ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ №3(26), 2015, с. 61–72

УДК 519.682.3

А. А. Кузнецов

Параллельный алгоритм подбора одноблочной MD5-коллизии

Аннотация. В работе описан параллельный алгоритм поиска коллизий хэшфункции MD5 и его имплементация с результатами прогона на вычислительном кластере. Параллельная программа поиска коллизии реализована на языке Си++ с использованием библиотеки MPI. Исходный код программы базируется на последовательной версии пограммы поика коллизий от нидерландского ученого Марка Стивенса. Автор уверен что алгоритм распараллеливания может быть применен для разработки эффективных параллельных программ поиска коллизий хэш-функций, алгоритм работы которых основан на разностном методе Вань. В ходе данного исследования с использованием высокопроизводительного кластера открыта новая пара одноблочных сообщений, MD5-дайджесты которых совпадают (образуют коллизию).

Ключевые слова и фразы: криптоанализ, параллельное программирование, ускорители вычислений, MPI, MD5, хэш-функции.

Введение

Дайджест-функция MD5 [1] принадлежит к семейству хэш-функций, которые основаны на структуре Меркла–Дамгарда. Это одна из наиболее распространенных хэш-функций. Доказано что алгоритм MD5 не является устойчивым к коллизиям, что влечет за собой проблемы нарушения безопасности при его применении [10].

Пару различных сообщений (M, M') называют MD5-коллизией, если их MD5-дайджесты совпадают. В 2004 году китайским исследователем Вань и ее коллегами был открыт разностный метод поиска MD5-коллизий [2], и представлена пара сообщений каждое длиной 1024 бит (двухблочная коллизия). В 2010 году исследователи Xie и

Работы, положенные в основу данной статьи, были выполнены в рамках НИР «Методы и программные средства разработки параллельных приложений и обеспечения функционирования вычислительных комплексов и сетей нового поколения» (№01201354596).

© А. А. Кузнецов, 2015
© Институт программных систем имени А. К. Айламазяна РАН, 2015
© Программные системы: теория и приложения, 2015

Feng впервые опубликовали одноблочную коллизию [3], но из соображений безопасности не стали раскрывать деталей о своем методе поиска коллизий. Они объявили конкурс с призовым фондом в размере $10 000 на поиск новой одноблочной коллизии. В 2012 году нидерландский исследователь Марк Стивенс ответил на вызов, предложив свой алгоритм атаки на MD5 и пару новых сообщений, MD5-дайджесты которых совпадают [4].

В данной работе представлен параллельный алгоритм поиска одноблочной MD5-коллизии и программа, которая реализует его с использованием библиотеки MPI. Алгоритм основан на последовательном (однопоточном) методе Марка Стивенса для поиска коллизий. Также представлена новая пара одноблочных сообщений (длиной по 512 бит), найденная на высокопроизводительном кластере. Поиск занял всего около 11 часов, против 3 недель у последовательной версии.

До недавних пор существовал всего один параллельный метод поиска коллизий. Цитата из [5]: “Согласно этому методу, для выполнения параллельного поиска коллизии каждый процессор выполняет следующую работу. Выбирается начальная точка (сообщение) $x_0 \in S$ и вырабатывается множество точек $x_i = f(x_i - 1)$, для $i = 1, 2, \ldots$ до тех пор, пока не будет найдена выделенная точка x_d. Критерий выбора выделенной точки основывается на некотором легко проверяемом свойстве, таким как число старших нулевых битов. Выделенная точка добавляется в список точек, общий для всех процессоров, затем происходит генерация нового пути с началом в другой точке. Исходя из целевого назначения поиска коллизий, может потребоваться хранить другую информацию вместе с выделенной точкой (например, хранить x_0 и d для быстрого поиска точек таких, что $f(a) = f(b)$). Считается что коллизия найдена если одна и та же выделенная точка дважды вошла в общий список”.

Оригинальная программа Марка Стивенса для поиска коллизий также может быть запущена в параллельном режиме. Она была изначально разработана как однопоточная и допускает распараллеливание по данным через запуск нескольких копий программы на разных ЭВМ с различными произвольно выбранными исходными данными. Каждая копия программы работает с разными начальными данными и своей областью поиска. Для каждого процесса существует одинаковая вероятность обнаружить коллизию. Поэтому запуск N копий программы на N процессорных ядрах дает ускорение поиска в N раз.

