ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г. DOI 10.17587/issn.2220-3397 ISSN 2220-3397

Журнал издается при поддержке Отделения математических наук РАН, Отделения нанотехнологий и информационных технологий РАН, МГУ имени М.В. Ломоносова, МГТУ имени Н.Э. Баумана

СОДЕРЖАНИЕ

Журнал зарегистрирован в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций. Свидетельство о регистрации ПИ № ФС77-38590 от 24 декабря 2009 г.

Журнал распространяется по подписке, которую можно оформить в любом почтовом отделении (индекс по Объединенному каталогу "Пресса России" — **22765**) или непосредственно в редакции.
Тел.: (499) 269-53-97. Факс: (499) 269-55-10.
Http://novtex.ru/prin/rus E-mail: prin@novtex.ru
Журнал включен в систему Российского индекса научного цитирования и базу данных RSCI на платформе Web of Science.
Журнал входит в Перечень научных журналов, в которых по рекомендации ВАК РФ должны быть опубликованы научные результаты диссертаций на соискание ученой степени доктора и кандидата наук.

THEORETICAL AND APPLIED SCIENTIFIC AND TECHNICAL JOURNAL

SOFTWARE ENGINEERING

PROGRAMMNAYA INGENERIA

Vol. 10
N 7-8
2019

Published since September 2010 DOI 10.17587/issn.2220-3397 ISSN 2220-3397

CONTENTS

Information about the jornal is available online at:
http://novtex.ru/prin/eng e-mail:prin@novtex.ru

УДК 004.4’233 DOI: 10.17587/prin.10.291-296

В. А. Галатенко, д-р физ.-мат. наук, зав. сектором, e-mail: galat@niisi.ras.ru,
К. А. Костюхин, канд. физ.-мат. наук, ст. науч. сотр., e-mail: kost@niisi.ras.ru,
Научно-исследовательский институт системных исследований Российской академии наук (НИИСИ РАН), Москва

Обратимая отладка*

Представлены результаты авторского исследования истории и методов обратимой отладки, рассмотрен ряд как коммерческих, так и бесплатных средств обратимой отладки. Исследование носит обзорный характер и в кратком изложении обосновывает актуальность постановки задачи, описывает подходы к ее решению.

Ключевые слова: *отладка, обратимая отладка, воспроизведение выполнения, системы реального времени, встроенные системы, симуляторы, виртуальные машины, обратный шаг, обратное выполнение*

Введение

Обратимая отладка — это способность отладчика остановить выполнение программы/системы после возникновения ошибки/нештатной ситуации и провести "обратное" по времени (физическому или логическому) выполнение отлаживаемой программы до момента возникновения ошибки. Эксперименты в области обратимого выполнения начались еще 40 лет назад [1]. Тема обратимой отладки стала актуальной около 10 лет назад [2], когда разработчики сложных, зачастую распределенных и разнородных аппаратно-программных комплексов столкнулись с проблемой возникновения недетерминированных ошибок и сложностью их воспроизведения и локализации.

Брайан Керниган писал [3]: "Известно, что отладка в два раза сложнее написания программы. Поэтому, если Вы были предельно хитроумны при написании программы, то что же Вы будете делать при ее отладке?"

Определим обратимую отладку, как способность отладчика установить точку останова в программе и затем вернуться назад во времени до момента срабатывания точки останова. Существуют различные способы реализации обратимой отладки, например, запись/воспроизведение контрольных событий программы, обратное выполнение, характерное для симуляторов аппаратно-программных комплексов.

Обратимая отладка иногда называется двунаправленной отладкой. Обратимая отладка отличается от классической циклической отладки, при которой выполняется повторный запуск программы под управлением отладчика до тех пор, пока ошибка не будет воспроизведена и локализована. Отметим, что для успешной циклической отладки необходимо, чтобы при каждом запуске программа вела себя одним и тем же образом, т. е. была детерминированной. Это характерно для обычных неинтерактивных однопоточных программ. Однако в случае программ реального времени, параллельных программ или программ, которые используют асинхронный ввод/вывод, повторное выполнение не всегда приводит к одному и тому же результату. Использование отладчика часто нарушает временные циклограммы параллельных программ и программ реального времени, происходит маскировка ошибок. Эти ошибки называются гейзенберговскими (Heisenbugs), поскольку акт наблюдения изменяет поведение отлаживаемой системы.

Таким образом, задача реализации инструментов обратимой отладки при создании средств разработки специального программного обеспечения представляется весьма актуальной задачей, основные способы решения которой будут кратко представлены далее.

Для чего нужна обратимая отладка

Программное обеспечение с годами становится все более сложным. Разработка критически важных систем напрямую зависит от способности средств отладки предоставить программисту возможность быстрой и эффективной локализации ошибок.

С увеличением времени отладки увеличивается и общее время разработки системы, а значит, и ее стоимость. Исследование 2013 г. [4] показало, что

* Результаты исследований, представленные в публикации, выполнены в рамках государственного задания по проведению фундаментальных научных исследований по теме (проекту) "38. Проблемы создания глобальных и интегрированных информационно-телекоммуникационных систем и сетей, развитие технологий и стандартов GRID. Исследование и реализация программной платформы для перспективных многоядерных процессоров (0065-2019-0002)".

разработчики тратят примерно половину своего рабочего времени на отладку существующего кода вместо создания нового. В свою очередь, подавляющая часть времени, необходимого на отладку, тратится на поиск и локализацию ошибок, тогда как исправление найденной ошибки не занимает много времени.

Идея обратимой отладки появилась практически с момента возникновения сложных вычислительных систем. Однако она долго была неприменима ввиду отсутствия необходимого для ее реализации инструментария и вычислительных мощностей. Прогресс в этой области наметился в 2000 г., когда появился первый пример практической реализации обратимой отладки систем, включавший в себя возможность устанавливать специальные точки останова и осуществлять выполнение программы "назад" по этим точкам [1]. До этого все предыдущие исследования были сосредоточены на том, чтобы сохранить и восстановить старое состояние программы для воспроизведения предыдущего сеанса работы программы, породившего сбой.

Рассмотрим небольшой пример из представленного листинга.

```
1. x = 0;
2. while (f(x) > 100) {
3. if (x/2 == 0)
4. x = g(x);
5. else
6. x = h(x);
7. }
8. y = A[x];
```

Пример программы для обратимой отладки

Предположим, в строке 8 листинга переменная *y* получила некорректное значение. Причиной этого явилось некорректное значение переменной *x*. Программисту необходимо исследовать возникшую ситуацию и понять, что, в свою очередь, явилось причиной последнего неправильного изменения значения переменной *x*. Переменная *x* могла изменить значение на строках 1, 4 или 6. Установка точек прерывания на этих строках и повторный запуск программы не является в данном случае оптимальным решением, поскольку обычное "прямое" выполнение программы может занимать неопределенно долгое время. Кроме того, если выяснится, что на последней итерации перед выходом из цикла *x* уже имела некорректное значение, то придется еще раз повторно запускать программу, отслеживая значение переменной *x* уже на предпоследней итерации.

Разумеется, программист может переделать программу, добавив счетчик цикла, чтобы иметь возможность устанавливать условные точки прерывания, срабатывающие на определенной итерации цикла. Однако это влечет за собой модификацию программы, проекта, его пересборку и повторную отладку.

Гораздо проще выявить причину ошибки, если использовать отладчик, поддерживающий возможность обратимой отладки. В этом случае достаточно установить две точки прерывания на строки 1 и 8. Затем, когда выполнение достигнет первой точки, включить в отладчике функцию обратимой отладки (для сохранения необходимых данных) и продолжить выполнение. Теперь можно, остановив выполнение программы на строке 8, сделать несколько шагов "назад" во времени и посмотреть, когда именно переменная *x* получила некорректное значение.

Можно выделить несколько уровней применения средств обратимой отладки в зависимости от конкретных задач, которые решает пользователь.

Обратимая отладка на уровне системы работает на уровне аппаратуры или ядра операционной системы. Главная сложность реализации механизма обратимой отладки на уровне системы заключается в необходимости обработки событий от нескольких процессов/потоков управления или даже нескольких процессоров. В последнем случае разработчики могут воспользоваться средствами самоконтроля систем [5]. Эти средства могут помочь собрать информацию, необходимую для построения прошлого состояния системы. В этом случае разработчик сам расставляет контрольные точки, в которых возможно откатить состояние системы.

Обратимая отладка на уровне пользователя обеспечивается более простыми механизмами, чем обратимая отладка на уровне системы, но имеет некоторые ограничения и сталкивается с серьезными трудностями при работе, например с устройствами ввода/вывода.

Сначала необходимо воспроизвести выполнение системы до некого начального состояния, в котором поведение системы еще не отличается от прогнозируемого. Назовем это начальное состояние контрольной точкой. Затем происходит так называемое выполнение "вперед", во время которого программист исследует поведение системы с целью выявить расхождения реального поведения с прогнозируемым. В определенные моменты времени, пока ошибка не проявилась, можно ставить дополнительные контрольные точки в целях уменьшения времени на следующих сеансах отладки.

Как только ошибка проявилась, программист может повторно воспроизводить работу системы до ближайшей по времени к месту проявления ошибки контрольной точки. Выполнение "вперед" всегда должно быть детерминированным, чтобы добиться одного и того же конечного состояния. Любые асинхронные события должны воспроизводиться управляющей системой таким образом, чтобы не нарушать общую последовательность значимых событий отлаживаемой системы.

Основные способы реализации обратимой отладки

Принципиально невозможно фактически обратить выполнение компьютерной программы. Многие машинные инструкции уничтожают информацию,

например, команда XOR (исключающее ИЛИ) одного и того же регистра или запись нового значения в ячейку памяти. Не существует никакого способа, чтобы получив состояние выполнения программы после подобного действия, можно было сделать вывод о состоянии программы перед его выполнением, если не протоколировать состояние (регистры и память) перед выполнением каждой машинной инструкции. Итак, есть два способа восстановить прошлое состояние. Либо каждый раз сохранять полный протокол выполнения программы, либо реконструировать ход выполнения программы, начиная с какого-то сохраненного состояния. Второй вариант гораздо легче реализуется на практике, поскольку необходимо хранить только ту информацию, которая не может быть реконструирована. Такой подход называется *Record-Replay Debugging* (RRD) — воспроизведение сохраненного сеанса работы системы.

Отметим, что хранение полного протокола выполнения возможно для некоторых симуляторов, а также встраиваемых систем, где широкие возможности трассировки поддерживаются аппаратно.

Промежуточным вариантом между циклической и обратимой отладкой является отладка, основанная на воспроизведении журнала выполнения программы. В этом случае ключевые события, которые могут быть источником недетерминизма программы, записываются в ходе выполнения, а затем воспроизводятся в процессе отладки. То есть фактически отладка по-прежнему выполняется только в прямом направлении по времени. Таким образом, этот подход позволяет применять циклическую отладку к недетерминированной программе. Отметим, что реализация этого подхода значительно проще, чем полноценная обратимая отладка. По такому принципу работает, например, отладчик RR (*Record-Replay*) компании Mozilla [6] или отладчик, интегрированный в симулятор QEMU [7].

Симулятор, используемый в проекте Data General Eagle, еще в 1980 г. имел возможности записи микрокода выполнения системы для облегчения отладки [2]. Такой способ сохранения трассы и последующего воспроизведения является общей чертой многих старых систем моделирования, поскольку идея довольно очевидна и требования к способу сохранения и объему собираемых данных минимальны.

В 1987 г. отладчик Instant Replay Debugger позволил осуществить детерминированное воспроизведение выполнения параллельных программ пользовательского уровня на компьютере BBN Butterfly [8]. Отладчик ориентирован на параллелизм на уровне потоков управления и требует наличия модифицированного ядра операционной системы.

Средства Microsoft Visual Studio 2010 IntelliTrace позволяют записывать историю процесса выполнения для отладки программ, запущенных под управлением виртуальной машины CLR [9] (.NET Framework). При этом имеется возможность выбора данных, которые будут протоколироваться в ходе выполнения программы. Программные механизмы IntelliTrace позволяют пользователю перемещаться назад во времени выполнения программы к выбранным точкам останова, чтобы просмотреть предыдущие сохраненные состояния.

Более слабой формой отладки по воспроизведению является протоколирование и воспроизведение в правильной последовательности только тех событий, которые могли привести к появлению проблемной ситуации. В частности, если источником недетерминизма программы являются сетевые вызовы (например, при использовании протокола UDP), то логично протоколировать и воспроизводить именно их.

Большинство подходов к реализации этого типа обратимой отладки предполагают аппаратную поддержку режима трассировки, что позволяет осуществлять сбор трассы в ходе выполнения целевой системы с минимальным воздействием на саму систему и ее ресурсы.

Одним из таких обратимых отладчиков является Green Hills Time Machine [10], выпущенный в 2003 г. Средства Lauterbach CTS — еще один аппаратный отладчик с возможностью обратимой отладки [11], однако он не имеет обратных точек останова и поэтому, вероятно, его лучше классифицировать как RRD-отладчик.

Отладчик Omniscient [12] реализует обратимую отладку программ, выполняющихся на виртуальной машине Java. Его инструментальные средства поддерживают процессы регистрации всех изменений состояния целевой (подлежащей разработке) программы и в ходе отладки позволяют возвращаться к предыдущим состояниям.

В отладчике GNU Debugger (gdb) 7.0 [13] появилась возможность осуществлять обратимую удаленную отладку через стандартный remote-протокол. Отлаживаемый удаленный процесс осуществляет протоколирование выполненных команд и данных, на которые эти команды воздействует (такая возможность реализуема в ряде симуляторов). Положительным моментом этого решения является доступность и открытость, а также возможность использовать его в собственных системах при условии реализации необходимых команд remote-протокола. На этом решении можно также строить собственные средства отладки, используя gdb в качестве промежуточного (*middleware*) программного средства с помощью, например, протокола gdb-MI. Примером таких обратимых отладчиков являются Simics [14] и UndoDB [4].

Частным случаем такого типа отладки является обратимая отладка с использованием оснащенного исходного кода программы. Наиболее популярным методом в этом случае является оснащение программы на уровне ассемблерного кода [15]. Так, отладчик TRAPEDS выборочно устанавливает контрольные точки в командах перехода и ветвления только в тех участках кода, которые содержат потенциально опасные фрагменты. Отладчик RDXP использует для выполнения оснащенного кода виртуальную машину QEMU.

"Обратный" шаг, "обратные" точки останова и "обратимое" состояние программы

Практически все инструментальные средства обратимой отладки предоставляют пользователю две основные операции: сделать один шаг "назад" в направлении выполнения программы (команда reverse-step, "обратный" шаг) и продолжить выполнение программы в обратном направлении до достижения "обратной" точки останова (команда reverse-continue). Эти команды были описаны и представлены в составе двунаправленного отладчика (*Bidirectional Debugger*) в 2000 г. [1].

Для того чтобы выполнить "обратный" шаг, необходимо, чтобы отладчик имел понятие логического времени, чтобы под этим временем не подразумевалось (например, единицей времени может быть момент выполнения очередной ассемблерной команды или момент отправки/получения сетевого пакета). Тогда предполагая, что в момент выполнения "обратного" шага программа находится во времени t, после выполнения "обратного" шага программа переходит в состояние, соответствующее логическому времени $t-1$. Это означает, что при выполнении "обратного" шага происходит откат состояния программы к предыдущему моменту логического времени. Для многопоточных программ "обратный" шаг может быть глобальным для всех потоков или в пределах текущего потока.

Чтобы вернуться назад до "обратной" точки останова (выполнить команду reverse-continue), необходим немного другой подход, описанный далее.

Шаг 1. Сначала происходит обычный запуск программы до момента проявления ошибки или некоторой заранее определенной контрольной точки.

Шаг 2. Затем программа запускается первый раз в отладочном режиме с расстановкой пользователем "обратных" точек останова до той же контрольной точки из шага 1. Когда выполнение программы доходит до очередной "обратной" точки останова (допустим, с номером n), сохраняется логическое время $t(n)$ в этой точке, после чего выполнение программы продолжается. Все это происходит прозрачным для пользователя образом.

Шаг 3. На этом шаге программа запускается второй раз в отладочном режиме и останавливается в момент логического времени $t(N)$, где N — номер последней "обратной" точки останова, если нумерация "обратных" точек идет по возрастанию от начала выполнения программы.

Шаг 4. Для достижения "обратной" точки останова с номером $N-1$ шаг 3 повторяется до момента $t(N-1)$.

Ранее уже отмечалось, что для эффективной реализации обратимой отладки необходимо протоколировать состояние отлаживаемой программы. Назовем набор свойств и характеристик программы, необходимый для реализации обратимой отладки, обратимым состоянием.

Вопрос о том, что именно должно включать в себя обратимое состояние, имеет основополагающее значение для реализации обратимой отладки на основе реконструкции состояний программы.

Очевидно, что для любой компьютерной программы основным состоянием является содержание регистров процессора и системной памяти, используемой для хранения переменных, стека вызовов и другие данные, на которых работает программа.

Если программа взаимодействует с внешним миром, обратимое состояние программы становится более сложным. В идеале необходимо в каждый момент времени при каждом запуске получать извне одни и те же данные, что невозможно. Поэтому для реализации обратимой отладки такой программы необходимо иметь возможность как можно более детального протоколирования внешних данных с возможностью имитации их поступления в режиме воспроизведения в дальнейшем. Отметим, что протоколировать нужно именно те данные, которые необходимы для реализации обратимой отладки участка кода программы, интересующего пользователя.

Аналогичным образом должна быть решена задача в общем случае асинхронного межпоточного и межпроцессного взаимодействия. И в этом случае также работает Record-Replay-стратегия с ограничениями на критически важные участки кода. Кроме того, должна протоколироваться работа планировщика и средств синхронизации.

Интересное решение предложено в работе [16], где описано iDNA — средство сохранения и повторного детерминированного воспроизведения трассы хода выполнения программы. Это средство в режиме воспроизведения предоставляет пользователю возможность менять местами выполнение отдельных участков кода для анализа возможных последствий и обнаружения еще непроявившихся ошибок.

В случае если программа работает с файлами на диске, необходимо обеспечить возможность откатывать содержимое этих файлов от запуска к запуску программы в режиме воспроизведения. Обычно это достигается на уровне виртуальных машин, где жесткий диск включен в список объектов, важных для реализации обратимого состояния, и фактически неотличим от виртуальной памяти программы.

Ключевой задачей обратимой отладки является получение контроля над целевой системой для организации детерминированного воспроизведения хода выполнения программы. Обычно компьютерные системы не обладают свойством повторяемости, поэтому необходимо реализовать некоторый промежуточный уровень, обеспечивающий необходимый уровень детерминизма.

Наличие инструментальных средств в стандартных системных библиотеках для перехвата и обработки входных и выходных данных, межпотоковых и мехпроцессных взаимодействий — это необходимое условие для реализации воспроизведения выполнения на уровне пользователя, как это показано в работах [1, 4, 8, 17, 18].

Зачастую также становится необходимо проводить модификацию пользовательского кода, чтобы

добавить счетчики времени, журналы модификации данных [1, 17, 18].

Другой подход заключается в работе с программами, которые выполняются на виртуальной машине или платформе (например, JVM [9, 12]). На виртуальных машинах можно проводить обратимую отладку на уровне целевой операционной системы (UML [19] и VmWare [20]), а также осуществлять разработку на виртуальных платформах вроде Simics [14].

Заключение

Для сложных, ответственных систем, встраиваемых систем и систем реального времени поиск и исправление ошибок является приоритетной и трудоемкой задачей. Это означает, что средства отладки должны предоставлять способы и механизмы поиска ошибок за минимальное время с минимальными издержками.

Обратимая отладка обеспечивает жизнеспособный, экономичный способ обнаружения ошибок, поскольку разработчики теперь могут детерминированно воспроизводить выполнение своего кода. Это упрощает быстрый поиск и исправление критических ошибок, обеспечивает сокращение сроков разработки и повышает общую производительность.

Значительно сокращая время отладки, разработчики получают больше времени на создание нового кода, тем самым повышая свою эффективность. Таким образом, исследование и реализация средств и методов обратимой отладки представляется крайне важным в современных отраслях разработки критически важных систем.

Список литературы

1. **Boothe B.** Efficient Algorithms for Bidirectional Debugging // Proc. of ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation (PLDI). May 2000. P. 299—310.
2. **Kidder T.** The Soul of a New Machine. Hachette Book Group USA, NY, 1981. 293 p.
3. **Керниган Б., Плоджер Ф.** Элементы стиля программирования М.: Радио и связь, 1984. 160 с.
4. **UndoDB.** URL: https://undo.io/resources/whitepapers/reverse-debugging-whitepaper/ (дата обращения 05.04.2019).
5. **Галатенко В. А., Костюхин К. А.** Проблемы отладки многопроцессных систем // Программные продукты и системы. 2017. Т. 30, № 3. С. 5—14.
6. **Mozilla** Record-Replay Framework. URL: https://github.com/mozilla/rr (дата обращения 05.04.2019).
7. **QEMU** Record-Replay Feature. URL: https://wiki.qemu.org/Features/record-replay (дата обращения 05.04.2019).
8. **LeBlanc T., Mellor-Crummey J.** Debugging Parallel Programs with Instant Replay // IEEE Transactions on Computers. 1987. Vol. 36, Issue 4. P. 471—482.
9. **Huff I.** Debugging Applications with IntelliTrace // Visual Studio Magazine, 08/01/2010. URL: https://visualstudiomagazine.com/articles/2010/08/01/debugging-applications-with-intellitrace.aspx (дата обращения 05.04.2019).
10. **Lindahl M.** The Device Software Engineer's Best Friend // IEEE Computer. May 2006. P. 95—97.
11. **Lauterbach.** Trace-based Debugging. URL: http://www.lauterbach.com/cts.html (дата обращения 05.04.2019).
12. **Lewis B., Ducasse M.** Using Events to Debug Java Programs Backwards in Time // Proc. of the ACM SIGPLAN 2003 Conference on Object-oriented programming, systems, languages, and applications (OOPSLA). October 2003. P. 96—97.
13. **GDB,** GNU Debugger. URL: https://www.gnu.org/software/gdb (дата обращения 05.04.2019).
14. **Engblom J., Aarno D., Werner B.** Full-System Simulation from Embedded to High-Performance Systems // Processor and System-on-Chip Simulation, Leupers, Rainer and Temam, Springer Verlag, 2010, P. 25—45.
15. **Engblom J.** A review of reverse debugging // Proc. of the 2012 System, Software, SoC and Silicon Debug Conference, Vienna, Austria. 19—20 September 2012. P. 1—6.
16. **Narayanasamy S.** Automatically Classifying Benign and Harmful Data Races Using Replay Analysis // Proc. of ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation (PLDI). June 2007. P. 22—31.
17. **Russinovich M., Cogswell B.** Replay for concurrent nondeterministic shared-memory applications // Proc. of ACM SIGPLAN 1996 Conference on Programming Language Design and Implementation (PLDI). June 1996. P. 258—266.
18. **Bhansali S.** Framework for Instruction-level Tracing and Analysis of Program Executions // Proc. of the 2nd International Conference on Virtual Execution Environments (VEE). ACM Press. June 2006. P. 154—163.
19. **Dunlap G., King S., Cinar S., Bazrai M., Chen P.** ReVirt: enabling intrusion analysis through virtual-machine logging and replay // Proc. of the 5th symposium on Operating systems design and implementation (OSDI). 2002. P. 211—224.
20. **Lewis E.** VMware Workstation 6.5: Reverse and Replay Debugging is Here! URL: http://www.replaydebugging.com/2008/08/vmware-workstation-65-reverse-and.html (дата обращения 05.04.2019).

Reversible Debugging

V. A. Galatenko, galat@niisi.ras.ru, **K. A. Kostyukhin**, kost@niisi.ras.ru,
Federal State Institution "Scientific Research Institute for System Analysis of the Russian Academy of Sciences", Moscow, 117218, Russian Federation

Corresponding author:

Kostyukhin Konstantin A., Senior Researcher, Federal State Institution "Scientific Research Institute for System Analysis of the Russian Academy of Sciences", Moscow, 117218, Russian Federation
E-mail: kost@niisi.ras.ru

Received on April 08, 2019
Accepted on April 16, 2019

Reversible debugging is the ability of the debugger to stop the execution of a program/system after an error/abnormal situation occurs and to perform a "reverse" in time (physical or logical) execution of the debugged program until the error occurs.

Despite the fact that experiments in the field of reversible execution began 40 years ago, the topic of reversible debugging became relevant about 10 years ago, when developers of complex, often distributed and heterogeneous, hardware and software systems faced problems of non-deterministic errors and the complexity of their reproduction and localization.

Let's define reversible debugging as the ability of the debugger to set a breakpoint in the program and then go back in time until the breakpoint is triggered. There are various ways to implement reversible debugging, for example, program control events record-replay, reverse execution, typical for simulators of hardware and software systems.

Reversible debugging is sometimes called bidirectional debugging. Reversible debugging is different from classic cyclic debugging, where you run the program again under debugger control until the error is reproduced and localized. Successful cyclic debugging requires that each run the debugged program behaves the same way, i.e., is deterministic. This is typical for non-interactive single-threaded programs, but for real-time programs, parallel programs, or programs that use asynchronous I/O, re-execution does not always produce the same result. Using the debugger often breaks the cyclograms of parallel programs and real-time programs, so the error becomes masked. Such errors are called Heisenbugs because the act of observation changes the behavior of the system being debugged.

This article will review the history and methods of reversible debugging, consider a number of both commercial and free reversible debugging tools.

Keywords: *debugging, reversible debugging, record-replay debugging, real-time systems, embedded systems, simulators, virtual machines, reverse step, reverse execution*

For citation:

Galatenko V. A., Kostyukhin K. A. Reversible Debugging, *Programmnaya Ingeneria*, 2019, vol. 10, no. 7—8, pp. 291—296.

DOI: 10.17587/prin.10.291-296

References

1. **Boothe B.** Efficient Algorithms for Bidirectional Debugging, *Proc. of ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation (PLDI),* May 2000, P. 299—310.

2. **Kidder T.** *The Soul of a New Machine*, Hachette Book Group USA, NY, 1981, 293 p.

3. **Brian W. Kernighan, P. J. Plauger.** *The Elements of Programming Style*, 2nd Edition. McGraw-Hill, 1978. 168 p.

4. **UndoDB**, available at: https://undo.io/resources/whitepapers/reverse-debugging-whitepaper / (accessed 05.04.2019).

5. **Galatenko V. A., Kostiukhin K. A.** Problemy otladki mnogoprocessnyh system (The multiprocess systems debugging problems), *Programmnye produkty i sistemy*, 2017, vol. 30, no. 3, pp. 5—14 (in Russian).

6. **Mozilla** Record-Replay Framework, available at: https://github.com/mozilla/rr (accessed 05.04.2019).

7. **QEMU** Record-Replay Feature, available at: https://wiki.qemu.org/Features/record-replay (accessed 05.04.2019).

8. **LeBlanc T., Mellor-Crummey J.** Debugging Parallel Programs with Instant Replay, *IEEE Transactions on Computers*, 1987, vol. 36, issue 4, pp. 471—482.

9. **Huff I.** Debugging Applications with IntelliTrace, *Visual Studio Magazine,* 08/01/2010, available at: https://visualstudiomagazine.com/articles/2010/08/01/debugging-applications-with-intellitrace.aspx (accessed 05.04.2019).

10. **Lindahl M.** The Device Software Engineer's Best Friend, *IEEE Computer*, May 2006, pp. 95—97.

11. **Lauterbach.** Trace-based Debugging, available at: http://www.lauterbach.com/cts.html (accessed 05.04.2019).

12. **Lewis B., Ducasse M.** Using Events to Debug Java Programs Backwards in Time: Proc. of the ACM SIGPLAN 2003 Conference on Object-oriented programming, systems, languages, and applications (OOPSLA), October 2003, pp. 96—97.

13. **GDB,** GNU Debugger, available at: https://www.gnu.org/software/gdb (accessed 05.04.2019).

14. **Engblom J., Aarno D., Werner B.** Full-System Simulation from Embedded to High-Performance Systems*, Processor and System-on-Chip Simulation, Leupers, Rainer and Temam,* Springer Verlag, 2010, pp. 25—45.

15. **Engblom J.** A review of reverse debugging, *Proc. of the 2012 System, Software, SoC and Silicon Debug Conference*, Vienna, Austria, 19—20 September, 2012, pp. 1—6.

16. **Narayanasamy S.** Automatically Classifying Benign and Harmful Data Races Using Replay Analysis, *Proc. of ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation (PLDI),* June 2007, pp. 22—31.

17. **Russinovich M., Cogswell B.** Replay for concurrent nondeterministic shared-memory applications, *Proc. of ACM SIGPLAN* 1996 *Conference on Programming Language Design and Implementation (PLDI)*, June 1996, pp. 258—266.

18. **Bhansali S.** Framework for Instruction-level Tracing and Analysis of Program Executions, *Proc. of the 2nd International Conference on Virtual Execution Environments (VEE),* ACM Press, June 2006, pp. 154—163.

19. **Dunlap G., King S., Cinar S., Bazrai M., Chen P.** ReVirt: enabling intrusion analysis through virtual-machine logging and replay, *Proc. of the 5th symposium on Operating systems design and implementation (OSDI),* 2002, pp. 211—224.

20. **Lewis E.** VMware Workstation 6.5: Reverse and Replay Debugging is Here!, available at: http://www.replaydebugging.com/2008/08/vmware-workstation-65-reverse-and.html (accessed 05.04.2019).

УДК 004.5 DOI: 10.17587/prin.10.297-304

Д. И. Читалов, мл. науч. сотр., e-mail: cdi9@yandex.ru,
С. Т. Калашников, канд. техн. наук, зав. отдела, Южно-Уральский федеральный научный центр минералогии и геоэкологии УрО РАН, Челябинская обл., г. Миасс, Ильменский заповедник

Разработка модуля для реализации зеркального отображения расчетных сеток вокруг заданной плоскости в графическом интерфейсе пользователя платформы OpenFOAM

Настоящая статья посвящена особенностям разработки модуля для модификации расчетных сеток посредством зеркального отображения сеточных моделей с помощью утилиты mirrorMesh на этапе препроцессинга моделирования задач механики сплошных сред. Представлены диаграмма, описывающая механизмы работы модуля, и стек используемых в его составе технологий. Описаны результаты применения модуля на примере одной из учебных задач механики сплошных сред для платформы OpenFOAM. Сформулированы выводы по итогам исследования, определена их практическая значимость.

Ключевые слова: *численное моделирование, механика сплошных сред, графический интерфейс пользователя, OpenFOAM, язык программирования Python 3.5, открытое программное обеспечение, утилита mirrorMesh, библиотека PyQt5, СУБД SQLite*

Введение

Настоящая статья является продолжением исследований, начатых авторами в 2015 г. и опубликованных в 2016—2018 гг. в отечественных научных журналах. В рамках этих исследований был разработан графический интерфейс для постановки численных экспериментов с использованием платформы OpenFOAM, которая применяется при моделировании задач в механике сплошной среды (МСС) [1]. Авторы также дополнили графическую оболочку несколькими модулями для подготовки расчетных сеток. Такая подготовка является важным этапом препроцессинга и во многом определяет точность итогового результата [2—4].

В процессе тестирования разработанный авторами программный продукт показал свою практическую ценность на одном из ведущих предприятий в сфере ракетостроения — АО ГРЦ им. Макеева. Данный программный продукт является достойной альтернативой существующим графическим оболочкам для работы с платформой OpenFOAM — Salome [5], Helyx-OS [6], Visual-CFD [7]. Вместе с тем следует отметить, что сама платформа OpenFOAM [8] зарекомендовала себя в качестве эффективного средства построения компьютерных моделей для задач МСС и не уступает коммерческим решениям, в частности, ANSYS.

В настоящей работе представлены результаты дальнейших исследований, направленных на совершенствование созданной авторами графической оболочки. В первую очередь это касается обеспечения доступа пользователей к другим возможностям платформы, в частности, к утилите, отвечающей за зеркальное отображение расчетных сеток вокруг заданной плоскости.

Авторами предложено реализовать задачу путем создания соответствующего модуля и его внедрения в существующую версию графической оболочки. Модуль должен обеспечивать подготовку необходимых служебных файлов и запуск утилиты mirrorMesh. Необходимость в разработке модуля возникла у специалистов АО ГРЦ им. Макеева в процессе работы над различными задачами МСС.

Использование модуля в совокупности с графическим интерфейсом призвано снизить затраты времени специалистов на проведение этапа препроцессинга численного эксперимента. Для эффективного решения этой задачи проанализирована сопроводительная документация по платформе OpenFOAM [9] и руководство пользователя [10].

1. Назначение утилиты mirrorMesh

Для построения сеточных моделей на базе платформы OpenFOAM применяют встроенные утилиты, прежде всего blockMesh и snappyHexMesh. Они позволяют формировать расчетные сетки с помощью шестигранников, а также на основе геометрий триангулированных областей. Однако в некоторых случаях возможностей указанных утилит оказывается недостаточно и возникает необходимость доработок моделей. В таких случаях на помощь приходят допол-

нительные утилиты, например, mirrorMesh, которая позволяет зеркально отобразить сетку вокруг заданной плоскости, определенной в служебном файле mirrorMeshDict. Параметры формирования зеркальной геометрии определяются специалистом в файле mirrorMeshDict. При этом симметрия может быть задана в различных плоскостях. Соответственно, в рамках проекта, направленного на решение одной задачи МСС, может потребоваться создание нескольких файлов-словарей, например, mirrorMeshDict.x и mirrorMeshDict.y, если требуется реализовать отражение расчетной сетки по осям *x* и *y*. В таблице приведены основные параметры для определения плоскости.

Утилита mirrorMesh обеспечивает зеркалирование сетки вокруг заданной плоскости, описанной в словаре system/mirrorMeshDict, т. е. обеспечивает симметрию между узлами сетки. При этом плоскость может определяться тремя путями: через набор коэффициентов для уравнения $ax + by + cz + d$; через набор точек, каждая из которых определяется тремя координатами; через базовую точку и вектор нормали с указанием их параметров. Величина planeTolerance файла mirrorMeshDict определяет значение допустимой погрешности для плоскости.

Благодаря утилите mirrorMeshDict разрешается вопрос отсутствия точной симметрии в сеточной модели. Платформа OpenFOAM, осуществляя автоматическое построение сетки, не гарантирует идеальную симметрию. В этом случае дополнительная утилита mirrorMesh обеспечивает симметричное расположение узлов сетки.

2. Постановка цели и задач исследования

Исследование, результаты которого представлены в настоящей работе, направлено на изучение особенностей модификации сеточных моделей задач МСС путем зеркального отображения моделей вокруг заданной плоскости. В рамках этого исследования авторами проанализированы особенности таких модификаций с использованием утилиты mirrorMesh на основе сопроводительной документации по платформе OpenFOAM [9—10].

Авторами проанализированы функциональные возможности mirrorMesh на этапе подготовки соответствующих служебных файлов, их структура, назначение параметров, допустимые типы данных для них. Для работы с утилитой mirrorMesh помимо файла mirrorMeshDict необходимо подготовить дополнительные служебные файлы topoSetDict и createPatchDict. Эти файлы предназначены для работы с наборами поверхностей, ячеек и точек через словарь (topoSetDict), а также для создания патчей из выбранных граничных поверхностей (createPatchDict). Поверхности определяются либо на основе существующих патчей, либо на основе наборов поверхностей.

Цель работы, результаты которой представлены в настоящей статье, состоит в расширении исходного кода реализованной графической оболочки. Такое расширение направлено на: создание графических элементов (экранных форм и элементов управления); реализацию их программных механизмов, обеспечивающих возможности подготовки пользователем отмеченных ранее служебных файлов; реализацию механизмов запуска утилиты зеркального отображения сеточных моделей вокруг определенной плоскости.

Для достижения сформулированной цели необходимо было выполнить обозначенный далее комплекс работ.

1. Написать исходный код для файлов-модулей, описывающих структуру и логику функционирования окон интерфейса для редактирования служебных файлов mirrorMeshDict, topoSetDict и createPatchDict. Интегрировать исходный код файлов-модулей в программный код графической оболочки.

2. Расширить панель инструментальных средств интерфейса за счет дополнительной кнопки запуска зеркалирования сеточной модели вокруг заданной плоскости.

3. Реализовать вывод содержимого сгенерированных файлов в окне отображения результатов.

4. Реализовать возможность указания параметров зеркалирования для проекта постановки и решения новой задачи МСС, а также возможность редактирования параметров существующего проекта.

3. Средства разработки

Рассматриваемый в настоящей статье программный модуль, как и графическая оболочка, представленная в работе [1], реализован на базе средств для создания настольных приложений Python [11] и PyQt [12]. Язык программирования Python 3.5 описывает логику работы программы, а библиотека PyQt5 реализует элементы графического интерфейса.

Параметры файла mirrorMeshDict

Параметр	Первый вариант определения (плоскость)	Второй вариант определения (встроенные точки)	Третий вариант определения (точка и нормаль)
planeType	planeEquation	planeEquation	pointAndNormal

planeEquationDict	Пример: {
	a 1.0; b 2.0;
	c 3.0; d 0.0;
	}
		Пример: {
		point1 (0 1 0);
		point2 (1 0 0);
		point3 (0 0 1);
		}
			Не используется

pointAndNormalDict	Не используется	Не используется	{
			basePoint (0 0 0);
			normalVector (0 1 0);
			}

Общие параметры			
planeTolerance	Пример: 1e-3	Пример: 1e-3	Пример: 1e-3

Связка технологий Python и PyQt позволяет создавать полноценные настольные программные продукты для решения достаточно широкого спектра задач в различных предметных областях. Освоение рассматриваемых технологий по сравнению с другими средствам разработки занимает меньше времени и не требует приобретения лицензии. Благодаря использованию таких средств разработки приложений повышается производительность программиста и упрощается последующая оптимизация программного кода и доработка функциональных возможностей программ [13].

Для работы с программным кодом модуля применялась интегрированная среда разработки PyCharm. Созданное приложение предназначено для эксплуатации на вычислительных устройствах, работающих под управлением ОС Linux. Кроме того, для его использования необходимо наличие установленного дополнительного программного обеспечения, включая платформу OpenFOAM, среду визуализации ParaView [14], интерпретатор Python 3.5, библиотеку PyQt5.

