Advanced Micro Devices, Inc [Офиц. сайт] URL: https://www.amd.com/ru-ru/markets/game/products/r9# (дата обращения: 01.03.2018)

5. Infographics: Operation Costs in CPU Clock Cycles [Электронный ресурс] posted September 12, 2016 by "No Bugs" Hare, translated by Sergey Ignatchenko URL: http://ithare.com/infographics-operation-costs-in-cpu-clock-cycles/ (дата обращения: 01.03.2018)

6. Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs [Электронный ресурс] By Agner Fog. Technical University of Denmark. Copyright 1996 – 2017. Last updated 2017-05-02. URL: http://www.agner.org/optimize/instruction_tables.pdf (дата обращения: 01.03.2018)

7. Ноден П., Китте К. Алгебраическая алгоритмика. М.: Мир, 1999. С 426.

8. Панфилов Б.А., Черепнёв М.А., Панфилов Ю.Б. Электронные замки на основе смешанной системы счисления, реализованной с помощью резистивной

матрицы памяти на базе полярнозависимого электромассопереноса в кремнии. // Радиотехника и электроника, 2005, т. 50, № 12, с. 1523-1527

9. Панфилов Б.А., Черепнёв М.А.. Симметричная криптосистема шифрования на основе смешанной системы счисления. // Радиотехника и электроника, 2008 г., т.53, №10, с. 1314-1316

10. Фергюсон Нильс, Шнайер Брюс. Практическая криптография.: Пер. с англ. - Москва, Вильямс, 2004. 432 с. 424 ISBN 5-8459-0733-0

Приложение 1. Статистическое исследование функции хэширования на основе смешанной системы счисления как генератора псевдослучайной последовательности

По Шеннону совершенно стойкая криптографическая система характеризуется статистической независимостью открытого текста и шифротекста [1]. Очевидно, что отсутствие статистического различителя между выходом функции и случайным текстом является необходимым требованием для криптографической функции. С целью доказать практическую неразличимость результатов преобразования на основе ССС от случайной битовой последовательности было выполнено тестирование с помошью специализированного пакета DIEHARD [2]. В качестве входных данных для функции использовалась последовательность простых чисел {3,…,2268743947}, разделенная на 100 текстовых файлов одного размера. Число раундов для функции хэширования с фиксированным ключевым набором было выбрано равным 1, размер блока данных - 128 бит. Для целей тестирования была написана программа, фактически использующая рассматриваемую хэш-функцию как блочный шифр в режиме “электронной кодовой книги” – ECB, в результате выполнения которой было получено 100 бинарных файлов размером 11 Мбайт каждый, что соответствует требованиям DIEHARD. Подобная же программа для тех же входных данных была реализована с использованием алгоритма SHA3 - действующего стандарта США на функцию хэширования, после чего также проводилось тестирование полученных файлов пакетом DIEHARD.

Статистический пакет DIEHARD содержит 16 основных тестов, больше половины из которых сами являются наборами связанных между собой тестов. Например тест OPSO фактически представляет собой 23 теста (1-й раз выбираются биты с 1-го по 10-й, 2-й – со 2-го по 11-й и т.д. до 23-го). В связи с этим были последовательно пронумерованы все тесты из пакета DIEHARD, включая вложенные, всего получилось 146 тестов. При этом порядок перечисления сохранен такой же, как и в выходном файле DIEHARD, а именно:

1. BIRTHDAY SPACINGS TEST
2. OVERLAPPING 5-PERMUTATION TEST
3. BINARY RANK TEST

и так далее. Обобщенные результаты тестирования по обоим преобразованиям представлены на Рис.2 .

