УДК 004.8 DOI 10.17223/20710410/71/7

НАХОЖДЕНИЕ ПРООБРАЗОВ НЕПОЛНОРАУНДОВОЙ ФУНКЦИИ СЖАТИЯ КРИПТОГРАФИЧЕСКОЙ ХЕШ-ФУНКЦИИ SKEIN-512-256 ПРИ ПОМОЩИ SAT-РЕШАТЕЛЯ[1]

О. С. Заикин

Математический центр НГУ, г. Новосибирск, Россия
ИДСТУ СО РАН, г. Иркутск, Россия

E-mail: oleg.zaikin@icc.ru

Рассматривается криптографическая хеш-функция Skein-512-256, вышедшая в 2010 г. в финал конкурса NIST на новую криптографическую хеш-функцию. Функция сжатия Skein-512-256 обрабатывает 512-битовый блок сообщения в течение 72 раундов, каждый раунд состоит из 12 операций. Предлагается практическая алгебраическая атака нахождения прообраза на неполнораундовые версии функции сжатия Skein-512-256. Соответствующие вычислительные задачи сводятся к экземплярам проблемы булевой выполнимости (SAT). С помощью последовательного SAT-решателя были найдены прообразы функции сжатия Skein-512-256, состоящей из первого раунда и семи первых операций второго раунда. Задействование параллельного SAT-решателя позволило увеличить число операций второго раунда до восьми. Ранее в литературе была предложена практическая атака нахождения прообраза максимум на один раунд функции сжатия Skein-512-256.

Ключевые слова: *криптографическая хеш-функция, Skein, атака нахождения прообраза, алгебраический криптоанализ, SAT.*

PREIMAGE ATTACK ON ROUND-REDUCED SKEIN512-256 COMPRESSION FUNCTION USING SAT SOLVER

O. S. Zaikin

NSU Mathematical Center, Novosibirsk, Russia
ISDCT SB RAS, Irkutsk, Russia

Consider the cryptographic hash function Skein-512-256, which became a finalist of the NIST hash function competition in 2010. Skein-512-256 compression function processes a 512-bit block message block in 72 rounds, 12 operations each. A practical algebraic preimage attack on round-reduced versions of the Skein-512-256 compression function is proposed. The corresponding computational problems are reduced to instances of the Boolean satisfiability problem (SAT). Using sequential SAT solvers, preimages of the first round and the first 7 operations of the second round of the compression function have been found. By applying a parallel SAT solver, the number of the second round's operations has been increased to 8. Earlier, a practical preimage attack on at most 1-round Skein-512-256 compression function was published.

Keywords: *cryptographic hash function, Skein, preimage attack, algebraic cryptanalysis, SAT.*

[1]Работа выполнена при поддержке Математического центра в Академгородке, соглашение с Министерством науки и высшего образования Российской Федерации № 075-15-2025-349.

Введение

Криптографические хеш-фунции широко используются в современном цифровом мире для хеширование паролей, проверки целостности данных, формирования электронных цифровых подписей и эмиссии криптовалюты. Безопасные криптографические хеш-функции должны обладать рядом свойств, среди которых стойкость к нахождению коллизий, прообразов и вторых прообразов [1].

В 2007 г. Национальный институт стандартов и технологий США объявил конкурс на новую криптографическую хеш-функцию. Основной целью конкурса была замена действующего на тот момент стандарта — семейства криптографических хеш-функций SHA-2. В 2012 г. из пяти финалистов был выбран Keccak, и именно он стал новым стандартом под названием SHA-3. С тех пор криптографическая стойкость Keccak была тщательно проанализирована с помощью различных подходов. При этом стойкость остальных финалистов остаётся относительно слабо изученной.

Одним из финалистов конкурса стало семейство Skein [2]. Основной криптографической хеш-функцией семейства является Skein-512-256, в которой функция сжатия базируется на 72-раундовом блочном шифре Threefish, работающем на 512-битовых блоках сообщения. В каждом раунде Threefish к 64-битовым частям внутреннего состояния применяются следующие операции: исключающее или, сложение по модулю 2^{64} и циклический сдвиг. Результатом Skein-512-256 является хеш длиной 256 бит.

Нахождение прообраза первого раунда функции сжатия Skein-512-256 является простой вычислительной задачей — соответствующая практическая алгебраическая атака была представлена в [3]. Под алгебраической атакой понимается атака путём решения систем алгебраических уравнений [4]. Конкретнее, в [3] использован SAT-подход, который является одним из наиболее эффективных на практике способов реализации алгебраических атак.

SAT формулируется следующим образом: по данной пропозициональной булевой формуле ответить на вопрос, существует ли набор значений переменных этой формулы, при которых она истинна [5]. Такой набор называется выполняющим. Программы для решения SAT называются SAT-решателями. При этом булева формула обычно рассматривается в конъюнктивной нормальной форме (КНФ), т. е. в виде конъюнкции дизъюнктов. Дизъюнкт — это дизъюнкция литералов, а литерал — это булева переменная либо её отрицание. Если SAT-решатель может доказать выполнимость КНФ, то, как правило, он выдаёт также и соответствующий выполняющий набор.

В [3] для решения экземпляров SAT использованы SAT-решатели, основанные на полном алгоритме Conflict-Driven Clause Learning (CDCL) [6]. По-видимому, впервые SAT-решатели были применены для анализа стойкости криптографических хеш-функций в работе [7], с тех пор такой подход используется довольно широко.

Отметим, что найти прообраз первых двух раундов функции сжатия Skein-512-256 на текущий момент очень сложно. Каждый раунд этой функции состоит из двенадцати основных операций. В настоящей работе предлагается новая алгебраическая атака нахождения прообраза на неполнораундовые версии функции сжатия Skein-512-256. Конкретнее, проанализированы версии функции сжатия, в которых первый раунд присутствует полностью, во втором раунде оставлены только несколько первых операций, а остальные раунды отброшены. Как и в [3], для этих целей используется SAT-подход. С помощью последовательного CDCL-решателя Kissat найдены прообразы ослабленной функции сжатия Skein-512-256, состоящей из первого раунда и семи операций второго раунда. Задействован метод Cube-and-Conquer, согласно которому SAT-задача разбивается на более простые подзадачи, каждая из которых решается с помощью

CDCL-решателя [8]. Это позволяет увеличить число операций во втором раунде до восьми.