4. Механизм работы модуля

Модуль для реализации зеркального отображения расчетных сеток авторы встроили в основную версию графической оболочки. Зеркалирование сеточных моделей осуществляется на этапе препроцессинга моделирования задач МСС. Поэтому перед реализацией процедуры зеркалирования пользователю графической оболочки необходимо подготовить новый проект постановки и решения задачи МСС или выбрать один из существующих. Далее выполняется стандартная подготовка расчетной сеточной модели посредством утилит blockMesh, snappyHexMesh или foamyQuadMesh. После такой подготовки пользователь может перейти непосредственно к доработке сеточной модели, т. е., например, к определению параметров зеркалирования. Кроме того, специфика моделируемой задачи МСС может потребовать указания дополнительных параметров сетки. К их числу относятся те, которые связаны с наборами поверхностей, ячеек, точек, а также указания параметров патчей из выбранных граничных поверхностей (файлы-словари topoSetDict и createPatchDict). На рис. 1 представлена диаграмма, описывающая механизм использования модуля.

Работа пользователя с модулем зеркалирования расчетных сеток начинается с выбора проекта постановки и решения задачи МСС, для которого необходимо выполнить зеркальное отображения сеточной модели. У пользователя существует возможность создания нового проекта или выбора одного из существующих. Далее возникает необходимость определить параметры расчетной сетки и выполнить ее генерацию с помощью стандартных утилит платформы OpenFOAM: blockMesh, snappyHexMesh, foamyQuadMesh. После завершения подготовки сеточной модели пользователь может перейти к выполнению дополнительных операций над сеточной моделью, в частности, к реализации ее зеркального отображения. Перечисленные шаги осуществляются через главное окно интерфейса модуля, который описан далее. По итогам модификации сетки пользователь может визуализировать сеточную модель с помощью пакета ParaView и завершить подготовку сеточной модели либо вновь перейти к редактированию параметров.

5. Результаты исследования

По итогам проведенного исследования авторы расширили функциональные возможности предложенной в работе [1] графической оболочки. Она дополняет исходный код приложения модулем для реализации возможности зеркального отображения расчетных сеток вокруг заданной плоскости. Механизм работы модуля основан на подготовке пользователем служебного файла mirrorMeshDict с параметрами процесса зеркалирования и запуска этого процесса посредством утилиты mirrorMesh. При этом добавлена возможность подготовки для расчетного случая дополнительных файлов с параметрами topoSetDict (операции с наборами поверхностей, ячеек и точек) и createPatchDict (создание патчей из выбранных граничных поверхностей).

Для определения структуры и содержимого отмеченных служебных файлов реализованы соответствующие экранные формы с набором элементов управления. Реализована возможность программного запуска утилиты зеркалирования. Предложенный модуль призван упростить выполнение одного из важнейших этапов препроцессинга численного эксперимента — этапа подготовки расчетных сеток. К уже описанным в работах [2—4] опциям подготовки сеточных моделей на базе утилит blockMesh, snappyHexMesh и foamyQuadMesh добавлен доступ к еще одной важной для специалистов

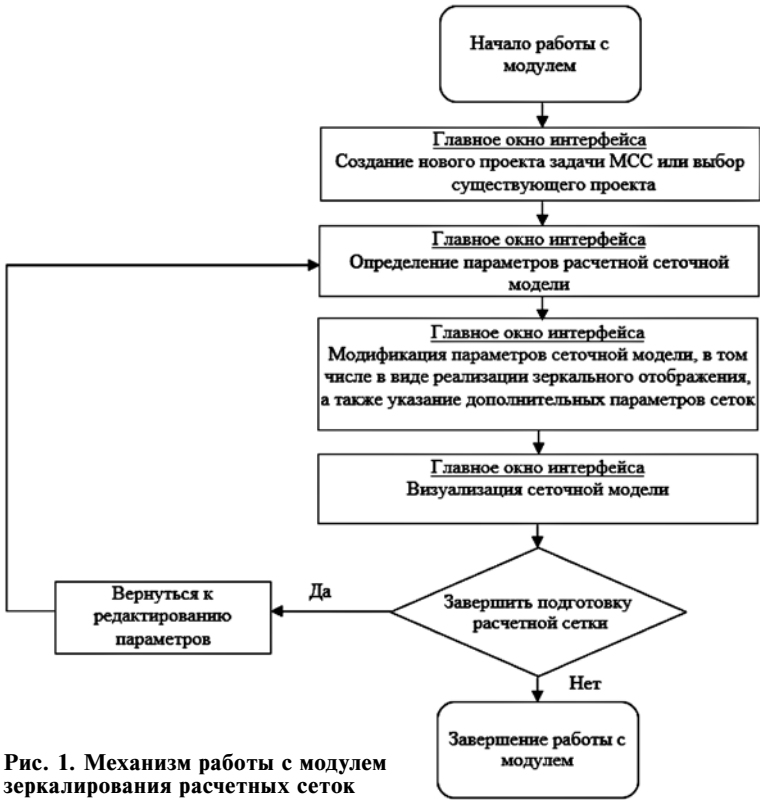


Рис. 1. Механизм работы с модулем зеркалирования расчетных сеток

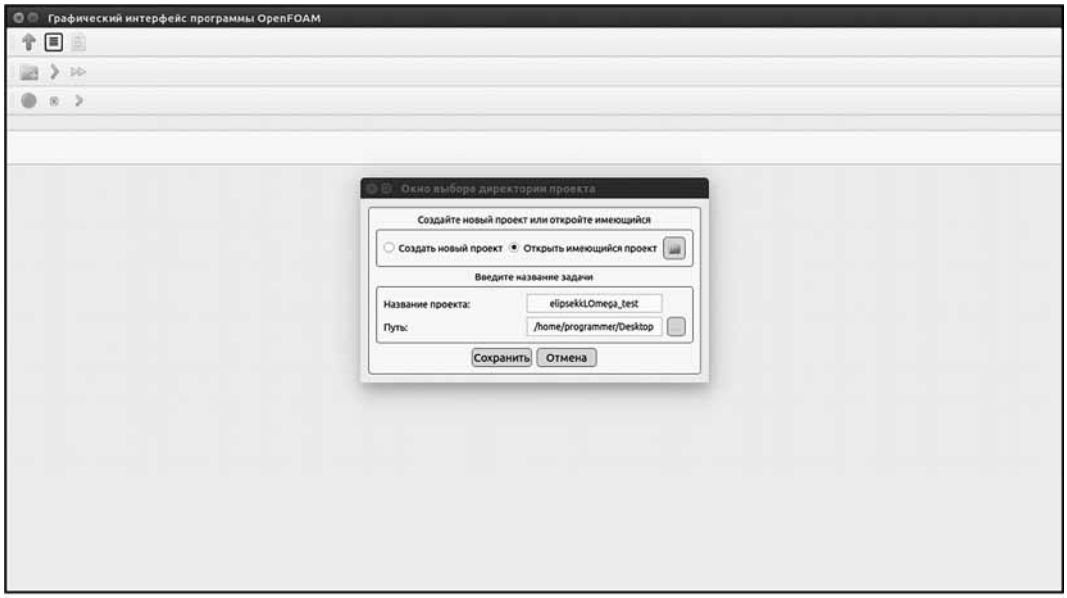


Рис. 2. Главное окно графической оболочки на этапе выбора проекта постановки и решения задачи МСС

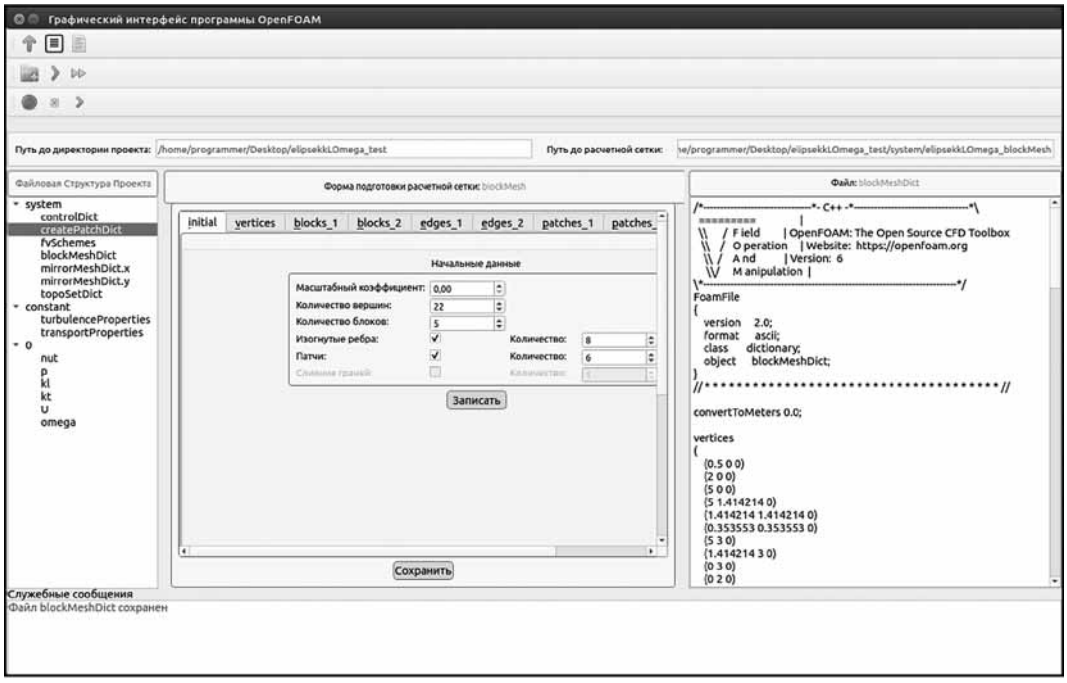


Рис. 3. Главное окно графической оболочки на этапе завершения подготовки файла с параметрами расчетной сетки

утилите — mirrorMesh. Расширенная версия графической оболочки, как и ее первоначальная версия, могут применяться на предприятиях различных отраслей промышленности. В настоящее время работа модуля тестируется специалистами АО ГРЦ им. Макеева в рамках работы по текущим проектам в области ракетно-космического строения.

На рис. 2—7 приведены изображения главного окна модуля на различных этапах постановки численного эксперимента, моделирующего одну из учебных задач МСС, которые прилагаются к дистрибутиву платформы OpenFOAM [15]. Это задача МСС, моделирующая один из процессов для несжимаемого потока. Авторами выполнен весь комплекс сформулированных задач, а именно реализованы файлы-модули, описывающие устройство и механизм функционирования экранных форм для реадактирования параметров служебных файлов mirrorMeshDict, topoSetDict и createPatchDict.

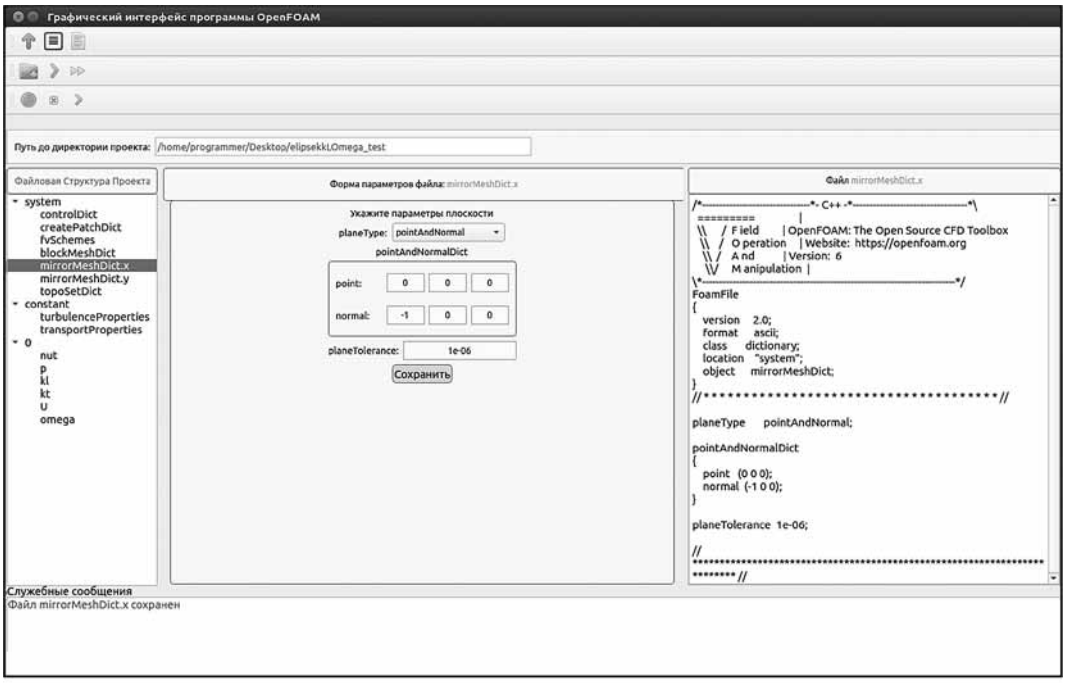


Рис. 4. Главное окно графической оболочки на этапе завершения подготовки файла с параметрами зеркалирования расчетной сетки mirrorMeshDict.x

Исходный код перечисленных файлов модулей интегрирован в приложение с графическим интерфейсом, описанное в работе [1]. Реализован вывод содержимого перечисленных файлов в окне отображения результатов. Реализован программный механизм запуска утилиты mirrorMesh, а также возможность редактирования параметров файлов mirrorMeshDict, topoSetDict и createPatchDict не только для нового проекта, включающего постановку и решение задачи МСС, но и для существующего расчетного случая. Далее будем именовать такой проект проектом решения новой задачи МСС.

В случае, представленном на рис. 2—7, проект решения задачи МСС уже подготовлен. Пользова-

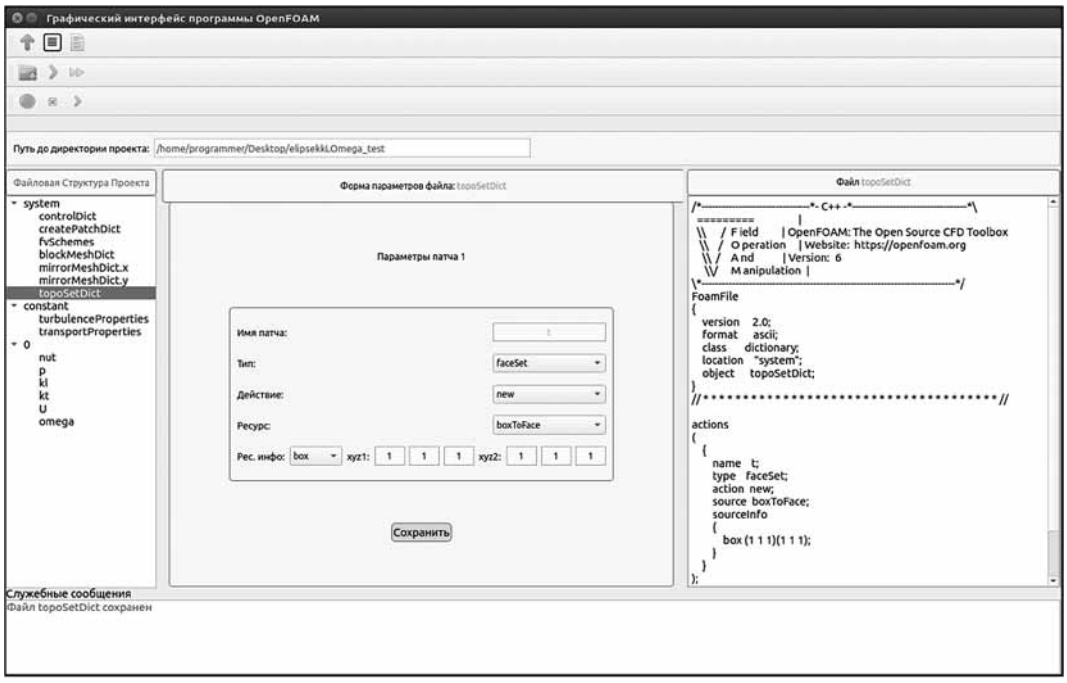


Рис. 5. Главное окно графической оболочки на этапе завершения подготовки файла для операций с наборами поверхностей, ячеек и точек topoSetDict

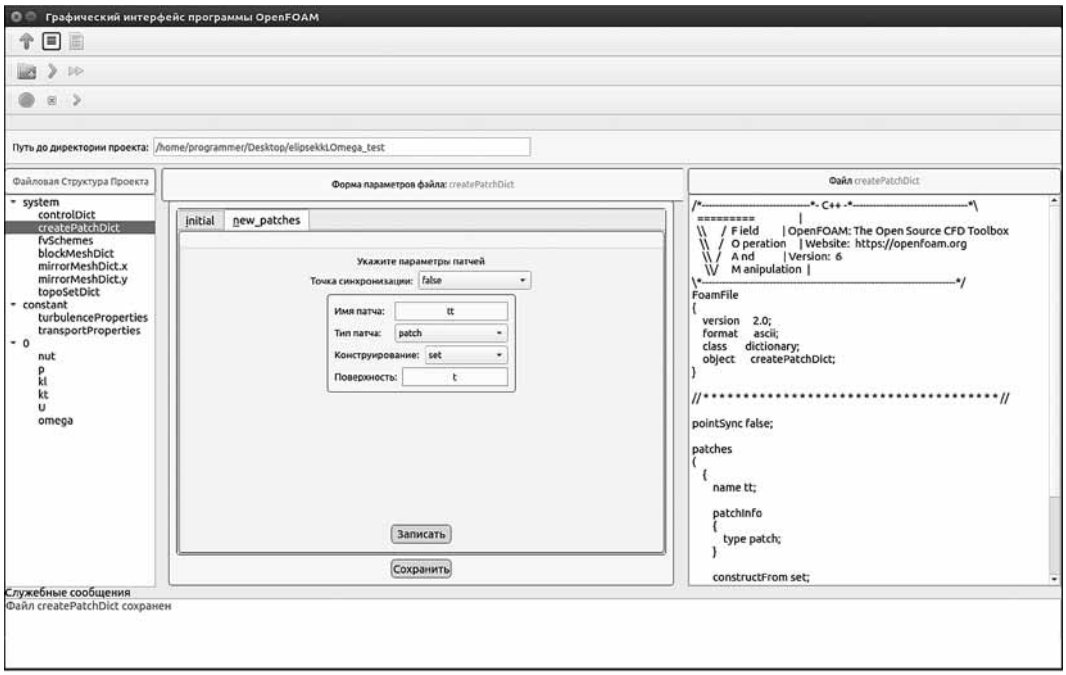


Рис. 6. Главное окно графической оболочки на этапе завершения подготовки файла createPatchDict для управления созданием патчей из выбранных граничных поверхностей

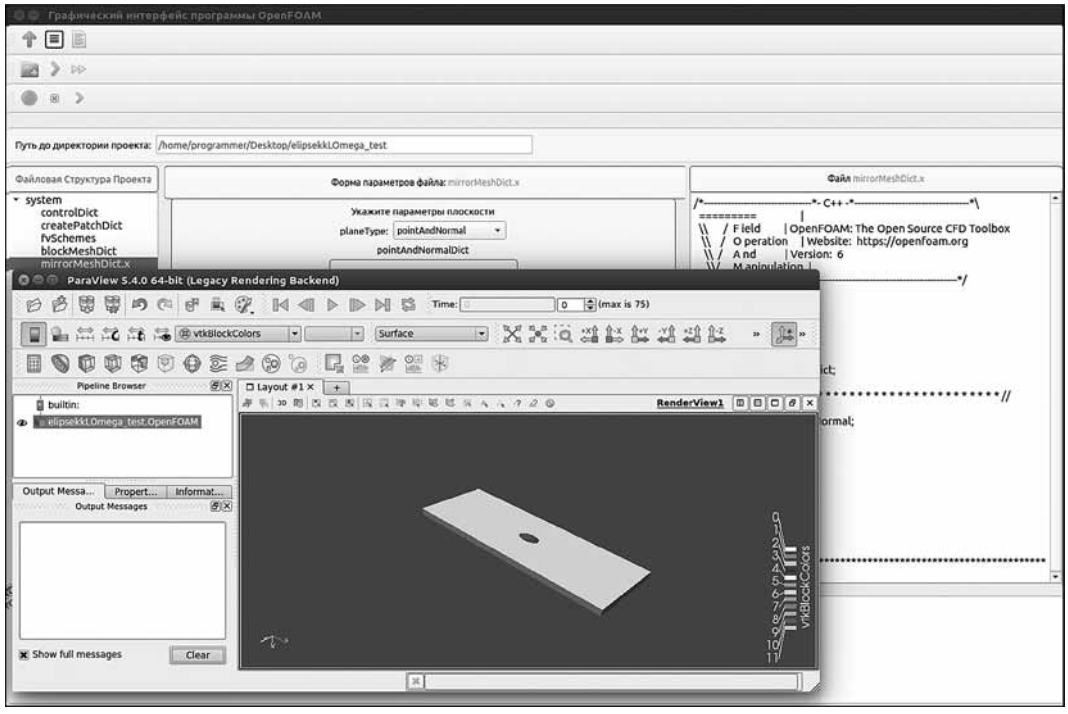


Рис. 7. Главное окно графической оболочки на этапе визуализации результатов подготовки расчетной сетки

телю необходимо выбрать режим работы с приложением "Открыть имеющийся проект" (см. рис. 2). При этом открывается доступ к выбору пути до директории проекта решения задачи — элемент управления "Путь" формы. При этом в элемент управления "Название проекта" автоматически подставляется название проекта постановки и решения задачи МСС.

На следующем шаге (см. рис. 3) выполняется подготовка файла с параметрами для базовой сеточной модели (без дополнительных модификаций) для работы с утилитой blockMesh. Представлено содержимое подготовленного файла. Далее (см. рис. 4) выполняется определение параметров для отображения расчетной сетки по оси *x*, т. е. подготовка файла mirrorMeshDict.x. Аналогично выполня-

ется подготовка mirrorMeshDict.y. В файлах topoSetDict и createPatchDict определены параметры поверхности и патча на основе указанных граничных условий (см. рис. 5—6). В данном случае это поверхность, сформированная на основе прямоугольника, для которого определены координаты двух противоположных вершин.

На последнем этапе осуществляются визуализация результатов, т. е. отображение сеточной модели средствами пакета ParaView (см. рис. 7). В этом случае с учетом модификации осуществляется реализация зеркалирования сеточной модели. Это трехмерная цифровая модель, определяющая исходные данные для численного эксперимента, которая отвечает за дискретизацию пространства и времени.

Заключение

В настоящей работе описан процесс проектирования дополнительного программного модуля для более эффективной реализации графического интерфейса, представленного в работе [1]. Данный модуль расширяет исходный код интерфейса и позволяет выполнять модификацию сеточных моделей задач МСС через зеркальное отображение сеток вокруг заданной плоскости посредством утилиты mirrorMesh платформы OpenFOAM. Предложенный программный модуль упрощает подготовку соответствующих файлов с параметрами и обеспечивает программный запуск утилиты mirrorMesh.

В рамках настоящего исследования авторами предложены механизмы, перечисленные ниже.

- Для редактирования параметров служебных файлов mirrorMeshDict, topoSetDict и createPatchDict разработаны экранные формы.
- Выполнение консольной утилиты mirrorMesh осуществляется через генерацию и запуск соответствующего bash-скрипта [16].
- Для контроля корректности параметров служебных файлов реализованы валидаторы, связанные с элементами управления форм.
- Для обеспечения возможности редактирования параметров файлов расчетного случая предложен подход сериализации параметров путем их сохранения в соответствующие таблицы СУБД SQLite.

Каждая таблица при этом повторяет структуру соответствующего служебного файла. Такой подход обеспечивает сериализацию и последующее восстановление параметров служебных файлов.

- Для минимизации ошибок в процессе проведения численного эксперимента предложен механизм проверки комплектности служебных файлов проекта постановки и решения задачи МСС.

Очередной этап численного эксперимента не может быть выполнен без проверки наличия всех необходимых служебных файлов.

- Предложен механизм подготовки различных версий служебных файлов (с различными параметрами).

Данный механизм позволяет в рамках одного проекта постановки и решения задачи МСС подготовить любое число версий одного и того же служебного файла, тем самым упростить процесс проведения численного эксперимента применительно к различным условиям в задаче МСС.

Описываемый в настоящей работе программный модуль способствует экономии рабочего времени специалистов — инженеров и исследователей. Он упрощает процесс освоения платформы OpenFOAM при моделировании задач МСС и позволяет избежать возможных ошибок численного моделирования (благодаря наличию валидаторов и системы проверки комплектности служебных файлов).

Современные высокоуровневые языки программирования и другие инструментальные средства создания программных приложений становятся все более востребованными среди отечественных разработчиков ПО [17], в том числе при разработке интерфейсов [18]. Предложенный модуль реализован на базе языка программирования Python 3.5, библиотеки графических элементов PyQt5, СУБД SQLite и размещен на веб-сервисе GitHub.

Представленный модуль расширяет исходный код разработанной версии графической оболочки. Обновленная версия также находится в свободном доступе [19] и может применяться коллективами исследователей и инженеров при работе над учебными и реальными задачами в процессе численного моделирования процессов в области МСС.

Список литературы

1. **Читалов Д. И., Меркулов Е. С., Калашников С. Т.** Разработка графического интерфейса пользователя для программного комплекса OpenFOAM // Программная инженерия. 2016. Т. 7, № 12. С. 568—574. DOI: 10.17587/prin.7.568-574.
2. **Читалов Д. И., Калашников С. Т.** Разработка приложения для подготовки расчетных сеток с градуирующими и изогнутыми краями для программной среды OpenFOAM // Системы и средства информатики. 2018. Т. 28, № 4. С. 122—135. DOI: 10.14357/08696527180412.
3. **Читалов Д. И., Калашников С. Т.** Разработка приложения для подготовки расчетных сеток посредством утилиты snappyHexMesh программной среды OpenFOAM // Программные продукты и системы. 2018. Т. 31, № 4. С. 715—722. DOI: 10.15827/0236-235X.124.715-722.
4. **Читалов Д. И., Калашников С. Т.** Разработка приложения для подготовки расчетных сеток с помощью утилиты foamyQuadMesh платформы OpenFOAM//Программная инженерия. 2018. Т. 9, № 7. С. 311—317. DOI: 10.17587/prin.9.311-317.
5. **SALOME.** URL: http://www.salome-platform.org (дата обращения 01.03.2019).
6. **HELYX-OS**. URL: http://engys.com/products/helyx-os (дата обращения 01.03.2019).
7. **Visual-CFD** for OpenFOAM. URL: https://www.esi-group.com/software-solutions/virtual-environment/cfd-multiphysics/visual-cfd-openfoam (дата обращения 01.03.2019).
8. **OpenFOAM.** URL: https://www.openfoam.com/ (дата обращения 01.03.2019).
9. **OpenFOAM.** User Guide. URL: http://foam.sourceforge.net/docs/Guides-a4/OpenFOAMUserGuide-A4.pdf (дата обращения 01.03.2019).
10. **OpenFOAM.** Tutorial Guide. URL: https://www.openfoam.com/documentation/tutorial-guide/index.php (дата обращения 01.03.2019).
11. **Python 3.5** documentation. URL: https://docs.python.org/3.5/ (дата обращения 01.03.2019).
12. **PyQt5** Reference Guide. URL: http://pyqt.sourceforge.net/Docs/PyQt5/ (дата обращения 01.03.2019).
13. **Прохоренок Н. А.** Python 3 и PyQt. Разработка приложений. СПб.: БХВ-Петербург, 2012. 704 с.
14. **ParaView.** URL: https://www.paraview.org/ (дата обращения 01.03.2019).
15. **elipsekkLOmega.** URL: https://github.com/OpenFOAM/OpenFOAM-6/tree/master/tutorials/incompressible/pimpleFoam/RAS/elipsekkLOmega (дата обращения 01.03.2019).
16. **Advanced** bash-scripting guide. URL: https://www.tldp.org/LDP/abs/html/ (дата обращения 01.03.2019).
17. **Пащенко Д. С.** Отражение в российской практике мировых тенденций в технологиях, средствах и подходах в разработке программного обеспечения // Программная инженерия. 2017. Т. 8, № 8. С. 339—344. DOI: 10.17587/prin.8.339-344.
18. **Ченцов П. А.** Об одном подходе к построению интерфейсов консольных приложений: технология TextControlPages // Программная инженерия. 2016. Т. 7, № 12. С. 539—546. DOI: 10.17587/prin.7.539-546.
19. **OpenFOAM_mirror_mesh.** URL: https://github.com/DmitryChitalov/OpenFOAM_mirror_mesh (дата обращения 01.03.2019).

Development of a Module for Implementing the Mirroring of Computational Meshes around a Given Plane in the Graphical User Interface of the OpenFOAM Platform

D. I. Chitalov, cdi9@yandex.ru, **S. T. Kalashnikov**, src@makeyev.ru, South Urals Federal ResearchCentre of Mineralogy and Geoecology of the UB RAS, Chelyabinsk region, 456317, Miass, Ilmen reserve, Federation

Corresponding author:

Chitalov Dmitry Iv., Junior Researcher, South Urals Federal Research Centre of Mineralogy and Geoecology of the UB RAS, Russia, Chelyabinsk region, 456317, Miass, Ilmen reserve, Russian Federation
E-mail: cdi9@yandex.ru

Received on April 11, 2019
Accepted on May 25, 2019

This article is devoted to the features of the development of a module for modifying computational meshes by mirroring mesh models using the mirrorMesh utility at the preprocessing stage of modeling continuum mechanics (CM). Mirror mapping of mesh models is carried out in OpenFOAM using the mirrorMesh utility and allows you to change the geometry characteristics if the CM task requires it. The module is supposed to be integrated into the graphical interface developed by the authors for carrying out numerical simulation of CM problems based on the OpenFOAM platform. This will expand the capabilities of the interface and provide the possibility of its application in more areas of the CM. The article defines the set of tasks and development tools necessary to achieve the goal, a diagram describing the mechanisms of the proposed module is presented. The results of using the module are described using the example of one of the CM training tasks for the OpenFOAM platform, including the visualization of the results using the ParaView package. The mechanisms proposed and implemented in the module are described. The conclusions of the study are formulated, their practical significance is determined. A link is provided to the source code of the module on the GitHub IT project hosting service.

Keywords: *numerical simulation, continuum mechanics, graphical user interface, OpenFOARM, Pyton 3.5, open source software, mirrorMesh utility, PyQt5, SQLite*

For citation:

Chitalov D. I., Kalashnikov S. T. Development of a Module for Implementing the Mirroring of Computational Meshes around a Given Plane in the Graphical User Interface of the OpenFOAM Platform, *Programmnaya Ingeneria,* 2019, vol. 10, no. 7—8, pp. 297—304.

DOI: 10.17587/prin.10.297-304

References

1. **Chitalov D. I., Merkulov E. S., Kalashnikov S. T.** Development of a Graphical User Interface for the OpenFOAM Toolbox, *Programmnaya ingeneria*, 2016, vol. 7, no. 12, pp. 568—574 (in Russian).
2. **Chitalov D. I., Kalashnikov S. T.** Development of an Application for the Preparation of Computational Meshes with Graduating and Curved Edges for the OpenFOAM Software, *Sistemy i sredstva informatiki*, 2018, vol. 28, no. 4, pp. 122—135 (in Russian).
3. **Chitalov D. I., Kalashnikov S. T.** Development of an application for preparing calculation grids using the snappyHexMesh utility of the OpenFOAM software environment, *Programmnye produkty i sistemy*, 2018, vol. 31, no 4, pp. 715—722 (in Russian).
4. **Chitalov D. I., Kalashnikov S. T.** Application Development for Meshes Preparation Using foamyQuadMesh Utility for the OpenFOAM Toolbox, *Programmnaya ingeneria*, 2018, vol. 9, no. 7, pp. 311—317 (in Russian).
5. **Salome.** The Open Source Integration Platform for Numerical Simulation, available at: http://www.salome-platform.org.
6. **Helyx-OS.** Open Source GUI for OpenFOAM, available at: http://engys.com/products/helyx-os.
7. **Visual-CFD** for OpenFOAM. CFD simulation software aimed at solving complex flow applications, available at: https://www.esi-group.com/software-solutions/virtual-environment/cfd-multiphysics/visual-cfd-openfoam.
8. **OpenFOAM.** The open source CFD toolbox, available at: https://www.openfoam.com/.
9. **The OpenFOAM** Foundation. User Guide, available at: http://foam.sourceforge.net/docs/Guides-a4/OpenFOAMUserGuide-A4.pdf.
10. **OpenFOAM.** Tutorial Guide, available at: https://www.openfoam.com/documentation/tutorial-guide/index.php.
11. **Python 3.5** documentation, available at: https://docs.python.org/3.5/.
12. **PyQt5** Reference Guide, available at: http://pyqt.sourceforge.net/Docs/PyQt5/.
13. **Prohorenok N. A.** Python 3 and PyQt. Application Development. St. Petersburg: BHV-Petersburg, 2012. 704 p. (in Russian).
14. **ParaView.** Open-source, multi-platform data analysis and visualization application, available at: https://www.paraview.org/.
15. **OpenFOAM.** Training tasks, available at: https://github.com/OpenFOAM/OpenFOAM-6/tree/master/tutorials/incompressible/pimpleFoam/RAS/elipsekkLOmega.
16. **Advanced** bash-scripting guide, available at: https://www.tldp.org/LDP/abs/html/.
17. **Pashchenko D. S.** Reflection in the Russian Practice of World Trends in Technologies, Tools and Approaches to Software Development, *Programmnaya ingeneria*, 2017, vol. 8, no. 8, pp. 339—344 (in Russian).
18. **Chentsov P. A.** New Way to Construct Console Application Interfaces: Technology TextControlPages, *Programmnaya ingeneria*, 2016, vol. 7, no. 12, pp. 539—546 (in Russian).
19. **OpenFOAM_mirror_mesh**, available at: https://github.com/DmitryChitalov/OpenFOAM_mirror_mesh.

УДК 004.056.55 DOI: 10.17587/prin.10.305-310

Е. А. Матвеев[1], директор, e-mail: eugene.cs@hotmail.com,
С. Е. Игошина[1, 2], канд. физ.-мат. наук, доц., научный консультант, e-mail: sigoshina@mail.ru,
А. А. Карманов[1, 2], канд. физ.-мат. наук, научный консультант, e-mail: starosta07km1@mail.ru
[1] НТП "Криптософт", г. Пенза,
[2] Пензенский государственный университет

Квантовый фазовый переход как основа для практической реализации ATF-технологии связи

ATF-технология является новейшей технологией квантовой связи, на основе которой могут быть построены перспективные аппаратно-программные комплексы защищенных систем передачи информации. В контексте практической реализации ATF-технологии проанализированы основные результаты передовых работ в области квантовых фазовых переходов. Показано, что гипотеза T, лежащая в основе ATF-технологии связи, не противоречит современным экспериментальным данным. Отмечено, что гипотеза T может быть распространена на нестационарные квантовые системы, в которых необходимо учитывать динамику квантовых фазовых переходов.

Ключевые слова: *защита информации, ATF-технология связи, гипотеза T, квантовая связь, квантовый фазовый переход, квантовый механизм Киббла—Зурека (Kibble—Zurek), квантовые флуктуации*

Введение

Цель настоящей работы — аналитический обзор основных результатов недавно вышедших публикаций [1, 2], посвященных исследованию физического явления под названием квантовый фазовый переход (КФП). Указанные работы проанализированы в контексте применимости к реализации ATF-технологии связи [3] — перспективной технологии квантовой связи, основанной на использовании нелокальных эффектов квантовых несепарабельных состояний и КФП. ATF-технология может быть эффективно использована в качестве базовой при построении аппаратно-программных комплексов, служащих для защищенной передачи информации. В настоящей работе, в частности, проведен анализ работ [1, 2] на содержание экспериментальных данных, подтверждающих или опровергающих гипотезу *T*, которая, согласно ранней работе [4], формулируется следующим образом.

Существуют квантовая система BC из двух кубитов B и C и действительное число $\alpha_{KBC} > \frac{1}{\sqrt{2}}$, такие, что состояния этой квантовой системы $a|01\rangle + b|10\rangle$ и $|01\rangle$ статистически неотличимы при измерениях в вычислительном базисе при условии, что квантовая система BC в состоянии $|\psi^{(2)}\rangle = a|01\rangle + b|10\rangle$ (где $a, b \in C$, $|a| > |b| > 0$, $|a|^2 + |b|^2 = 1$) обладает мерой несепарабельности $V(|\psi^{(2)}\rangle)$ меньшей, чем

$$V_{KBC} = 2\alpha_{KBC}\sqrt{1-\alpha_{KBC}^2}.$$

В работе [4] также отмечено, что гипотеза T не противоречит современным представлениям об известном физическом явлении под названием "квантовый фазовый переход" и доказано, что мера несепарабельности квантовой системы BC (во внешнем магнитном поле со значением модуля M вектора магнитной индукции) в состояниях $\left|\psi_3^{(2)}\right\rangle$ и $\left|\psi_4^{(2)}\right\rangle$ стремится к 0 при $M \to \infty$, но не достигает своего предельного значения 0 ни при каком значении M. При этом возникает два основных вопроса, требующих экспериментальной проверки, обозначенных далее.

1. Принимает ли мера несепарабельности квантовой системы BC значение 0, что отвечает современным физическим представлениям, или стремится к 0, но не достигает своего предельного значения 0 ни при каком значении управляющего физического параметра (величины внешнего магнитного поля, отстройки от частоты Раби для атомной системы и т. д.)?

2. Каково поведение реальной физической системы при квантовом фазовом переходе, если она находится в произвольных состояниях, не обязательно стационарных, что отвечает формулировке гипотезы *T*, в которой нет ограничений на стационарные состояния?

Весь представленный ниже материал представляет собой попытку ответить на эти вопросы.

Анализ работы "Квантовый механизм Киббла—Зурека и критическая динамика на программируемом симуляторе Ридберга" в контексте практической реализации ATF-технологи

Анализируемая в настоящем разделе работа [1] посвящена экспериментальному исследованию квантовых фазовых переходов, неравновесную динамику которых описывает квантовый механизм Киббла—Зурека. Изначально механизм Киббла—Зурека (первые публикации [5, 6] относятся к 1976 и 1985 гг.) был предназначен для описания классических фазовых переходов второго рода, простейшим примером которых является переход проводник — сверхпроводник, который наблюдается при охлаждении системы до температур, близких к абсолютному нулю. Оригинальные работы [5, 6] вообще не имели связи с квантовой механикой и относились к области космологии, при этом в качестве основного управляющего параметра, изменение которого приводит к фазовому переходу второго рода, выступала температура. При этом согласно данным работам около точки перехода динамика системы замедляется, и на первый план выходят тепловые флуктуации, которые определяют формирование топологических дефектов.

Только относительно недавно (публикация [7] относится к 2006 г.) механизм Киббла—Зурека был распространен на квантовые системы, в которых динамика фазового перехода определяется не тепловыми, а квантовыми флуктуациями. Следует отдельно отметить, что согласно современным физическим представлениям в квантовых системах могут реализовываться основные состояния различной природы (например, проводящая и сверхпроводящая фаза в металлах), в результате чего эти состояния оказываются конкурирующими друг с другом. В таких квантовых системах даже при температуре абсолютного 0 малое изменение управляющего параметра (например, величины внешнего магнитного поля, давления, отстройки от частоты Раби и т. д.) может приводить к переходу систему из одной фазы в другую.

