УДК 004.056
doi: 10.26583/bit.2024.2.05

Владлен Д. Афонин[1], Сергей В. Запечников[2], Игорь А. Простов[3],
[1,2,3]Национальный исследовательский ядерный университет «МИФИ»,
Каширское ш., 31, Москва, 115409, Россия
[1]e-mail: vladlenafonin.university@yandex.ru, https://orcid.org/0009-0003-0255-6571
[2]e-mail: svzapechnikov@mephi.ru, https://orcid.org/0000-0002-7975-6040
[3]e-mail: igorprostoff@gmail.com, https://orcid.org/0009-0007-5240-1585

АНАЛИЗ ВОЗМОЖНОСТЕЙ ПРИМЕНЕНИЯ АЛГОРИТМА ГОСТ 34.11-2018 В СИСТЕМАХ ДОКАЗАТЕЛЬСТВА С НУЛЕВЫМ РАЗГЛАШЕНИЕМ

Аннотация. Доказательства с нулевым разглашением находят все большее применение в решении широкого круга задач в современных распределенных информационных системах. Представляют особый интерес такие направления деятельности, как системы управления цифровыми активами, системы анонимного электронного голосования, а также анонимные удостоверяющие документы. Тем не менее, учитывая нынешние потребности в локализации разрабатываемых продуктов, возникает вопрос соответствия используемых алгоритмов законодательству Российской Федерации, обязывающему разработчиков использовать отечественные криптографические стандарты. Как показывает наш анализ, в литературе уделено недостаточно внимания применимости этих стандартов в системах доказательства с нулевым разглашением. В частности, не проанализированы сложность выработки доказательств, параметры арифметических схем, отсутствует сравнение вычислительной сложности и криптографических свойств с зарубежными аналогами. В статье подробно рассмотрены особенности реализации арифметической схемы для хэш-функции ГОСТ 34.11-2018 на наиболее распространённом для этих задач языке Circom. Разработанный программный код открыт и доступен для использования и модификации. Проанализированы характеристики схемы, время компиляции и генерации свидетельства (witness). Проведено сравнение полученных результатов с другими популярными хэш-функциями: входящей в стандартную библиотеку языка криптографической хэш-функцией SHA256, а также специализированной и оптимизированной для применения в системах доказательств с нулевым разглашением хэш-функцией Poseidon. Результаты показали, что, хотя использование российской хэш-функции возможно, тем не менее, оно неудобно в приложениях, не требующих использования отечественных криптографических стандартов, ввиду существенно большей временной сложности генерации свидетельства и, как следствие, выработки доказательства. Приведены рекомендации по сценариям использования и предложены дальнейшие направления исследований.

Ключевые слова: доказательства с нулевым разглашением, криптография, алгоритмы, распределенные информационные системы, информационная безопасность, хэш функции.
Для цитирования: АФОНИН, Владлен Д.; ЗАПЕЧНИКОВ, Сергей В.; ПРОСТОВ, Игорь А. АНАЛИЗ ВОЗМОЖНОСТЕЙ ПРИМЕНЕНИЯ АЛГОРИТМА ГОСТ 34.11-2018 В СИСТЕМАХ ДОКАЗАТЕЛЬСТВА С НУЛЕВЫМ РАЗГЛАШЕНИЕМ. Безопасность информационных технологий, [S.l.], т. 31, № 2, с. 81–89, 2024. ISSN 2074-7136. URL: https://bit.spels.ru/index.php/bit/article/view/1643. DOI: http://dx.doi.org/10.26583/bit.2024.2.05.

Vladlen D. Afonin[1], Sergey V. Zapechnikov[2], Igor A. Prostov[3]
[1,2,3]National Research Nuclear University MEPhI (Moscow Engineering Physics Institute),
Kashirskoe sh., 31, Moscow, 115409, Russia
[1]e-mail: vladlenafonin.university@yandex.ru, https://orcid.org/0009-0003-0255-6571
[2]e-mail: svzapechnikov@mephi.ru, https://orcid.org/0000-0002-7975-6040
[3]e-mail: igorprostoff@gmail.com, https://orcid.org/0009-0007-5240-1585

