DOI: 10.15514/ISPRAS–2020–32(4)–8

Протокол сертификации целостности облачных вычислений

[1] *Е.С. Шишкин, ORC-ID: 0000-0002-6221-5651 <evgeny.shishkin@infotecs.ru>*
[1,2] *Е.С. Кислицын, ORC-ID: 0000-0003-0373-9866 <evgeny.kislitsyn@infotecs.ru>*
[1] *ОАО «ИнфоТеКС»,*
127287, Россия, Москва, Старый Петровско-Разумовский проезд, д. 1/23, стр. 1
[2] *Московский государственный университет имени М.В. Ломоносова,*
119991, Россия, Москва, Ленинские горы, д. 1

Аннотация. В статье сформулирована задача сертификации целостности вычислений, проводимых стороной, которой мы не обязательно доверяем. Предложен интерактивный многопользовательский протокол решающий эту задачу при заданных ограничениях. По сравнению с ближайшим аналогом, предложенный протокол упрощает процедуру построения доказательства с *O(nlogn)* до *O(n)*, а сложность коммуникации сводит к одному раунду при сопоставимой длине сертификата.

Ключевые слова: целостность вычислений; интерактивные доказательства; вычислительные сертификаты

Для цитирования: Шишкин Е.С., Кислицын Е.С. Протокол сертификации целостности облачных вычислений. Труды ИСП РАН, том 32, вып. 4, 2020 г., стр. 115–132. DOI: 10.15514/ISPRAS–2020–32(4)–8

Благодарности. Благодарим Андрея Рыбкина и Михаила Бородина за конструктивную критику первой версии протокола: благодаря их замечаниям удалось обнаружить и устранить уязвимость в логике протокола. Мы также благодарим Джэйсона Тетча (Jason Teutsch) за конструктивную критику черновика статьи и за объяснения деталей протокола TrueBit.

Protocol for Certifying Cloud Computations Integrity

[1] *E.S. Shishkin, ORC-ID: 0000-0002-6221-5651 <evgeny.shishkin@infotecs.ru>*
[1, 2] *E.S. Kislitsyn, ORC-ID: 0000-0003-0373-9866 <evgeny.kislitsyn@infotecs.ru>*
[1] *InfoTeCS, Advanced Research Department*
1/23, bld.1, Stariy Petrovsko-Razumovskiy proezd, Moscow, 127287, Russia
[2] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia

Abstract. We define a problem of certifying computation integrity performed by some remote party we do not necessarily trust. We present a multi-party interactive protocol that solves this problem under specified constraints. The protocol rely on a computing entity called weak reliable computing device that is able to produce certificate for a tiny computation. A complex computation of a user can be certified by relying on such weak entity if we translate the user program into an iterative form that converges to the result, a style resembling continuation-passing style programming paradigm. Each iteration of the computation can then be certified by the weak reliable computing device. To ensure the user that computation result was computed fairly and correctly, we put this mechanism into a multi-party incentivized context of several computing providers competing with each other for publishing the result and taking the reward, or finding an error in the published result of other parties. Together with computation result, the computation certificate is published. The certificate is constructed in such a way that other fair parties (auditors) are able to find divergent computation step in

O(logn) steps, where n is a number of computing steps taken before reaching the result. If error is found, the auditor sends proof of the error (called refutation) in the trusted weak computing device that plays the role of autonomous transparent arbiter able to check refutation by replaying a tiny fraction of the computation. Comparing to the nearest related work, our protocol reduces a proof construction complexity from *O(nlogn)* to *O(n)*, turning a communication complexity to exactly one round using a certificate of a comparable length. We also give the formal specification for the protocol and prove some of its security properties in a specified threat-model. Experimental evaluation of the protocol in a model setting is provided.

Keywords: computation integrity, interactive proofs, computation certificates.

For citation: Shishkin E.S., Kislitsyn E.S. Protocol for Certifying Cloud Computations Integrity. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 4, 2020. pp. 115–132 (in Russian). DOI: 10.15514/ISPRAS–2020–32(4)–8

Acknowledgements. We are grateful to Andrey Rybkin and Mikhail Borodin for constructive criticism of the first version of the protocol: thanks to their comments, a vulnerability in the protocol logic was discovered and eliminated. We also thank Jason Teutsch for his constructive criticism of the draft article and for explaining the details of the TrueBit protocol.

1. Введение

Предположим, пользователю требуется произвести какое-то сложное вычисление $C(x)$. Для такого вычисления может потребоваться большой вычислительный ресурс, например, кластер из множества вычислительных узлов, и такого ресурса у пользователя может и не быть.

Для решения своей задачи, пользователь обращается за услугой к облачному сервису – исполнителю вычислений. Пользователь передает исполнителю задачу $C(x)$ и начальные данные d, желая получить результат $C(d)$.

Бывают такие вычисления, результат которых легко валидировать, например, результат функции сортировки списка целых чисел. Факт отсортированности проверяется простой однопроходной функцией и совпадением состава элементов отсортированного и исходного списков.

Напротив, есть вычисления, результат которых в общем случае валидировать трудно без проведения повторного вычисления: например, вычисление, проверяющее *отсутствие* гамильтонова цикла в графе[1]. Если исполнителю удаётся найти хотя бы один гамильтонов цикл, то этот цикл легко проверить. Но что делать, если исполнитель отвечает, что в графе *нет таких циклов*: как проверить достоверность такого ответа?

Возьмем другой пример: автоматическая верификация моделей программ – известная вычислительно сложная задача. Предположим, пользователь передал исполнителю модель программы и список свойств, которым она должна удовлетворять, и исполнитель отвечает пользователю, что в переданной программе найти отклонений от заявленных свойств не удалось. Но как проверить, что: 1) поиск действительно производился 2) был произведен в полном объеме?

В текущей практике пользователь вынужден полагаться на добросовестность исполнителя, на его авторитет, при этом результат проведенного вычисления заверяется цифровой подписью исполнителя.

В этой статье рассматривается распределенный протокол, позволяющий решить описанную задачу технически, без необходимости привнесения дополнительного доверия исполнителю вычисления.

Протокол опирается на наличие доверенного линейного вычислителя, например, смарт-контракта в публичном блокчейне, выполняющего роль авторитетного автономного арбитра в разрешении разногласий по поводу правильности вычисления.

[1] Гамильтонов цикл - цикл в графе, проходящий через каждую вершину ровно один раз.

По сравнению с ближайшим аналогом – протоколом TrueBit [17] – наш протокол существенно уменьшает оценку сложности построения доказательства и минимизирует расходы на коммуникацию сторон в случае разногласий, сводя их количество к одному раунду. Такой эффект достигается за счёт иного механизма конструирования доказательного сертификата и другого способа представления пользовательского вычисления.

Статья устроена следующим образом: в разд. 2 приводится строгая формулировка решаемой задачи. Разд. 3 описывает предлагаемое нами решение в простых терминах. Разд. 4 посвящен описанию метода представления пользовательского вычисления в виде, пригодном для итеративного вычисления. Далее, в разд. 5 приведена формальная запись всего протокола. Разд. 6 посвящен анализу надёжности протокола в рамках заданной модели угроз, кратко обсуждается экономическая составляющая протокола. Разд. 7 описывает возможные атаки на протокол и способы противодействия. Первые экспериментальные результаты приведены в разд. 8. Статью заключает разд. 9, в котором приведены краткие сведения о схожих работах и их технических результатах по сравнению с нашей работой.