В данной работе представлен параллельный алгоритм поиска

коллизий, использующий одни и те же начальные данные для всех копий программы, запущенных на узлах высокопроизводительного кластера.

1. Разностный метод поиска коллизий

В 2004 году Вань и ее коллеги опубликовали описание разностного метода поиска MD5-коллизии [2]. Поиск совершается с использованием техники дифференциального криптоанализа на основе XOR-разностей и модульных разностей (по модулю 232). С использованием этой техники исследователи обнаружили способ построения разностных путей для функции сжатия MD5, которые характеризуют степень влияния разницы между входными сообщениями на внутренние состояния и функции сжатия, что в свою очередь влияет на выходные MD5-дайджесты. Чтобы нужный для коллизии разностный путь был пройден, требуется чтобы внутренние состояния соответствовали так называемым достаточным условиям (бит-условиям).

2. Структура последовательной программы поиска коллизий

Далее приведено краткое описание последовательного алгоритма Марка Стивенса для поиска одноблочной MD5-коллизии (цитата из [4]):

(1) Инициализация: произвольно выбрать значения с $\mathcal{Q}_{14}$ до $\mathcal{Q}_{21}$, удовлетворяющие заданным бит-условиям. Эти значения напрямую влияют на выбор значений m_6, m_{11}, m_0, m_5 и $\mathcal{Q}_1$. Значения с m_0 по m_{15} (каждое по 4 байта) содержат в себе первое сообщение искомой пары сообщений (коллизии).

(2) Предвычисления: сначала порождается таблица поиска, содержащая кортежи подходящих значений с $\mathcal{Q}_2$ до $\mathcal{Q}_7$ и $\mathcal{Q}_{13}$, которые удовлетворяют уравнениям для шагов алгоритма, использующих m_1, m_5, m_6, и заданные бит-условия. Индексами для таблицы поиска служат значения $\mathcal{Q}_7$ и $\mathcal{Q}_{13}$, которые зависят от $\mathcal{Q}_8$ и $\mathcal{Q}_{12}$.

(3) Основной цикл: просмотреть валидные значения с $\mathcal{Q}_8$ по $\mathcal{Q}_{12}$, удовлетворяющие бит-условиям, и шаговому уравнению, использующему значение m_{11}. Найти все значения в таблице поиска, удовлетворяющие всем побочным бит-условиям, связывающим $\mathcal{Q}_7$ и $\mathcal{Q}_8$, и $\mathcal{Q}_{12}$ и $\mathcal{Q}_{13}$. Для каждого из этих значений известны все переменные с $\mathcal{Q}_{-3}$ до $\mathcal{Q}_{16}$, что дает возможность вычислить

все сообщение целиком. Далее вычислить $\mathcal{Q}_{22}$ и $\mathcal{Q}_{23}$, и если они удовлетворяют бит-условиям, то перейти на следующий шаг.

(4) Тоннели: использовать три наиболее подходящих тоннеля для внесения поправок в пару сообщений так, чтобы соблюдались все бит-условия вплоть до $\mathcal{Q}_{23}$. Тоннелем называется такая модификация обоих сообщений, которая не влияет на выполнение бит-условий до некоторого шага вычисления функции компрессии MD5. Для сообщений, удовлетворяющих всем бит-условиям с $\mathcal{Q}_{-3}$ по $\mathcal{Q}_{29}$ проверить являются ли сообщения коллизией.

Далее представлена общая структура (на языке псевдокода) последовательной программы Марка–Стивенса:

Pseudocode

```
main():
        filltables();
        while (true) collinit();
collinit():
        // Инициализация
        вычислить Q_1, Q_4, Q_5, Q_12 − Q_18, m_0, m_5, m_6, m_11;
        вычислить значение Q_3 Q_6 cnt;
        если Q_3 Q_6 cnt < 2^24: вызов collinit();
        четыре вложенных цикла 'do..while':
                во внутреннем цикле 'do..while':
                        вычислить остальные Q_i;
                        вычислить M и M';
                        если md5(M) = md5(M'):
                                // найдена коллизия
                                распечатать (M, M') и Q_i;
                                выход из программы;
```

3. Структура параллельной программы

Параллельная программа создана на базе последовательной версии Марка Стивенса с использованием стандарта передачи сообщений MPI. Стандарт MPI определяет синтаксис и семантику библиотечных функций, используемых при создании параллельных программ на различных языках программирования.

