2024, **25** (3), 259–273. doi 10.26089/NumMet.v25r320

doi 10.26089/NumMet.v25r320 УДК 519.6, 004.8

Применение алгоритмов решения проблемы булевой выполнимости для анализа финалистов конкурса SHA-3

О. С. Заикин
Новосибирский государственный университет, Новосибирск, Российская Федерация
Институт динамики систем и теории управления имени В. М. Матросова СО РАН, Иркутск, Российская Федерация
ORCID: 0000-0002-0145-5010, e-mail: oleg.zaikin@icc.ru

В. В. Давыдов
Санкт-Петербургский государственный университет аэрокосмического приборостроения, Санкт-Петербург, Российская Федерация
QApp, Москва, Российская Федерация
ORCID: 0000-0002-5544-2434, e-mail: vadimdavydov@outlook.com

А. П. Кирьянова
Университет ИТМО, Санкт-Петербург, Российская Федерация
ORCID: 0009-0006-0344-5111, e-mail: anastaciakosanovskaya@gmail.com

Аннотация: Рассматриваются финалисты конкурса SHA-3, организованного для принятия нового стандарта криптографической хеш-функции. Все пять финалистов до сих пор являются стойкими к поиску прообраза, т.е. для них невозможно за реальное время найти вход по известному выходу. Исследуется возможность нахождения прообразов неполнораундовых версий рассмотренных криптографических хеш-функций. Соответствующие задачи сведены к проблеме булевой выполнимости (SAT) с помощью инструментального средства CBMC, предназначенного для проверки свойств программ на языке C. Для решения построенных экземпляров SAT использован современный решатель Kissat. В сравнении с ранее опубликованными результатами, для четырех из пяти функций-финалистов удалось найти прообразы более сложных неполнораундовых версий.

Ключевые слова: проблема булевой выполнимости, SAT-решатель, Kissat, CBMC, проверка свойств программ, криптографическая хеш-функция, поиск прообраза, конкурс SHA-3.

Благодарности: Исследование О. С. Заикина выполнено при поддержке Математического центра в Академгородке, соглашение с Министерством науки и высшего образования Российской Федерации № 075–15–2022–282.

Для цитирования: Заикин О.С., Давыдов В.В., Кирьянова А.П. Применение алгоритмов решения проблемы булевой выполнимости для анализа финалистов конкурса SHA-3 // Вычислительные методы и программирование. 2024. **25**, № 3. 259–273. doi 10.26089/NumMet.v25r320.

© О. С. Заикин, В. В. Давыдов, А. П. Кирьянова

2024, **25** (3), 259–273. doi 10.26089/NumMet.v25r320

SAT-based analysis of SHA-3 competition finalists

Oleg S. Zaikin
Novosibirsk State University, Novosibirsk, Russia
Matrosov Institute for System Dynamics and Control Theory (IDSTU) SB RAS, Irkutsk, Russia
ORCID: 0000-0002-0145-5010, e-mail: oleg.zaikin@icc.ru

Vadim V. Davydov
Saint Petersburg State University of Aerospace Instrumentation, Saint Petersburg, Russia
QApp, Moscow, Russia
ORCID: 0000-0002-5544-2434, e-mail: vadimdavydov@outlook.com

Anastasia P. Kiryanova
ITMO University, Saint Petersburg, Russia
ORCID: 0009-0006-0344-5111, e-mail: anastaciakosanovskaya@gmail.com

Abstract: SHA-3 competition was held to develop a new standard cryptographic hash funcion. In the present study, finalists of SHA-3 are considered. All of them are still preimage resistant — i.e., it is infeasible to find their outputs given inputs. Preimage resistance of round-reduced versions of the functions is investigated. The corresponding problems are reduced to the Boolean satisfiability problem (SAT) via the CBMC model checker for programs written in C. To solve the constructed SAT instances, the state-of-the-art SAT solver Kissat is applied. Compared to previously published results, for four out of five SHA-3 finalists preimages were found for harder round-reduced versions.

Keywords: Boolean satisfiability problem, SAT solver, Kissat, CBMC, model checking, cryptographic hash function, preimage attack, SHA-3 competition.

Acknowledgements: Oleg S. Zaikin was supported by the Mathematical Center in Akademgorodok under the agreement № 075–15–2022–282 with the Ministry of Science and Higher Education of the Russian Federation.

For citation: O. S. Zaikin, V. V. Davydov, and A. P. Kiryanova, "SAT-based analysis of SHA-3 competition finalists," Numerical Methods and Programming. **25** (3), 259–273 (2024). doi 10.26089/NumMet.v25r320.

---

1. Введение. Хеш-функция по входу произвольной длины генерирует хеш фиксированной длины [1]. Криптографическая хеш-функция должна иметь еще несколько дополнительных свойств, к которым относится стойкость к нахождению прообразов [2]. Согласно этому свойству, должно быть невозможно за реальное время найти вход по известному хешу. В современном цифровом мире криптографические хеш-функции используются очень широко — для хеширования паролей, проверки целостности передаваемых данных и ускорения операций с электронными подписями.

Одним из подходов к анализу стойкости криптографических хеш-функций является построение соответствующей системы нелинейных алгебраических уравнений с последующим применением для решения такой системы разнообразных вычислительных методов. Такой подход называется алгебраическим криптоанализом [3]. Подвидом алгебраического криптоанализа является логический криптоанализ [4], согласно которому формируется пропозициональная булева формула, для которой решается задача булевой выполнимости (SAT). В поисковом варианте SAT состоит в нахождении такого набора значений переменных, при котором формула принимает значение 1. Задача SAT в такой постановке NP-сложна [5], но в последние 25 лет произошел большой прогресс в разработке алгоритмов, которые эффективно справляются с экземплярами SAT, кодирующими задачи из многих практических областей. Ключевым полным алгоритмом решения SAT является CDCL [6].

По-видимому, впервые использовать логический криптоанализ было предложено в 1989 г. [7], но при этом первые практические результаты по его применению к анализу стойкости криптографических хеш-функций были опубликованы только в 2006 г. [8]. С тех пор были проанализированы десятки таких функций, и для некоторых из них логический криптоанализ позволил получить передовые результаты по сравнению с другими подходами.

2024, **25** (3), 259–273. doi 10.26089/NumMet.v25r320

В 2007–2012 гг. Национальным институтом стандартов и технологий США (далее в статье NIST) был проведен конкурс на новый стандарт криптографической хеш-функции SHA-3 [9]. В финал вышло пять функций, из которых был выбран победитель под названием Keccak. При этом остальные четыре финалиста также являются привлекательными объектами для исследований. В 2012 г. вышла статья [10], в которой для анализа стойкости всех пяти финалистов SHA-3 был применен логический криптоанализ.

Так же как и в [10], в настоящем исследовании анализируются стойкость к нахождению прообразов неполнораундовых версий финалистов SHA-3. В отличие от [10] для создания SAT-кодировок применено инструментальное программное средство CBMC, а для решения полученных SAT-задач — решатель Kissat. CBMC в первую очередь предназначено для тестирования корректности программ на языке C, но оно также удобно для кодирования задач криптоанализа. В результате проведенных экспериментов для большинства финалистов удалось найти прообразы более сложных неполнораундовых вариантов по сравнению с [10].

Статья имеет следующую структуру. Во втором разделе даны базовые сведения о криптографических хеш-функциях и конкурсе SHA-3. В третьем разделе описаны финалисты конкурса SHA-3. Четвертый раздел формулирует задачу SAT и кратко описывает алгоритм CDCL. Новые SAT-кодировки финалистов конкурса SHA-3 предлагаются в пятом разделе. Шестой раздел содержит результаты вычислительных экспериментов, в ходе которых были найдены прообразы неполнораундовых версий финалистов SHA-3.