2. Формулировка задачи

Мы ищем эффективный способ перепроверки правильности результата вычисления, проведенного исполнителем, которому мы не обязательно доверяем. Вместе с результатом вычисления исполнитель предоставляет некое доказательство - сертификат, размер которого должен разумно коррелировать со сложностью проводимого вычисления. По сертификату мы хотим иметь возможность однозначно установить правдивость полученного результата и сделать это обязательно существенно быстрее, чем повторное вычисление.

Определение 1. (Задача сертификации вычислений) Для произвольной вычислимой в точке $d \in \mathbb{N}$ функции $C\colon \mathbb{N} \to Result$, задать вычислимые функции:

$$proof(C, d) \to Result \times Proofs,$$
$$verify(C, d, r, prf) \to \{True, False\},$$

где $r \in Result$ называют проверяемым результатом вычисления, т.е. $r \stackrel{?}{=} C(d)$, значение $prf \in Proofs$ – доказательство целостности вычисления, называемое также *сертификатом*. При этом выполняются требования:

- значение $verify(C, d, r, prf) = True$ тогда и только тогда, когда $C(d) = r$, и $False$ в противном случае; **[1.1]**
- вычислительная сложность $proof(C, d)$ линейно зависит от сложности C;
- вычислительная сложность $verify$ существенно меньше сложности $proof$ для любой фиксированной задачи C, т.е.

  $$\lim_{n\to\infty} \frac{v_C(n)}{e_C(n)} = 0,$$

  где $v_C(n), e_C(n)$ - асимптотические оценки вычислительной сложности функций $verify$ и $proof$ для задачи C в зависимости от условного размера входа n;
- с ростом сложности $e_C(n)$, размер доказательства prf растёт не быстрее, чем линейно, т.е.

  $$\exists k \forall n\,.\, |prf(n)| \le k \cdot e_C(n).$$

2.1 Вероятностный критерий проверки сертификата

У описанной задачи есть вариации, ослабляющие либо усиливающие некоторые из приведенных требований.

Далее, мы намеренно ослабляем требование **[1.1]**, т.к. оказывается чрезмерно строгим на практике.

Определение 2. (Вероятностный критерий проверки сертификата) Рассмотрим такие события:

$$E_0(\lambda)\colon proof_\lambda(C, d) = \langle r, prf_\lambda \rangle$$
$$E_1(\lambda)\colon verify_\lambda(C, d, r, prf_\lambda) = True$$
$$E_2\colon C(d) = r.$$

Если для функций $proof_\lambda$, $verify_\lambda$ выполняется такой критерий:

$$\lim_{\lambda\to\infty} \Pr[E_0(\lambda) \cdot E_1(\lambda) \cdot E_2] = 1,$$

где λ – это некий параметр метода, величина которого может влиять на вычислительную сложность функций и размер сертификата, в этом случае утверждаем, что алгоритмы удовлетворяют вероятностному критерию проверки сертификата.

Другими словами, мы описываем не детерминированный метод, а вероятностный, где вероятность нужного исхода регулируется специальным параметром.

2.2 Интерактивность

Для некоторых способов решения описанной задачи свойственна *интерактивность*, т.е. процесс проверки решения может происходить в несколько раундов обмена сообщениями между исполнителем и проверяющим. Оценку на количество необходимых раундов мы называем *коммуникационной сложностью* протокола. Интерактивный способ решения описанной задачи также называют *протоколом сертификации целостности вычислений*.

2.3 Доверенный линейный вычислитель

Одна из разновидностей протоколов сертификации вычислений полагается на наличие доверенной стороны, которая хотя и не способна провести *всё вычисление* $C(d)$ *целиком*, но помогает в проверке более простых вычислительных действий.

Например, устройство, способное доказательно выполнить операцию сложения пары натуральных чисел, пусть и не способно посчитать факториал, но если разбить задачу на множество операций сложения, мы бы могли провести всё вычисление достоверным образом.

Определение 3. (Доверенный Линейный Вычислитель) Устройство, решающее задачу сертификации вычислений для вычислимых функций f, таких, что вычислительная сложность $f(n)$ растёт с ростом n не быстрее, чем линейно, т.е. $f(n) \in O(n)$, мы называем *доверенным линейным вычислителем*.

К таким вычислителям относятся, например, безопасные анклавы [12] и смарт-контракты, выполняемые участниками блокчейн-протокола [4].

2.4 Многопользовательский протокол

В классической интерпретации задачи сертификации вычисления подразумевается, что исполнитель вычисления доказывает заказчику вычисления корректность полученного им результата, а заказчик проверяет доказательство, взаимодействие происходит напрямую.

Наша же модель подразумевает иной сценарий взаимодействия: заказчик вычисления и множество исполнителей взаимодействуют посредством устройства доверенного линейного вычислителя (ДЛВ). Несколько исполнителей может выполнять задачу одновременно. Результат вместе с доказательством записывается в ДЛВ, давая возможность другим исполнителям выполнить перепроверку. В случае разногласий относительно результата, исполнители через ДЛВ доказывают ошибочность опубликованного результата. В противном случае, по прошествии периода времени, вычисление считается выполненным корректно, и заказчик получает ответ. Такую модель взаимодействия мы называем *многопользовательским протоколом*.

2.5 Решаемая задача

В этой статье рассматривается многопользовательский протокол сертификации целостности вычислений с вероятностным критерием проверки сертификата, при условии наличия доверенного линейного вычислителя, выполненного с возможностью аутентификации пользователей.

Для простоты изложения *в качестве устройства доверенного линейного вычислителя мы используем смарт-контракт, размещенный в публичном блокчейне*, но можно представить и другие варианты устройств.

3. Схема работы протокола

В этом разделе описывается устройство работы протокола без использования формальных определений.

1. Заказчик преобразует вычислимую функцию $C(x)$ в другую вычислимую функцию $f(x)$, такую, что после n-кратного её применения к точке d, значение сходится к $C(d)$, при этом $f(m) \in O(m)$. Далее будет показано, что такое преобразование всегда можно выполнить.
2. Заказчик публикует функцию f и точку d в смарт-контракте, доступ к которому имеет множество потенциальных исполнителей.
3. Исполнитель проводит вычисление $r = f(f \dots f(d))$, продолжая итерации до тех пор, пока результат не сойдется к неподвижной точке. После каждой итерации $f(x)$, дополнительно вычисляется хэш-значение $c_i \coloneqq H(x \circ c_{i-1})$ Полученный таким образом список значений $cert \coloneqq \langle c_1, \dots, c_n \rangle$ образует *сертификат*. Исполнитель вычисляет слепок сертификата: $hc \coloneqq H(H(cert))$ и *проекцию сертификата* $cp \coloneqq \langle \pi(c_1), \pi(c_2), \dots, \pi(c_n) \rangle$.

