УДК 004.056.55, 004.832.25

ЗАДАЧИ ПОИСКА КОЛЛИЗИЙ ДЛЯ КРИПТОГРАФИЧЕСКИХ ХЕШ-ФУНКЦИЙ СЕМЕЙСТВА MD КАК ВАРИАНТЫ ЗАДАЧИ О БУЛЕВОЙ ВЫПОЛНИМОСТИ

И. А. Богачкова[1], **О. С. Заикин**[2], **С. Е. Кочемазов**[3], **И. В. Отпущенников**[4], **А. А. Семенов**[5], **О. О. Хамисов**[6]

Рассмотрена реализация разностной атаки на криптографические хеш-функции MD4 (Message Digest 4) и MD5 (Message Digest 5) через сведение задачи поиска коллизий для этих хеш-функций к задаче о булевой выполнимости (SAT, SATisfiability). Новизна полученных результатов заключается в том, что предложены существенно более экономные (в сравнении с известными) SAT-кодировки рассматриваемых алгоритмов, а также в использовании для решения полученных SAT-задач современных многопоточных и параллельных SAT-решателей. Задачи поиска одноблоковых коллизий для MD4 в данной постановке оказались чрезвычайно простыми. Кроме того, найдены несколько десятков двухблоковых коллизий для MD5. В процессе соответствующих вычислительных экспериментов определен некоторый класс сообщений, дающих коллизии: построено множество пар дающих коллизии сообщений, у которых первые 10 байтов нулевые.

Ключевые слова: криптографические хеш-функции, коллизии, разностные атаки, задача о булевой выполнимости (SAT), CDCL (Conflict-Driven Clause Learning), параллельные SAT-решатели.

1. Введение. Далее под $\{0,1\}^*$ понимается множество всех возможных двоичных слов произвольной конечной длины. Иными словами, $\{0,1\}^* = \bigcup\limits_{n=0}^{\infty} \{0,1\}^n$, где через $\{0,1\}^n$, $n \in N$, обозначается множество всех двоичных слов длины n (считается, что $\{0,1\}^0 = \emptyset$). Алгоритмически вычислимые и определенные всюду на $\{0,1\}^*$ функции вида $\chi : \{0,1\}^* \to \{0,1\}^C$, где C — некоторая константа, называются хеш-функциями. Можно сказать, что хеш-функция переводит двоичные слова произвольной длины в слова фиксированной длины. Если длина входа n превосходит C, обязательно найдутся такие M, $M' \in \{0,1\}^n$, что $\chi(M) = \chi(M')$. Такая ситуация называется коллизией.

Хеш-функции, помимо их использования для ускорения доступа к данным, находят широкое применение в разнообразных криптографических протоколах. От криптографических хеш-функций требуют дополнительных свойств, заключающихся в вычислительной трудности задач, связанных с их обращением. А именно, должна быть трудной собственно задача обращения: зная $Y \in \{0,1\}^C$, найти произвольный $M \in \{0,1\}^*$, такой, что $\chi(M) = Y$. Кроме того, обычно требуется устойчивость к нахождению коллизий, т.е. трудной должна быть задача нахождения произвольных M и M', таких, что $\chi(M) = \chi(M')$. Существует еще ряд требований к криптографическим хеш-функциям, которые предполагают устойчивость к атакам других видов. В настоящей статье мы исследуем только задачу поиска коллизий.

Мы концентрируемся на решении данной задачи для криптографических хеш-функций семейства MD — а именно, для функций MD4 (Message Digest 4) и MD5 (Message Digest 5) [1, 2]. В работах X. Wang с соавторами [3, 4] описаны разностные атаки, позволяющие за разумное время строить одноблоковые коллизии для функции MD4 и двухблоковые коллизии для функции MD5. Анализу и развитию метода,

[1] Иркутский государственный университет, Институт математики, экономики и информатики, бульвар Гагарина, 20, 664003, г. Иркутск; студент, e-mail: the42dimension@gmail.com
[2] Институт динамики систем и теории управления имени В. М. Матросова, ул. Лермонтова, 134, 664033, г. Иркутск; науч. сотр., e-mail: zaikin.icc@gmail.com
[3] Институт динамики систем и теории управления имени В. М. Матросова, ул. Лермонтова, 134, 664033, г. Иркутск; программист, e-mail: veinamond@gmail.com
[4] Институт динамики систем и теории управления имени В. М. Матросова, ул. Лермонтова, 134, 664033, г. Иркутск; науч. сотр., e-mail: otilya@yandex.ru
[5] Институт динамики систем и теории управления имени В. М. Матросова, ул. Лермонтова, 134, 664033, г. Иркутск; зав. лабораторией, e-mail: biclop.rambler@yandex.ru
[6] Иркутский государственный университет, Институт математики, экономики и информатики, бульвар Гагарина, 20, 664003, г. Иркутск; студент, e-mail: cygx151@gmail.com

© Научно-исследовательский вычислительный центр МГУ им. М. В. Ломоносова

предложенного в этих в работах, в последующие годы был посвящен целый ряд статей (см., например, [5–10]). Метод X. Wang предполагает отслеживание выполнения весьма большого числа условий, и прямая его реализация требует значительных усилий. В статье [11] предложен автоматизированный вариант этого метода для разностной атаки на функции MD4 и MD5. В качестве алгоритмов, которые пытаются найти пары сообщений, удовлетворяющие предложенному разностному пути, в [11] используются SAT-решатели. Следует отметить, что работа [11] является, по-видимому, первым примером успешного применения SAT-подхода к решению задач криптоанализа реально использующихся криптографических систем. Как это ни удивительно, но результаты [11] не получили развития в последующие годы. Основная цель настоящей работы состоит в восполнении этого пробела. Далее мы кратко описываем содержание нашей статьи.

Сначала мы вводим понятия, необходимые для дальнейшей работы, формулируем исследуемые задачи и даем краткий обзор предшествующих результатов. Затем мы описываем механизмы построения пропозициональных кодировок для задач обращения дискретных функций и останавливаемся на особенностях SAT-кодировок алгоритмов MD4 и MD5. Далее мы кратко описываем способы решения трудных SAT-задач, кодирующих обращение криптографических функций. В заключительной части статьи мы описываем вычислительные эксперименты и обсуждаем их результаты.

Основные результаты нашей статьи состоят в следующем. Пропозициональные кодировки, полученные с использованием предлагаемой нами технологии, существенно экономнее известных аналогов. Соответствующие SAT-задачи, кодирующие поиск одноблоковых коллизий для MD4, являются очень легкими даже для последовательных SAT-решателей. Кроме того, при помощи параллельных SAT-решателей мы выделили класс сообщений, на основе которых строятся двухблоковые коллизии для функции MD5. Более точно, с использованием вычислительного кластера мы построили набор двухблоковых коллизий для MD5, в которых первые блоки начинаются с 10 нулевых байтов. В Приложении 2 приведены 10 коллизий указанного типа.

2. Примитивы, лежащие в основе криптографических хеш-функций семейства MD. Существует ряд базовых алгоритмических конструкций, используемых для построения криптографических хеш-функций. Одной из наиболее широко применяемых является конструкция Меркля–Дамгарда [12, 13], лежащая в основе таких известных семейств криптографических функций, как MD и SHA (Secure Hash Algorithm). Данная конструкция предполагает, что исходное сообщение разбито на блоки равной длины (при необходимости длина сообщения увеличивается за счет специальным образом добавляемых битов, называемых “padding”). Обработка исходного сообщения представляет собой последовательное применение функции сжатия к каждому блоку. Функция сжатия преобразует блок в более короткое сообщение, которое сохраняется в специально выделенных регистрах памяти. Значения, записываемые в данных регистрах, рассматриваются как значения “переменных сцепления” (chaining variables). После обработки всех блоков сообщения конкатенация текущих значений переменных сцепления дает значение хеша. В начальный момент времени переменным сцепления присваиваются исходные значения, которые известны и обычно прописаны в спецификации алгоритма. Общая схема конструкции Меркля–Дамгарда представлена на рис. 1 (через $M_1, \ldots, M_N$ обозначены блоки, на которые разбито исходное сообщение, через IV обозначено инициальное значение переменных сцепления, через H_N — итоговое значение хеш-функции от сообщения $M_1|\ldots|M_N$).

Проиллюстрируем работу конструкции Меркля–Дамгарда на примере алгоритма MD5. Работа алгоритма MD5 начинается с дополнения исходного сообщения специальной битовой последовательностью $(p_1, \ldots, p_q)$, $q \geqslant 65$. Бит p_1 всегда равен 1, а биты $p_{q-63}, \ldots, p_q$ являются двоичной записью 64-битного числа, равного числу бит в исходном сообщении. Биты $(p_2, \ldots, p_{q-64})$, если они есть, равны 0. Параметр $q \geqslant 65$ подбирается таким образом, чтобы длина дополненного сообщения была кратна 512 битам. Дополненное таким образом сообщение делится на 512-битные блоки $M = (M_1, \ldots, M_N)$. Согласно схеме Меркля–Дамгарда, итеративный процесс вычисления хеш-функции описывается соотношением $H_i = f(H_{i-1}, M_i)$, $1 \leqslant i \leqslant N$, где $H_0 = IV$ — исходное значение и H_N — значение хеш-функции. Для каждого $i \in \{1, \ldots, N\}$ значение H_i является конкатенацией значений 32-битных переменных сцепления a, b, c и d. В начальный момент времени эти переменные имеют следующие значения (в соответствии со спецификацией MD5):

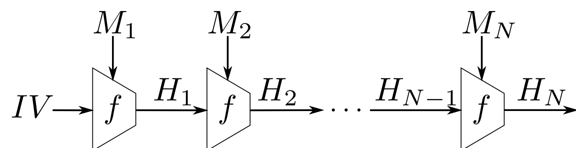

Рис. 1. Конструкция Меркля–Дамгарда

$$a = \texttt{0x67452301}, \quad b = \texttt{0xEFCDAB89}, \quad c = \texttt{0x98BADCFE}, \quad d = \texttt{0x10325476}.$$

Конкатенация этих значений, т.е. слово $a|b|c|d$, образует значение IV.