2. Криптографические хеш-функции и конкурс SHA-3. В данном разделе приведены базовые понятия, относящиеся к криптографическим хеш-функциям, а также описан конкурс SHA-3.

2.1. Криптографические хеш-функции. Пусть задан алфавит $\mathscr{A}$. Хеш-функция — это функция h [11]:

$$h:\ \mathscr{A}^* \to \mathscr{A}^n,\ n \in \mathbb{N}.$$

Такая функция трансформирует входную строку любой длины в выходную строку определенной заданной длины. Функция f называется однонаправленной, если вычислительно сложно найти прообраз x такой, что $f(x) = y$ для заданного образа y. В качестве основных компонентов большинства криптографических хеш-функций используются функции сжатия. Функция сжатия g преобразует строку длины m, принадлежащую алфавиту $\mathscr{A}$, в строку меньшей длины n, также принадлежащую алфавиту $\mathscr{A}$:

$$g:\ \mathscr{A}^m \to \mathscr{A}^n,\ n, m \in \mathbb{N},\ m > n.$$

Сами по себе хеш-функции можно разделить на два основных типа — не криптографические и криптографические. В первом случае такие функции возможно использовать для свертки данных. Например, их можно использовать как функцию индексации значений. Криптографические хеш-функции должны обладать определенными свойствами [12], которые будут гарантировать их безопасность. Этими свойствами являются:

1. Стойкость к нахождению прообраза (нахождению первого прообраза): по полученному значению хеш-функции $H = h(m)$ вычислительно сложно найти m.
2. Стойкость к нахождению второго прообраза: при заданном сообщении m_1 вычислительно сложно найти другое сообщение m_2 такое, что $h(m_1) = h(m_2)$.
3. Стойкость к нахождению коллизий: вычислительно сложно найти произвольную пару различных сообщений m_1, m_2 такую, что $h(m_1) = h(m_2)$. Однако в силу того, что область определения хеш-функции больше области ее значений, коллизии для хеш-функций обязательно существуют.

Одним из требований к хорошей хеш-функции является наличие лавинного эффекта [13] — изменение хотя бы одного бита открытого текста ведет к “лавинному” изменению множества битов шифртекста. Подобного эффекта можно добиться, используя в основе хеш-функций блочные шифры с хорошо выраженными свойствами запутывания (confusion) и рассеивания (diffusion) [14]. В отличие от методов сжатия файлов, хеш-функции не нацелены на сохранение входной информации (что подразумевало бы хотя бы частичную обратимость). Напротив, они стремятся вести себя как случайные функции, тем самым устраняя любую структуру в своих входных значениях.

Криптографические хеш-функции играют важную роль в современной криптографии в целом. Например, при постановке электронной подписи для документа большого размера гораздо эффективнее с точки зрения вычислений и используемой памяти сначала получить хеш-значение от данных, а затем поставить подпись, где в качестве сообщения взять это хеш-значение фиксированной длины. В таком случае

2024, **25** (3), 259–273. doi 10.26089/NumMet.v25r320

необходимо использовать криптографическую хеш-функцию, поэтому важно анализировать ее стойкость, чтобы впоследствии электронная подпись не могла быть подделана.

Хеш-функции активно применяются при безопасном хранении паролей в базах данных. Однако поскольку хеш-функция не содержит в своей структуре элемент случайности, для предотвращения различных атак (например, перебор по словарю) используется так называемая криптографическая соль. Под солью понимаются некоторые случайные данные, которые добавляются к входному сообщению, тем самым делая каждый хешированный пароль уникальным. Также можно использовать хеш-функции, к примеру, для реализации доказательств с нулевым разглашением, когда доказывающему необходимо убедить проверяющего в знании какой-либо информации без раскрытия ее содержания, для построения схем обязательств, аутентификации, при реализации дистанционно-ограниченных протоколов.

Стойкость криптографических хеш-функций определяется выполнением трех свойств, указанных выше. В [15] и [16] была предложена схема Меркла–Дамгарда (сокращенно MD), согласно которой стойкая к нахождению коллизий криптографическая хеш-функция строится на основе итеративного вызова стойкой к поиску коллизий односторонней функции сжатия. На рис. 1 представлено изображение конструкции MD, где IV — это вектор инициализации (начальное значение, с которого начинаются все дальнейшие вычисления), padding (дополнение) — добавление каких-либо данных (обычно нулей) ко входному значению так, чтобы длина полученного массива была кратна числу блоков, на которое будет разбито сообщение, f — итеративная функция обработки данных, finalization — заключительный этап обработки полученного значения для уменьшения длины выходного хеша. Режим итерации Меркла–Дамгарда состоит из двух этапов: сначала предварительная обработка данных для хеширования (этап дополнения), а затем фактическая обработка данных. Существуют и другие схемы построения криптографических хеш-функций, однако односторонняя функция сжатия остается одним из главных компонентов таких построений. В связи с этим во многих работах анализируется стойкость именно функций сжатия.

Необходимо не забывать о криптоанализе. Он служит для создания более стойких хеш-функций, а также для разработки новых, более эффективных способов криптоанализа хеш-функций, блочных и поточных шифров и других структур. Для идеальной хеш-функции с размером выхода n бит на поиск первого или второго прообраза требуется около 2^n операций, а самый быстрый способ найти коллизию — это атака “дней рождения”, для проведения которой необходимо примерно $2^{n/2}$ операций [17].

Существуют различные атаки, направленные на поиск коллизий или прообразов классических хеш-функций: атака удлинением сообщения, атака по открытому тексту, дифференцированная атака, встреча посередине, коллизионная атака, атака перебором по словарю, атака по радужным таблицам, атака на основе адаптивно выбранного сообщения. Но одной из самых универсальных атак является поиск прообраза функции сжатия с помощью логического криптоанализа.

2.2. Конкурс SHA-3. В 1993 г. NIST впервые утвердил криптографическую хеш-функцию SHA-1 в качестве стандарта для хеширования данных. Впоследствии был проведен конкурс SHA-2, а в 2007 г. был анонсирован на сегодняшний день последний конкурс NIST на разработку новой криптографической хеш-функции SHA-3 [18]. Конкурс завершился в 2012 г., а новым и актуальным стандартом хеширования стал алгоритм Keccak [19]. Помимо него в финал вышли еще четыре хеш-функции: BLAKE [20], Grøstl [21], Skein [22] и JH [23]. Важным отличием от конкурса SHA-2 являлась необходимость поддержки длин хеша 224, 256, 384 и 512 бит. Исследователям необходимо было представить детальное описание хеш-функции и ее оптимизированные реализации для 32- и 64-разрядных систем, а также оценить производительность [24].

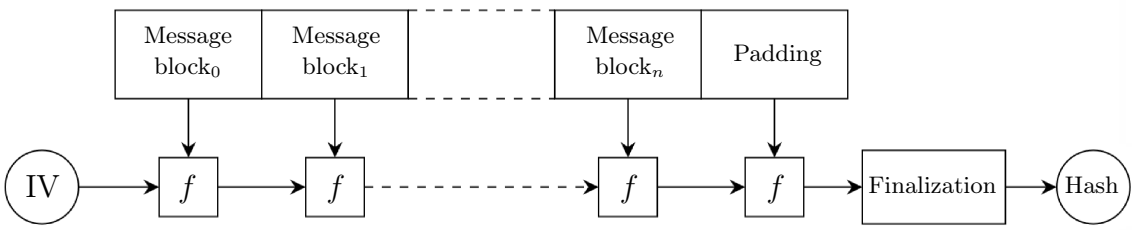

Рис. 1. Структура Меркла–Дамгарда

Fig. 1. Merkle–Damgård construction

2024, **25** (3), 259–273. doi 10.26089/NumMet.v25r320

При выборе из 14 представленных хеш-функций на отбор в третий финальный раунд влияли следующие критерии [25].

