Учёные записки
Комсомольского-на-Амуре государственного технического университета
№ IV-1 (4)
2010

Трещев И.А., Коньшин А.В.
Treshchev I.A., Konshin A.V.

ИСПОЛЬЗОВАНИЕ КРИПТОПРЕОБРАЗОВАНИЙ НА ОСНОВЕ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ ДЛЯ СИСТЕМ С SMP-АРХИТЕКТУРОЙ В ЗАДАЧАХ ЗАЩИТЫ ИНФОРМАЦИИ

USING CRYPTOTRANSFORMATIONS BASED ON GENETIC ALGORITHMS FOR SYSTEMS WITH THE SMP-ARCHITECTURE FOR INFORMATION SECURITY PROBLEMS

Трещев Иван Андреевич – кандидат технических наук, доцент кафедры «Математическое обеспечение и применение ЭВМ» Комсомольского-на-Амуре государственного технического университета (Россия, Комсомольск-на-Амуре), 681013, Комсомольск-на-Амуре, пр. Ленина, д. 27, тел.: 8-962-287-19-91. E–mail: kalkt@yandex.ru
Mr. Ivan A. Treshchev - Ph.D., Assistant Professor, Department of Computer Software and Computing, Komsomolsk-on-Amur State Technical University, 27, Lenina prospect, 681013 Komsomolsk-on-Amur, Khabarovsky region, Russian Federation, tel.: 8 (962) 2871991, e-mail: kalkt@yandex.ru

Коньшин Алексей Владимирович – студент пятого курса (специальность 010503 «Математическое обеспечение и администрирование информационных систем») Комсомольского-на-Амуре государственного технического университета (Россия, Комсомольск-на-Амуре), 681013, Комсомольск-на-Амуре, пр. Ленина, д. 27, тел.: 8-914-779-66-86. E–mail: alexey@konshin.ru
Mr. Alexey V. Konshin – fifth-year MA student of "Software and administration of information systems", Komsomolsk-on-Amur State Technical University, 27, Lenina prospect, 681013 Komsomolsk-on-Amur, Khabarovsky region, Russian Federation, tel.: 8 (914) 7796686, e-mail: alexey@konshin.ru

Аннотация. Данная работа посвящена рассмотрению возможности применения генетических алгоритмов в задачах защиты информации. Построена математическая модель простого генетического алгоритма с одноточечным кроссинговером, простой мутацией и элитным отбором, описан процесс его функционирования. Авторы предлагают многоэтапную схему шифрования с использованием генетического алгоритма с учетом мутации генов отдельных особей и возможные пути распараллеливания данной схемы для систем с SMP-архитектурой.

Summary: The paper is concerned with the issue of using genetic algorithms for information security problems. Proposed is a mathematical model of a simple genetic algorithm with one-point crossing over and easy mutation with elite pick; the work of the algorithm is described. The authors propose a multistage encryption scheme using the genetic algorithm with mutations in the genes of individuals, and possible ways of parallelizing this scheme for systems with the SMP-architecture.

Ключевые слова: параллельные генетические алгоритмы, защита информации, моделирование, криптография.

Key-words: parallel genetic algorithms, information security, modeling, cryptography

УДК 003.26, 004.056.5

Введение

Генетические алгоритмы (ГА) «базируются на идее эволюции и естественного отбора и функционируют подобно естественным системам» [8, 43]. Главной идей является адапта-

ция начальной популяции индивидов к некоторому окружению таким образом, что выживание и воспроизводство индивидов «основывается на определении полезности и вредности признаков» [1, 87].

Выделяют «три основных типа параллельных ГА» [7, 142]:

1) глобальные однопопуляционные ПГА, модель «master-slave»;
2) однопопуляционные ПГА;
3) многопопуляционные ПГА.