Схематично работа сжимающей функции f_{MD5}, используемой в алгоритме MD5, представлена на рис. 2. Прокомментируем данный рисунок. На вход f_{MD5} поступает массив, заполненный текущими значениями переменных сцепления — a_0, b_0, c_0, d_0. Соответствующие значения представлены 32-битными словами. Кроме того, функция f_{MD5} оперирует с входным 512-битным блоком M_i, разбитым на 16 слов по 32 бита в каждом: $M_i = (m_1, m_2, \ldots, m_{16})$. Процесс вычисления f_{MD5} условно можно разбить на 4 этапа, называемые раундами. В каждом раунде происходит итеративный пересчет значений переменных сцепления, причем значение каждой переменной обновляется 4 раза (на рис. 2 каждый раунд представлен преобразованием Φ^j, $j \in \{1, 2, 3, 4\}$). Преобразования, соответствующие первому обновлению значений переменных сцепления в первом раунде, задаются следующими соотношениями:

$$
\begin{aligned}
a_1 &= b_0 + \Big(\big(a_0 + \phi^1(b_0, c_0, d_0) + m_1 + t_1\big) <<< s_a^1\Big),\\
d_1 &= a_0 + \Big(\big(d_0 + \phi^1(a_0, b_0, c_0) + m_2 + t_2\big) <<< s_d^1\Big),\\
c_1 &= d_0 + \Big(\big(c_0 + \phi^1(d_0, a_0, b_0) + m_3 + t_3\big) <<< s_c^1\Big),\\
b_1 &= c_0 + \Big(\big(b_0 + \phi^1(c_0, d_0, a_0) + m_4 + t_4\big) <<< s_b^1\Big),
\end{aligned}
\qquad (1)
$$

Здесь “+” означает сложение по $\mathrm{mod}\, 2^{32}$; “$<<<\ s$” — означает s-кратный циклический сдвиг влево 32-битного слова; s_ξ^1, $\xi \in \{a, b, c, d\}$, — известные константы, заданные в спецификации алгоритма (так, $s_a^1 = 7$, $s_d^1 = 12$, $s_c^1 = 17$, $s_b^1 = 22$); константы t_k, $k \in \{1, 2, 3, 4\}$, тоже известны. Результатом второго обновления переменных сцепления в первом раунде являются значения a_2, b_2, c_2 и d_2, вычисляемые по формулам, отличающимся от (1) тем, что в их правых частях фигурируют a_1, b_1, c_1, d_1 вместо a_0, b_0, c_0, d_0 и m_5, m_6, m_7, m_8 вместо m_1, m_2, m_3, m_4, а также используются известные константы t_k, $k \in \{5, 6, 7, 8\}$. Раунд заканчивается после четырех обновлений, т.е. после того как были задействованы все слова $m_1, m_2, \ldots, m_{16}$. Соотношения обновления для последующих раундов в целом аналогичны (1) с тем отличием, что в них используются другие константы и функции ϕ^j. Функции ϕ^j, $j \in \{1, 2, 3, 4\}$, — это функции вида $\phi^j : \{0,1\}^{32} \times \{0,1\}^{32} \times \{0,1\}^{32} \to \{0,1\}^{32}$. Они называются раундовыми функциями и задаются следующим образом:

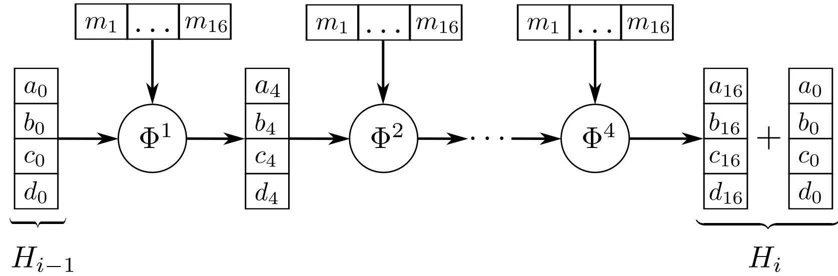

Рис. 2. Схема вычисления сжимающей функции f_{MD5}

$$
\phi^1(X,Y,Z) = (X \vee Y) \wedge (\neg X \vee Z), \quad \phi^2(X,Y,Z) = (X \vee Y) \wedge (Y \vee \neg Z),
$$
$$
\phi^3(X,Y,Z) = X \oplus Y \oplus Z, \quad \phi^4(X,Y,Z) = Y \oplus (X \vee \neg Z).
$$

В последних формулах подразумеваются покомпонентные булевы операции над 32-битными словами.

3. Разностные (дифференциальные) атаки на хеш-функции семейства MD. Под разностными атаками в криптоанализе подразумеваются алгоритмы, которые используют дополнительные соотношения на шифруемые данные, записываемые обычно в виде разностей по модулю некоторого натурального числа. По-видимому, метод дифференциального криптоанализа, описанный в [14], можно рассматривать как первый пример разностной атаки. Основная идея разностных атак заключается в построении “дифференциальных путей”, т.е. наборов разностных условий, выполняющихся в отношении шифруемых данных с некоторыми вероятностями.

Опишем общую схему разностной атаки в применении к задаче поиска коллизии криптографической хеш-функции. Итак, рассмотрим два сообщения $M = (M_1, M_2, \ldots, M_N)$ и $M' = (M'_1, M'_2, \ldots, M'_N)$, $M \neq M'$. Пусть $H_i = a|b|c|d$ и $H'_i = a'|b'|c'|d'$ — значения переменных сцепления после вычисления хешей для блоков сообщений с номером $i \in \{0, 1, \ldots N\}$. Рассмотрим соотношение: $\Delta H_i = a - a'|b - b'|c - c'|d - d'$, в котором фигурируют целочисленные разности по mod 2^{32} между соответствующими значениями переменных сцепления. С учетом этого, дифференциальный путь для рассматриваемой хеш-функции можно определить следующим образом:

$$
\Delta H_0 \xrightarrow{(M_1, M'_1)} \Delta H_1 \xrightarrow{(M_2, M'_2)} \ldots \xrightarrow{(M_N, M'_N)} \Delta H_N = \Delta H.
$$

Очевидно, что $\Delta H_0 = 0$. Если же $\Delta H = 0$, то соответствующие сообщения M и M' образуют коллизию. Если из тех или иных соображений выбран конкретный вид разностей ΔH_i, $1 \leqslant i \leqslant N$, то задача поиска коллизии превращается в задачу подбора пары (M, M'), удовлетворяющей полученному дифференциальному пути. Именно такова основная идея разностных атак на алгоритмы семейства MD, представленных в работах [3, 4]. Более точно, в [3, 4] для алгоритмов семейства MD были предложены разностные пути и описан метод поиска коллизий, состоящий из двух простых этапов: этапа случайного выбора сообщений и этапа детерминированной модификации сообщений с целью подгонки под конкретный дифференциальный путь. Данный метод позволил строить одноблоковые коллизии для алгоритма MD4 (т.е. находить такие пары 512-битных сообщений M и M', что $\Delta H_1 = 0$), а также строить двухблоковые коллизии для алгоритма MD5 (т.е. находить такие пары 1024-битных сообщений M и M', что $\Delta H_2 = 0$). Как уже отмечалось, атаки описанного типа неоднократно применялись в дальнейшем. Из последних достижений в этом направлении следует отметить результаты статьи [10], в которой посредством разностной атаки была построена одноблоковая коллизия для хеш-функции MD5.

4. SAT-подход к задаче обращения дискретных функций и основанные на нем атаки на криптографические хеш-функции. В этом разделе мы описываем элементы техники пропозиционального кодирования алгоритмов, а также приводим основные соображения по применению SAT-подхода к решению задач поиска коллизий криптографических хеш-функций.

Переменные, принимающие значения в множестве $\{0,1\}$, называются булевыми. Формулы вида x и $\neg x$, где x — булева переменная, называются литералами. Литералы x и $\neg x$ называются контрарными. Если X — некоторое множество булевых переменных, то произвольная дизъюнкция литералов над данным множеством, среди которых нет одинаковых и контрарных, называется дизъюнктом над X. Произвольная конъюнкция различных дизъюнктов над X называется конъюнктивной нормальной формой (КНФ).

Рассмотрим алгоритмически вычислимую дискретную функцию $f : \{0,1\}^* \to \{0,1\}^*$, определенную всюду (тотальную) на $\{0,1\}^*$. Алгоритм A_f вычисления f задает счетное семейство тотальных функций вида $f_n : \{0,1\}^n \to \{0,1\}^*$. Задача обращения конкретной f_n в произвольной точке $y \in \mathrm{Range}\, f_n$ состоит в следующем: требуется, зная y и зная алгоритм A_f, найти такой $x \in \{0,1\}^n$, что $y = f_n(x)$. Если алгоритм A_f эффективный (полиномиальный), то данная задача может быть эффективно (в общем случае за полиномиальное от n время) сведена к задаче поиска выполняющего набора выполнимой КНФ. Этот факт является прямым следствием теоремы Кука [15] и может быть объяснен следующим образом. Пусть $N(n)$ — наибольшее число шагов, которые совершает алгоритм A_f на словах из $\{0,1\}^n$. Тогда за время $O\big(N(n)\big)$ мы можем представить функцию f_n в виде комбинационной схемы $S(f_n)$, составленной из функциональных элементов некоторого полного базиса. Затем в общем случае за линейное от размера схемы $S(f_n)$ время делается переход к КНФ C, кодирующей задачу обращения функции f_n в произвольной точке $y \in \mathrm{Range}\, f_n$. Для этой цели используются преобразования Цейтина [16]. Кратко опишем данный процесс.

Итак, пусть $S(f_n)$ — комбинационная схема, представляющая функцию f_n над базисом $\{\neg, \wedge\}$. Далее считаем, что f_n преобразует произвольное входное слово из $\{0,1\}^n$ в выходное слово из $\{0,1\}^m$. Каждому входному полюсу схемы сопоставим отдельную булеву переменную, получив тем самым множество $X = \{x_1, \ldots, x_n\}$. Выходным полюсам схемы аналогичным образом сопоставим множество переменных $Y = \{y_1, \ldots, y_m\}$. Каждому внутреннему узлу схемы соответствует некоторый элемент базиса $\{\neg, \wedge\}$, называемый логическим гейтом. Каждому логическому гейту G сопоставим отдельную переменную $v(G) \notin X$. Множество всех таких переменных обозначим через V. Заметим, что $Y \subseteq V$. Пусть $v(G)$ — произвольная переменная из V и G — соответствующий ей логический гейт. Если G — NOT-гейт и $u \in X \cup V$ — переменная, которая приписана входу G, то с данным гейтом связывается формула $v(G) \equiv \neg u$, где через "$\equiv$" обозначается логическая эквивалентность. Данная формула в КНФ имеет следующий вид:

$$\big(v(G) \vee u\big) \wedge \big(\neg v(G) \vee \neg u\big). \tag{2}$$

Если же G — AND-гейт и $u, w \in X \cup V$ — переменные, приписанные входам G, то с G связывается формула $v(G) \equiv u \wedge w$, выглядящая в КНФ следующим образом:

$$\big(v(g) \vee \neg u \vee \neg w\big) \wedge \big(\neg v(G) \vee u\big) \wedge \big(\neg v(G) \vee w\big). \tag{3}$$

Таким образом, с произвольным гейтом G схемы $S(f_n)$ связана КНФ $C(G)$ вида (2) или (3). Функции f_n сопоставляется КНФ $C(f_n) = \bigwedge\limits_{G \in S(f_n)} C(G)$. Рассмотрим КНФ

$$C(f_n) \wedge y_1^{\beta_1} \wedge \ldots \wedge y_m^{\beta_m}, \quad \text{где} \quad y^\beta = \begin{cases} \neg y & \text{if } \beta = 0, \\ \ \ y & \text{if } \beta = 1. \end{cases} \tag{4}$$

Если $(\beta_1, \ldots, \beta_m) \in \text{Range } f_n$, в силу свойств преобразований Цейтина [16], КНФ (4) выполнима. Если найден выполняющий ее набор, из него эффективно извлекается такой $x \in \{0,1\}^n$, что $f_n(x) = (\beta_1, \ldots, \beta_m)$.

Задача о выполнимости произвольной КНФ сокращенно обозначается как SAT (от SATisfiability). Несмотря на NP-трудность (NP-hard, Non-deterministic Polynomial-time hard), данная задача исключительно важна ввиду обширного спектра ее практических применений. Комбинаторные задачи из, казалось бы, совершенно различных областей эффективно сводятся к SAT-задаче [17]. В последние 10 лет отмечен впечатляющий прогресс в эффективности алгоритмов решения SAT-задач. Одним из достоинств современных SAT-решателей является тот факт, что с их помощью удается решать даже некоторые задачи криптоанализа.