При применении Cube-and-Conquer к описанным задачам обнаружено, что многие подзадачи оказываются очень простыми. Предлагается подход к предварительному препроцессингу КНФ, в рамках которого итеративно запускается Cube-and-Conquer, с помощью ограниченного CDCL решаются только простые подзадачи, а информация о них добавляется в КНФ. Данный подход позволяет найти некоторые прообразы быстрее, а также отыскать прообразы для ряда случаев, которые ранее решению не поддавались.

Работа имеет следующую структуру: в п. 1 описана криптографическая хеш-функция Skein-512-256; п. 2 посвящён SAT-кодировке функции сжатия Skein-512-256; в п. 3 стойкость неполнораундовых версий функции сжатия Skein-512-256 к нахождению прообразов исследуется при помощи CDCL-решателя Kissat и метода Cube-and-Conquer. Кроме того, описан новый подход к препроцессингу КНФ с помощью итеративного применения метода Cube-and-Conquer и приведены соответствующие результаты вычислительных экспериментов.

1. Криптографическая хеш-функция Skein-512-256

Семейство криптографических хеш-функций Skein разработано группой авторов во главе с Брюсом Шнайером [2]. Здесь кратко опишем основную криптографическую хеш-функцию данного семейства, Skein-512-256, которая оперирует с 512-битовыми блоками сообщения и формирует 256-битовый хеш. Отметим, что Skein-512-256 была предложена в качестве замены SHA-256 из семейства SHA-2.

В Skein-512-256 на 512-битовых блоках сообщения итеративно вызывается функция сжатия, которая формирует 512-битовый выход. При этом гораздо более распространённой является ситуация, в которой длина выхода функции сжатия в несколько раз меньше, чем длина входа. Например, в MD5, SHA-1 и SHA-256 длина выхода в 4, 3,2 и 2 раза меньше длины входа соответственно. Данная особенность функции сжатия Skein-512-256 связана с тем, что она базируется на блочном шифре Threefish, который оперирует с 512-битовыми блоками.

Введём следующие термины:

— секретный ключ K длиной 512 бит;
— параметр T длиной 128 бит;
— открытый текст P длиной 512 бит;
— внутреннее состояние S длиной 512 бит;
— шифртекст C длиной 512 бит.

В Threefish выполняется 72-раундовая функция шифрования $C = E(K, T, P)$. Перед первым раундом формируются раундовые ключи, зависящие от секретного ключа K. После этого формируется внутреннее состояние S — восемь 64-битовых слов. Инициализация S осуществляется путём смешивания значений T, P и раундовых ключей. Затем в каждом из 72 раундов модифицируются все восемь слов S. Раз в четыре раунда внутреннее состояние дополнительно смешивается с раундовыми ключами и T.

Опишем раунд более подробно. Сначала формируются четыре пары 64-битовых слов внутреннего состояния, а затем каждая пара обновляется с помощью вызова нелинейной функции MIX. Эта функция получает на вход два 64-битовых слова (x_0, x_1) и вычисляет два 64-битовых слова (y_0, y_1) следующим образом:

$$y_0 = (x_0 + x_1) \bmod 2^{64}, \quad y_1 = (x_1 \lll R) \oplus y_0.$$

Здесь $\lll$ — циклический сдвиг влево; R — константа, которая задана для каждого раунда. В итоге соответствующей паре слов внутреннего состояния присваиваются значения (y_0, y_1).

Вернёмся к функции сжатия Skein-512-256. Её 512-битовый выход вычисляется следующим образом: $E(K, T, P) \oplus P$, где P — это текущий блок сообщения. Таким образом, стойкость функции сжатия полностью зависит от стойкости блочного шифра Threefish.

Для вычисления хеша функция сжатия вызывается в режиме Unique Block Iteration (UBI), который является вариантом режима Matyas — Meyer — Oseas [9]. На первом блоке сообщения на вход функции сжатия в качестве K подаётся набор инициализирующих констант, указанных в стандарте [2], а на всех остальных блоках сообщения — 512-битовый выход функции сжатия на предыдущем блоке сообщения. Также подаётся на вход и 128-битовое значение T, которое содержит различную служебную информацию: сколько байт сообщения уже обработано, является ли текущий блок сообщения первым или последним и т. д. Благодаря режиму UBI функция сжатия является параметризованной, а каждый блок сообщения обрабатывается уникальной версией этой функции. Это сделано для повышения криптографической стойкости хеш-функций семейства Skein. На последнем блоке сообщения функция сжатия вызывается дважды, затем итоговый хеш Skein-512-256 вычисляется взятием 256 старших битов внутреннего состояния.

Лучшая на данный момент теоретическая атака нахождения прообраза на полнораундовую хеш-функцию Skein-512-256 требует перебора $2^{511,76}$ вариантов [10], что лишь немного лучше, чем полный перебор. В [3] представлена практическая атака нахождения прообраза на первый раунд хеш-функции Skein-512-256, а также на первый раунд функции сжатия Skein-512-256.

В настоящей работе исследуется стойкость неполнораундовых версий функции сжатия Skein-512-256 к практическим алгебраическим атакам нахождения прообраза. Как и в [3], для этого применяется SAT-подход. В следующем пункте с этой целью строится SAT-кодировка функции сжатия Skein-512-256.

2. SAT-кодировка нахождения прообраза функции сжатия Skein-512-256

Опишем постановку задачи нахождения прообраза неполнораундовой функции сжатия Skein-512-256, SAT-кодировку этой задачи и семейство КНФ, сгенерированных на её основе.

