УДК 004.832.25

**Заикин О.С.[1, 2], Чугунов А.А.[3]**

[1] Институт динамики систем и теории управления им. В.М. Матросова Сибирского отделения Российской академии наук, г. Иркутск, Россия
[2] Иркутский государственный университет, г. Иркутск, Россия
[3] Байкальский государственный университет, г. Иркутск, Россия

# ПАРАЛЛЕЛЬНЫЙ ЛОГИЧЕСКИЙ КРИПТОАНАЛИЗ ГЕНЕРАТОРА ПЕРЕМЕННОГО ШАГА

**АННОТАЦИЯ**

*В статье рассматривается задача криптоанализа генератора переменного шага. В рамках данной задачи по известному фрагменту ключевого потока необходимо восстановить исходное внутреннее состояние генератора. Мы свели эту задачу к проблеме булевой выполнимости. Для версии данного генератора с 96-битным внутренним состоянием была найдена декомпозиция, обладающая приемлемой прогнозной трудоемкостью. При этом был использован алгоритм локального поиска, основанный на методе Монте-Карло. С помощью найденной декомпозиции был решен ряд экземпляров задачи криптоанализа указанной версии генератора. Большая часть экспериментов была выполнена на вычислительном кластере.*

**КЛЮЧЕВЫЕ СЛОВА**

*Генератор ключевого потока; генератор переменного шага; криптоанализ; проблема булевой выполнимости; Монте-Карло; локальный поиск; параллельные вычисления.*

**Zaikin O.S.[1,2], Chugunov A.A.[3]**

[1] Matrosov Institute for system dynamics and control theory of Siberian branch of Russian academy of sciences, Irkutsk, Russia
[2] Irkutsk state university, Irkutsk, Russia
[3] Baikal state university, Irkutsk, Russia

# PARALLEL SAT-BASED CRYPTANALYSIS OF ALTERNATING STEP GENERATOR

**ABSTRACT**

*In the paper, a cryptanalysis of Alternating step generator is considered. It is formulated as follows: given a keystream segment, the goal is to find an initial state of the generator. We reduced this problem to SAT. We also found a partitioning with good time estimation for 96-bit version of this generator. It was done using a local search algorithm, which is based on Monte Carlo. Several cryptanalysis instances for the considered version of the generator were solved using the partitioning. Most of the experiments were carried out on a computing cluster.*

**KEYWORDS**

*Keystream generator; Alternating step generator; cryptanalysis; SAT; Monte Carlo, local search, parallel computing.*

## Введение

Поточный шифр – это симметричный шифр, в котором каждый символ открытого текста преобразуется в один символ шифртекста на основе не только секретного ключа, но и расположения этого символа в открытом тексте [1]. При этом используется пседослучайная последовательность символов (ее часто называют гаммой или ключевым потоком). Разработка и анализ поточных шифров – это бурно развивающееся направление современной криптографии. Это обусловлено тем, что при передаче больших объемов данных (графики, звука, видео) их зачастую приходится шифровать в режиме онлайн. Главные требования, которые при этом предъявляются к поточным шифрам – это криптостойкость и высокая скорость работы.

Для получения ключевого потока в большинстве поточных шифров используются генераторы ключевого потока. Фактически каждый такой генератор – это дискретная функция, которая по данному на вход двоичному слову (исходному внутреннему состоянию регистров

генератора) выдает двоичное слово (ключевой поток) заданной длины, совпадающей с длиной открытого текста, который необходимо зашифровать. Анализ стойкости поточного шифра, основанного на генераторе ключевого потока, зачастую сводится к анализу стойкости соответствующего генератора. Задача криптоанализа при этом ставится следующим образом: по известному алгоритму генератора и известному фрагменту ключевого потока найти исходное внутреннее состояние генератора перед началом генерации соответствующего фрагмента. Задачу криптоанализа в такой постановке можно рассматривать как задачу обращения дискретной функции [2], где дискретная функция – это генератор.

В задаче булевой выполнимости (SAT) требуется определить выполнимость булевой формулы. Зачастую рассматривается именно поисковый вариант данной задачи: по данной булевой формуле найти ее выполняющий набор (набор значений переменных, который обращает ее в значение «истина»), либо доказать, что такого набора не существует. Под SAT-подходом к решению некоторой задачи далее будем понимать сведение этой задачи к SAT с последующим ее решением с помощью SAT-решателей [3]. SAT-решателем называется специализированный программный комплекс, предназначенный для решения SAT-задач. Большинство SAT-решателей основаны на алгоритме CDCL, который, в свою очередь основан на алгоритме DPLL [4].