Вопрос применения SAT-подхода к атакам на криптографические хеш-функции начал вызывать интерес с появлением первых скоростных SAT-решателей. Основной аргумент привлекательности такого подхода состоит в том, что соответствующие задачи имеют большое число решений, и, казалось бы, в достаточной степени эффективный алгоритм должен находить хотя бы одно такое решение за приемлемое время. По-видимому, первая попытка реализовать данную идею была предпринята в [20]. Однако стратегия криптоанализа, использованная в этой работе, была весьма прямолинейной и не позволила добиться существенных результатов в отношении реально применяемых криптографических хеш-функций. Статью [11], по-видимому, следует считать первой работой, в которой были продемонстрированы успешные SAT-атаки на реально используемые на тот момент криптографические алгоритмы. Конкретно, в этой работе при помощи SAT-подхода были построены одноблоковые коллизии для алгоритма MD4 и двухблоковые коллизии для MD5. Основная идея заключалась в том, что к КНФ, кодирующим работу алгоритмов MD4 и MD5, добавлялись дизъюнкты, представляющие, по сути, условия X. Wang на попадание в дифференциальный путь. Из текста [11] можно сделать вывод, что трудоемкость описанной SAT-атаки в применении к поиску двухблоковых коллизий для MD5 была весьма высока и, по сути, не давала выигрыша в сравнении с дифференциальной атакой из [4]. Как уже было отмечено выше, результаты статьи [11] не получили дальнейшего развития. Особенно удивительным это кажется в свете достигнутого в последние годы существенного прогресса в области параллельных алгоритмов решения SAT-задач. Основная цель настоящей статьи заключается в том, чтобы воспроизвести и улучшить результаты работы [11] на основе новой техники пропозиционального кодирования алгоритмов и параллельных методов решения SAT-задач.

5. Система Transalg трансляции алгоритмов вычисления дискретных функций в SAT. Система Transalg [21, 22] (http://gitorious.org/transalg/) была разработана специально для автоматизации процесса сведения к SAT задач обращения дискретных функций, заданных алгоритмическими описаниями. При этом были использованы идеи С. Кука по пропозициональному кодированию алгоритмов [15], а также идеи Дж. Кинга по символьному исполнению программ [23]. В качестве языка описания в Transalg используется проблемно-ориентированный язык TA, имеющий C-подобный синтаксис. При трансляции TA-описаний используются стандартные техники теории компиляции. Результатом компиляции TA-программы является не машинный код, а множество булевых формул, которому естественным образом соответствует система булевых уравнений. От данного множества формул делается переход к SAT-задаче при помощи преобразований Цейтина.

Итак, начальный этап построения SAT-задачи, кодирующей проблему обращения некоторой дискретной функции, состоит в описании данной функции на специализированном языке TA. Язык TA является процедурным проблемно-ориентированным языком программирования с C-подобным синтаксисом [24]. Таким образом, обычно достаточно внести минимальные правки в существующую реализацию алгоритма на языке C, чтобы получить соответствующую TA- программу.

Везде далее под трансляцией мы понимаем процесс построения пропозициональной кодировки дискретной функции по вычисляющей ее TA-программе. Трансляция любой TA-программы состоит из двух основных этапов. На первом этапе Transalg разбирает текст TA-программы и строит дерево синтаксического разбора, используя стандартные техники теории компиляции [25]. На втором этапе Transalg реализует концепцию символьного исполнения [23] для построения пропозициональной кодировки обрабатываемой TA-программы. Получаемый пропозициональный код может выдаваться в стандартных формах КНФ, ДНФ (Дизъюнктивная Нормальная Форма) и АНФ (Алгебраическая Нормальная Форма).

Программа на языке TA имеет блочную структуру. Блоком (составным оператором) называется произвольный список инструкций, заключенный в фигурные скобки. Каждый блок образует локальное пространство имен. Блок, определенный внутри другого блока, называется вложенным. В языке TA глубина вложения блоков не ограничивается. В процессе анализа текста программы конструируется дерево областей видимости. Корнем такого дерева является глобальное пространство имен. Каждый идентификатор

ТА-программы относится к некоторому пространству имен. Переменные и массивы, определенные вне блоков, а также все функции относятся к глобальному пространству имен и доступны в любой точке программы (порядок объявления не важен).

Любая ТА-программа по сути является списком функций. Специальная функция main является точкой входа и поэтому должна существовать в любой ТА-программе. Язык ТА поддерживает базовые конструкции, характерные для процедурных языков программирования (объявление переменных, массивов и функций; оператор присваивания; оператор условного перехода; циклы; вызов функции), арифметико-логические операции над целыми числами, побитовые операции, включая операции битового сдвига, и операции сравнения чисел.

Основным типом данных языка ТА является тип `bit`. Система Transalg использует этот тип данных для установления связей между переменными, используемыми в ТА-программе, и булевыми переменными, входящими в получаемый пропозициональный код. Важно различать эти два множества переменных. Везде далее мы будем называть переменные, которые появляются в ТА-программе, *переменными программы*, а все булевы переменные, входящие в пропозициональный код, будем называть *переменными кода*. В процессе трансляции инструкций ТА-программы, содержащих переменные типа `bit`, система Transalg связывает эти переменные программы с соответствующими переменными кода. Подчеркнем, что Transalg устанавливает такие связи только для переменных программы типа `bit`. Переменные других типов, а именно, типа `int` и типа `void` используются в качестве служебных переменных — например, для счетчиков циклов или для определения типа возвращаемого функцией значения.

Глобальные переменные типа `bit` могут быть определены с атрибутом `__in` или `__out`. Атрибут `__in` отмечает переменные программы, которые содержат входные данные для алгоритма. При помощи атрибута `__out` помечаются переменные, которые должны содержать выход алгоритма. Локальные переменные типа `bit` не могут объявляться с этими атрибутами.

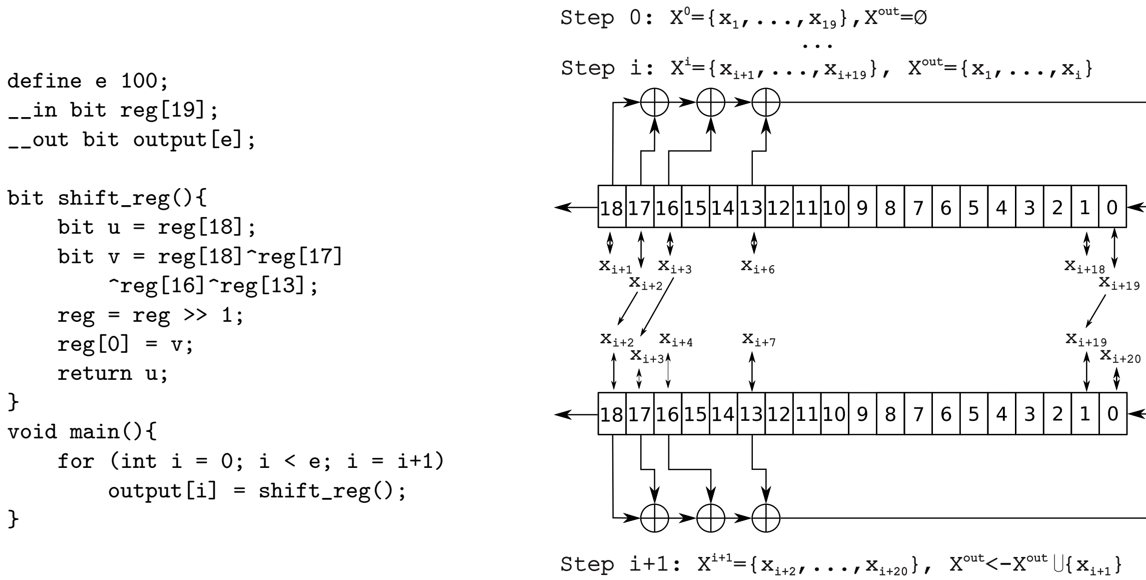

Рис. 3. Пропозициональное кодирование РСЛОС

Рассмотрим последовательность вычислений, задаваемую ТА-программой, в виде последовательности модификаций данных в памяти абстрактной вычислительной машины в моменты времени $0, 1, \ldots, e$. В каждый момент $i \in \{0, 1, \ldots, e\}$ система Transalg связывает множество переменных кода X^i с переменными программы типа `bit`. Обозначим множество всех переменных кода через $X = \bigcup_{i=0}^{e} X^i$. Полагаем, что множество X^{in} состоит из переменных кода, которые соответствуют входным данным, и множество X^{out} состоит из переменных кода, соответствующих выходу рассматриваемой дискретной функции. Таким образом, $X^{\text{in}} \subseteq X^0$ и $X^{\text{out}} \subseteq X$.

Система Transalg использует концепцию трансляции, которая позволяет существенно сокращать избыточность пропозиционального кода. Проиллюстрируем данную концепцию на примере пропозиционального кодирования широко используемого криптографического примитива — регистра сдвига с линейной обратной связью (РСЛОС; англ. Linear Feedback Shift Register, LFSR) [26]. На рис. 3 показана ТА-программа, описывающая функционирование РСЛОС с полиномом обратной связи $P(z) = z^{19} + z^{18} +$

$z^{17}+z^{14}+1$ над $GF(2)$ (здесь z — формальная переменная).

Отметим, что при трансляции перехода от шага i к шагу $i+1$ не для всех ячеек регистра необходимо создание новых переменных кода. Действительно, на самом деле новая информация возникает только при вычислении бита обратной связи (соответствующий бит записывается в ячейку с номером 0). Содержимое произвольной ячейки с номером $i \in \{0,1,\ldots,17\}$ переписывается в ячейку с номером $i+1$, а содержимое ячейки с номером 18 выдается в ключевой поток. Таким образом, для переменных программы reg [1], ..., reg [18], которые связаны с ячейками с номерами от 1 до 18, нет смысла создавать новые переменные кода: если $x_{i+19}, x_{i+18}, \ldots, x_{i+2}$ — переменные кода, которые были связаны с reg [0], ..., reg [17] на шаге с номером i (см. рис. 3), то в точности эти же переменные кода связываются на шаге с номером $i+1$ с переменными reg [1], ..., reg [18]. Transalg отслеживает подобные ситуации за счет специальной структуры данных, названной словарем термов.

В применении к рассматриваемому примеру множеством переменных кода, кодирующих начальное заполнение регистра, является $X^{\text{in}} = X^0 = \{x_1, x_2, \ldots, x_{19}\}$. После каждого сдвига кодируем значения ячеек регистра множествами $X^1 = \{x_2, x_3, \ldots, x_{20}\}$, $X^2 = \{x_3, x_4, \ldots, x_{21}\}$, ..., $X^e = \{x_{e+1}, x_{e+2}, \ldots, x_{e+19}\}$. Очевидно, что $X^{\text{out}} = \{x_1, x_2, \ldots, x_e\}$.