2.1. Постановка задачи

Рассмотрим функцию сжатия Skein-512-256 при её вызове на первом блоке сообщения. В этом случае 512-битовый секретный ключ K блочного шифра Threefish известен (см. п. 1). Значения двух 64-битовых слов параметра T в случае первого вызова равны 64 и 8070450532247928832 соответственно [2]. Из входных данных остаётся неизвестным только 512-битовый блок сообщения P. Рассматривается следующая задача: по известному 512-битовому выходу неполнораундовой функции сжатия найти неизвестный 512-битовый вход (т. е. блок сообщения). Несмотря на то, что в такой постановке секретный ключ известен, это не делает задачу простой. Действительно, перед первым раундом шифра Threefish внутреннее состояние S инициализируется путём смешивания раундовых ключей (которые известны, так как они зависят от секретного ключа), T и P. Раз P неизвестно, то и S тоже неизвестно.

2.2. Кодирование в SAT с помощью CBMC

На предварительном этапе были проанализированы существующие трансляторы, способные сводить задачи анализа стойкости криптографических хеш-функций к SAT. Зачастую для этих целей используется Transalg [11]. В частности, доступны онлайн построенные с его помощью КНФ, кодирующие задачи нахождения коллизий и прообразов функций сжатия MD4, MD5, SHA-1 и SHA256. При этом Transalg поддерживает только сложение по модулю 2^{32}, а в функции сжатия Skein-512-256 используется сложение по модулю 2^{64}. По этой причине Transalg использован не был.

В результате предварительного анализа был выбран транслятор Bounded Model Checker for C programs (CBMC), предназначенный для проверки свойств программ на языке программирования C [12]. Одним из способов проверки является сведение проверяемого свойства и описанного в программе алгоритма к SAT путём формирования КНФ. Отметим, что CBMC не может закодировать в SAT операции над вещественными числами — все переменные, с которыми оперирует алгоритм, должны быть целочисленными. В построенной с помощью CBMC КНФ часть булевых переменных соответствует входу алгоритма, другая часть — выходу алгоритма, а остальные переменные являются дополнительными и нужны для преобразований, которые вычисляют выход по входу. Если в КНФ присвоить значения только выходным переменным, то сформируется задача поиска входа по известному выходу. При этом достаточно закодировать сам алгоритм, т. е. никакие свойства программы проверять не требуется. Именно в таком режиме CBMC используется далее.

CBMC принимает на вход программу на языке C после предварительной подготовки, которая подробно описана в [13]. Эта подготовка состоит из следующих ключевых этапов:

1) из заголовочных файлов и файлов с исходным кодом сформировать один файл с исходным кодом;
2) если целочисленной переменной `var` присваивается значение `val`, но эта переменная, а также операции над ней должны быть закодированы в КНФ, то добавить в исходный код следующую строку: `__CPROVER_assume(var==val)`.

Отметим, что `__CPROVER_assume(var==val)` — это специальная CBMC-функция, которой нет в составе языка C. При её использовании в КНФ добавляются дизъюнкты, кодирующие равенство `var=val`. Булевы переменные КНФ, кодирующие целочисленную переменную `var`, должны быть уже введены ранее путём объявления переменной `var` (без инициализации) и присваивания ей результата операций над другими переменными программы. Если в исходном коде переменной `var` присвоить значение `val` напрямую, то переменные и дизъюнкты в КНФ для неё добавлены не будут. В случае функции сжатия Skein-512-256 для построения корректной SAT-кодировки специальная функция `__CPROVER_assume()` должна быть применена для присваивания секретному ключу инициализирующих констант, а также для означивания выхода функции сжатия. Используемые значения выхода рассмотрены далее.

Для генерации КНФ взята реализация криптографической хеш-функции Skein на языке C, поданная ранее на конкурс Национального института стандартов и технологий США [2]. Конкретнее — речь о Skein версии 1.3. Затем была выделена реализация функции сжатия Skein-512-256, которая в основном содержится в функции `Skein_512_Process_Block()`. После этого была проведена предварительная подготовка для запуска CBMC, о которой речь шла выше.

2.3. Генерация задач нахождения прообраза

Нахождение прообраза первого раунда функции сжатия Skein-512-256 в описанной постановке является простой задачей для современных SAT-решателей. При этом для первых двух раундов соответствующая задача уже очень сложная. По этой причине были исследованы промежуточные варианты. Напомним, что на каждом раунде внутреннее состояние модифицируется путём четырёх вызовов функции MIX, которая состоит из трёх операций: сложения по модулю 2^{64}, циклического сдвига, сложения по модулю 2. Таким образом, каждый раунд состоит из двенадцати базовых операций. Рассмотрим одиннадцать версий неполнораундовой функции сжатия Skein-512-256, таких, что в i-й функции используется первый раунд и первые i операций второго раунда, где $i \in \{1, \ldots, 11\}$.

Для каждой из одиннадцати функций сжатия анализировалось нахождение прообразов следующих десяти выходов:

— 512 нулевых битов;
— 512 единичных битов;
— восемь 512-битовых последовательностей с регулярной структурой и одинаковым числом нулей и единиц.

Выходы с регулярной структурой были сформированы следующим образом: j-й выход равен последовательности из 2^j единичных битов и 2^j нулевых битов, которая повторялась $512 \cdot 2^{-j-1}$ раз, где $j \in \{1, \ldots, 8\}$. Таким образом формируется периодическая последовательность с периодом $512 \cdot 2^{-j-1}$: первый выход с регулярной структурой равен последовательности 1100, повторённой 128 раз, а последний — 256 единичным и 256 нулевым битам.

Для генерации 110 КНФ использован транслятор CBMC версии 5.95.1, который вызывался с ключом `--dimacs`. Характеристики построенных КНФ приведены в табл. 1. Ключевые характеристики не зависят от значения выхода, поэтому в таблице приведены КНФ только для первого выхода, состоящего из 512 нулевых битов.

Таблица 1

Характеристики КНФ, кодирующих нахождение прообразов нулевого выхода

i	Переменные	Дизъюнкты	Литералы
1	4 925	15 404	41 942
2	4 989	15 532	42 198
3	5 053	15 788	42 966
4	5 181	16 677	45 757
5	5 245	16 805	46 013
6	5 309	17 061	46 781
7	5 437	17 950	49 572
8	5 501	18 078	49 828
9	5 565	18 334	50 596
10	5 693	19 223	53 387
11	5 757	19 351	53 643

В дополнение к данным табл. 1 отметим, что CBMC кодирует каждый вызов функции MIX путём добавления 256 булевых переменных и 1 273 дизъюнктов, эти дизъюнкты содержат 3 815 литералов.