Математическая модель генетического алгоритма, описание процесса его функционирования

Пусть $E_2 = \{0,1\}$.Под операцией $\oplus$ будем понимать суммирование по модулю 2. В дальнейшем будем рассматривать генетические алгоритмы, где гены в наборе хромосом кодируются одним из элементов множества E_2. Простой генетический алгоритм – это пятерка <*S, F, C, M, h*>, где $S \subseteq E_2^h$ — популяция особей, каждый элемент которой представляет из себя набор генов, h – количество хромосом; $F : E_2^h \to R$ – функция жизнеспособности особей, определяющая значение из области положительных действительных чисел, характеризующее степень выживания особи в условиях естественного отбора; $C : S \times S \times \{0,1,...,h-1\} \to E_2^h \times E_2^h$ – функция одноточечного кроссинговера особей, определяющая скрещивание, в результате которого образуются новые особи, замещающие своих родителей и набор хромосом которых складывается из разбиения хромосом родителей взятых попарно, причем функция С действует для «хвостов» наборов хромосом родителей следующим образом:

$$\forall s, v \in S \; s = (x_0, x_1, \ldots, x_{h-1}), v = (y_0, y_1, \ldots, y_{h-1}) \Rightarrow$$

$$C(s, v, l) = ((x_0, x_1, \ldots, y_l, y_{l+1}, \ldots, y_{h-1}), (y_0, y_1, \ldots, x_l, x_{l+1}, \ldots, x_{h-1})).$$

Каждый шаг эволюции можно описать при помощи следующего алгоритма:

1. $i = 0, R = \varnothing$.
2. Выбираются две наиболее жизнеспособные особи:

$T = \varnothing, v_i = \max_{t \in S}(F(t)), T = T \cup \{v_i\}, u_i = \max_{t \in S \setminus T}(F(t)), T = T \cup \{u_i\}$, если максимумов несколько, то выбирается любой.

3. Генерируется случайное $l_i \in \{0,1,...,h-1\}$.
4. Выполняется кроссинговер

$C(v_i, u_i, l_i) = (m, n), S = S \setminus T, R = R \cup \{m, n\}, i = i + 1$, где m,n – новые особи.

5. Шаги 2 – 4 выполняются до тех пор, пока $|S| \geq 2$.
6. Если $|S| = 1$, тогда $R = R \cup S$.
7. Для следующего этапа эволюции принимается $S = R$.

В результате скрещивания в общем случае особи изменяются и для них «необходимо выполнить пересчет функций жизнеспособности» [9, 122].

$M : S \times \{0,1,...,h-1\} \times E_2 \to E_2^h$ – функция одноточечной мутации особей, определяющая видоизменение случайного гена в наборе хромосом случайно взятых особей, действующая следующим образом: $M((x_0, x_1, \ldots, x_q, \ldots x_{h-1}), q, k) = (x_0, x_1, \ldots, x_q \oplus k, \ldots x_{h-1})$, где число $q \in \{0,1,...,h-1\}$ определяет номер гена для мутации, а $k \in E_2$ – будет ли производиться мутация гена с номером *q* для данной особи.

Процесс мутации в генетическом алгоритме можно описать при помощи следующей схемы:

1. $i = 0, R = \varnothing$.

2. Генерируется случайное $k_i \in E_2$ и $q_i \in \{1,2,\ldots,h-1\}$.
3. Выбираем произвольный элемент $s_i \in S$.
4. $R = R \cup M(s_i, q_i, k_i)$.
5. $S = S \setminus \{s_i\}, i = i + 1$.
6. Шаги 2 – 5 выполняются до тех пор пока $|S| > 0$.
7. Для следующего этапа мутации принимается $S = R$.

При разработке программного обеспечения обратимого криптопреобразования с использованием многошаговой схемы на основе генетического алгоритма с учетом мутации авторами была использована модель простого синхронного генетического алгоритма – шестерка <*S, F, C, M, h, t*>, где <*S, F, C, M, h*> - описывает генетический алгоритм, *t* – константа, задающая время, через которое будет происходить процесс кроссинговера. Синхронные генетические алгоритмы позволяют более точно моделировать процесс эволюции на ЭВМ, задавая временные ограничения на процесс кроссинговера и синхронизировать работу вычислительных узлов для проведения расчетов.