SAT-подход активно используется для решения задач из различных предметных областей: верификация, биоинформатика, комбинаторика и др. Начиная с работы [5] SAT-подход активно используется для решения задач криптоанализа. В настоящей работе с помощью SAT-подхода анализируется стойкость генератора переменного шага.

Опишем структуру статьи. В первом разделе приведено описание пропозициональной кодировки задачи криптоанализа генератора переменного шага (то есть описано, как эта задача сводится к SAT-задаче). Во втором разделе приведены результаты вычислительных экспериментов, в рамках которых были решены задачи криптоанализа двух вариантов генератора переменного шага – с внутренним состоянием длиной 64 и 96 бит соответственно.

## 1. Пропозициональное кодирование задачи криптоанализа генератора переменного шага

Генератор переменного шага был предложен в работе [6]. Данный генератор состоит из трех регистров сдвига с линейной обратной связью (РСЛОС). Нелинейность генератора достигается за счет того, что РСЛОС № 0 управляет сдвигами РСЛОС № 1 и 2. Конкретнее, каждый такт генератора в зависимости от выхода РСЛОС № 0 сдвигается только один из РСЛОС № 1 и 2. Если выход РСЛОС № 0 равен 0 (1), то сдвигается РСЛОС № 1 (2). При этом каждый такт очередной бит ключевого потока формируется как результат операции XOR от выходов РСЛОС № 1 и 2.

Мы рассмотрели два версии данного генератора – с 64-битным и 96-битным внутренним состоянием. В 64-битной версии были использованы такие же полиномы, как в генераторе A5/1, а именно:

- РСЛОС № 0 (19-битный): $X^{19}+X^{18}+X^{17}+X^{14}+1$;
- РСЛОС № 1 (22-битный): $X^{22}+X^{21}+1$;
- РСЛОС № 2 (23-битный): $X^{23}+X^{22}+X^{21}+X^{8}+1$.

В 96-битной версии были использованы следующие полиномы:

- РСЛОС № 0 (31-битный): $X^{31}+ X^{7}+1$;
- РСЛОС № 1 (32-битный): $X^{32}+X^{7}+ X^{5}+X^{3}+X^{2}+X+1$;
- РСЛОС № 2 (33-битный): $X^{33}+X^{16}+X^{4}+X+1$.

Для получения пропозициональных кодировок задач криптоанализа рассмотренных версий генератора был использован программный комплекс Transalg [7]. Были написаны две программы на специализированном TA-языке. Этот проблемно-ориентированный C-подобный язык используется для записи алгоритмов, которые необходимо транслировать в SAT с помощью Transalg. Файл с исходным кодом на этом языке подается на вход Transalg. Результатом работы Transalg является булева формула в виде конъюнктивной нормальной формы (КНФ), которая кодирует соответствующий алгоритм. Далее приводятся части исходного кода программы на TA-языке, в которой записан алгоритм генерации 128 первых бит ключевого потока по произвольному начальному заполнению 64-битной версии генератора переменного шага. На рис. 1 приведено объявление константы *len*, значение которой равно длине рассматриваемого фрагмента ключевого потока. Также приведено объявление глобальных переменных с атрибутами *__in* (*__out)*, что указывает на массивы входных (выходных) данных.

```
define len 128;

__in bit regA[19];
```

```
__in bit regB[22];
__in bit regC[23];
__out bit result[len];
```

*Рис.1. Объявление констант и глобальных переменных в TA-программе для криптоанализа 64-битной версии генератора переменного шага*

На рис. 2 приведена функция *shift_regA()*, в которой сдвигается РСЛОС № 0 в соответствии с полиномом, приведенным выше. Этому РСЛОС соответствует глобальный массив *regA*.

```
void shift_regA()
{
bit y = regA[18]^regA[17]^regA[16]^regA[13];
for (int j = 18; j > 0; j = j - 1)
{
      regA[j] = regA[j - 1];
}
regA[0] = y;
}
```

*Рис. 2. Функция сдвига РСЛОС № 0 в TA-программе для криптоанализа 64-битной версии генератора переменного шага*

