vypiska-iz-trebovanij-po-bezopasnosti-informatsii-utverzhdennykh-prikazom-fstek-rossii-ot-2-iyunya-2020-g-n-76.

7. ГОСТ Р 59453.1-2021 «Защита информации. Формальная модель управления доступом. Ч. 1. Общие положения». М.: Стандартинформ, 2021. 35 с.
8. ГОСТ Р 59453.2-2021 «Защита информации. Формальная модель управления доступом. Ч. 2. Рекомендации по верификация формальной модели управления доступом». М.: Стандартинформ, 2021. 23 с.
9. Операционная система специального назначения Astra Linux Special Edition. https://astralinux.ru/products/astra-linux-special-edition/
10. *Буренин П. В., Девянин П. Н., Лебеденко Е. В. и др.* Безопасность операционной системы специального назначения Astra Linux Special Edition: учеб. пособие для вузов / под ред. П. Н. Девянина. 3-е изд., перераб. и доп. М.: Горячая линия — Телеком, 2019. 404 с.
11. *Девянин П. Н., Ефремов Д. В., Кулямин В. В. и др.* Моделирование и верификация политик безопасности управления доступом в операционных системах. М.: Горячая линия — Телеком, 2019. 214 с.
12. *Leuschel M. and Butler M.* ProB: an automated analysis toolset for the B method // Int. J. Softw. Tools Technol. Transf. 2008. No. 10(2). P. 185–203.
13. *Abrial J.-R. and Hallerstede S.* Refinement, decomposition, and instantiation of discrete models: Application to Event-B // Fundamenta Informaticae. 2007. V. 77. Iss. 1–2. P. 1–28.
14. *Девянин П. Н., Леонова М. А.* Применение подтипов и тотальных функций формального метода Event-B для описания и верификации МРОСЛ ДП-модели // Программная инженерия. 2020. Т. 11. № 4. С. 230–241.
15. *Девянин П. Н., Леонова М. А.* Приёмы по доработке описания модели управления доступом ОССН Astra Linux Special Edition на формализованном языке метода Event-B для обеспечения ее автоматизированной верификации с применением инструментов Rodin и ProB // Прикладная дискретная математика. 2021. № 52. С. 83–96.

УДК 004.75 DOI 10.17223/2226308X/14/28

МЕТОД ОБЕСПЕЧЕНИЯ КОНФИДЕНЦИАЛЬНОСТИ ДАННЫХ НА ОСНОВЕ zk-SNARK[1]

Д. О. Кондырев

Представлен метод обеспечения конфиденциальности данных с возможностью проверки корректности на основе протокола доказательства с нулевым разглашением zk-SNARK. Разработанный метод позволяет создавать алгоритмы на основе zk-SNARK в смарт-контрактах Ethereum, используя высокоуровневые базовые криптографические схемы.

Ключевые слова: *распределённые системы, блокчейн, доказательство с нулевым разглашением, zk-SNARK, платформа Ethereum.*

Среди технических проблем, препятствующих внедрению технологии распределённых реестров, масштабируемость и конфиденциальность являются особенно существенными. В настоящий момент ведутся активные исследования, направленные на поиск решения проблемы конфиденциальности.

[1]Работа выполнена при поддержке Математического центра в Академгородке, соглашение с Министерством науки и высшего образования Российской Федерации № 075-15-2019-1613, и лаборатории криптографии JetBrains Research.

Особенно остро эта проблема встаёт в открытых распределённых реестрах (таких, как блокчейн-системы). В таких реестрах все данные сохраняются в открытом виде и доступны всем участникам, что не всегда приемлемо при создании промышленных программных систем. Кроме того, идентификация пользователей происходит по адресу их аккаунта. Таким образом, существует возможность отслеживать действия пользователя путём анализа транзакций, в которых участвует конкретный адрес, и сопоставления адреса аккаунта и пользователя.

В работе предложен метод обеспечения конфиденциальности данных с возможностью проверки корректности. В основе метода лежит криптографический протокол неинтерактивного доказательства знания с нулевым разглашением zk-SNARK [1]. Данная работа развивает результаты, полученные в [2].

В качестве базовой системы для реализации метода выбрана платформа Ethereum — блокчейн-система общего назначения, поддерживающая смарт-контракты.

В zk-SNARK процедура проверки доказательства состоит из операций на эллиптических кривых. В частности, верификатор требует скалярного умножения и сложения на группе эллиптических кривых, а также вычислительно более сложной операции — билинейного спаривания. Ethereum предоставляет реализацию этих операций в виде предварительно скомпилированных контрактов. С их помощью есть возможность реализовать схемы на основе доказательства с нулевым разглашением в коде смарт-контрактов. Используя только встроенные инструменты, приходится оперировать низкоуровневыми примитивами, что не позволяет реализовать сложные алгоритмы.