3. Вычислительные эксперименты

Представим результаты применения SAT-решателей к описанным КНФ. Все эксперименты проведены на персональном компьютере, оснащённом 64 гигабайтами оперативной памяти и 16-ядерным процессором AMD Ryzen 3950X.

3.1. Нахождение прообразов при помощи CDCL-решателя

На первом этапе был использован CDCL-решатель Kissat [14], который последние годы несколько раз становился победителем конкурса SAT Competition. Конкретнее — использована версия 4.0.1 этого решателя. Kissat является однопоточной программой, поэтому в рамках экспериментов каждому экземпляру Kissat было выделено одно ядро процессора. На каждой из 110 КНФ Kissat был запущен с лимитом времени 24 ч. При $i = 1$ и 2 (напомним, что i означает количество операций во втором раунде) все SAT-задачи были решены, при этом в среднем решатель работал менее одной секунды. Примерно для половины выходов был найден выполняющий набор, т. е. были найдены прообразы ослабленных версий функции сжатия Skein512-256. Для оставшейся половины выходов решателем была доказана невыполнимость КНФ, т. е. доказано, что у этих выходов нет прообразов для данных версий функции сжатия.

Для $i = 3$ ситуация поменялась — SAT-задачу для выхода № 10 решатель не смог решить за 24 ч; напомним, что этот выход состоит из 256 единичных и 256 нулевых битов. При этом большинство SAT-задач были решены менее чем за секунду или за несколько секунд. Дла $i = 4$ картина оказалась примерно такой же — не была решена только SAT-задача для выхода № 10. Для $i = 5$ уже две SAT-задачи не были решены. Далее сложность SAT-задач существенно растёт с увеличением i. Для $i = 8$ и всех последующих i ни одна SAT-задача решена не была. В табл. 2 приведены результаты для $i = 4, 5, 6, 7, 8$. Запись «SAT»/«UNSAT» значит, что для соответствующей КНФ была доказана выполнимость/невыполнимость. В первом случае это означает нахождение прообраза, а во втором — доказательство, что для конкретного выхода прообраза не существует. Запись «–» означает, что Kissat не смог решить SAT-задачу за 24 ч. Если SAT-задача решена менее чем за секунду, то указана дробная часть; в остальных случаях время округлено до целого значения.

Таблица 2

Результаты решения SAT-задач с помощью Kissat и время решения, с

№ выхода	$i = 4$		$i = 5$		$i = 6$		$i = 7$		$i = 8$	
1	UNSAT	0,40	UNSAT	0,44	UNSAT	2	UNSAT	4	–	–
2	UNSAT	0,15	UNSAT	0,29	–	–	–	–	–	–
3	SAT	29	UNSAT	0,06	SAT	133	–	–	–	–
4	UNSAT	18	UNSAT	20	UNSAT	67	UNSAT	109	–	–
5	SAT	4	SAT	2	UNSAT	419	SAT	26 066	–	–
6	SAT	1	SAT	0,39	SAT	180	–	–	–	–
7	UNSAT	2	UNSAT	2	UNSAT	21	UNSAT	33	–	–
8	UNSAT	0,15	–	–	–	–	–	–	–	–
9	UNSAT	0,15	UNSAT	840	UNSAT	396	UNSAT	1731	–	–
10	–	–	–	–	–	–	–	–	–	–

Судя по результатам, приведённым в табл. 2, наличие прообраза, а также сложность решения SAT-задачи существенно зависит от выхода. Для выхода № 1 (т. е. для 512 нулевых битов) доказано отсутствие прообраза в четырёх случаях. Кроме того, SAT-задачи для этого выхода оказались очень простыми. SAT-задачи для выхода № 10 оказались самыми сложными — решатель не смог решить ни одну из них. Наибольшее

количество прообразов найдено для выходов № 5 и 6, а время решения этих SAT-задач почти во всех случаях существенно больше, чем для выхода № 1. Из представленных результатов следует, что рассмотренные версии функции сжатия при $1 \leqslant i \leqslant 7$ (из $\{0,1\}^{512}$ в $\{0,1\}^{512}$) не биективны, так как они не сюръективны. Можно выдвинуть гипотезу, что и полнораундовая функция сжатия Skein-512-256 не биективна.

3.2. Нахождение прообразов при помощи Cube-and-Conquer

Одним из самых эффективных подходов к решению трудных SAT-задач является Cube-and-Conuqer [8]. Согласно этому подходу, сначала на КНФ запускается lookahead-эвристика, которая осуществляет декомпозицию задачи на более простые подзадачи. При этом lookahead строит двоичное дерево, в котором узлы соответствуют переменным КНФ, а рёбра — значениям переменных. Есть два типа листьев такого дерева — отсечённые листья, для которых lookahead самостоятельно доказал отсутствие решений, и прерванные листья. Каждая из подзадач формируется путём подстановки в КНФ значений всех переменных от корня дерева до соответствующего прерванного листа. Набор этих значений называется кубом, а процедура декомпозиции — кубированием.

На этапе кубирования ключевым является значение порога прерывания n. Прерванный лист формируется в том случае, когда при подстановке текущего куба в КНФ и простейшего препроцессинга (итеративного применения правила единичного дизъюнкта) в модифицированной КНФ оказывается меньше n переменных. На втором этапе метода Cube-and-Conuqer на каждой подзадаче, соответствующей прерванному листу, запускается CDCL-решатель. Если исходная КНФ выполнима, то в результате решения хотя бы одной подзадачи будет найден выполняющий набор. Если же исходная КНФ невыполнима, то на всех подзадачах будет доказана невыполнимость. Все подзадачи можно решать независимо друг от друга, поэтому Cube-and-Conuqer хорошо подходит для решения трудных SAT-задач в параллельных вычислительных системах.