- Безопасность: в некоторых случаях NIST не выбирал алгоритмы с исключительной производительностью, главным образом потому, что иногда они не казались абсолютно безопасными, хотя не было известно ни о какой явной атаке на весь алгоритм.
- Производительность: несколько алгоритмов были исключены из-за очень большого объема занимаемой памяти и сложностей при реализации на устройствах с ограниченными ресурсами.
- Криптоанализ: NIST исключил несколько алгоритмов из-за масштабов их доработок на втором этапе или из-за относительного отсутствия сообщений о криптоанализе — и то и другое создавало подозрения, что конструкция, возможно, еще не полностью протестирована и не доработана.
- Разнообразие: для финала NIST выбрал хеш-функции, основанные на разных режимах работы, включая HAIFA, конструкцию губки, а также с различной внутренней структурой, в том числе на основе S-блоков AES, Bit slice и различных комбинаций операций сложения и XOR.

Алгоритм Keccak был выбран в качестве стандарта, так как среди остальных финалистов он оказался самым производительным при аппаратной реализации, а его структура была лаконична и понятна. Кроме того, в основе Keccak использовалась так называемая функция губки (sponge construction), на тот момент мало распространенная и непопулярная. По этой причине атаки, успешные для алгоритма SHA-2, не могут быть применимы для Keccak. Также алгоритм Keccak являлся очень гибким: в отличие от остальных есть возможность его реализации на миниатюрных встраиваемых устройствах.

3. Финалисты конкурса криптографических хеш-функций для стандарта SHA-3. В данном разделе кратко рассмотрен принцип работы всех финалистов. Основное внимание было уделено функциям сжатия представленных хеш-функций (кроме хеш-функции Keccak), поскольку, как было описано ранее, функция сжатия является основным компонентом при построении криптографически стойкой хеш-функции. Далее при описании каждого из алгоритмов используются авторские обозначения.

3.1. BLAKE. Алгоритм хеширования BLAKE был создан на основе потокового шифра ChaCha [26], измененной конструкции Меркла–Дамгарда — HAIFA [27] и структуры Wide-Pipe [28].

В работе рассматривается версия BLAKE-256. Данный вариант оперирует 32-битными словами и возвращает хеш длиной 256 бит. Алгоритм состоит из трех основных этапов:

1) инициализация начального состояния (параметры можно найти в [20, раздел 3.1.1]);
2) применение функции сжатия;
3) режим итерации.

Функция сжатия состоит из инициализации, функции итерации раунда (общее число раундов — 14) и “финализации” (на данном этапе происходит сложение некоторых значений по модулю два). На первом шаге итерации происходит дополнение длины входного сообщения; полученная строка разбивается на блоки длиной 512 бит, чтобы впоследствии быть обработанными итеративной функцией сжатия. Функция сжатия состоит из восьми функций $G_i(G_0, \ldots, G_7)$, каждая из которых, в свою очередь, состоит из восьми преобразований: a, b, c, d (каждое преобразование выполняется дважды за раунд).

3.2. Grøstl. Основной особенностью итеративной криптографической хеш-функции Grøstl (Groestl) является заимствование у блочного шифра AES [29] структуры перестановок и S-блоков. Алгоритм хеш-функции состоит из следующих шагов:

1) дополнение сообщения M до определенной длины и разбиение его на блоки;
2) инициализация начального состояния [21];
3) применение функции сжатия f;
4) выходное преобразование.

Всего в Grøstl проводится 10 раундов. В функции сжатия f две ℓ-битовые перестановки P и Q, в свою очередь, имеют раунды `AddRoundConstant`, `SubBytes`, `ShiftBytes` и `MixBytes`. Основное внимание уделяется раунду `SubBytes`, который можно разделить на 10 подраундов `RND512P` и 10 подраундов `RND512Q`. Каждый из этих подраундов можно разделить еще на 8 вызовов функций `COLUMN`.

3.3. JH. Хеш-функция JH использует в основе измененный метод построения AES, обобщенный на большие размеры, чтобы можно было легко построить большой блочный шифр из небольших компонентов. При хешировании входное сообщение дополняется до определенной длины и разделяется на блоки, которые далее последовательно обрабатываются функцией сжатия [23].

2024, **25** (3), 259–273. doi 10.26089/NumMet.v25r320

Функция сжатия F_d содержит в себе биективную функцию E_d (блочный шифр с постоянным ключом), которая строится на обобщенном d-мерном методе построения AES. К d-мерному массиву применяется подстановочно-перестановочная сеть (SPN) и MDS-код. Биективная функция E_d состоит из $6(d-1)$ раундов R_d. Сама раундовая функция R_d подобно AES состоит из трех слоев: S-блоков, линейного преобразования и перестановок P_d. Общее число раундов в хеш-функции JH — 42.

3.4. Keccak. Хеш-функция Keccak [19] основана на функции губки, являющейся итеративной конструкцией. В процессе "впитывания" губки блок сообщения длиной r бит преобразуется во внутреннее состояние губки, после чего к этому состоянию применяется перестановка. Процесс повторяется до тех пор, пока не будут обработаны все блоки сообщения. В процессе "отжатия" губки, пока длина итогового значения меньше заданной длины хеш-функции, к нему добавляется r первых бит внутреннего состояния, а затем все перемешивается. В конце результат обрезается до длины хеша.

Внутри процессов "впитывания" и "отжатия" губки используется функция Keccak-f, которая является функцией перестановок. Существует семь различных функций f, которые зависят от ширины перестановки $b \in \{25, 50, 100, 200, 400, 800, 1600\}$. В настоящей работе рассматривается только вариант f с шириной $b = 1600$, так как он наиболее широко используется на практике. Функция f, в свою очередь, имеет в своем составе пять преобразований-перестановок: θ, ρ, π, χ и ι. Количество раундов n_r в Keccak зависит от b, в рассматриваемом случае $n_r = 24$.

Отдельно стоит отметить, что в Keccak нет функции сжатия, а сама по себе функция f не обладает криптографической стойкостью. Поэтому при анализе данной хеш-функции следует рассматривать всю конструкцию губки целиком.

3.5. Skein. Основной идеей при построении хеш-функции Skein является использование в основе настраиваемого блочного шифра. Это позволяет хешировать данные конфигурации вместе с входным текстом в каждом блоке и делать каждый экземпляр функции сжатия уникальным.

Построение алгоритма Skein базируется на трех основных компонентах: настраиваемый симметричный блочный шифр Threefish, уникальная итерация блоков (UBI) и дополнительная система аргументов. Шифр Threefish имеет 72 раунда, каждые четыре из которых происходит формирование раундового ключа из основного ключа и tweak-значения (дополнительного параметра настройки). Основная стойкость Skein, так же как и почти всех остальных рассмотренных хеш-функций, заключена в функции сжатия, которая строится с помощью уникальной итерации блоков. В основе UBI лежит настраиваемый шифр, используя который, режим цепочки UBI гарантирует, что каждый блок обрабатывается с применением уникального варианта функции сжатия.

В настоящей работе рассмотрена версия Skein-512-256 (число раундов — 72), где на вход подается сообщение, которое сначала дополняется до определенной длины, после разбивается на блоки размером 512 бит, а на выходе получается 256-битный хеш.

4. Задача булевой выполнимости. В данном разделе формулируется задача булевой выполнимости, кратко описывается основной полный алгоритм для ее решения, а также обсуждается применение этого алгоритма для решения задач анализа стойкости криптографических примитивов.