GOST 34.11-2018 Analysis in The Context of Zero-Knowledge Proofs

Abstract. Zero-knowledge proofs are being increasingly applied to a wide range of tasks in modern distributed information systems. Of particular interest are such areas of activity as digital asset management systems, anonymous electronic voting systems, and anonymous credentials. Nevertheless, within the framework of the desire of Russian developers to localize the developed products, there is a question of compliance of the used algorithms with the legislative framework of the Russian Federation, which obliges developers to use local cryptographic standards. As our analysis shows, insufficient attention has been paid in the literature to the applicability of these standards to the scenario of use in zero-knowledge proof systems. In particular, the complexity of proof generation, parameters of arithmetic schemes are not analyzed, there is no comparison of computational complexity and cryptographic properties with foreign alternatives. In this paper we consider in detail the peculiarities of implementation of the arithmetic scheme for the function of GOST 34.11-2018 in the most widespread language for arithmetic circuit programming Circom. The developed program code is open and available for use and modification. The characteristics of the scheme, compilation and generation times are analyzed. The obtained results were compared with other popular hash functions: the cryptographic hash function SHA256 included in the standard language library, and the hash function Poseidon, specialized and optimized for use in zero-knowledge proof systems. The results show that while the use of the Russian hash function is possible, it is not desirable in applications that do not require the use of local cryptographic standards, due to the greater time complexity of witness generation and consequently proof generation. Recommendations on usage scenarios are given and further research directions are suggested.
Keywords: zero-knowledge proofs, cryptography, algorithms, distributed information systems, information security, hash functions.
For citation: AFONIN, Vladlen D.; ZAPECHNIKOV, Sergey V.; PROSTOV, Igor A. GOST 34.11-2018 Analysis in The Context of Zero-Knowledge Proofs. IT Security (Russia), [S.l.], v. 31, no. 2, p. 81–89, 2024. ISSN 2074-7136. URL: https://bit.spels.ru/index.php/bit/article/view/1643. DOI: http://dx.doi.org/10.26583/bit.2024.2.05.

Введение

Доказательства с нулевым разглашением находят все большее применение в решении широкого круга задач в современных распределенных информационных системах. В последние годы доказательства с нулевым разглашением стали использоваться в сферах управления цифровыми активами, анонимных криптовалют, систем анонимного электронного голосования [1], анонимных удостоверяющих документов [2, 3], в области прикладной математики для доказательств математических теорем [4] и многих других.

Современные системы доказательства с нулевым разглашением работают с полиномами над конечными полями с большим числом элементов. Эти полиномы представляют собой закодированные трассировки вычислений арифметических схем. Для доказательства требуемого прикладной задачей утверждения (например, утверждений об обладании секретным ключом либо знании прообраза хэш-функции) оно должно быть представлено в виде арифметической схемы. Такой процесс называется арифметизацией.

Представление произвольных функций в виде арифметических схем представляет собой сложную аналитическую задачу, что привело к созданию множества предметно-ориентированных языков программирования, языков описания аппаратуры, программных библиотек для высокоуровневых языков программирования, а также стандартов описания этих схем.

Распространение этой технологии повлекло за собой разработку множества программных инструментов, предназначенных для создания систем, использующих доказательства с нулевым разглашением. В числе таких программных инструментов – особый язык описания арифметических схем Circom [5], ставший в последнее время стандартом де-факто их низкоуровневого описания. К числу самых употребительных

функций в доказательствах с нулевым разглашением принадлежат криптографические хэш-функции.

Далее в статье рассматривается российская криптографическая хэш-функция ГОСТ 34.11-2018 (также известная под названием «Стрибог») в контексте доказательств с нулевым разглашением. Представлена реализация арифметической схемы, соответствующей функции, на языке программирования схем Circom. Проанализирована сложность генерации доказательства знания прообраза функции, проведено сравнение с популярной хэш-функцией SHA256, а также со специальной функцией Poseidon [6], ориентированной на применение в области доказательств с нулевым разглашением. В частности, проведено сравнение количества связей между элементами арифметических схем и времени их компиляции.

1. Существующие подходы к реализации криптографических хэш-функций в форме арифметических схем

В научной литературе выделяется два основных подхода к разработке арифметических схем хэш-функций:

1. Реализация существующих описанных алгоритмов с адаптацией под ограничения математического аппарата, применяемого в доказательствах с нулевым разглашением (вся арифметика происходит в кольце полиномов над конечными полями).

2. Разработка специализированных алгоритмов, заранее адаптированных для работы в арифметических схемах, использующие операции, не выходящие за рамки указанного математического аппарата. Например, в таких алгоритмах отсутствуют операции XOR-суммирования и битового сдвига, как и любые другие булевы функции.