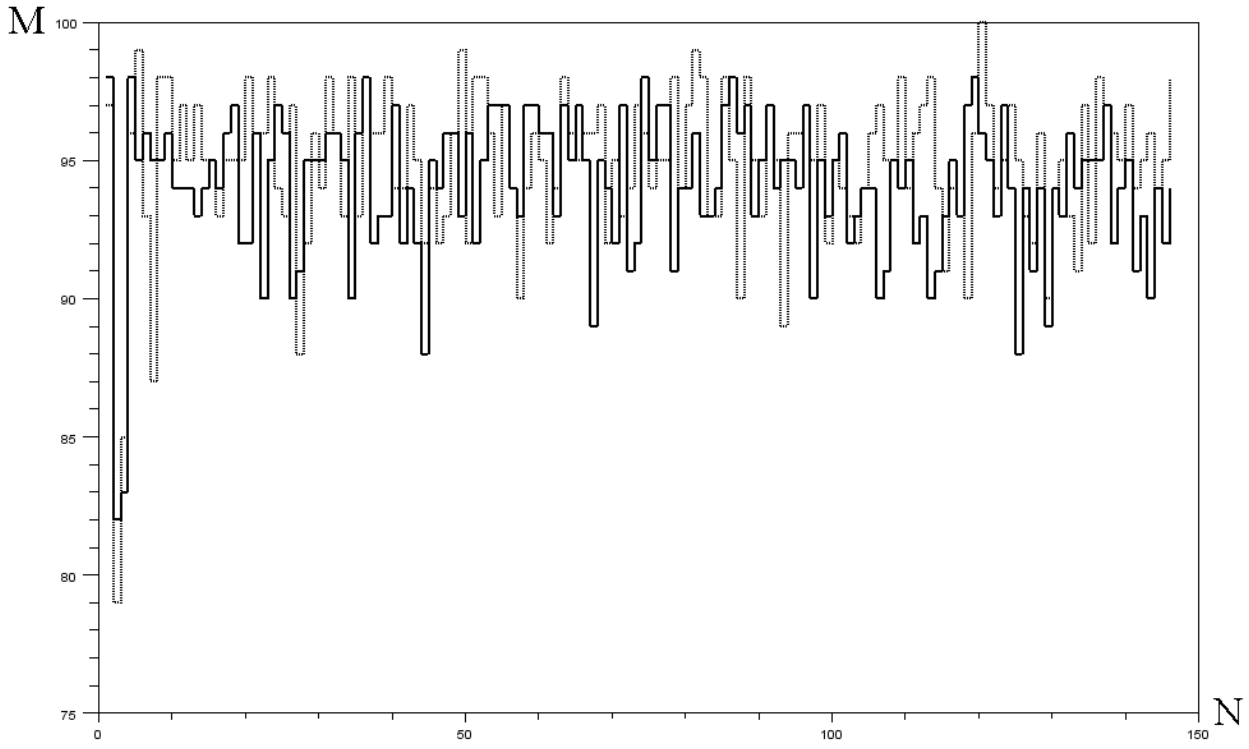

Рис.2. Обобщённые результаты тестирования

На графике по вертикальной оси M отражены частоты успешных прохождений по каждому тесту DIEHARD, по горизонтальной оси N отложены номера тестов в указанном выше порядке, то есть всего 146 тестов. Светло-серая пунктирная линия соответствует результатам для SHA3, более тёмная

отражает результаты тестирования для функции хэширования на основе смешанной системы счисления.

Обе функции как источники псевдослучайных последовательностей показывают в общем высокие и вполне сравнимые друг с другом результаты, так что сложно отдать предпочтение в чью-либо пользу. Исходные тексты программ, результаты вычислений и тестирования доступны в электронном виде по адресу: https://mega.nz/#!2iwmgIjJ!hIMu1uEvMNNNY9kzSEpT0dsAwkJnLtUQU5XOzMTatyQ.