В контексте ответов на сформулированные для проверки гипотезы T вопросы основной интерес представляют экспериментальные данные по поведению квантовой системы при условии, что управляющий параметр превышает некоторое критическое значение, отвечающее квантовой критической точке. В рамках анализируемой работы [1] в качестве такого управляющего параметра выступала отстройка Δ возбуждающего излучения от частоты Раби Ω для квантовой системы (цепочки) из 51 атома рубидия (^{87}Rb) в оптической ловушке. При этом атомы были равномерно разделены контролируемым расстоянием и находились в электронном основном состоянии $|g\rangle$, гомогенно связанном с возбужденным состоянием Ридберга $|r\rangle$, в котором они испытывают взаимодействие Ван-дер-Ваальса с силой, которая затухает как $V(r) \propto 1/r^6$, где r — межатомное расстояние. Для описания такой цепочки вводится гамильтониан многих тел следующего вида:

$$\frac{H}{\hbar} = \frac{\Omega}{2}\sum_i \left(|g_i\rangle\langle r_i| + |r_i\rangle\langle g_i|\right) - \Delta\sum_i n_i + \sum_{i<j} V_{ij} n_i n_j,$$

где $n_i = |r_i\rangle\langle r_i|$ — проектор на ридберговское состояние с позицией i; Δ и Ω — расстройка и частота Раби когерентной лазерной связи между состояниями $|g\rangle$ и $|r\rangle$ соответственно; V_{ij} — сила взаимодействия между атомами в ридбергоских состояниях с позициями i и j; $\hbar$ — приведенная постоянная Планка.

Основная идея эксперимента, описанного в работе [1], заключалась в том, что для отрицательных значений Δ основное состояние многих тел соответствует состоянию, в котором все атомы находятся в электронном основном состоянии $|g\rangle$ до квантовых флуктуаций, и принадлежат так называемой "беспорядочной" фазе с несломанной пространственной симметрией. Для $\Delta > 0$ несколько пространственно упорядоченных фаз возникают в результате соревнования между условием расстройки, которая благоприятствует большой ридберговской фракции, и ридберговской блокады, которая запрещает одновременное возбуждение атомов отделенными расстояниями, меньшими, чем радиус блокады R_b, определяемый $V(R_b) \equiv \Omega$. Таким образом, меняя значение Δ, авторы работы [1] реализовывали условия возникновения КФП.

Рисунок 1 (см. вторую сторону обложки) иллюстрирует основную идею эксперимента, описанного в работе [1]. В начале эксперимента имеется 51 атом рубидия, причем каждый из них находится в основном состоянии $|g\rangle$ и взаимодействует с ближайшими соседями за счет сил Ван-дер-Ваальса. Затем рассматриваемая квантовая система переводится в возбужденное состояние $|r\rangle$ за счет воздействия лазерного излучения. Если энергия данного излучения *точно* отвечает частоте Раби Ω, то по истечении некоторого времени атомы перейдут в основное состояние и испустят кванты света.

Однако при условии, что энергия возбуждающего лазерного излучения *не строго* соответствует частоте Раби, а отличается от нее на некоторое значение Δ (расстройку), в рассматриваемой квантовой системе возможно создание условий, симулирующих КФП, описываемый механизмом Киббла—Зурека. При этом цепочка из 51 атома рубидия распадется на ряд доменов (красные и серые области 2-й и 4-й полосок на рис. 1, границы доменов обозначены прямоугольниками на 3-й и 5-й полосках соответственно), которые отвечают Z_2-упорядоченной фазе и их размер характеризуется корреляционной длиной ξ.

На рис. 2 (см. вторую сторону обложки) представлены результаты вычисления корреляционной длины как функция конечной расстройки (*final detung* Δ_f).

Анализ представленных на рис. 2 (см. вторую сторону обложки) зависимостей показывает, что корреляционная длина увеличивается по мере при-

ближения к квантовой критической точке КФП, по достижении которой наблюдается ее насыщение до некоторого постоянного значения.

Отметим достаточно важный факт в контексте проверки гипотезы T: размер доменов, а следовательно, и корреляционная длина зависят от скорости изменения расстройки Δ_f (что отмечено стрелками на рис. 1, а также пятью различными кривыми на рис. 2, см. вторую сторону обложки).

На рис. 3 (см. вторую сторону обложки) представлена зависимость корреляционной длины от скорости изменения (развертки) расстройки *s*.

Исходя из анализа представленных на рис. 3 (см. вторую сторону обложки) зависимостей, авторы работы [1] делают вывод, что корреляционная длина как функция скорости изменения (развертки) расстройки может быть описана степенной масштабной моделью вида $\xi(s) = \xi_0(s_0/s)^{\mu}$ с $\mu = 0{,}50(3)$, где неопределенность представляет одно стандартное отклонение. Полученное в результате моделирования значение $\mu = 0{,}50(3)$ полностью согласуются с теоретическими значениями для одномерной модели Изинга и результатами моделирования с использованием матричных результирующих состояний.

Дополнительно авторами в рамках работы [1] было проведено моделирование КФП, отвечающих образованию Z_N-упорядоченных фаз (рис. 4, см. третью сторону обложки).

Основная идея проведенного моделирования, по сути, очень близка к симуляции КФП с образованием Z_2-упорядоченной фазы (рис. 1, см. вторую сторону обложки). Основное отличие в данном случае заключается в том, что учитываются ридберговские возбуждения, отделенные $N > 2$ позициями. В настоящее время для Z_3-упорядоченной фазы выделяют три состояния (красные, синие и зеленые области 2-й полоски сверху на рис. 4, см. третью сторону обложки). При этом значение критического показателя для КФП с образованием Z_3-упорядоченной фазы $\mu_{CCM} < 0{,}45$ и $\mu_{CCM} > 0{,}25$ по данным различных литературных источников. Переход в Z_4-упорядоченную фазу еще более не ясен и, как отмечено в работе [1], "...в настоящее время полное понимание этого перехода недоступно вследствие потенциального присутствия промежуточной непрерывной несоизмеримой фазы".

Полученные авторами работы [1] результаты (рис. 5, см. третью сторону обложки) в основном согласуются с теоретическими результатами и данным численного моделирования.

Анализ экспериментальных данных, представленных на рис. 5 (см. третью сторону обложки), показывает, что для переходов в Z_2-упорядоченную фазу некоторые извлеченные значения μ немного больше, чем экспонента, ожидаемая от модели Изинга — $\mu_{Ising} = 0{,}5$. В рамках работы [1] данные отклонения приписывают комбинации далеко распложенных взаимодействий конечного размера и/или временным эффектам, и систематическим эффектам, связанным с инверсными чередующимися паттернами. Для КФП в Z_3-упорядоченную фазу получено значение $\mu \approx 0{,}38$, что согласуется с теоретическими значениями, полученными путем объединения результатов численного моделирования (что отображено в центральной части рис. 5). Для КФП в Z_4-упорядоченную фазу получено значение $\mu \approx 0{,}25$, о чем в работе [1] сказано: "...подробное теоретическое понимание наших экспериментально наблюдаемых экспонент в Z_4-режиме требует дальнейших исследований".

Таким образом, в настоящем разделе кратко изложена основная идея и результаты работы [1]. В целом представленный материал достаточно интересен с научной точки зрения, однако, по всей видимости, не позволяет достоверно ответить ни на один из двух вопросов, сформулированных во введении.

В частности, нельзя сказать, принимает ли мера несепарабельности квантовой системы *BC* значение 0, что отвечает современным физическим представлениям, или стремится к 0, но не достигает своего предельного значения 0 ни при каком значении управляющего физического параметра (отстройки Δ от частоты Раби для рассматриваемой квантовой системы). Также не ясно, каково поведение рассматриваемой квантовой системы при квантовом фазовом переходе, если она находится в произвольных состояниях, не обязательно стационарных.

Однако при описании основных результатов работы [1] был отдельно выделен значимый факт, что *корреляционная длина зависит от скорости изменения расстройки* Δ_f. По всей видимости, данный факт не является чисто технической деталью, относящейся к конкретному проведенному эксперименту. По своей сути он лежит в основе механизма (модели) Киббла—Зурека, согласно которому при КФП явным образом *должна учитываться* динамика квантовой системы.

В этом контексте приведенное в работе [4] размышление, что "...гипотеза *T* не противоречит явлению под названием квантовый фазовый переход", должно быть дополнено с учетом новых данных, согласно которым КФП *не является стационарным явлением* и при его рассмотрении необходимо учитывать динамику (время) квантовых флуктуаций. Напомним, что при формулировке гипотезы *T* время не учитывается и никакие ограничения на значения *a* и *b* не накладываются, кроме $|a| > |b| > 0$, $|a|^2 + |b|^2 = 1$. Однако, исходя из результатов работы [1] и работы [2], которая будет рассмотрена в следующем разделе, в реальных физических системах *a* и *b* являются функциями времени $a(t)$ и $b(t)$.

Анализ работы "Отображение квантовых флуктуаций около критичности" в контексте практической реализации ATF-технологии

Анализируемая в настоящем разделе работа [2] посвящена исследованию квантовых фазовых переходов, управляемых квантовыми флуктуациями. Основная идея проведенного эксперимента заключалась в том, чтобы раскрыть подробности КФП за счет изучения информации о размерах, продолжи-

тельности (характерном времени) и пространственном распределении квантовых флуктуаций вблизи квантовой критической точки.

Для *прямого экспериментального наблюдения* квантовых флуктуаций вблизи КФП сверхпроводника в изолятор авторы используют сканирующее сверхпроводящее квантовое интерференционное устройство (СКВИД-магнетометр), которое может измерять сверхмалые вариации магнитного поля с пространственным разрешением менее 1 мкм. Выбранная авторами система является двумерной пленкой сверхпроводника NbTiN, в котором может наблюдаться КФП типа сверхпроводник—изолятор.

В ходе эксперимента исследованы три образца (*S*1, *S*2 и *S*3) различной толщины, которые охлаждают до сверхнизких температур (где T_c — критическая температура), в результате чего тепловые флуктуации гасятся и на первый план выходят квантовые флуктуации. Далее, при перемещении СКВИД-магнетометра по поверхности образца эти флуктуации визуализируются (рис. 6). Визуализация принципиально возможна за счет того, что при КФП в рассматриваемой квантовой системе происходит изменение диамагнитной восприимчивости χ образца, которую и измеряет СКВИД-магнетометр.

Анализ изображения, представленного на рис. 6, показывает, что квантовые флуктуации проявляют себя как небольшие черные полосы (области подавленной сверхпроводимости с другим значением диамагнитной восприимчивости) в направлении сканирования. Отметим один очень важный факт, что согласно экспериментальным данным работы [2], наблюдаемые области могут иметь размер десятков микрометров и *не являются статичными,* т. е. самопроизвольно возникают и исчезают.

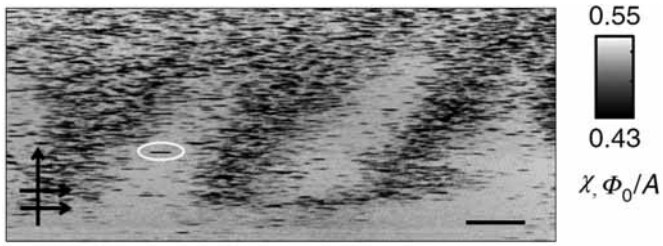


Рис. 6. Изображение, полученное с помощью СКВИД-магнетометра образца *S*1. Показаны более слабые и более сильные области сверхпроводимости. Более слабые области, появляющиеся как полосы в направлении просмотра, отмечены стрелками [2]. Цветовая шкала иллюстрирует значения диамагнитной восприимчивости χ, которая варьируется от минимального значения 0,43 Φ_0/A (где Φ_0 — квант магнитного потока; A — ампер) до максимального значения 0,55 Φ_0/A

На рис. 7 показаны квантовые флуктуации, наблюдаемые в образце *S*1, при различных температурах вблизи квантовой критической точки КФП.

Анализ полученных авторами [2] изображений показывает, при температуре выше критической (T/T_c = 1,08, 6-е по счету изображение на рис. 7) никаких характерных областей не наблюдается (виден только шум), так как система находится вне квантовой критической точки КФП. При температурах ниже T_c возникают области подавленной сверхпроводимости в определенном месте образца *S*1. Отдельно отметим, что данные области сохраняются при температурах значительно ниже T_c, что подтверждает их квантовую, а не тепловую природу. Исходя из этого можно говорить, что они соответствуют именно КФП, а не какому-либо другому физическому явлению.

Основной экспериментальный результат работы [2] в контексте проверки гипотезы T заключается в измерении *продолжительности* квантовых флуктуаций, наблюдаемых вблизи квантовой критической точки КФП (рис. 8). Суть эксперимента состояла в том, что СКВИД-магнетометр устанавливали в точке соответствующей области ослабленной сверхпроводимости (см. рис. 7), а затем в течение продолжительного времени отслеживали изменение $\Delta\chi$ диамагнитной восприимчивости.

Анализ представленных на рис. 8 временных зависимостей показывает, что диамагнитная восприимчивость в заданной локальной области образца *не является постоянной величиной* и для нее наблюдается ярко выраженная динамика.

Также следует особо отметить, что время изменения диамагнитной восприимчивости (время пере-

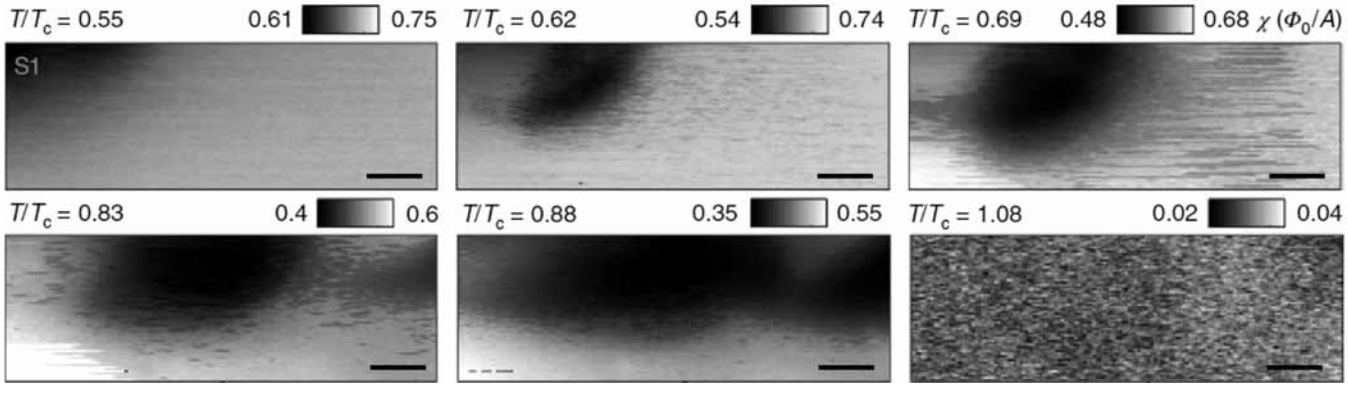


Рис. 7. Температурный диапазон и характерный размер областей квантовых флуктуации вблизи квантовой критической точки КФП [2]. Цветовая шкала иллюстрирует значения диамагнитной восприимчивости χ

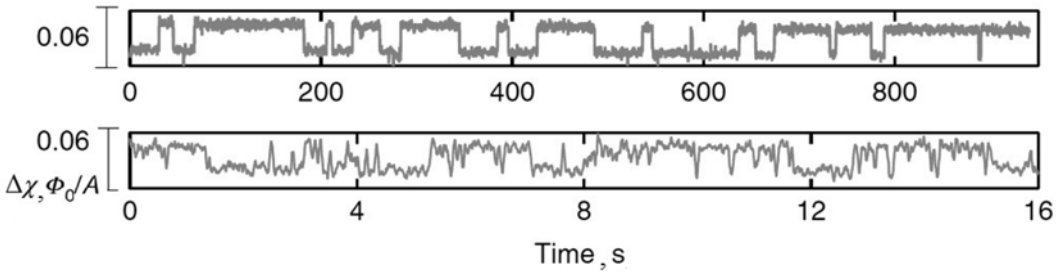


Рис. 8. Динамика появления квантовых флуктуаций в образце *S*1 [2]. По оси абсцисс отложено время, по оси ординат — изменение диамагнитной восприимчивости $\Delta\chi$

ключения t_{max}) может варьироваться от нескольких десятков миллисекунд до десятков минут, и как указывают авторы [2], *"...такое поведение не предсказывается ни одной из существующих теорий"*.

На рис. 9 (см. третью сторону обложки) представлено аналогичное представленному на рис. 8 исследование для различных участков образца (крестообразными маркерами отмечены точки, в которых было проведено измерение). Гистограмма показывает, что время переключения зависит от конкретной области образца и может составлять секунды и даже минуты.

Также отметим, что авторы работы [2] планируют продолжить данный эксперимент, для чего предполагается использование двух СКВИД-магнетометров, одного фиксированного, а другого подвижного для сканирования в качестве средства для изучения квантовой запутанности. В такой установке можно было бы одновременно изучать квантовые флуктуации в двух удаленных местах и искать корреляции, что позволит извлечь информацию об энтропии запутанности.

В настоящем разделе кратко изложена основная идея и результаты работы [2]. В целом представленный материал позволяет сделать заключение, что явление под названием "квантовый фазовый переход" не является статичным, а следовательно, необходимо учитывать его динамику.

По всей видимости, в контексте гипотезы *T*, изложенной в работе [4], наблюдаемое в работе [2] явление (время переключение, а по факту время квантовый флуктуаций, составляющее *от нескольких десятков миллисекунд до десятков минут*) можно интерпретировать как изменение меры несепарабельности рассматриваемой квантовой системы *BC*.

Поскольку в гипотезе *T* предполагается, что "состояния квантовой системы $a|01\rangle + b|10\rangle$ и $|01\rangle$ статистически неотличимы при измерениях в вычислительном базисе при условии, что квантовая система BC в состоянии $|\psi^{(2)}\rangle = a|01\rangle + b|10\rangle$ (где $a, b \in C$, $|a| > |b| > 0$, $|a|^2 + |b|^2 = 1$) обладает мерой несепарабельности $V(|\psi^{(2)}\rangle)$ меньшей, чем $V_{KBC} = 2\alpha_{KBC}\sqrt{1-\alpha_{KBC}^2}$ ", нужно также учитывать, что в *реальных физических системах* при КФП мера несепарабельности квантовой системы $V(|\psi^{(2)}\rangle)$ может изменяться, а следовательно, условие $V(|\psi^{(2)}\rangle) < V_{KBC}$ принципиально может нарушаться.

Заключение

В настоящей работе проведен анализ основных идей и результатов недавно вышедших научных статей, посвященных исследованию физического явления под названием "квантовый фазовый переход" в контексте практической реализации ATF-технологии связи.

Приведенный в работах [1] и [2] материал не позволяет достоверно ответить ни на один из вопросов, сформулированных во введении. Исходя из полученных результатов невозможно сказать, подтверждают они или опровергают гипотезу *T* в контексте ее непротиворечивости с современными представлениями об известном физическом явлении под названием "квантовый фазовый переход".

Однако полученные в работах [1] и [2] результаты могут быть использованы при распространении гипотезы *T* на нестационарные квантовые системы, в которых необходимо учитывать динамику КФП.

В частности, исходя из результатов работ [1] и [2], а также учитывая механизм Киббла—Зурека, при формулировании гипотезы *T* необходимо учитывать, что реальные физические системы, находящиеся в состоянии $|\psi^{(2)}\rangle = a|01\rangle + b|10\rangle$ (где $a, b \in C$, $|a| > |b| > 0$, $|a|^2 + |b|^2 = 1$), *не являются стационарными*, и в общем случае $a(t)$ и $b(t)$, т. е. являются функциями времени. Отдельно оговоримся, что полагая $a(t)$ и $b(t)$, мы не сводим изменения a и b во времени только к процессам декогеренции, которые разрушают запутанность, а следовательно, сопровождаются *уменьшением* меры несепарабельности $V(|\psi^{(2)}\rangle)$. Полагая, что экспериментальные результаты работы [2] достоверны, можно предположить, что мера несепарабельности $V(|\psi^{(2)}\rangle)$ квантовой системы *BC может изменяться* за счет квантовых флуктуаций вблизи квантовой критической точки КФП.

Дополнительно, исходя из работы [1], при обобщении гипотезы *T* на КФП в реальных физических системах целесообразно учитывать, что для данных систем *скорость изменения* управляющего параметра влияет на $a(t)$ и $b(t)$, а как следствие этого — на меру несепарабельности $V(|\psi^{(2)}\rangle)$ квантовой системы *BC*.

Список литературы

1. **Keesling A., Omran A., Levine H.** et al. Quantum Kibble—Zurek mechanism and critical dynamics on a programmable Rydberg simulator // Nature. 2019. Vol. 568. P. 207—211.
2. **Kremen A., Khan H., Loh Y. L.** et al. Imaging quantum fluctuations near criticality // Nature Physics. 2018. Vol. 14. P. 1205—1210.
3. **Алиев Ф. К., Бородин А. М., Васенков А. В.** и др. ATF-технология связи, основанная на использовании ресурса несепарабельных состояний квантовых систем // Наукоемкие технологии. 2015. Т. 16, № 1. С. 65—78.
4. **Алиев Ф. К., Корольков А. В., Матвеев Е. А.** Несепарабельные состояния многокубитных квантовых систем / Под ред. Ф. К. Алиева. М.: Радиотехника, 2017. 320 с.
5. **Kibble T. W. B.** Topology of cosmic domains and strings // J. Phys. A: Math. Gen. 1976. Vol. 9, No. 8. P. 1387—1398.
6. **Zurek W. H.** Cosmological experiments in superfluid helium? // Nature. 1985. Vol. 317. P. 505—508.
7. **Zurek W. H., Dorner U., Zoller P.** Dynamics of a Quantum Phase Transition // Phys. Rev. Lett. 2005. Vol. 95, No. 10. P. 105701.
8. **Гантмахер В. Ф., Долгополов В. Т.** Квантовые фазовые переходы "локализованные-делокализованные электроны" // УФН. 2008. Т. 178. С. 3—24.

Quantum Phase Transition as a Basis for Practical Realization of the ATF Communication Technology

E. A. Matveev, eugene.cs@hotmail.com, Cryptosoft, Penza, 440026, Russian Federation,
S. E. Igoshina, sigoshina@mail.ru, **A. A. Karmanov**, starosta07km1@mail.ru, Penza State University, Penza, 440026, Russian Federation

Corresponding author:

Igoshina Svetlana E., Associate Professor, Penza State University, 440026, Penza, Russian Federation
E-mail: sigoshina@mail.ru

Received on April 24, 2019
Accepted on May 29, 2019

This article analyzes the main ideas and results of recently published scientific articles devoted to the study of a physical phenomenon called as "quantum phase transition" in context of the practical implementation of ATF communication technology. ATF-technology is the newest technology of quantum communication, on the basis of which promising hardware-software complexes of secure information transmission systems can be built. The purpose of this paper is an analytical review of the main results of recent publications [1, 2] devoted to the study "quantum phase transition". These articles are analyzed in context of applicability to the implementation of ATF communication technology [3], a promising technology of quantum communication based on the use of non-local effects of quantum nonseparable states and KPP. ATF-technology can be effectively used as a base when building hardware-software systems used for secure transmission of information. In context of the practical implementation of ATF-technology, the main results of advanced work in the field of quantum phase transitions are analyzed. It is shown that the hypothesis T, underlying the ATF communication technology, does not contradict modern experimental data. It is noted that the hypothesis T can be extended to nonstationary quantum systems in which the dynamics of quantum phase transitions must be taken into account.

Based on the results obtained, it is impossible to say whether they confirm or disprove hypothesis T, in the context of its consistency with modern ideas about the well-known physical phenomenon called "quantum phase transition". However, the results obtained in [1] and [2] can be used to extend the hypothesis T to nonstationary quantum systems, in which the dynamics of a quantum phase transition must be taken into account.

Keywords: *information security, ATF-technology of communication, hypothesis T, quantum phase transition, quantum Kibble—Zurek mechanism, quantum fluctuations*

For citation:

Matveev E. A., Igoshina S. E., Karmanov A. A. Quantum Phase Transition as a Basis for Practical Realization of the ATF Communication Technology, *Programmnaya Ingeneria,* 2019, vol. 10, no. 7—8, pp. 305—310.

DOI: 10.17587/prin.10.305-310

References

1. **Keesling A., Omran A., Levine H., Bernien H., Pichler H., Choi S., Samajdar R., Schwartz S., Silvi P., Sachdev S., Zoller P., Endres M., Greiner M., Vuletic V., Lukin M. D.** Quantum Kibble—Zurek mechanism and critical dynamics on a programmable Rydberg simulator, *Nature*, 2019, vol. 56, pp. 207—211.
2. **Kremen A., Khan H., Loh Y. L., Baturina T. I., Trivedi N., Frydman A., Kalisky B.** Imaging quantum fluctuations near criticality, *Nature Physics*, 2018, vol. 14, pp. 1205—1210.
3. **Aliev F. K., Borodin A. M., Vassenkov A. V., Matveev E. A., Zarkov A. N., Sheremet I. A.** ATF-technology of communication based on using the resource of entangled states of quantum systems, *Visokie Teknologgii*, 2015, vol. 16, no. 1, pp. 65—78 (in Russian).
4. **Aliev F. K., Korolkov A. V., Matveev E. A.** Nonseparable states of multiqubit quantum systems, Moscow, *Radioteknika*, 2017, 320 p. (in Russian).
5. **Kibble T. W. B.** Topology of cosmic domains and strings, *J. Phys. A: Math. Gen.*, 1976, vol. 9, no. 8, pp. 1387—1398.
6. **Zurek W. H.** Cosmological experiments in superfluid helium?, *Nature*, 1985, vol. 317, pp. 505—508.
7. **Zurek W. H., Dorner U., Zoller P.** Dynamics of a Quantum Phase Transition, *Phys. Rev. Lett.*, 2005, vol. 95, no. 10, pp. 105701.
8. **Gantmakher V. F., Dolgopolov V. T.** Localized-delocalized electron quantum phase transitions, *Physics-Uspekhi*, 2008, vol. 51, no. 1, pp. 3—24.

УДК 004.4 DOI: 10.17587/prin.10.311-316

А. А. Скворцов, канд. техн. наук, доц., e-mail: skvalexei@mail.ru, Вятский государственный университет, г. Киров

Проектирование и реализация многозадачных встроенных систем управления на микроконтроллерах: операционная система или автоматы?

На представительном примере проведено сравнение двух способов проектирования и реализации многозадачных встроенных систем управления на микроконтроллерах: с использованием операционной системы реального времени и с помощью конечных автоматов. Показано, что использование конечных автоматов при небольшом увеличении времени на проектирование значительно сокращает объем памяти программ микроконтроллера, объем памяти данных, а также сроки реализации систем управления.

Ключевые слова: *встроенная система управления, микроконтроллер, проектирование, реализация, операционная система реального времени, автоматное программирование, конечный автомат, switch-технология*

Введение

С ростом производительности микроконтроллеров (МК) все большую популярность приобретают операционные системы реального времени (ОСРВ). Однако применение ОСРВ отнимает существенную часть памяти программ МК, а также процессорное время на переключение подлежащих выполнению задач. В связи с этим обстоятельством применение ОСРВ оправдано только в многозадачных системах управления. Однако для таких систем управления есть альтернативный подход, который может существенно сэкономить ресурсы МК — автоматное программирование. В работе [1] рассмотрены вопросы замены многозадачности ОСРВ параллельной работой нескольких управляющих автоматов, а в работах [2, 3] — способы обмена сообщениями между автоматами. По мнению автора, упомянутые статьи дают лишь начальные представления о замене функций ОСРВ набором взаимодействующих автоматов, касаясь вопросов проектирования встроенных многозадачных систем управления недостаточно подробно.

Целью исследования, результаты которого представлены в настоящей работе, является выбор способа проектирования и реализации встроенных многозадачных систем управления на МК на основе сравнительного анализа возможностей, которые представляют ОСРВ и конечные автоматы.

Методика проектирования

Рассмотрим проектирование встроенной системы управления на МК на представительном примере.

Допустим, ведомая консоль управления получает команды по интерфейсу RS-485 со скоростью 57 600 бит/с, обрабатывает их, сканирует клавиатуру, передает по этому же интерфейсу код нажатой кнопки, отображает информацию на 16-символьном жидкокристаллическом (ЖК) дисплее, осуществляет светодиодную и звуковую сигнализацию разной длительности.

Проектирование с использованием механизмов ОСРВ проводилось в такой последовательности:

1) выделение задач;
2) выявление условий запуска задач;
3) определение семафоров, сообщений и точек переключения контекста;
4) разработка алгоритмов задач;
5) разработка алгоритма обработчика прерываний.

По описанию работы ведомой консоли можно выделить пять задач:

1) прием команд по интерфейсу RS-485;
2) вывод символов из буфера на ЖК дисплей;
3) вывод звука заданной длительности;
4) сканирование клавиатуры и передача кода нажатой кнопки по интерфейсу RS-485;
5) светодиодная сигнализация.

Задача 1 должна быть запущена при поступлении байта команды на последовательный порт МК по интерфейсу RS-485. Время между поступлением байтов на последовательный порт при скорости 57 600 бод составляет около 140 мкс, что сопоставимо со временем на переключение задач ОСРВ Salvo (около 130 мкс). За 140 мкс переключиться на другую задачу и обратно будет невозможно. Поэтому задачу 1 удобнее реализовать с помощью обработчика пре-

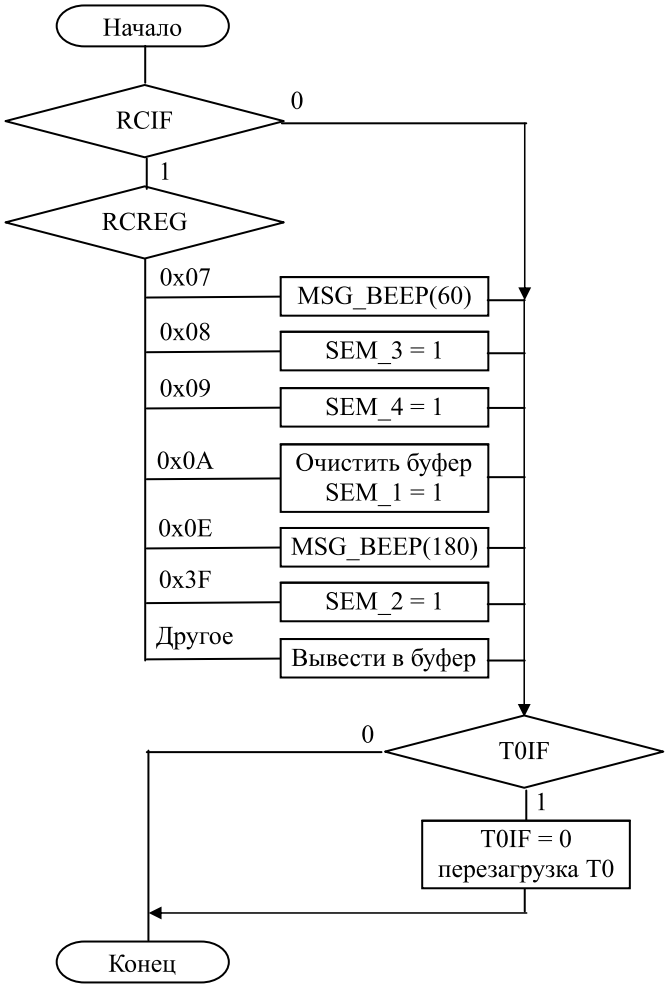


Рис. 1. Алгоритм обработчика прерывания

рывания нулевого уровня. Следовательно, алгоритм задачи 1 будет входить в алгоритм обработчика прерывания (рис. 1). При поступлении байта команды в регистр приемника последовательного порта МК аппаратно устанавливает флаг RCIF. Если данный флаг установлен, то в зависимости от полученной в регистре RCREG команды программа прерывания передает сообщение соответствующей задаче или записывает символы в буфер FIFO. Для отсчета такта системного времени используется таймер T0. При его переполнении загружается начальное значение, флаг T0IF сбрасывается и таймер отсчитывает следующую миллисекунду.

Задача 2 — вывод символов на ЖК дисплей — должна быть запущена после получения по интерфейсу команды "0x0A". В обработчике прерываний семафор 1 устанавливается в "1". Задача запускается, и семафор 1 сбрасывается в "0". Затем при отключенных прерываниях символы выводятся из буфера FIFO на ЖК дисплей (рис. 2, *а*). По окончании выполнения задача ожидает нового переключения семафора 1.

Задача 3 — вывод звука заданной длительности — должна выполняться после получения одной из команд вывода звука: "0x07" — короткий звук (60 мс) или "0x0E" — длинный звук (180 мс). В зависимости от кода команды в обработчике прерываний вырабатывается сообщение MSG_BEEP с параметром "*i*", равным числу миллисекунд. Задача запускается, включается звуковой сигнал и OCPB переключает контекст на время, указанное в параметре "*i*" (см. рис. 2, *б*). В это время возможно выполнение других задач. По истечении указанного времени звуковой сигнал выключается и задача ожидает следующего сообщения MSG_BEEP.

Задача 4 — сканирование клавиатуры — должна быть запущена после получения по интерфейсу команды "0x3F" (ASCII — код знака вопроса). В обработчике прерываний семафор 2 устанавливается в "1". Задача запускается, и семафор 2 сбрасывается в "0". Затем при отключенных прерываниях сканируется клавиатура и передается код клавиши в последовательный порт (рис. 3, *а*). По окончании выполнения задача ожидает нового переключения семафора 2.

Задача 5 — мигание светодиодов — должна быть запущена после получения по интерфейсу команды "0x08". В обработчике прерываний семафор 3 устанавливается в "1". Задача запускается, семафор 3 сбрасывается в "0". Затем происходит циклическое переключение светодиода с задержкой 100 мс после включения, проверкой значения семафора 4 и задержкой 250 мс после выключения светодиода (см. рис. 3, *б*). На время задержки OCPB переключает контекст, что позволяет выполнять другие задачи. При поступлении по интерфейсу команды "0x09" в обработчике прерываний семафор 4 устанавливается в "1". Как только в задаче прочитается значение

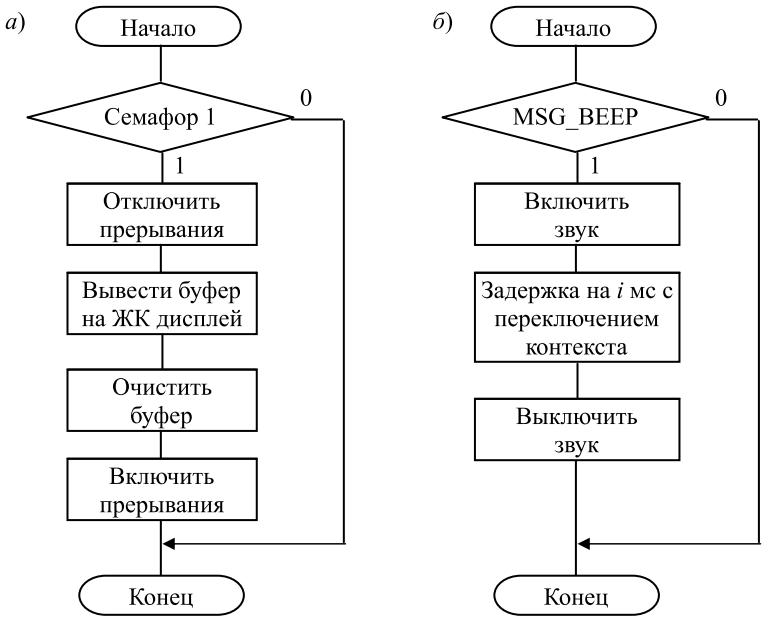


Рис. 2. Алгоритмы задач 2 (*а*) и 3 (*б*)

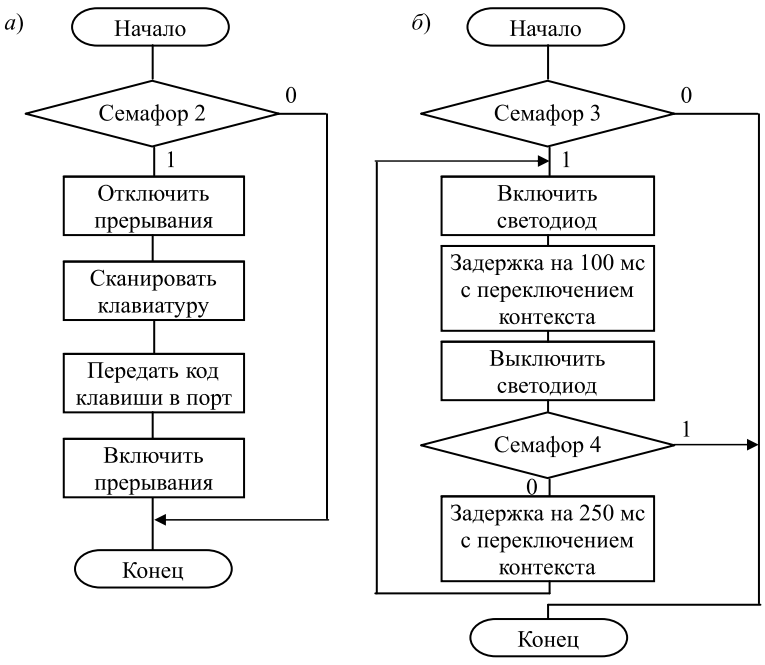


Рис. 3. Алгоритмы задач 4 (*а*) и 5 (*б*)

семафора 4, семафор сбрасывается в "0". Происходит выход из цикла мигания светодиода и задача ожидает нового переключения семафора 3. Аналогичный алгоритм можно разработать для управления двигателем (низкочастотная широтно-импульсная модуляция), звуковой сигнализации и т. п.

Запуск рассматриваемых задач и переключение между ними осуществляет оболочка операционной системы. Рабочий цикл МК состоит всего из одной функции — оболочки ОСРВ Salvo:

```
<Инициализация выводов и устройств МК>
<Инициализация ОС>
<Создание задач>
<Создание семафоров (сообщений)>
while (1) // Рабочий цикл
{
    OSSched();
}
```

Разработка кода проводилась на языке Си в среде MPLAB IDE версии 8.6. Использовалась ОСРВ Salvo, которая занимает меньший объем памяти программ по сравнению с другими популярными ОСРВ. Файлы исходного кода ОСРВ подключались в проект, и затем программа (прошивка) МК собиралась компилятором HI-TECH PICC версии 8.05.