К алгоритмам первого типа относятся практически все наиболее распространенные хэш-функции: семейства SHA, KECCAK, BLAKE и SHAKE, а также китайский стандарт SM3 [7], что демонстрирует, помимо прочего, научный интерес к анализу государственных стандартов на предмет их применимости в контексте систем доказательства с нулевым разглашением. К функциям же второго типа относятся, например, Monolith [8], Poseidon [6], Tip5 [9] и MiMC [10]. Отметим также, что отдельным направлением исследований является разработка так называемых ZK-адаптированных (ZK-friendly) функций и схем, на основе которых, в свою очередь, возможно построение как хэш-функций, так и других криптографических примитивов [11, 12].

2. Особенности предметно-ориентированного языка Circom

Circom не является языком описания классических программ – он строго направлен на то, чтобы описывать арифметические схемы, в связи с чем имеет ряд особенностей и ограничений.

К первой особенности относится концепция сигналов, которые в арифметических схемах обозначают интерфейс входных и выходных данных некоторой операции. Так как комбинация сигнала и операции описывает некоторое математическое выражение, которое затем переводится в форму полинома, формат любых преобразований над данным, входящими и выходящими через сигналы, также ограничен – они обязаны иметь квадратичную форму вида $s = ax^2 + bx + c$. Именно это и составляет особую сложность для адаптации классических, то есть рассчитанных на цифровую обработку данных, алгоритмов. Такие операции как битовый сдвиг, вычитание и деление, а также любые логические операции (AND, XOR и другие) должны быть описаны в виде квадратичных формул.

Другой особенностью Circom является необходимость для компилятора знать множество параметров схемы уже на этапе компиляции. В частности, такое ограничение приводит к невозможности присваивать значения сигналам в условных операторах и получать доступ до случайного элемента массива в ходе работы программы – так как значение условия либо индекса элемента, к которому происходит обращение, неизвестно на этапе компиляции.

Названные особенности, как и множество других, приводят к необходимости самостоятельной переработки описанных в стандарте алгоритмов в вид, приемлемый для работы Circom, что, на первый взгляд, отдаляет конечное решение от оригинального стандарта – происходит обогащение оригинального алгоритма множеством сторонних функций, позволяющих выполнять широкий диапазон арифметических и логических операций.

3. Описание реализации преобразований

Рассмотрим детали реализации арифметической схемы для функции ГОСТ 34.11-2018 на языке Circom. Основными структурными элементами криптографической хэш-функции являются[1]:

– линейное преобразование,
– нелинейное биективное преобразование,
– сложение по модулю 2,
– перестановка.

Перечисленные структурные элементы последовательно объединены в функцию E. Эта функция составляет собой основную часть раундовой функции алгоритма хэширования g. Ниже приведено описание реализации каждого из перечисленных преобразований.

Нелинейное биективное преобразование. Это преобразование в алгоритме ГОСТ 34.11-2018 представляет собой замену входящих байтов сообщения (блока) на байты, индексы которых в зафиксированной в стандарте таблице замены, соответствуют значению заменяемых байтов.

Собственно замена происходит путем поиска по массиву элемента с необходимым индексом. Так как значение искомого индекса неизвестно на этапе компиляции, при реализации на языке Circom данного преобразования использована следующая схема (рис. 1):

1. При переборе байтов блока сообщения происходит создание массива так называемых компонентов – арифметических схем, созданных из заранее описанных шаблонов. Аналогом компонента в классических языках программирования можно считать функции.
2. Для каждого байта блока сообщения создается свой компонент, осуществляющий преобразование для одного байта.
3. На вход компонента подается значение байта сообщения.
4. В теле компонента происходит перебор всех элементов вектора Pi описанного в стандарте.
5. Каждый из индексов перебираемых элементов проверяется на равенство переданному байту блока сообщения. Проверка равенства, которая могла бы быть описана условным оператором, как было упомянуто выше, невозможна: значение хэшируемого сообщения неизвестно на этапе компиляции. Функция проверки равенства реализована

[1]ГОСТ 34.11-2018. Информационная технология. Криптографическая защита информации. Функция хэширования. URL: https://docs.cntd.ru/document/1200161707 (дата обращения: 18.04.2024).

путем проверки результата вычитания одного значения (байта блока) из другого (индекса) на равенство нулю. Равенство нулю, в свою очередь, проверяется путем нахождения обратного числа по модулю – для нуля такого числа не существует.