На рис. 3 описана функция *main()*. В ней выполняется *len* итераций цикла, в каждой из которых формируется один бит ключевого потока на основе выходов РСЛОС № 1 и 2 (в качестве выходов используется последние элементы в соответствующих массивах). Биты ключевого потока записываются в глобальный массив *result*. При этом в каждой итерации значение массива *regA* обновляется каждую итерацию, а массивов *regB* и *regC* – по условию.

```
void main()
{
for (int i = 0; i < len; i = i + 1)
{
      shift_regA();

      if (regA[18]&1)
      {
            shift_regB();
      }

      if (!(regA[18]&1))
      {
            shift_regC();
      }

      result[i] = regB[21] ^ regC[22];
}
}
```

*Рис. 3. Функция main() TA-программы для криптоанализа 64-битной версии генератора переменного шага*

Для 96-битной версии исходный код TA-программы отличается незначительно. Для этой версии у константы *len* было значение 200.

**2. Вычислительные эксперименты**

С помощью комплекса Transalg (см. предыдущий раздел) мы получили две КНФ. Первая из них состоит из 5706 булевых переменных и 25530 дизъюнктов. Данная КНФ кодирует вычисление 128-битного начального фрагмента ключевого потока по 64-битному начальному заполнению генератора переменного шага (с указанными выше полиномами). Отметим, что данная КНФ не кодирует саму по себе задачу криптоанализа – в ней просто записан алгоритм генерации. Такая КНФ называется шаблонной. Ее можно использовать, в частности, для того, чтобы по произвольному входу получать ключевой поток.

Мы сгенерировали десять экземпляров задач по этой КНФ следующим образом. Сначала были сгенерированы случайным образом десять 64-битных начальных заполнений, затем по ним с помощью алгоритма генератора были получены десять 128-битных фрагментов ключевого потока. По каждому из этих фрагментов была сделана отдельная КНФ путем подстановки в шаблонную КНФ

соответствующих 128 однолитеральных дизъюнктов. В каждой такой КНФ закодирована отдельная задача криптоанализа. Для решения этих десяти SAT-задач мы использовали последовательный CDCL-решатель rokk., который хорошо себя зарекомендовал ранее при решении задач криптоанализа других генераторов ключевого потока [8]. Решение указанных десяти SAT-задач было запущено на одном ядре процессора AMD Opteron 6276. Все задачи были успешно решены, в таблице 1 приведено время решения для каждой из них.

*Таблица 1. Время последовательного решения задач криптоанализа для 64-битной версии генератора переменного шага*

| № | Время |
|---|---|
| 1 | 1 мин. 12 сек. |
| 2 | 42 сек. |
| 3 | 43 сек. |
| 4 | 1 мин. 51 сек. |
| 5 | 52 сек. |
| 6 | 23 сек. |
| 7 | 3 мин. 32 сек. |
| 8 | 59 сек. |
| 9 | 1 мин. 47 сек. |
| 10 | 2 мин. 31 сек. |

Шаблонная КНФ, которая была построена для 96-битной версии генератора переменного шага, состоит из 12642-х переменных и 62614-ти дизъюнктов. В ней закодировано вычисление 200-х битного начального фрагмента ключевого потока по 96-битному начальному заполнению генератора. С помощью этой шаблонной КНФ мы сделали три экземпляра задачи криптоанализа. Сначала мы запустили на каждой из трех полученных КНФ решатель rokk, но он не смог решить ни одну из этих SAT-задач, при том что лимит времени на каждую из них был равен 24-м часам. Поэтому для решения этих задач мы решили задействовать MPI-программу PDSAT [9]. Эта программа базируется на идеях, использованных в грид-системе BNB-Grid [10] для решения с помощью SAT-подхода задачи криптоанализа генератора A5/1. В последнее время PDSAT активно используется для поиска декомпозиций трудных SAT-задач для их последующего решения в проекте добровольных распределенных вычислений SAT@home [11].