После разработки кода и сборки программы компилятором проводилась проверка корректности переключения задач и их параллельного выполнения в симуляторе MPLAB SIM.

Теперь рассмотрим проектирование описанной выше встроенной системы управления с помощью нескольких конечных автоматов.

Последовательность проектирования с помощью конечных автоматов предусматривала:

1) выделение задач;

2) проектирование автомата для каждой задачи;

3) разработку алгоритма рабочего цикла;

4) разработку алгоритма обработчика прерываний.

Каждой из упомянутых выше выделенных задач можно поставить в соответствие автомат. В начальном состоянии (в данной работе — нулевом) автомат ожидает какого-либо события (переключения семафора, флага поступления сообщения и т. п.). При этом задача не выполняется. Если в алгоритме задачи есть задержка на определенное время, то чтобы обеспечить параллельную работу автоматов, ее нужно выделить в отдельное состояние. В других состояниях задача выполняется.

Автоматы задач вызываются последовательно в рабочем цикле микроконтроллера. В каждой итерации рабочего цикла выполняется один такт каждого автомата. Таким образом осуществляется псевдопараллельное выполнение задач, напоминающее "многопоточность" в операционной системе, в которой каждому потоку выделяется квант процессорного времени. Ниже представлен шаблон рабочего цикла такой автоматной программы:

```
while (1)
{
    <Ожидание начала такта>
    Task1(); // такт автомата задачи 1
    Task2(); // такт автомата задачи 2
    ...
    Task N(); // такт автомата задачи N
}
```

Функции Task1()...Task*N*() реализуют один такт автоматов задач. Для таких автоматов можно составить обобщенную диаграмму переходов, которая представлена на рис. 4.

Для передачи входной информации автоматам, а также для передачи информации между автоматами используется заимствованный из операционных систем механизм сообщений. Вопросы корректного приема сообщений автоматами рассмотрены в работах [2, 3]. В настоящем исследовании они решаются аналогично.

Первоначально автоматы задач находятся в состоянии ожидания сообщения. Принятое сообщение

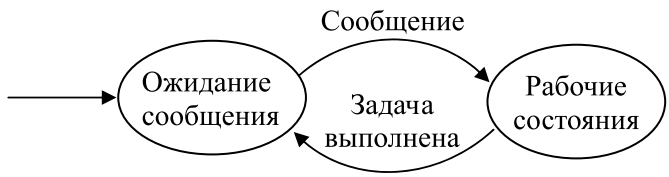


Рис. 4. Обобщенная диаграмма переходов автоматов задач

переводит автоматы в одно из рабочих состояний. После выполнения задачи (или после приема сообщения о ее завершении) автоматы переходят обратно в состояние ожидания сообщения.

Задача 1 так же как при проектировании на базе ОСРВ реализуется в обработчике прерывания (см. рис. 1).

Автомат задачи 2 — вывод символов на ЖК дисплей — имеет два состояния: "0" — ожидание семафора 1; "1" — выполнение задачи. После получения по интерфейсу команды "0x0A" в обработчике прерываний семафор 1 устанавливается в "1". Автомат сбрасывает семафор 1 в "0" и переходит в состояние "1". После окончания выполнения задачи автомат переходит обратно в состояние "0" и ждет нового переключения семафора 1 (рис. 5, *а*). Аналогично работает автомат задачи 4 (сканирование клавиатуры), только здесь переключается семафор 3.

Автомат задачи 3 — вывод звука заданной длительности — имеет два состояния: "0" — ожидание семафора 2; "1" — задержка на время вывода звука. В зависимости от кода команды (0x07 или 0x0E) в обработчике прерываний семафор 2 устанавливается в "1" и вырабатывается сообщение, равное числу миллисекунд времени звука. Автомат принимает сообщение, сбрасывает семафор 2 в "0" и переходит в состояние "1". По окончании времени звука (определяется по счетчику тактов) автомат выключает звук, переходит в состояние "0" и ждет нового переключения семафора 2 (см. рис. 5, *б*). Данный автомат построен по модели Мили: включение и выключение звука являются выходными сигналами. В этой задаче вместе с сообщением используется семафор 2, поэтому в дальнейшем нумерация семафоров будет отличаться от нумерации, принятой при проектировании на базе ОСРВ.

Автомат задачи 5 — мигание светодиодов — имеет три состояния: "0" — ожидание семафора 4; "1" — отсчет времени включения светодиода; "2" — отсчет времени выключения светодиода. После получения по интерфейсу команды "0x08" в обработчике прерываний семафор 4 устанавливается в "1". Автомат сбрасывает семафор 4 в "0", включает светодиод и переходит в состояние "1". По окончании отсчета времени включения светодиода (определяется по счетчику тактов) автомат выключает светодиод и переходит в состояние 2. Как только время выключения пройдет, автомат включит светодиод и перейдет обратно в состояние "1". Автомат будет находиться в состояниях "1" и "2" до тех пор, пока по интерфейсу не придет команда "0x09", которая приведет к переключению в "1" семафора 5 в обработчике прерываний. Тогда автомат из состояния "2" перейдет в состояние "0" и будет ждать нового переключения семафора 4 (см. рис. 5, *в*). Данный автомат также построен по модели Мили: включение и выключение светодиода являются выходными сигналами.

На рис. 6 в качестве примера представлен код задачи 5 на языке Си при двух вариантах реализации, а именно на базе ОСРВ и с помощью нескольких автоматов.

При реализации на базе ОСРВ функции задач вызываются из функции оболочки OSSched(), которая в свою очередь вызывается в каждой итерации рабочего цикла. Каждая задача имеет свой цикл while(1), который обычно начинается с функции ожидания сообщения, запускающего задачу, например, OS_WaitBinSem(). В данной задаче после получения входного сообщения (установки семафора 3 в "1") светодиод будет циклически включаться на 100 мс и выключаться на 250 мс, пока не установится в "1" семафор 4. Функция OSTryBinSem() позволяет прочитать значение семафора 4 и выйти из цикла переключения светодиода.

При автоматной реализации функция автомата задачи вызывается из рабочего цикла. Дополнительных циклов не требуется. Состояния, семафоры, сообщения, счетчики задач удобно оформить в соответствующих массивах. Ожиданию переключения семафора OS_WaitBinSem() соответствует состояние "0", а задержке с переключением контекста OS_Delay() соответствуют состояния "1" в автоматной реализации. Таким образом, при небольшом усложнении программы автоматная реализация системы управления позволяет обойтись без базовых функций ОСРВ.

Разработка кода автоматной реализации также проводилась на языке Си в среде MPLAB IDE версии 8.6, проверка переключения и параллельного выполнения задач — в симуляторе MPLAB SIM.

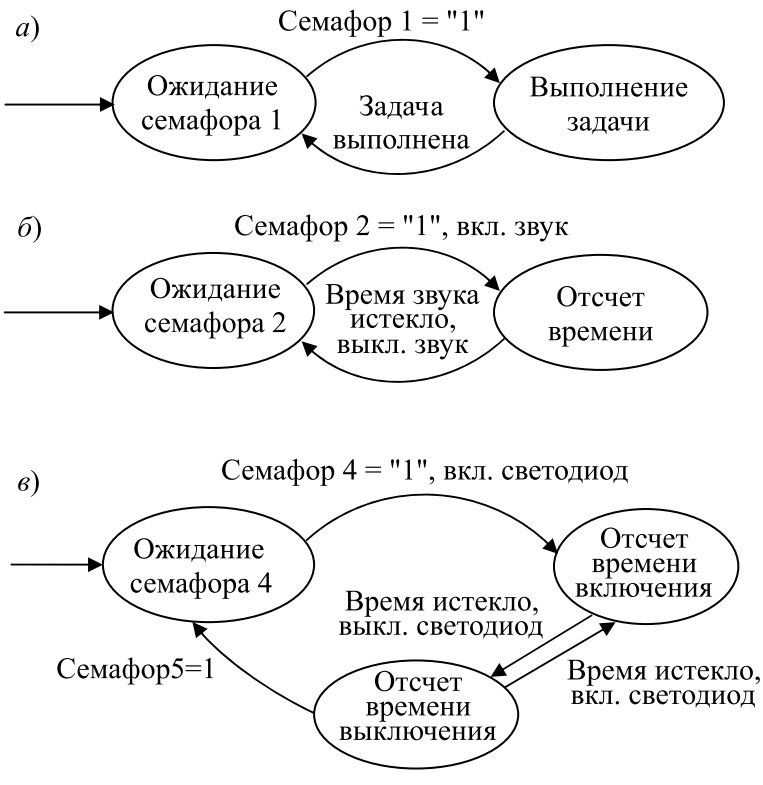


Рис. 5. Диаграммы переходов автоматов задач 2 и 4 (*а*), 3 (*б*) и 5 (*в*)

```
void task_led_mig(void) // реализация на базе ОС Salvo
{
 while(1)
 {
    OS_WaitBinSem(BINSEM_3,OSNO_TIMEOUT,task_led_mig1);
    while(1)
    {
        RC1=1;
        OS_Delay(100,task_led_mig2);
        RC1=0;
        if(OSTryBinSem(BINSEM_4)) {break;}
        OS_Delay(250,task_led_mig3);
    }
 }
}

void task_led_mig(void) // автоматная реализация
{
    switch (States[4]) {
            case 0: if(Sem[4]==1) {RC1=1; Sem[4]=0; States[4]=1;} break;
            case 1: if(tm[4]==100) {tm[4]=0; RC1=0; States[4]=2;} else tm[4]++; break;
            case 2: if(Sem[5]==1) {tm[4]=0; Sem[5]=0; States[4]=0;}
            if(tm[4]==250) {tm[4]=0; RC1=1          ; States[4]=1;} else tm[4]++;
            break;
    }
}
```

Рис. 6. Код задачи 5 при реализации с использованием механизмов ОСРВ и при автоматной реализации

Результаты исследования

При реализации системы управления на базе ОСРВ объем памяти программ МК составил 2755 из 8192 слов (33,6 %), объем памяти данных — 140 из 368 байтов (38,0 %). При автоматной реализации объем памяти программ — 1678 слов (20,5 %), объем памяти данных — 105 из 368 байтов (28,5 %).

Таким образом, для данных пяти задач автоматная реализация системы управления позволила сэкономить 1077 слов (13,1 %) памяти программ и 35 байт (9,5 %) памяти данных. В случае более сложной системы управления экономия памяти будет еще больше.

Применение автоматов также позволило существенно сэкономить время на реализацию многозадачной программы. На первый взгляд, реализовать программу на базе ОС довольно просто. Для этого необходимо подключить библиотеки ОСРВ, определить задачи, разработать их код, определить точки переключения контекста, сообщения, семафоры и таймеры. Однако следует отметить, что на этом пути немало "подводных камней". Значительное количество времени затрачивается на изучение функциональных возможностей ОС, подключение нужных и отключение неиспользуемых функций (в целях экономии памяти программ МК). Настройка файла конфигурации, а именно максимальное число задач, сообщений, семафоров, очередей сообщений, установка системного таймера, количество байт на переменную — счетчик таймера и т. д., также занимает много времени. При неправильной настройке файла конфигурации ОСРВ, как правило, возникает большое число ошибок компиляции, либо программа работает некорректно. На практике оказалось, что правильная работа такой многозадачной программы возможна лишь при сборке в старых версиях компилятора HI-TECH PICC.

Заключение

Исследование на представленном в настоящей работе примере показало, что проектирование и реализация встроенных многозадачных систем управления на МК с помощью конечных автоматов имеют ряд преимуществ. Такой подход позволяет значительно сократить объем памяти программ МК, объем памяти данных. Кроме того, при небольшом увеличении сроков разработки на построение функций переходов и выходов автоматов удается сократить сроки реализации систем управления, так как не возникает необходимости изучения и долгой настройки ОСРВ.

С высокой долей уверенности можно предполагать, что представленный способ проектирования и реализации встроенных многозадачных систем управления с помощью конечных автоматов будет эффективен для многих аналогичных примеру разработок.

Список литературы

1. **Татарчевский В.** Применение Switch-технологии при разработке прикладного программного обеспечения для микроконтроллеров. Часть 1 // Компоненты и технологии. 2006. № 11. URL: https://www.kit-e.ru/articles/circuit/2006_11_164.php

2. **Татарчевский В.** Применение Switch-технологии при разработке прикладного программного обеспечения для микроконтроллеров. Часть 2 // Компоненты и технологии. 2006. № 12. URL: https://www.kit-e.ru/articles/circuit/2006_12_118.php

3. **Татарчевский В.** Применение Switch-технологии при разработке прикладного программного обеспечения для микроконтроллеров. Часть 3 // Компоненты и технологии. 2007. № 1. URL: https://www.kit-e.ru/articles/circuit/2007_1_146.php

Design and Implementation of Multitasking Embedded Control Systems on Microcontrollers: Operating System or State Machines?

A. A. Skvortsov, skvalexei@mail.ru, Vyatka State University, Kirov, 610000, Russian Federation

Corresponding author:

Skvortsov Aleksey A., Associate Professor**,** Vyatka State University, Kirov, 610000, Russian Federation
E-mail: skvalexei@mail.ru

Received on June 06, 2019
Accepted on June 17, 2019

The article presents a comparison of two ways of designing and implementing multitasking embedded control systems on microcontrollers: using a real-time operating system and using finite state machines. With the growth of computing power of microcontrollers (MC), real-time operating systems are becoming increasingly popular. But the application of real-time OS takes up a substantial part of the program memory of the MC, and the CPU time for task switching. In this regard, the use of real-time OS is justified only in multitasking control systems. However, for such control systems there is an alternative approach that can significantly save MC resources — automata-based programming. The purpose of this study is to choose the method of design and implementation of embedded multitasking control systems on the MC: on the basis of real-time OS or using finite state machines. An example of the development and implementation of a simple multitasking embedded control system consisting of five typical tasks is given: receiving commands from the serial port; displaying symbols on the liquid crystal display; producing sound of a given duration; scanning the keyboard and transmitting the code of the pressed button to the port; led alarm. The efficiency of design and implementation of this multitasking system with the help of real-time OS and with the help of several finite state machines was investigated. The study showed that the use of finite state machines with a small increase in design time significantly reduces the amount of programs memory of MC, the amount of data memory, as well as the implementation time of simple control systems. With a high degree of confidence, we can assume that the presented method of designing and implementing embedded multitasking control systems using finite state machines will be effective for many similar example developments.

Keywords: *embedded control system, microcontroller, design, implementation, real-time operating system, automata-based programming, finite state machine, switch-technology*

For citation:

Skvortsov A. A. Design and Implementation of Multitasking Embedded Control Systems on Microcontrollers: Operating System or State Machines? *Programmnaya Ingeneria*, 2019, vol. 10, no. 7—8, pp. 311—316.

DOI: 10.17587/prin.10.311-316

References

1. **Tatarchevskij V.** The use of Switch-technology in the development of application software for microcontrollers. Part 1, *Komponenty i tekhnologii,* 2006, no. 11 (in Russian), available at: https://www.kit-e.ru/articles/circuit/2006_11_164.php

2. **Tatarchevskij V.** The use of Switch-technology in the development of application software for microcontrollers. Part 2, *Komponenty i tekhnologii,* 2006, no. 12 (in Russian), available at: https://www.kit-e.ru/articles/circuit/2006_12_118.php

3. **Tatarchevskij V.** The use of Switch-technology in the development of application software for microcontrollers. Part 3, *Komponenty i tekhnologii,* 2007, no. 1 (in Russian), available at: https://www.kit-e.ru/articles/circuit/2007_1_146.php

УДК 004.056.55 DOI: 10.17587/prin.10.317-323

Е. А. Матвеев, директор, e-mail: eugene.cs@hotmail.com, НТП "Криптософт", г. Пенза

Квантовый телеграф

Представлены основные положения новой технологии передачи и приема данных, названной "квантовый телеграф". Ключевой идеей технологии является наблюдение интерференционной картины (либо ее отсутствие), создаваемой частицами из набора запутанных пар при их прохождении через экран с двумя щелями и фильтрами. Предложена схема для экспериментальной проверки квантового телеграфа с использованием запутанных по поляризации пар фотонов.

Ключевые слова: *квантовый телеграф, квантовые вычисления, квантовая информация, несепарабельные (запутанные) состояния, интерференционная картина на двух щелях*

Введение

Квантовая нелокальность, ярко проявляющаяся в эксперименте Эйнштейна—Подольского—Розена, служит основой для некоторых систем квантовой криптографии [1—3]. В то же время случайный, вероятностный характер результатов измерений над запутанными парами не позволяет напрямую воспользоваться квантовой нелокальностью для передачи информационных сообщений (организации детерминированной квантовой связи). Целью настоящей работы является рассмотрение некоторых теоретических и практических вопросов построения новой технологии приема и передачи данных, названной "квантовый телеграф". Квантовый телеграф, как и ATF-технологии связи [4, 5], является технологией детерминированной квантовой связи. В частности, предложена схема, использующая квантовую запутанность для обеспечения нелокальности и наблюдения интерференционной картины, создаваемой частицами из набора запутанных пар, как возможный способ передачи информации.

При разработке квантового телеграфа были учтены эксперименты по наблюдению интерференционной картины от одиночных фотонов [6], запутанных пар фотонов [7, 8], а также исследования, связанные со "стиранием" информации о выборе пути, известные как "квантовый ластик" [9, 10]. При этом в контексте предлагаемой технологии наиболее важными являются экспериментальные данные, демонстрирующие, что принципиальная возможность получения информации о выборе пути частицей разрушает интерференционную картину в любом аналоге двухщелевого эксперимента [11, 12], а "стирание" информации о выборе пути приводит к ее восстановлению.

1. Анализ известных экспериментов по наблюдению интерференционной картины, создаваемой одиночными фотонами

В настоящее время исследования по наблюдению интерференции частицы в двухщелевом эксперименте (рис. 1) стали классикой физики. Они были неоднократно осуществлены для различных возможных физических реализаций частиц: фотонов [13, 14], электронов [15], нейтронов [16], атомов [17] и т. д. При этом независимо от типа используемых частиц было однозначно

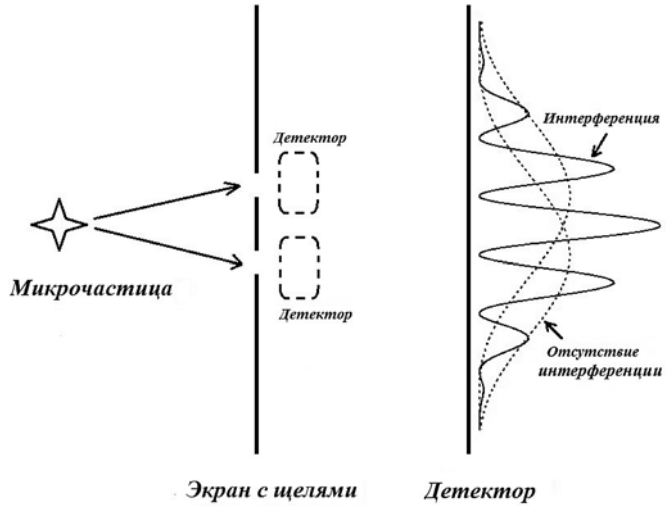


Рис. 1. Схема классического эксперимента по наблюдению интерференции частицы на двух щелях

показано, что если обе щели в экране открыты, на детекторе наблюдается интерференционная картина. Однако если одна из щелей закрыта или каким-либо способом обнаруживается, через какую щель "прошла" частица, то интерференция в экспериментах не проявляется.

В контексте предлагаемой технологии квантового телеграфа наибольший интерес представляют эксперименты по наблюдению интерференционной картины от **одиночных** фотонов, проходящих через экран с двумя щелями, в которые "встроены" фильтры. В основном данные эксперименты проводят в связи с уже вышеупомянутыми исследованиями "квантового стирания" ("ластика"). Хорошим примером является работа 2011 г. [6], в которой описана схема, позволяющая с помощью однофотонного детектора (фотоумножителя) наблюдать интерференционную картину, создаваемую одиночными фотонами, имеющими диагональную поляризацию и "проходящими" через экран с двумя щелями (рис. 2).

При этом рассматривают следующие основные случаи:

1) фотон с диагональной поляризацией "проходит" через экран с двумя щелями (DS) (рис. 2, *а*);

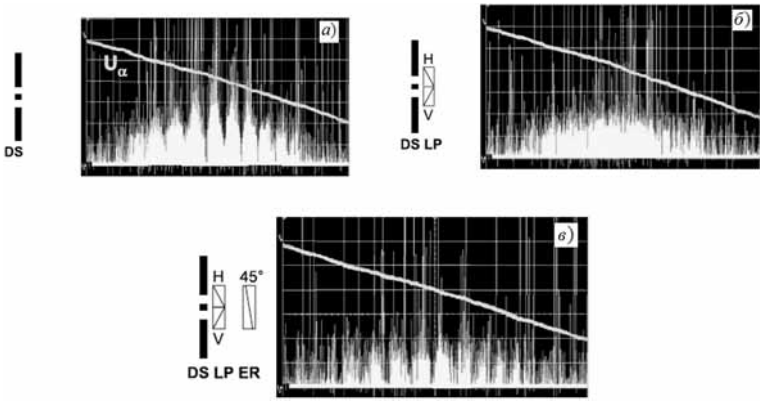


Рис. 2. Результаты эксперимента по наблюдению интерференционной картины от одиночных фотонов, проходящих через экран с двумя щелями и фильтрами [6]. Наклонная линия соответствуют напряжению U_α на фотоумножителе

2) фотон с диагональной поляризацией "проходит" через экран с двумя щелями и фильтрами (LP), в качестве которых использовались линейные поляризаторы, пропускающие только горизонтально (H) и вертикально поляризованные фотоны (V) (рис. 2, *б*);

3) фотон с диагональной поляризацией "проходит" через экран с двумя щелями и фильтрами, а также поляризатор (ER), ориентированный под 45° и "стирающий" информацию о "пути" фотонов (рис. 2, *в*).

Рассмотрим случай 1. "Посылаются" фотоны с диагональной поляризацией, описываемой вектором Джонса следующего вида:

$$|D\rangle = \frac{1}{\sqrt{2}}\left(|H\rangle + |V\rangle\right) = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix},$$

где состояние $|H\rangle \equiv |0\rangle = \begin{pmatrix}1\\0\end{pmatrix}$ соответствует фотону с горизонтальной поляризацией, а состояние $|V\rangle \equiv |1\rangle = \begin{pmatrix}0\\1\end{pmatrix}$ соответствует фотону с вертикальной поляризацией.

Поскольку поляризация фотона в данном случае никак не влияет на его "прохождение" через экран с двумя щелями (без фильтров), в эксперименте наблюдается интерференция (см. рис. 2, *а*). То, что мы видим (линии на черном поле) — это интерференционная картина, создаваемая при "посылке" большого числа фотонов ($N > 10^5$) с диагональной поляризацией и регистрируемая с помощью фотоумножителя.

Рассмотрим случай 2. "Посылаются" фотоны с диагональной поляризацией на экран с двумя щелями и фильтрами.

Поскольку в качестве фильтров используются линейные поляризаторы (H и V), описываемые матрицами Джонса вида

$$\hat{H} = \begin{pmatrix}1 & 0\\0 & 0\end{pmatrix} \text{ и } \hat{V} = \begin{pmatrix}0 & 0\\0 & 1\end{pmatrix},$$

то при "прохождении" фотона через первую щель имеем

$$\hat{H}\,|D\rangle = \begin{pmatrix}1 & 0\\0 & 0\end{pmatrix}\frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\0\end{pmatrix} = \frac{1}{\sqrt{2}}|H\rangle,$$

а при "прохождении" фотона через вторую щель имеем

$$\hat{V}\,|D\rangle = \begin{pmatrix}0 & 0\\0 & 1\end{pmatrix}\frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}0\\1\end{pmatrix} = \frac{1}{\sqrt{2}}|V\rangle,$$

т. е. после "прохождения" экрана с двумя щелями и фильтрами, поляризация фотона становится либо вертикальной $|H\rangle$, либо горизонтальной $|V\rangle$. Поскольку эти состояния ортогональны, никакой интерференции не будет, что и было показано в эксперименте (см. рис. 2, *б*). Ортогональность состояний означает еще и то, что имеется возможность получить достоверную информацию о "пути" фотона.

Рассмотрим случай 3. Дополнительно на пути фотонов, "прошедших" через экран с двумя щелями и фильтрами, устанавливают поляризатор ("ластик"), ориентированный под углом θ, равным 45°, и "стирающий" информацию о "пути" фотонов. Такому поляризатору соответствует матрица Джонса вида

$$\hat{Q} = \begin{pmatrix}\cos^2\theta & \sin\theta\cos\theta\\ \sin\theta\cos\theta & \sin^2\theta\end{pmatrix} = \begin{pmatrix}0{,}5 & 0{,}5\\0{,}5 & 0{,}5\end{pmatrix}.$$

Тогда при "прохождении" фотона через первую щель и фильтр имеем

$$\hat{Q}\hat{H}\,|D\rangle = \begin{pmatrix}0{,}5 & 0{,}5\\0{,}5 & 0{,}5\end{pmatrix}\begin{pmatrix}1 & 0\\0 & 0\end{pmatrix}\frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} = \frac{1}{2\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} = \frac{1}{2}|D\rangle,$$

а при "прохождении" фотона через вторую щель и фильтр имеем

$$\hat{Q}\hat{V}\,|D\rangle = \begin{pmatrix}0{,}5 & 0{,}5\\0{,}5 & 0{,}5\end{pmatrix}\begin{pmatrix}0 & 0\\0 & 1\end{pmatrix}\frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} = \frac{1}{2\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} = \frac{1}{2}|D\rangle.$$

Следовательно, установка "ластика" приводит к поляризационной неразличимости фотонов, "прошедших" через первую и вторую щель. Таким образом, информация о "пути" фотона "стирается", и интерференционная картина восстанавливается, что и было показано в эксперименте (см. рис. 2, *в*). Следует отдельно отметить, что поскольку "ластик" пропускает в среднем только половину фотонов, наблюдаемые в реальном эксперименте интерференционные максимумы имеют меньшую интенсивность.

Отметим важный факт в контексте предлагаемого квантового телеграфа. Поляризационные фильтры, "встроенные" в экран с двумя щелями, ***разрушают*** *интерференционную картину, так как принципиально возможно получение информации о "пути" фотона. Для* ***восстановления*** *интерференционной картины достаточно провести* ***"квантовое стирание"****, т. е. сделать принципиально невозможным получение информации о "пути" фотона, например, разместив после экрана с двумя щелями и фильтрами поляризатор, ориентированный под 45°.*

Далее будет показано, что аналогичный результат восстановления интерференционной картины может быть получен для одного из фотонов запутанной пары (Боба) за счет некоторого воздействия на второй фотон (Алису), которое позволяет "стереть" информацию о выборе "пути" и восстановить интерференционную картину.

2. Анализ известных экспериментов по наблюдению интерференционных картин, создаваемых запутанными фотонами

В настоящее время проведены исследования по наблюдению в двухщелевом эксперименте интерференции одного из фотонов запутанной пары. В качестве примеров таких работ отметим исследования 1994 г. P. H. S. Ribeiro и соавторов [7], а также 1995 г. Д. В. Стрекалова и соавторов [8]. На рис. 3 представлена схема эксперимента из работы [7].

В рамках реализованного эксперимента генерация запутанных пар фотонов осуществлялась в процессе спонтанного параметрического рассеяния, протекающего в нелинейном кристалле $LiIO_3$ при воздействии лазерного излучения, которым освещали кристалл с помощью системы зеркал (M_1 и M_2). В ходе данного процесса возникали сигнальный луч (*signal beam*) и холостой луч (*idlear beam*), которые выделялись с помощью малых отверстий (P_1, P_2 и P_3). При этом сигнальный луч направлялся на экран со щелями, расположенный на расстоянии r_S от нелинейного кристалла, а излучение лазерной накачки останавливалось с помощью экрана (A). Далее сигнальный и холостой луч детектировались с помощью детекторов (D_1 и D_2) в соответствии со схемой совпадения (C). Абсорбционный фильтр (F), интерференционной фильтр (IF), а также линзы (L_1 и L_2) использовали для фильтрации шума.

Результаты проведенного эксперимента представлены на рис. 4, они однозначно показывают наличие интерференционной картины, создаваемой одним из фотонов запутанной пары.

Отметим важный факт в контексте предлагаемого квантового телеграфа: интерференционная картина ***наблюдается*** *в двухщелевом эксперименте для* ***одновременных*** *срабатываний детекторов D_1 и D_2. Одновременное срабатывание служит своеобразным фильтром для детектора D_2, способом учитывать только те фотоны, которые принадлежат запутанной паре. В работе [6] показано, что интерференционная картина наблюдается как от одиночных фотонов, так и от лазерного луча.*

Далее рассмотрим основную идею и результаты эксперимента по наблюдению интерференционной картины, создаваемой одним из фотонов запутанной пары при условии его "прохождения" через экран со щелями и фильтрами. Данный эксперимент (рис. 5) был проведен бразильской группой S. P. Walborn и изложен в работе [9], посвященной "квантовому ластику".

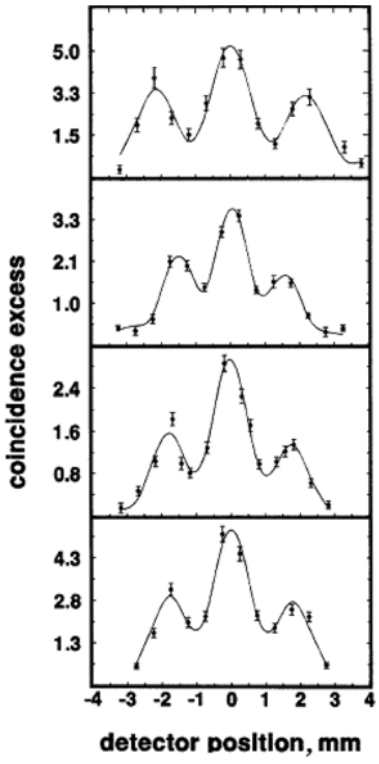


Рис. 4. Экспериментальные данные по избытку совпадений (coincidence excess) отсчетов детекторов (D_1 и D_2) как функция позиции детектора (detector position) при различных параметрах эксперимента [7]

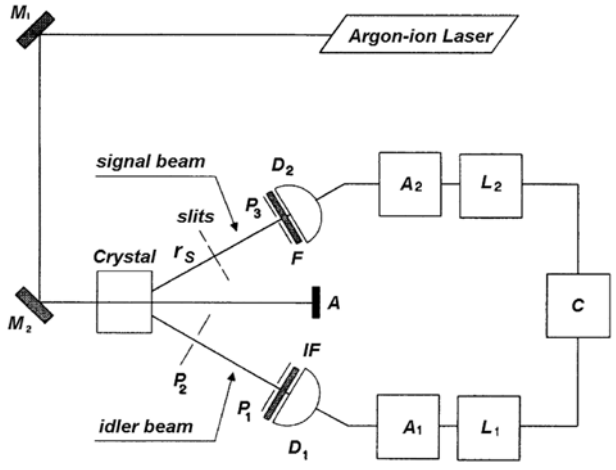


Рис. 3. Схема эксперимента по наблюдению интерференционной картины, создаваемой одним из фотонов запутанной пары [7]

Из описания экспериментальной установки, приведенного в работе [9], можно сделать следующие выводы.

- Генерацию запутанных по поляризации пар фотонов можно осуществить с помощью процесса спонтанного параметрического рассеяния в нели-

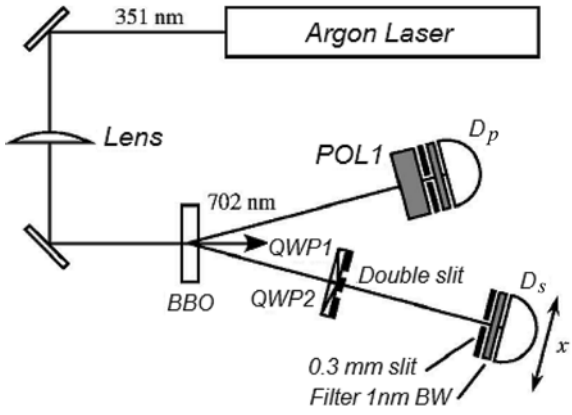


Рис. 5. Схема эксперимента по наблюдению интерференционной картины, создаваемой одним из фотонов запутанной пары с учетом наличия фильтров в экране [9]

нейном кристалле бета бората бария (BBO). Например, в работе [9] таким образом были получены запутанные пары фотонов в состоянии Белла:

$$\left|\Psi^{\pm}\right\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle|1\rangle \pm |1\rangle|0\rangle\right) = \frac{1}{\sqrt{2}}\left(|H\rangle|V\rangle \pm |V\rangle|H\rangle\right),$$

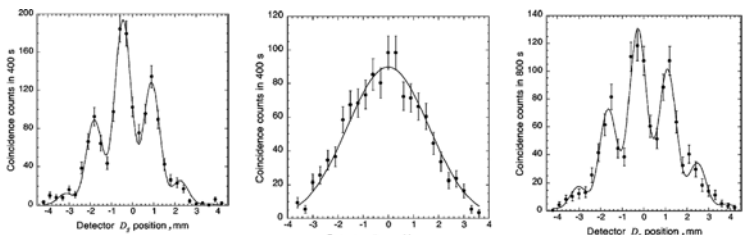
где состояние $|H\rangle = \begin{pmatrix}1\\0\end{pmatrix}$ соответствует фотону с горизонтальной поляризацией, а состояние $|V\rangle = \begin{pmatrix}0\\1\end{pmatrix}$ соответствует фотону с вертикальной поляризацией.

- В качестве аналога щелей (double slit) с фильтрами можно использовать четвертьволновые пластины QWP1 и QWP2, ориентированные под углом в ±45°.
- В качестве детектора интерференционной картины можно использовать однофотонный детектор D_s с полосовым (1 нм) фильтром BW и узкой щелью (0,3 нм), линейно перемещаемый вдоль заданного направления с помощью пьезодвигателя.
- Поляризационный куб POL1, ориентированный под углом 45° и выставленный на "пути" одного из фотонов запутанной пары, можно использовать в качестве "квантового ластика", т. е. стирать информацию о выборе "пути" вторым фотоном (через какую щель с фильтрами он прошел). Регистрация такого фотона может быть осуществлена с помощью однофотонного детектора D_p.

Рис. 6 демонстрирует основные результаты проведенного эксперимента.

Анализ представленных на рис. 6 зависимостей позволяет сделать следующие основные выводы, являющиеся важными в контексте предлагаемого квантового телеграфа:

- если в экран с двумя щелями *не установлены* фильтры, то *наблюдается* интерференционная картина (рис. 6, *а*);
- если в экран с двумя щелями *установлены* фильтры, то интерференционная картина для фотона Боба *не наблюдается* (рис. 6, *б*);
- если один из фотонов запутанной пары пропустить через поляризационный куб POL1, установленный под углом 45°, соответствующим быстрой оси четвертьволновой пластины QWP1, и учитывать только те пары, для которых фотон "прошел" через POL1, информация о выборе пути таким фотоном Боба *стирается* и интерференционная картина *восстанавливается* (рис. 6, *в*).



а) экран с двумя щелями без фильтров

б) экран с двумя щелями и фильтрами

в) экран с двумя щелями и фильтрами, с помощью POL1 под углом 45° стерта информация о выборе «пути»

Рис. 6. Результаты эксперимента по наблюдению интерференционной картины, создаваемой одним из фотонов запутанной пары с учетом наличия фильтров в экране [9]. По оси ординат дано число совпадений отсчетов детекторов D_S и D_p за 400 и 800 с соответственно, по оси абсцисс — позиция детектора D_S

3. Предложения по схеме квантового телеграфа

Проведенный анализ известных экспериментов по наблюдению интерференционной картины, создаваемой как одиночными фотонами, так и одним из фотонов запутанной пары, позволяет предложить схему квантового телеграфа (рис. 7).

Предлагаемая схема может использоваться для передачи между двумя пространственно удаленными абонентами (Алисой и Бобом) битов классической информации "0" и "1", даже при отсутствии между ними классического канала связи. Как и в работе [18], будем полагать, что когда-то давно Алиса и Боб встречались, но теперь живут далеко друг от друга. Будучи вместе, они сгенерировали достаточное число $N \in \mathbf{N}$ запутанных пар частиц A_iB_i (где $i = \overline{1,N}$) в состоянии Белла $|\psi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$, где $\mathbf{N}$ — множество натуральных чисел.

Работа предлагаемого квантового телеграфа состоит из двух частей и включает в себя действия передающей стороны (Алисы) и приемной стороны (Боба) при передаче битов классической информации "0" и "1" соответственно. Боб всегда "пропускает" свои частицы через установку с двумя щелями, фильтрами и "ластиком". Алиса либо ничего не делает со своей частицей (для передачи Бобу "1"), либо "пропускает" через поляризационный фильтр (для передачи Бобу "0").

Часть 1. Передача Алисой бита "1" классической информации.

Шаг 1. Алиса детектирует имеющуюся у нее частицу запутанной пары без проведения каких-либо дополнительных манипуляций над ее состоянием (иначе говоря, Алиса ничего не делает со своей частицей).

Шаг 2. Боб "пропускает" свою частицу запутанной пары через экран с двумя щелям, в который встроены фильтры, "пропускающие" только частицу в состоянии $|0\rangle$ или $|1\rangle$ соответственно, и через "ластик".

Шаг 3. Боб интерпретирует наличие интерференционной картины как факт передачи Алисой бита "1" классической информации.

Часть 2. Передача Алисой бита "0" классической информации.

Шаг 1. Алиса "пропускает" имеющуюся у нее частицу запутанной пары через фильтр $|0\rangle$ и детектирует ее (иначе говоря, Алиса проводит измерение в вычислительном базисе над своей частицей).

Шаг 2. Боб "пропускает" свою частицу запутанной пары через экран с двумя щелям, в который встроены фильтры, "пропускающие" только частицу в состоянии $|0\rangle$ или $|1\rangle$ соответственно, и через "ластик".

Шаг 3. Боб интерпретирует отсутствие интерференционной картины как факт передачи Алисой бита "0" классической информации.