Для возможности создавать произвольные криптографические схемы, в основе которых лежит zk-SNARK, были разработаны сторонние инструменты, такие, как ZoKrates [3]. Эти решения позволяют реализовать схему в виде кода на довольно высокоуровневом языке, который затем компилируется в код смарт-контрактов. Однако такой подход имеет ряд ограничений, которые не позволяют применять его для схем произвольного размера и сложности.

Для решения проблем существующих подходов предлагается добавить поддержку более высокоуровневых криптографических примитивов, выраженных в виде систем ограничений ранга 1 (R1CS — rank-1 constraint systems), непосредственно в код Ethereum-клиента. Таким образом, мы добавляем механизм задания произвольных схем непосредственно в коде смарт-контрактов. При таком подходе нет необходимости напрямую использовать операции над эллиптическими кривыми, вместо этого новые алгоритмы строятся как комбинация добавленных примитивов. Кроме того, такой подход оказывается более вычислительно эффективным за счёт реализации непосредственно в Ethereum-клиенте.

В качестве базовых примитивов добавлены схемы, реализующие логические операции (AND, OR, NOT) и операции сравнения. Их реализация выполнена на основе libsnark — криптографической библиотеки с открытым исходным кодом, которая обеспечивает эффективные реализации конструкций zk-SNARK [4].

В Ethereum-клиент добавлены соответствующие операции для возможности вызова этих методов как из кода контрактов, так и вне блокчейна. Для этого модифицирована виртуальная машина Ethereum, куда были добавлены функции создания схемы, генерации доказательства и его верификации.

Схема, описанная разработчиком в коде смарт-контракта, транслируется в набор добавленных примитивов. Далее на их основе формируется система ограничений ранга 1 (R1CS), с которой работают алгоритмы генерации и верификации доказательства zk-SNARK.

Для каждой новой схемы необходима генерация новой пары ключей доказательства и верификации. Их генерация выполняется вне блокчейна, поскольку в алгоритме генерации используется параметр безопасности, зная который, можно создавать некорректные доказательства, которые будут приняты верификатором как корректные.

Таким образом, создана система, которая позволяет разработчикам реализовывать произвольные алгоритмы на основе добавленных базовых схем непосредственно в коде смарт-контрактов. Метод позволяет сократить размер кода смарт-контрактов и, кроме того, оказывается более вычислительно эффективным.

ЛИТЕРАТУРА

1. *Ben-Sasson E., Chiesa A., Genkin D., et al.* SNARKs for C: Verifying program executions succinctly and in zero knowledge // CRYPTO'2013. LNCS. 2013. V. 8043. P. 90–108.
2. *Кондырев Д. О.* Разработка метода сокрытия приватных данных для системы тендеров на основе технологии блокчейн // Прикладная дискретная математика. 2020. № 48. С. 63–81.
3. *Eberhardt J. and Tai S.* ZoKrates — scalable privacy-preserving off-chain computations // IEEE Intern. Conf. Blockchain. Halifax, Canada, 2018. P. 1084–1091.
4. `https://github.com/scipr-lab/libsnark` — libsnark: a C++ library for zkSNARK proofs.

УДК 004.021 DOI 10.17223/2226308X/14/29

ДЕОБФУСКАЦИЯ CONTROL FLOW FLATTENING СРЕДСТВАМИ СИМВОЛЬНОГО ИСПОЛНЕНИЯ

В. В. Лебедев

Метод обфускации Control Flow Flattening заменяет в коде программы все условные и безусловные переходы на переход в специальный управляющий блок — диспетчер, который определяет, куда на самом деле будет передано управление в программе. Это делает невозможным исследователю быстро определить, в какой последовательности исполняется код в программе. Предлагается алгоритм восстановления исходной логики программ, обфусцированных этим методом. В основе алгоритма лежит символьное исполнение.

Ключевые слова: *реверс-инжиниринг, символьное исполнение, обфускация, control flow flattening.*

Введение

Control Flow Flattening [1] — техника обфускации, с помощью которой скрываются ветвления в коде. Вместо последовательного выполнения базовых блоков (линейных участков кода) каждому из них присваивается определённый номер. Вместо прямого перехода на следующий блок номер этого блока записывается в управляющий регистр, затем делается переход в специальный управляющий блок — диспетчер, который, исходя из номера блока, делает на него переход (рис. 1). В коде на языке Си это выглядит как switch внутри цикла while (рис. 2).