С помощью Cube-and-Conuqer был впервые решён ряд трудных комбинаторных задач. Самой известной из них является булева проблема пифагоровых троек [15]. В [16] предложен параллельный SAT-решатель EnCnC, реализующий метод Cube-and-Conuqer для анализа стойкости криптографических примитивов с учётом ряда особенностей этой предметной области. С помощью этого решателя были впервые найдены прообразы 43-шаговой функции сжатия MD4, 29-раундовой функции сжатия MD5 и 24-раундовой функции сжатия SHA-1 [16, 17].

Кратко рассмотрим принцип работы EnCnC в режиме полного SAT-решателя. В качестве начального значения n выбирается число переменных в КНФ после препроцессинга. Затем n уменьшается с некоторым шагом и для каждого из новых значений с помощью lookahead-решателя генерируются все подзадачи, при этом отслеживается число отсечённых листьев. Процесс заканчивается, когда на очередном n число подзадач превышает заданную верхнюю границу. Затем формируется набор значений n, таких, что число соответствующих подзадач находится в заданном диапазоне (т. е. учитывается не только верхняя, но и нижняя граница), а число отсечённых листьев не меньше заданного лимита. На следующем этапе для каждого выбранного таким образом n формируется случайная выборка подзадач. Подзадачи из выборки решаются с помощью CDCL-решателя с некоторым лимитом времени. Для выборок, в которых все подзадачи решены, вычисляется прогнозное время решения всех подзадач. Прогнозное время вычисляется путём умножения среднего времени решения подзадач из выборки на общее число подзадач. В итоге выбирается лучшее значение n с точки

зрения прогнозного времени и запускается решение всех оставшихся нерешёнными подзадач, сформированных на этом n.

SAT-решатель EnCnC был запущен на четырёх КНФ, которые соответствуют $i = 8, 9$ и выходам № 5 и 6. Эти выходы выбраны по той причине, что на них чаще всего находились прообразы при меньших значениях i. Напомним, что эти выходы равны последовательности 1111111100000000, повторённой 32 раза, и последовательности 11111111111111110000000000000000, повторённой 16 раз, соответственно; при $i = 8$ рассматривается задача нахождения прообраза функции сжатия Skein512-256, которая состоит из первого раунда и 8 первых операций (из 12) второго раунда. Для $i = 9$ задача определяется аналогично. Вычислительные эксперименты проводились на том же компьютере с 16-ядерным процессором. Использованы следующие значения входных параметров EnCnC:

— -nstep = 5 — значение n уменьшается с шагом 5;
— -minc = 1000 — выбираются только такие n, для которых формируется минимум 1 000 подзадач;
— -maxc = 20000 — выбираются только такие n, для которых формируется максимум 20 000 подзадач;
— -minref = 1 — выбираются только такие n, для которых отсекается минимум 1 лист;
— -sample = 100 — 100 подзадач в случайной выборке для каждого выбранного n;
— -lasolver = march_cu — lookahead-решатель march_cu;
— -cdclsolvers = kissat4.0.1 — CDCL-решатель Kissat версии 4.0.1;
— -maxcdcltime = 5000 — CDCL-решатель работает с лимитом времени 5 000 с.

Результаты этапа прогнозирования для $i = 8$ представлены в табл. 3. Прогнозное время указано в сутках при условии использования 16 ядер; приведены только выбранные на предварительном этапе значения n. Для обеих КНФ начальное значение n равно 2 465.

Таблица 3

Результаты этапа прогнозирования для КНФ при $i = 8$

n	Подзадач	Отсечённых листьев	Прогнозное время решения
Выход № 5			
2 355	1 431	1	8 ч 1 мин
2 350	4 619	11	2 дн. 2 ч 54 мин
2 345	7 876	23	2 дн. 8 ч 44 мин
2 340	13 262	42	3 дн. 6 мин
Выход № 6			
2 355	2 567	1	2 дн. 7 ч 18 мин
2 345	7 916	3	4 дн. 5 ч 5 мин
2 340	6 141	4	2 дн. 17 ч 53 мин
2 330	16 096	19	5 дн. 4 ч 2 мин

Согласно табл. 3, лучшие прогнозные значения для обеих КНФ найдены на $n = 2355$, поэтому именно этот порог прерывания был использован на этапе решения. Все подзадачи для выхода № 5 были решены за 8 ч 46 мин на 16 ядрах. Оказалось, что все соответствующие КНФ невыполнимы, т. е. доказано, что прообразов для выхода № 5 при $i = 8$ не существует. В случае выхода № 6 выполняющий набор (соответственно и прообраз для данного выхода) на одной из КНФ был найден за 6 ч 7 мин, при этом было решено 15,7 % подзадач.

Результаты прогнозирования для случая $i = 9$ представлены в табл. 4.

Таблица 4

Результаты этапа прогнозирования для КНФ при $i = 9$

n	Подзадач	Отсечённых листьев	Прогнозное время решения
Выход № 5			
2 445	1 879	102	40 с
2 440	2 622	146	44 с
2 435	3 649	216	52 с
2 430	5 054	301	1 м 26 с
2 425	6 938	432	1 м 24 с
2 420	9 343	572	1 м 42 с
2 415	12 622	802	2 м 13 с
2 410	17 020	1078	3 м 19 с
Выход № 6			
2 430	2 321	98	11 с
2 425	3 136	151	11 с
2 420	4 168	200	15 с
2 415	5 448	298	19 с
2 410	7 260	402	25 с
2 405	9 426	559	32 с
2 400	12 034	741	34 с
2 395	15 070	1044	51 с
2 390	18 634	1392	53 с

Прогнозное время для $i = 9$ оказалось необычно маленьким. Это контринтуитивно, так как для $i = 9$ SAT-задачи должны быть сложнее (или, по крайней мере, не проще), чем для $i = 8$. Причина в том, что во всех случаях в выборках подзадачи были очень простыми — Kissat решал их быстрее чем за 1 с. Эти прогнозы проверены в режиме решения. В случае выхода № 5 использовалось значение $n = 2445$; 1 876 подзадач были решены очень быстро (в среднем за 0,5 с), но оставшиеся 3 подзадачи не были решены даже за 24 ч. Для выхода № 6 ситуация повторилась — 2320 подзадач были решены в среднем за 0,1 с при $n = 2430$, но оставшаяся подзадача также не решилась за 24 ч. Таким образом, прогнозное время оказалось неточным из-за того, что в выборки были включены только очень простые подзадачи. Отметим, что ранее при анализе стойкости MD4, MD5 и SHA-1 с помощью EnCnC такой ситуации не возникало.