Таким образом, множеством переменных кода данной ТА-программы будет являться множество $X = \{x_1, x_2, \ldots, x_{e+19}\}$, а полученный пропозициональный код — это следующее множество булевых формул: $x_{20} \equiv x_1 \oplus x_2 \oplus x_3 \oplus x_6$, ..., $x_{e+19} \equiv x_e \oplus x_{e+1} \oplus x_{e+2} \oplus x_{e+5}$. Получаемый таким способом пропозициональный код может быть преобразован в КНФ при помощи преобразований Цейтина.

В общем случае Transalg комбинирует преобразования Цейтина с использованием стандартных процедур булевой минимизации, включенных в библиотеку Espresso, исходный код которой на языке C является открытым (http://embedded.eecs.berkeley.edu/pubs/downloads/espresso/index.htm). Библиотека Espresso интегрирована в систему Transalg в виде отдельного программного модуля. В Espresso процедуры минимизации булевых формул манипулируют таблицами истинности этих формул. Соответственно, сложность процедуры минимизации формулы растет экспоненциально от числа переменных в формуле. На практике это означает, что для того чтобы Espresso справлялся с минимизацией формул за приемлемое время, требуется ограничивать множество булевых переменных, над которыми построена формула, 10–12 переменными. В случае если в ходе трансляции ТА-программы возникают формулы над бóльшим числом переменных, для их разбиения на подформулы применяются преобразования Цейтина.

6. Пропозициональное кодирование хеш-функций семейства MD. Сжимающие функции алгоритмов MD4 и MD5 очень похожи. В этих функциях используются следующие элементарные операции: целочисленное сложение по модулю 2^{32}, циклический битовый сдвиг, а также побитовые операции отрицания, конъюнкции, дизъюнкции и сложения по модулю 2.

Пусть $a = (a_n, \ldots, a_1)$ и $b = (b_n, \ldots, b_1)$ — n-битовые целые числа (используем запись, в которой старший бит — крайний левый). Операция целочисленного сложения $a+b$ кодируется следующим множеством булевых формул:

$$\begin{array}{llll} c_1 \equiv a_1 \oplus b_1, & & p_1 \equiv a_1 \wedge b_1, & \\ c_i \equiv a_i \oplus b_i \oplus p_{i-1}, & i=\overline{2,n}, & p_i \equiv a_i \wedge b_i \vee a_i \wedge p_{i-1} \vee b_i \wedge p_{i-1}, & i=\overline{2,n}, \\ c_{n+1} \equiv p_n. & & & \end{array}$$

Здесь p_i, $i=\overline{1,n}$, — биты переноса, $c = a+b = (c_{n+1}, c_n, \ldots, c_1)$ — результат сложения. В случае операции целочисленного сложения по модулю 2^{32} числа a и b представлены 32-битными векторами ($n=32$), а результатом являются младшие 32 бита вектора c.

При кодировании операций циклического сдвига Transalg меняет связи между переменными программы и соответствующими переменными кода так, как это было описано в разделе 5. Подчеркнем, что новые переменные кода при этом не создаются.

Операции, фигурирующие в выражениях для раундовых функций, являются побитовыми. Таким образом, пропозициональный код, например, для функции $\phi^1(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$ имеет вид $u_i \equiv (x_i \wedge y_i) \vee (\neg x_i \wedge z_i)$, $i=\overline{1,32}$, где $X=(x_1,\ldots,x_{32})$, $Y=(y_1,\ldots,y_{32})$, $Z=(z_1,\ldots,z_{32})$.

Рассмотрим пропозициональное кодирование задачи поиска коллизий хеш-функций семейства MD при помощи системы Transalg. Для этой цели используем ТА-программу, вычисляющую рассматриваемую хеш-функцию (пример такой программы приведен в приложении 1). Фактически используются две отдельных SAT-кодировки процесса вычисления хеш-функции. Затем в полученную КНФ добавляются условия равенства значений переменных, кодирующих хеш. Следует отметить, что полученные таким образом SAT-задачи являются вычислительно трудными, несмотря на тот факт, что соответствующая КНФ имеет большое число выполняющих наборов. Поэтому к полученной КНФ добавляются дизъюнк-

ты, кодирующие дифференциальный путь и, при необходимости, дополнительные условия (например, “bit conditions” [3, 4]). Средства системы Transalg позволяют легко осуществить все перечисленные выше операции.

7. Алгоритмы решения SAT и их применение к задачам криптоанализа. В исходной постановке задача о выполнимости булевой формулы (SAT) состоит в том, чтобы по произвольной булевой формуле F решить, является ли эта формула выполнимой (т.е. найдется ли набор значений переменных, входящих в эту формулу, на котором она истинна). При помощи преобразований Цейтина за полиномиальное время данная задача сводится к задаче о выполнимости КНФ $C(F)$ над множеством переменных, которое, вообще говоря, включает в себя переменные, входящие в F. Таким образом, мы всегда можем рассматривать SAT как задачу о выполнимости некоторой КНФ. В силу теоремы Кука SAT-задача NP-полна; следовательно, поисковый вариант SAT (т.е. ответить “нет”, если КНФ невыполнима, и выдать выполняющий ее набор в противном случае) является NP-трудной. Сказанное означает, что мало надежды на то, что для решения SAT существуют детерминированные алгоритмы с полиномиальной трудоемкостью. Тем не менее, “эффективные” (в плане реального времени) алгоритмы решения SAT необходимы в целом ряде важных практических областей, главной из которых является формальная верификация.

Эта востребованность привела в последние годы к настоящему “буму” в разработке различных стратегий и эвристик для решения SAT. Существует ряд общих концепций, которые закладываются в основу современных SAT-решателей. Одной из наиболее плодотворных стала концепция CDCL (Conflict Driven Clause Learning), впервые описанная в [27]. Поскольку именно CDCL-решатели показывают самую высокую эффективность на задачах обращения криптографических функций, далее мы кратко остановимся на описании их конструктивных особенностей. В основе CDCL решателя лежит алгоритм DPLL (Davis–Putnam–Logemann–Loveland) [28], дополненный техникой Clause Learning, которая позволяет сохранять информацию о пройденных фрагментах пространства поиска в виде конфликтных дизъюнктов. Конфликтный дизъюнкт является логическим следствием исходной КНФ C; поэтому его конъюнкция с C дает КНФ C', выполнимую в точности на тех же наборах, что и C (либо же обе эти КНФ одновременно невыполнимы). В дальнейшем конфликтный дизъюнкт участвует в выводе новой информации и, тем самым, направляет поиск по новому пути. Алгоритм DPLL и алгоритм DPLL+CDCL экспоненциальны в худшем случае: это следует из экспоненциальности метода резолюций [29] и результатов статьи [30]. Тем не менее, современные CDCL-решатели на удивление успешны при работе с “индустриальными” SAT-задачами огромных размерностей (десятки тысяч переменных, сотни тысяч и миллионы дизъюнктов).

Впервые идея использовать задачи обращения криптографических функций для построения аргументированно трудных SAT-задач была высказана С. Куком и Дж. Митчелом в [31]. Работа [32] содержит, по-видимому, первый пример пропозиционального кодирования криптографической функции. Следует отметить, что построенные в [32] и некоторых последующих работах SAT-задачи оказывались слишком сложными и, таким образом, не позволяли осуществить криптоанализ соответствующих криптосистем. Как уже отмечалось, по-видимому, в [11] содержится первый пример успешной SAT-атаки на реально используемые криптографические алгоритмы.

В статье [33] приведены примеры успешного применения SAT-подхода к некоторым криптографически слабым генераторам ключевого потока. В [34] описана техника распараллеливания SAT, которая показала хорошие результаты в применении к задачам криптоанализа ряда поточных шифров. В статье [35] представлена оценка вычислительных ресурсов, требуемых для распределенного решения SAT-задач, кодирующих криптоанализ неослабленного генератора A5/1. Реализация техники из [34, 35] в грид-системе BNB-Grid [36] позволила летом 2009 г. осуществить криптоанализ неослабленного генератора ключевого потока A5/1 [37]. Объем вычислительных ресурсов, затраченных в этих экспериментах, хорошо согласуется с оценкой из [35]. Отметим, что это было сделано примерно на полгода раньше появления первых Rainbow-таблиц для криптоанализа A5/1, которые были построены в рамках проекта A5/1 Cracking Project (https://opensource.srlabs.de/projects/a51-decrypt) в декабре 2009 г. Дополненные и улучшенные результаты работы [37] были опубликованы в англоязычном издании только в 2011 г. [19]. В дальнейшем технология, описанная в [19, 34, 37], эволюционировала в метод Монте-Карло оптимизации специальных прогнозных функций [38]. Декомпозиционные множества, которые находятся этим методом, используются для крупноблочного распараллеливания SAT-задачи с последующей обработкой получаемого параллельного списка заданий на кластере или в грид-системе. В последнее время для этой цели используется проект добровольных распределенных вычислений SAT@home [39, 40].

8. Вычислительные эксперименты. В данном разделе мы описываем результаты наших вычислительных экспериментов по применению SAT-подхода к задачам построения коллизий криптографических хеш-функций MD4 и MD5. Пропозициональное кодирование алгоритмов, добавление ограничений

дифференциальных путей и достаточных условий из работ [3, 4] осуществлялось при помощи системы Transalg. Использовались следующие SAT-решатели:

— MiniSat (http://minisat.se/),

— CryptoMiniSat (https://github.com/msoos/cryptominisat),

— Plingeling,

— Treengeling (http://fmv.jku.at/lingeling/),

— PDSAT (https://github.com/Nauchnik/pdsat).

На первом этапе мы рассмотрели задачу поиска коллизий для хеш-функции MD4. В табл. 1 приведены параметры SAT-кодировок задачи поиска одноблоковой коллизии для хеш-функции MD4 (с учетом дифференциальных путей из [3]).

Таблица 1

Число	Кодировки из работы [11]	Transalg
переменных	53 228	19 363
дизъюнктов	221 440	184 689

Таблица 2

Число коллизий	Время поиска (в секундах)
10	30
100	110
1000	610
10 000	5200

SAT-задачи, построенные системой Transalg, оказались очень легкими даже для последовательных решателей: SAT-решатель MiniSat версии 2.2 тратил на поиск одной коллизии не более нескольких секунд. Затем мы использовали SAT-решатель CryptoMiniSat, имеющий функцию перечисления решений. В табл. 2 приведены данные по скорости поиска коллизий MD4 SAT-решателем cryptominisat в однопоточном режиме (1 ядро Intel i5-3570T, 4 Gb RAM). Так, на поиск 10 000 коллизий для MD4 CryptoMiniSat затратил менее двух часов.

Мы исследовали задачи построения двухблоковых коллизий для функции MD5. Процесс решения каждой такой задачи разбивается на два этапа. На первом этапе мы ищем два 512-битных блока M_1 и M_1', разность хешей которых есть (`0x80000000, 0x82000000, 0x82000000, 0x82000000`) в соответствии с условиями на дифференциальный путь из статьи [4]. Мы фиксируем найденные хеши, обозначая их через H_1 и H_1'. Затем мы ищем вторые 512-битные блоки M_2 и M_2', такие, что $f_{\text{MD5}}(H_1, M_2) = f_{\text{MD5}}({H'}_1, M_2')$. В итоге имеем двухблоковую коллизию. В табл. 3 приведены параметры SAT-кодировок задачи поиска блоков M_1 и M_1', построенных системой Transalg, и приводится их сравнение с кодировками из работы [11].