6. Все значения элементов вектора *Pi* суммируются, предварительно, будучи умноженными на 1 или 0 в зависимости от того, являлся ли их индекс искомым.

7. Значение суммы возвращается в виде выходного сигнала компонента. Так как все элементы, кроме искомого, были умножены на ноль перед сложением, сумма равняется только искомому элементу.

8. После перебора всех байтов блока массив компонентов заполнен компонентами, содержащими в выходном сигнале значения подставленных байтов.

```
template SubstitutePi() {
    signal input index;
    signal output out;
    var Pi[256] = [...];
    signal intermediate[256];
    for (var i = 0; i < 256; i++) {
        var isIndex = IsEqual()([index, i]);
        intermediate[i] <== isIndex * Pi[i];
    }
    out <== Adder(256)(intermediate);
}
```

Рис. 1. Функция линейного поиска элемента в массиве

Перестановка P. Перестановка байтов осуществляется способом, схожим с нелинейным биективным преобразованием:

1. При переборе индексов от 0 до 63 происходит заполнение массива компонентов, получающих на вход индекс элемента, а на выходе – значение элемента вектора *Tau*.

2. После заполнения массива значениями элементов вектора *Tau* происходит перебор байтов сообщения.

3. На каждой итерации цикла перебора байт происходит передача в компонент перемешивания на вход сигнала с изначальным массивом данных и индексом перемешиваемого элемента в векторе *Tau*, соответствующий индексу текущего элемента.

4. Внутри компонента перемешивания происходит перебор всех индексов элементов входящего сообщения (от 0 до 63) и, в случае равенства с индексом заменяемого элемента, происходит замена перемешиваемого элемента соответствующим элементом из вектора *Tau*.

Операция покомпонентного сложения по модулю 2 (X). Операция побитового сложения чисел по модулю 2 не может быть представлена в квадратичной форме. Поэтому в реализованной схеме данная операция реализована путем последовательного побитового разложения байтов и вызова арифметизированного аналога сложения по модулю два. Для двух бит a и b данную арифметическую схему можно представить как $a + b - 2ab$.

Линейное преобразование L. Первая операция, проводимая при линейном преобразовании, – это преобразование массива входных байт длиной в 64 байта в массив битов длиной 512 бит. Затем происходит перебор полученных бит по 64 за один раз. На каждом этапе цикла происходит проверка равенства бита нулю. После этого происходит

безусловное получение значение 64-битного целого числа, представляющего собой строку из 8 байт матрицы A и умножение этого числа на результат проверки на ноль с последующим побитовым сложением по модулю 2 полученного результата. Данная операция проводится для каждого из 8 наборов по 64 бита. В конце функции полученный путем побитового сложения массив 64-битных чисел переводится в биты и им заполняется выходной массив значений.

Функция расписания ключей. Функция расписания ключей реализована путем вызова множества преобразований *L, P, S, X* над раундовым ключом. Основным элементом функции расписания ключей является функция получения раундового ключа, представляющего собой массив из 64 байт. Массив таких ключей, соответственно, является двумерным массивом. Для получения искомого элемента, являющегося одномерным массивом, двумерного массива требуется провести ряд операций, а именно:

1. Создать массив, размерность которого совпадает с размерностью массива, для которого осуществляется поиск.
2. Основываясь на функции получения элемента по индексу в одномерном массиве, осуществить перебор всех элементов массива.
3. При переборе каждого элемента проверять, является ли его индекс равным искомому. При равенстве присвоить флагу, хранящему информацию о равенстве, значение 1, иначе – значение 0.
4. Провести перебор всех значений элементов текущего перебираемого элемента (одномерного массива).
5. Каждый элемент последовательно умножить на флаг равенства, установленный ранее и присвоить полученное значение соответствующему элементу массива, созданному на шаге 1.
6. Провести суммирование всех элементов, имеющих одинаковый индекс. Элементы массивов, чей индекс не совпал с искомым, будут равны нулю, другие же сохранят своё значение, и сумма всех элементов будет равна им.

После получения необходимого набора байт раундового ключа над ними выполняются ранее описанные операции.