3.3. Применение Cube-and-Conquer для препроцессинга КНФ

Для решения проблемы с излишне оптимистичными прогнозами в случае $i = 9$ в рамках настоящего исследования был разработан алгоритм итеративного применения Cube-and-Conquer. Цель этого алгоритма состоит в преобразовании КНФ таким образом, чтобы Cube-and-Conquer не формировал по ней слишком простые подзадачи.

Алгоритм принимает на вход следующие данные:

1) КНФ C;
2) число кубов k;
3) lookahead-решатель lasolver;
4) CDCL-решатель cdclsolver;
5) максимальное число конфликтов confl для CDCL-решателя.

Алгоритм состоит из следующих шагов:

Шаг 1. Начиная с порога прерывания n, равного числу переменных в C после препроцессинга по правилу единичного дизъюнкта, уменьшать n и запускать lasolver на C; найти минимальное n такое, что lasolver формирует не более k подзадач на C.

Шаг 2. Запустить на каждой из k подзадач cdclsolver с лимитом в confl конфликтов.

Шаг 3. Пусть cdclsolver решил $s \leqslant k$ подзадач, а на остальных $k - s$ подзадачах он был прерван при достижении confl конфликтов. Возникает пять случаев:

1) $s = 0$: выдать UNKNOWN и модфицированную КНФ C; остановить алгоритм;
2) $s = k$ и на всех s подзадачах cdclsolver доказал невыполнимость: выдать UNSAT и остановить алгоритм;
3) $s > 0$ и хотя бы на одной из s подзадач cdclsolver нашёл выполняющий набор: выдать SAT и остановить алгоритм;
4) $0 < s < k - 1$ и для всех s подзадач cdclsolver доказал невыполнимость: добавить в C s дизъюнктов, являющихся отрицаниями s соответствующих кубов (на которых C оказалась невыполнимой); перейти к шагу 1;
5) $s = k - 1$ и для всех s подзадач cdclsolver доказал невыполнимость: добавить в C s дизъюнктов, являющихся отрицаниями s соответствующих кубов (на которых C оказалась невыполнимой); добавить в C однолитеральные дизъюнкты, состоящие из литералов, входящих в оставшийся куб; перейти к шагу 1.

Рассмотрим пример. Допустим, что $k = 4$ и на первом шаге найдено минимальное n, такое, что lasolver сгенерировал на КНФ C четыре куба. Конкретнее, речь о следующих кубах: $\neg x_3 \wedge x_7$; $\neg x_4 \wedge x_9$; $x_1 \wedge \neg x_3 \wedge x_9$; $x_5 \wedge x_{15} \wedge \neg x_{21}$. Допустим также, что на шаге 2 на подзадачах № 1 и 3 cdclsolver доказал невыполнимость (т. е. доказана невыполнимость КНФ $C \wedge \neg x_3 \wedge x_7$ и $C \wedge x_1 \wedge \neg x_3 \wedge x_9$), а на оставшихся двух подзадачах cdclsolver был прерван. Тогда на шаге 3 возникает четвёртый случай: в C добавляются дизъюнкты $\neg(\neg x_3 \wedge x_7) = (x_3 \vee \neg x_7)$ и $\neg(x_1 \wedge \neg x_3 \wedge x_9) = (\neg x_1 \vee x_3 \vee \neg x_9)$. Иными словами, C заменяется на $C \wedge (x_3 \vee \neg x_7) \wedge (\neg x_1 \vee x_3 \vee \neg x_9)$ и на этой модифицированной КНФ выполняется шаг 1 алгоритма.

Если бы в рассмотренном примере cdclsolver смог на шаге 2 успешно решить ещё и подзадачу № 4, то на шаге 3 возник бы пятый случай и к КНФ C были бы добавлены следующие дизъюнкты: $(x_3 \vee \neg x_7)$; $(\neg x_1 \vee x_3 \vee \neg x_9)$; $(\neg x_5 \vee \neg x_{15} \vee x_{21})$; $(\neg x_4)$; (x_9). Последние два однолитеральных дизъюнкта кодируют тот факт, что куб № 2 является истинным.

Отметим, что предложенный алгоритм идейно схож с добавлением к КНФ дизъюнктов, сгенерированных с помощью лазеек (англ. backdoors) [18]. И в том и в другом случае добавляемые дизъюнкты являются логическим следствием исходной КНФ, а значит, никакие выполняющие наборы (если они есть) таким образом не запрещаются. При этом пятый случай шага 3 особенный: фактически с помощью добавления однолитеральных дизъюнктов осуществляется присваивание значений соответствующих переменных. Это следствие того, что на всех остальных кубах было доказано, что КНФ невыполнима.

Представленный выше алгоритм, выполняющий препроцессинг с помощью Cube-and-Conquer, реализован на языке программирования C++ в виде многопоточной программы. Исходный код программы, а также все используемые в этом исследовании КНФ и программы для CBMC доступны онлайн [19]. В определённых условиях эта многопоточная программа может решить SAT-задачу, но основная её цель состоит в препроцессинге КНФ таким образом, чтобы по ней Cube-and-Conquer не генерировал слишком простые задачи (такие, как в п. 3.2 при $i = 9$). Кроме того, на модифицированной таким образом КНФ SAT-задача может быть решена быстрее, чем до модификации.