Таблица 3

Число	Кодировки из работы [11]	Transalg
переменных	89 748	35 477
дизъюнктов	375 176	304 728

Задача поиска блоков M_1 и M_1' довольно сложна для SAT-решателей даже с учетом условий на дифференциальный путь. Дополнительно мы добавляли в кодировку ряд достаточных условий (“bit conditions”), приведенных в статье [4]. Попутно было выявлено, что некоторые из этих условий некорректны — при их добавлении получались невыполнимые тесты. Все такие условия в SAT-кодировку не включались. Для решения полученных (с учетом всех перечисленных нюансов) SAT-задач мы использовали многопоточные SAT-решатели Plingeling и Treengeling. Каждый из этих решателей запускался на одном рабочем узле кластера “Академик В. М. Матросов” Иркутского суперкомпьютерного центра СО РАН (http://hpc.icc.ru/, 2 процессора Opteron6276, всего 32 ядра). Время работы данных решателей на задаче поиска блоков M_1 и M_1' весьма существенно варьировалось: от нескольких часов до нескольких суток в различных запусках. Данный эффект объясним тем фактом, что эти решатели используют стратегии рандомизации поиска.

По результатам экспериментов было сделано следующее наблюдение: решатель Treengeling находил блоки M_1 и M_1' с большим числом нулей в начале. Более детальное исследование показало, что означивание нулями первых десяти байтов блоков M_1 и M_1' дает выполнимый тест, тогда как фиксация первых одиннадцати нулевых байтов этих блоков дает невыполнимую КНФ (невыполнимость доказывается очень быстро). Таким образом, мы выделили класс пар вида M_1, M_1', удовлетворяющих разностному пути из [4], в которых первые 10 байтов нулевые. Затем мы исследовали вопрос, можно ли построить за приемлемое время несколько таких пар. Для решения данной задачи мы использовали параллельный SAT-решатель PDSAT [41, 42], разработанный специально для решения SAT-задач, распараллеленных

в соответствии с Partitioning-концепцией [43, 44]. Решатель PDSAT был запущен на 15 узлах кластера “Академик В. М. Матросов” (т.е. суммарно на 480 ядрах) и за 89 часов работы нашел 20 пар вида M_1, M_1', удовлетворяющих разностному пути из [4], у которых начальные 10 байтов нулевые. На основе каждой такой пары мы строили несколько различных двухблоковых коллизий для функции MD5. Для этой цели мы использовали решатель Plingeling, запуская несколько его независимых копий на различных узлах кластера “Академик В. М. Матросов”. Задача поиска пар вида M_2, M_2' существенно проще задачи поиска M_1 и M_1' — в среднем одна пара вида M_2, M_2' находилась решателем Plingeling за 400 секунд работы на одном узле данного кластера. В Приложении 2 мы приводим 10 коллизий описанного выше вида (с нулевыми первыми 10 байтами).

9. Заключение. В настоящей статье для криптографических хеш-функций MD4 и MD5 мы описали разностные атаки в форме задач о выполнимости булевых формул (SAT). При помощи системы Transalg были построены SAT-кодировки задач поиска коллизий для данных хеш-функций, которые оказались существенно экономнее известных аналогов. Мы использовали многопоточные и распределенные SAT-решатели для вычислительной реализации описанных атак. SAT-задачи, кодирующие поиск одноблоковых коллизий для MD4, оказались чрезвычайно простыми даже для последовательных решателей. С использованием вычислительного кластера за разумное время были построены несколько десятков пар двухблоковых коллизий для MD5 с начальной частью фиксированного вида (с первыми десятью нулевыми байтами).

В ближайшем будущем мы планируем применить распределенные SAT-решатели к задаче обращения хеш-функции MD4 (т.н. “preimage attack”), преследуя цель улучшить результаты работы [45]. Предполагается, что в вычислительных экспериментах будет задействован проект добровольных вычислений SAT@home (http://sat.isa.ru/pdsat/).

Работа выполнена при частичной поддержке грантов РФФИ (14–07–00403а, 14–07–31172мол-а, 15–07–07891а); стипендии Президента РФ СП–3667.2013.5; Совета по грантам Президента РФ для гос. поддержки ведущих научных школ (НШ–5007.2014.9).

Приложение 1. TA-программа, вычисляющая сжимающую функцию алгоритма MD4.

```
__in bit M[16][32];
__out bit Out[4][32];

// Инициализируем переменные сцепления
bit A[32] = 0x67452301;
bit B[32] = 0xEFCDAB89;
bit C[32] = 0x98BADCFE;
bit D[32] = 0x10325476;

// Раундовые функции
bit F(bit X[32], bit Y[32], bit Z[32])
{
    return (X&Y)|(!X&Z);
}
bit G(bit X[32], bit Y[32], bit Z[32])
{
    return X&Y | X&Z | Y&Z;
}
bit H(bit X[32], bit Y[32], bit Z[32])
{
    return X^Y^Z;
}

bit FF(bit a[32], bit b[32], bit c[32], bit d[32], bit M[32], int s)
{
    a = sum(sum(a, F(b, c, d), 32), M, 32);
    return (a <<< s);
}
bit GG(bit a[32], bit b[32], bit c[32], bit d[32], bit M[32], int s)
{
```

```
    a = sum(sum(sum(a, G(b, c, d), 32), M, 32), 0x5A827999, 32);
    return (a <<< s);
}
bit HH(bit a[32], bit b[32], bit c[32], bit d[32], bit M[32], int s)
{
    a = sum(sum(sum(a, H(b, c, d), 32), M, 32), 0x6ED9EBA1, 32);
    return (a <<< s);
}

void main()
{
    bit a[32] = A; bit b[32] = B;
    bit c[32] = C; bit d[32] = D;

   ~// Round 1
    a = FF(a, b, c, d, M[0], 3);    d = FF(d, a, b, c, M[1], 7);
    c = FF(c, d, a, b, M[2], 11);   b = FF(b, c, d, a, M[3], 19);

    a = FF(a, b, c, d, M[4], 3);    d = FF(d, a, b, c, M[5], 7);
    c = FF(c, d, a, b, M[6], 11);   b = FF(b, c, d, a, M[7], 19);

    a = FF(a, b, c, d, M[8], 3);    d = FF(d, a, b, c, M[9], 7);
    c = FF(c, d, a, b, M[10], 11);  b = FF(b, c, d, a, M[11], 19);

    a = FF(a, b, c, d, M[12], 3);   d = FF(d, a, b, c, M[13], 7);
    c = FF(c, d, a, b, M[14], 11);  b = FF(b, c, d, a, M[15], 19);

   ~// Round 2
    a = GG(a, b, c, d, M[0], 3);    d = GG(d, a, b, c, M[4], 5);
    c = GG(c, d, a, b, M[8], 9);    b = GG(b, c, d, a, M[12], 13);

    a = GG(a, b, c, d, M[1], 3);    d = GG(d, a, b, c, M[5], 5);
    c = GG(c, d, a, b, M[9], 9);    b = GG(b, c, d, a, M[13], 13);

    a = GG(a, b, c, d, M[2], 3);    d = GG(d, a, b, c, M[6], 5);
    c = GG(c, d, a, b, M[10], 9);   b = GG(b, c, d, a, M[14], 13);

    a = GG(a, b, c, d, M[3], 3);    d = GG(d, a, b, c, M[7], 5);
    c = GG(c, d, a, b, M[11], 9);   b = GG(b, c, d, a, M[15], 13);

   ~// Round 3
    a = HH(a, b, c, d, M[0], 3);    d = HH(d, a, b, c, M[8], 9);
    c = HH(c, d, a, b, M[4], 11);   b = HH(b, c, d, a, M[12], 15);

    a = HH(a, b, c, d, M[2], 3);    d = HH(d, a, b, c, M[10], 9);
    c = HH(c, d, a, b, M[6], 11);   b = HH(b, c, d, a, M[14], 15);

    a = HH(a, b, c, d, M[1], 3);    d = HH(d, a, b, c, M[9], 9);
    c = HH(c, d, a, b, M[5], 11);   b = HH(b, c, d, a, M[13], 15);

    a = HH(a, b, c, d, M[3], 3);    d = HH(d, a, b, c, M[11], 9);
    c = HH(c, d, a, b, M[7], 11);   b = HH(b, c, d, a, M[15], 15);

    A = sum(A, a, 32);   B = sum(B, b, 32);
    C = sum(C, c, 32);   D = sum(D, d, 32);

   ~// Хеш-значение
```

```
    Out[0] = A; Out[1] = B; Out[2] = C; Out[3] = D;
}
```

Приложение 2. 10 двухблоковых коллизий с нулевыми первыми 10 байтами для хеш-функции MD5.

№		
1	M	00 00 00 00 00 00 00 00 00 00 e0 5c 2f 3c f5 48 32 1e cc a0 bf 25 b9 bd ed 93 8d 88 c3 c9 f5 e4 55 2d 34 05 06 c6 b3 00 9b f4 b2 83 75 71 fa 1e f3 26 84 73 04 57 ab 23 0e ca 73 02 d6 5b a3 aa 54 4f 48 19 c2 3d b1 f4 12 2b 6e 8d 9f 31 40 ad c6 f4 66 99 fc 02 44 dd 14 09 a0 47 d0 c8 5d af c1 bf b6 6e 51 d7 f5 87 d6 81 32 d8 93 00 e4 dd 0f 59 e5 6b 96 f9 9b e4 13 df 64 ae 90 69 b6 a6
	M'	00 00 00 00 00 00 00 00 00 00 e0 5c 2f 3c f5 48 32 1e cc 20 bf 25 b9 bd ed 93 8d 88 c3 c9 f5 e4 55 2d 34 05 06 c6 b3 00 9b f4 b2 83 75 f1 fa 1e f3 26 84 73 04 57 ab 23 0e ca 73 82 d6 5b a3 aa 54 4f 48 19 c2 3d b1 f4 12 2b 6e 8d 9f 31 40 ad c6 f4 66 19 fc 02 44 dd 14 09 a0 47 d0 c8 5d af c1 bf b6 6e 51 d7 f5 87 d6 81 32 d8 93 80 e3 dd 0f 59 e5 6b 96 f9 9b e4 13 df 64 2e 90 69 b6 a6
	H	178477e15fde4ff267aa55438d539b16

№		
2	M	00 00 00 00 00 00 00 00 00 00 60 55 bf af f4 48 48 3f 82 ce be 34 b9 bc 75 a2 8d 65 ca d8 f5 42 55 ed 33 06 06 3e 73 e1 ef c0 54 8d 19 2b c9 25 ff 22 b2 bf 28 d7 6e 75 a9 7b 86 38 79 55 d8 fa 82 2e 0b b0 c6 90 e8 af 1b 1a fa dc d3 62 54 c2 02 3a 7b c7 48 00 50 0e 14 fd af 15 51 fd cd 6f 4d c7 f3 0f e8 f3 9e aa f2 13 2d 54 5f b8 98 e4 bb dc a5 3d 57 f3 9b b2 9b cd 17 eb e7 3a 19 77
	M'	00 00 00 00 00 00 00 00 00 00 60 55 bf ef f4 48 48 3f 82 4e be 34 b9 bc 75 a2 8d 65 ca d8 f5 42 55 ed 33 06 06 3e 73 e1 ef c0 54 8d 19 ad c9 25 ff 22 b2 bf 28 d7 6e 75 a9 7b 86 b8 79 55 d8 fa 82 2e 0b b0 c6 90 e8 af 1b 1d fa dc d3 62 54 c2 02 3a 7b 47 48 00 50 0e 14 fd af 15 51 fd cd bf 4d c7 f3 0f e8 f3 9e aa f2 13 2d 54 5f 38 98 e4 bb dc a5 3d 57 f3 9b b2 9b cd 17 6b e7 3a 19 77
	H	5dc59708b6193c9ba717f9666833c3fc