MPI-программа исполняется параллельно путем запуска экземпляра программы на каждом вычислительном ядре каждого узла высокопроизводительного кластера. Каждому процессу присваивается ранг — десятичное число от 0 до $N-1$, где N — число процессорных ядер в кластере.

В параллельной программе поиска MD5-коллизии используются следующие обертки вокруг стандартных MPI-функций:

- mpi_init() – инициализирует MPI-окружение, создает группу процессов;
- mpi_final() – финализирует MPI-окружение, завершает все процессы;
- mpi_send() – посылает массив беззнаковых целых чисел указанному процессу;
- mpi_recv() – принимает массив беззнаковых целых чисел;
- mpi_barrier() – функция барьерной синхронизации, блокирует выполнение программы пока все процессы группы не вызовут ее;
- mpi_size() – возвращает число процессов;
- mpi_rank() – возвращает ранг вызывающего процесса;
- mpi_headrank() – проверяет является ли вызывающий процесс головным (ранг 0).

Каждый вызов представляет собой обертку вокруг собственно MPI-вызова с добавлением проверок на ошибки. Например, mpi_barrier() определен следующим образом:

Pseudocode

```
void mpi_barrier() {
    int rc = MPI_Barrier(MPI_COMM_WORLD);
     if (rc != MPI_SUCCESS) {
         printf("Ошибка в MPI_Barrier\n");
         exit(1);
     }
}
```

Параллельная программа поиска MD5-коллизий имеет следующую структуру (на псевдокоде):

Pseudocode

```
main():
        mpi_init();
        filltables();
        while (true) collinit();
        mpi_final();
collinit():
        mpi_barrier();
        если mpi_headrank(): // ранг 0: инициализация
                // здесь используется генератор случайных чисел:
                вычислить Q_1, Q_4, Q_5, Q_12 − Q_18, m_0, m_5, m_6, m_11;
        если mpi_headrank():
                mpi_send(Q); // разослать Q_i
```

```
13              mpi_send(M); // разослать M
14      иначе:
15              mpi_recv(Q); // рабочие процессы получают Q_i
16              mpi_recv(M); // рабочие процессы получают M
17      вычислить значение Q_3 Q_6 cnt на каждом узле;
18      если Q_3 Q_6 < 2^24: вызов collinit();
19      mpi_barrier();
20      четыре вложенных цикла 'do..while':
21              внутри вложенного цикла 'do..while':
22                      если numIter  mod  mpi_size() = mpi_rank():
23                      // где numIter - число итераций цикла
24                              вычислить остальные Q_i;
25                              вычислить M и M';
26                              если md5(M) = md5(M'):
27                                      // коллизия найдена
28                                      распечатать (M,M') и Q_i;
29                                      mpi_final();
30                                      выход из программы;
```

Обратите внимание на то, что в параллельной программе шаг инициализации выполняется только на головном процессе, после чего значения массивов Q и M рассылаются рабочим процессам. После этого каждый процесс выполняет шаг предвычислений, за которым следует примитив синхронизации (mpi_barrier()). Весь объем вычислений поровну распределяется между всеми процессами. Если какой-либо процесс обнаружил MD5-коллизию, распечатывается результат счета (массивы M и Q) и программа завершается.

4. Запуск параллельного приложения в мультипроцессорной системе

Для целей тестирования разработанного параллельного приложения были задействованы мощности высокопроизводительного вычислительного кластера «Торнадо» Южно-Уральского государственного университета [6]. Во время тестирования было произведено несколько запусков MPI-приложения поиска коллизий, каждый продолжительностью около 24 часов. MD5-коллизия была найдена во время одного из этих запусков. Было задействовано 30 узлов с 12 ядрами на каждом (число рангов — 360). На поиск коллизии потребовалось 10 часов 42 минуты 20 секунд. Напомним, что оригинальная последовательная версия поиска коллизии заняла около 3 недель.

Очевидно, что чем больше узлов будет задействовано в переборе, тем выше вероятность найти коллизию за разумное время (несколько

часов). Время выполнения поиска варьируется от запуска к запуску, поскольку в программе задействован генератор случайных чисел.