   Смысл формирования слепка в том, чтобы, опубликовав это значение, дать возможность другим исполнителям сравнить их сертификат с исходным, но при этом не публиковать само значение в открытом виде.

   Проекция сертификата нужна для того, чтобы быстро (за $\log n$) отыскать ту часть сертификата, с которой начинается разногласие, не публикуя последовательность $cert$ в открытом виде.

   Сам сертификат играет роль секрета, по которому в дальнейшем протокол установит всех добросовестных исполнителей; поэтому это значение не публикуется в открытом виде.
4. Исполнитель публикует решение задачи r, слепок сертификата hc и проекцию cp в смарт-контракте.
5. Другие исполнители, также решавшие задачу, но получившие решение позже, могут перепроверить опубликованное решение сравнив слепок сертификата с тем, которое получилось у них. Эту категорию исполнителей мы называем *проверяющими*.
6. В случае появления разногласий относительно результата, проверяющий отправляет в смарт-контракт номер i той части проекции сертификата, в которой наблюдается расхождение, вместе с точкой r_{i-1} и предыдущим значением c_{i-1}
7. Смарт-контракт по заданным значениям i, r_{i-1}, c_{i-1} убеждается либо в ошибочности опубликованного решения, либо в его верности, выполняя одну итерацию вычисления (детали приведены в разделе **5**).
8. Если в течение некоторого времени для опубликованного решения не предъявлено опровержения, решение считается корректным.
9. Проверяющие отправляют в смарт-контракт значение $H(H(cert) \circ id)$, где id – уникальный идентификатор проверяющего. Это значение играет роль доказательства проделанной работы по перепроверке решения.

10. Исполнитель отправляет в смарт-контракт значение s, такое, что $hc = H(s)$, и по этой информации формируется список добросовестных проверяющих и тех, кто попытался нарушить правила протокола.

4. Итеративная вычислимость

Описываемый протокол требует представления пользовательского вычисления в итеративной форме. Пример на Рис. 1а демонстрирует реализацию функции факториала в стандартной функциональной записи, и на Рис. 1б – в итеративной форме. Если последовательно вычислять значения функции $factFP(\{N, 1\})$, передавая в качестве очередной точки значение, полученное на предыдущем шаге, то функция сойдется к значению $\{0, N!\}$.

```
fact(0) ->                          factFP({0, Acc}) ->
    1;                                  {0, Acc};
fact(N) when N > 0 ->               factFP({N, Acc}) when N > 0 ->
    N * fact(N-1).                      {N - 1, Acc * N}.
```

(a) Стандартная рекурсивная реализация (Recursive implementation) — *(b) Итеративная реализация (Iterative implementation)*

Рис. 1. Варианты записи функции факториала на языке Erlang
Fig. 1. Recursive vs. Iterative factorial implementation (Erlang)

Принципиальная разница между двумя реализациями в том, что первая реализация (Рис.1а) производит сразу всё вычисление, когда вторая (Рис.1б) производит часть вычисления и возвращает промежуточный результат, который можно продолжить вычислять дальше до достижения неподвижной точки.

Можно усомниться в том, что любая программа представима в итеративной форме. Следующая теорема показывает, что это всегда можно сделать, множеством способов.

Теорема 1. Для любой вычислимой в точке d функции $C: \mathbb{N} \to \mathbb{N}$ существуют такие вычислимые функции

$$inj: \mathbb{N} \to T, proj: T \to \mathbb{N}, F: T \to T$$

такие, что

$$proj(\text{fix } F\ inj(d)) = C(d)$$

где fix: $(T \to T) \times \mathbb{N} \to T$ – оператор, вычисляющий неподвижную точку функции, стартуя из заданной точки, т. е.

$$\text{fix } F\ p = F(\text{fix } F\ p)$$

Здесь T – некоторое конечное множество.

Доказательство. Доказательство вынесено в Приложение 1.

5. Описание протокола

Протокол опирается на использование доверенного линейного вычислителя – в данном случае это смарт-контракт в блокчейне – к которому участники осуществляют запросы.

Назовём состоянием смарт-контракта набор переменных определенного типа.

Каждый пользовательский запрос потенциально может изменять состояние смарт-контракта. В описании протокола мы используем понятие текущего состояния переменных и нового измененного состояния переменных, которое получается в процессе обработки запроса.

Используемые обозначения.

$\mathbb{N}_p = \{0 \dots 2^p - 1\}$ – множество натуральных чисел с заданной верхней границей;

X, X' – значение переменной X на момент получения запроса и значение, на момент окончания обработки запроса соответственно.

Обозначения переменных.

$Id \in 2^{\mathbb{N}} \cup \{0\}$ – множество идентификаторов пользователей системы; условимся использовать число 0 для обозначения *неопределённого* идентификатора.

$r \in \mathbb{N}$ – решение задачи,

$s \in \mathbb{N}$ – производное значение от сертификата,

$solver \in Id$ – идентификатор пользователя, первым представившего правильное решение задачи,

$V \in 2^{Id}$ – множество идентификаторов пользователей, представивших доказательство проведенной перепроверки опубликованного сертификата,

$L \in 2^{Id}$ – множество идентификаторов пользователей, осуществлявших попытку компромитировать работу протокола каким-либо способом,

$P \subseteq Id \times \mathbb{N}$ – множество пар – идентификатор и доказательство перепроверки – присланных пользователями, предположительно проводившими перепроверку, до раскрытия секрета, позволяющего установить правдивость перепроверки.

Вход: Пользовательская задача $f: \mathbb{N} \to \mathbb{N}$ и входные данные $d \in \mathbb{N}$.

Выход: $\langle r, s, solver, V, L \rangle$

Предварительные действия.

1. Пользователь задаёт функцию f и начальную точку d. Про требования, накладываемые на функцию f и её взаимосвязи с пользовательской задачей C было сказано ранее.
2. Выбирают криптографически стойкую хэш-функцию $H: \mathbb{N} \to \mathbb{N}_q$, параметр q выбирают по рекомендациям к хэш-функции.
3. Пользователь конструирует вычислительную функцию
   $$F(\langle x, c \rangle) \coloneqq \textbf{\textit{IF}}\ x == f(x)\ \textbf{\textit{THEN}}\ \langle x, c \rangle\ \textbf{\textit{ELSE}}\ \langle f(x), H(x \circ c) \rangle$$
   Здесь $\circ: \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ – какая-либо бинарная операция. Вычисление начинается из точки $\langle d, H(d) \rangle$
4. Выбирают семейство функций $\pi_k: \mathbb{N}^k \to \mathbb{N}_p^k$ для $k \geq 1$, такое, что
   $$\pi_1: \mathbb{N} \to \mathbb{N}_p - \text{какая-либо хэш-функция (не обязана совпадать с } H(x));$$
   $$\pi_k(\langle c_1, \dots, c_k \rangle) \coloneqq \langle \pi_1(c_1), \dots, \pi_1(c_k) \rangle, \text{ для } k > 1.$$
   Далее индекс k мы будем опускать, так как он понятен из контекста – длина кортежа, поданного на вход.
5. В блокчейне публикуется смарт-контракт, выполненный с возможностью:
   - Публиковать заявку на вычисление, указывая функцию F и начальное значение d; заявке назначается статус “опубликована”.
     На каждую новую заявку заводится отдельный набор переменных состояния $\langle r, s, solver, V, L, P \rangle$, изначально пустые.
   - Получать актуальную информацию про заявки: F, d, статус заявки;
     Если заявка находится в статусе “исполнена”, то дополнительно получать значение проекции сертификата cp, решение задачи r и соответствующее ему значение c_n.
     Если заявка находится в статусе “валидирована”, то дополнительно получать набор значений $\langle r, s, solver, V, L \rangle$
   - Для неисполненных заявок, помечать заявку как исполненную, указывая значения r_n, c_n, cp, hc , такие, что:
     $$F(\langle r_n, c_n \rangle) = \langle r_n, c_n \rangle,$$
     $$cp \coloneqq \pi(\langle c_1, \dots, c_n \rangle),$$
     $$hc \coloneqq H(H(\langle c_1, \dots, c_n \rangle)).$$
     Пусть u – идентификатор пользователя, приславшего решение.
     Смарт-контракт проверяет, что предъявленная пара $\langle r_n, c_n \rangle$ является неподвижной