№		
3	M	00 00 00 00 00 00 00 00 00 00 a0 4c 0b 13 f7 48 71 3b 9a ca 3f 46 b7 c4 6d b5 8a 70 43 ea f3 4f 51 ad 32 06 04 18 f3 03 c7 b3 4e 95 63 d5 c8 18 17 a6 ce 80 fc 04 0e b0 86 e1 81 50 3b 46 f0 3a 21 a5 02 a8 0a 91 5f 65 5d 44 58 0a cb 70 39 8d 90 15 cb 98 8f 80 37 de 48 39 27 55 51 e7 91 b2 a1 a7 37 51 ae 51 e1 a9 9e 01 d4 99 4c 38 94 dd c9 24 cc c5 ae de 23 52 b4 b9 b0 d4 88 07 f1 a6
	M'	00 00 00 00 00 00 00 00 00 00 a0 4c 0b 13 f7 48 71 3b 9a 4a 3f 46 b7 c4 6d b5 8a 70 43 ea f3 4f 51 ad 32 06 04 18 f3 03 c7 b3 4e 95 63 55 c9 18 f7 a6 ce 80 fc 04 0e b0 86 e1 81 d0 3d 46 f0 3a 21 a5 02 a8 0a 91 5f 65 5d 44 58 0a cb 70 39 8d 90 15 cb 18 8f 80 37 de 48 39 27 55 51 e7 91 b2 a1 a7 37 51 ae 51 e1 a9 9e 01 d4 99 c4 b8 93 dd c9 24 cc c5 ae de 23 52 b4 b9 b0 54 88 07 f1 a6
	H	f5ee1ed452335f22ccfd9f9e91042cdf

№		
4	M	00 00 00 00 00 00 00 00 00 00 a0 ab 83 5d f7 48 f7 41 5e 8f 3e 88 b8 bc 5d 35 8d d4 31 2c 35 b1 d5 ed 34 06 01 4c 34 81 a7 e5 a2 88 86 27 51 d5 dd 16 5e 82 39 5e 3c d6 0a aa 1b 72 2a 26 9e db 8d ba 8e 59 ae 54 92 e5 86 3a 97 82 14 27 af 04 cc 10 8d 24 4f 94 ad 2b 26 79 9a 35 4f 79 42 90 fd 3f 37 77 f8 f6 86 29 ea 5f af a1 4e e0 de 75 cb 48 c8 7b 72 e3 3b e9 e6 37 a8 69 a6 22 bd 9a
	M'	00 00 00 00 00 00 00 00 00 00 a0 ab 83 5d f7 48 f7 41 5e 0f 3e 88 b8 bc 5d 35 8d d4 31 2c 35 b1 d5 ed 34 06 01 4c 34 81 a7 e5 a2 88 86 a7 51 d5 dd 16 5e 82 39 5e 3c d6 0a aa 1b f2 2a 26 9e db 8d ba 8e 59 ae 54 92 e5 86 3a 97 82 14 27 af 04 cc 10 8d a4 4f 94 ad 2b 26 79 9a 35 4f 79 42 90 fd 3f 37 77 f8 f6 86 29 ea 5f af a1 4e 60 de 75 cb 48 c8 7b 72 e3 3b e9 e6 37 a8 e9 a6 22 bd 9a
	H	5117d130ae71fb35afa956332dfaeefd

№		
5	M	00 00 00 00 00 00 00 00 00 00 20 74 67 a6 f5 48 cb c1 6d a5 3e f7 b8 bc 67 a3 8d d9 3c 9b f5 b8 55 ed 32 06 06 0a 74 a3 0f b6 84 87 47 cf 91 d0 db 4c 6f 43 ef 64 f0 8d a4 1d 50 c6 26 df 95 fe ff d1 2e c9 a0 90 aa b3 7d e7 e5 bc f2 3a 4e ab 24 b8 d4 13 4c cc 7b 1b 00 29 eb f5 53 7a 0d d1 5d 1f b7 79 af 36 ce 08 1e 44 a2 d0 51 ec 91 fb c5 4c a2 89 75 b3 a3 84 ac 97 7f f2 7e 50 d4 56
	M'	00 00 00 00 00 00 00 00 00 00 20 74 67 a6 f5 48 cb c1 6d 25 3e f7 b8 bc 67 a3 8d d9 3c 9b f5 b8 55 ed 32 06 06 0a 74 a3 0f b6 84 87 47 4f 92 d0 db 4c 6f 43 ef 64 f0 8d a4 1d 50 46 26 df 95 fe ff d1 2e c9 a0 90 aa b3 7d e7 e5 bc f2 3a 4e ab 24 b8 d4 93 4c cc 7b 1b 00 29 eb f5 53 7a 0d d1 5d 1f b7 79 af 36 ce 08 1e 44 a2 d0 51 6c 91 fb c5 4c a2 89 75 b3 a3 84 ac 97 7f 72 7e 50 d4 56
	H	22664780a9766ceb57065eba36af06b

6	M	00 00 00 00 00 00 00 00 00 00 a0 2b a3 02 f7 48 f1 3f ce aa 42 88 b7 c4 73 35 8c 8c 47 2c f4 e9 55 6d 34 06 06 22 73 a3 ef c0 aa 90 48 7b b9 a6 bd 2c 2d 9f fb a6 aa 1e de 2b 23 aa c4 5f e4 7b e2 15 a8 f3 08 ae a9 2c b3 72 d5 1b 6a 49 de 9e 14 77 19 16 f6 a5 ff 1b e8 fc a4 76 d1 1b dd de fd 9e 3b 87 4f 74 cf 88 62 00 7a 74 a0 40 2d be 12 4d 59 7b 9a 97 a3 df f5 7a 7a 30 21 9b 0a 64
	M'	00 00 00 00 00 00 00 00 00 00 a0 2b a3 02 f7 48 f1 3f ce 2a 42 88 b7 c4 73 35 8c 8c 47 2c f4 e9 55 6d 34 06 06 22 73 a3 ef c0 aa 90 48 fb b9 a6 bd 2c 2d 9f fb a6 aa 1e de 2b 23 2a c4 5f e4 7b e2 15 a8 f3 08 ae a9 2c b3 72 d5 1b 6a 49 de 9e 14 77 19 96 f6 a5 ff 1b e8 fc a4 76 d1 1b dd de fd 9e 3b 87 4f 74 cf 88 62 00 7a 74 a0 c0 2c be 12 4d 59 7b 9a 97 a3 df f5 7a 7a b0 21 9b 0a 64
	H	27f687621ee95ffdf587d00dfb637ed2

7	M	00 00 00 00 00 00 00 00 00 00 e0 55 2f 0f f5 48 4e 3e 4c 9f be 33 b9 bd ed 9f 8d c8 c2 d7 35 25 55 6d 34 05 05 4e d4 00 0b da 70 7b 57 f9 72 d5 00 95 ab 08 df c9 49 b8 9e e5 44 01 10 c3 23 65 a4 15 32 17 ee 5c 48 8c c0 f3 39 a9 8b ae 19 d5 53 93 e9 a6 77 ce 81 eb 0c ad d9 44 53 1d fd fd ad 87 b3 80 de ce 22 8e 82 26 b5 93 87 10 0a 04 b8 00 29 91 5f a0 03 a9 ab 0a 6b 56 a1 70 45 fe
	M'	00 00 00 00 00 00 00 00 00 00 e0 55 2f 0f f5 48 4e 3e 4c 1f be 33 b9 bd ed 9f 8d c8 c2 d7 35 25 55 6d 34 05 05 4e d4 00 0b da 70 7b 57 79 73 d5 00 95 ab 08 df c9 49 b8 9e e5 44 81 10 c3 23 65 a4 15 32 17 ee 5c 48 8c c0 f3 39 a9 8b ae 19 d5 53 93 e9 26 77 ce 81 eb 0c ad d9 44 53 1d fd fd ad 87 b3 80 de ce 22 8e 82 26 b5 93 87 90 09 04 b8 00 29 91 5f a0 03 a9 ab 0a 6b d6 a1 70 45 fe
	H	edf90bfab0f3496117e25a6048c1cdd5

8	M	00 00 00 00 00 00 00 00 00 00 60 db 73 4a f7 48 2c 21 26 c7 c2 28 b8 bc 69 96 8c b8 bd cc 34 16 d5 2c 35 06 05 70 14 e2 bb 9d 5c 78 ab 57 02 14 c4 18 ab d7 de 0d e0 2d d2 a9 3c 87 43 e8 5d 72 24 21 ab 36 5a 9c 41 80 c9 79 29 3a 0d 42 ad 87 92 17 6e 98 2e f1 48 de 12 6d 3c 75 51 7e c4 ae 4d 67 36 8f f6 74 9d 46 12 6a 5d aa 80 04 01 1c af f8 6d ab c0 e6 0b 42 b0 c3 92 f8 60 a9 84 82
	M'	00 00 00 00 00 00 00 00 00 00 60 db 73 4a f7 48 2c 21 26 47 c2 28 b8 bc 69 96 8c b8 bd cc 34 16 d5 2c 35 06 05 70 14 e2 bb 9d 5c 78 ab d7 02 14 c4 18 ab d7 de 0d e0 2d d2 a9 3c 07 43 e8 5d 72 24 21 ab 36 5a 9c 41 80 c9 79 29 3a 0d 42 ad 87 92 17 6e 18 2e f1 48 de 12 6d 3c 75 51 7e c4 ae 4d 67 36 8f f6 74 9d 46 12 6a 5d aa 80 84 00 1c af f8 6d ab c0 e6 0b 42 b0 c3 92 78 60 a9 84 82
	H	de58a6db2d75a5d05c2faad167cf3a96

9	M	00 00 00 00 00 00 00 00 00 00 a0 aa 97 fd f6 48 f5 41 54 af 43 8a b8 c4 73 39 8d cd 49 2e f5 2c d5 ec 33 06 06 32 74 81 ff b9 7a 83 86 f1 71 17 0d 6d 51 42 ed d3 7e 68 d6 bb 7f dd 80 ab cb b1 32 cc 2d d0 9c 10 4a 32 66 29 ea 3e 67 bc 35 2c b1 52 04 46 24 ca 37 4d 2c 65 2a 75 53 e8 71 a0 fd 67 73 54 7d 7a 19 cd 36 14 46 c5 4d b4 36 33 d8 40 df 5a 35 b3 03 65 40 9a 5a 8d 27 d6 ab ce
	M'	00 00 00 00 00 00 00 00 00 00 a0 aa 97 fd f6 48 f5 41 54 2f 43 8a b8 c4 73 39 8d cd 49 2e f5 2c d5 ec 33 06 06 32 74 81 ff b9 7a 83 86 71 72 17 0d 6d 51 42 ed d3 7e 68 d6 bb 7f 5d 80 ab cb b1 32 cc 2d d0 9c 10 4a 32 66 29 ea 3e 67 bc 35 2c b1 52 04 c6 24 ca 37 4d 2c 65 2a 75 53 e8 71 a0 fd 67 73 54 7d 7a 19 cd 36 14 46 c5 4d 34 36 33 d8 40 df 5a 35 b3 03 65 40 9a 5a 0d 27 d6 ab ce
	H	b39188a68cfdd8b1460cac7b7ae23a86