Разработанный параллельный алгоритм хорошо масштабируется благодаря тому, что во внутреннем цикле алгоритма все итерации (их несколько миллиардов) распределяются поровну между процессами.

Во время запусков не были задействованы имеющиеся мощности ускорителей (например, Intel Xeon Phi), но это вполне осуществимо. Спустя некоторое время после успешных запусков на кластере, с целью проведения экспериментов были разработаны еще две параллельные версии программы поиска коллизий: многопоточная (multi-threaded) и CUDA-версия.

5. Пара сообщений, образующих коллизию под MD5

В таблице 1 представлена пара одноблочных (длиной по 512 бит) сообщений, которые образуют коллизию под MD5. Сообщения были найдены путем запуска параллельной программы поиска коллизии на вычислительном кластере.

Таблица 1. Сообщения, образующие коллизию под MD5

M	5D 11 69 3E 1E 33 4B 2C B3 88 EF AA F0 D0 EC F3 91 2D 73 0A 1C DD 7A AC 6E 3C E0 E4 CE 06 7B B1 8E 73 C7 **BA** A2 6A A8 19 66 C2 86 16 B3 4F 3D 07 AA B7 C8 1E 32 94 89 **64** 7C 11 73 4A 3F AF 03 EA
M'	5D 11 69 3E 1E 33 4B 2C B3 88 EF AA F0 D0 EC F3 91 2D 73 0A 1C DD 7A AC 6E 3C E0 E4 CE 06 7B B1 8E 73 C7 **BC** A2 6A A8 19 66 C2 86 16 B3 4F 3D 07 AA B7 C8 1E 32 94 89 **E4** 7C 11 73 4A 3F AF 03 EA
Общий MD5-дайджест: 746c4e219320eae3fd23bcf3ebb7d71d	

Эти сообщения доступны в сети Интернет по адресам [7].

В таблице 2 представлен список значений $\mathcal{Q}_i$, вычисленных при поиске коллизии и используемый для генерации сообщения M (все остальные значения $\mathcal{Q}_i$ ($\mathcal{Q}_{30}$–$\mathcal{Q}_{64}$) равны 0).

Сообщение M может быть получено из $\mathcal{Q}_i$ следующим образом:

$$m_t = RR(\mathcal{Q}_{t+1} - \mathcal{Q}_t, RC_t) - AC_t - f_t(\mathcal{Q}_t, \mathcal{Q}_{t-1}, \mathcal{Q}_{t-2}) - \mathcal{Q}_{t-3},$$

Таблица 2. Список значений $\mathcal{Q}_i$

$\mathcal{Q}_{-3}$ = 0x67452301	$\mathcal{Q}_8$ = 0x29F20526	$\mathcal{Q}_{19}$ = 0x410F3F70
$\mathcal{Q}_{-2}$ = 0x10325476	$\mathcal{Q}_9$ = 0x3E1893ED	$\mathcal{Q}_{20}$ = 0x71936434
$\mathcal{Q}_{-1}$ = 0x98BADCFE	$\mathcal{Q}_1$ = 0x00000040	$\mathcal{Q}_{21}$ = 0xF7D2E265
$\mathcal{Q}_0$ = 0xEFCDAB89	$\mathcal{Q}_{11}$ = 0xFFFFFDFE	$\mathcal{Q}_{22}$ = 0x09D6ECD5
$\mathcal{Q}_1$ = 0xD9A89593	$\mathcal{Q}_{12}$ = 0xB62EA109	$\mathcal{Q}_{23}$ = 0xF8B84FB6
$\mathcal{Q}_2$ = 0xDA361481	$\mathcal{Q}_{13}$ = 0x062DA1C8	$\mathcal{Q}_{24}$ = 0xBCCE16A3
$\mathcal{Q}_3$ = 0x0660DFEA	$\mathcal{Q}_{14}$ = 0x1661D7EA	$\mathcal{Q}_{25}$ = 0x463268A8
$\mathcal{Q}_4$ = 0x04812801	$\mathcal{Q}_{15}$ = 0x00050621	$\mathcal{Q}_{26}$ = 0x34EFF95F
$\mathcal{Q}_5$ = 0xEB78D1DC	$\mathcal{Q}_{16}$ = 0x14810A21	$\mathcal{Q}_{27}$ = 0x5E7E0F7D
$\mathcal{Q}_6$ = 0x77D76EFF	$\mathcal{Q}_{17}$ = 0xA8009748	$\mathcal{Q}_{28}$ = 0xE8514E70
$\mathcal{Q}_7$ = 0xBE675C82	$\mathcal{Q}_{18}$ = 0xADABC8E8	$\mathcal{Q}_{29}$ = 0xC677D867