PDSAT может работать в двух режимах: прогнозирования и решения. Режим прогнозирования направлен на то, чтобы для данной SAT-задачи найти декомпозиционное множество с хорошей прогнозной трудоемкостью. Под декомпозиционным множеством понимается некоторое подмножество множества булевых переменных рассматриваемой КНФ. Каждому декомпозиционному множеству с помощью метода Монте-Карло ставится в соответствие прогнозная трудоемкость (обычно в секундах) решения всех подзадач, получаемых при варьировании значений переменных, входящих в это множество. Таким образом, каждому декомпозиционному множеству соответствует точка в пространстве поиска, а поиск хороших декомпозиционных множеств можно рассматривать как задачу оптимизации целевой функции, которая не задана аналитически. Для решения этой задачи в PDSAT используется локальный поиск, усиленный поиском с запретами и имитацией отжига. В режиме решения PDSAT использует данное на вход декомпозиционное множество для того, чтобы сформировать по нему семейство подзадач. Обработка подзадач прерывается, если при решении одной из них найден выполняющий набор (по нему можно эффективно получить выполняющий набор исходной КНФ). При этом в PDSAT в качестве базового последовательного SAT-решателя используется rokk.

Мы запустили PDSAT в режиме прогнозирования на первой из трех построенных КНФ. Этот эксперимент выполнялся 12 часов на десяти узлах вычислительного кластера «Академик В.М. Матросов» Иркутского суперкомпьютерного центра СО РАН [12]. На каждом узле данного кластера установлены два 16-ти ядерных центральных процессора AMD Opteron 6276 и 64 гигабайта оперативной памяти. При этом в качестве стартовой точки локального поиска использовалось декомпозиционное множество, состоящее из 96-ти переменных, кодирующих начальное заполнение генератора. В результате было найдено декомпозиционное множество, состоящее из 35-ти переменных с прогнозом, равным 3.3e+06 секунд работы одного процессорного ядра. Это множество состоит из следующих переменных (с разбивкой по трем РСЛОС, нумерация сквозная и начинается с единицы):

- РСЛОС № 0 (31-битный): 2, 4, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31;
- РСЛОС № 1 (32-битный): 44, 52, 53, 56, 58, 59, 60, 61, 62;
- РСЛОС № 2 (33-битный): 83, 89, 91, 92, 93, 94, 95.

Чтобы нагляднее продемонстрировать каким именно ячейкам исходного состояния РСЛОС соответствует найденное множество, опишем его другим образом:

- РСЛОС № 0 (31-битный): -+-+--------+--+++++++++++++++;
- РСЛОС № 1 (32-битный): ----------+-------++--+-+++++-;
- РСЛОС № 2 (33-битный): ----------+-----+-+++++-.

В приведенной выше записи если в ячейке находится «+» (начало каждого РСЛОС слева), то эта ячейка РСЛОС соответствует булевой переменной, находящейся в найденном декомпозиционном множестве. Видно, что большинство переменных находится в РСЛОС № 0, который является управляющим. При этом они сконцентрированы в правой части регистра. Это означает, что в каждой получаемой таким образом подзадаче первые несколько тактов (в данном случае 15) работы генератора SAT-решатель будет точно «знать», какие именно из двух других РСЛОС сдвигаются.

Видно, что в РСЛОС № 1 и 2 большинство ячеек также находится в правой части регистров. Таким образом, первые 5 тактов известны выходы всех трех РСЛОС. Благодаря этому в режиме решения большая часть (около 99.94 %) подзадач из $2^{35}$ решаются за долю секунды на этапе препроцессинга SAT-решателя. Отметим, что декомпозиционные множества, найденные в [10, 11] для генератора A5/1, выглядят очень похоже (в структурном смысле) на множество, описанное выше.

Мы сравнили указанное выше множество с множеством, состоящем из 31-й переменной, которые соответствуют всем ячейкам управляющего РСЛОС (РСЛОС № 0). Это множество можно построить вручную без какой-либо автоматической процедуры. В каждой подзадаче, полученной варьированием переменных из этого множества, состояние РСЛОС № 0 известно на каждом такте работы генератора. Это значит, что каждый такт будет известно, какой именно из РСЛОС № 1 и 2 будет сдвигаться. Для сравнения этого множества с тем, которое было найдено с помощью PDSAT, мы использовали методику, предложенную в работе [13]. Кратко идею данной методики можно изложить следующим образом. Для декомпозиционного множества рассчитывается прогноз (в секундах) для нескольких значений объема случайной выборки, которая используется в методе Монте-Карло. Если при возрастании объема выборки значение прогноза увеличивается незначительно (в статье [13] приведен соответствующий критерий), то можно считать, что и прогноз для наименьшего рассматриваемого значения объема выборки достаточно точен. В таблице 2 для построенного вручную множества из 31-й переменной приведены значения прогноза в зависимости от объема выборки.