10	M	00 00 00 00 00 00 00 00 00 00 a0 f4 3b 6f f7 48 cb bc 83 d0 3e f6 b7 bc f5 a4 8c 64 49 9a f4 c1 d5 ad 33 05 06 be b4 01 13 b7 aa 75 f4 a4 71 93 f3 0c d4 7a 38 18 0e 19 e7 23 4a c3 43 6a 53 04 3f 9f c0 03 d2 08 82 0e 3b 68 c0 23 f5 e1 d2 0b 5c fc ae 47 8d 9e 47 4e 44 7d 33 57 4e 7b 39 9e 9d 7f bb 7f 36 d1 19 aa 32 40 eb 64 7f 60 af fc d6 74 f9 d9 83 de 0b d8 df 36 11 18 42 a1 95 ab
	M'	00 00 00 00 00 00 00 00 00 00 a0 f4 3b 6f f7 48 cb bc 83 50 3e f6 b7 bc f5 a4 8c 64 49 9a f4 c1 d5 ad 33 05 06 be b4 01 13 b7 aa 75 f4 24 72 93 f3 0c d4 7a 38 18 0e 19 e7 23 4a 43 43 6a 53 04 3f 9f c0 03 d2 08 82 0e 3b 68 c0 23 f5 e1 d2 0b 5c fc ae c7 8d 9e 47 4e 44 7d 33 57 4e 7b 39 9e 9d 7f bb 7f 36 d1 19 aa 32 40 eb 64 7f e0 ae fc d6 74 f9 d9 83 de 0b d8 df 36 11 98 42 a1 95 ab
	H	4699e1b0af9fb2896241ddd8eeea60d6

СПИСОК ЛИТЕРАТУРЫ

1. *Rivest R.L.* The MD4 message digest algorithm // Lecture Notes in Computer Science. Vol. 537. Heidelberg: Springer, 1991. 303–311.

2. *Rivest R.L.* The MD5 message-digest algorithm. Request for Comments (RFC): 1321 (http://www.rfc-base.org/txt/rfc-1321.txt).
3. *Wang X., Lai X., Feng D., Chen H., Yu X.* Cryptanalysis of the hash functions MD4 and RIPEMD // Lecture Notes in Computer Science. Vol. 3494. Heidelberg: Springer, 2005. 1–18.
4. *Wang X., Yu H.* How to Break MD5 and Other Hash Functions // Lecture Notes in Computer Science. Vol. 3494. Heidelberg: Springer, 2005. 19–35.
5. *Klima V.* Finding MD5 collisions on a notebook PC using multimessage modifications // Cryptology ePrint Archive. Report 2005/102. 2005 (http://eprint.iacr.org/2005/102).
6. *De Canniere C., Rechberger C.* Finding SHA-1 characteristics: general results and applications // Lecture Notes in Computer Science. Vol. 4284. Heidelberg: Springer, 2006. 1–20.
7. *De Canniere C., Mendel F., Rechberger C.* Collisions for 70-step SHA-1: on the full cost of collision search // Lecture Notes in Computer Science. Vol. 4876. Heidelberg: Springer, 2007. 56–73.
8. *Гречников Е.А., Адинец А.В.* Построение коллизии для 75-шаговой версии хеш-функции SHA-1 с использованием ГПУ кластеров // Вычислительные методы и программирование: новые вычислительные технологии. 2012. **13**, вып. 2. 82–89.
9. *Карпунин Г.А., Ермолаева Е.З.* Оценки сложности поиска коллизий для хеш-функции RIPEMD // Прикладная дискретная математика. Приложение. 2012. № 5. 43–44.
10. *Stevens M.* Single-block collision attack on MD5. Cryptology ePrint Archive. Report 2012/040. 2012 (http://eprint.iacr.org/2012/040).
11. *Mironov I., Zhang L.* Applications of SAT Solvers to Cryptanalysis of Hash Functions // Lecture Notes in Computer Science. Vol. 4121. Heidelberg: Springer, 2006. 102–115.
12. *Merkle R.A.* Certified digital signature // Lecture Notes in Computer Science. Vol. 435. 218–238. Heidelberg: Springer, 1990. 218–238.
13. *Damgard I.A.* Design Principle for Hash Functions // Lecture Notes in Computer Science. Vol. 435. Heidelberg: Springer, 1990. 416–427.
14. *Biham E., Shamir A.* Differential cryptanalysis of DES-like cryptosystems // Proc. of the 10th Annual International Cryptology Conference on Advances in Cryptology. London: Springer, 1990. 2–21.
15. *Cook S.A.* The complexity of theorem-proving procedures // Proc. of the Third Annual ACM Symposium on Theory of Computing. New York: ACM Press, 1971. 151–159.
16. *Цейтин Г.С.* О сложности вывода в исчислении высказываний // Записки научных семинаров ЛОМИ АН СССР. 1968. **8**. 234–259.
17. *Biere A., Heule V., van Maaren H., Walsh T.* Handbook of satisfiability. Amsterdam: IOS Press, 2009.
18. *Soos M., Nohl K., Castelluccia C.* Extending SAT solvers to cryptographic problems // Lecture Notes in Computer Science. Vol. 5584. Heidelberg: Springer, 2009. 244–257.
19. *Semenov A., Zaikin O., Bespalov D., Posypkin M.* Parallel logical cryptanalysis of the generator A5/1 in BNB-Grid system // Lecture Notes in Computer Science. Vol. 6873. Heidelberg: Springer, 2011. 473–483.
20. *Jovanović D., Janičić P.* Logical analysis of hash functions // Lecture Notes in Artificial Intelligence. Vol. 3717. Heidelberg: Springer, 2005. 200–215.
21. *Отпущенников И.В., Семенов А.А.* Технология трансляции комбинаторных проблем в булевы уравнения // Прикладная дискретная математика. 2011. № 1. 96–115.
22. *Otpuschennikov I., Semenov A., Kochemazov S.* Transalg: a tool for translating procedural descriptions of discrete functions to SAT (tool paper). (arXiv:1405.1544 [cs.AI]; Cornell Univ. Library), 2014 (http://arxiv.org/abs/1405.1544).
23. *King J.C.* Symbolic execution and program testing // Communications of the ACM. 1976. Vol. 19, N 7. 385–394.
24. *Керниган Б.У., Ритчи Д.М.* Язык программирования C. М.: Вильямс, 2006.
25. *Ахо А., Сети Р., Ульман Д.* Компиляторы. Принципы, технологии, инструменты. Москва: Вильямс, 2001.
26. *Menezes A. J., van Oorshot P. C., Vanstone S. A.* Handbook of applied cryptography. Boca Raton: CRC Press, 1996.
27. *Marques-Silva J.P., Sakallah K.A.* GRASP: a search algorithm for propositional satisfiability // IEEE Transactions on Computers. 1999. **48**, N 5. 506–521.
28. *Davis M., Logemann G., Loveland D.* A machine program for theorem proving // Communication of the ACM. 1962. **5**, N 7. 394–397.
29. *Haken A.* The intractability of resolution // Theor. Comp. Sci. 1985. **39**. 297–308.
30. *Beame P., Kautz H., Sabharwal A.* Towards understanding and harnessing the potential of clause learning // Journal of Artificial Intelligence Research. 2004. **22**. 319–351.
31. *Cook S.A., Mitchel G.* Finding hard instances of the satisfiability problem: a survey // DIMACS Series in Discrete Mathematics and Theoretical Computer Science. Vol. 35. Providence: AMS Press, 1997. 1–17.
32. *Massacci F., Marraro L.* Logical cryptanalysis as a SAT problem // Journal of Automated Reasoning. 2000. **24**. N 1–2. 165–203.
33. *Семенов А.А., Заикин О.С., Беспалов Д.В., Ушаков А.А.* SAT-подход в криптоанализе некоторых систем поточного шифрования // Вычислительные технологии. 2008. **13**, № 6. 134–150.
34. *Заикин О.С., Семенов А.А.* Технология крупноблочного параллелизма в SAT-задачах // Проблемы управле-

ния. 2008. № 1. 43–50.

35. *Семенов А.А., Заикин О.С., Беспалов Д.В., Буров П.С., Хмельнов А.Е.* Решение задач обращения дискретных функций на многопроцессорных вычислительных системах // Тр. IV Международной конференции “Параллельные вычисления и задачи управления” (PACO’2008). М.: ИПУ РАН им. В.А. Трапезникова, 2008. 152–176.
36. *Evtushenko Y., Posypkin M., Sigal I.* A framework for parallel large-scale global optimization // Computer Science — Research and Development. 2009. **23**, N 3–4. 211–215.
37. *Посыпкин М.А., Заикин О.С., Беспалов Д.В., Семенов А.А.* Решение задач криптоанализа поточных шифров в распределенных вычислительных средах // Тр. Института системного анализа Российской академии наук. 2009. **46**. 119–137.
38. *Заикин О.С., Семенов А.А.* Применение метода Монте-Карло к прогнозированию времени параллельного решения проблемы булевой выполнимости // Вычислительные методы и программирование: новые вычислительные технологии. 2014. **15**, вып. 1. 22–35.
39. *Заикин О.С., Посыпкин М.А., Семенов А.А., Храпов Н.П.* Опыт организации добровольных вычислений на примере проектов OPTIMA@home и SAT@home // Вестник Нижегородского университета им. Н.И. Лобачевского. 2012. № 5. 340–347.
40. *Заикин О.С., Семенов А.А., Посыпкин М.А.* Процедуры построения декомпозиционных множеств для распределенного решения SAT-задач в проекте добровольных вычислений SAT@home // Управление большими системами. Вып. 43. М.: Ин-т проблем управления РАН, 2013. 138–156.
41. *Заикин О.С.* Реализация процедур прогнозирования трудоемкости параллельного решения SAT-задач // Вестник Уфимского государственного авиационного технического университета. 2010. **14**, № 4. 210–220.
42. *Заикин О.С., Отпущенников И.В., Семенов А.А.* Параллельные алгоритмы решения проблемы выполнимости в применении к оптимизационным задачам с булевыми ограничениями // Вычислительные методы и программирование: новые вычислительные технологии. 2011. **12**. 205–212.
43. *Hyvärinen A.E.J., Junttila T.A., Niemelä I.* A distribution method for solving SAT in grids // Lecture Notes in Computer Science. Vol. 4121. Heidelberg: Springer, 2006. 430–435.
44. *Hyvärinen A.E.J.* Grid based propositional satisfiability solving. Aalto: Aalto University, 2011.
45. *De D., Kumarasubramanian A., Venkatesan R.* Inversion attacks on secure hash functions using SAT solvers // Lecture Notes in Computer Science. Vol. 4501. Heidelberg: Springer, 377–382.