О построении параллельных генетических алгоритмов для систем с SMP-архитектурой

Для построения параллельного алгоритма, благодаря использованию модели синхронных генетических алгоритмов, мы предлагаем для систем с SMP-архитектурой поместить начальную популяцию в общей памяти и «использовать идеологию master-slave» [3, 79]. Для выполнения кроссинговера отсортируем популяцию в порядке убывания функции жизнеспособности особей, после чего начальную популяцию «расслоим» на непересекающиеся области.

Представим начальную популяцию – *A* как объединение *n* непересекающихся подпопуляций, где *n* – число вычислительных узлов, тогда

$$A = \bigcup_{i=1}^{n} A_i, A_i \subseteq A, \forall i, j \; i \neq j \Rightarrow A_i \cap A_j = \varnothing.$$

Для равномерной загрузки вычислительных узлов авторами использовалось разбиение с равномощными подпопуляциями. Причем и кроссинговер и мутация в каждой из подпопуляций выполнялись независимо. Если определить ускорение вычислений как отношение времени исполнения на одном вычислительном узле ко времени исполнения на *n* узлах, то рассматриваемый подход позволяет добиться максимального теоретически возможного ускорения.

Каждый порождаемый поток на вычислительных узлах проводит кроссинговер и мутацию для своей подпопуляции.

После одного шага эволюции необходимо вновь отсортировать популяцию в соответствии с новыми значениями функции жизнеспособности. Для синхронизации используется параметр *t* математической модели, который вычисляется как отношение предполагаемого времени выполнения генетического алгоритма на одном вычислительном узле к числу вычислительных узлов. Главный поток начинает опрашивать состояние дочерних только по прошествии времени *t*, это позволяет свести к минимуму затраты на синхронизацию потоков.

Альтернативный подход к распараллеливанию заключается в организации «многостадийного конвейера» [4, 183]. Каждый этап – сортировка, кроссинговер, мутация – выполняется на отдельном вычислительном узле. Каждая последующая стадия конвейера ожидает, когда данные предыдущего этапа будут готовы, после чего начинает функционировать. В данном случае, так же как и для приведенного ранее метода, необходимо «расслоить» популяцию, и выполнять каждую стадию конвейера для отдельной подпопуляции. По мере готовности подпопуляция «продвигается по конвейеру» а на текущую стадию поступает новая часть популяции. Отметим, что в отличие от рассмотренного выше метода применение данной схемы будет требовать гораздо больше затрат на синхронизацию потоков и является ме-

нее масштабируемой, хотя сочетает в себе как функциональный, так и параллелизм по данным. В данном случае параметр *t* – время, через которое очередная стадия начинает опрашивать предыдущую, что также снижает затраты на синхронизацию потоков.

Схема шифрования на основе генетического алгоритма и сравнение с аналогами

В [5, 31] рассмотрено шифрование на основе генетического алгоритма, выполняемое за один раунд эволюции, без мутации. Мы предлагаем модифицированную многоэтапную схему шифрования, где эволюция проводится в несколько этапов с использованием генетического алгоритма с учетом мутации генов отдельных особей.

«Псевдослучайная последовательность для шифра генерируется с использованием NLFFSR» [6, 452] (Non Linear Feed Forward Shift Register – нелинейный «выталкивающий» сдвиговый регистр с обратной связью). Общая схема получения бита псевдослучайной последовательности представлена на рис. 1, где суммирование ведется по модулю 2. Нелинейный сдвиговый регистр представляет собой линейный сдвиговый регистр – LFFSR (Linear Feed Forward Shift Register – линейный «выталкивающий» сдвиговый регистр с обратной связью), над каждой *n*–кой выходных бит которого выполняется заданное нелинейное преобразование, где *n* – разрядность сдвигового регистра. Причем выделяют два вида NLFFSR: первый – когда нелинейное преобразование выполняется над результирующими битами LFFSR, второй – когда нелинейное преобразование выполняется внутри LFFSR. Авторами в работе исследовались генетические алгоритмы на основе первого типа.