Теперь после изложения схемы квантового телеграфа покажем, что при действиях Алисы, направленных на передачу бита "1" классической информации, Боб будет наблюдать интерферен-

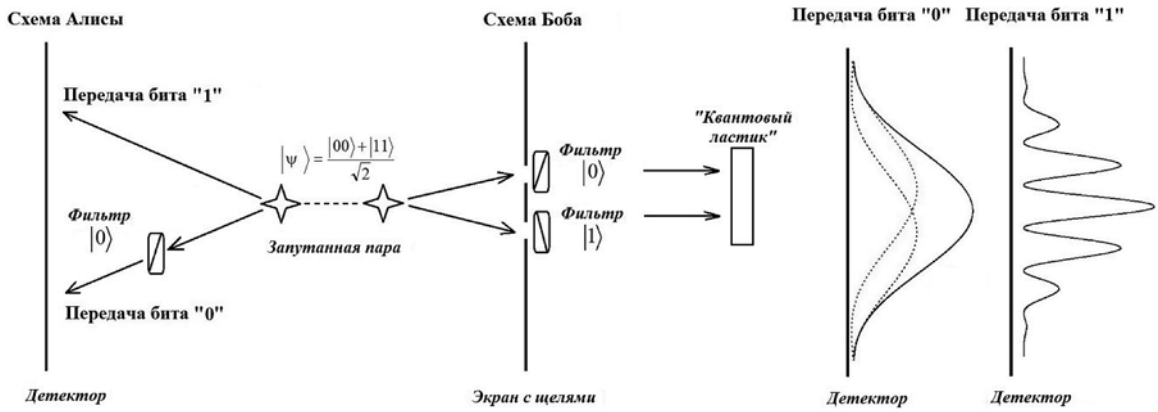


Рис. 7. Схема квантового телеграфа

цию своей частицы в двухщелевом эксперименте, а при действиях Алисы, направленных на передачу бита "0" классической информации, — нет.

Рассмотрим передачу Алисой бита "1" классической информации.

В данном случае, поскольку Алиса только детектировала свою частицу без ее измерения, частица Боба может с равной вероятностью "пройти" либо через первую, либо через вторую щель с фильтрами. При этом состояние частицы Боба становится определенным, $|0\rangle$ или $|1\rangle$ соответственно, в зависимости от того, через какую щель она "прошла". Однако установленный после экрана со щелями и фильтрами "квантовый ластик", в качестве которого можно использовать поляризатор, ориентированный под 45°, стирает информацию о выборе пути частицей. Данная ситуация аналогична рассмотренной в разд. 1, и ожидается *наличие* интерференционной картины (рис. 7).

Рассмотрим передачу Алисой бита "0" классической информации.

В данном случае, поскольку Алиса предварительно "пропустила" свою частицу через фильтр $|0\rangle$, состояние частицы Боба до прохождения экрана со щелями и фильтрами известно, а следовательно, она может "пройти" либо **только** через первую щель (если частица Алисы прошла фильтр), либо **только** через вторую щель (если частица Алисы была поглощена фильтром). При этом у частицы Боба отсутствует альтернатива выбора "пути". Данная ситуация аналогична рассмотренной в разд. 2, и ожидается *отсутствие* интерференционной картины.

4. Предложения по экспериментальной проверке квантового телеграфа

При разработке схемы квантового телеграфа были учтены эксперименты по наблюдению интерференционной картины от одиночных и запутанных пар фотонов, однако для подтверждения работоспособности предложенной технологии передачи информации требуется проведение натурных экспериментов. Поэтому в данном разделе приводятся предложения по экспериментальной проверке квантового телеграфа с использованием запутанных по поляризации пар фотонов. Схема предлагаемого к реализации эксперимента представлена на рис. 8 и включает в себя следующие основные элементы: лазер (с аттенюатором), нелинейный кристалл, экран со щелями, поляризационные фильтры (ПФ), "квантовый ластик" (КЛ) и детекторы. Эта схема близка к установке из работы [6]. Разница в том, что в качестве источника фотонов используется один из запутанных лучей, выходящих из нелинейного кристалла, а не луч лазера. Кроме того, на "пути" второго луча либо устанавливается (для передачи "0"), либо нет (для передачи "1") поляризационный фильтр.

Представленная на рис. 8 схема эксперимента обобщает исследования, анализ результатов которых представлен в разд. 1 и 2. Для генерации запутанных по поляризации пар фотонов в состоянии Бела $|\psi\rangle = \frac{|HH\rangle + |VV\rangle}{\sqrt{2}}$ предлагается использовать процесс спонтанного параметрического рассеяния типа II в вырожденном по частоте режиме в нелинейном кристалле типа BBO. В качестве фильтров, пропускающих только фотоны с заданной поляризацией $|H\rangle$ или $|V\rangle$, можно использовать специальные поляризационные фильтры. Для реализации функции "квантового

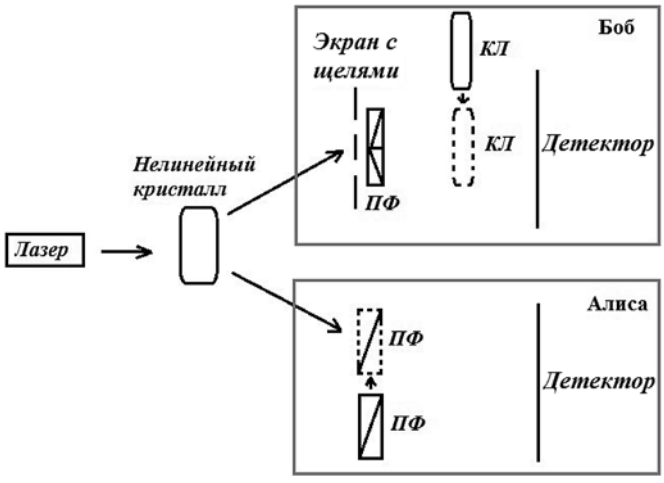


Рис. 8. Схема по экспериментальной реализации квантового телеграфа с использованием запутанных по поляризации пар фотонов

ластика", стирающего информацию о выборе "пути" фотоном Боба, предлагается применять поляризатор, ориентированный под углом 45°. Детектирование фотонов может быть реализовано, как с использованием однофотонных лавинных фотодиодов, имеющих квантовую эффективность ≈20 %, так и с помощью фотоумножителей с квантовой эффективностью 3...5 %.

При проведении эксперимента по реализации квантового телеграфа представляется важным осуществление следующих опытов.

1. "Пропускание" фотона Боба через экран со щелями без фильтров.

Ожидаемый результат эксперимента: *наличие* интерференционной картины, как и на рис. 2, *а*.

2. "Пропускание" фотона Боба через экран со щелями и фильтрами, при условии, что элемент, выполняющий функции "квантового ластика" *убран*, а Алиса просто детектирует свой кубит.

Ожидаемый результат эксперимента: *отсутствие* интерференционной картины, как и на рис. 2, *б*.

3. "Пропускание" фотона Боба через экран со щелями и фильтрами, при условии, что элемент, выполняющий функции "квантового ластика" *установлен*, а Алиса просто детектирует свой кубит.

Ожидаемый результат эксперимента: *наличие* интерференционной картины, как и на рис. 2, *в*.

4. "Пропускание" фотона Боба через экран со щелями и фильтрами, при условии, что элемент, выполняющий функции "квантового ластика" *установлен*, а Алиса предварительно пропускает свой кубит через поляризационный фильтр.

Ожидаемый результат эксперимента: *отсутствие* интерференционной картины, как и на рис. 2, *б*), но с менее интенсивными, чем в опыте 2 максимумами.

Заключение

В настоящей работе изложена основная идея новой технологии передачи информации, названной квантовый телеграф. Продемонстрирована схема побитовой передачи сообщения между Алисой и Бобом, в основе которой лежит принципиальная возможность наблюдения Бобом интерференционной картины, создаваемой одним из кубитов запутанной пары. Показано, что при передаче бита информации Алиса может руководствоваться стратегией "квантового ластика", а именно "стирать" или оставлять принципиально доступную информацию о выборе "пути" фотоном Боба. В свою очередь, у любого потенциального нарушителя (Евы), не имеющего доступ к множеству пар кубитов, распределенных между Алисой и Бобом, отсутствует возможность получения какой-либо информации о передаваемом сообщении.

Дополнительно отметим, что для восстановления состояния Белла $|\psi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$, заранее распределенного между Алисой и Бобом и разрушенного в процессе выполнения протокола, существует принципиальная возможность с применением операции *свопинга* (подкачки) провести дистанционную регенерацию, например, с использованием протокола восстановления состояний носителей-кубитов [19]. Предложенная технология может стать основой для разработки перспективных программно-аппаратных комплексов защищенных систем связи.

Список литературы

1. **Wengerowsky S., Joshi S. K., Steinlechner F.** et al. Entanglement distribution over a 96-km-long submarine optical fiber // PNAS. 2019. Vol. 116, No. 14. P. 6684—6688.
2. **Gisin N.** Entanglement 25 years after quantum teleportation: testing join measurements in quantum networks // Entropy. 2019. Vol. 21. P. 321.
3. **Aspelmeyer M., Böhm R. H., Gyatso T.** et al. Long-Distance Free-Space Distribution of Quantum Entanglement // Science. 2003. Vol. 301, No. 5633. P. 621—623.
4. **Алиев Ф. К., Бородин А. М., Васенков А. В.** и др. ATF-технология связи, основанная на использовании ресурса несепарабельных состояний квантовых систем // Наукоемкие технологии. 2015. Т.16, № 1. С. 65—78.
5. **Алиев Ф. К., Корольков А. В., Матвеев Е. А.** Несепарабельные состояния многокубитных квантовых систем / Под ред. Ф. К. Алиева. М.: Радиотехника. 2017. 320 с.
6. **Dimitrova T. L., Weis A.** A portable double-slit quantum eraser with individual photons // European Journal of Physics. 2011. Vol. 32. P. 1535—1546.
7. **Ribeiro P. H. S., Padua S., Machado da Silva J. C., Barbosa G. A.** Controlling the degree of visibility of Young's fringes with photon coincidence measurements // Physical Review A. 1994. Vol. 49, No. 5. P. 4176—4180.
8. **Strekalov D. V. Sergienko A. V., Klyshko D. N., Shih Y. H.** Observation of Two-Photon "Ghost" Interference and Diffraction // Phys. Rev. Lett. 1995. Vol. 74. P. 3600.
9. **Walborn S. P., Terra Cunha M. O., Padua S., Monken C. H.** Double-slit quantum eraser // Phys. Rev. A. 2002. Vol. 65. P. 033818.
10. **Kim Y.-H., Yu R., Kulik S.P, Shih Y., Scully M. O.** Delayed "Choice" Quantum Eraser // Physical Review Letters. 2000. Vol. 84, No. 1. P. 0031—9007.
11. **Graniger P., Roger G., Aspect A.** Experimental evidence for a photon anticorrelation effect on a beamsplitter // Europhys. Lett. 1986. Vol. 1. P. 173—179.
12. **Zhang Y., Roux F. S., Konrad T.** et al. Engineering two-photon high-dimensional states through quantum interference // Science Advances. 2016. Vol. 2, No. 2. P. e1501165
13. **Menzel R., Puhlmann D., Heuer A., Schleich W. P.** Wave-particle dualism and complementarity unraveled by a different mode // PNAS. 2012. Vol. 109. No. 24. P. 9314—9319.
14. **Zhang D.-J., Wu S., Li H.-G.** et al. Young's double-slit interference with two-color biphotons // Scientific Reports. 2017. Vol. 7. P. 17372.
15. **Tonomura A., Endo J., Matsuda T., Kawasaki T., Exawa H.** Demonstration of single-electron buildup of an interference pattern // Amer. J. Phys. 1989. Vol. 57. P. 117—120.
16. **Gahler R., Zeilinger A.** Wave-optical experiments with cery cold neutrons // Amer. J. Phys. 1991. Vol. 59. P. 316—324.
17. **Keith D. W., Eckstrom C. R., Turchette Q. A., Pritchard D. E.** An interferometer for atoms // Phys. Rev. Lett. 1991. Vol. 66. P. 2693—2696.
18. **Нильсен М., Чанг И.** Квантовые вычисления и квантовая информация. М.: Мир, 2006. 824 с.
19. **Матвеев Е. А.** Протокол восстановления состояний носителей-кубитов для формирования ключевой информации квантовой криптографической системы АКМ2017 // Системы высокой доступности. 2018. Т. 14, № 4. С. 73—78.

Quantum Telegraph

E. A. Matveev, eugene.cs@hotmail.com, Cryptosoft, Penza, 440026, Russian Federation

Corresponding author:

Matveev Evgeny A., Director, Cryptosoft, Penza, 440026, Russian Federation
E-mail: eugene.cs@hotmail.com

Received on June 11, 2019
Accepted on June 25, 2019

This article presents the main provisions of the new technology of transmitting and receiving data, called "quantum telegraph". The key idea of the technology is the observation of the interference pattern (or its absence) created by particles from a set of entangled pairs as they pass through the screen with two slits and filters. While developing it, we took into account the experiments on the observation of the interference pattern from single photons, entangled photon pairs, as well as studies related to the "erasure" of information about the choice of the path, known as a "quantum eraser". Moreover, in the context of the proposed technology, the most important are experimental data demonstrating that the fundamental possibility of obtaining information about the choice of a path by a particle destroys the interference pattern in any analogue of a double-slit experiment, and "erasing" information about the choice of a path leads to its restoration. The scheme of a bit-by-bit message transfer between Alice and Bob is demonstrated, the basis of which is the fundamental possibility that Bob can observe the interference pattern created by one of the quits of the entangled pair. It is shown that when transmitting a bit of information, Alice can be guided by the "quantum eraser" strategy, namely, "erase" or leave fundamentally accessible information about the choice of the path by Bob's photon. It was concluded that any potential intruder who does not have access to the multiple pairs of particles distributed between Alice and Bob does not have the possibility of obtaining any information about the transmitted message. That to restore the state of Bell, pre-distributed between Alice and Bob, and destroyed during the execution of the protocol, there is a fundamental possibility of using a swapping (swap) operation to conduct remote regeneration. The proposed technology can be used as a basis for the development of promising software and hardware complexes of secure communication systems.

Keywords: *quantum telegraph, quantum computations, quantum information, nonseparable (entangled) states, interference pattern on two slits*

For citation:

Matveev E. A. Quantum Telegraph, *Programmnaya Ingeneria,* 2019, vol. 10, no. 7—8, pp. 317—323

DOI: 10.17587/prin.10.317-323

References

1. **Wengerowsky S., Joshi S. K., Steinlechner F., Zichi J. R., Dobrovolskiy S. M., van der Molen R., Los J. W.N., Zwiller V., Versteegh M. A.M., Mura A., Calonico D., Inguscio M., Hübel H., Bo L., Scheidl T., Zelinger A., Xuereb A. Ursin R.** Entanglement distribution over a 96-km-long submarine optical fiber, *PNAS*, 2019, vol. 116, no. 14, pp. 6684—6688.

2. **Gisin N.** Entanglement 25 years after quantum teleportation: testing join measurements in quantum networks, *Entropy*, 2019, vol. 21, pp. 321.

3. **Aspelmeyer M., Böhm R. H., Gyatso T., Jennewein T., Kaltenbaek R., Lindenthal M., Molina-Terriza G., Roppe A., Resch K., Taraba M., Ursin R., Walther P., Zeilinger A.** Long-Distance Free-Space Distribution of Quantum Entanglement, *Science*, 2003, vol. 301, no. 5633, pp. 621—623.

4. **Aliev F. K., Borodin A. M., Vassenkov A. V., Matveev E. A., Zarkov A. N., Sheremet I. A.** ATF-technology of communication based on using the resource of entangled states of quantum systems, *Visokie Teknologgii*, 2015, vol. 16, no. 1, pp. 65—78 (in Russian).

5. **Aliev F. K., Korolkov A. V., Matveev E. A.** Nonseparable states of multiqubit quantum systems, Moscow, *Radioteknika*, 2017, 320 p. (in Russian).

6. **Dimitrova T. L., Weis A**. A portable double-slit quantum eraser with individual photons, *European Journal of Physics*, 2011, vol. 32, pp. 1535—1546.

7. **Ribeiro P. H. S., Padua S., Machado da Silva J. C., Barbosa G. A.** Controlling the degree of visibility of Young's fringes with photon coincidence measurements, *Physical Review A*, 1994, vol. 49, no. 5, pp. 4176—4180.

8. **Strekalov D. V. Sergienko A. V., Klyshko D. N., Shih Y. H.** Observation of Two-Photon "Ghost" Interference and Diffraction, *Phys. Rev. Lett,* 1995, vol. 74, pp. 3600.

9. **Walborn S. P., Terra Cunha M. O., Padua S., Monken C. H.** *Double-slit quantum eraser*, Phys. Rev. A, 2002, vol. 65, pp. 033818

10. **Kim Y.-H., Yu R., Kulik S. P., Shih Y., Scully M. O.** Delayed "Choice" Quantum Eraser, *Physical Review Letters*, 2000, vol. 84, no. 1, pp. 0031—9007.

11. **Graniger P., Roger G., Aspect A.** Experimental evidence for a photon anticorrelation effect on a beamsplitter, *Europhys. Lett,* 1986, vol. 1, pp. 173—179.

12. **Zhang Y., Roux F. S., Konrad T., Agnew M., Leach J., Forbes A.** Engineering two-photon high-dimensional states through quantum interference, *Science Advances*, 2016, vol. 2, no. 2, pp. e1501165.

13. **Menzel R., Puhlmann D., Heuer A., Schleich W. P.** Wave-particle dualism and complementarity unraveled by a different mode, *PNAS*, 2012, vol. 109, no 24, pp. 9314—9319.

14. **Zhang D-J., Wu S., Li H.-G. Wang H.-B., Xiong J., Wang K.** Young's double-slit interference with two-color biphotons, *Scientific Reports*, 2017, vol. 7, pp.17372.

15. **Tonomura A., Endo J., Matsuda T., Kawasaki T., Exawa H.** Demonstration of single-electron buildup of an interference pattern, *Amer. J. Phys*., 1989, vol. 57, pp. 117—120.

16. **Gahler R., Zeilinger A.** Wave-optical experiments with cery cold neutrons, *Amer. J. Phys.,* 1991, vol. 59, pp. 316—324.

17. **Keith D. W., Eckstrom C. R., Turchette Q. A., Pritchard D. E.** An interferometer for atoms, *Phys. Rev. Lett*, 1991, vol. 66, pp. 2693—2696.

18. **Nielsen M. A., Chuang I. L.** *Quantum Computation and Quantum Information*, Cambridge University Press, 2001, 824 p.

19. **Matveev E. A**. The protocol of the recovery of carrier-qubit states for the formation of key information in the quantum cryptographic system AKM2017, *Sistemy' vy'sokoj dostupnosti,* 2018, no. 4, pp. 73—78 (in Russian).

УДК 004.492.3 DOI: 10.17587/prin.10.324-333

А. А. Сковорода, программист, e-mail: nastya_jane@seclab.cs.msu.su,
Д. Ю. Гамаюнов, ст. науч. сотр, e-mail: gamajun@seclab.cs.msu.su,
Московский государственный университет имени М. В. Ломоносова

Динамический анализ мобильных приложений

Рассмотрена задача автоматического обнаружения вредоносных Android-приложений. Предлагаемое решение содержит метод построения динамических моделей поведения приложений, а также их сравнения и автоматической классификации с использованием алгоритмов машинного обучения. Наименьшая доля ложных срабатываний (0,5 %) была получена по результатам тестовых испытаний программной реализации метода при использовании алгоритма градиентного бустинга. При этом правильно было классифицировано 85 % вредоносных приложений из контрольной выборки.

Ключевые слова: *Android-приложения, динамический анализ, эмулятор Android, автоматизация взаимодействия с пользователем, API-вызовы, привилегии, динамическая модель поведения, машинное обучение*

Введение

В последнее десятилетие существенно возрос интерес пользователей и разработчиков программного обеспечения к мобильным устройствам. Наиболее распространенными являются устройства на платформе Android, для которой регулярно разрабатывается и публикуется большое число новых приложений. Например, общее число приложений, представленных в Google Play, превысило 3 млн в конце 2017 г. [1]. Вместе с тем остается высоким и риск распространения вредоносных мобильных приложений, особенно приобретаемых на неофициальных рынках. Поэтому проблема эффективного обнаружения вредоносных мобильных приложений актуальна и привлекает внимание исследователей по всему миру.

Одним из направлений в области исследований по обнаружению и предотвращению активности вредоносных мобильных приложений является динамический анализ. При таком подходе поведение подконтрольных приложений анализируется исходя из трасс, которые получаются в ходе их исполнения или эмуляции исполнения.

В основном используемые технологии динамического анализа мобильных приложений заимствованы из более общей задачи обнаружения вредоносных приложений (ВП). Однако принимаются во внимание некоторые детали функционирования мобильных приложений, в частности, их ориентированность на взаимодействие с пользователем. Динамический анализ по сравнению со статическим позволяет более точно описывать поведение анализируемых приложений. Однако динамический анализ имеет и ряд недостатков. К их числу относятся:

- сложность получения полного покрытия программного кода;
- сложность создания реалистичной среды выполнения, возможное детектирование анализатора;
- ресурсоемкость, а именно — большие затраты ресурсов оперативной памяти и процессорного времени для эмуляции;
- большие временные затраты на анализ.

Первые два из перечисленных недостатков в той или иной степени преодолеваются в некоторых средствах динамического анализа, описанных в литературе. Вопросы временных затрат решают только за счет увеличения вычислительных ресурсов анализатора.

В настоящей работе описан метод построения динамических моделей Android-приложений, а также алгоритм сравнения и автоматической классификации приложений на основе таких моделей. Для реализации построения динамических моделей авторы использовали средства эмуляции приложений эмулятора qemu, который входит в состав средства разработки мобильных приложений Android SDK. Для обеспечения покрытия кода приложений было использовано программное средство DroidBot [2]. Такая схема динамического анализа наиболее эффективна в качестве одной из проверок механизма фильтрации рынков мобильных приложений, так как не требует большого количества ресурсов для построения моделей. Предложенные динамические модели могут быть использованы исследователями в области безопасности и как отдельные структуры, облегчающие понимание функционирования приложения.

1. Обзор исследований по теме динамического анализа Android-приложений

Одной из первых систем, предложенных для динамического анализа Android-приложений, была система TaintDroid [3], созданная В. Энк и др. в 2010 г. Система проводит taint-анализ приложения сразу на

четырех уровнях: переменных, методов, сообщений и файлов. В ходе такого анализа помечаются данные, которые берут начало от конфиденциальных ресурсов. Основная цель при таком подходе заключается в определении того, что данные являются производными от конфиденциальных, перед тем как они покинут устройство по сети, перейдя к злоумышленникам. Для организации такого анализа в TaintDroid модифицируется код операционной системы (ОС) Android. Программное средство Droidbox [4], разработанное в рамках программы Google Summer of Code в 2011 г., использует этот модифицированный образ ОС Android4.1. Кроме информации об утечках данных, это средство позволяет получать трассы действий (операции чтения/записи в файлы, динамическая загрузка кода и др.), совершенных анализируемыми приложениями. Обе системы — TaintDroid и Droidbox не предполагают автоматизированного взаимодействия с анализируемым приложением, эта задача возлагается на пользователя системы. В рамках программы Google Summer of Code в 2015—2017 гг. для автоматизации тестирования и анализа Android-приложений было разработано программное средство DroidBot [2]. Это средство посылает на устройство/в эмулятор различные события, основываясь на текущей иерархии элементов графического интерфейса приложения. В системе при этом используются различные алгоритмы обхода состояний приложения. Выходными данными DroidBot являются данные о переданных событиях и описания состояний приложения в виде наблюдаемой иерархии классов, образующих графическое представление. С помощью дополнительных ключей запуска также можно получить данные профилировщика, содержащие реакцию на посылаемые события.

В 2012 г. была представлена система SmartDroid [5], предназначенная для автоматического определения последовательности действий для работы с графическим интерфейсом приложения. Такая последовательность приводит к проявлению и возможности определения его вредоносной активности. Система SmartDroid сначала строит граф вызовов функций (*Function Call Graph*) и граф активностей (*Activities Call Graph*) анализируемого приложения на основе его smali-кода, т. е. декомпилированного представления приложения. Затем выполняет динамический анализ для получения последовательности событий графического интерфейса, приводящих к выполнению "опасного" API (*Application Programming Interface*). Динамический анализ включает обход элементов графического интерфейса с отсечением тех порождаемых активностей, которые не приводят к выполнению "опасного" API. SmartDroid модифицирует эмулятор Android для сохранения исполняемых API-вызовов и их параметров в файле журнала. Эта система показала свою эффективность в выявлении предусловий вредоносного поведения некоторых приложений.

В средстве DroidRanger [6] 2012 г. применяется комбинация методик статического и динамического анализа для классификации приложения как вредоносного или легитимного. Динамический анализ здесь представляет собой запись в журнал API-вызовов с конкретными параметрами и системных вызовов, такой анализ выполняется для приложений с динамической загрузкой кода, либо использующих нативный код. Интерпретация результатов динамического анализа — анализ записей в журнале — предполагает рассмотрение экспертами и не является автоматизированной.

Ю. Жанг и др. в 2013 г. реализовали систему VetDroid, использующую схему динамического анализа с учетом привилегий [7]. Анализируемое приложение загружается в драйвер приложений (*Application Driver*), который извлекает активности (*Activities*) и сервисы (*Services*), объявленные в манифесте приложения, и затем исполняет их в виртуальном окружении в течение некоторого времени. Для имитации пользовательских действий используется Android Monkey. Драйвер приложений порождает события (новое SMS, изменившееся местоположение и др.), если для них прописан обработчик. Тем не менее полное покрытие поведения программы все-таки не гарантируется системой. Модуль анализа привилегий извлекает все E-PUPs (точки явного использования привилегий, возникающие при запрашивании ресурса) и I-PUPs (точки неявного использования привилегий), а также связи между ними. При этом I-PUPs определяются на функциональном уровне taint-анализа с учетом запрашиваемых привилегий. Обнаруженное поведение сохраняется в файл журнала модулем Log Tracer. Файл журнала анализируется модулем Behavior Profiler, который строит граф вызовов функций для дальнейшего отложенного анализа. Система VetDroid модифицирует ядро Linux, драйвер Binder и виртуальную машину Dalvik для проведения такого анализа.

В 2013 г. была представлена многомодульная система динамического анализа AppsPlayground [8], построенная на основе TaintDroid [3]. Помимо taint-анализа, унаследованного от TaintDroid, в AppsPlayground реализовано отслеживание опасного API и обнаружение эксплоитов. В системе реализовано автоматизированное взаимодействие с анализируемым приложением, алгоритм обхода упрощен по сравнению с используемым в DroidBot: сначала с учетом контекста заполняются виджеты, ожидающие какого-то входа, далее осуществляется прокрутка для тех элементов, к которым это применимо, после этого обрабатываются все остальные элементы. В системе AppsPlayground также уделяется внимание созданию более реалистичной среды для анализа.

В 2014 г. А. Шабтай и др. предложили подход к обнаружению ВП, основанный на анализе трафика [9]. Метод нацелен на обнаружение самообновляющихся приложений, которые проявляют вредоносную активность спустя некоторое время после установки. Например, такое может происходить после обновления приложения или после динамической загрузки

скомпилированного кода. Наблюдаемое поведение системы может быть классифицировано как нормальное или аномальное на основе подсчитанной модели статистики нормального трафика. Для классификации использовались алгоритмы Decision Table и REPTree, поскольку они показали наилучшие результаты. Если последовательность состояний приложения классифицируется как аномальная, анализируемое приложение считается вредоносным. Данный метод обнаружения реализован в виде Android-приложения, работающего на устройстве пользователя и сообщающегося с удаленным сервером.

Система CopperDroid [10], представленная в 2015 г., выполняет динамический анализ на модифицированной версии эмулятора Android. В ходе такого анализа отслеживаются системные вызовы, исполняемые приложением. Информация о системных вызовах обрабатывается "сортирующим Оракулом" (*unmarshalling Oracle*) для извлечения высокоуровневых абстракций — межпроцессных взаимодействий и вызовов удаленных процедур (IPC/ICC и RPC). В системе реализован анализ потока данных для учета зависимостей между наблюдаемыми системными вызовами. Анализ включает также стимуляцию приложения событиями, вызывающими вредоносное поведение. Такие события получаются в процессе статического анализа кода приложения и его манифеста.

Система GlassBox [11], представленная в 2016 г., предполагает использование реальных устройств для анализа Android-приложений. В системе собираются трассы API-вызовов Android Framework (авторы называют их Java-вызовами), системных вызовов, а также сетевые запросы. Для автоматизации работы с приложениями, также как в DroidBot и AppsPlayground, используются методики обхода элементов графического интерфейса с учетом контекста. Учитывается также информация об обработчиках, заданных в манифесте, и некоторый набор возможных скрытых обработчиков, типичных для вредоносных приложений. Предлагаемая организация анализа исключает реальное исполнение вредоносных запросов за счет особенностей сетевой инфраструктуры.

В табл. 1 приведены характеристики упомянутых выше методов, наибольший интерес из которых представляют выходные данные. Для каждого из методов в таблице в скобках указан год публикации работы, в которой он был описан. Информация об обнаруженных вредоносных приложениях предоставляется только в работе [9]. В работах [3, 8] в качестве результатов анализа выдаются обнаруженные утечки данных или известные эксплоиты. Остальные методы предоставляют только информацию, которая нуждается в дальнейшей интерпретации в ходе ручного анализа или автоматизированных методах классификации. Следует также отметить, что для большинства рассмотренных методов исходный код недоступен, для метода [10] web-интерфейс недоступен в настоящий момент. В данной работе описаны динамические модели поведения, которые могут быть использованы и для ручного анализа, и для автоматической классификации.

Таблица 1

Характеристики методов динамического анализа Android-приложений, предложенных в литературе

Метод	Организация анализа	Выходные данные	Доступ к анализу/коду
TaintDroid (2010)	На устройстве/эмуляторе с модифицированной ОС Android	Сообщения об утечках данных	Доступны образы модифицированной ОС
Droidbox (2011)	На устройстве/ эмуляторе с модифицированной ОС Android	Трассы действий приложения	Доступен исходный код
DroidBot (2017)	На устройстве/эмуляторе	Граф состояний приложения с переходами по событиям, скриншоты, информация о состояниях	Доступен исходный код
SmartDroid (2012)	Эмулятор с модифицированной ОС Android	Последовательность действий для отображения вредоносной активности	Нет
DroidRanger (2012)	Эмулятор с модифицированной ОС Android	Журнал API-вызовов и системных вызовов	Нет
VetDroid (2013)	Эмулятор с модифицированной ОС Android	Поведенческие паттерны в виде связи E-PUPs и IPUPs	Нет
AppsPlayground (2013)	Эмулятор с модифицированной ОС Android	Утечки данных, журнал API-вызовов, информация об эксплоитах	Код высылают по запросу
Анализ трафика (2014)	Приложение на устройстве + удаленный сервер	Иформация об обнаруженных ВП	Нет
CopperDroid (2015)	Отладчик gdb + эмулятор с модифицированной ОС Android	Журнал системных вызовов, IPC/ICC и RPC	Web-интерфейс для анализа
GlassBox (2016)	Мобильные устройства + удаленные серверы	Журнал вызовов, системных вызовов, сетевых запросов	Нет

2. Постановка задачи исследования

В настоящей работе представлено решение задачи мультиклассовой классификации мобильных приложений. Пусть X — множество всевозможных .apk-файлов Android-приложений; $Y = \{Y_1, ..., Y_{N_Y}, Y_b\}$ — множество классов, включающее в себя N_Y семейств вредоносных приложений и класс Y_b легитимных приложений. Существует f: $X \to Y$ — сюръективное отображение, для которого известны значения $f(X_1)$, ..., $f(X_K)$ для аргументов $X_{known} = \{X_1, ..., X_K\}$, f представляет собой экспертную оценку. Требуется построить алгоритм, вычисляющий $g(x) \approx f(x) \forall x \in X$. Для оценки качества предложенного алгоритма используются характеристики, описанные в разделе "Результаты".

Для решения поставленной задачи в работе вводятся динамические модели поведения Android-приложений, на основе которых строятся векторы признаков, использующихся для классификации.

3. Динамические модели поведения Android-приложений

Определим множество $S = \{s_1, ..., s_{N_s}\}$ как множество возможных событий (также именуемых "входами" или стимулами), посылаемых приложению, и множество $R = \{r_1, ..., r_{N_r}\}$ как множество возможных реакций приложения. Определим R^* как множество всевозможных конечных последовательностей реакций, включая последовательность нулевой длины.

Множество возможных событий S состоит из касаний экрана в некоторых заданных точках, а также длительных касаний (*long-touch*), жестов смахивания, нажатий кнопок ("домой", "назад"), ввода текста с клавиатуры, широковещательных системных сообщений. Множество реакций может быть задано двумя различными способами: API-вызовами фреймворка Android и *действиями* приложения. Концепция совершенных действий взята из средства DroidBox, в котором используется модифицированный образ ОС Android с добавленной записью в журнал в момент совершения некоторых действий. Список отслеживаемых совершенных действий в DroidBox:

- утечки данных;
- операции чтения/записи;
- операции доступа к файлу;
- сетевые операции (установление соединений, отправление/получение данных);
- операции запуска сервисов;
- криптографические операции;
- телефонные звонки и отправление SMS;
- операции динамической загрузки кода.

Динамической моделью поведения Android-приложения будем называть помеченный мультиорграф $G = (V, E)$ со множеством вершин $V = \{v_1, ..., v_n\}$ и множеством дуг $E = \{e_1, ..., e_m\}$, такой что

- $\forall v \in V$ соответствует состоянию приложения — совокупности элементов графического интерфейса, а также запущенных компонентов приложения и внешней среды; во множестве V также выделены подмножества V_{start} и V_{stop} начальных и конечных вершин;
- $\forall e = (v_i, v_j) \in E$ соответствует переходу между состояниями приложения, $\forall e \in E$ задана пометка $\{s, r\}$, $s \in S$ — событие, по которому осуществляется переход между состояниями; $r \in R^*$ — наблюдаемая при переходе реакция.

Соответственно, динамические модели поведения, в которых реакция задана в виде совершенных API-вызовов, будем называть *моделями API-вызовов*, а динамические модели поведения, в которых реакция задана в виде совершенных действий, будем называть *моделями действий*.

Следует отметить, что динамические модели поведения Android-приложений можно также рассматривать как недетерминированные конечные автоматы со множеством состояний, определенным как множество вершин графа V (начальные и конечные состояния будут соответствовать множествам V_{start} и V_{stop}), и отношением переходов, соответствующим множеству дуг E, а также алфавитом распознаваемого языка, заданным как множество событий S. Однако для определения сходства Android-приложений мы оперируем графовым представлением таких конечных автоматов, пренебрегая их динамической сущностью.

Рисунок 1 иллюстрирует пример динамической модели поведения, вершины черного цвета соответствуют вершинам $v \in V_{start}$, а вершины, изображенные как вложенные окружности, — конечным вершинам $v \in V_{stop}$. Начальных вершин может быть несколько, поскольку совокупность элементов графического интерфейса на момент запуска приложения может зависеть от сохраненных настроек и меняться при

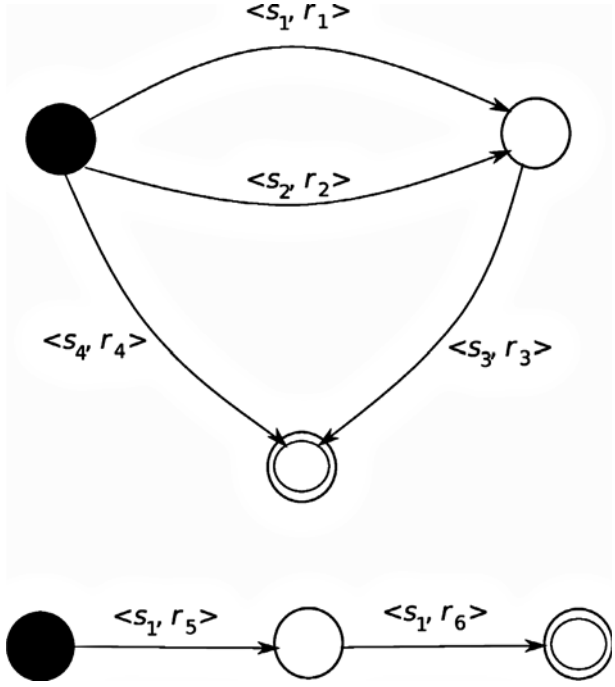


Рис. 1. Пример динамической модели поведения

перезапуске. Конечные вершины соответствуют вершинам, в которых приложение было остановлено умышленно или произошел аварийный останов.

4. Построение моделей

Для составления моделей, в частности для автоматизации взаимодействия с пользователем и для обхода возможных состояний приложения, было использовано программное средство DroidBot. Это средство было рассмотрено в разд. 1. В настоящем разделе рассмотрим подробнее некоторые детали его функционирования.

DroidBot — это программное средство для формирования тестовых входов Android-приложений. Оно разрабатывалось с 2015 г. и поддерживается в настоящее время. Средство DroidBot использует отдельное приложение, включающее Accessibility Service для получения иерархии графических компонентов, отображаемых на экране. В более ранней версии DroidBot для этой цели использовался ViewServer [12]. Полученная иерархия классов образует *состояние* приложения. Состояния считаются идентичными, если совпадает основная Activity (*foreground Activity*), а также для каждого элемента графического интерфейса (View) совпадают параметры *class, resource_id, text, enabled, checked, selected.* Для текущего состояния, в зависимости от режима работы, проводится обход элементов с применением к ним возможных допустимых операций. Такими операциями могут быть касания элементов или ввод текста. Если ни для одного элемента не найдено ни одной допустимой операции, посылаются некоторые широковещательные сообщения, на которые зарегистрирована реакция, либо нажимаются клавиши. У DroidBot шесть режимов работы, задающих следующий алгоритм обхода (выбираются с помощью ключа *policy*):

- *none*: режим предполагает взаимодействие пользователя с устройством/эмулятором вручную, события не порождаются автоматически;
- *monkey*: для порождения событий используется команда *adb shell monkey*;
- *dfs_naive*: для каждого из состояний перебираются все элементы интерфейса, которым можно послать события; после исследования всех элементов выбирается элемент, меняющий текущую Activity;
- *bfs_naive*: режим аналогичен *dfs_naive,* за исключением выбора элемента при смене Activity, предпочтительным является "возврат назад";
- *dfs_greedy*: режим отличается более тщательным выбором следующего исследуемого элемента, сначала отбираются все элементы с истинным значением свойства *enabled — enabled*-элементы, далее перебираются все *clickable*-элементы, затем *scrollable*, *checkable* и *long-clickable*, а потом *editable*;
- *bfs_greedy:* режим аналогичен *dfs_greedy,* за исключением выбора элемента при смене Activity.