Введем базовые термины, необходимые для определения задачи булевой выполнимости. Литерал — булева переменная либо ее отрицание. Дизъюнктом называется дизъюнкция литералов. Конъюнктивной нормальной формой (КНФ) называется конъюнкция дизъюнктов. Например, булева формула $(x_1 \vee \neg x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$ — это КНФ над двумя булевыми переменными. Выполняющим набором КНФ называется такой набор значений всех ее переменных, что при их подстановке она принимает значение 1. Если у КНФ существует хотя бы один выполняющий набор, то она называется выполнимой, а в противном случае — невыполнимой. Приведенная в качестве примера КНФ является выполнимой, а ее выполняющий набор равен $(x_1 = 1, x_2 = 0)$.

Задача булевой выполнимости (SAT) состоит в том, чтобы по произвольной КНФ определить ее выполнимость. SAT — это задача распознавания, так как у нее всего два возможных ответа: "да" или "нет". Согласно теореме Кука–Левина, SAT является NP-полной задачей [5]. Существует также поисковый вариант задачи SAT: по произвольной КНФ найти ее выполняющий набор либо доказать, что такого набора не существует. Данная задача не принадлежит классу NP, поэтому она не является NP-полной. При этом она NP-сложна, так как к ней за полиномиальное время сводится NP-полный распознавательный вариант SAT.

2024, **25** (3), 259–273. doi 10.26089/NumMet.v25r320

Несмотря на то что оба варианта SAT являются вычислительно сложными, в последние 30 лет были разработаны алгоритмы, позволяющие эффективно решать экземпляры SAT, кодирующие многие важные практические задачи. Основной полный алгоритм решения SAT — это CDCL (Conflict-Driven Clause Learning) [6], представляющий собой поиск с возвратом, который использует память. Схематично CDCL можно представить в виде следующих шагов.

1. С помощью некоторой эвристики выбирается (угадывается) некоторая неозначенная переменная и ее значение. Если все переменные означены, то КНФ выполнима и текущие значения переменных формируют ее выполняющий набор.
2. Производится распространение булевых ограничений: значение угаданной переменной подставляется во все дизъюнкты, в которые она входит. Тем самым могут быть выведены значения других неозначенных переменных. Если в результате для некоторой переменной были выведены противоположные значения, то такая ситуация помечается как конфликт и обрабатывается на шаге 3. Иначе происходит переход на шаг 1.
3. Осуществляется анализ причин конфликта, результатом которых является так называемый “конфликтный дизъюнкт”, который, будучи приписанным к дизъюнктам исходной КНФ, запрещает возникновение аналогичной конфликтной ситуации. Если конфликтный дизъюнкт пуст, то КНФ является невыполнимой. Иначе он добавляется к дизъюнктам КНФ, осуществляется отмена означивания некоторого числа угаданных переменных и происходит переход на шаг 1.

Отметим, что каждый конфликтный дизъюнкт является логическим следствием исходной КНФ и поэтому не изменяет количество выполняющих наборов. Добавление конфликтных дизъюнктов не только сокращает пространство поиска, но и позволяет эффективнее выбирать переменные для угадывания. В настоящей работе в вычислительных экспериментах используется SAT-решатель Kissat [30], который является современной реализацией CDCL, дополненной многочисленными эвристиками. Этот решатель был выбран потому, что в 2020–2023 гг. в ежегодном соревновании SAT-решателей первое место занимал либо он, либо его модификация.

Современные полные SAT-решатели на практике чаще всего применяются для решения задач верификации и планирования. Тем не менее они позволяют решать и задачи анализа стойкости криптографических примитивов. Соответствующий подход называется логическим криптоанализом. Этот вид криптоанализа позволил найти прообразы для рекордно сложных неполнораундовых версий криптографической хеш-функции MD4, предложенной в 1990 г. [31, 32]. В настоящей статье логический криптоанализ применяется для поиска прообразов более современных криптографических хеш-функций, которые вышли в финал конкурса SHA-3 в 2012 г.

5. SAT-кодировки финалистов конкурса SHA-3. В данном разделе описано инструментальное программное средство CBMC, с помощью которого можно сводить к SAT программы на языке C. Затем обсуждается построение с помощью CBMC SAT-кодировок задач поиска прообразов неполнораундовых версий финалистов SHA-3.

5.1. Сведение к SAT при помощи CBMC. CBMC (Bounded Model Checker for C programs) предназначено для проверки свойств программ на языке C [33]. Одним из способов проверки является сведение проверяемого свойства и описанного в программе алгоритма к SAT путем формирования КНФ. При этом часть переменных КНФ соответствует входу алгоритма, другая часть соответствует выходу алгоритма, а остальные переменные являются дополнительными и нужны для преобразований, которые формируют выход по входу. Если подставить произвольный вход алгоритма в КНФ путем присваивания значений входным переменным и запустить на полученной КНФ SAT-решатель, то будет найден выполняющий набор, частью которого будут значения выходных переменных. Именно в этом смысле КНФ реализует соответствующий алгоритм. При этом если присвоить значения только выходным переменным, то сформируется задача поиска входа по известному выходу. Именно в таком режиме CBMC используется далее.

Помимо CBMC, существуют и другие инструментальные программные средства, которые могут быть использованы для сведения алгоритмов к SAT. Например, это может быть сделано с помощью Transalg [34], который работает с описанием алгоритмов на предметно-ориентированном C-подобном языке. При этом CBMC принимает на вход программы на языке C после небольшой предварительной подготовки. Конкретнее, необходимо выполнить следующие шаги.

1. Из набора файлов с расширениями .h и .c сформировать один файл с расширением .c.
2. Переменные, значение которых необходимо найти, нельзя инициализировать.

2024, **25** (3), 259–273. doi 10.26089/NumMet.v25r320

3. Для каждой переменной `var`, которой в КНФ необходимо присвоить значение `value`, добавить в исходный код следующую строку: `__CPROVER_assume(var == value);`.
4. В конце функции `main` добавить строку `__CPROVER_assert(0, "test");`.

В случае сведения к SAT задачи поиска прообраза криптографической хеш-функции пункт 2 необходимо применить к переменным, соответствующим входному сообщению. Пункт 3 применяется к переменным, которые соответствуют известному хешу. Пункт 4 добавляет пустое свойство, которое необходимо проверить. При выполнении этого пункта CBMC сводит к SAT сам алгоритм, описанный на языке C.

5.2. BLAKE. Для анализа криптографической хеш-функции BLAKE была взята официальная реализация на языке C от автора [35], в которой сжатие осуществляется функцией `blake256_compress`. Функция `blake256_compress` по 512-битному входу генерирует 256-битный выход. В каждом из 14 раундов функции сжатия 8 раз вызывается функция G (см. раздел 3.1). В свою очередь, в функции G выполняется 8 операций. Перед вызовом функции сжатия внутренний регистр был проинициализирован в соответствии с реализацией. Путем присваивания значений выходным переменным была поставлена задача поиска прообраза. Для изменения сложности задачи поиска прообраза использовалось разное число раундов, в последнем раунде менялось количество вызовов G, а также в последнем вызове G менялось количество операций.

Для проверки корректности КНФ для двух тестовых входов функции сжатия, 512 нулей и 512 единиц, были получены выходы функции сжатия с помощью исходной реализации на языке C. Затем в CBMC-программе с помощью `__CPROVER_assume` входным и выходным переменным были присвоены соответствующие значения. Полученные КНФ были выполнимы, т.е. проверка корректности была пройдена успешно. Подобные проверки были сделаны и для остальных анализируемых криптографических хеш-функций.