Описание принципов функционирования LFFSR можно найти, например в [2, 49]. Свойства последовательности выдаваемой LFFSR тесно связаны с характеристическим неприводимым полиномом степени *n* над полем F_2 – $C(c_1, c_2, \ldots, c_n, x) = 1 + \sum_{i=1}^{n} c_i x^i$.

Нами был использован восьмиразрядный линейный сдвиговый регистр с обратной связью. В качестве характеристического неприводимого полинома выбран $f(x) = x^8 + x^4 + x^3 + x^2 + 1$, то есть *c8*, *c4*, *c3* и *c2* равны единице и нелинейное преобразование

$$g(a_{n-1}, a_{n-2}, a_{n-3}, a_{n-4}, a_{n-5}, a_{n-6}, a_{n-7}, a_{n-8}) = \left[(a_{n-1} \wedge a_{n-3}) \oplus (a_{n-2} \wedge \bar{a}_{n-4})\right] \oplus$$
$$\oplus \left[(a_{n-5} \wedge a_{n-7}) \oplus (a_{n-6} \wedge \bar{a}_{n-8})\right],$$

где сложение выполняется по модулю 2.

Инициализирующее значение для сдвигового регистра задавалось 11111111, что позволяет получить последовательность бит периода 255 или 32 десятичных чисел.

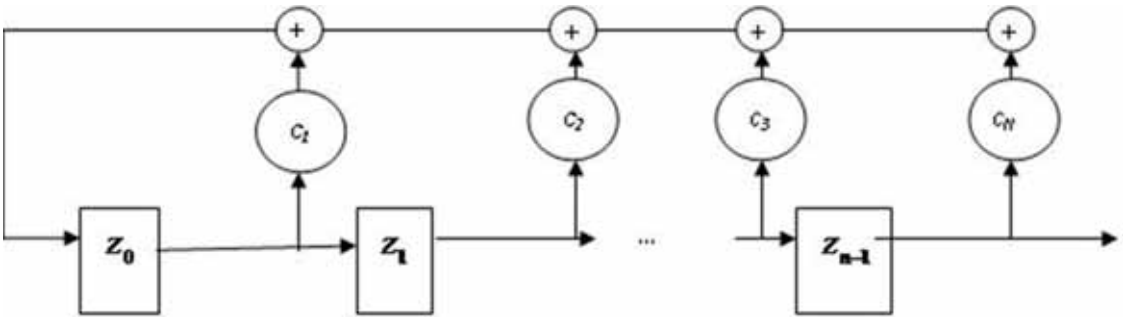

Рис. 1. Линейный сдвиговый регистр

После каждого шага значение $z_0 = \sum_{i=0}^{n-1} z_i \wedge c_{i+1}$, где суммирование ведется по модулю 2, и каждое z_{n-1} формирует бит выходной последовательности.

Последовательность бит, генерируемая LFFSR:

1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1 …

Всего сгенерировано 1024 бита, причем последний бит совпадает с первым. Далее последовательность, генерируемая LFFSR, повторяется.