Для формирования динамических моделей использовался режим работы *dfs_greedy*, поскольку его порядок обхода элементов графического интерфейса задан более точно, а также минимизировано число "возвратов назад" по сравнению с режимом *bfs_greedy.*

Средство DroidBot также поддерживает создание более реалистичного окружения. Эта операция включает в себя добавление контактов устройства, эмуляцию входящих или исходящих звонков, а также SMS, задание GPS-координат. Отметим, что в работе, результаты которой представлены в настоящей статье, эти опции для создания моделей не использовались, что может быть направлением для дальнейшего исследования.

Для приложений, использованных в тестовых испытаниях, полнота покрытия кода при использовании DroidBot для взаимодействия с приложением составила в среднем 22,6 % (21 % для легитимных и 24 % для вредоносных приложений). Полнота покрытия кода оценивалась как отношение числа вызванных попарно различных методов приложения и общего числа методов приложения, полученных в процессе декомпиляции. Таким образом, для многих приложений не удалось построить модели, полностью отражающие их функциональные возможности. Однако даже при такой полноте покрытия кода предложенный метод классификации показал свою эффективность.

Построение моделей API-вызовов. Для построения моделей API-вызовов использовалась опция профилирования в DroidBot (*-use_method_profiling full*). При использовании данной опции DroidBot сохраняет трассы вызовов, соответствующие отправленным событиям. Данные трассы имеют стандартизованный формат и могут быть просмотрены в графическом виде в приложении Traceview, входящем в состав Android Studio [13]. Для задания реакции в модели API-вызовов из трасс были выделены участки, относящиеся к методам анализируемого приложения[1]. Внутри таких участков формировались цепочки API-вызовов Android Framework в порядке их следования без учета вложенных API-вызовов Android Framework.

Построение моделей действий. Для генерации модели действий были собраны образы Android 4.1.1 (r6) для архитектуры ARM и Android 7.0.0 (r1) для архитектуры процессора x86 с добавлением записи в журнал совершенных действий по аналогии с программным средством DroidBox [4]. Были перенесены все изменения, внесенные для DroidBox, кроме изменений, связанных с taint-анализом и обнаружением утечек данных. Необходимые изменения доступны в репозитории [14]. Модели действий были построены на модифицированных образах согласно схеме, изображенной на рис. 2 (см. четвертую сторону обложки). Из записанных действий были выделены реакции на поданные на вход события согласно времени их появления.

[1] При этом выделяются только собственные методы анализируемого приложения, не являющиеся статически скомпилированными сторонними библиотеками.

5. Сравнение динамических моделей поведения

Согласно принятой схемы анализа динамическая модель поведения анализируемого приложения сравнивается с динамическими моделями поведения ВП. В данном разделе рассмотрим алгоритм сравнения двух моделей приложений, одна из которых соответствует анализируемому приложению, а другая — известному ВП. Полная схема анализа дана в разд. 6.

Изначально сравниваемые модели преобразуются во множество цепочек реакций. Для этого используется следующий алгоритм.

- Множество цепочек реакций R_{app} инициализируется пустым множеством.
- От каждой начальной вершины $v \in V_{start}$ вызывается рекурсивная функция обхода, использующая поиск в глубину. Аргументом функции также является текущая цепочка реакций, заданная как пустая цепочка для начальных вершин.
- Функция обхода реализуется следующим образом.

Шаг 1. Рассматривается текущая вершина — аргумент вызова функции. Если она помечена, переходим к шагу 4. Текущая вершина помечается.

Шаг 2. В текущую цепочку реакций добавляются реакции-пометки всех дуг, образующих петли в текущей вершине.

Шаг 3. Для текущей вершины рассматриваются все дуги, исходящие из нее, и вызывается функция обхода соответствующих смежных вершин. В качестве параметра функции обхода также передается текущая цепочка реакций, дополненная реакцией-пометкой рассматриваемой дуги.

Шаг 4. Выделение текущей цепочки реакций во множество R_{app} проводится в конечных вершинах $v \in V_{stop}$, а также в помеченных вершинах, т. е. в момент обнаружения цикла.

Шаг 5. С текущей вершины снимается пометка.

Цепочки реакций приложений R_{app_1} и R_{app_2} попарно сравнивают, для каждой пары вычисляют значения длины наибольшей общей подпоследовательности. Также вычисляют отношение этой длины и длины цепочки во вредоносной модели. Максимальное среди всех пар значение длины наибольшей общей подпоследовательности S_{ab}, а также относительная величина S_{rel}, соответствующая ей, образуют *показатели сходства* сравниваемых моделей.

6. Предлагаемая схема анализа

Далее рассмотрим предлагаемую и реализованную авторами схему анализа приложений.

По построенным моделям приложений строятся векторы признаков для дальнейшей их классификации. Для каждого из семейств вредоносных приложений выделяется некоторое число *основных моделей*, исходя из результатов кластеризации по API-вызовам. Для составления вектора признаков приложения использовались значения *показателей сходства* моделей действий этого приложения и основных моделей семейств ВП. Кроме этих признаков были использованы признаки наличия/отсутствия фактов использования некоторого множества API-вызовов F, которое применяли для создания *моделей API-вызовов Android Framework* в работе [15]. Эта часть вектора признаков была построена по динамическим моделям API-вызовов. В формировании вектора признаков участвовали также базовые характеристики приложений, а именно — число цепочек в модели действий, максимальная и средняя длины цепочек.

Таким образом, векторы признаков для анализируемых приложений имели следующий вид:

$$V_{app} = \left(L_{app}, L_{\max_{seq}}, L_{avg_{seq}}, S_{ab_1}, S_{rel_1}, \ ..., \ S_{ab_N}, \ S_{rel_N}, A_1, \ ..., A_M\right),$$

где N — суммарное число основных моделей семейств; M — число элементов в выбранном множестве F; S_{ab_i}, S_{rel_i} — показатели сходства выбранного приложения с моделью i; A_i — бинарное значение, соответствующее использованию API-вызова $f_i \in F$; L_{app} — число цепочек в модели действий, $L_{\max_{seq}}$, $L_{avg_{seq}}$ — максимальная и средняя длины цепочек соответственно.

Для классификации векторов признаков применяли: метод k ближайших соседей (KNN); метод опорных векторов (SVM); метод опорных векторов с ядром (Kernel SVM); метод дерева решений (Decision Tree), а также ансамбля деревьев решений (Random Forest) и градиентного бустинга. Наилучшие результаты классификации были получены при использовании в качестве классификатора метода градиентного бустинга. Результаты для всех классификаторов представлены в разд. 8.

Выделение основных моделей. Поскольку приложения, объединенные в одно семейство ВП, могут очень сильно отличаться как в реализации вредоносных функций, так и в реализации каких-то дополнительных функций, из семейств вредоносных приложений были выделены *основные модели* с помощью кластеризации. В качестве набора признаков было принято использование приложением *защищенных* API-вызовов, для исполнения которых приложение должно иметь некоторые привилегии. Таким образом, векторы признаков, использованные для кластеризации, являются двоичными векторами длины $K < M$, каждый из компонентов которых соответствует использованию определенного API-вызова $A \in F$.

В ходе иерархической кластеризации с использованием в качестве метрики расстояния между объектами евклидова пространства, а в качестве критерия объединения кластеров — минимизации среднего квадратичного отклонения, были выделены *основные модели*. Суммарное число таких моделей составило $N = 17$ для выбранных экспериментальных данных.

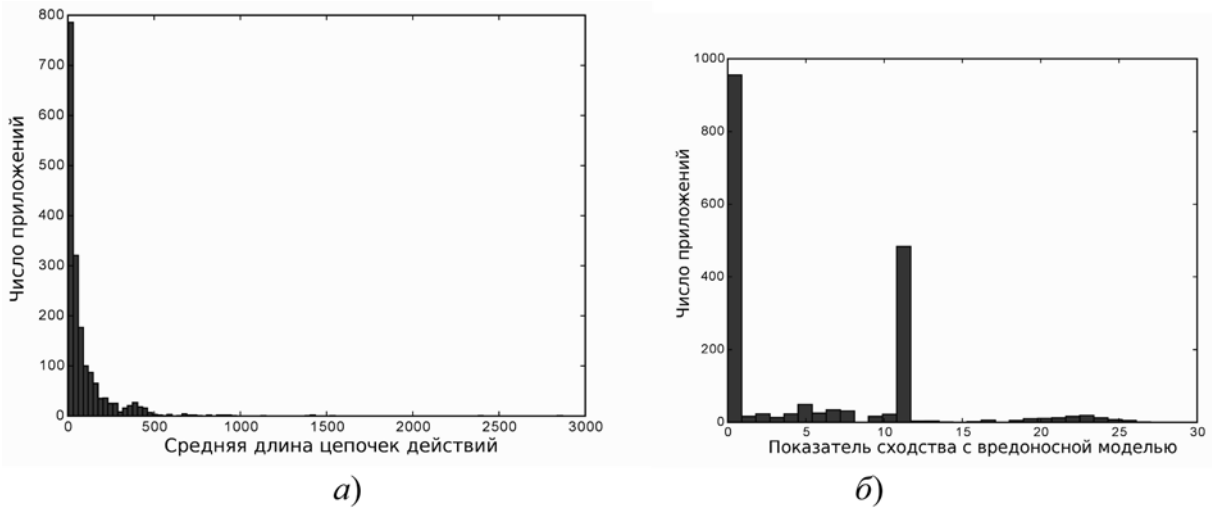


Рис. 4. Распределение признаков, использованных для классификации:

а — гистограмма распределения средней длины цепочек в моделях; *б* — гистограмма распределения показателя сходства с одной из моделей семейства Anserverbot

Для каждого из кластеров внутри семейства в качестве основных моделей были взяты объекты, наиболее близкие к центроидам кластеров.

На рис. 3 (см. четвертую сторону обложки) показаны итоговые дендрограммы для использованных в тестовых испытаниях семейств ВП. Каждый из образованных кластеров внутри семейств выделен в дендрограммах отдельным цветом. Согласно дендрограммам, наибольшие различия, соответствующие евклидову расстоянию между кластерами, были выявлены среди приложений семейства Opfake. Для остальных семейств расстояния между векторами признаков были значительно меньше.

7. Программа тестовых испытаний

Были проведены тестовые испытания предлагаемого авторами метода на семействах вредоносных приложений Anserverbot, DroidDream, Opfake и Plankton, а также коллекции легитимных приложений из Google Play. Для тестовых данных были отобраны только те вредоносные приложения, которые содержали в модели API-вызовов хотя бы один *защищенный* вызов. Таким образом были отобраны приложения, проявившие некоторую подозрительную активность. Для каждой группы тестовых данных ниже отмечено число экземпляров приложений в группе.

Anserverbot	145
DroidDream	71
Opfake	170
Plankton	136
Легитимные	1565

Выбор алгоритма классификации. Набор признаков, выделенный для решения задачи классификации, является неоднородным. Часть таких признаков определяется бинарными значениями, т. е. имеет распределение Бернулли с различными параметрами. Другие признаки имеют распределения, не являющиеся ни одним из известных в математической статистике распределений. На рис. 4 приведены примеры распределения некоторых признаков.

В связи с отмеченными выше обстоятельствами не все используемые для задач классификации методы машинного обучения подходят для поставленной задачи. Например, наивный байесовский классификатор в библиотеке scikit.learn имеет три доступных вида классификатора, отличающихся предположением относительно распределения признаков. Доступны классификаторы для набора признаков, имеющих нормальное, биномиальное и мультиномиальное распределение. При этом предполагается, что все признаки имеют заданное распределение. В нашем случае это условие не выполнено. Кроме того, не выполнено "наивное" предположение байесовского классификатора о независимости используемых признаков. Даже после отбора признаков это условие не гарантируется.

Метод k ближайших соседей часто используется для получения предварительных результатов перед использованием сложных классификаторов. Одним из основных недостатков метода является то обстоятельство, что он может деградировать от большого числа признаков, поскольку они влияют на оценку расстояния.

Метод опорных векторов (SVM) также имеет существенные ограничения: при использовании линейного ядра предполагается линейная разделимость классов. Для данного метода применяются различные функции ядра, которые переводят признаковое пространство в пространство более высокой размерности, в котором классы будут линейно разделимы.

Наиболее универсальными алгоритмами классификации являются те, которые основаны на решаю-

Таблица 2

Оптимальные гиперпараметры классификации

Классификатор	Параметры
KNN	k=3; metric = “euclidean”; weights = “uniform”
SVM (линейная классификация)	C = 1000
SVM (общая)	kernel = “poly”; gamma = 0,8; degree = 3
Дерево решений	criterion = “entropy”; max_features = 0,75; min_samples_leaf = 3
Ансамбль деревьев решений	n_estimators = 20; criterion = “entropy”; min_samples_leaf = 2; max_features = 0,75
Градиентный бустинг	n_estimators = 20; colsample_bytree = 0,5; max_depth = 5; booster = “dart”

щих деревьях. Наиболее простая реализация — единственное дерево решений. Ансамбль деревьев решений может дать лучшие результаты за счет того, что усредняется результат по всем деревьям решений. Деревья из ансамбля строятся на разных подмножествах тестовой выборки и также с разным выбором признаков, на основе которых проводится разбиение. Еще одним алгоритмом, основанным на композиции деревьев решений, является градиентный бустинг. В отличие от ансамбля деревьев решений, в нем осуществляется композиция не случайных деревьев, а специально подобранных, итеративно минимизирующих итоговую функцию потерь.

Для тестовых испытаний в решении поставленной задачи были выбраны все перечисленные выше алгоритмы, за исключением наивного байесовского классификатора. Алгоритмы, заведомо уступающие остальным по качеству классификации (*k* ближайших соседей, дерево решений), было решено использовать для сравнения результатов.

Отбор признаков и подбор оптимальных параметров алгоритмов. Для отбора признаков были использованы оценки важности (*feature ranking*), полученные на основе решающих деревьев [16]. Из изначального набора признаков были исключены три признака, являющиеся показателями сходства с выбранными основными моделями. Два из этих признаков были показателями сходства со слишком короткой основной моделью. Еще один признак имел распределение, дублирующее распределение другого признака — показателя сходства по длине для той же основной модели. Были также исключены 46 признаков, соответствующих использованию некоторых API-вызовов.

Оптимальные гиперпараметры алгоритмов машинного обучения были подобраны в ходе серии предварительных тестовых испытания с перекрестной проверкой и разбиением тестовых данных на четыре части. Они перечислены в табл. 2. При подсчете лучших параметров результаты классификации сравнивались по характеристике доли правильных ответов (*accuracy*).

8. Результаты тестовых испытаний

После получения оптимальных гиперпараметров классификации были проведены тестовые испытания для выбранных алгоритмов машинного обучения. Каждое тестовое испытание представляло собой перекрестную проверку (*k-fold cross validation*) с разбиением множества векторов признаков на четыре части. В качестве итоговых результатов, представленных в табл. 3, были взяты усреднения по 100 тестовым испытаниям[1].

Характеристики, используемые для сравнения классификаторов. Таблица 3 содержит следующие характеристики качества классификаторов.

- Доля ошибок первого рода или ложных срабатываний (FPR) определяется как отношение числа легитимных приложений, неверно классифицированных как вредоносные, и общего числа легитимных приложений.
- Доля ошибок второго рода (FNR) определяется как отношение числа вредоносных приложений, неверно классифицированных как легитимные, и общего числа вредоносных приложений.
- Неверная метка семейства определяется как отношение числа вредоносных приложений, которые были классифицированы как вредоносные, но настоящая метка семейства которых не совпала с меткой семейства, выданной классификатором, и общего числа вредоносных приложений.
- Доля правильных ответов (*accuracy*) определяется как отношение числа верно классифицированных приложений к общему числу приложений, участвовавших в классификации.
- Точность (*precision*) определяется как отношение числа вредоносных приложений, верно классифицированных как вредоносные, и общего числа приложений, классифицированных как вредоносные.
- Полнота (*recall*) определяется как отношение числа вредоносных приложений, верно классифицированных как вредоносные, и общего числа вредоносных приложений.

Характеристики точности, полноты и ошибок второго рода заданы как характеристики оценки задачи бинарной классификации. Для исходной задачи мультиклассовой классификации эти характеристики определяются отдельно для каждого из классов — в нашем случае семейств вредоносных приложений. Для упрощения восприятия было решено не включать данные характеристики в таблицу сравнения.

[1] Каждое из тестовых испытаний отличается распределением векторов признаков в разбиении перекрестной проверки.

Таблица 3

Результаты классификации, %

Классификатор	Характеристики качества					
	FPR	FNR	Неверная метка семейства	Accuracy	Precision	Recall
KNN (k = 3)	1,6	9,8	0	96,5	94,5	90,2
SVM	2,0	7,1	0,4	96,7	93,1	92,9
SVM (ядро полиномиальное)	1,5	9,3	0,3	96,7	94,8	90,7
Дерево решений	3,1	11,3	0,8	94,8	89,5	88,7
Ансамбль деревьев решений	0,9	11,1	0,3	96,7	96,8	88,9
Градиентный бустинг	**0,5**	**14,3**	**0**	**96,3**	**98,0**	**85,7**

Наилучшие результаты в отношении характеристик ложных срабатываний и неверной метки семейства были получены с использованием градиентного бустинга в качестве алгоритма машинного обучения. Метод опорных векторов (SVM) с линейным ядром также оказался эффективен для решения данной задачи и показал наименьшую ошибку второго рода среди всех классификаторов.

В работе [17] представлено сравнение 179 классификаторов, принадлежащих 17 различным семействам, на 121 наборе данных. По результатам сравнения по точности (*accuracy*) наилучшие результаты показывают различные вариации леса деревьев решений и SVM с ядром Гаусса. Градиентный бустинг, наиболее подходящий для решения поставленной в настоящей работе задачи, в работе [17] не рассматривался. Тем не менее его также можно отнести к различным вариациям леса деревьев решений. Из SVM-классификаторов, использующих отличные от линейной функции ядра, наиболее эффективным для решаемой задачи оказался классификатор с полиномиальным ядром.

Заключение

Предложен метод построения моделей Android-приложений и их классификации на основе динамического анализа. Построенные модели очень подробно отражают поведение приложений. Однако эти модели существенно зависят от возможностей инструментальной среды — от эмулятора и генератора событий для исследования анализируемых приложений. Как следствие, для некоторых приложений не удается построить достаточно подробных моделей и провести для них автоматическую классификацию. Результаты проведенных тестовых испытаний показывают, что предложенный в настоящей работе метод позволяет обнаруживать около 85 % вредоносных приложений в тестовой выборке. При этом правильно определяется метка семейства и выдается незначительное количество ложных срабатываний (0,5 %). Процесс построения моделей является ресурсоемким, поэтому наиболее целесообразным является использование метода для отложенного анализа, например, в качестве одной из проверок на стороне рынков мобильных приложений.

Список литературы

1. **Number** of available applications in the Google Play Store from December 2009 to December 2018. URL: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/ (дата обращения 18.03.2019).
2. **honeynet**/droidbot: A lightweight test input generator for Android. URL: https://github.com/honeynet/droidbot (дата обращения: 18.03.2019).
3. **Enck W., Gilbert P., Chun B.-G.** et al. TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones // Proc. of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10) — Vancouver: USENIX Association, 2010. P. 1—6.
4. **Dynamic** analysis of Android apps. URL: https://github.com/pjlantz/droidbox (дата обращения 18.03.2019).
5. **Zheng C., Zhu S., Dai S., Gu G.** et al. SmartDroid: An Automatic System for Revealing UI-based Trigger Conditions in Android Applications // Proc. of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM '12) Raleigh: ACM, 2012. P. 93—104.
6. **Zhou Y., Wang Z., Zhou W., Jiang X.** Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets // Proc. of the 19th Annual Network & Distributed System Security Symposium (NDSS). San Diego: Internet Society, 2012.URL: https://www.ndss-symposium.org/wp-content/uploads/2017/09/07_5.pdf (дата обращения 03.05.2019).
7. **Zhang Y., Yang M., Xu B.** et al. Vetting Undesirable Behaviors in Android Apps with Permission Use Analysis // Proc. of the 2013 ACM SIGSAC Conference on Computer and Communications Security (CCS'13) Berlin: ACM, 2013. P. 611—622.
8. **Rastogi V., Chen Y., Enck W.** AppsPlayground: Automatic Security Analysis of Smartphone Applications // Proc. of the Third ACM Conference on Data and Application Security and Privacy (CODASPY '13) San Antonio: ACM, 2013. P. 209—220.
9. **Shabtai A., Tenenboim-Chekina L., Mimran D.** et al. Mobile malware detection through analysis of deviations in application network behavior // Computers & Security. 2014. Vol. 43. P. 1—18.
10. **Tam K., Khan S. J., Fattori A., Cavallaro L.** CopperDroid: Automatic Reconstruction of Android Malware Behaviors // Proceedings of the Network and Distributed System Security Symposium (NDSS). San Diego: Internet Society, 2015. URL: https://www.ndss-symposium.org/wp-content/uploads/2017/09/02_4_1.pdf (дата обращения 03.05.2019).
11. **Irolla P., Filiol E.** Glassbox: Dynamic Analysis Platform for Malware Android Applications on Real Devices. URL: https://arxiv.org/abs/1609.04718, 2016 (дата обращения 18.03.2019).
12. **romainguy**/ViewServer: Local Server for Android's HierarchyViewer. URL: https://github.com/romainguy/ViewServer (дата обращения 18.03.2019).
13. **Inspect** trace logs with Traceview. URL: https://developer.android.com/studio/profile/traceview (дата обращения 18.03.2019).
14. **Patches** to add operation logging in Android. URL: https://github.com/nastya/droid_patches (дата обращения 18.03.2019).
15. **Сковорода А. А., Гамаюнов Д. Ю.** Анализ мобильных приложений с использованием моделей привилегий и API-вызовов вредоносных приложений // Прикладная дискретная математика. 2017. № 36. С. 84—105.
16. **Python** implementations of the Boruta all-relevant feature selection method // URL: https://github.com/scikit-learn-contrib/boruta_py (дата обращения 18.03.2019).
17. **Fernández-Delgado M., Cernadas E., Barro S., Amorim D.** Do we Need Hundreds of Classifiers to Solve Real World Classification Problems? // Journal of Machine Learning Research. 2014. No. 15. P. 3133—3181.

Dynamic Analysis of Android Applications

A. A. Skovoroda, nastya_jane@seclab.cs.msu.su, **D. Yu. Gamayunov,** gamajun@seclab.cs.msu.su, Lomonosov Moscow State University, Moscow, 119991, Russian Federation

Corresponding author:

Skovoroda Anastasia A., Programmer, Lomonosov Moscow State University, 119991, Moscow, Russian Federation
E-mail: nastya_jane@seclab.cs.msu.su

Received on April 10, 2019
Accepted on April 19, 2019

In this article we consider a problem of automated detection of mobile malware applications and propose a method based on dynamic analysis of the application code. The proposed method is based on building dynamic model of the application represented by graph model of a special kind consisting of vertices representing the application states and edges representing transitions between states marked with "input"-"reaction" pairs. Input can be some user action or system event and reaction is execution of some API calls or actions sequence. Built models are compared with basic malware models which are preliminary obtained from malware collection using hierarchical clustering. The results of model comparison together with some other characteristics including API call related information form feature vectors which are then used in classification with machine learning algorithms. The best classification results were obtained using gradient boosting algorithm — 85 % of the malicious applications from the test set were classified correctly while false alarms rate on real applications from Google Play resulted in 0,5 %. The proposed method suits for the usage as one of the automated checks on application marketplace side, it can also be used in corporate systems of Mobile Device Management class. Built models have a special value and might be used as auxiliary structures for manual analysis of the suspicious applications.

Keywords: *Android applications, dynamic analysis, Android emulator, user interaction automation, API calls, permissions, dynamic behavior model, machine learning*

For citation:

Skovoroda A. A., Gamayunov D. Yu. Dynamic Analysis of Android Applications, *Programmnaya Ingeneria,* 2019, vol. 10, no. 7-8, pp. 324—333.

DOI: 10.17587/prin.10.324-333

References

1. **Number** of available applications in the Google Play Store from December 2009 to December 2018, available at: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/

2. **honeynet/**droidbot: A lightweight test input generator for Android, available at https://github.com/honeynet/droidbot.

3. **Enck W., Gilbert P., Chun B.-G., Cox L. P., Jung J., McDaniel P., Sheth A. N.** TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones, *Proc. of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*, Vancouver, USENIX Association, 2010, pp. 1—6.

4. **Dynamic** analysis of Android apps, available at: https://github.com/pjlantz/droidbox (accessed 18.03.2019).

5. **Zheng C., Zhu S., Dai S., Gu G., Gong X., Han X., Zou W.** SmartDroid: An Automatic System for Revealing UI-based Trigger Conditions in Android Applications, *Proc. of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM '12)*, Raleigh: ACM, 2012, pp. 93—104.

6. **Zhou Y., Wang Z., Zhou W., Jiang X.** Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets, *Proc. of the 19th Annual Network& Distributed System Security Symposium (NDSS)*, San Diego, Internet Society, 2012, available at: https://www.ndss-symposium.org/wp-content/uploads/2017/09/07_5.pdf.

7. **Zhang Y., Yang M., Xu B., Yang Z., Gu G., Ning P., Wang X., Zang B.** Vetting Undesirable Behaviors in Android Apps with Permission Use Analysis, *Proc. of the 2013 ACM SIGSAC Conference on Computer and Communications Security (CCS'13),* Berlin, ACM, 2013, pp. 611—622.

8. **Rastogi V., Chen Y., Enck W.** AppsPlayground: Automatic Security Analysis of Smartphone Applications, *Proc. of the Third ACM Conference on Data and Application Security and Privacy (CODASPY '13),* San Antonio, ACM, 2013, pp. 209—220.

9. **Shabtai A., Tenenboim-Chekina L., Mimran D., Rokach L., Shapira B., Elovici Y.** Mobile malware detection through analysis of deviations in application network behavior, *Computers & Security,* 2014, vol. 43, pp. 1—18.

10. **Tam K., Khan S. J., Fattori A., Cavallaro L.** CopperDroid: Automatic Reconstruction of Android Malware Behaviors, *Proceedings of the Network and Distributed System Security Symposium (NDSS),* San Diego, Internet Society, 2015, available at: https://www.ndss-symposium.org/wp-content/uploads/2017/09/02_4_1.pdf.

11. **Irolla P., Filiol E.** Glassbox: Dynamic Analysis Platform for Malware Android Applications on Real Devices, available at: https://arxiv.org/abs/1609.04718, 2016.

12. **romainguy**/ViewServer: Local Server for Android's HierarchyViewer, available at: https://github.com/romainguy/ViewServer.

13. **Inspect** trace logs with Traceview, available at: https://developer.android.com/studio/profile/traceview.

14. **Patches** to add operation logging in Android, available at: https://github.com/nastya/droid_patches (accessed: 18.03.2019).

15. **Skovoroda A. A., Gamayunov D. Yu.** Automated static analysis and classification of Android malware using permission and API call models, *Prikladnaya diskretnaya matematika*, 2017, no. 36, pp. 84—105 (in Russian).

16. **Python** implementations of the Boruta all-relevant feature selection method, available at: https://github.com/scikit-learn-contrib/boruta_py.

17. **Fernández-Delgado M., Cernadas E., Barro S., Amorim D.** Do we Need Hundreds of Classifiers to Solve Real World Classification Problems? *Journal of Machine Learning Research,* 2014, no. 15, pp. 3133—3181.

УДК 004.422 DOI: 10.17587/prin.10.334-343

В. В. Драчев, программист, e-mail: vowa.drachyov@yandex.ru, ООО "iRidium mobile", Нижний Тагил, **Н. В. Бужинская,** канд. пед. наук, доц., e-mail: nadezhdabuzh@gmail.com, **Е. С. Васева,** канд. пед. наук, доц., e-mail: e-s-vaseva@mail.ru, Нижнетагильский государственный социально-педагогический институт (филиал) ФГАОУ ВО "Российский государственный профессионально-педагогический университет", Нижний Тагил

Разработка программного решения для автоматизации оперативного изменения контента сайтов, созданных с помощью CMS WordPress

Проведен анализ программных механизмов для оперативного обновления контента сайтов, созданных в системе управления контентом WordPress. Область сайта, который необходимо оперативно обновлять, предлагается определить с помощью шорткода. Для удобства работы пользователя было создано мобильное приложение для смартфонов на базе Android, посредством которого пользователь обновляет контент. Разрабатываемый для этого плагин состоит из двух обработчиков. Первый обработчик принимает POST-запросы от мобильного приложения, позволяющие определить способ обновления контента. Второй обработчик принимает шорткод, выбирает информацию из базы и выводит контент на сайт. Возможности реализуемой системы продемонстрированы на диаграмме прецедентов. Представлены коды для обработчиков на языке PHP. Описана технология разработки мобильного приложения на языке Kotlin. Настройки мобильного приложения могут меняться в зависимости от перехватываемых широковещательных сигналов, отправленных другими мобильными приложениями типа менеджеров задач.

Ключевые слова: *система управления контентом, шорткод, плагин, обработчик, диаграмма прецедентов, мобильное приложение, класс*

Введение

Глобальная сеть Интернет является практически незаменимой частью жизни любого человека. Решение почти любой задачи в настоящее время, как правило, сопровождается предварительным поиском и анализом информации в Интернете [1]. Такое стремительное развитие сети Интернет привело к тому, что сегодня большинство компаний имеют свои сайты, владение интернет-ресурсом стало уже необходимостью. Такой сайт представляет возможность организовать информирование пользователей, является эффективным способом рекламы. Он позволяет не только представить визитную карточку компании, но и организовать виртуальный круглосуточный офис. В любом случае, с какой целью бы не был создан сайт, его популярность и посещаемость зависят от актуальности представленной на нем информации.

Новейшие технологии позволяют создавать сайт различными методами и средствами. Популярным способом является система управления контентом — content management system (CMS) [2]. Система управления контентом используется для организации, хранения и управления содержимым и стилем текста, графикой, видеоинформацией на веб-сайте. Веб-сайт при этом, как правило, является динамичным, т. е. вся информация на странице сайта генерируется в зависимости от действий пользователя. Для уменьшения объема кода и развития функциональных возможностей сайта содержимое организовано в виде базы данных. Одной из популярных среди разработчиков и пользователей систем управления контентом является WordPress. Чтобы обновить контент на сайте, созданном в системе WordPress, необходимо зайти в панель управления администратора, дописав /wp-login.php к адресу, ввести логин и пароль администратора. После авторизации осуществляется переход в панель управления. Для изменения контента необходимо выбрать соответствующий пункт в меню администратора (страницы, записи). В открывшемся окне показаны поля редактирования, содержание которых можно интерактивно изменять [3].

Изменения контента могут происходить не только каждый день, но и каждый час. Так, например, частота изменения контента играет большую роль для

новостных региональных сайтов — в случае несвоевременного представления информации такой сайт просто теряет конкурентоспособность. Отражение на сайте интернет-магазина цен и наличия товара, соответствующих действительности, положительно сказывается на формировании имиджа организации. Если клиент уже в процессе оформления заказа выясняет, что информация на сайте устаревшая, то это вызывает недоверие к ресурсу и, как следствие, поиск другого более надежного интернет-магазина. Процесс добавления или изменения информации через административную панель может занять достаточно продолжительное время. По этой причине разработка программных механизмов оперативного изменения контента является востребованной на практике задачей.

Анализ существующих программных средств на этом направлении показал, что в настоящее время нет решений, позволяющих пользователю, не обращаясь к административной панели, изменять контент сайтов на WordPress [3], а также использовать WordPress в связке с менеджерами задач. Разработка мобильного приложения, устраняющего этот недостаток, и соответствующего плагина позволит быстро и своевременно обновлять контент сайта, автоматизировать его публикацию в зависимости от определенных условий, выполняющихся в менеджере задач.

Постановка задачи

В системе управления контентом WordPress содержание и оформление страниц и записей меняются в административной панели. Чтобы добавить внутрь записи или страницы результат некоторой программной обработки, используются шорткоды. Шорткод — это функция, которая может использовать сведения из базы данных сайта. Управление содержанием этой функции должно осуществляться через мобильное приложение. Разрабатываемый плагин при установке будет создавать таблицу, в которой каждая запись соответствует одному шорткоду.

Возможности реализуемой системы можно продемонстрировать на диаграмме прецедентов (рис. 1). Диаграмма прецедентов отражает функциональные требования, которые определяют поведение программной системы [4].

Для разработки предложенной системы необходимо решить перечисленные далее задачи.

1. Создать таблицу в базе данных, которая будет содержать описание шорткодов, определенных на страницах и в записях сайта.

2. Разработать плагин, позволяющий принимать шорткод, выбирать из таблицы базы данных строку и выводить контент на сайт с учетом запроса, полученного от мобильного приложения (создать, обновить или удалить строку).

3. Разработать мобильное приложение, осуществляющее формирование запросов на обновление контента на сайте.

Разработка шорткода

Для создания плагина для WordPress необходимо ввести код для двух обработчиков (рис. 1). Будем работать с языком программирования PHP, который используется большинством хостингов и обладает

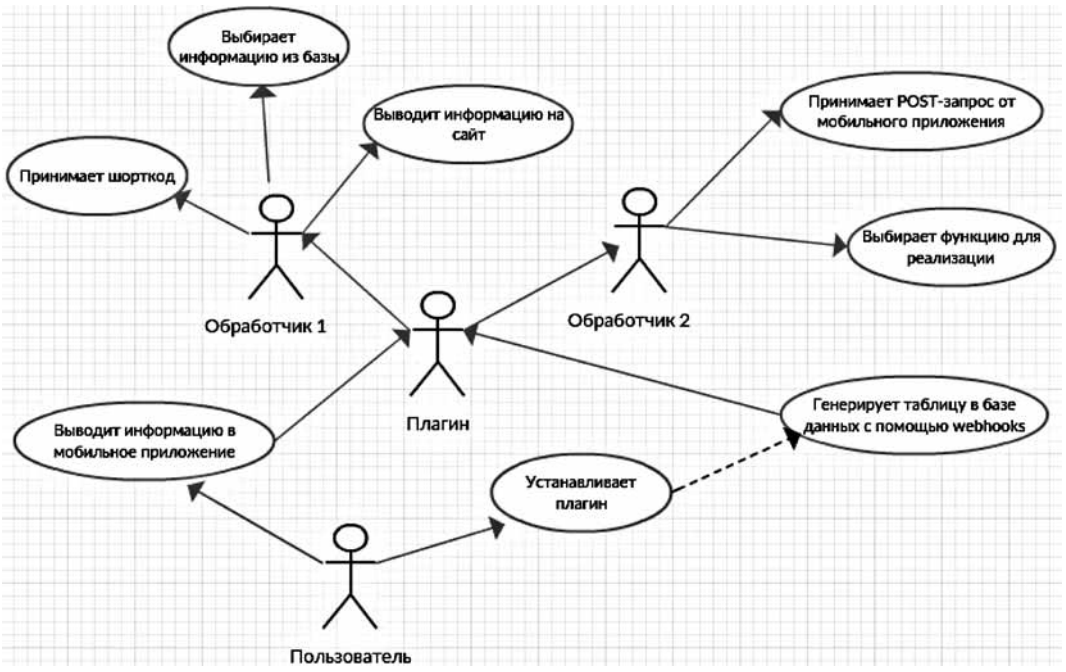


Рис. 1. Диаграмма прецедентов

```php
<?php
/*
Plugin Name: Шорткаты
Description: Описание плагина
Version: 1.0
Author: Автор плагина
Author URI: http://mk-master.ru
*/

global $wpdb; // Переменная для SQL-запросов
//При активации плагина создаём таблицу
function myplugin_activate() {
global $wpdb; // Переменная для SQL-запросов
$sql="CREATE TABLE my_short( ".
      "id INT NOT NULL AUTO_INCREMENT, ".
      "title VARCHAR(40) NOT NULL, ".
      "data VARCHAR(400) NOT NULL, ".
         "flag INT(10) NOT NULL, ".
      "PRIMARY KEY (id)); ";

$wpdb->query($sql);
}
register_activation_hook( __FILE__, 'myplugin_activate' );

//Функция получения шорткода
function ha_but($atts,$content=NULL) {
extract( shortcode_atts( array(
'title' => 'Перейти',
'content' => false
), $atts) );
      global $wpdb; // Переменная для SQL-запросов
      $output_dat=""; // Переменная вывода
      $sql="SELECT data, flag FROM my_short WHERE title='".$title."'";
      $output=$wpdb->get_results($sql);
      foreach ($output as $output) {
            if($output->flag == 1){
                  $output_dat=$output_dat . $output->data;
            }
      }
            return $output_dat;
}
add_shortcode('but','ha_but');
?>
```

Рис. 2. Обработчик, принимающий шорткод

	#	Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию	Дополнительно
☐	1	id	int(11)			Нет	*Нет*	AUTO_INCREMENT
☐	2	title	varchar(40)	utf8_general_ci		Нет	*Нет*	
☐	3	data	varchar(400)	utf8_general_ci		Нет	*Нет*	
☐	4	flag	int(10)			Нет	*Нет*	

Рис. 3. Структура таблицы

необходимыми функциональными возможностями для реализации поставленной задачи [5, 6].

Первый из обработчиков (рис. 2) принимает шорткод через WordPressAPI, выбирает из таблицы базы данных необходимую строку и выводит на сайт.

При выполнении кода происходит создание таблицы в базе данных WordPress, в которой будут храниться атрибуты и контент всех шорткодов (рис. 3).

Создание таблицы в базе данных происходит при установке созданного авторами плагина. В процессе его установки вызывается специальный PlaginAPI/HOOKS, который генерирует таблицу в базе данных.

Далее в таблицу можно добавлять информацию посредством второго обработчика. Второй обработчик выполняет POST-запросы от мобильного приложения (рис. 4) и в зависимости от поступивших данных определяет функцию, которую нужно реализовать, а именно — создание, обновление строки или ее удаление.

Например, если приложение отправляет POST-запрос и указывает в нем ID, следовательно, это запрос на обновление существующей строки в таблице, описывающей шорткоды. Если ID отсутствует — запрос на добавление шорткода.

Разработка мобильного приложения

Было разработано мобильное приложение в среде AndroidStudio 3.2. В качестве основного языка разработки был выбран перспективный, бесплатный для использования язык Kotlin, так как Java для Google распространяется на платной основе [7—9, 11].