5.3. Grøstl. В случае Grøstl использовалась реализация на языке C из открытого репозитория криптовалюты Monero [36]. Функция сжатия состоит из двух этапов. На первом из них вызывается функция `compress`, которая состоит из 20 подраундов: по 10 `RND512P` и `RND512Q`. В каждом из подраундов 8 раз вызывается функция `COLUMN`. Итого в 10 раундах выполняется 160 вызовов функции `COLUMN` — по 16 на каждый из раундов. Принимая 512-битный вход, функция `compress` выдает 512-битный выход. Следуя статье [10], на втором этапе функции сжатия в качестве выхода берутся первые 256 бит выхода `compress`. Внутренний регистр `chaining` был проинициализирован, а выходным переменным функции сжатия были присвоены значения. В итоге это дало задачу поиска прообраза. Для варьирования сложности были оставлены первый и последний вызовы `RND512P` и аналогично для `RND512Q`. В каждом из четырех подраундов менялось количество вызовов функции `COLUMN`.

5.4. JH. Для анализа криптографической хеш-функции JH была взята ее реализация на языке C с сайта автора [37]. В этой реализации функция сжатия состоит из двух этапов. На первом этапе вызывается функция `F8`, которая смешивает 512-битный блок сообщения с 1024-битным внутренним регистром `H`. На втором этапе в качестве выхода функции сжатия берутся последние 256 бит регистра `H`. Была поставлена задача поиска прообраза путем инициализации регистра `H` и присваивания значений выходным переменным функции сжатия. Для варьирования сложности задачи менялось количество раундов функции сжатия.

5.5. Keccak. Для данной хеш-функции реализация на языке C была взята с открытого репозитория GitHub [38]. Как было отмечено в разделе 3.1, в Keccak нет функции сжатия. Поэтому была проанализирована вся криптографическая хеш-функция, а именно следующие ее этапы: дополнение сообщения `pad101`; впитывание губки `sponge_absorb`; отжатие губки `sponge_squeeze`. Функция отжатия формирует 256-битный хеш. Как и в статье [10], задача поиска прообраза формулировалась следующим образом: на вход функции `pad101` подается 448-битное сообщение, значение которого неизвестно. Необходимо по известному 256-битному хешу найти сформированное 512-битное сообщение.

5.6. Skein. В случае криптографической хеш-функции Skein была взята реализация на языке C, поданная на конкурс SHA-3 [39]. В этой реализации функция сжатия соответствует двум последовательно примененным функциям. Первая из них — `Skein_512_Process_Block`, которая в течение 72 раундов обновляет значение 512-битного промежуточного регистра на основе 512-битного блока сообщения. После этого функция `Skein_Put64_LSB_First` формирует 256-битный выход функции сжатия, получая на вход блок сообщения и промежуточный регистр.

2024, **25** (3), 259–273. doi 10.26089/NumMet.v25r320

Отметим, что задача поиска прообраза хотя бы одного раунда Skein слишком сложна [10]. По этой причине в [10] была рассмотрена задача поиска псевдопрообраза неполнораундовой версии Skein, которая существенно проще задачи поиска прообраза. При такой постановке внутренний регистр не инициализируется. В настоящей работе была поставлена такая же задача, а ее сложность варьировалась путем изменения количества раундов.

6. Поиск прообразов неполнораундовых версий финалистов конкурса SHA-3. В данном разделе описаны результаты использования современного SAT-решателя для поиска прообразов неполнораундовых версий финалистов SHA-3.

6.1. Вычислительная платформа. При проведении экспериментов использовались следующие аппаратные и программные средства:

- компьютер с 12-ядерным процессором AMD Ryzen 3900X;
- инструментальное средство CBMC версии 5.89 [33];
- SAT-решатель Kissat версии 3.0 [30].

На каждой КНФ, построенной с помощью CBMC, запускался Kissat на одном ядре процессора с лимитом времени 24 часа. Все построенные КНФ, а также соответствующие программы для CBMC доступны онлайн[1].

6.2. BLAKE. В работе [10] был найден прообраз одного (первого) из четырнадцати раундов функции сжатия BLAKE-256. Действительно, оказалось, что эта задача решается за долю секунды. Для двух раундов за отведенное время найти прообраз не удалось, поэтому было принято решение разбить второй раунд на несколько частей и найти прообраз для полного первого раунда и частичного второго раунда.

Как было сказано в разделе 3.1, в одном раунде функции сжатия BLAKE вызываются восемь функций G_i, а в каждой из них выполняется восемь последовательных действий. Таким образом, один раунд состоит из 64 действий. В табл. 1 приведено время нахождения прообраза при варьировании числа раундов. Запись "n $m/64$" означает n полных раундов и m из 64 первых действий второго раунда. Например, 1 $8/64$ раунда означает полный первый раунд (т.е. все 8 вызовов функций G_i), а также вызов функции G_0 второго раунда. "Нулевой хеш" означает 256 нулевых бит, а "единичный хеш" — 256 единичных бит. Нахождение прообразов именно этих хешей является общепринятой практикой.

Таблица 1. Время нахождения прообразов неполнораундовых версий функции сжатия BLAKE-256

Table 1. Runtimes of preimage attacks on round-reduced BLAKE-256 compression function

Число раундов Number of rounds	Нулевой хеш Null hash	Единичный хеш Unit hash
1 $8/64$	0.49 с 0.49 s	0.28 с 0.28 s
1 $9/64$	5 ч 4 мин 5 h 4 min	35 мин 35 min.
1 $10/64$	3 ч 25 мин 3 h 25 min	9 ч 26 мин 9 h 26 min
1 $11/64$	7 ч 25 мин 7 h 25 min	42 мин 42 min
1 $12/64$	4 ч 14 мин 4 h 14 min	не решено not solved
1 $13/64$	не решено not solved	не решено not solved

По полученным результатам можно сделать несколько выводов. Во-первых, при добавлении вызова функции G_0 второго раунда задача нахождения прообраза остается очень простой. Во-вторых, при добавлении первого действия из функции G_1, а именно

$$a := a + b + (m_{\sigma_{r \bmod 10}(2i)} \oplus u_{\sigma_{r \bmod 10}(2i+1)}),$$

время решения возрастает в тысячи раз. В таблице это соответствует 1 $9/64$ раунда. Далее при увеличении

[1] https://github.com/olegzaikin/satsha3finalists

2024, **25** (3), 259–273. doi 10.26089/NumMet.v25r320

числа действий до 12 включительно время решения остается примерно на том же уровне, а иногда даже уменьшается. На первый взгляд, это контринтуитивно, но такая же картина наблюдалась в работе [32]. Конкретнее, при выполнении действия, в котором происходит смешивание части внутреннего состояния регистра с входом функции сжатия, происходит скачок сложности задачи поиска прообраза. Если же часть внутреннего состояния регистра смешивается только с другими частями этого же регистра, то такой скачок не наблюдается. В случае BLAKE следующий такой скачок сложности происходит на 1 13/64 раунда. По сравнению с работой [10] удалось продвинуться вперед на 3/16 раунда при нахождении прообраза функции сжатия.

6.3. Grøstl. В табл. 2 приведено время нахождения прообразов функции сжатия Grøstl-256 в зависимости от числа раундов. Запись "x*P*-yCOL_z*Q*-kCOL" означает, что было вызвано x раундовых функций (из 10) для перестановки P, притом в каждой из них было y (из 8) вызовов функции `COLUMN`; z и k имеют аналогичные значения для перестановки Q. Как было сказано в разделе 5.3, значения x и z равны 2 во всех случаях. При этом значения y и k варьировались от 4 до 8, т.е. для каждого из рассмотренных хешей было построено 25 КНФ. Для краткости в таблице присутствуют только основные результаты. Первые пять записей соответствуют $k = 5$. Далее для каждого из значений y, кроме 8, приведена самая сложная из задач (в смысле значения k), решенных хотя бы для одного хеша.