где $RR(X, n)$ — циклический сдвиг вправо на n бит вектора значений X; RC_t — константа сдвига: $(RC_t, RC_{t+1}, RC_{t+2}, RC_{t+3}) = (7, 12, 17, 22)$ для $t = (0, 4, 8, 12)$; AC_t — константа $\lfloor 2^{32} |\sin(t + 1)| \rfloor$; $f_t(X, Y, Z) = F(X, Y, Z) = (X \wedge Y) \oplus (\overline{X} \wedge Z)$.

Сообщение M' может быть получено из M следующим образом:

$$M' = M(0, 0, 0, 0, 0, 0, 0, 0, 0, 2^{25}, 0, 0, 0, 0, 2^{31}, 0, 0).$$

Значения $\mathcal{Q}_i$, приведенные в этом разделе, в общем случае удовлетворяют бит-условиям ([4, таблица 3]), но не все из них. Существуют вспомогательные бит-условия для тоннелей (на значения $\mathcal{Q}_3$, $\mathcal{Q}_4$, $\mathcal{Q}_9$ и $\mathcal{Q}_{14}$), которые ортогональны разностному пути.

Заключение

В данной работе представлен параллельный алгоритм поиска MD5-коллизий, разработанный на основе оригинального алгоритма Марка Стивенса. Алгоритм был реализован в виде параллельного приложения с использованием стандарта MPI, которое было успешно использовано для поиска одноблочной MD5-коллизии.

Разработанный алгоритм может быть применен для разработки эффективных параллельных программ, в которых используется разностный метод Вань.

Оригинальный алгоритм Марка Стивенса имеет вычислительную сложность 2^{50} вызовов подпрограммы md5compress(). Есть уверенность в том, что алгоритм поиска коллизий может быть существенно упрощен. Это предмет для дальнейших исследований.

Параллельная программа поиска коллизий может быть адаптирована для работы на других массивно-параллельных устройствах: многоядерные процессоры, CUDA-ускорители и сопроцессоры Intel Xeon Phi. Это может сильно ускорить поиск коллизий на рабочих станциях и/или вычислительных кластерах.

Список литературы

[1] R. L. Rivest. *The MD5 Message-Digest Algorithm*, Internet Request for Comments: RFC 1321, https://www.ietf.org/rfc/rfc1321.txt, April 1992 ↑ 61.

[2] X. Wang, D. Feng, X. Lai, H. Yu. *Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD*, Cryptology ePrint Archive, Report 2004/199, https://eprint.iacr.org/2004/199.pdf, 2004 ↑ 61, 63.

[3] T. Xie, D. Feng. *Construct MD5 Collisions Using Just A Single Block Of Message*, Cryptology ePrint Archive, Report 2010/643, https://eprint.iacr.org/2010/643, 2010 ↑ 62.

[4] M. Stevens. *Single-block collision attack on MD5*, Cryptology ePrint Archive, Report 2012/040, https://marc-stevens.nl/research/md5-1block-collision/, 2012 ↑ 62, 63, 68.

[5] P. C. Van Oorschot, M. J. Wiener. "Parallel collision search with cryptanalytic applications", *Journal of Cryptology*, **12** (1999), pp. 1–28 ↑ 62.

[6] http://supercomputer.susu.ac.ru/computers/tornado/ ↑ 66.

[7] http://www.botik.ru/~botik/rnd/message1ak, http://www.botik.ru/~botik/rnd/message2ak ↑ 67.

[8] http://marc-stevens.nl/research/md5-1block-collision/ ↑ .

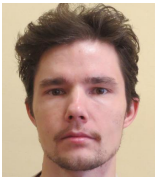
[9] A. A. Kuznetsov. *An algorithm for MD5 single-block collision attack using high-performance computing cluster*, Cryptology ePrint Archive, Report 2014/871, https://eprint.iacr.org/2014/871, 2014 ↑ .

[10] H. Dobbertin. "The status of MD5 after a recent attack", *CryptoBytes*, **2**:2 (1996), 6 p ↑ 61.