*Таблица 2. Значения прогноза (в секундах) для построенного вручную множества, состоящего из 31-й переменной*

| Объем выборки | 1e4 | 1e5 | 1e6 | 1e7 |
|---|---|---|---|---|
| Прогноз в секундах | 3.05e+07 | 2.84e+07 | 2.87e+07 | 2.98e+07 |

Из указанных в таблице значений видно, что при возрастании объема случайной выборки значение прогноза меняются незначительно. Возьмем наиболее точный прогноз, полученный для выборки объемом 10 000 000. Его значение равно 2.98e+07 секунд. Следует отметить, что прогноз для множества, найденного с помощью PDSAT, был получен на выборке точно такого же объема. Напомним, что он равен 3.3e+06 секунд. Таким образом, у множества, найденного автоматически с помощью PDSAT, значение прогноза примерно в 9 раз меньше, чем для множества, построенного вручную.

Для решения каждой из трех рассмотренных SAT-задач PDSAT был запущен на 10 узлах кластера. При этом для распараллеливания использовалось найденное автоматически множество, состоящее из 35 переменных. Время решения этих трех задач указано в таблице 3. Данные задачи оказались довольно сложными даже при их решении на вычислительном кластере. Тем не менее, описанный подход позволил решить их все.

*Таблица 3. Время параллельного решения на десяти узлах кластера трех задач криптоанализа для 96-битной версии генератора переменного шага*

| № | Время решения |
|---|---|
| 1 | 3 ч. 9 мин. |
| 2 | 27 ч. 15 мин. |
| 3 | 18 ч. 3 мин. |

**Заключение**

В данной работе с помощью алгоритмов решения проблемы булевой выполнимости нам удалось решить ряд задач криптоанализа для двух версий генератора переменного шага. Для 64-

битной версии оказалось достаточно обычного персонального компьютера и последовательного SAT-решателя. Задача криптоанализа 96-битной версии оказалась значительно сложнее, поэтому для ее решения мы задействовали параллельные вычисления. Для поиска «хорошей» декомпозиции нами был задействован алгоритм локального поиска, базирующийся на методе Монте-Карло. При этом как поиск декомпозиции, так и решение задач для 96-битной версии были проведены на вычислительном кластере. Отметим, что в открытой литературе нам не удалось найти упоминание об успешном криптоанализе 96-битной версии (или же версии с большей длиной внутреннего состояния) данного генератора.

*Работа частично поддержана РФФИ (гранты № 14-07-00403-а, 15-07-07891-а и 16-07-00155-а), советом по грантам Президента РФ (стипендия № СП-1184.2015.5, грант № НШ-8081.2016.9). The research leading to these results has received funding from the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme FP7/2007-2013/ under REA grant agreement number 609642.*

## Литература

1. Menezes A. J., van Oorschot P. C., Vanstone S.A. Handbook of applied cryptography. CRC Press. 1996. 810 p.
2. Семенов А.А. О сложности обращения дискретных функций из одного класса // Дискретный анализ и исследование операций. 2004. Т. 11. № 4. С. 44-55.
3. Семенов А.А. Декомпозиционные представления логических уравнений в задачах обращения дискретных функций // Известия Российской академии наук. Теория и системы управления. 2009. № 5. С. 47-61.
4. Семенов А.А., Беспалов. Д.В. Технологии решения многомерных задач логического поиска // Вестник Томского государственного университета. 2005. № 14. С. 61–73.
5. Cook, S.A., Mitchell, D.G. Finding hard instances of the satisfiability problem: A survey. American Mathematical Society (1997). pp. 1–17.
6. Günther C.G. Alternating step generators controlled by de Bruijn sequences // Advances in Cryptology - EuroCrypt '87. pp. 5-14.
7. Otpuschennikov I. V., Semenov A. A., Kochemazov S. E. Transalg: a tool for translating procedural descriptions of discrete functions to SAT // Proceedings of the 5th International Workshop on Computer Science and Engineering: Information Processing and Control Engineering (WCSE 2015-IPCE). 2015. pp. 289-294.
8. Otpuschennikov I.V., Semenov A.A., Gribanova I.A., Zaikin O.S., Kochemazov S.E. Encoding cryptographic functions to SAT using Transalg system. Frontiers in Artificial Intelligence and Applications. Vol 285. pp. 1594-1595.
9. Заикин О.С., Семенов А.А. Применение метода Монте-Карло к прогнозированию времени параллельного решения проблемы булевой выполнимости // Вычислительные методы и программирование: новые вычислительные технологии. 2014. Т. 15. № 1. С. 22-35.
10. Посыпкин М.А., Заикин О.С., Беспалов Д.В., Семенов А.А. Решение задач криптоанализа поточных шифров в распределенных вычислительных средах // Труды Института системного анализа Российской академии наук. 2009. Т. 46. С. 119-137.
11. Заикин, О.С., Семенов А.А., Посыпкин М.А. Процедуры построения декомпозиционных множеств для распределенного решения SAT-задач в проекте добровольных вычислений SAT@home // Управление большими системами. М.: ИПУ РАН. Вып. 43. 2013. С. 138–156.
12. Иркутский суперкомпьютерный центр СО РАН. URL: http://hpc.icc.ru.
13. Semenov A., Zaikin O. On the accuracy of statistical estimations of SAT partitionings effectiveness in application to discrete function inversion problems // Proceedings of the International conference on Discrete optimization and operations research (DOOR). 2016. pp. 261-275.