Таблица 2. Время нахождения прообразов неполнораундовых версий функции сжатия Grøstl-256

Table 2. Runtimes of preimage attacks on round-reduced Grøstl-256 compression function

Число раундов Number of rounds	Подраунды Subrounds	Нулевой хеш Null hash	Единичный хеш Unit hash
1 2/16	2*P*-4COL_2*Q*-5COL	12 с 12 s	20 с 20 s
1 4/16	2*P*-5COL_2*Q*-5COL	5 мин 5 min	2 ч 54 мин 2 h 54 min
1 6/16	2*P*-6COL_2*Q*-5COL	7 мин 7 min	3 ч 2 мин 3 h 2 min
1 8/16	2*P*-7COL_2*Q*-5COL	4 мин 4 min	18 ч 18 h
1 10/16	2*P*-8COL_2*Q*-5COL	7 мин 7 min	11 мин 11 min
1 8/16	2*P*-4COL_2*Q*-8COL	47 мин 47 min	37 мин 37 min
1 8/16	2*P*-5COL_2*Q*-7COL	1 ч 37 мин 1 h 37 min	не решено not solved
1 10/16	2*P*-6COL_2*Q*-7COL	20 ч 49 мин 20 h 49 min	не решено not solved
1 10/16	2*P*-7COL_2*Q*-6COL	1 ч 29 мин 1 h 29 min	не решено not solved

Во-первых, из результатов следует, что задачи поиска прообразов единичного хеша в большинстве случаев значительно сложнее, чем задачи для нулевого хеша. Во-вторых, перестановка Q дает больший прирост сложности при увеличении количества вызовов `COLUMN`, чем перестановка P. Также из первых пяти строк следует, что увеличение числа вызовов `COLUMN` для перестановки P в некоторых случаях приводит к уменьшению времени решения. В итоге удалось найти прообразы для 1.625 раунда, при том что в работе [10] это было сделано только для 0.5 раунда.

6.4. JH. В табл. 3 приведено время нахождения прообразов неполнораундовых версий функции сжатия JH-256. В результате удалось найти прообразы для 4 раундов, в то время как в [10] это было сделано только для 2 раундов.

6.5. Keccak. Как было описано в разделе 3.4, в случае Keccak рассматривалась вся криптографическая хеш-функция целиком. Для одного раунда прообразы единичного и нулевого хешей были найдены за 0.02 с, но для двух раундов решатель не смог найти решения даже за 24 ч. Отметим, что в [10] прообраз был найден для двух раундов Keccak.

2024, **25** (3), 259–273. doi 10.26089/NumMet.v25r320

Таблица 3. Время нахождения прообразов неполнораундовых версий функции сжатия JH-256

Table 3. Runtimes of preimage attacks on round-reduced JH-256 compression function

Число раундов Number of rounds	Нулевой хеш, с Null hash, s	Единичный хеш, с Unit hash, s
1	0.17	0.18
2	0.69	0.59
3	2.19	1.86
4	28.54	60.63
5	не решено not solved	не решено not solved

Таблица 4. Время нахождения псевдопрообразов неполнораундовых версий функции сжатия Skein-256

Table 4. Runtimes of pseudo-preimage attacks on round-reduced Skein-256 compression function

Число раундов Number of rounds	Нулевой хеш, с Null hash, s	Единичный хеш, с Unit hash, s
3	0.01	0.01
4	0.03	0.03
5	0.05	0.07
6	0.05	0.02
7	33.67	19.98
8	не решено not solved	не решено not solved

6.6. Skein. В табл. 4 приведено время нахождения псевдопрообразов неполнораундовых версий функции сжатия Skein-256. В результате удалось найти псевдопрообразы для 7 раундов, при этом от 6 до 7 раундов сложность возрастает в сотни раз. Отметим, что в [10] прообразы были найдены максимум для 6 раундов.

6.7. Обсуждение результатов. В табл. 5 полученные результаты сравниваются с теми, которые были опубликованы в статье [10].

По сравнению с работой [10] достигнут прогресс в анализе четырех из пяти рассмотренных криптографических хеш-функций. Это свидетельствует о развитии SAT-решателей и/или средств сведения задач к SAT применительно к задачам криптоанализа из рассмотренного класса. К сожалению, КНФ из

Таблица 5. Результаты анализа неполнораундовых версий финалистов конкурса SHA-3

Table 5. Analisys results of round-reduced SHA-3 finalists

Функция Function	Задача Problem	Обращено раундов Rounds reversed		Всего раундов Total rounds
		[10]	настоящая статья current paper	
Функция сжатия BLAKE-256 BLAKE-256 compression function	поиск прообраза preimage attack	1	**1.1875**	14
Функция сжатия Grøstl-256 Grøstl-256 compression function	поиск прообраза preimage attack	0.5	**1.625**	10
Функция сжатия JH-256 JH-256 compression function	поиск прообраза preimage attack	2	**4**	42
Хеш-функция Keccak-256 Keccak-256 hash function	поиск прообраза preimage attack	**2**	1	24
Функция сжатия Skein-256 Skein-256 compression function	поиск псевдопрообраза pseudo-preimage attack	6	**7**	72

2024, **25** (3), 259–273. doi 10.26089/NumMet.v25r320

статьи [10] недоступны, поэтому мы не смогли запустить на них тот же SAT-решатель, который мы применили в наших экспериментах. Поэтому не представляется возможным наверняка ответить на вопрос, что является главной причиной достижения прогресса. Отметим, что в отличие от работы [10] построенные нами КНФ доступны онлайн.

7. Заключение. В статье исследованы криптографические функции, вышедшие в финал конкурса SHA-3 в 2012 г. Для неполнораундовых версий функции Skein была рассмотрена задача поиска псевдопрообразов, а для остальных функций — задача поиска прообразов. Эти задачи были сведены к SAT при помощи инструментального средства CBMC, предназначенного для тестирования программ на языке C. В вычислительных экспериментах использовался современный SAT-решатель Kissat, основанный на полном алгоритме CDCL. В сравнении с ранее опубликованными результатами, для всех финалистов, кроме Keccak, удалось найти прообразы (или псевдопрообразы) более сложных неполнораундовых версий.