Для приложения было создано три Activity-компонента (экрана) мобильного приложения, обладающих определенными функциональными возможностями. Для каждого Activity создавался отдельный класс. Первый класс отвечает за вход в приложение (рис. 5).

```
<?php
$host = 'localhost'; // адрес сервера
$database = $_POST['baza'];// имя базы данных
$user = $_POST['login'];// имя пользователя
$password = $_POST['pass'];// пароль
// Получаем строку из базы по ID из GET-запроса
if(isset($_POST['id'])){$get_id=$_POST['id'];}
$link = mysqli_connect($host, $user, $password, $database) or die("Ошибка " .
mysqli_error($link));
if($_POST['id']=="" and $_POST['flag']=="" and $_POST['data']=="" and
$_POST['title']==""){echo "ок";die();exit();}
$link->set_charset("utf8");
// Запрос на получение данных
if($get_id == "-1"){
        $query ="SELECT * FROM my_short";
        $result = $link->query($query) or die("Ошибка " . mysqli_error($link));
        while($row = mysqli_fetch_array($result)){
                echo $row['id']." ++ ".$row['title']." ++ ".$row['data']." ++
".$row['flag']."+<br>";
        }}
// Запрос на изменение данных
else{
        $query ="SELECT * FROM my_short WHERE id='".$get_id."'";
        $result = $link->query($query) or die("Ошибка " . mysqli_error($link));
        $row = mysqli_fetch_array($result);
        // Готовим переменные для SQL-запроса
        if($_POST['title']!=""){$get_title=$_POST['title'];}else{$get_title=$row['title'];}
        //if($_POST['data']!=""){$get_data=$_POST['data'];}else{$get_data=$row['data'];}
        $get_data=$_POST['data'];
        if($_POST['flag']!=""){$get_flag=$_POST['flag'];}else{$get_flag=$row['flag'];}
        // Узнаём, добавить новую строку или изменить существующую, и делаем
SQL-запрос
        if($_POST['id']!="" and $_POST['flag']!="delete"){$query ="UPDATE my_short
SET title='".$get_title."', data='".$get_data."', flag='".$get_flag."' WHERE id='".$get_id."'";}
        if($_POST['id']!="" and $_POST['flag']=="delete"){$query ="DELETE FROM
my_short WHERE id='".$get_id."'";}
        if($_POST['id']=="" and $_POST['flag']!="delete"){$query ="INSERT INTO
my_short (title, data, flag) VALUES ('".$get_title."', '".$get_data."', '".$get_flag."')";}
        $result = $link->query($query) or die("Ошибка " . mysqli_error($link));
        //if(mysql_num_rows($result)==0){echo "OK";}
}
// Чистим переменные для освобождения оперативной памяти и закрываем связь
с базой
mysqli_close($link);
?>
```

Рис. 4. Обработчик, принимающий запросы от мобильного приложения

```
var pd: ProgressDialog? = null
class MainActivity : AppCompatActivity() {
    private var mSettings: SharedPreferences? = null
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        mSettings = getSharedPreferences("nastroiki", Context.MODE_PRIVATE)
        sait.setText(mSettings!!.getString("sait",""))
        pass.setText(mSettings!!.getString("pass",""))
        login.setText(mSettings!!.getString("login",""))
        bazadan.setText(mSettings!!.getString("baza",""))
        Toast.LENGTH_SHORT).show()
        if(mSettings!!.getInt("vhod",0)==1){
            start_home(button)
        }
    }
    fun start_home(view: View){
        val editor = mSettings!!.edit()
        editor.putString("sait", sait.text.toString())
        editor.putString("login", login.text.toString())
        editor.putString("pass", pass.text.toString())
        editor.putString("baza", bazadan.text.toString())
        editor.apply()
        val zapros = PostZapros()
        val progres = MyCustomTask(this)
        zapros.sait = sait.text.toString()
        zapros.login = login.text.toString()
        zapros.pass = pass.text.toString()
        zapros.baza = bazadan.text.toString()
        //Thread.sleep(1000)
        //progres.onPreExecute()
        zapros.execute()
        //progres.onPostExecute()
        val rez = zapros.get()
        if (rez.length == 2) {
            startes()
        }
            else{
            if(rez=="Проблемы с интернетом"){Toast.makeText(applicationContext,"Нет
соединения с интернет!", Toast.LENGTH_SHORT).show()}
            else{Toast.makeText(applicationContext,"Данные не верны!",
Toast.LENGTH_SHORT).show()}
            return
        }
        }
    private fun startes(){
        val editor = mSettings!!.edit()
        editor.putInt("vhod", 1)
        editor.apply()
        val home = Intent(this, Home::class.java)
        startActivity(home)
    }
}
```

Рис. 5. Класс, отвечающий за вход в приложение

Второй класс отвечает за вывод информации на экран (рис. 6).

Третий класс отвечает за создание, редактирование и удаление шорткода. Дополнительно был создан класс для выполнения потоковых задач, таких как удаление, добавление, получение, изменение шорткодов. Класс написан на языке Kotlin (рис. 7).

Пятый класс служит для приема Broadcast — широковещательного сообщения. Он обеспечивает взаимодействие разрабатываемого приложения со сторонними приложениями, например, менеджерами задач (рис. 8). Менеджеры задач позволяют автоматизировать некоторые действия на смартфоне, т. е. выполнять их без участия пользователя. В менеджере задач есть две группы обязательных параметров — события и действия, а также необязательные параметры — условия. В нашем случае происходит действие — отправка широковещательного сообщения при наступлении события. Событие определяется в менеджере задач, например, начало события в календаре, уровень заряда батареи, тип активной сети и т. д., список возможных событий достаточно обширен [10]. Также могут быть заданы определенные условия выполнения действия, например, местоположение пользователя. Разработанное авторами мобильное приложение принимает Broadcast-сообщение, отправленное менеджером задач.

Результаты тестирования программных механизмов

Для изменения контента на сайте необходимо запустить мобильное приложение на смартфоне, осуществить вход. При первом запуске нужно ввести адрес сайта, имя базы данных, логин и пароль.

```
class Home : AppCompatActivity() {
    val catNames = ArrayList<String>()
    val catNames2 = ArrayList<String>()
    val catNames3 = ArrayList<String>()
    val catNames4 = ArrayList<String>()
    private var mSettings: SharedPreferences? = null
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_home)
        val progres = MyCustomTask(this)
        //progres.onPreExecute()
        //progres.onPostExecute()
        sendGet()
    }
    override fun onCreateOptionsMenu(menu: Menu): Boolean {
        menuInflater.inflate(R.menu.menu_main, menu)
        return true
    }
    fun btnRemoveClick(v: View) {
        val position = list_spisik.getPositionForView(v.parent as View)
        val Redaktor = Intent(this, Redaktor::class.java)
        Redaktor.putExtra("id", catNames[position])
        Redaktor.putExtra("title", catNames2[position])
        Redaktor.putExtra("data", catNames3[position])
        Redaktor.putExtra("flag", catNames4[position])
        startActivity(Redaktor)
    }
        fun sendGet(){
            mSettings = getSharedPreferences("nastroiki", Context.MODE_PRIVATE)
            val zapros = PostZapros()
            zapros.id = "-1"
            zapros.sait = mSettings!!.getString("sait","")
            zapros.login = mSettings!!.getString("login","")
            zapros.pass = mSettings!!.getString("pass","")
            zapros.baza = mSettings!!.getString("baza","")
            zapros.execute()
            val someText = zapros.get()
            if(someText==""){return}
                val adapter = ArrayAdapter<String>(this, R.layout.list_item, R.id.TextView,
catNames2)
                list_spisik.adapter = adapter
                adapter.clear()
                val str= someText!!.dropLast(5)
                val strs = str.split("+<br>")
                for (strs2 in strs){
                    val strs3 = strs2.split(" ++ ")
                    catNames.add(strs3[0])
                    catNames2.add(strs3[1])
                    catNames3.add(strs3[2])
                    catNames4.add(strs3[3])
                }
                adapter.notifyDataSetChanged();
            }
    fun btnAdd(v: View) {
        val Redaktor = Intent(this, Redaktor::class.java)
        Redaktor.putExtra("id", "")
        Redaktor.putExtra("title", "")
        Redaktor.putExtra("data", "")
        Redaktor.putExtra("flag", "1")
        startActivity(Redaktor)
    }
    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        when (item.itemId) {
            R.id.action_settings -> {
                val editor = mSettings!!.edit()
                editor.putInt("vhod", 0)
                editor.apply()
                val home = Intent(this, MainActivity::class.java)
                startActivity(home)
                return true
            }
            else -> return super.onOptionsItemSelected(item)
        }
    }
}
```

Рис. 6. Класс, отвечающий за вывод информации на экран

```
        public class PostZapros extends AsyncTask<Context,String,String>{
            String id = "";
            String title = "";
            String data = "";
            String flag = "";
            String baza = "";
            String sait = "";
            String login = "";
            String pass = "";
            protected void onPostExecute(String result) {
                super.onPostExecute(result);
            }
            @Override
            protected String doInBackground(Context... params) {
                try {
                    final HttpClient httpclient = new DefaultHttpClient();
                    final HttpPost http = new HttpPost(sait+"/wp-
content/plugins/my_short/write_baz.php/");
                    List nameValuePairs = new ArrayList(8);
                    nameValuePairs.add(new BasicNameValuePair("id", id));
                    nameValuePairs.add(new BasicNameValuePair("title", title));
                    nameValuePairs.add(new BasicNameValuePair("flag", flag));
                    nameValuePairs.add(new BasicNameValuePair("data", data));
                    nameValuePairs.add(new BasicNameValuePair("sait", sait));
                    nameValuePairs.add(new BasicNameValuePair("baza", baza));
                    nameValuePairs.add(new BasicNameValuePair("login", login));
                    nameValuePairs.add(new BasicNameValuePair("pass", pass));
                    http.setEntity(new UrlEncodedFormEntity(nameValuePairs, "UTF-8"));
                    String response = (String) httpclient.execute(http, new
BasicResponseHandler());
                    if(response != null){
                        return response;
                    }
                    else{
                        return "No string.";
                    }
                }
                catch(Exception e){
                    return "Проблемы с интернетом";
                }
            }
}
```

Рис. 7. Класс для выполнения потоковых задач

```
    @Suppress("UNREACHABLE_CODE")
    class MyReceiver : BroadcastReceiver() {
        private var mSettings: SharedPreferences? = null
        var intent_id: String? =""
        var intent_title: String? = ""
        var intent_data: String? = ""
        var intent_flag: String? = ""
        override fun onReceive(context: Context, intent: Intent) {
            intent_id = intent!!.getStringExtra("id")
            intent_title = intent!!.getStringExtra("title")
            if(intent_title==null){intent_title=""}
            intent_data = intent!!.getStringExtra("data")
            if(intent_data==null){intent_data=""}
            if(intent_data=="null"){intent_data=""}
            intent_flag = intent!!.getStringExtra("flag")
            if(intent_flag==null){intent_flag=""}
           mSettings = context.applicationContext.getSharedPreferences("nastroiki",
Context.MODE_PRIVATE)
            val zapros = PostZapros()
            zapros.id = intent_id
            zapros.title = intent_title
            zapros.data = intent_data
            zapros.flag = intent_flag
            zapros.sait = mSettings!!.getString("sait","")
            zapros.login = mSettings!!.getString("login","")
            zapros.pass = mSettings!!.getString("pass","")
            zapros.baza = mSettings!!.getString("baza","")
            zapros.execute()
            }
        }
```

Рис. 8. Класс для приема широковещательных сообщений

Рис. 9. Проверка данных

Данную информацию можно получить из настроек WordPress. Приложение проверяет правильность ввода данных, а также наличие доступа в сеть Интернет. При последующем запуске приложения эти данные сохраняются (рис. 9). Далее необходимо ввести имя шорткода, которое затем будет использоваться на сайте. Ввести информацию, которую будет хранить данный шорткод.

При написании шорткода на сайте указывается его заголовок. Пример добавления шорткода в административной панели и отображение его содержимого на сайте представлено на рис. 10. При вво-

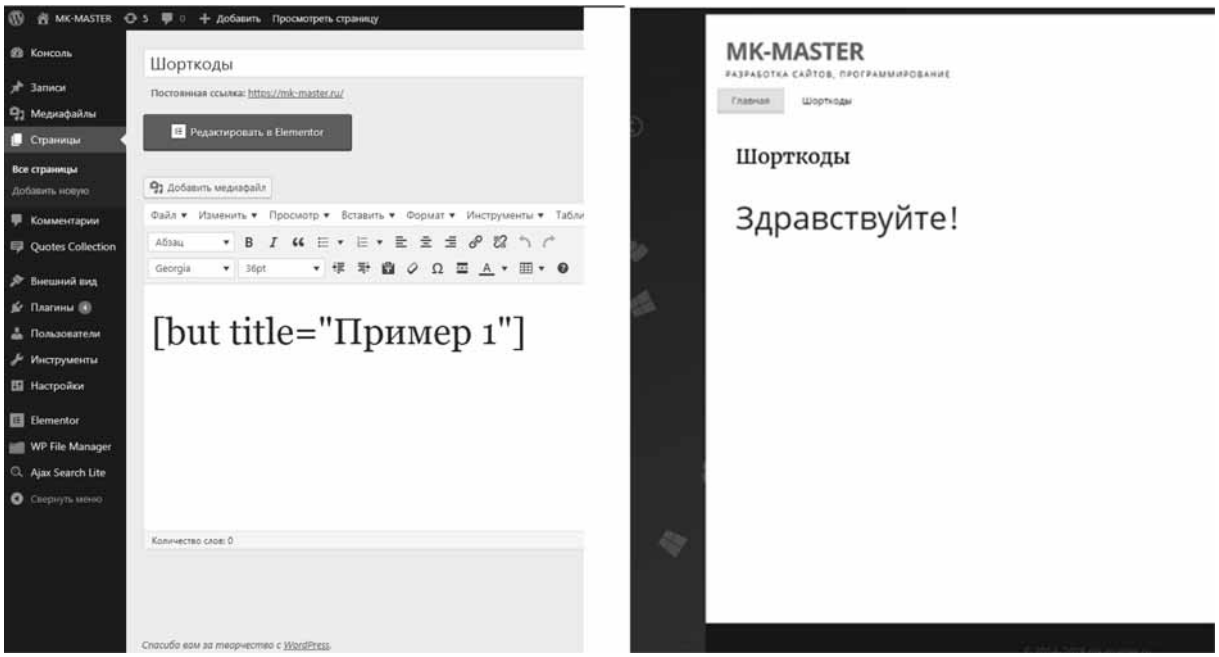


Рис. 10. Добавление шорткода на сайт

Действие	Ожидаемый результат	Результат теста
Открытие мобильного приложения	Отображается страница авторизации. Название окна "Вход в SHORTCAT". На форме четыре поля: адрес сайта, имя базы данных, логин и пароль, кнопка "Войти" доступна	Соответствует ожидаемому результату
Ввод корректных данных: адрес сайта/имя базы данных/логин/пароль при входе в мобильное приложение	Осуществляется вход в приложение. Отображается экран с выбором шорткода	Соответствует ожидаемому результату
Ввод некорректной пары логин/пароль при входе в мобильное приложение	Отображается страница авторизации. Название окна "Вход в SHORTCAT". На форме четыре пустых поля: адрес сайта, имя базы данных, логин и пароль. Кнопка "Войти" доступна	Соответствует ожидаемому результату

де информации для определенного шорткода через мобильное приложение происходит обновление информации.

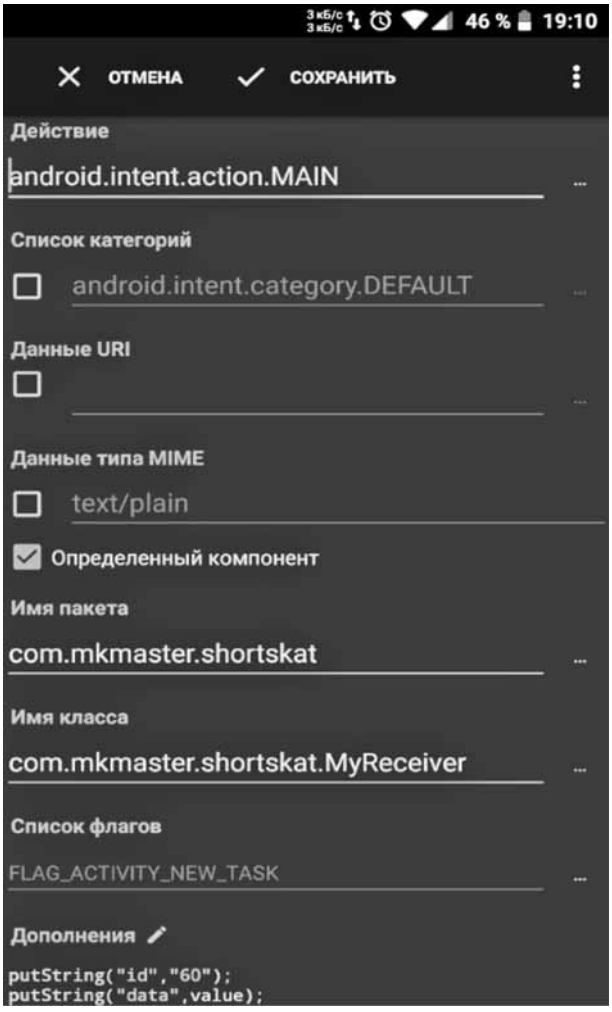


Рис. 11. Пример связи мобильного приложений с Automagic

Для взаимодействия с шорткодом через стороннее приложение необходимо отправить широковещательное сообщение, которое содержит имя пакета и имя класса. Класс будет обрабатывать Intent-объект, описывающий действие, которое необходимо выполнить. Объект Intent типа String содержит ID шорткода и данные. Также можно включать и выключать шорткод, менять его заголовок и удалять. Пример настройки связи мобильного приложения с внешним приложением Automagic представлен на рис. 11.

Разработанные программные механизмы были протестированы. Тестирование показало, что все возможные условия и действия пользователя (очевидные и неочевидные) имеют ожидаемые результаты. Фрагмент листа тестовых случаев представлен в таблице.

Заключение

Область сайта, в которой необходимо оперативно обновлять контент, определяется с помощью шорткода. Шорткод может содержать описание стиля css, JavaScript, html-разметку, следовательно, можно менять не только контент на сайте, но и изменять внешний вид и функциональность сайта.

Существенным достоинством представленных в настоящей работе программных механизмов является то, что настройки мобильного приложения могут меняться в зависимости от перехватываемого Broadcast-сообщения, отправленного другим мобильным приложением типа менеджеров задач. Содержание Broadcast формируется в зависимости от срабатывания триггера, который может зависеть от самых разнообразных условий. В результате изменяется информация в базе данных сайта и, как следствие, изменяется контент.

Разрабатываемые программные механизмы были протестированы при получении POST-запросов, отправляемых с мобильного приложения, установленного на различных версиях Android. Результаты тестовых испытаний соответствуют ожидаемым.

Список литературы

1. **Жеребин В. М., Ермакова Н. А.** Информатизация повседневной жизни населения // Вопросы статистики. 2010. № 10. С. 10—20.
2. **Макдональд М.** Создание web-сайта. Недостающее руководство. СПб.: БХВ-Петербург, 2013. 624 с.
3. **Официальный** сайт WordPress. URL: https://ru.wordpress.org/.
4. **Шелехов В. И.** Классификация программ, ориентированная на технологию программирования // Программная инженерия. 2016. № 12. С. 531—538.
5. **Веллинг Л., Томсон Л.** Разработка веб-приложений с помощью PHP и MySQL. М.: Вильямс, 2016. 848 с.
6. **Колисниченко Д.** PHP и MySQL. Разработка Web-приложений. СПб.: БХВ-Петербург, 2013. 560 с.
7. **Голощапов А.** Google Android. Программирование для мобильных устройств. СПб.: БХВ-Петербург. 2012. 443 с.
8. **Официальный** сайт Kotlin. URL: https://kotlinlang.org.
9. **Официальный** блог Kotlin. URL: https://blog.jetbrains.com/kotlin/.
10. **Automagic.** URL: http://automagic4android.com/home_en.html.
11. **Get** Started with Kotlin on Android. URL: https://developer.android.com/kotlin/get—started.

Development of a Software Solution to Automate the Operational Change of the Content of Sites Created Using CMS WordPress

V. V. Drachev, vowa.drachyov@yandex.ru, LLC "iRidium mobile", Nizhny Tagil, 622000, Russian Federation, **N. V. Buzhinskaya,** nadezhdabuzh@gmail.com, **E. S. Vaseva,** e-s-vaseva@mail.ru, Nizhny Tagil state socio-pedagogical Institute (branch) of Federal State Autonomous educational institution "Russian state vocational pedagogical University", Nizhny Tagil, 622031, Russian Federation.

Corresponding author:

Vaseva Elena S., Associate Professor, Nizhny Tagil state socio-pedagogical Institute (branch) of Federal state autonomous educational institution "Russian state vocational pedagogical university", Nizhny Tagil, 622031, Russian Federation
E-mail: e-s-vaseva@mail.ru

Received on April 13, 2019
Accepted on June 17, 2019

Nowadays, a majority of people as well as companies has a website. One of easy ways to create a site is to use CMS. Developers and users stick to a CMS called WordPress.

To update site content developed in WordPress it is necessary to use a control panel after being logged in. Then it is necessary to choose a site area and after that to update the content. The procedure takes significant time and that is why the development of duly updates software is a top priority objective.

The article describes software for duly updates at websites developed in WordPress. The updated area is determined by a shortcode. Attributes and content of shortcodes are in a database.

A technology for developing a plugin to accept the shortcode, choose a string in the database and show the content after a request from a mobile app is described. Plugin consists of two Events handlers, codes for Events handlers in PHP are described. The technology of a mobile app using Kotlin language is described.

This software ensures the interaction of the website developed in WordPress and the mobile app. The scope of the system is illustrated in the diagram of precedents.

These mechanisms were tested. Tests showed that every possible circumstance and user's action have the expected result.

A conclusion can be carried out that the software coordinates with the demands: to change exterior of the website and text on any page. Changes depend on triggers and can be arranged by a task manager.

Keywords: *content management system, shortcode, plug-in, event handler, use case diagram, mobile application, class*

For citation:

Drachev V. V., Buzhinskaya N. V., Vaseva E. S. Development of a Software Solution to Automate the Operational Change of the Content of Sites Created Using CMS WordPress, *Programmnaya Ingeneria,* 2019, vol. 10, no. 7—8, pp. 334—343.

DOI: 10.17587/prin10.334-343

References

1. **Zherebin V. M., Ermakova H. A.** Informatization of the daily life of the population, *Voprosy statistiki,* 2010, no. 10, pp. 10—20 (in Russian).
2. **Makdonal'd M.** *Creating a web site*. Missing guide, St. Petersburg, BHV-Peterburg, 2013, 624 p. (in Russian).
3. **Official** WordPress site, available at: https://ru.wordpress.org/.
4. **Shelekhov V. I.** Classification of programs focused on programming technology), *Programmnaya ingeneria*, 2016, vol. 7, no 12, pp. 531—538 (in Russian).
5. **Velling L., Tomson L.** *Web application development using PHP and MySQL*, Moscow, Vil'yams, 2016, 848 p. (in Russian).
6. **Kolisnichenko D.** *PHP and MySQL. Web application development*, St. Petersburg, BHV-Peterburg, 2013, 560 p. (in Russian).
7. **Goloshchapov A.** *Programming for mobile devices,* St. Petersburg, BHV-Peterburg, 2012, 443 p. (in Russian).
8. **Official** Kotlin, available at: https://kotlinlang.org.
9. **Official** Kotlin Blog, available at: https://blog.jetbrains.com/kotlin//
10. **Automagic,** available at: http://automagic4android.com/home_en.html
11. **Get Started** with Kotlin on Android, available at: https://developer.android.com/kotlin/get—started.

УДК 004.056 DOI: 10.17587/prin.10.344-352

А. В. Комарова, аспирант, e-mail: piter-ton@mail.ru, Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики,
А. Г. Коробейников, д-р техн. наук, проф., e-mail: korobeynikov_a_g@mail.ru,
Институт Земного магнетизма, ионосферы и распространения радиоволн имени Н. В. Пушкова Российской академии наук, Санкт-Петербург

Обзор истории и тенденций развития постквантовой криптографии на основе теории решеток

Рассмотрены основные положения, определения и задачи, связанные с разработкой криптографических примитивов, основанных на теории решеток. В настоящее время эта тенденция является одним из перспективных направлений развития постквантовой криптографии. Приведена справочная информация об истории зарождения, а также о последних тенденциях развития данного раздела криптографии. Обозначены наиболее перспективные, по мнению авторов, практические постквантовые схемы электронной подписи.

Ключевые слова: *криптография, постквантовая криптография, теория решеток, задача поиска кратчайшего вектора решетки, задача поиска ближайшего вектора решетки, задача обучения с ошибками*

Введение

Последнее время в области информационной безопасности киберфизических систем особое внимание уделяется вопросам создания квантового компьютера. После получившего известность квантового алгоритма П. Шора (P. Shor) [1] мировое криптографическое сообщество стало бурно развивать постквантовую отрасль криптографии. Начиная с 2006 г. проводится ежегодная конференция PQCRYPTO, посвященная вопросам постквантовой криптографии. Это направление включает в себя ряд подходов, однако, с точки зрения авторов, наиболее перспективным из них является криптография, основанная на теории решеток. К сожалению, в русскоязычной литературе мало публикаций, посвященных постквантовой криптографии и теории решеток в частности. Это обстоятельство побудило авторов к написанию настоящей работы обзорного характера, призванной восполнить этот недостаток.

К основным публикациям по данной тематике можно отнести работу [2] авторов А. В. Шокурова, Н. Н. Кузюрина и С. А. Фомина, в которой достаточно подробно освещены основные понятия теории сложности и теории групп. В ней в кратком изложении рассмотрены основные сложно вычислимые задачи и криптосистемы на решетках. Большую работу на этом направлении проводили также О. В. Кузьмин и В. С. Усатюк [3—6]. В работах [3—6] представлены обзоры основных сложных в вычислительном плане задач на решетках и некоторые криптосистемы. Однако большинство упомянутых выше работ датируются 2010—2011 гг. За прошедший промежуток времени в такой стремительно развивающейся области науки, как постквантовая криптография, произошло немало изменений, появились новые тенденции и подходы. Целью настоящей работы является ознакомление читателей с базовыми определениями, разновидностями решеток и основными трудно решаемыми задачами на их основе.

Авторы уже изучали рассматриваемый вопрос, и результаты были опубликованы в работах [7—9]. Однако настоящая работа отличается от них более подробным изложением материала и описанием новейших тенденций. Следует отметить, что исследования на этом направлении являются актуальными, поскольку наиболее распространенные на настоящее время криптографические протоколы, схемы и алгоритмы, основанные на задачах теории чисел (задача факторизации, задача дискретного логарифмирования), объективно перестанут быть безопасными в случае реализации квантового вычислителя.

Историческая справка

Решетки начали исследовать еще с конца XVIII века такие ученые, как Ж. Л. Лагранж (J. L. Lagrange), И. Гаусс (J. Gauss), П. Дирихле (P. Dirichlet) и Г. Миньковский (H. Minkowski).

Ключевое место в схемах на основе решеток занимает алгоритм приведения базиса к ортогональному виду. Наиболее эффективным считается LLL-алгоритм (алгоритм Ленстры — Ленстры — Ловаша). Впервые алгоритм построения LLL-приведенного базиса решетки был предложен учеными А. Ленстрой (A. Lenstra), Х. Ленстрой (H. Lenstra) и Л. Ловашем (L. Lovász) в 1982 г. в работе [10]. В ней были предло-

жены приближенные решения задач теории решеток. Однако аппроксимирующий фактор в данном алгоритме растет экспоненциально или, по крайней мере, суперполиномиально по отношению к размерности решетки. Первоначально данный алгоритм использовали только для целей криптоанализа, для атак на существующие криптосистемы, в том числе на некоторые модификации RSA (RSA — криптографический асимметричный алгоритм, основывающийся на вычислительной сложности задачи разложения больших целых чисел на простые множители).

Для создания криптографических примитивов теория решеток стала использоваться после публикации работы [11] М. Айтая (M. Ajtai) в 1996 гг. Это время можно считать началом создания современной криптографии на решетках. Важнейшим результатом работы М. Айтая стало доказательство того, что безопасность теории решеток базируется на доказательстве сложности в наихудшем случае (*worst-case hardness*). Практически все другие криптографические примитивы не обладают таким свойством и лишь базируются на доказательстве сложности в среднем (*average-case hardness*).

Поясним связь между понятиями "сложность в наихудшем случае" и в "сложность в среднем". Временная сложность решения некоторой задачи определяется как характеристика по крайней мере одного случая ее решения. Этот факт не означает, что все случаи будут столь же сложны. Например, для задачи факторизации необходимо выбрать два простых числа-множителя из некоторого распределения. При этом сложность решения задачи разложения на множители будет существенно отличаться в зависимости от конкретного выбора чисел (числа следует выбирать сильно простыми) и от того, какова будет разница между этими числами. Под сильно простым числом p понимается достаточно большое простое число, такое, что числа $p + 1$ и $p - 1$ имеют достаточно большие простые делители, которые также имеют достаточно большие простые делители. В криптографических конструкциях, основанных на сложности в наихудшем случае, вопросы выбора строго определенных параметров для достижения требуемого уровня безопасности не возникают [12].

В частности, в работе [11] М. Айтая было показано, что если существует алгоритм, решающий известную задачу с какой-то вероятностью, то можно построить полиномиальный алгоритм, решающий эту же задачу в наихудшем случае. Применительно к решеткам любую сложность в среднем можно свести к сложности в наихудшем случае.

Первая схема шифрования, основанная на решетках, была предложена в работе М. Айтая и С. Дворк (C. Dwork) [13]. Однако большие размеры зашифрованного текста и большие размеры ключей делали эту схему неприемлемой для использования в реальных условиях. Позже схема была упрощена и улучшена О. Регевом (O. Regev) в 2003—2005 гг. [12, 14]. Одним из наиболее важных результатов, представленных в этих работах, стало предложение решения новой сложновычислимой задачи, так называемой задачи обучения с ошибками (*learning with errors problem*, LWE).

Также в 1996 г. учеными О. Голдрайхом (O. Goldreich), С. Голдвассером (S. Goldwasser) и С. Халеви (S. Halevi) была разработана криптосистема GGH [15]. В ее основе лежала задача декодирования ограниченного расстояния (*bounded distance decoding*, BDD). В силу ограниченного набора параметров криптосистема была взломана П. Нгуеном в 1999 г. [16].

В том же 1996 г. на конференции CRYPTO'96 была впервые предложена криптосистема NTRU (*Nth-degree truncated polynomial ring*) — криптосистема с открытым ключом на решетках [17] авторов Д. Хоффстайна (J. Hoffstein), Д. Пайфера (J. Pipher), Ж. Силвермана (J. Silverman). В основе NTRU лежит задача поиска кратчайшего вектора решетки (*closest vector problem*, CVP). В данной криптосистеме используются операции над полиномиальным кольцом $R = \frac{Z_q[X]}{(X^N - 1)}$ усеченных многочленов с целыми коэффициентами степени, не превосходящей $N - 1$:

$$a(X) = a_0 + a_1X + a_2X^2 + ...$$
$$+a_{N-2}X^{N-2} + a_{N-1}X^{N-1} = \sum_{i=0}^{N-1} a_iX^i,$$

где $Z_q[X]$ — конечное кольцо вычетов по модулю q; $(X^N - 1)$ неприводимый в данном кольце многочлен (т. е. не разложимый на нетривиальные многочлены).

В векторном виде этот многочлен можно представить следующим образом:

$$\mathbf{a}(X) = \sum_{i=0}^{N-1} a_iX^i = [a_0, a_1, a_2, ..., a_{N-2}, a_{N-1}].$$

В таком кольце произведение задается как "произведение свертки", т. е. $a(X)a(X)^{-1} \equiv 1 \bmod q$ заменяется на 1, X^{N+1} — на X и т. д. [18].

Обратный элемент к элементу $a(X)$ по модулю q задается следующим образом: $a(X)a(X)^{-1} \equiv 1 \bmod q$. Если у полинома существует обратный элемент, то его можно достаточно легко вычислить с помощью расширенного алгоритма Евклида и леммы Гензеля.

На базе криптосистемы NTRU можно реализовать алгоритмы и протоколы шифрования и электронной подписи. В 2008 г. она была включена в стандарт IEEE 1363.1 "Lattice-based public-key cryptography", а модифицированная версия данного алгоритма была взята за основу стандарта ANSI X9.98-2010 "Lattice-Based Polynomial Public Key Establishment Algorithm for the Financial Services Industry".

Протокол электронной подписи NTRUSign [19], разработанный в 2003 г., стал одним из первых протоколов электронной подписи на решетках. Использование решеток, предложенных в криптосистеме NTRU, обеспечивало компактность и удобство вычислений. Однако детерминированный процесс формирования подписи привел к появлению деструктивных воздействий атак [20] и последующему полному взлому этой системы.

Недетерминированная реализация процесса подписания как успешное разрешение проблемы формирования подписи была предложена в 2008 г. в работе [21] К. Джентри (C. Gentry), К. Пикерта (C. Peikert) и В. Вайкунтанатана (V. Vaikuntanathan). В предлагаемом в работе [21] подходе не только была исправлена дефектная процедура подписания, но и был предложен доказательно безопасный способ создания электронной подписи. Немного позже эта структура была реализована на решетках из системы NTRU [22].

В 2014 г. также на базе решеток NTRU была предложена схема шифрования на основе идентификатора (*identity-based encryption*, IBE) [23], которую можно конвертировать в схему электронной подписи. В такого рода схемах в качестве ключа используется некоторая идентификационная информация получателя. Однако время подписания в этом случае оценивалось в $O(n^2)$, где n — размерность используемой квадратичной матрицы. Для решения рассматриваемой задачи Л. Дюкасом (L. Ducas) и Т. Престом (T. Prest) было предложено использовать быстрое преобразование Фурье для ортогонализации матриц [24]. Такой подход имел в основном теоретическое значение, однако он позволил свести время работы алгоритма к оценке в $O(n\log_2 n)$.

Схема FALCON [25] стала практической реализацией упомянутых выше подходов, объединившей в себе решетки NTRU [19], структуру Джентри — Пикерта — Вайкунтанатана [21] и быстрое преобразование Фурье [24].

В 2016 г. Национальный институт стандартов и технологий США (NIST) объявил о старте конкурса на создание новых, постквантовых стандартов шифрования, обмена ключами и электронной подписи [26]. Среди схем подписи во второй тур конкурса прошли девять схем, в том числе три схемы на решетках, включая qTESLA [27], CRYSTALS-DILITHIUM [28] и FALCON [25].

Преимущества, недостатки и сферы применения криптографии на решетках

Криптографические протоколы, основанные на методах теории решеток, имеют ряд преимуществ перед традиционными протоколами на основе асимметричных схем. К их числу относятся:

- проверенная криптостойкость, доказанная с использованием математического аппарата;
- предположительно высокая стойкость по отношению к вычислениям деструктивного характера с помощью квантового компьютера;
- более высокое быстродействие при сопоставимом уровне стойкости;
- возможности решения большого числа NP-сложных задач в криптографии.

К недостаткам криптографических алгоритмов на решетках можно отнести перечисленные далее.

- Сложность их внедрения, так как для поддержки новых криптографических алгоритмов придется модифицировать большую часть существующего программного обеспечения, которое не предназначено для использования теории решеток. Процессы реализации соответствующего обеспечения и его настройки займут немалое время.
- Специальные параметры, которые потребуются для построения стойких к различного рода атакам схем. Этот факт может быть использован представителями административных структур и другими заинтересованными организациями для достижения собственных целей, а именно для внедрения заведомо "ослабленных" параметров в целях дальнейшей эксплуатации этих уязвимостей.
- Увеличенные размер подписи и время ее формирования. Однако следует отметить, что этот недостаток не представляется столь существенным для приложений, в которых первостепенным требованием является обеспечение безопасности в условиях появления в перспективе квантового компьютера.

В настоящее время теория решеток нашла активное применение в различных областях знаний. К их числу относятся: задачи классической математики (упаковывание сферы, теория чисел, диофантово приближение и др.), инженерные (теория кодирования, беспроводная связь) и компьютерные науки (целочисленное программирование, теория вычислительной сложности, обфускация программного кода). Как уже отмечалось ранее, одной из важнейших и востребованных практикой областей знаний является криптография (односторонние функции, функции с секретом, стойкие к коллизиям хэш-функции, псевдослучайный генератор чисел, схемы асимметричного шифрования, схемы электронной подписи, шифрование на основе атрибутов, криптосистемы с открытым ключом на основе идентификационных данных, предикативное (функциональное) шифрование, полностью гомоморфное шифрование).

Основные положения теории решеток

Настоящая статья носит обзорный характер. Поэтому для более детального понимания представленных далее положений теории решеток, которые используются для решения сложно вычислимых криптографических задач, введем ряд общеизвестных определений.

Определение 1. *Решетка* — это совокупность точек в n-мерном евклидовом пространстве с периодической структурой. Более точно решетку L можно определить как абелеву подгруппу, заданную в пространстве R^n.

Пусть базис решетки $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_n\}$ задан линейно независимыми векторами. Тогда множество целочисленных линейных комбинаций этих векторов будем называть решеткой L:

$$L = \left\{a_1\mathbf{b}_1 + a_2\mathbf{b}_2 + ... + a_n\mathbf{b}_n : (a_1, ..., a_n) \in Z^n\right\} = \sum_{i=1}^{n} a_i\mathbf{b}_i.$$

Решетка может иметь много базисов. "Хорошим" базисом будем называть базис, векторы которого короткие и почти ортогональные (рис. 1). Соответственно "плохой" базис — базис, векторы которого длинные и почти параллельные (рис. 2).

Определение 2. *Детерминантом* решетки называется число, равное абсолютной величине определителя, строками которого являются координаты базисных векторов [30]:

$$\det(L) = \sqrt{\begin{vmatrix} (\mathbf{b}_1,\mathbf{b}_1) & (\mathbf{b}_1,\mathbf{b}_2) & \dots & (\mathbf{b}_n,\mathbf{b}_n) \\ (\mathbf{b}_2,\mathbf{b}_1) & (\mathbf{b}_2,\mathbf{b}_2) & \dots & (\mathbf{b}_2,\mathbf{b}_n) \\ \dots & \dots & \dots & \dots \\ (\mathbf{b}_n,\mathbf{b}_1) & (\mathbf{b}_n,\mathbf{b}_2) & \dots & (\mathbf{b}_n,\mathbf{b}_n) \end{vmatrix}}.$$

Детерминант решетки $\det(L)$ будет равен площади параллелепипеда, натянутого на базисные векторы. В многомерном случае $\det(L)$ равен объему параллелепипеда.

Площади фундаментальных параллелепипедов, образованных всевозможными базисами одной и той же решетки, будут равны, независимо от выбора "хорошего" или "плохого" базиса (рис. 3, 4). Данное условие обеспечивается тем, что матрица перехода от одного базиса решетки к другому произвольному базису унимодулярна, таким образом, значение определителя не меняется.