Раундовая функция E. Функция *E*, многократно осуществляющая преобразования *L, P, S* и *X*, описанные ранее, с раундовыми ключами и блоком данных, реализована путем линейного двенадцатикратного вызова ранее описанных функций (рис. 2).

```
component s1 = S();
component p1 = P();
component l1 = L();
component x1 = X();

s1.a <== state;
p1.a <== s1.out;
l1.a <== p1.out;
K_state = Key_schedule()(K, 0);
x1.a <== l1.out;
x1.b <== K;
state = x1.out;
```

Рис. 2 Функция E. Последовательный вызов ранее описанных функций (шаблонов)

Итоговая функция сжатия g. Функция g аналогично реализована путем многократного вызова раундовой функции E и представляет собой итоговую функцию сжатия.

4. Характеристики в сравнении со схемами SHA256 и Poseidon

В данном разделе приведены параметры и замеры времени компиляции арифметической схемы и генерации свидетельства (witness) для разработанной Circom-реализации отечественной хэш-функции ГОСТ 34.11-2018 в сравнении с зарубежным аналогом SHA256, а также специализированной для задач доказательств с нулевым разглашением функции Poseidon (табл. 1). Замеры проведены на Apple MacBook Pro с процессором Apple M1, 16 ГБ ОЗУ. Исходный код реализации открыт и доступен для использования[2].

Таблица 1. Сравнительные характеристики арифметических схем, реализующих криптографические хэш-функции

	ГОСТ 34.11-2018	SHA256	Poseidon
Число нелинейных условий	3396608	30134	321
Число использованных шаблонов	25	102	74
Число связей (проводников) в схеме	3367417	29822	327
Число именованных (неконстантных) связей	17726977	204154	1353
Время компиляции, сек	433	4	1,1
Время генерации свидетельства (witness), сек	12,9	0,2	0,07

5. Направления дальнейших исследований

В качестве одного из направлений дальнейших исследований можно привести проведение более точных замеров времени работы систем неинтерактивных доказательств целиком (не только время генерации свидетельства) с использованием имеющихся современных средств, таких как zk-Harness, как это было сделано в [13] или [14]. Возможно сравнение нескольких реализаций ГОСТ 34.11-2018.

С точки зрения приложений реализация арифметической схемы ГОСТ Р 34.11-2018 представляет интерес сама по себе и может быть проанализирована как на предмет криптографической стойкости, например, методом построения графов зависимостей схем [15], так и на предмет потенциального ускорения работы и уменьшения количества нелинейных условий полученной схемы [16].

Заключение

В данной работе проанализирована применимость российской хэш-функции ГОСТ 34.11-2018 в контексте доказательств с нулевым разглашением. Арифметическая схема хэш-функции реализована на языке Circom. Исходный код реализации открыт и доступен для модификации и использования.

[2]Circom implementation of Streebog (GOST 34.11-2018) hash algorithm. URL: https://github.com/zk-crypto-research/streebog-circom (дата обращения: 18.04.2024).

Разработанное доказательство может применяться в системах, требующих в силу законодательных ограничений применения российских криптографических стандартов. В частности, возможным выглядит их применение в системах дистанционного электронного голосования либо в системах управления отечественными цифровыми финансовыми активами.

Тем не менее, время генерации свидетельства для функции «Стрибог» в ≈ 100 раз превосходит соответствующее время для функции SHA256 и в ≈ 1000 раз превосходит время генерации свидетельства для функции Poseidon, что делает использование функции «Стрибог» неподходящим в широком диапазоне практических задач, принимая во внимание почти всегда существующие ограничения по скорости выполнения программного кода и аппаратным ресурсам. Низкая скорость выработки доказательства делает хэш-функцию практически неконкурентоспособной в сценариях разработки приложений, требующих высокой производительности, таких как распределённый реестр.

СПИСОК ЛИТЕРАТУРЫ:

1. Muth R., Tschorsch F. (2023). Tornado Vote: Anonymous Blockchain-Based Voting. In: 2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). Dubai, United Arab Emirates. DOI: https://doi.org/10.1109/ICBC56567.2023.10174889.
2. Rosenberg M., White J., Garman C., Miers I. (2022). zk-creds: Flexible Anonymous Credentials from zkSNARKs and Existing Identity Infrastructure. In: 2023 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA. DOI: https://doi.org/10.1109/SP46215.2023.10179430.
3. Dutto S., Margaria D., Sanna C., Vesco A. (2022). Toward a Post-Quantum Zero-Knowledge Verifiable Credential System for Self-Sovereign Identity. In: Cryptology ePrint Archive. DOI: http://dx.doi.org/10.13140/RG.2.2.26238.46401.
4. Laufer E., Ozdemir A., Boneh D. (2024). zkPi: Proving Lean Theorems in Zero-Knowledge. In: Cryptology ePrint Archive. URL: https://eprint.iacr.org/2024/267 (дата обращения: 18.04.2024).
5. Bellés-Muñoz, M., Isabel, M., Muñoz-Tapia, J.L., Rubio, A. and Baylina, J. (2022). Circom: A circuit description language for building zero-knowledge applications. In: IEEE Transactions on Dependable and Secure Computing. DOI: 10.1109/TDSC.2022.3232813.
6. Grassi L., Khovratovich D., Rechberger C., Roy A., Schofnegger M. (2019). Poseidon: A New Hash Function for Zero-Knowledge Proof Systems. In: 30th USENIX Security Symposium (USENIX Security 21).
7. Steidtmann C., Gollapudi S. (2023). Benchmarking ZK-Circuits in Circom. In: Cryptology ePrint Archive. URL: https://eprint.iacr.org/2023/681 (дата обращения: 18.04.2024).
8. Lorenzo Grassi et al. (2023). Monolith: Circuit-Friendly Hash Functions with New Nonlinear Layers for Fast and Constant-Time Implementations. In: Cryptology ePrint Archive. URL: https://eprint.iacr.org/2023/1025 (дата обращения: 18.04.2024).
9. Szepieniec A., Lemmens A., Sauer J. F., Threadbare B., Al-Kindi. (2023). The Tip5 Hash Function for Recursive STARKs. In: Cryptology ePrint Archive. URL: https://eprint.iacr.org/2023/107 (дата обращения: 18.04.2024).
10. Albrecht M., Grassi L., Rechberger C., Roy A., Tiessen T. (2016). MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity. In: Cryptology ePrint Archive. URL: https://eprint.iacr.org/2016/492 (дата обращения: 18.04.2024).
11. Ashur T., Dhooghe S. (2018). MARVELlous: a STARK-Friendly Family of Cryptographic Primitives. In: Cryptology ePrint Archive. URL: https://eprint.iacr.org/2018/1098 (дата обращения: 18.04.2024).
12. Bouvier C., Briaud P., Chaidos P., Perrin L., Salen R., Velichkov V., Willems D. (2023). New Design Techniques for Efficient Arithmetization-Oriented Hash Functions: Anemoi Permutations and Jive Compression Mode. In: Crypto 2023 – 43rd International Cryptology Conference, Santa Barbara (CA), United States. DOI: https://doi.org/10.1007/978-3-031-38548-3_17.
13. Yang Y., Shangbin H., Ping X., Yan Z., Zhenyang D., Shengjie H., Shicheng X., Haibin Z. (2022). Implementation and Optimization of Zero-Knowledge Proof Circuit Based on Hash Function SM3. In: Sensors. MDPI, Basel. DOI: https://doi.org/10.3390/s22165951.
14. Ernstberger J., Chaliasos S., Kadianakis G., Steinhorst S., Jovanovic P., Gervais A., Livshits B., Orrù M., (2022). zk-Bench: A Toolset for Comparative Evaluation and Performance Benchmarking of SNARKs. In: Cryptology ePrint Archive. URL: https://eprint.iacr.org/2023/1503 (дата обращения: 18.04.2024).

15. Wen H., Stephens J., Chen Y., Ferles K., Pailoor S., Charbonnet K., Dillig D., Feng Y. (2023). Practical Security Analysis of Zero-Knowledge Proof Circuits. In: IACR Cryptol. ePrint Arch. URL: https://eprint.iacr.org/2023/190 (дата обращения: 18.04.2024).
16. Coglio A., McCarthy E., Smith E. W. (2023). Formal Verification of Zero-Knowledge Circuits. In: Electronic Proceedings in Theoretical Computer Science. DOI: https://doi.org/10.4204/eptcs.393.9.