точкой F, в этом случае заявка помечается как “исполненная” и
$$solver' = u \wedge r' = r_n.$$
В случае, если $\langle r_n, c_n \rangle$ не является неподвижной точкой, решение отклоняется, статус заявки не меняется. В этом случае,
$$L' = L \cup \{u\}.$$

- Для исполненных заявок, опровергать предъявленное решение: для опубликованной проекции сертификата cp предъявлять значения i, c_{i-1}, c_i, r_{i-1}, такие, что:
  $$\pi(c_{i-1}) = cp[i-1] \wedge F(\langle r_{i-1}, c_{i-1} \rangle) = \langle r_i, c_i \rangle \wedge \pi(c_i) = cp[i], \quad (5.1)$$
  $$c_{i+1} \coloneqq H(r_i \circ c_i), \quad (5.2)$$
  $$\pi(c_{i+1}) \neq cp[i+1]. \quad (5.3)$$
  Здесь $cp[i]$ – элемент кортежа cp, стоящий на позиции i.
  Пусть u – идентификатор пользователя, приславшего опровержение.
  В случае, если смарт-контракт устанавливает, что предъявленное опровержение – корректное, заявка переводится в статус “опубликована”. В этом случае
  $$L' = L \cup \{solver\} \wedge V' = V \cup \{u\} \wedge solver' = 0.$$
- Для заявок, находящихся в статусе “исполнена”, предъявлять доказательство проверки решения prf, такое, что:
  $$prf := H(H(\langle c_1, \dots, c_n \rangle) \circ id),$$
  где id – уникальный идентификатор предъявляющей стороны.
  При этом
  $$P' = P \cup \{(id, prf)\}.$$
- Для заявок, находящихся в статусе “исполнена”, по прошествии периода времени T, предъявлять секрет s пользователем с идентификатором $solver$, такой что
  $$hc = H(s).$$
  Если проверка не проходит, заявка возвращается в статус “опубликована” и
  $$L' = L \cup \{solver\},$$
  $$solver' = 0.$$
  Иначе статус заявки меняется на “валидирована”. В этом случае
  $$V' = V \cup \{x: (x, p) \in P \wedge H(hc \circ x) = p\},$$
  $$L' = L \cup \{x: (x, p) \in P \wedge H(hc \circ x) \neq p\}.$$

Шаги протокола

1. Пользователь размещает заявку на вычисление в смарт-контракте, указывая F, d.
2. Участники вычитывают из смарт-контракта заявки со статусом “опубликована”, и проводят поиск неподвижной точки функции F, начиная поиск из точки $\langle d, H(d) \rangle$.
3. После получения результата, исполнитель публикует значения r_n, c_n, cp, hc вычисленые по описанным выше правилам в смарт-контракте.
4. Другие участники, выполнявшие вычисление, перепроверяют опубликованный результат, сравнивая значение cp с тем, которое получилось у них.
   - В случае обнаружения несовпадения, отправляют в смарт-контракт опровержение решения.
   - В случае совпавшего решения, отправляют в смарт-контракт доказательство проверки решения.
5. Исполнитель отправляет в смарт-контракт значение s – хэш-значение, посчитанное от сертификата – делая возможным валидировать проверяющих.
   - В случае успешной валидации секрета смарт-контрактом, для данной заявки получен

выход протокола: $\langle r, s, solver, V, L\rangle$

- Иначе заявка возвращается в статус “опубликована” и работа протокола продолжается с пункта 2.

6. Безопасность протокола

Под *нарушителем* мы понимаем участника протокола, желающего нарушить функциональные свойства протокола с какой-либо целью. Под степенью безопасности протокола мы понимаем вероятность наступления нежелательного события, при котором нарушаются функциональные свойства протокола в рамках заданной модели нарушителя.

Определение 4. (Модель нарушителя M_1)

- Для нарушителя, поиск прообраза по заданному образу выбранной хэш-функции $H(x)$ является вычислительно трудной задачей.
- Нарушитель обладает знанием устройства функции $f(x)$ не меньшим, чем заказчик вычисления.

Определение 5. (Угрозы безопасности протокола)

- Событие E_1 : Опровержение корректного решения.
- Событие E_2 : Ложное доказательство перепроверки решения.

Теорема 2. Опровержение опубликованного кем-то корректного решения (событие E_1) является вычислительно трудной задачей для нарушителя из M_1.

Доказательство. Для опровержения опубликованного решения, протокол требует предоставить такие i, c_{i-1}, c_i, r_{i-1}, такие что:

$$\pi(c_{i-1}) = cp[i-1] \wedge \pi(c_i) = cp[i] \wedge H(r_{i-1} \circ c_{i-1}) = c_i \quad (6.1)$$

$$F(\langle r_i, c_i\rangle) = (r_{i+1}, c_{i+1}) \wedge \pi(c_{i+1}) \neq cp[i+1] \quad (6.2)$$

Предположим, что $F: \mathbb{N} \times \mathbb{N}_q$ всюду определена (на практике это далеко не так, но чтобы облегчить работу злоумышленнику, положим). Заметим, что в этом случае выполнить требование (6.1) просто, оно будет выполняться почти для любой точки $\langle r, c\rangle$. Поэтому основная нагрузка ложится на предикат (6.2).

Пусть хэш-функции имеют сигнатуры: $H: \mathbb{N} \to \mathbb{N}_q$, $\pi: \mathbb{N}_q \to \mathbb{N}_p$, и известно, что H является криптографически стойкой, тогда задача нахождения подстановки в предикат (1) сводится к задаче поиска прообразов указанных функций по заданным значениям образов – известная вычислительно сложная задача, которая при выборе оптимальных p, q является трудноосуществимой для нарушителя из M_1.

Теорема 3. Предоставление ложного доказательства перепроверки опубликованного решения (событие E_2) является вычислительно трудной задачей для нарушителя из M_1.