Последовательность бит, генерируемая NLFFSR, на основе последовательности LFFSR:

0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0

Всего 128 бит. Ключ для шифрования и дешифрования формируется из групп по 8 восьмиразрядных двоичных чисел получаемых на выходе NLFFSR

Пусть мы получили следующую часть последовательности:

$$G = \begin{pmatrix} g_{1,1} & g_{1,2} & g_{1,3} & g_{1,4} & g_{1,5} & g_{1,6} & g_{1,7} & g_{1,8} \\ g_{2,1} & g_{2,2} & g_{2,3} & g_{2,4} & g_{2,5} & g_{2,6} & g_{2,7} & g_{2,8} \\ g_{3,1} & g_{3,2} & g_{3,3} & g_{3,4} & g_{3,5} & g_{3,6} & g_{3,7} & g_{3,8} \\ g_{4,1} & g_{4,2} & g_{4,3} & g_{4,4} & g_{4,5} & g_{4,6} & g_{4,7} & g_{4,8} \\ g_{5,1} & g_{5,2} & g_{5,3} & g_{5,4} & g_{5,5} & g_{5,6} & g_{5,7} & g_{5,8} \\ g_{6,1} & g_{6,2} & g_{6,3} & g_{6,4} & g_{6,5} & g_{6,6} & g_{6,7} & g_{6,8} \\ g_{7,1} & g_{7,2} & g_{7,3} & g_{7,4} & g_{7,5} & g_{7,6} & g_{7,7} & g_{7,8} \\ g_{8,1} & g_{8,2} & g_{8,3} & g_{8,4} & g_{8,5} & g_{8,6} & g_{8,7} & g_{8,8} \end{pmatrix}$$

Транспонируем данную матрицу, после чего для шифрования и дешифрования будем использовать строки полученной G^T, первый байт ключа – $G^T_{<1>} = (g_{1,8}, g_{1,7}, g_{1,6}, g_{1,5}, g_{1,4}, g_{1,3}, g_{1,2}, g_{,1,1})$, где $g_{i,j} \in E_2, i = \overline{1..8}, j = \overline{1..8}.$.

Представим набор входных данных, нуждающихся в криптопреобразовании, в виде ленты *D*. Из *D* берутся блоки по четыре байта и преобразуются в популяцию, каждая особь которой будет являться словом входных данных, а её хромосомы его битами. Опишем алгоритм шифрования, используя для обозначения операции вычисления остатка от деления – mod, а для выполнения операции поразрядного исключающего или – xor и div для выполнения целочисленного деления.

1. Генерируем псевдослучайную двоичную последовательность. Выделяем из нее блоки по 8 бит – A_j, как это описано выше.
2. $A_j = \mathrm{mod}(A_j, 256)$.
3. $i = 0$ – счетчик слов на входной ленте.
4. $j = 0$ – счетчик байт в последовательности А.
5. Выберем из входного потока 2 слова по 16 бит – D_i, D_{i+1}.
6. $k = 0$ – счетчик этапов эволюции.
7. Производим кроссинговер слов D_i, D_{i+1} используя в качестве точки разбиения хромосом позицию $(A_{k*j} \bmod 16)$, то есть выполняем $C(D_i, D_{i+1}, A_{k*j} \bmod 16)$.

8. Производим мутацию слова D_i если $A_{k*j+1} > 127$, инвертируя (A_{k*j+2} mod16) бит слова, то есть выполняем $M(D_i, A_{k*j+2} \bmod 16, A_{k*j+1} \; div \; 128)$.

9. Производим мутацию слова D_{i+1} если $A_{k*j+3} > 127$, инвертируя (A_{k*j+4} mod16) бит слова, то есть выполняем $M(D_i, A_{k*j+4} \bmod 16, A_{k*j+3} \; div \; 128)$.

10. $k = k + 1$.

11. Повторяем шаги 7–10, пока $k < M$, где M – требуемое количество этапов эволюции.

12. $j = j + 5 * M$.

13. Выберем A_j и A_{j+1}.

14. Пусть $Y_i = A_j \; xor \; (A_j << 8)$, $Y_{i+1} = A_{j+1} \; xor \; (A_{j+1} << 8)$.

15. Получаем два зашифрованных слова $E_i = D_i \; xor \; Y_i$, $E_{i+1} = D_{i+1} \; xor \; Y_{i+1}$.