REFERENCES:

[1] Muth R., Tschorsch F. (2023). Tornado Vote: Anonymous Blockchain-Based Voting. In: 2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). Dubai, United Arab Emirates. DOI: https://doi.org/10.1109/ICBC56567.2023.10174889.
[2] Rosenberg M., White J., Garman C., Miers I. (2022). zk-creds: Flexible Anonymous Credentials from zkSNARKs and Existing Identity Infrastructure. In: 2023 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA. DOI: https://doi.org/10.1109/SP46215.2023.10179430.
[3] Dutto S., Margaria D., Sanna C., Vesco A. (2022). Toward a Post-Quantum Zero-Knowledge Verifiable Credential System for Self-Sovereign Identity. In: Cryptology ePrint Archive. DOI: http://dx.doi.org/10.13140/RG.2.2.26238.46401.
[4] Laufer E., Ozdemir A., Boneh D. (2024). zkPi: Proving Lean Theorems in Zero-Knowledge. In: Cryptology ePrint Archive. URL: https://eprint.iacr.org/2024/267 (accessed: 18.04.2024).
[5] Bellés-Muñoz, M., Isabel, M., Muñoz-Tapia, J.L., Rubio, A. and Baylina, J. (2022). Circom: A circuit description language for building zero-knowledge applications. In: IEEE Transactions on Dependable and Secure Computing. DOI: 10.1109/TDSC.2022.3232813.
[6] Grassi L., Khovratovich D., Rechberger C., Roy A., Schofnegger M. (2019). Poseidon: A New Hash Function for Zero-Knowledge Proof Systems. In: 30th USENIX Security Symposium (USENIX Security 21).
[7] Steidtmann C., Gollapudi S. (2023). Benchmarking ZK-Circuits in Circom. In: Cryptology ePrint Archive. URL: https://eprint.iacr.org/2023/681 (accessed: 18.04.2024).
[8] Lorenzo Grassi et al. (2023). Monolith: Circuit-Friendly Hash Functions with New Nonlinear Layers for Fast and Constant-Time Implementations. In: Cryptology ePrint Archive. URL: https://eprint.iacr.org/2023/1025 (accessed: 18.04.2024).
[9] Szepieniec A., Lemmens A., Sauer J. F., Threadbare B., Al-Kindi. (2023). The Tip5 Hash Function for Recursive STARKs. In: Cryptology ePrint Archive. URL: https://eprint.iacr.org/2023/107 (accessed: 18.04.2024).
[10] Albrecht M., Grassi L., Rechberger C., Roy A., Tiessen T. (2016). MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity. In: Cryptology ePrint Archive. URL: https://eprint.iacr.org/2016/492 (accessed: 18.04.2024).
[11] Ashur T., Dhooghe S. (2018). MARVELlous: a STARK-Friendly Family of Cryptographic Primitives. In: Cryptology ePrint Archive. URL: https://eprint.iacr.org/2018/1098 (accessed: 18.04.2024).
[12] Bouvier C., Briaud P., Chaidos P., Perrin L., Salen R., Velichkov V., Willems D. (2023). New Design Techniques for Efficient Arithmetization-Oriented Hash Functions: Anemoi Permutations and Jive Compression Mode. In: Crypto 2023 – 43rd International Cryptology Conference, Santa Barbara (CA), United States. DOI: https://doi.org/10.1007/978-3-031-38548-3_17.
[13] Yang Y., Shangbin H., Ping X., Yan Z., Zhenyang D., Shengjie H., Shicheng X., Haibin Z. (2022). Implementation and Optimization of Zero-Knowledge Proof Circuit Based on Hash Function SM3. In: Sensors. MDPI, Basel. DOI: https://doi.org/10.3390/s22165951.
[14] Ernstberger J., Chaliasos S., Kadianakis G., Steinhorst S., Jovanovic P., Gervais A., Livshits B., Orrù M., (2022). zk-Bench: A Toolset for Comparative Evaluation and Performance Benchmarking of SNARKs. In: Cryptology ePrint Archive. URL: https://eprint.iacr.org/2023/1503 (accessed: 18.04.2024).
[15] Wen H., Stephens J., Chen Y., Ferles K., Pailoor S., Charbonnet K., Dillig D., Feng Y. (2023). Practical Security Analysis of Zero-Knowledge Proof Circuits. In: IACR Cryptol. ePrint Arch. URL: https://eprint.iacr.org/2023/190 (accessed: 18.04.2024).
[16] Coglio A., McCarthy E., Smith E. W. (2023). Formal Verification of Zero-Knowledge Circuits. In: Electronic Proceedings in Theoretical Computer Science. DOI: https://doi.org/10.4204/eptcs.393.9.

Поступила в редакцию – 18 апреля 2024 г. Окончательный вариант – 29 мая 2024 г.
Received – April 18, 2024. The final version – May 29, 2024.