Сначала разработанная программа была запущена на компьютере на каждой из одиннадцати SAT-задач, на которых Kissat не смог найти решение за 24 ч при $i \in \{4, 5, 6, 7\}$ (см. п. 3.1). Использовались следующие входные параметры: $k = 16$ (по числу ядер процессора); lookahead-решатель march_cu; CDCL-решатель Kissat; confl = 30 000. При таком ограничении на число конфликтов Kissat работает примерно 0,5 с. Время работы программы и число итераций алгоритма зависело от КНФ: всего потребовалось от 4 до 67 с и от 2 до 12 итераций. На модифицированных КНФ был снова запущен Kissat на 24 ч. В результате была решена одна SAT-задача, а именно — найден прообраз выхода № 10 для $i = 5$ за 5 ч. Приведём подробную статистику препроцессинга на этой КНФ: выполнены 4 итерации в течение 54 с; при этом было три пятых случая на шаге 3 и затем один первый случай. Всего по результатам препроцессинга в КНФ было добавлено 64 дизъюнкта, 16 из них — однолитеральных. Отметим, что этот выход является самым сложным — до этого Kissat смог решить SAT-задачу для него максимум для $i = 2$.

На следующей стадии программа была применена с теми же параметрами к двум КНФ для $i = 9$, рассмотренным выше. На КНФ для выхода № 5 препроцессинг ничего не дал, так как в первой же итерации на шаге 3 возник первый случай, т. е. на всех подзадачах Kissat был прерван. По этой причине программа была запущена ещё раз, но с параметром confl = 100 000. С таким ограничением Kissat работает примерно 3 с. На этот раз на КНФ для выхода № 5 программа отработала за 207 с, в течение которых было выполнено 30 итераций алгоритма. В модифицированной КНФ в итоге оказалось на 239 дизъюнктов больше, чем в исходной. На КНФ для выхода № 6 программа выполнила 9 итераций за 99 с (при этом также было использовано confl = 100 000). В результате в КНФ были добавлены 164 дизъюнкта. На обеих модифицированных КНФ был запущен Cube-and-Conquer-решатель EnCnC в режиме прогнозирования с теми же входными параметрами, что и в п. 3.2. Результаты представлены в табл. 5.

Таблица 5

Результаты для КНФ при $i = 9$ после препроцессинга

n	Подзадач	Отсечённых листьев
Выход № 5		
1355	2 631	3
1350	4 761	3
1345	8 485	6
1340	15 028	9
Выход № 6		
1345	3 018	1
1340	5 376	2
1335	9 700	2
1330	17 103	4

Во-первых, отметим, что значения n значительно меньше, чем до препроцессинга. Для выхода № 5 они меняются в диапазоне 1340–1355 переменных, а ранее было 2410–2445. Для выхода № 6 ситуация аналогичная. Это один из ключевых результатов предложенного препроцессинга — по сути, в модифицированных КНФ с точки зрения Cube-and-Conquer примерно на 1 000 переменных меньше, чем в исходной. Во-вторых, во всех выборках подзадачи оказались сложными и не были решены за 5 000 с, поэтому в табл. 5 отсутствует столбец с прогнозным временем решения. Таким образом, EnCnC теперь даёт гораздо более реалистичную картину, чем рань-

ше. Можно построить такую нижнюю оценку прогнозного времени для выхода № 5: если каждую из 15 028 подзадач можно решить за 5 000 с, то на 16-ядерном компьютере на это потребуется 55 дней. Для выхода № 6 аналогичная нижняя оценка составляет 62 дня работы компьютера.

Заключение

Проанализирована стойкость неполнораундовых вариантов функции сжатия Skein512-256 к нахождению прообразов. Ранее в данном контексте рассматривался только первый раунд данной функции сжатия; в настоящей работе рассмотрены варианты с первым раундом и несколькими первыми операциями второго раунда. Задачи нахождения прообраза сведены к SAT с помощью транслятора CBMC. Показано, что время решения соответствующих SAT-задач при помощи алгоритма CDCL и метода Cube-and-Conquer существенно зависит от значения выхода функции сжатия, а также от используемого числа операций второго раунда. В результате найдены прообразы функции сжатия, состоящей из первого раунда и 8 (из 12) операций второго раунда. Предложен подход к препроцессингу КНФ с помощью итеративного использования метода Cube-and-Conquer. Предварительное применение этого препроцессинга позволило найти прообраз для одной конфигурации, которая ранее оказывалась слишком трудной для CDCL-решателя. Также этот препроцессинг позволил построить более точные оценки времени нахождения прообраза версии функции сжатия, в которой число операций второго раунда увеличено до 9. Можно констатировать, что на данный момент даже 2 (из 72) раунда функции сжатия Skein512-256 являются стойкими к нахождению прообраза с помощью SAT-подхода.

ЛИТЕРАТУРА

1. *Алферов А. П., Зубов А. Ю., Кузьмин А. С., Черемушкин А. В.* Основы криптографии. 2-е изд. М.: Гелиос АРВ, 2002. 480 с.
2. `https://www.schneier.com/academic/skein/` — The Skein Hash Function Family. 2010.
3. *Homsirikamol E., Morawiecki P., Rogawski M., and Srebrny M.* Security margin evaluation of SHA-3 contest finalists through SAT-based attacks // LNCS. 2012. V. 7564. P. 56–67.
4. *Bard G. V.* Algebraic Cryptanalysis. N.Y.: Springer, 2009. 356 p.
5. *Семёнов А. А., Беспалов Д. В.* Технологии решения многомерных задач логического поиска // Вестн. Том. гос. ун-та. 2005. № 14. С. 61–73.
6. *Marques-Silva J. and Sakallah K.* GRASP: a search algorithm for propositional satisfiability // IEEE Trans. Computers. 1999. V. 48. No. 5. P. 506–521.
7. *Mironov I. and Zhang Z.* Applications of SAT solvers to cryptanalysis of hash functions // LNCS. 2006. V. 4121. P. 102–115.
8. *Heule M. J. H., Kullmann O., Wieringa S., and Biere A.* Cube and Conquer: guiding CDCL SAT solvers by lookaheads // LNCS. 2012. V. 7261. P. 50–65.
9. *Matyas S. M., Meyer C. H., and Oseas J.* Generating strong one-way functions with cryptographic algorithm // IBM Techn. Disclosure Bull. 1985. V. 27. No. 10A. P. 5658–5659.
10. *Khovratovich D., Rechberger C., and Savelieva A.* Bicliques for preimages: attacks on Skein-512 and the SHA-2 family // LNCS. 2012. V. 7549. P. 244–263.
11. *Отпущенников И. В., Семёнов А. А.* Технология трансляции комбинаторных проблем в булевы уравнения // Прикладная дискретная математика. 2011. № 1(11). С. 96–115.
12. *Clarke E. M., Kroening D., and Lerda F.* A tool for checking ANSI-C programs // LNCS. 2004. V. 2978. P. 168–176.