Литература

1. Фомичев В. М. ДИСКРЕТНАЯ МАТЕМАТИКА И КРИПТОЛОГИЯ. Курс лекций / Под общ. ред. д-ра физ.-мат. н. Н. Д. Подуфалова. — М.: ДИАЛОГ-МИФИ, 2003-400 с. ISBN 5-86404-185-8

2. The Marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness [Электронный ресурс] 1995. George Marsaglia. Florida State University. URL: https://web.archive.org/web/20160125103112/http://stat.fsu.edu/pub/diehard/ (дата обращения: 01.03.2018)

Приложение 2. Взаимная однозначность преобразований на основе смешанной системы счисления

М.А. Черепнёв

Пусть имеется набор из k ячеек, которые могут принимать конечное число K значений. Поставим в соответствие i-й ячейке, обладающей j-м значением, натуральное число $n_{i,l_i} \neq 1$. Набор возможных значений $l_1,\ldots,l_k, l_i \in \{0,\ldots,K-1\}$ будем считать секретным ключём, ключ считается верным при выполнении равенства $A = \sum_{i=0}^{k} z_i \prod_{j=1}^{i-1} n_{j,l_j}$. Если выбирать n_{i,l_i}

случайным образом, то вполне вероятно существование другого набора $l_1^{'},\ldots,l_k^{'},l_i^{'}\in\{0,\ldots,K-1\}$, для которого указанное соотношение также будет выполняться, то есть

$$\sum_{i=0}^{k} z_i \prod_{j=1}^{i-1} n_{j,l_j} = \sum_{i=0}^{k} z_i \prod_{j=1}^{i-1} n_{j,l_j^{'}} \qquad (3)$$

Для предотвращения существования еще одного ключевого набора необходимо ввести некоторые правила выбора n_{i,l_i}, при которых отображение $A\to(z_1,\ldots,z_k)$ в обе стороны будет выполняться только для единственного соответствующего набора $\{n_{i,l_i}\}$.

Утверждение. Пусть p - наименьшее простое число, для которого p>K. Если *i*-й набор допустимых оснований вычислять по формуле:

$$n_{i,j} = p + jp^{k+(k-1)+\ldots+(i+1)},\ i=1,\ldots,k-1,\ n_{k,j} = p + j,\ j=1,\ldots,K, \qquad (4)$$

то решение уравнения (1) относительно l_i будет единственно.

Доказательство. Пусть $N_i = p^{k+(k-1)+\ldots+(i+1)}$, $n_{i,j} > K$, $\left|n_{i,j_1} - n_{i,j_2}\right| < KN_i$.

Перепишем уравнение (3) в виде:

$$\sum_{i=0}^{k} z_i \left(\prod_{j=1}^{i-1} n_{j,l_j} - \prod_{j=1}^{i-1} n_{j,l_j^{'}}\right) = 0, \quad \text{где } l_i^{'},l_j^{'}\in\{1,\ldots,K\} \qquad (5)$$

при этом:

$$z_i \in N\cap[1;p-1],\ i=0,\ldots,k-1,\ z_k\in N\cap[1;n-1] \qquad (6)$$

Пусть также j_0 - наибольшее значение индекса при котором $j_{j_0,l_{j_0}} \neq j_{j_0,l^{'}_{j_0}}$.

Рассмотрим равенство (4-5) по модулю N_{j_0-1}. Получим

$$\prod_{j=1}^{j_0-1} p\left(\sum_{i=j_0}^{k} z_i \prod_{j=j_0+1}^{i} n_{j,l_j}\right)\left(n_{j_0,l_{j_0}} - n_{j_0,l^{'}_{j_0}}\right) \equiv 0 (\mathrm{mod}\, N_{j_0-1})$$

Сумма, являющаяся вторым сомножителем, взаимно проста с *p* по построению, поэтому её можно сократить. Получим

$$p^{j_0-1} N_{j_0}\left(l_{j_0} - l_{j^{'}_0}\right) \equiv 0 (\mathrm{mod}\, N_{j_0-1})$$

или

$$l_{j_0} - l_{j'_0} \equiv 0 (\mathrm{mod}\, N_{j_0 - 1}),$$

что по построению невозможно. Утверждение доказано.

Для цитирования:
Ю. Б. Панфилов. Формирование функции хэширования паролей, стойкой к ускоренному перебору значений. Журнал радиоэлектроники [электронный журнал]. 2018. №3. Режим доступа: http://jre.cplire.ru/jre/mar18/12/text.pdf