Доказательство. Для доказательства перепроверки опубликованного кем-то решения, протокол требует предоставить такое значение:

$$prf = H(s \circ id)$$

Здесь id – уникальный идентификатор участника, предоставляющего доказательство. На каждый уникальный id допускается публиковать только одно такое prf в качестве доказательства. Из полезной информации, злоумышленнику известны значения:

$$hc = H(s) \quad (3)$$

$$prf = H(s \circ id_i) \quad (4)$$

при этом идентификаторы id_i также известны. Здесь $s = H(\langle c_1, c_2, \dots c_n\rangle)$, т.е. хэш от сертификата вычисления. Вероятность отыскать искомый прообраз для s:

$$P_s = \frac{1}{2^{q \cdot n}}, \text{где } , q \geq 64, n \gg 10^3$$

то есть событие маловероятное. Нарушитель может попробовать отыскать коллизию в (3), т.е. найти s', такое что $H(s') = hc$, но в этом случае, при удачно выбранной операции $\circ$, вероятность одновременного выполнения всех равенств из (4) (для всех id) с найденным s' также будет незначительной.

6.1 Экономические стимулы

Ранее мы обсудили некоторые аспекты криптографической безопасности протокола. Учитывая, что описанный протокол в качестве доверенного линейного вычислителя использует смарт-контракт, выполняемый в публичной блокчейн-сети, есть возможность введения дополнительных мер, существенно повышающих мотивацию рациональных участников следовать протоколу – *экономические стимулы*.

В этой парадигме, стабильность протокола зависит не только от безопасности криптографических механизмов, но и от выбранной системы штрафов и поощрений для участников. Мы не будем подробно останавливаться на этой теме, опишем общую идею возможных вариантов такой системы в виде поправок к описанию протокола.

1. Пользователь размещает заявку на вычисление в смарт-контракте, указывая F, d, снабжая свой запрос криптовалютным депозитом размером D_r.
2. Участники вычитывают задачу и начинают поиск решения.
3. После получения результата кем-то из вычислителей, первый, кто нашел решение, публикует результат, снабжая свое решение криптовалютным депозитом D_s.
4. Другие участники перепроверяют результат.
   - В случае расхождения, публикуют опровержение, снабжая опровержение депозитом D_p.
   - В случае совпадения, отправляют в смарт-контракт доказательство проверки решения, с депозитом D_w.

В конце работы протокола, смарт-контракт имеет на своём счету количество криптовалюты, равное $s = D_r + D_s + m \cdot D_p + n \cdot D_w$, где n – количество участников, доказавших факт перепроверки решения, m – количество раз, когда решение опровергалось. Напомним, что выходом протокола является набор: $\langle r, s, solver, V, L\rangle$, который позволяет перераспределить криптовалютный фонд размера s между исполнителем $solver$ и добросовестными проверяющими V, штрафуя нарушителей из L.

7. Особенности протокола

У предложенного протокола есть несколько мест, которые могут вызвать трудности в практической реализации, а некоторые моменты могут представлять вектора атак для злоумышленников. Ниже мы обсуждаем возможные пути их преодоления.

7.1 Преобразование вычисления в итеративный вид

Как было показано в Теореме 1, вычисление всегда можно привести к итеративному виду, и существует множество способов это сделать.

Например, в работе [17] используется такой подход: пользовательское вычисление компилируется из языка высокого уровня в байткод некоторой виртуальной машины. В качестве одного шага вычисления берется n шагов интерпретатора этого байткода, достигая какого-то состояния, т.е. значений регистров и памяти. Это состояние берется за промежуточный результат вычисления, который подается на вход на следующей итерации, и так далее, до достижения результата.

Мы используем другой подход: сама вычислительная программа из её оригинального вида преобразуется в вид, напоминающий т.н. *стиль передачи продолжений* (Continuation Passing Style) [14], но вместо хвостового вызова, аргумент вместе с указанием какой вызов делать следующим возвращается в качестве вычисленного значения. Такое представление, достигая того же технического результата, может приводить к более компактным представлениям состояний и не требует наличия специального интерпретатора. Существуют способы автоматической трансляции функциональных программ в подобное представление [1].

7.2 Расходящаяся функция F

Представим атаку, при которой в смарт–контракт записывается расходящееся вычисление. В этом случае, исполнители потратят ресурс, но решения не найдут - налицо убыток.

Есть несколько способов преодолеть эту трудность:

- Оформлять вычисление на языке, *гарантирующем* завершаемость. Например, *язык примитивно-рекурсивных функций* даёт такую гарантию и позволяет реализовывать большинство практически интересных алгоритмов [21]. При этом, в смарт-контракт вместе с байткодом функции f передавать ее исходный код P. Вычислитель, компилируя P в f', выполняет проверку $f = f'$. В случае, если проверка успешна, вычислитель убеждается в завершаемости f и принимается за работу.
- Расширить протокол, добавив возможность сертифицировать частичный результат вычислений, т.е. результат, который не сошелся к неподвижной точке, но выполнен корректно с точки зрения сертификата. В случае, если никто из проверяющих не представит более полного (длинного) решения, представленное решение считается удовлетворительным.

7.3 Размер байткода F

Может оказаться так, что размер вычислительной функции или начальных данных слишком велик для размещения в смарт-контракте или ином выбранном ДЛВ. На текущий момент, это принципиальное ограничение рассматриваемого способа: публикуемые F и d должны удовлетворять критерию: $|F| + |d| \leq T_{max}$, где T_{max} – максимальный размер операции для выбранного ДЛВ.

В случае смарт-контрактов, если это условие не выполняется, то такая транзакция отклоняется.

7.4 Размер точки r

В случае разногласий по решению, протокол обязывает предъявить точки c_{i-1}, c_i, r_{i-1}, доказывающие несостоятельность опубликованного решения.

Точка r_{i-1} – это по сути промежуточное состояние вычисления. Может быть так, что размер r_{i-1} становится слишком большим для загрузки в смарт-контракт. На текущий момент, это принципиальное ограничение рассматриваемого способа. Публикуемое вычисление должно удовлетворять критерию: $\forall r, |f(r)| \leq T_{max}$, где T_{max} – максимальный размер операции для выбранного ДЛВ.

7.5 Размер сертификата

Может быть так, что проекция сертификата по размеру не помещается в допустимые размеры операции выбранного ДЛВ. В случае использования смарт-контракта, возможно реализовать хранение проекции во внешнем неизменяемом хранилище, например, в распределенной файловой системе IPFS [2], доступ к ней можно реализовать с помощью технологии оракулов [10].

8. Экспериментальная часть

Для испытания протокола на практике мы запрограммировали логику смарт-контракта и одну пользовательскую задачу.

В силу временных ограничений, мы не стали использовать блокчейн-платформу как среду для испытаний, а реализовали логику смарт-контракта и прикладную задачу для BEAM-машины функционального языка программирования Erlang. [2]

В качестве пользовательской задачи, подаваемой на вход смарт-контракту, мы выбрали задачу UNSAT – доказательство отсутствия набора присваиваний для булевых переменных заданной формулы КНФ, такой, что после подстановки значений в формулу, формула вычисляется в истину. Задача UNSAT сопряжена с известной NP-полной задачей SAT [5], но, в отличии от SAT, её решение в общем случае не может быть проверено за полиномиальное время.