Поступила в редакцию
18.01.2015

Problems of Search for Collisions of Cryptographic Hash Functions of the MD Family as Variants of Boolean Satisfiability Problem

I. A. Bogachkova[1], O. S. Zaikin[2], S. E. Kochemazov[3], I. V. Otpushchennikov[4], A. A. Semenov[5], and O. O. Khamisov[6]

[1] *Irkutsk State University, Institute of Mathematics, Economics and Informatics; bul’var Gagarina 20, Irkutsk, 664003, Russia; Student, e-mail: the42dimension@gmail.com*

[2] *Matrosov Institute for System Dynamics and Control Theory, Siberian Branch of Russian Academy of Sciences; ulitsa Lermontova 134, Irkutsk, 664033, Russia; Ph.D., Scientist, e-mail: zaikin.icc@gmail.com*

[3] *Matrosov Institute for System Dynamics and Control Theory, Siberian Branch of Russian Academy of Sciences; ulitsa Lermontova 134, Irkutsk, 664033, Russia; Programmer, e-mail: veinamond@gmail.com*

[4] *Matrosov Institute for System Dynamics and Control Theory, Siberian Branch of Russian Academy of Sciences; ulitsa Lermontova 134, Irkutsk, 664033, Russia; Ph.D., Scientist, e-mail: otilya@yandex.ru*

[5] *Matrosov Institute for System Dynamics and Control Theory, Siberian Branch of Russian Academy of Sciences; ulitsa Lermontova 134, Irkutsk, 664033, Russia; Ph.D., Associate Professor, Head of Laboratory, e-mail: biclop.rambler@yandex.ru*

[6] *Irkutsk State University, Institute of Mathematics, Economics and Informatics; bul’var Gagarina 20, Irkutsk, 664003, Russia; Student, e-mail: cygx151@gmail.com*

Received January 18, 2015

Abstract: An implementation of the differential attacks on cryptographic hash functions MD4 (Message Digest 4) and MD5 (Message Digest 5) by reducing the problems of search for collisions of these hash functions to the Boolean satisfiability problem (SAT) is considered. The novelty of the results obtained consists in a more compact (compared to already known) SAT encodings for the algorithms considered and in the use of modern parallel and distributed SAT solvers in applications to the formulated SAT problems. Searching for single block collisions of MD4 in this approach turned out to be very simple. In addition, several dozens of double block collisions of MD5 are found. In the process of the corresponding numerical experiments, a certain class of messages that produce the collisions is found: in particular, a set of pairs of such messages with first 10 zero bytes is constructed.

Keywords: cryptographic hash functions, collisions, differential attacks, Boolean satisfiability problem, SAT, CDCL (Conflict-Driven Clause Learning), parallel SAT solvers.

References

1. R. L. Rivest, "The MD4 Message Digest Algorithm," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 1991), Vol. 537, pp. 303–311.

2. R. L. Rivest, "The MD5 Message-Digest Algorithm," Request for Comments (RFC): 1321. http://www.rfc-base.org/txt/rfc-1321.txt. Cited January 15, 2015.

3. X. Wang, X. Lai, D. Feng, et al., "Cryptanalysis of the Hash Functions MD4 and RIPEMD," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2005), Vol. 3494, pp. 1–18.

4. X. Wang and H. Yu, "How to Break MD5 and Other Hash Functions," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2005), Vol. 3494, pp. 19–35.

5. V. Klima, "Finding MD5 Collisions on a Notebook PC Using Multimessage Modifications," Cryptology ePrint Archive. Report 2005/102. 2005. http://eprint.iacr.org/2005/102. Cited January 15, 2015.

6. C. de Canniere and C. Rechberger, "Finding SHA-1 Characteristics: General Results and Applications," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2006), Vol. 4284, pp. 1–20.

7. C. de Canniere, F. Mendel, and C. Rechberger, "Collisions for 70-Step SHA-1: On the Full Cost of Collision Search," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2007), Vol. 4876, pp. 56–73.

8. E. A. Grechnikov and A. V. Adinets, "Finding a Collision for the 75-Round SHA-1 Hash Function Using Clusters of GPUs," Vychisl. Metody Programm. **13** (2), 82–89 (2012).

9. G. A. Karpunin and E. Z. Ermolaeva, "Estimates of Collision Search Complexity for the Hash Function RIPEMD," Prikl. Diskr. Mat. Suppl., No. 5, 43–44 (2012).

10. M. Stevens, "Single-Block Collision Attack on MD5," Cryptology ePrint Archive. Report 2012/040. 2012. http://eprint.iacr.org/2012/040. Cited January 15, 2015.

11. I. Mironov and L. Zhang, "Applications of SAT Solvers to Cryptanalysis of Hash Functions," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2006), Vol. 4121, pp. 102–115.

12. R. A. Merkle, "Certified Digital Signature," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 1990), Vol. 435, pp. 218–238.

13. I. A. Damgård, "A Design Principle for Hash Functions," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 1990), Vol. 435, pp. 416–427.

14. E. Biham and A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems," in *Proc. 10th Annu. Int. Cryptology Conf. on Advances in Cryptology, Santa Barbara, USA, August 11–15, 1990* (Springer, London, 1991), pp. 2–21.

15. S. A. Cook, "The Complexity of Theorem-Proving Procedures," in *Proc. 3rd Annu. ACM Symp. on Theory of Computing, Shaker Heights, USA, May 3–5, 1971* (ACM Press, New York, 1971), pp. 151–158.

16. G. S. Tseitin, "On the Complexity of Proof in Prepositional Calculus," Zap. Nauch. Sem. LOMI **8**, 234–259 (1968).

17. A. Biere, V. Heule, H. van Maaren, and T. Walsh, *Handbook of Satisfiability* (IOS Press, Amsterdam, 2009).

18. M. Soos, K. Nohl, and C. Castelluccia, "Extending SAT Solvers to Cryptographic Problems," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2009), Vol. 5584, pp. 244–257.

19. A. Semenov, O. Zaikin, D. Bespalov, and M. Posypkin, "Parallel Logical Cryptanalysis of the Generator A5/1 in BNB-Grid System," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2011), Vol. 6873, pp. 473–483.

20. D. Jovanović and P. Janičić, "Logical Analysis of Hash Functions," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2005), Vol. 3717, pp. 200–215.

21. I. V. Otpushchennikov and A. A. Semenov, "Technology for Translating Combinatorial Problems into Boolean Equations," Prikl. Diskr. Mat., No. 1, 96–115 (2011).

22. I. Otpushchennikov, A. Semenov, S. Kochemazov, *Transalg: A Tool for Translating Procedural Descriptions of Discrete Functions to SAT (tool paper)*, arXiv preprint: 1405.1544 [cs.AI] (Cornell Univ. Library, Ithaca, 2014). http://arxiv.org/abs/1405.1544. Cited January 15, 2015.

23. J. C. King, "Symbolic Execution and Program Testing," Commun. ACM **19** (7), 385–394 (1976).

24. B. W. Kernighan and D. M. Ritchie, *The C Programming Language* (Prentice Hall, Englewood Cliffs, 1988; Vil'yams, Moscow, 2006).

25. A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools* (Addison-Wesley, Boston, 1986; Vil'yams, Moscow, 2001).

26. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography* (CRC Press, Boca Raton, 1996).

27. J. P. Marques-Silva and K. A. Sakallah, "GRASP: A Search Algorithm for Propositional Satisfiability," IEEE Trans. Comput. **48** (5), 506–521 (1999).

28. M. Davis, G. Logemann, and D. Loveland, "A Machine Program for Theorem Proving," Commun. ACM **5** (7), 394–397 (1962).

29. A. Haken, "The Intractability of Resolution," Theor. Comp. Sci. **39**, 297–308 (1985).

30. P. Beame, H. Kautz, and A. Sabharwal, "Towards Understanding and Harnessing the Potential of Clause Learning," J. Artif. Intell. Res. **22**, 319–351 (2004).

31. S. A. Cook and D. G. Mitchell, "Finding Hard Instances of the Satisfiability Problem: A Survey," in *Satisfiability Problem: Theory and Applications. DIMACS Series in Discrete Mathematics and Theoretical Computer Science* (AMS Press, Providence, 1997), Vol. 35, pp. 1–17.

32. F. Massacci and L. Marraro, "Logical Cryptanalysis as a SAT Problem," J. Autom. Reason. **24** (1–2), 165–203 (2000).

33. A. A. Semenov, O. S. Zaikin, D. V. Bespalov, and A. A. Ushakov, "SAT-approach for Cryptoanalysis of Some Stream Ciphering Systems," Vychisl. Tekhnol. **13** (6), 134–150 (2008).

34. O. S. Zaikin and A. A. Semenov, "Large-Block Parallelism Technology in SAT Problems," Probl. Upr., No. 1, 43–50 (2008).

35. A. A. Semenov, O. S. Zaikin, D. V. Bespalov, et al., "Inversion of Discrete Functions Using Multiprocessor Computers," in *Proc. 4th Int. Conf. on Parallel Computations and Control Problems, Moscow, October 27–29, 2008* (Trapeznikov Inst. Control Sci., Moscow, 2008), pp. 152–176.

36. Yu. Evtushenko, M. Posypkin, and I. Sigal, "A Framework for Parallel Large-Scale Global Optimization," Comput. Sci. Res. Dev. **23** (3–4), 211–215 (2009).

37. M. A. Posypkin, O. S. Zaikin, D. V. Bespalov, and A. A. Semenov, "Solving the Cryptanalysis Problems of Stream Ciphers on Distributed Computer Systems," Tr. Inst. Systems Anal., Ross. Akad. Nauk **46**, 119–137 (2009).

38. A. A. Semenov and O. S. Zaikin, "Application of the Monte Carlo Method for Estimating the Total Time of Solving the SAT Problem in Parallel," Vychisl. Metody Programm. **15** (1), 22–35 (2014).

39. O. S. Zaikin, M. A. Posypkin, A. A. Semenov, and N. P. Khrapov, "Using Volunteer Computation by the Example of the OPTIMA@home and SAT@home Projects," Vestn. Univ. Nizhni Novgorod, No. 5, 340–347 (2012).

40. O. S. Zaikin, A. A. Semenov, and M. A. Posypkin, "Procedures of Constructing Decomposition Sets for the Distributed Solution of SAT problems in the SAT@home Project," in *Large System Control* (Inst. Problem Upravl., Moscow, 2013), Issue 43, pp. 138–156.

41. O. S. Zaikin, "Implementation of Prediction Procedures to Evaluate the Complexity of Parallel Solution of SAT Problems," Vestn. Ufimsk. Gos. Aviats. Tekhn. Univ. **14** (4), 210–220 (2010).

42. O. S. Zaikin, I. V. Otpuschennikov, and A. A. Semenov, "Parallel Algorithms for Solving SAT-Problems in Application to Optimization Problems with Boolean Constraints," Vychisl. Metody Programm. **12**, 205–212 (2011).

43. A. E. J. Hyvärinen, T. A. Junttila, and I. Niemelä, "A Distribution Method for Solving SAT in Grids," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2006), Vol. 4121, pp. 430–435.

44. A. E. J. Hyvärinen, *Grid Based Propositional Satisfiability Solving*, Ph.D. Thesis (Aalto Univ., Aalto, 2011).

45. D. De, A. Kumarasubramanian, and R. Venkatesan, "Inversion Attacks on Secure Hash Functions Using SAT Solvers," in *Lecture Notes in Computer Science* (Springer, Heidelberg, 2007), Vol. 4501, pp. 377–382.