16. $j = j + 2$.

17. $i = i + 2$.

18. Повторяем шаги 5 – 12, пока не закончатся слова на входной ленте.

Для обратного криптопреобразования необходимо выполнить все шаги описанного алгоритма шифрования в обратном порядке.

Результаты тестирования

Для тестирования использовались различные потоки входных данных (изображения). Результаты выполнения криптопреобразования над потоком входных данных и обратного преобразования на основе генетического алгоритма с одноточечным кроссинговером и простой мутацией представлен на рис. 2. Как видно, результаты обратимого криптопреобразования достаточно сильно влияют на исходные данные и затрудняют анализ исходных данных.

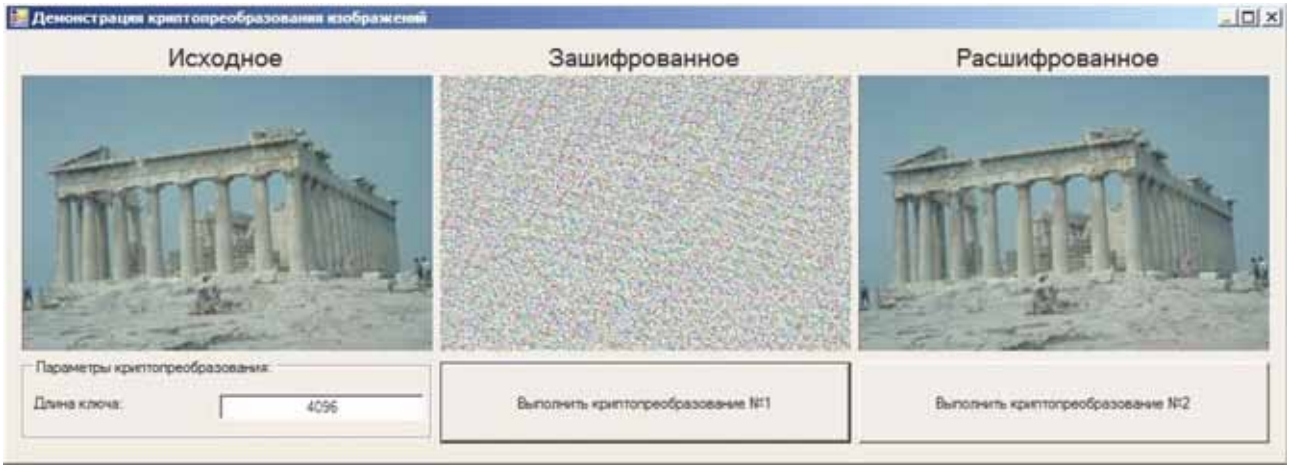

Рис 2. Пример криптопреобразования исходного сообщения

На рис. 3 представлены графики зависимости времени исполнения параллельного и последовательного алгоритма на основе расслоения и организации конвейера для различных изображений (рассматривается время на выполнение криптопреобразования исходных данных и обратное преобразование).

Тестирование разработанных приложений производилось на HP Workstation xw8200 под управлением Microsoft Windows 2000, оснащенных двумя микропроцессорами Intel Pentium 4 Xeon 3,6Ghz (1Mb L1 cache), с поддержкой технологии Hyper Threading, объем оперативной памяти 4Gb.