Наш выбор объясняется тем, что алгоритм решения задачи UNSAT имеет прямое практическое применение – верификация моделей программ, закодированных в виде системы КНФ.

Существует множество алгоритмов решения SAT/UNSAT, и известно, что никакой из них не гарантирует время работы меньшее, чем $O(2^n)$, где n – число булевых переменных КНФ, хотя на практике оказываются весьма эффективны.

Мы выбрали один из простейших таких алгоритмов – алгоритм DPLL [6]. Перед передачей его в смарт-контракт, необходимо было преобразовать его в вид, описанный в разд. **4**.

В качестве данных, передаваемых в алгоритм DPLL, мы выбрали задачу доказательства эквивалентности двух программ, реализующих логику очереди FIFO (задача взята из [3]). Задача закодирована в виде КНФ-формулы таким образом, что, в случае, если $DPLL(d) = Unsat$, то это означает их эквивалентность. В противном случае, алгоритм вернет различия двух поведений.

В данном эксперименте, нас интересовало следующее:

- Насколько изменится время вычисления функции f (обозначим за T_1) после представления её в итеративной форме (T_2)?
- Насколько изменится размер байткода функции f (обозначим за S_1) после представления её в итеративной форме (S_2)?
- Каким окажется количество итераций вычисления f до получения неподвижной точки (обозначим за n), и полный размер сертификата (C_f)?
- Каким окажется размер наибольшей промежуточной точки $f^i(d)$ (обозначим за d_{max}), и размер исходных данных d_0?

Эксперименты проводились на серверном узле Intel Xeon Gold 6254 CPU 3.10GHz x 4 Cores, 16GB RAM; среда исполнения: Erlang 20, ERTS 9.3.3.11, под управлением ОС Linux Fedora 29 (виртуальная машина). Результаты приведены в Табл. 1.

Табл. 1. Результаты экспериментов
Table 1. Experimental results.

	d_0	S_1	S_2	T_1	T_2	d_{max}	C_f	n
$QueueInvar_2$	5703	2064	2272	0.1	0.4	42028	52992	1656
$QueueInvar_4$	13707	2064	2272	193	423	170758	13145664	410802

[2] Первоначальные попытки реализовать код прикладного алгоритма на языке Solidity (блокчейн-платформа Ethereum) показали, что это крайне изматывающая деятельность, в силу ограниченной выразительной силы этого языка и ограничений виртуальной машины EVM.

T_1, T_2 измеряется в секундах, $d_0, d_{max}, S_1, S_2, C_f$ – в байтах, n – в штуках
T_1, T_2 are measured in seconds, $d_0, d_{max}, S_1, S_2, C_f$ – in bytes, n – number of iterations

Кроме этого, был реализован тестовый сценарий, проверяющий работу шага опровержения неверного решения, состоящий из таких шагов:

1. Публикуется задача DPLL(x), представленная в итеративной форме, и начальные данные d. Размеры пользовательской программы S_2 и данных d_0 позволяют отправить их в доверенный вычислитель – смарт-контракт.
2. Один из исполнителей намеренно публикует *неверное* решение задачи, состоящее из значений r_n, cp, hc. Размер значения cp может оказаться больше, чем допускает смарт-контракт. В этом случае, содержимое cp публикуется на внешнем распределенном неизменяемом хранилище IPFS, и вместо cp отправляется ссылка на объект в IPFS. При этом, обеспечивать доступность этого объекта – обязанность исполнителя.
3. Другой исполнитель проверяет опубликованное решение и, обнаружив ошибку, конструирует опровержение, отправляя в смарт-контракт значения i, c_{i-1}, c_i, r_{i-1}
4. Смарт-контракт проверяет корректность опровержения, осуществляя проверки (5.1), (5.2), (5.3) из разд. **5**. Если вместо значения cp была отправлена ссылка в IPFS, смарт-контракт запрашивает содержимое этой ссылки по адресу $i-1$ и i через доверенного оракула, получая значения c_{i-1}, c_i. В случае если ссылка недоступна, решение отклоняется.

Мы выложили программные тексты макета протокола и прикладной задачи в открытый доступ [3]. Всё, что касается работы с хранилищем IPFS и оракулом, моделируется с помощью обычного программного кода, без обращений к реальным сервисам.

9. Обзор работ

Верификация результатов вычислений, проведенных третьей стороной – это одна из актуальных задач в теории современной криптографии. Было предложено большое количество разных подходов, каждый со своим набором компромиссов.

Известно семейство способов решения описанной задачи, основанное на применении PCP теоремы для вычислений из класса NP. Напомним, что класс NP – класс задач разрешимости, у которых есть решения, проверяемые за полиномиальное время.

PCP теорема говорит о том, что решение любой задачи в классе NP может быть проверено полиномиальным алгоритмом с помощью фиксированного числа случайно выбранных структурных элементов решения. Более формально, $NP = PCP[O(\log n), O(1)]$, где n – размер входных данных, $\log n$ – количество случайных бит на 1 запрос к решению.

Для способов из этого семейства характерна такая схема взаимодействия сторон: 1) Заказчик оформляет свою задачу в виде булевой схемы (схемы функциональных элементов [22]), по схеме строит многочлен – алгебраическую нормальную форму (АНФ; количество слагаемых в АНФ зависит от количество функциональных элементов в схеме [19]); посылает исполнителю схему и данные для задачи 2) Исполнитель из булевой схемы выводит соответствующую ей булеву функцию, из которой получает АНФ, кодирует выход каждого из узлов схемы в АНФ; далее формирует полученный результат вычисления и сертификат, при этом сертификат может храниться у исполнителя, в облаке, либо передаваться с результатом заказчику 3) Заказчик использует один из подходов для проверки сертификата, при этом ему достаточно случайно выбрать ограниченное количество узлов сертификата для проверки в каждом из подходов.

[3] https://bitbucket.org/unboxed_type/safecomp/src

Известны такие подходы к проверке сертификата: 1) Интерактивный подход без договоренностей заключается в осуществлении запросов к сертификату, находящемуся у исполнителя (или в облаке) произвольных значений узлов – элементов схемы – до тех пор, пока заказчик не удостоверится в правильности вычисления [8] [20] [18] 2) Интерактивный подход с договорённостями заключается в формировании договорённостей посредством криптографических протоколов, проверяющих целостность сертификата. Ключевым отличием от предыдущего подхода является значительное расширение класса проверяемых задач [15] [16] [13].