## References

1. Menezes A. J., van Oorschot P. C., Vanstone S.A. Handbook of applied cryptography. CRC Press. 1996. 810 p.
2. Semenov A.A. About complexity of the inversion of discrete functions from one class // Discrete analysis and operation research. 2004. Vol. 11. No 4. pp. 44-55.
3. Semenov A.A. Decomposition representations of logical equations in problems of inversion of discrete functions // Journal of Computer and Systems Sciences International. 2009. Vol. 48. No. 5. pp. 718-731.
4. Semenov A.A., Bespalov D.V. Technology for solving multidimensional problems of the logical search // Bulletin of Tomsk State University. 2005. No. 14. pp. 61–73.
5. Cook, S.A., Mitchell, D.G. Finding hard instances of the satisfiability problem: A survey. American Mathematical Society (1997). pp. 1–17.
6. Günther C.G. Alternating step generators controlled by de Bruijn sequences // Advances in Cryptology - EuroCrypt '87. pp. 5-14.
7. Otpuschennikov I. V., Semenov A. A., Kochemazov S. E. Transalg: a tool for translating procedural descriptions of discrete functions to SAT // Proc. 5th Intern. Workshop on Computer Science and Engineering: Information Processing and Control Engineering (WCSE 2015-IPCE). 2015. P. 289-294.
8. Otpuschennikov I.V., Semenov A.A., Gribanova I.A., Zaikin O.S., Kochemazov S.E. Encoding cryptographic functions to SAT using Transalg system. Frontiers in Artificial Intelligence and Applications. Vol 285. pp. 1594-1595.
9. Zaikin O.S., Semenov A.A. Application of the Monte Carlo method for estimating the total time of solving the SAT problem in parallel // Numerical methods and programming. 2014. Vol. 15. No. 1. pp. 22-35.
10. Posypkin M.A., Zaikin O.S., Bespalov D.V., Semenov A.A. Solving cryptanalysis problems for stream ciphers in distributed computing environments. Proceedings of ISA RAS. 2009. Vol. 46. pp. 119-137.
11. Zaikin O.S., Semenov A.A., Posypkin M.A. Constructing decomposition sets for distributed solution of SAT problems in volunteer computing project SAT@home // Large-Scale Systems Control. 2013. Issue 43. P. 138–156.

12. Irkutsk supercomputing center SB RAS. URL: http://hpc.icc.ru.
13. Semenov A., Zaikin O. On the accuracy of statistical estimations of SAT partitionings effectiveness in application to discrete function inversion problems // Proceedings of the International conference on Discrete optimization and operations research (DOOR). 2016. pp. 261-275.

Поступила 21.10.2016

**Об авторах:**

**Заикин Олег Сергеевич**, научный сотрудник Института динамики систем и теории управления им. В.М. Матросова Сибирского отделения Российской академии наук, старший преподаватель кафедры информационных технологий Иркутского государственного университета, кандидат технических наук, zaikin.icc@gmail.com;

**Чугунов Андрей Александрович**, студент магистратуры факультета информатики, учета и сервиса Байкальского государственного университета, anu8is@yandex.ru.