[11] А. А. Кузнецов. «Исследование криптостойкости протокола аутентификации BotikKey к компрометации уязвимостей алгоритма хеширования MD5», *Программные системы: теория и приложения*, **6**:1 (2015), с. 135–145, URL http://psta.psiras.ru/read/psta2015_1_135-145.pdf ↑ .

Рекомендовал к публикации *д.ф.-м.н. С. В. Знаменский*

Об авторе:



Антон Александрович Кузнецов

Научный сотрудник ИПС им. А.К. Айламазяна РАН, разработчик системного и прикладного ПО. Один из разработчиков системы параллельного программирования «OpenTS». Область научных интересов: параллельное программирование, компиляторы, распределенные вычисления в гетерогенных средах, геоинформационные системы.

e-mail: tonic@pereslavl.ru

Пример ссылки на эту публикацию:

А. А. Кузнецов. «Параллельный алгоритм подбора одноблочной MD5-коллизии», *Программные системы: теория и приложения*, 2015, **6**:3(26), с. 61–72. URL http://psta.psiras.ru/read/psta2015_3_61-72.pdf

Anton Kuznetsov. *Parallel algorithm for MD5 collision attack.*

Abstract. The parallel algorithm and its implementation for performing a single-block collision attack on MD5 are described. The algorithm is implemented as MPI program based upon the source code of Dr Marc Stevens' collision search sequential program. In this paper we present the parallel single-block MD5 collision searching algorithm itself and details of its implementation together with optimizations. We believe that this algorithm can be further used to derive a program parallelizing method, and for implementing an efficient parallel implementation for an arbitrary collision search program that is based on Wang et al's differential method. We also disclose a pair of new single-block messages colliding under MD5 that were found using our algorithm on the high-performance computing cluster. (*In Russian*).

Key Words and Phrases: cryptanalysis, parallel programming, accelerators, MPI, MD5, hash functions.

References

[1] R. L. Rivest. *The MD5 Message-Digest Algorithm*, Internet Request for Comments: RFC 1321, https://www.ietf.org/rfc/rfc1321.txt, April 1992.

[2] X. Wang, D. Feng, X. Lai, H. Yu. *Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD*, Cryptology ePrint Archive, Report 2004/199, https://eprint.iacr.org/2004/199.pdf, 2004.

[3] T. Xie, D. Feng. *Construct MD5 Collisions Using Just A Single Block Of Message*, Cryptology ePrint Archive, Report 2010/643, https://eprint.iacr.org/2010/643, 2010.

[4] M. Stevens. *Single-block collision attack on MD5*, Cryptology ePrint Archive, Report 2012/040, https://marc-stevens.nl/research/md5-1block-collision/, 2012.

[5] P. C. Van Oorschot, M. J. Wiener. "Parallel collision search with cryptanalytic applications", *Journal of Cryptology*, **12** (1999), pp. 1–28.

[6] http://supercomputer.susu.ac.ru/computers/tornado/.

[7] http://www.botik.ru/~botik/rnd/message1ak, http://www.botik.ru/~botik/rnd/message2ak.

[8] http://marc-stevens.nl/research/md5-1block-collision/.

[9] A. A. Kuznetsov. *An algorithm for MD5 single-block collision attack using high-performance computing cluster*, Cryptology ePrint Archive, Report 2014/871, https://eprint.iacr.org/2014/871, 2014.

[10] H. Dobbertin. "The status of MD5 after a recent attack", *CryptoBytes*, **2**:2 (1996), 6 p.

[11] A. A. Kuznetsov. "On the cryptographic security of the "BotikKey" authentication protocol against attacks on MD5 hash function", *Programmnye Sistemy: Teoriya i Prilozheniya*, **6**:1 (2015), pp. 135–145, URL http://psta.psiras.ru/read/psta2015_1_135-145.pdf.

© A. A. Kuznetsov, 2015
© Ailamazyan Program System Institute of RAS, 2015
© Program systems: Theory and Applications, 2015

Sample citation of this publication:

Anton Kuznetsov. "Parallel algorithm for MD5 collision attack", *Program systems: theory and applications*, 2015, **6**:3(26), pp. 61–72. (*In Russian.*)

URL http://psta.psiras.ru/read/psta2015_3_61-72.pdf