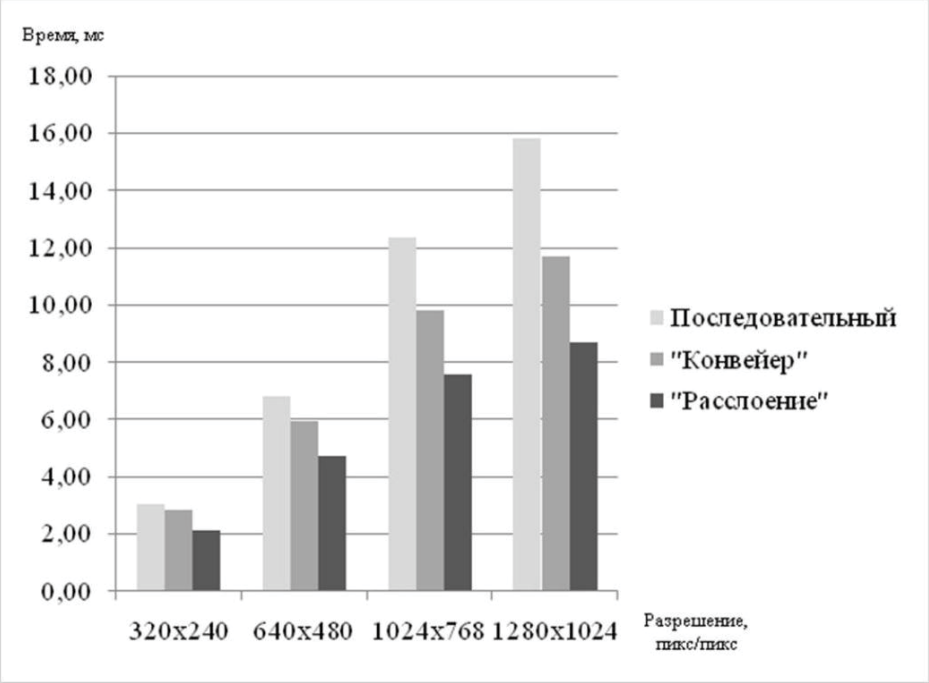

Рис. 3. Диаграмма зависимости времени выполнения прямого и обратного криптопреобразования для различных изображений.

Заключение

В работе рассмотрен один из подходов к построению математической модели простого генетического алгоритма с одноточечным кроссинговером и простой мутацией, описан процесс его функционирования. На основе рассматриваемой математической модели предложены пути распараллеливания генетических алгоритмов для систем с SMP-архитектурой. Построена схема для обратимого криптопреобразования исходных данных на основе нелинейного сдвигового регистра с обратной связью.

ЛИТЕРАТУРА

1. Курейчик, В. М. Генетические алгоритмы : моногр. / В. М. Курейчик. – Таганрог : Изд-во ТРТУ, 1998. – 242 с.
2. Мартынов, А. И. Методы и задачи криптографической защиты информации /А. И. Мартынов : учеб. пособие – Ульяновск : УлГТУ, 2007. – 92 с.
3. Топорков, В. В. Модели распределенных вычислений / В. В. Топорков. – М. : ФИЗМАТЛИТ, 2004. – 320 с.
4. Трещёв, И. А. Программное обеспечение для перебора последовательностей на компьютерах с SMP-архитектурой / И.А. Трещев // XXXI Дальневосточная школа-семинар имени академика Е.В. Золотова. – Владивосток : Дальнаука, 2006. – С. 183.
5. Alba E., Troya J.M.A Survey of Parallel Distributed Genetic Algorithms/ E. Alba, J.M. Troya // Complexity. – 1999. – Vol.4 – P.31–52.
6. Alba E., Troya J.M.Analyzing Synchronous and Asynchronous Parallel Distributed Genetic Algorithms / E. Alba, J.M. Troya // Future Generation Computer Systems. – 2001. – Vol.17. – p. 451-465.
7. Cantu-Paz E. A.Survey of Parallel Genetic Algorithms / E. A. Cantu-Paz // Reseaux et Systems Repartis. Paris: Hermes. – Vol. 10. – 1997. – 141–171 p.
8. Goldberg D. E., Genetic Algorithms in Search, Optimization, and Machine Learning / D. E. Goldberg// Boston: Addison-Wesley. – 1989. – 372 p.
9. Michalewicz Z. Genetic Algorithms + Data Structures = Evolution Programs / Z. Michalewicz // Berlin: Springer Verlag. – 1996. – 387 p.