Табл..2. Асимптотические оценки для некоторых алгоритмов сертификации
Table 2. Asymptotic estimates for several certification algorithms

	$inter(n)$	$proof(n)$	$verify(n)$	$cert(n)$
Libra Non-ZKP	$O(d(n)\cdot \log s(n))$	$O(s(n))$	$O(d(n)\cdot \log s(n))$	$O(d(n)\cdot \log s(n))$
TrueBit	$O(\log n)$	$O(n\log n)$	$O(1)$	$O(n)$
SafeComp	1	$O(n)$	$O(1)$	$O(n)$

$proof(n)$ – сложность построения доказательства, $verify(n)$ – проверка доказательства, $inter(n)$ – количество раундов, $cert(n)$ – размер сертификата, $s(n)$ – размер схемы функциональных элементов в зависимости от размера входа задачи. $d(n)$ – диаметр схемы, т.е. максимальная длина пути в схеме
$proof(n)$ – complexity of proof construction, $verify(n)$ – complexity of proof verification, $inter(n)$ – number of interaction rounds, $cert(n)$ – size of certificate, $s(n)$ – size of functional element circuit as a function of task input size, $d(n)$ – depth of a circuit, i.e. a maximum path length in the circuit

Подходы из семейства PCP накладывают существенные ограничения на проверяемые вычисления. Например, должны быть известны точные верхние оценки на количество итераций циклов и размеры структур данных.

Протокол Libra [20] относится к категории способов, опирающихся на PCP теорему. Он не сопоставим напрямую с нашим протоколом, так как имеет ряд ограничений, например, на структуру пользовательских вычислений. Тем не менее, в некоторых случаях, он может быть применен для получения того же технического результата, поэтому мы привели оценки и для него.

В Табл. 2 сравнивается асимптотика ближайших известных аналогов.

9.1 Сравнение с TrueBit

С появлением блокчейнов стал возможен подход, использующий смарт-контракт в качестве доверенного арбитра для разрешения разногласий, при этом экономические стимулы мотивируют участников выполнять протокол добросовестно.

Пионерской работой в этой области стал протокол TrueBit [17]. Протокол также использует смарт-контракт в качестве доверенного арбитра в случае разногласий исполнителей относительно результата, но имеет существенные отличия от нашего протокола.

Форма вычислительной задачи. В случае с TrueBit, пользовательская задача оформляется в виде байткода некоторой виртуальной машины, при этом интерпретатор такого байткода должен быть размещен в смарт-контракте. Данный подход теоретически позволяет компилировать существующие тексты программ без изменений, однако необходим интерпретатор, реализованный на языке смарт-контрактов блокчейна – механизм довольно сложный, и до сих пор, насколько нам известно, не существует реализаций такого интерпретатора для какой-либо популярной вычислительной архитектуры. В нашем способе, напротив, пользовательское вычисление должно быть представлено в виде итеративной сходящейся к ответу функции, оформленной на существующем языке смарт-контрактов выбранной платформы. Это можно считать неудобством, так как требуется представить

программу в ином виде, но за то делает её пригодной для использования без развертывания дополнительного интерпретатора.

Процедура разрешения разногласий. Для разрешения разногласий протокол TrueBit опирается на интерактивную игру между исполнителями. Игра проходит в несколько раундов: количество раундов оценивается как $O(\log n)$, где n – количество шагов вычисления. На каждом раунде, оба исполнителя-соперника вычисляют какое-то количество корней дерева Меркель: в качестве листьев берутся промежуточные состояния вычисления. Сложность получения одного такого корня имеет оценку $O(n)$. Совокупная сложность построения доказательства, за все раунды, оценивается как $O(n \log n)$. Такое устройство процедуры разрешения разногласий позволяет оптимизировать объем данных, отправляемых в смарт-контракт, однако ценой увеличения совокупной сложности получения доказательства.

В нашем протоколе используется другой подход: исполнитель вместе с решением публикует проекцию сертификата, по которой можно быстро – за $O(\log n)$ отыскать место разногласия и построить доказательство-опровержение. Длина и вычислительная сложность получения сертификата имеют оценку $O(n)$. В случае "длинных" вычислений, для хранения проекции сертификата мы используем внешнее неизменяемое хранилище, например, IPFS. В качестве связующего звена между смарт-контрактом и хранилищем может использоваться доверенный оракул. Таким образом, процедура разрешения разногласий проходит ровно за 1 раунд, понижая оценку сложности с $O(n\log n)$ до $O(n + \log n) = O(n)$, однако ценой записи дополнительных O(n) байт данных в сеть.

10. Заключение

В статье предложен вариант решения задачи сертификации целостности облачных вычислений при заданных ограничениях на вычислительную задачу. Приведено формальное описание протокола, проведен анализ безопасности протокола в рамках описанной модели нарушителя. По сравнению с ближайшим аналогом, наш протокол существенно снижает сложность построения доказательства, а количество раундов коммуникации сводит до одного. Приведены первые экспериментальные результаты, доказывающие, что протокол реализуем на практике с приемлемыми издержками.

В будущих работах по данной теме хотелось бы 1) глубже исследовать свойства протокола, влияющие на его стабильность, например, предложить доказуемо надёжную модель экономических стимулов для участников; 2) реализовать протокол на каком-то публичном блокчейне, и, например, провести верификацию актуальных моделей программ через решение задачи UNSAT.

Список литературы / References

[1] Appel A. W., Jim T. Continuation-passing, closure-passing style. In Proc. of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, 1989, pp. 293-302.

[2] Benet J. Ipfs-content addressed, versioned, p2p file system. arXiv preprint, arXiv:1407.3561, 2014.

[3] Biere A., Cimatti A., Clarke E., Zhu Y. Symbolic model checking without BDDs. Lecture Notes in Computer Science, vol. 1579, 1999, pp. 193-207.

[4] Buterin V. What is Ethereum? Ethereum Official webpage. Available at: http://www.ethdocs.org/en/latest/introduction/what-is-ethereum.html, accessed 14.08.2020.

[5] Cook S. A. The complexity of theorem-proving procedures. In Proc. of the Third Annual ACM Symposium on Theory of Computing, 1971, pp. 151-158.

[6] Davis M., Logemann G., Loveland D. A machine program for theorem-proving. Communications of the ACM, vol. 5, no. 7, 1962, pp. 394-397.

[7] Dershowitz N., Hanna Z., Nadel A. A scalable algorithm for minimal unsatisfiable core extraction. Lecture Notes in Computer Science, vol. 4221, 2006, pp. 36-41.

[8] Goldwasser S., Kalai Y.T., Rothblum G.N. Delegating computation: interactive proofs for muggles. Journal of the ACM, vol. 62, no. 4, 2015, pp. 1-64.

[9] Hopcroft J.E., Motwani R., Ullman J.D. Introduction to Automata Theory, Languages, and Computation: Pearson New International Edition. Pearson, 2013, 496 p.

[10] Kochovski P., Gec S., Stankovski V., Bajec M., Drobintsev P.D. Trust management in a blockchain based fog computing platform with trustless smart oracles. Future Generation Computer Systems, vol. 101, 2019, pp. 747-759.

[11] Luu L., Teutsch J., Kulkarni R., Saxena, P. Demystifying incentives in the consensus computer. In Proc. of the 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015, pp. 706-719.

[12] Mandt T., Solnik M., Wang D. Demystifying the secure enclave processor. Available at: http://mista.nu/research/sep-paper.pdf, accessed 14.08.2020.