Список литературы

1. *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.* Введение в алгоритмы. М.: Вильямс, 2013.
2. *Алферов А.П., Зубов А.Ю., Кузьмин А.С., Черемушкин А.В.* Основы криптографии. М.: Гелиос АРВ, 2002.
3. *Bard G.V.* Algebraic cryptanalysis. New York: Springer, 2009. doi 10.1007/978-0-387-88757-9.
4. *Massacci F., Marraro L.* Logical cryptanalysis as a SAT problem // J. Autom. Reason. 2000. **24**, N 1–2. 165–203. doi 10.1023/A:1006326723002.
5. *Garey M.R., Johnson D.S.* Computers and intractability: a guide to the theory of NP-completeness. San Francisco: W.H. Freeman, 1979.
6. *Marques-Silva J.P., Sakallah K.A.* GRASP — a new search algorithm for satisfiability // Proc. Int. Conf. on Computer Aided Design, San Jose, USA, November 10–14, 1996. doi 10.1109/ICCAD.1996.569607.
7. *Impagliazzo R., Levin L.A., Luby M.* Pseudo-random generation from one-way functions // Proc. twenty-first annual ACM symposium on Theory of computing. Washington, Seattle, USA, May 14–17, 1989. doi 10.1145/73007.73009.
8. *Mironov I., Zhang L.* Applications of SAT solvers to cryptanalysis of hash functions // Lecture Notes in Computer Science. Vol. 4121, pp. 102–115. Berlin: Springer, 2006. doi 10.1007/11814948_13.
9. First SHA-3 Candidate Conference. https://csrc.nist.gov/events/2009/first-sha-3-candidate-conference. Cited June 21, 2024.
10. *Homsirikamol E., Morawiecki P., Rogawski M., Srebrny M.* Security margin evaluation of SHA-3 contest finalists through SAT-based attacks // Lecture Notes in Computer Science. Vol. 7564, pp. 56–67. Berlin: Springer, 2012. doi 10.1007/978-3-642-33260-9_4.
11. *Buchmann J.A.* Introduction to cryptography. New York: Springer, 2004. doi 10.1007/978-1-4419-9003-7.
12. *Rogaway P., Shrimpton T.* Cryptographic hash-function basics: definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance // Lecture Notes in Computer Science. Vol. 3017, pp. 371–388. Berlin: Springer, 2004. doi 10.1007/978-3-540-25937-4_24.
13. *Feistel H.* Cryptography and computer privacy // Scientific American. 1973. **228**, N 5. 15–23. doi 10.1038/scientificamerican0573-15.
14. *Shannon C.E.* Communication theory of secrecy systems // The Bell System Technical Journal. 1949. **28**, N 4. 656–715. doi 10.1002/j.1538-7305.1949.tb00928.x.
15. *Merkle R.C.* A certified digital signature // Lecture Notes in Computer Science. Vol. 435, pp. 218–238. New York: Springer, 2001. doi 10.1007/0-387-34805-0_21.
16. *Damgård I.B.* A design principle for hash functions // Lecture Notes in Computer Science. Vol. 435, pp. 416–427. New York: Springer, 2001. doi 10.1007/0-387-34805-0_39.
17. *Paar C., Pelzl J.* Hash functions // Understanding Cryptography. Berlin: Springer, 2010. 293–317. doi 10.1007/978-3-642-04101-3_11.
18. *Kayser R.F.* Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family // Federal Register. 2007. Vol. 72, N 212. 62.
19. *Bertoni G., Daemen J., Peeters M., Van Assche G.* Keccak // Lecture Notes in Computer Science. Vol. 7881, pp. 313–314. Berlin: Springer, 2013. doi 10.1007/978-3-642-38348-9_19.
20. *Aumasson J.-P., Meier W., Phan R.C.-W., Henzen L.* The hash function BLAKE. Berlin: Springer, 2014. doi 10.1007/978-3-662-44757-4.

2024, **25** (3), 259–273. doi 10.26089/NumMet.v25r320

21. *Gauravaram P., Knudsen L.R., Matusiewicz K., et al.* Grøstl — a SHA-3 candidate // https://drops.dagstuhl.de/storage/16dagstuhl-seminar-proceedings/dsp-vol09031/DagSemProc.09031.7/DagSemProc.09031.7.pdf. Cited June 21, 2024.
22. *Ferguson N., Lucks S., Schneier B., et al.* The Skein hash function family. Submission to NIST (round 3). 2010.
23. *Wu H.* The hash function JH. Submission to NIST (round 3). 2011.
24. *Preneel B.* The state of hash functions and the NIST SHA-3 competition // Lecture Notes in Computer Science. Vol. 5487, pp. 1–11. Berlin: Springer, 2009. doi 10.1007/978-3-642-01440-6_1.
25. SHA-3 Finalists Announced by NIST. https://web.archive.org/web/20110709093032/http://crypto.junod.info/2010/12/10/sha-3-finalists-announced-by-nist/. Cited June 21, 2024.
26. *Bernstein D.J.* ChaCha, a variant of Salsa20. https://cr.yp.to/chacha/chacha-20080120.pdf. Cited June 21, 2024.
27. *Biham E., Dunkelman O.* A framework for iterative hash functions — HAIFA. Cryptology ePrint Archive, Paper 2007/278. https://eprint.iacr.org/2007/278. Cited June 21, 2024.
28. *Lucks S.* A failure-friendly design principle for hash functions // Lecture Notes in Computer Science. Vol. 3788, pp. 474–494. Berlin: Springer, 2005. doi 10.1007/11593447_26.
29. *Daemen J., Rijmen V.* The design of Rijndael. Berlin: Springer, 2002. doi 10.1007/978-3-662-04722-4.
30. *Biere A., Fleury M.* Gimsatul, IsaSAT and Kissat entering the SAT Competition 2022 // Proc. SAT Competition 2022 – Solver and Benchmark Descriptions. Vol. B-2022-1, pp. 10–11. University of Helsinki, 2022.
31. *Кондратьев В.С., Семенов А.А., Заикин О.С.* Дубликаты конфликтных ограничений в CDCL-выводе и их использование в задачах обращения некоторых криптографических функций // Вычислительные методы и программирование. 2019. **20**, № 1. 54–66. doi 10.26089/NumMet.v20r106.
32. *Zaikin O.* Inverting 43-step MD4 via Cube-and-Conquer // Proc. IJCAI-ECAI, Vienna, Austria, July 23–29, 2022. doi 10.24963/ijcai.2022/263.
33. *Clarke E., Kroening D., Lerda F.* A tool for checking ANSI-C programs // Lecture Notes in Computer Science. Vol. 2988, pp. 168–176. Berlin: Springer, 2004. doi 10.1007/978-3-540-24730-2_15.
34. *Semenov A., Otpuschennikov I., Gribanova I., et al.* Translation of algorithmic descriptions of discrete functions to SAT with applications to cryptanalysis problems // Log. Methods Comput. Sci. 2020. **16**, N 1. 1–29. doi 10.23638/LMCS-16(1:29)2020.
35. BLAKE. https://github.com/veorq/BLAKE.git. Cited June 21, 2024.
36. GRØSTL. https://github.com/monero-project/monero. Cited June 21, 2024.
37. Hash Function JH. https://www3.ntu.edu.sg/home/wuhj/research/jh/. Cited June 21, 2024.
38. SHA-3 Keccak. https://github.com/davidsteinsland/keccak. Cited June 21, 2024.
39. The Skein Hash Function Family https://www.schneier.com/wp-content/uploads/2015/01/skein.zip. Cited June 21, 2024.

Поступила в редакцию
26 декабря 2023 г.

Принята к публикации
12 июня 2024 г.

Информация об авторах

Олег Сергеевич Заикин — к.т.н., ведущий исследователь; 1) Новосибирский государственный университет, ул. Пирогова, 1, 630090, Новосибирск, Российская Федерация; 2) Институт динамики систем и теории управления имени В. М. Матросова СО РАН, ул. Лермонтова, д. 134, 664033, Иркутск, Российская Федерация.

Вадим Валерьевич Давыдов — к.т.н., доцент; 1) Санкт-Петербургский государственный университет аэрокосмического приборостроения, ул. Большая Морская, д. 67, 190000, Санкт-Петербург, Российская Федерация; 2) QApp, Большой бульвар, д. 30, стр. 1, 121205, Москва, Российская Федерация.

Анастасия Павловна Кирьянова — студент; Университет ИТМО, ул. Ломоносова, д. 9, 191002, Санкт-Петербург, Российская Федерация.