13. *Заикин О. С., Давыдов В. В., Кирьянова А. П.* Применение алгоритмов решения проблемы булевой выполнимости для анализа финалистов конкурса SHA-3 // Выч. мет. программирование. 2024. Т. 25. Вып. 3. С. 259–273.
14. *Biere A. and Fleury M.* Gimsatul, IsaSAT and Kissat entering the SAT Competition 2022 // Proc. SAT Competition 2022 — Solver and Benchmark Descriptions. University of Helsinki, 2022. P. 10–11.
15. *Heule M. J. H., Kullmann O., and Marek V. W.* Solving and verifying the Boolean Pythagorean triples problem via Cube-and-Conquer // LNCS. 2016. V. 9710. P. 228–245.
16. *Zaikin O.* Inverting cryptographic hash functions via Cube-and-Conquer // J. Artif. Int. Res. 2024. V. 81. P. 359–399.
17. *Zaikin O.* Inverting step-reduced SHA-1 and MD5 by parameterized SAT solvers // Proc. CP 2024. Leibniz Intern. Proc. Informatics (LIPIcs). 2024. V. 307. P. 31:1–31:19.
18. *Andreev A., Chukharev K., Kochemazov S., and Semenov A.* Using backdoors to generate learnt information in SAT solving // Proc. ECAI 2024. P. 4173–4180.
19. `https://github.com/olegzaikin/IterCnC` — Реализация итеративного препроцессинга КНФ при помощьи Cube-and-Conquer. 2025.

REFERENCES

1. *Alferov A. P., Zubov A. Yu., Kuzmin A. S., and Cheremushkin A. V.* Osnovy kriptografii [Basics of Cryptography]. 2nd ed. Moscow, Gelios ARV, 2002. 480 p. (in Russian)
2. `https://www.schneier.com/academic/skein/` — The Skein Hash Function Family. 2010.
3. *Homsirikamol E., Morawiecki P., Rogawski M., and Srebrny M.* Security margin evaluation of SHA-3 contest finalists through SAT-based attacks. LNCS, 2012, vol. 7564, pp. 56–67.
4. *Bard G. V.* Algebraic Cryptanalysis. N.Y., Springer, 2009. 356 p.
5. *Semenov A. A. and Bespalov D. V.* Tekhnologii resheniya mnogomernykh zadach logicheskogo poiska [Technologies for solving multidimensional logical search problems]. Vestnik TSU, 2005, no. 14, pp. 61–73. (in Russian)
6. *Marques-Silva J. and Sakallah K.* GRASP: a search algorithm for propositional satisfiability. IEEE Trans. Computers, 1999, vol. 48, no. 5, pp. 506–521.
7. *Mironov I. and Zhang Z.* Applications of SAT solvers to cryptanalysis of hash functions. LNCS, 2006, vol. 4121, pp. 102–115.
8. *Heule M. J. H., Kullmann O., Wieringa S., and Biere A.* Cube and Conquer: guiding CDCL SAT solvers by lookaheads. LNCS, 2012, vol. 7261, pp. 50–65.
9. *Matyas S. M., Meyer C. H., and Oseas J.* Generating strong one-way functions with cryptographic algorithm. IBM Techn. Disclosure Bull., 1985, vol. 27, no. 10A, pp. 5658–5659.
10. *Khovratovich D., Rechberger C., and Savelieva A.* Bicliques for preimages: attacks on Skein-512 and the SHA-2 family. LNCS, 2012, vol. 7549, pp. 244–263.
11. *Otpuschennikov I. V. and Semenov A. A.* Tekhnologiya translyatsii kombinatornykh problem v bulevy uravneniya [Technology for translating combinatorial problems into Boolean equations]. Prikladnaya Diskretnaya Matematika, 2011, no. 1(11), pp. 96–115. (in Russian)
12. *Clarke E. M., Kroening D., and Lerda F.* A tool for checking ANSI-C programs. LNCS, 2004, vol. 2978, pp. 168–176.
13. *Zaikin O. S., Davydov V. V., and Kiryanova A. P.* Primeneniye algoritmov resheniya problemy bulevoy vypolnimosti dlya analiza finalistov konkursa SHA-3 [SAT-based analysis of SHA-3 competition finalists]. Vychislitel'nye Metody i Programmirovanie, 2024, vol. 25, no. 3, pp. 259–273. (in Russian)

14. *Biere A. and Fleury M.* Gimsatul, IsaSAT and Kissat entering the SAT Competition 2022. Proc. SAT Competition 2022 — Solver and Benchmark Descriptions, University of Helsinki, 2022, pp. 10–11.
15. *Heule M. J. H., Kullmann O., and Marek V. W.* Solving and verifying the Boolean Pythagorean triples problem via Cube-and-Conquer. LNCS, 2016, vol. 9710, pp. 228–245.
16. *Zaikin O.* Inverting cryptographic hash functions via Cube-and-Conquer. J. Artif. Int. Res., 2024, vol. 81, pp. 359–399.
17. *Zaikin O.* Inverting step-reduced SHA-1 and MD5 by parameterized SAT solvers. Proc. CP 2024, Leibniz Intern. Proc. Informatics (LIPIcs), 2024, vol. 307, pp. 31:1–31:19.
18. *Andreev A., Chukharev K., Kochemazov S., and Semenov A.* Using backdoors to generate learnt information in SAT solving. Proc. ECAI 2024, pp. 4173–4180.
19. `https://github.com/olegzaikin/IterCnC` — Implementation of iterative Cube-and-Conquer-based preprocessing for SAT, 2025.