Определение 3. Ненулевой вектор решетки минимальной длины называется ее *кратчайшим вектором (shortest vector).*

Определение 4. *Целочисленная решетка* — решетка, скалярное произведение между любыми двумя векторами которой есть целое число: $\forall \mathbf{x}, \mathbf{y} \in L : (\mathbf{x}, \mathbf{y}) \in Z$.

Определение 5. *Четная решетка* — решетка, у которой норма любого вектора, принадлежащего этой решетке, четная [32]: $\forall \mathbf{x} \in L : (\mathbf{x}, \mathbf{x}) \in Z$.

Определение 6. Решетка L^* называется *двойственной решеткой* решетке L, если L^* задана над множеством всех векторов $\mathbf{y} \in R$, удовлетворяющих $(\mathbf{x}, \mathbf{y}) \in Z$ для всех векторов $\mathbf{x} \in L$ [31].

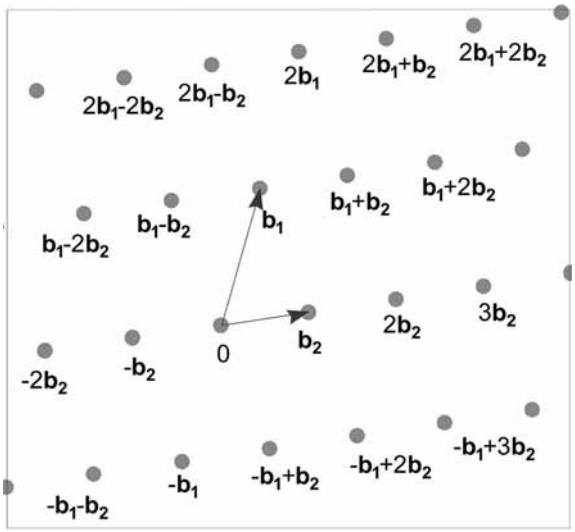


Рис. 1. "Хороший" базис решетки L в R^2

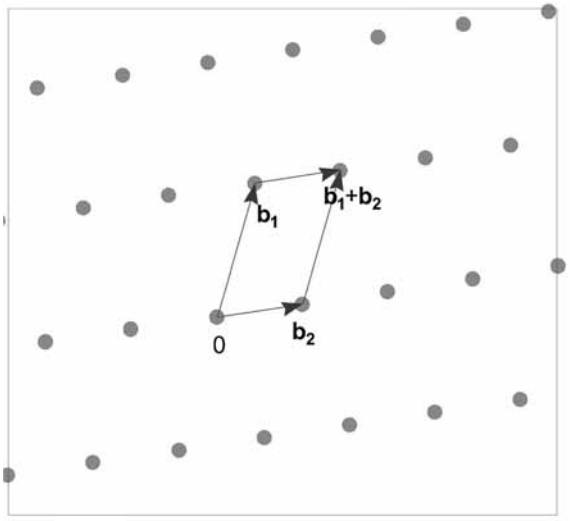


Рис. 3. Решетка с базисом B = {b1, b2}

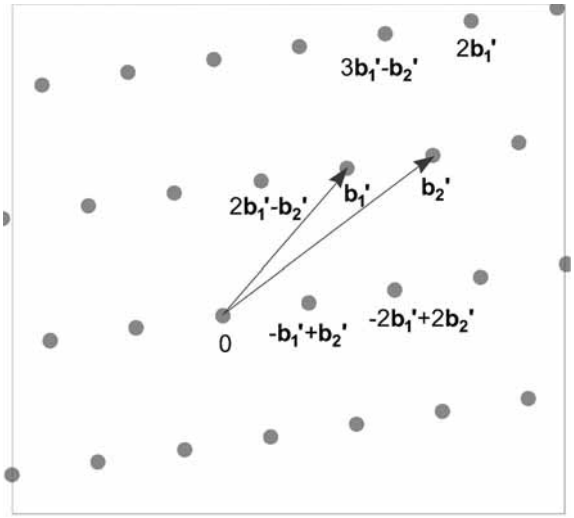


Рис. 2. "Плохой" базис решетки L в R^2

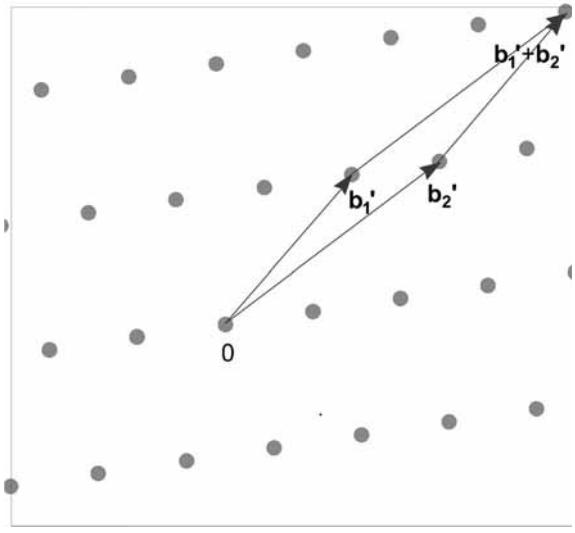


Рис. 4. Решетка с базисом B = {b1', b2'}

Несложно также видеть, что для любой квадратной матрицы $\mathbf{A} \in \mathrm{R}^{n\times n}$, $L(\mathbf{A})^* = L\left(\left(\mathbf{A}^{-1}\right)^{\mathrm{T}}\right)$, откуда $\det(\mathbf{A})^* = \frac{1}{\det(\mathbf{A})}$ [33].

Особое место в криптографии на основе теории решеток занимают q-арные решетки.

Определение 7. *q-арная решетка* — это решетка L, удовлетворяющая соотношению $qZ^n \subseteq L \subseteq Z^n$ для некоторого целого числа q. Другими словами, вектор $\mathbf{x}$ в решетке L определяется как $x \bmod q$.

Такие решетки находятся в однозначном соответствии с линейными кодами в Z_q^n [33]. Большинство криптографических конструкций используют q-арные решетки в качестве базы для определения сложной задачи.

Пусть дана матрица $\mathbf{A} \in Z_q^{n\times m}$ для некоторых целых чисел m, n, q. Можно определить две m-мерные q-арные решетки:

$$L_q(\mathbf{A}) = \left\{\mathbf{y} \in Z^m : \mathbf{y} = \mathbf{A}^{\mathrm{T}}\mathbf{s} \bmod q, \text{ где } \mathbf{s} \in Z^n\right\};$$

$$L_q^{\perp}(\mathbf{A}) = \left\{\mathbf{y} \in Z^m : \mathbf{A}\mathbf{y} = 0 \bmod q\right\}.$$

Первая решетка задается строками матрицы $\mathbf{A}$, вторая решетка содержит векторы, ортогональные строкам матрицы $\mathbf{A}$ по модулю q [33]. Другими словами, первая q-арная решетка соответствует коду, который генерирует строки матрицы $\mathbf{A}$, в то время как вторая решетка соответствует коду проверки контроля четности матрицы A. Как следует из определения, эти решетки *двойственны* друг другу с точностью до некоторого нормировочного фактора: $L_q^{\perp}(\mathbf{A}) = qL_q(\mathbf{A}^*)$ и $L_q(\mathbf{A}) = qL_q^{\perp}(\mathbf{A}^*)$ [33].

Определение 8. *Идеалом кольца K* называется замкнутое относительно операции умножения на элементы из K подкольцо P [29]: $\forall a \in P,\ \forall b \in K : ab \in P,\ ba \in P$.

Определение 9. *Идеальная решетка* — это решетка со свойствами идеала. В результате операций сложения и умножения векторов в данной решетке получается вектор, также принадлежащий решетке [4].

Определение 10. *Вектор с малыми коэффициентами* — это такой вектор, коэффициенты которого выбираются случайно и равномерно из диапазона $(-q/2,\ q/2]$ по модулю q.

Вычислительно трудные задачи теории решеток

Важным аспектом применения криптографических примитивов является их криптостойкость. Она основывается на сложности вычисления какой-либо трудной односторонней математической функции, нахождение эффективных методов решения которой влечет за собой взлом криптоалгоритма.

В приложениях теории решеток существует ряд вычислительно сложных задач, на решении которых основываются, например, схемы электронной подписи. Далее в кратком изложении представим постановки некоторых из них и покажем, на решении каких задач базируются наиболее перспективные, по мнению авторов, практические постквантовые схемы электронной подписи.

- Задача поиска кратчайшего вектора решетки (*shortest vector problem*, SVP) (рис. 5). Дан базис $\mathbf{B} = \{\mathbf{b}_1', \mathbf{b}_2'\}$ решетки L. Требуется найти кратчайший ненулевой вектор $\mathbf{s}$ длины λ.

В одной решетке может быть больше одного кратчайшего ненулевого вектора. Например, все четыре вектора (0, 1), (0, −1), (1, 0) и (−1, 0) в Z^2 являются решениями данной задачи.

- Приближенная задача поиска кратчайшего вектора решетки (*approximation shortest vector problem*, approxSVP_γ). Дан базис $\mathbf{B} = \{\mathbf{b}_1', \mathbf{b}_2'\}$ решетки L и вещественное число γ. Требуется найти ненулевой вектор, в γ раз больший кратчайшего вектора: $\|\mathbf{b}\| \leqslant \gamma\times\lambda$.
- Задача поиска кратчайшего вектора решетки с некоторым пробелом. Даны базис $\mathbf{B} = \{\mathbf{b}_1', \mathbf{b}_2'\}$ решетки L и некоторое вещественное число d. Требуется определить, какое выражение верно: $\lambda \leqslant d$ либо $\lambda \geqslant \gamma \times d$.
- Задача поиска ближайшего вектора решетки (*closest vector problem*, CVP) (рис. 6). Дан базис $\mathbf{B} = \{\mathbf{b}_1', \mathbf{b}_2'\}$ решетки L и точка c, не принадлежащая решетке. Требуется найти ближайший к точке c вектор в решетке.
- Задача поиска ближайшего вектора решетки с некоторым пробелом ($GapSVP_\gamma$). Дан базис $\mathbf{B} = \{\mathbf{b}_1', \mathbf{b}_2'\}$ решетки L, точка c, не принадлежащая решетке, некоторое вещественное число γ. Требуется определить, какое выражение верно: существует вектор решетки для заданной точки c, находящийся на расстоянии не более γ, либо все векторы решетки находятся от точки c на расстоянии большем, чем γ.
- Задача декодирования ограниченного расстояния (*bounded distance decoding*, BDD). Дан базис $\mathbf{B} = \{\mathbf{b}_1', \mathbf{b}_2'\}$ решетки L, точка c, не принадлежащая

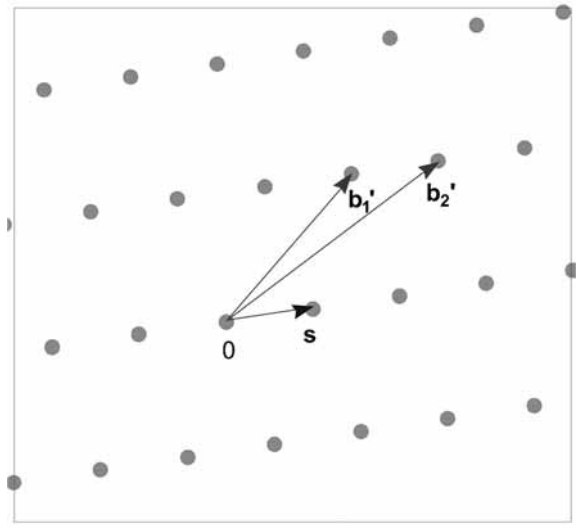


Рис. 5. Задача нахождения кратчайшего вектора решетки (SVP)

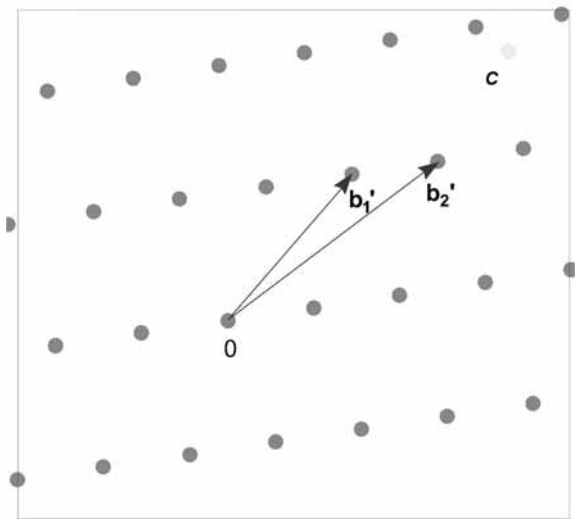


Рис. 6. Задача поиска ближайшего вектора решетки (CVP)

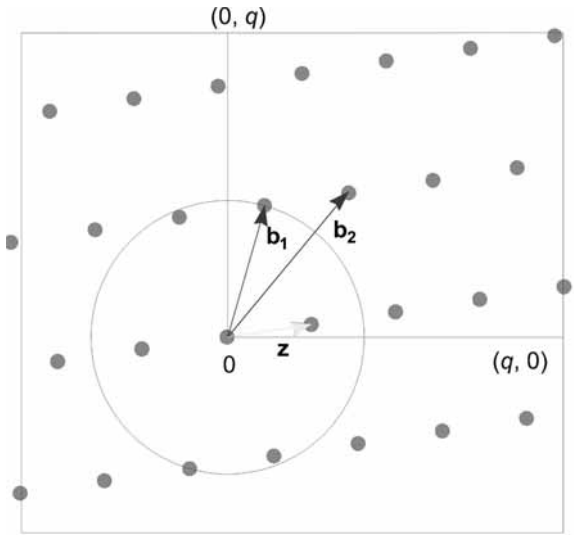


Рис. 7. Задача с коротким целочисленным решением (SIS)

решетке, но находящаяся на расстоянии, не более $\lambda/2$ от решетки. Требуется найти ближайший к точке c вектор в решетке.

- Задача с коротким целочисленным решением (*short integer solution problem*, SIS) (рис. 7). Дан базис $\mathbf{B} = \{\mathbf{b}'_1, \ldots, \mathbf{b}'_n\}$ q-арной m-мерной решетки $L_q^{n\times m}$ и некоторый параметр β, нормирующий связь между "хорошим" и "плохим" базисами решетки. Требуется найти нетривиальный вектор $\mathbf{z} \in Z_q^m$ такой, что $f_A(x) = \mathbf{Bz} = 0 \bmod q \in Z_q^n$ и $\|\mathbf{z}\| \leqslant \beta$.

Последнее условие необходимо, чтобы данная задача не имела тривиального решения с помощью метода Гаусса. Для отсутствия тривиального решения необходимо также, чтобы $\beta \geqslant \sqrt{n \log_2 q}$ и $m \geqslant n \log_2 q$. Эту задачу можно решать и над кольцами полиномиальных многочленов (задача Ring-SIS, R-SIS) [34]. Отметим, что постквантовая схема электронной подписи FALCON [25], одна из трех схем, прошедших во второй тур конкурса NIST, основывается на данной задаче.

- Задача обучения с ошибками (*learning with errors problem*, LWE) (рис. 8). Дан базис $\mathbf{B} = \{\mathbf{b}'_1, \ldots, \mathbf{b}'_n\}$ q-арной m-мерной решетки $L_q^{n\times m}$ и $\mathbf{e}$ — случайный вектор (вектор ошибок, вектор шума) с малыми коэффициентами $(\mathbf{e} \in Z_q)$, выбранными из некоторого случайного равномерного распределения. Известно множество значений $\{\mathbf{Bs} + \mathbf{e_i} \mid 1 \leqslant i \leqslant n\}$, где $\mathbf{s}$ — случайный вектор $\left(\mathbf{s} \in Z_q^n\right)$. Требуется найти $\mathbf{s}$, т. е. исключить шум.

Сложность задачи LWE основывается на трудности извлечения из равномерного распределения некоторого вектора ошибок $\mathbf{e}$. Впервые данная задача была предложена О. Регевом в работе [12]. Ее решение довольно просто в реализации, для того чтобы применяться в криптографии. В то же время оно обладает высокой вычислительной сложностью и, соответственно, имеет хорошую криптостойкость. Оно сравнимо со сложностью некоторых задач теории решеток, имеющих сложность в наихудшем случае.

Позже в работе [35] было показано, что секретный вектор $\mathbf{s}$ не обязательно должен быть выбран из некоторого случайного равномерного распределения. Для сохранения уровня стойкости достаточно, чтобы вектор $\mathbf{s}$ был выбран из такого же случайного распределения, как и вектор ошибок $\mathbf{e}$. Спустя год в работе [36] было предложено заменить вычисления над целыми числами на вычисления в кольцах полиноминальных многочленов, как в криптосистеме NTRU. В работе [36] было доказано, что задача LWE над кольцами полиномиальных многочленов, названная Ring-LWE (RLWE), также сложно вычислима, как базовая задача над простыми числами. На основе задачи RLWE базируется схема qTESLA.

Другая схема на основе решеток CRYSTALS-DILITHIUM базируется на задаче Module-LWE (MLWE). Данная задача была предложена в работе [37] для устранения недостатков как в LWE, так и в RLWE путем интерполяции между ними. Формально задачу MLWE можно рас-

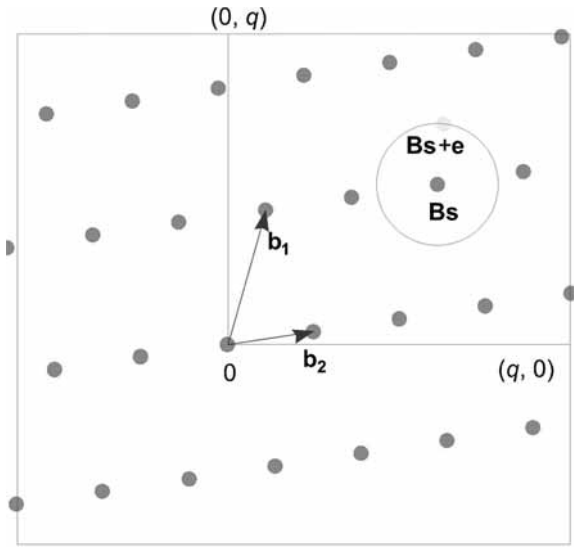


Рис. 8. Задача обучения с ошибками (LWE)

смотреть как замену одиночных кольцевых элементов (b и s) в задаче RLWE элементами модуля над тем же кольцом. Таким образом, задачу RLWE можно рассматривать как задачу MLWE с модулем ранга 1 [38]. Схемы на основе MLWE могут обеспечить более высокий уровень безопасности, чем схемы на основе RLWE. При этом сохраняются преимущества производительности по сравнению с обычной задачей LWE [39].

Заключение

Представлен краткий обзор существующего положения дел в области постквантовой криптографии, в частности, в области криптографии на основе теории решеток. Рассмотрены основные определения, которые необходимы для понимания представляемого материала. Описаны важнейшие задачи теории решеток, приведена историческая справка.

Выявлены основные трудоемкие задачи, на которых базируются современные схемы электронной подписи на решетках: задача обучения с ошибками в кольце полиномиальных многочленов (RLWE), модульная задача обучения с ошибками (MLWE) и задача с коротким целочисленным решением (SIS).

По мнению авторов, наиболее перспективными практическими схемами электронной подписи в настоящее время являются схемы qTESLA, CRYSTALS-DILITHIUM и FALCON.

Полученные результаты могут быть использованы другими авторами для дальнейшего исследования передовых постквантовых технологий, а также студентами, аспирантами и просто любителями криптографии в целях ознакомления и обучения.

Работа выполнена при поддержке НИР Университета ИТМО № 619296 "Разработка методов создания и внедрения киберфизических систем".

Список литературы

1. **Shor P.** Algorithms for Quantum Computation: Discrete Logarithms and Factoring// Foundations of Computer Science 1994. Proceedings, 35th Annual Symposium on. IEEE, 1994. P. 124—134.

2. **Шокуров А. В., Кузюрин Н. Н., Фомин С. А.** Решетки, алгоритмы и современная криптография. М.: Институт системного программирования РАН, 2011. 130 с.

3. **Усатюк В. С., Кузьмин О. В.** Системы шифрования, основанные на задачах теории решеток // В мире научных открытий. 2010. № 1—4 (7). С. 22—29.

4. **Кузьмин О. В., Усатюк В. С.** Роль задач теории решеток в постквантовой криптографии и их иерархия сложности // Комбинаторные и вероятностные проблемы дискретной математики: сб. науч. тр. Иркутск: Изд-во Иркут. гос. ун-та. Дискретный анализ и информатика. 2010. № 4. С. 71—79.

5. **Кузьмин О. В., Усатюк В. С.** Параллельные алгоритмы вычисления локальных минимумов целочисленных решеток // Программные продукты и системы. 2015. № 1. С. 55—62.

6. **Усатюк В. С.** Иерархия сложности задач теории решеток // В мире научных открытий. 2010. № 4—9 (10). С. 129—131.

7. **Пискова А. В., Коробейников А. Г.** Теория решеток в постквантовой криптографии // Сб. тр. V Всероссийского конгресса молодых ученых. 12—15 апреля 2016 г. Санкт-Петербург, 2016. Т. 2. С. 91—93.

8. **Пискова А. В., Коробейников А. Г.** Особенности применения теории решеток в схемах электронной цифровой подписи // Кибернетика и программирование. 2016. № 2. С. 8—12.

9. **Пискова А. В., Менщиков А. А., Коробейников А. Г.** Использование ортогонализации Грама—Шмидта в алгоритме приведения базиса решетки для протоколов безопасности // Вопросы кибербезопасности. 2016. № 1 (14). С. 47—52.

10. **Lenstra A. K., Lenstra H. W., Lov'asz L.** Factoring Polynomials with Rational Coefficients // Math. Annalen. 1982. Vol. 261. P. 515—534.

11. **Ajtai M.** Generating Hard Instances of Lattice Problem // Proc. of 28th ACM Symp. on Theory of Comp, Philadelphia: ACM Press. 1996. P. 99—108.

12. **Regev O.** On lattices, learning with errors, random linear codes, and cryptography // STOC '05 Proceedings of the thirty-seventh annual ACM symposium on Theory of computing, ACM Press, May 2005. P. 84—93.

13. **Ajtai M., Dwork C.** A public-key cryptosystem with worst-case/average-case equivalence // STOC '97 Proceedings of the twenty-ninth annual ACM symposium on Theory of computing. ACM Press, May 1997. P. 284—293.

14. **Regev O.** New lattice based cryptographic constructions// Conference Proceedings of the 35th Annual ACM Symposium on Theory of Computing. ACM Press, June 2003. P. 407—416.

15. **Goldreich O., Goldwasser S., Halevi S.** Public-key cryptosystems from lattice reduction problems // Advances in cryptology. Vol. 1294 of Lecture Notes in Comput. Sci. Springer, 1997. P. 112—131.

16. **Nguyen P. Q.** Cryptanalysis of the Goldreich—Goldwasser—Halevi cryptosystem from crypto '97 // Advances in Cryptology — Crypto 1999, Lecture Notes in Computer Science 1666. Springer-Verlag, 1999. P. 288—304.

17. **Hoffstein J., Pipher J., Silverman J. H.** NTRU: A Ring Based Public Key Cryptosystem // Algorithmic Number Theory (ANTS III), Portland, OR, June 1998. Lecture Notes in Computer Science 1423. Springer-Verlag, Berlin, 1998. P. 267—288.

18. **Киршанова Е. А.** Анализ структуры и стойкости криптосистемы NTRU // Вестник Балтийского федерального университета им. И. Канта. Сер.: Физико-математические и технические науки. 2010. № 10. С. 112—115.

19. **Hoffsteine J., Howgrave-Graham N., Pipher J., Silverman J. H., Whyte W.** NTRUSIGN: Digital signatures using the NTRU lattice // CT-RSA 2003. Vol. 2612 of LNCS. San-Francisco, CA, USA, April 13—17, 2003. Springer, Heidelberg, Germany, 2003. P. 122—140.

20. **Nguyen P. Q., Regev O.** Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures // EUROCRYPT 2006. Vol. 4004 of LNCS. St. Petersburg, Russia, May 28 — June 1, 2006. Springer, Heidelberg, Germany, 2006. P. 271—288.

21. **Gentry C., Peikert C., Vaikuntanathan V.** Trapdoors for hard lattices and new cryptographic constructions // 40th ACM STOC, Victoria, British Columbia, Canada, May 17—20, 2008. ACM Press, 2008. P. 197—206.

22. **Stehlé D., Steinfeld R.** Making NTRU as secure as worst-case problems over ideal lattices // EUROCRYPT 2011. Vol. 6632 of LNCS, Tallinn, Estonia, May 15—19, 2011. Springer, Heidelberg, Germany, 2011. P. 27—47.

23. **Ducas L., Lyubashevsky V., Prest T.** Efficient identity-based encryption over NTRU lattices // ASIACRYPT 2014, Part II. Vol. 8874 of LNCS, Kaoshiung, Taiwan, R. O. C., December 7—11, 2014. Springer, Heidelberg, Germany. 2014. P. 22—41.

24. **Ducas L., Prest T.** Fast Fourier orthogonalization // Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2016. Waterloo, ON, Canada, July 19—22, 2016.ACM, 2016. P. 191—198.

25. **FALCON.** URL: https://falcon-sign.info (дата обращения 25.04.2019).

26. **qTESLA.** URL: https://qtesla.org (дата обращения 25.04.2019).

27. **CRYSTALS-DILITHIUM.** URL: https://pq-crystals.org (дата обращения 25.04.2019).

28. **Chen L., Jordan S., Liu Y. K., Moody D., Peralta R., Perlner R., Smith-Tone D.** Report on Post-Quantum Cryptography, NISTIR 8105, National Institute of Standards and Technology, Gaithersburg, Maryland, April 2016. 10 p.

29. **Кострикин А. И., Манин Ю. И.** Линейная алгебра и геометрия. М.: Наука, 1986. 304 с.

30. **Скорняков Л. А.** Элементы алгебры. М.: Наука, 1986. С. 16—23.

31. **Кострикин А. И.** Введение в алгебру. М.: Наука, 1977. 496 с.

32. **Конвей Дж., Слоэн Н.** Упаковки шаров, решетки и группы. М.: Мир, 1990. 376 с.

33. **Lattice-based** Cryptography. URL: https://cims.nyu.edu/~regev/papers/pqc.pdf (дата обращения 05.04.2019).

34. **Micciancio D.** Generalized compact knapsacks, cyclic lattices, and efficient one-way functions from worst-case complexity assumptions // Computational complexity. 2007. Vol. 16, No. 4. P. 365—411.
35. **Applebaum B., Cash D., Peikert C., Sahai A.** Fast cryptographic primitives and circular-secure encryption based on hard learning problems // CRYPTO 2009. Vol. 5677 of LNCS. Springer, Heidelberg. August 2009. P. 595—618.
36. **Lyubashevsky V., Peikert C., Regev O.** On ideal lattices and learning with errors over rings // EUROCRYPT 2010. Vol. 6110 of LNCS. Springer, Heidelberg, May 2010. P. 1—23.
37. **Brakerski Z., Gentry C., Vaikuntanathan .** (Leveled) fully homomorphic encryption without bootstrapping // ITCS 2012: 3rd Innovations in Theoretical Computer Science. Association for Computing Machinery. January 2012. P. 309—325.
38. **Albrecht M. R., Deo A.** Large modulus Ring-LWE ⩾ Module-LWE // Advances in Cryptology — ASIACRYPT, 2017. P. 267—296.
39. **Langlois A., Stehl'e D.** Worst-case to average-case reductions for module lattices // Des. Codes & Cryptography. 2015. No. 75 (3). P. 565—599.

An Overview of the History and Trends of Post-Quantum Cryptography based on the Lattice Theory

A. V. Komarova, piter-ton@mail.ru, St. Petersburg National Research University of Information Technologies, Mechanics and Optics, St. Petersburg, 197101, Russian Federation,
A. G. Korobeynikov, korobeynikov_a_g@mail.ru, Institute of Terrestrial Magnetism, Ionosphere and Radio Wave Propagation of the Russian Academy of Sciences St.-Petersburg Filial, St. Petersburg, 199034, Russian Federation

Corresponding author

Komarova Antonina V., Postgraduate Student, St. Petersburg National Research University of Information Technologies, Mechanics and Optics, St. Petersburg, 197101, Russian Federation
E-mail: piter-ton@mail.ru

Received on April 26, 2019
Accepted on June 10, 2019

Every year quantum computing and quantum computer becomes more and more actual topic of the world scientific researches. In case of its appearance such a device will be able to compromise most of the asymmetric schemes used today. To ensure a proper level of security in single-key (symmetric) schemes and hash functions organizations will have to use so long keys and hash values, respectively, that will make it impossible to realize such schemes in real information and communication systems. The post-quantum cryptography began to rapidly develop after the sensational appearance of quantum Shor's algorithm in 1994. This direction includes a number of approaches, however from the authors' point of view, the most promising approach is a lattices-based cryptography. Unfortunately, in the Russian literature there are very few manuals devoted to post-quantum cryptography, in particular the lattice theory. The goal of this article is to fill this gap. The article deals with the main provisions, definitions and main lattice problems. The background information about the history as well as the latest trends is given. The studied question is relevant because the most part of popular cryptographic protocols, schemes and algorithms based on the number theory will become unsecure in the case of the quantum computer appearance.

Keywords: *cryptography, post-quantum cryptography, lattices-based cryptography, shortest vector problem, closest vector problem, learning with errors problem, short integer solution problem, public-key cryptography, qTESLA, CRYSTALS-DILITHIUM, FALCON*

Acknowledgements: *The work was supported by the National Research University of Information Technologies, Mechanics and Optics project nos. 619296 "Development of methods for the creation and implementation of cyberphysical systems".*

For citation:

Komarova A. V., Korobeynikov A. G. An Overview of the History and Trends of Post-Quantum Cryptography based on the Lattice Theory, *Programmnaya Ingeneria,* 2019, vol. 10, no. 7—8, pp. 344—352.

DOI: 10.17587/prin.10.344-352

References

1. **Shor P.** Algorithms for Quantum Computation: Discrete Logarithms and Factoring, *Foundations of Computer Science, 1994, Proceedings 35th Annual Symposium on,* IEEE, 1994, pp. 124—134.
2. **Shokurov A. V., Kuzyurin N. N., Fomin S. A.** *Lattices, algorithms and modern cryptography,* Moscow, Institut sistemnogo programmirovaniya RAN, 2011, 130 p. (in Russian).
3. **Usatyuk V. S., Kuzmin O. V.** Encryption systems based on lattice theory problems, *V mire nauchnyh otkrytij*, 2010, no. 1—4 (7), pp. 22—29 (in Russian).
4. **Kuzmin O. V., Usatyuk V. S.** The role of the task the lattice theory of post-quantum cryptography and a hierarchy of complexity, *Kombinatornye i veroyatnostnye problemy diskretnoj matematiki: sb. nauch. tr,* Irkutsk, Izd-vo Irkut. gos. un-ta, 2010 (Diskretnyj analiz i informatika), no. 4, pp. 71—79 (in Russian).

5. **Kuzmin O. V., Usatyuk V. S.** Parallel algorithms for computing local minima of integer lattices, *Programmnye produkty i sistemy*, 2015, no 1, pp. 55—62 (in Russian).
6. **Usatyuk V. S.** The hierarchy of difficulty of lattices problems, *V mire nauchnyh otkrytij*, 2010, no. 4—9 (10), pp. 129—131 (in Russian).
7. **Piskova A. V., Korobejnikov A. G.** The theory of lattices in post-quantum cryptography, *Sbornik trudov V Vserossijskogo kongressa molodyh uchenyh,* 12—15 April 2016, Saint-Petersburg, 2016, vol. 2, pp. 91—93 (in Russian).
8. **Piskova A. V., Korobejnikov A. G.** The application features of lattice theory in the electronic digital signature schemes, *Kibernetika i programmirovanie*, 2016, no. 2, pp. 8—12 (in Russian).
9. **Piskova A. V., Menshchikov A. A., Korobejnikov A. G.** The implementation of Gram-Schmidt orthogonalization lattice basis reduction algorithm for security protocols, *Voprosy kiberbezopasnosti*, 2016, no. 1 (14), pp. 47—52 (in Russian).
10. **Lenstra A. K., Lenstra H. W., Lov'asz L**. Factoring Polynomials with Rational Coefficients, *Math. Annalen*, 1982, vol. 261, pp. 515—534.
11. **Ajtai M.** Generating Hard Instances of Lattice Problem, *Proc. of 28th ACM Symp. on Theory of Comp*, Philadelphia: ACM Press, 1996, pp. 99—108.
12. **Regev O.** On lattices, learning with errors, random linear codes, and cryptography. *STOC'05 Proceedings of the thirty-seventh annual ACM symposium on Theory of computing,* ACM Press, May 2005, pp. 84—93.
13. **Ajtai M., Dwork C.** A public-key cryptosystem with worst-case/average-case equivalence, *STOC'97 Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, ACM Press, May 1997, pp. 284—293.
14. **Regev O.** New lattice based cryptographic constructions, *Conference Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, ACM Press, June 2003, pp. 407—416.
15. **Goldreich O., Goldwasser S., Halevi S.** Public-key cryptosystems from lattice reduction problems*, Advances in cryptology,* volume 1294 of Lecture Notes in Comput. Sci., Springer, 1997, pp. 112—131.
16. **Nguyen P. Q.** Cryptanalysis of the Goldreich-Goldwasser-Halevi cryptosystem from crypto '97, *Advances in Cryptology — Crypto 1999*, volume 1666 Lecture Notes in Computer Science, Springer-Verlag, 1999, pp. 288—304.
17. **Hoffstein J., Pipher J., Silverman J. H.** NTRU: A Ring Based Public Key Cryptosystem, *Algorithmic Number Theory (ANTS III)*, Portland, OR, June 1998 / J. P. Buhler (ed.), Lecture Notes in Computer Science 1423, Springer-Verlag, Berlin, 1998, pp. 267—288.
18. **Kirshanova E. A.** Analysis of the structure and the resistance of the NTRU cryptosystem, *Vestnik Baltijskogo federal'nogo universiteta im. I. Kanta. Seriya: Fiziko-matematicheskie i tekhnicheskie nauki*, 2010, no. 10, pp. 112—115 (in Russian).
19. **Hoffsteine J., Howgrave-Graham N., Pipher J., Silverman J. H., Whyte W.** NTRUSIGN: Digital signatures using the NTRU lattice., *CT-RSA 2003*, volume 2612 of LNCS, San-Francisco, CA, USA, April 13—17, 2003, Springer, Heidelberg, Germany, 2003, pp. 122—140/
20. **Nguyen P. Q., Regev O.** Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures, *EUROCRYPT 2006*, volume 4004 of LNCS, St. Petersburg, Russia, May 28 — June 1, 2006. Springer, Heidelberg, Germany, 2006, pp. 271—288.
21. **Gentry C., Peikert C., Vaikuntanathan V.** Trapdoors for hard lattices and new cryptographic constructions. *40th ACM STOC*, Victoria, British Columbia, Canada, May 17—20, 2008, ACM Press, 2008, pp. 197—206.
22. **Stehlé D., Steinfeld R.** Making NTRU as secure as worst-case problems over ideal lattices, *EUROCRYPT 2011*, volume 6632 of LNCS,, Tallinn, Estonia, May 15—19, 2011, Springer, Heidelberg, Germany, 2011, P. 27—47.
23. **Ducas L., Lyubashevsky V., Prest T.** Efficient identity-based encryption over NTRU lattices, *ASIACRYPT 2014*, Part II, volume 8874 of LNCS, Kaoshiung, Taiwan, R. O. C., December 7—11, 2014, Springer, Heidelberg, Germany, pp. 22—41.
24. **Ducas L., Prest T.** Fast Fourier orthogonalization, *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC 2016, Waterloo, ON, Canada, July 19—22, 2016,. ACM, 2016. pp. 191—198.
25. **FALCON,** available at: https://falcon-sign.info (date of access 25.04.2019).
26. **qTESLA,** available at: https://qtesla.org (date of access 25.04.2019).
27. **CRYSTALS-DILITHIUM,** available at: https://pq-crystals.org (date of access 25.04.2019).
28. **Chen L., Jordan S., Liu Y. K., Moody D., Peralta R., Perlner R., Smith-Tone D.** Report on Post-Quantum Cryptography, NISTIR 8105, National Institute of Standards and Technology, Gaithersburg, Maryland, April 2016, 10 p.
29. **Kostrikin A. I., Manin Yu. I.** *Linear algebra and geometry,* Moscow, Nauka, 1986, 304 p. (in Russian).
30. **Skornyakov L. A.** Elements of algebra, Moscow, Nauka, 1986, pp. 16—23.
31. **Kostrikin A. I.** *Introduction to algebra*, Moscow: Nauka, 1977, 496 p. (in Russian).
32. **Conway G., Sloan N.** *The packings, lattice and groups*, Moscow, Mir, 1990, 376 p. (in Russian).
33. **Lattice-based Cryptography**. available at: https://cims.nyu.edu/~regev/papers/pqc.pdf (date of access 25.04.2019).
34. **Micciancio D.** Generalized compact knapsacks, cyclic lattices, and efficient one-way functions from worst-case complexity assumptions, *Computational complexity,* 2007, vol. 16, no. 4, pp. 365—411.
35. **Applebaum B., Cash D., Peikert C., Sahai A.** Fast cryptographic primitives and circular-secure encryption based on hard learning problems, *CRYPTO 2009*, volume 5677 of LNCS, Springer, Heidelberg, August 2009, pp. 595—618.
36. **Lyubashevsky V., Peikert C., Regev O.** On ideal lattices and learning with errors over rings, *EUROCRYPT 2010*, volume 6110 of LNCS, Springer, Heidelberg, May 2010, pp. 1—23.
37. **Brakerski Z., Gentry C., Vaikuntanathan V.** (Leveled) fully homomorphic encryption without bootstrapping, *ITCS 2012: 3rd Innovations in Theoretical Computer Science*, Association for Computing Machinery, January 2012, pp. 309—325.
38. **Albrecht M. R., Deo A.** Large modulus Ring-LWE ⩾ Module-LWE, *Advances in Cryptology — ASIACRYPT*, 2017, pp. 267—296.
39. **Langlois A., Stehl'e D.** Worst-case to average-case reductions for module lattices, *Des. Codes & Cryptography*, 2015, no. 75 (3), pp. 565—599.

ООО "Издательство "Новые технологии". 107076, Москва, Стромынский пер., 4
Технический редактор *Е. М. Патрушева*. Корректор *Н. В. Яшина*

Сдано в набор 19.06.2019 г. Подписано в печать 29.07.2019 г. Формат 60×88 1/8. Заказ РІ7-819
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
119071, г. Москва, Ленинский пр-т, д. 19, стр. 1. Сайт: **www.aov.ru**