2024, **25** (3), 259–273. doi 10.26089/NumMet.v25r320

References

1. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms* (MIT Press, Cambridge, 2001; Williams, Moscow, 2013).
2. A. P. Alferov, A. Yu. Zubov, A. S. Kuz'min, and A. V. Cheremushkin, *Fundamentals of Cryptography* (Gelios ARV, Moscow, 2002) [in Russian].
3. G. V. Bard, *Algebraic Cryptanalysis* (Springer, New York, 2009). doi 10.1007/978-0-387-88757-9.
4. F. Massacci and L. Marraro, "Logical Cryptanalysis as a SAT Problem," J. Autom. Reason. **24** (1–2), 165–203 (2000). doi 10.1023/A:1006326723002.
5. M. R. Garey and D. S. Johnson, *Computers and Intractability: a Guide to the Theory of NP-Completeness* (W. H. Freeman, San Francisco, 1979).
6. J. P. Marques-Silva and K. A. Sakallah, "GRASP — a New Search Algorithm for Satisfiability," in *Proc. Int. Conf. on Computer Aided Design, San Jose, USA, November 10–14, 1996*. doi 10.1109/ICCAD.1996.569607.
7. R. Impagliazzo, L. A. Levin, and M. Luby, "Pseudo-Random Generation from One-Way Functions," in *Proc. Twenty-First Annual ACM Symposium on Theory of Computing, Washington, Seattle, USA, May 14–17, 1989*. doi 10.1145/73007.73009.
8. I. Mironov and L. Zhang, "Applications of SAT Solvers to Cryptanalysis of Hash Functions," in *Lecture Notes in Computer Science* (Springer, Berlin, 2006), Vol. 4121, pp. 102–115. doi 10.1007/11814948_13.
9. First SHA-3 Candidate Conference. https://csrc.nist.gov/events/2009/first-sha-3-candidate-conference. Cited June 21, 2024.
10. E. Homsirikamol, P. Morawiecki, M. Rogawski, and M. Srebrny, "Security Margin Evaluation of SHA-3 Contest Finalists through SAT-Based Attacks," in *Lecture Notes in Computer Science* (Springer, Berlin, 2012), Vol. 7564, pp. 56–67. doi 10.1007/978-3-642-33260-9_4.
11. J. A. Buchmann, *Introduction to Cryptography* (Springer, New York, 2004). doi 10.1007/978-1-4419-9003-7.
12. P. Rogaway and T. Shrimpton, "Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance," in *Lecture Notes in Computer Science* (Springer, Berlin, 2004), Vol. 3017, pp. 371–388. doi 10.1007/978-3-540-25937-4_24.
13. H. Feistel, "Cryptography and Computer Privacy," Sci. Am. **228** (5), 15–23 (1973). doi 10.1038/scientificamerican0573-15.
14. C. E. Shannon, "Communication Theory of Secrecy Systems," Bell Syst. Tech. J. **28** (4), 656–715 (1949). doi 10.1002/j.1538-7305.1949.tb00928.x.
15. R. C. Merkle, "A Certified Digital Signature," in *Lecture Notes in Computer Science* (Springer, New York, 2001), Vol. 435, pp. 218–238. doi 10.1007/0-387-34805-0_21.
16. I. B. Damgård, "A Design Principle for Hash Functions," in *Lecture Notes in Computer Science* (Springer, New York, 2001), Vol. 435, pp. 416–427. doi 10.1007/0-387-34805-0_39.
17. C. Paar and J. Pelzl, "Hash Functions," in *Understanding Cryptography* (Springer, Berlin, 2010), pp. 293–317. doi 10.1007/978-3-642-04101-3_11.
18. R. F. Kayser, "Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family," Federal Register. **72** (212), 62 (2007).
19. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Keccak," in *Lecture Notes in Computer Science* (Springer, Berlin, 2013), Vol. 7881, pp. 313–314. doi 10.1007/978-3-642-38348-9_19.
20. J.-P. Aumasson, W. Meier, R. C.-W. Phan, and L. Henzen, *The Hash Function BLAKE* (Springer, Berlin, 2014). doi 10.1007/978-3-662-44757-4.
21. P. Gauravaram, L. R. Knudsen, K. Matusiewicz, et al., *Grøstl — a SHA-3 Candidate*, https://drops.dagstuhl.de/storage/16dagstuhl-seminar-proceedings/dsp-vol09031/DagSemProc.09031.7/DagSemProc.09031.7.pdf. Cited June 21, 2024.
22. N. Ferguson, S. Lucks, B. Schneier, et al., *The Skein Hash Function Family. Submission to NIST (Round 3)*. 2010.
23. H. Wu, *The Hash Function JH. Submission to NIST (Round 3)*. 2011.
24. B. Preneel, "The State of Hash Functions and the NIST SHA-3 Competition," in *Lecture Notes in Computer Science* (Springer, Berlin, 2009), Vol. 5487, pp. 1–11. doi 10.1007/978-3-642-01440-6_1.
25. SHA-3 Finalists Announced by NIST. https://web.archive.org/web/20110709093032/http://crypto.junod.info/2010/12/10/sha-3-finalists-announced-by-nist/. Cited June 21, 2024.

2024, **25** (3), 259–273. doi 10.26089/NumMet.v25r320

26. D. J. Bernstein, *ChaCha, a Variant of Salsa20*. https://cr.yp.to/chacha/chacha-20080120.pdf. Cited June 21, 2024.
27. E. Biham and O. Dunkelman, "A Framework for Iterative Hash Functions — HAIFA," Cryptology ePrint Archive, Paper 2007/278. https://eprint.iacr.org/2007/278. Cited June 21, 2024.
28. S. Lucks, "A Failure-Friendly Design Principle for Hash Functions," in *Lecture Notes in Computer Science* (Springer, Berlin, 2005), Vol. 3788, pp. 474–494. doi 10.1007/11593447_26.
29. J. Daemen and V. Rijmen, *The Design of Rijndael* (Springer, Berlin, 2002). doi 10.1007/978-3-662-04722-4.
30. A. Biere and M. Fleury, "Gimsatul, IsaSAT and Kissat Entering the SAT Competition 2022," in *Proc. SAT Competition 2022 – Solver and Benchmark Descriptions*, Vol. B-2022-1, pp. 10–11. University of Helsinki, 2022.
31. V. S. Kondratiev, A. A. Semenov, and O. S. Zaikin, "Duplicates of Conflict Clauses in CDCL Derivation and Their Usage to Invert Some Cryptographic Functions," Numerical Methods and Programming. **20** (1), 54–66 (2019). doi 10.26089/NumMet.v20r106.
32. O. Zaikin, "Inverting 43-Step MD4 via Cube-and-Conquer," in *Proc. IJCAI-ECAI, Vienna, Austria, July 23–29, 2022*. doi 10.24963/ijcai.2022/263.
33. E. Clarke, D. Kroening, and F. Lerda, "A Tool for Checking ANSI-C Programs," in *Lecture Notes in Computer Science* (Springer, Berlin, 2004), Vol. 2988, pp. 168–176. doi 10.1007/978-3-540-24730-2_15.
34. A. Semenov, I. Otpuschennikov, I. Gribanova, et al., "Translation of Algorithmic Descriptions of Discrete Functions to SAT with Applications to Cryptanalysis Problems," Log. Methods Comput. Sci. **16** (1), 1–29 (2020). doi 10.23638/LMCS-16(1:29)2020.
35. BLAKE. https://github.com/veorq/BLAKE.git. Cited June 21, 2024.
36. GRØSTL. https://github.com/monero-project/monero. Cited June 21, 2024.
37. Hash Function JH. https://www3.ntu.edu.sg/home/wuhj/research/jh/. Cited June 21, 2024.
38. SHA-3 Keccak. https://github.com/davidsteinsland/keccak. Cited June 21, 2024.
39. The Skein Hash Function Family. https://www.schneier.com/wp-content/uploads/2015/01/skein.zip. Cited June 21, 2024.

Received
December 26, 2023

Accepted for publication
June 12, 2024

Information about the authors

Oleg S. Zaikin — Ph.D., leading researcher; 1) Novosibirsk State University, Pirogova ulitsa, 1, 630090, Novosibirsk, Russia; 2) Matrosov Institute for System Dynamics and Control Theory (IDSTU) SB RAS, Lermontova ulitsa, 134, 664033, Irkutsk, Russia.

Vadim V. Davydov — Ph.D., Assistant Professor; 1) Saint Petersburg State University of Aerospace Instrumentation, Bolshaya Morskaya ulitsa, 67, 190000, Saint Petersburg, Russia; 2) QApp, Bolshoy bul'var, 30, building 1, 121205, Moscow, Russia.

Anastasia P. Kiryanova — student; ITMO University, Lomonosova ulitsa, 9, 191002, Saint Petersburg, Russia.