[13] Parno B., Howell J., Gentry C., Raykova M. Pinocchio: Nearly practical verifiable computation. In Proc. of the IEEE Symposium on Security and Privacy, 2013, pp. 238-252.

[14] Reynold, J. C. The discoveries of continuations. Lisp and symbolic computation, vol. 6, no. 3-4, 1993, pp. 233-247.

[15] Setty S., Braun B., Vu V., Blumberg A. J., Parno B., Walfish M. Resolving the conflict between generality and plausibility in verified computation. In Proc. of the 8th ACM European Conference on Computer Systems, 2013, pp. 71-84.

[16] Setty S., Vu V., Panpalia N., Braun B., Blumberg A. J., Walfish M. Taking proof-based verified computation a few steps closer to practicality. In Proc. of the 21st USENIX Security Symposium, 2012, pp. 253-268.

[17] Teutsch J., Reitwießner C. A scalable verification solution for blockchains. arXiv preprint arXiv:1908.04756, 2019.

[18] Thaler J. Time-optimal interactive proofs for circuit evaluation. Lecture Notes in Computer Science, vol. 8043, 2013, pp. 71-89.

[19] Vollmer, H. Introduction to circuit complexity: a uniform approach. Springer Science & Business Media, 2013, 272 p.

[20] Xie T., Zhang J., Zhang Y., Papamanthou C., Song D. Libra: Succinct zero-knowledge proofs with optimal prover computation. Lecture Notes in Computer Science, vol. 11694, 2019, pp. 733-764.

[21] Верещагин Н.К., Шень, А. Лекции по математической логике и теории алгоритмов. Часть 3. Вычислимые функции. Учебное пособие. МЦНМО, 2013, 160 стр. / Vereshchagin N.K., Shen A. Lectures on mathematical logic and theory of algorithms. Part 3. Computable functions. MCCME, 2013, 160 p. (in Russian).

[22] Ложкин С. А. Лекции по основам кибернетики. Издательский отдел ф-та ВМиК МГУ, 2004 г., 251 стр. / Lozhkin S.A. Lectures on the basics of cybernetics. Publishing department of the Faculty of Computational Mathematics and Cybernetics, Moscow State University, 2004, 251 p. (in Russian).

Приложение 1

Теорема. Для любой вычислимой в точке d функции $C: \mathbb{N} \to \mathbb{N}$ существуют такие вычислимые функции

$$inj: \mathbb{N} \to T, proj: T \to \mathbb{N}, F: T \to T$$

такие, что

$$proj(\text{fix } F\ inj(d)) = C(d)$$

где fix: $(T \to T) \times \mathbb{N} \to T$ – оператор, вычисляющий неподвижную точку функции, стартуя из заданной точки, т. е.

$$\text{fix}\, F\, p = F(\text{fix}\, F\, p)$$

Здесь T – некоторое конечное множество.

Доказательство. Положим

$$inj(d) = \langle \mathrm{d}, 0 \rangle \qquad proj(\langle x, y \rangle) = x$$

$$F(\langle x, 0 \rangle) = \langle \mathrm{C}(x), 1 \rangle \qquad F(\langle x, 1 \rangle) = \langle x, 1 \rangle$$

Выбранные функции удовлетворяют условиям теоремы. Такое представление F можно назвать *тривиальным* – оно никак не помогает разделить функцию $C(x)$ на составные более

простые части. Можно ли убедиться в существовании представления F, отличного от тривиального? Да, можно.

Согласно тезису Тьюринга, любую вычислимую в точке функцию $C(x)$ можно представить в виде машины Тьюринга. Пусть M – машина, соответствующая функции $C(x)$, т.е.

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, Q_f),$$

$$Q - \text{множество состояний машины}, \Gamma - \text{алфавит символов ленты},$$

$$\Sigma \subseteq \Gamma - \text{входной алфавит}, Q_f \subseteq Q - \text{множество принимающих состояний},$$

$$q_0 \in Q - \text{начальное состояние},$$

$$\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R\} - \text{ функция шага машины}.$$

Тогда функцию F можно определить следующим образом:

$$F(\langle M, I, q, p\rangle) = \begin{cases}\langle M, I', q', p'\rangle, \text{ если } q \notin Q_f \\ \langle M, I, q, p\rangle, \text{ иначе}\end{cases},$$

где

$$(q', X, p') = \delta(q, I_p), \text{ где } I_p - \text{содержимое ленты на позиции } p,$$

$$I' = I[X/p].$$

Делая один шаг из состояния q, находясь на ленте на позиции p, машина M меняет состояние ленты, получая новое состояние ленты I' (заменяя содержимое клетки p на X) и новую позицию головки p'. Если достигнуто принимающее состояние, шагов больше не производится – достигается неподвижная точка F.

По условию теоремы функция C является вычислимой в точке d, поэтому через какое-то количество итераций машина обязательно перейдет в принимающее состояние.

Заметим, что элементы вида $\langle M, I, q, p\rangle$ могут быть закодированы, например, конечным числом натуральных чисел. Пример такой кодировки можно найти в [9].

Сама функция F вычислима по той причине, что опирается на функцию шага $\delta(q, X)$, функцию проверки принадлежности элемета к множеству $q \in Q_f$ и функцию выбора $if - then - else$. Все эти функции вычислимы в случае конечности множеств Q и Γ.

Пусть

$$inj(d) := \langle M, I_0, q_0, 1\rangle,$$

где I_0 – начальное состояние ленты, на которой записано в том числе d.

Функция $proj(\langle M, I, q, p\rangle)$ извлекает из ленты I ответ $C(d)$.

Такие функции можно построить, потому что на ленте всегда возможно зафиксировать область для начального значения и для ответа. Ответ $C(d)$ представим в виде конечного числа клеток ленты, так как вычислимость функции в точке гарантирует конечное число шагов машины.

Если размер области для ответа заранее неизвестен, мы можем зафиксировать на ленте позицию начала, и уникальный маркер, наличие на ленте которого означает конец области, отведенной для ответа.

Описанное представление тройки $F, inj, proj$ удовлетворяют требованиям теоремы, и не является тривиальным представлением.

Информация об авторах / Information about the authors

Евгений Сергеевич ШИШКИН – ведущий исследователь научного отдела. Сфера научных интересов: дедуктивная формальная верификация программ, автоматическая верификация моделей программ, логические дедуктивные системы, языки формальных спецификаций, интерактивные среды построения доказательств, распределенные системы, блокчейны и смарт-контракты.

Evgeny Sergeevich SHISHKIN – senior researcher in the research department. Research interests: deductive formal verification of computer programs, automatic verification of program models, logical deductive systems, formal specification languages, interactive proof assistants, distributed systems and algorithms, blockchains, smart-contracts.

Евгений Сергеевич КИСЛИЦЫН – аспирант кафедры высшей алгебры механико-математического факультета. Научные интересы включают исследование неассоциативных структур, теорию колец, гомоморфное шифрование, формализацию распределенных протоколов.

Evgeny Sergeevich KISLITSYN – Postgraduate student of the Department of Higher Algebra, Faculty of Mechanics and Mathematics. Research interests include the study of non-associative structures, ring theory, homomorphic encryption, and formalization of distributed protocols.