ISSN 1992-6502 (Print)
2016. Т. 20, № 1 (71). С. 161–167
Вестник УГАТУ
ISSN 2225-2789 (Online)
http://journal.ugatu.ac.ru

УДК 004.65

ОСОБЕННОСТИ ЧИСЛЕННОЙ РЕАЛИЗАЦИИ АЛГОРИТМОВ ПОТОЧНОГО И БЛОЧНОГО ШИФРОВАНИЯ

Т. Р. ЗМЫЗГОВА[1], **А. К. МАРКОВ**[2]

[1] tanja21.zm@gmail.com, [2] informatika@kgsu.ru

ФГБОУ ВО «Курганский государственный университет» (КГУ)

Поступила в редакцию 15.01.2016

Аннотация. Статья посвящена вопросам обеспечения безопасности информации. Кратко рассмотрены особенности численной реализации некоторых алгоритмов поточного и блочного шифрования. Проведен сравнительный анализ, сформулированы особенности использования генераторов псевдослучайных данных.

Ключевые слова: информационная безопасность, шифрование, криптография.

ВВЕДЕНИЕ

ПОТОЧНЫЕ И БЛОЧНЫЕ АЛГОРИТМЫ ШИФРОВАНИЯ

Интенсивное развитие информационных технологий неизбежно приводит к задаче обеспечения конфиденциальности и целостности информации. Технологии виртуальной инфраструктуры неизбежно внедряются в современную жизнь. Эксплуатация информационных систем в контексте обработки данных становится заурядным явлением, удобным, привычным и экономически оправданным. Комплексное обеспечение информационной безопасности представляет собой непрерывный процесс, который постоянно требует модификации и, как результат, постоянного усложнения ИТ-систем. Инфраструктура ИТ-систем подразумевает внедрение технических или программных компонентов, наличие которых приводит к увеличению возможностей несанкционированного доступа к обрабатываемой информации [1].

Одним из эффективных способов решения проблемы защиты информации является шифрование. Проектирование алгоритмов шифрования при передаче данных в режиме реального времени основывается на некотором ограниченном выборе функций, преобразующих открытый текст в зашифрованные сообщения. Следует отметить, что шифрование требовательно к вычислительным ресурсам системы и, как весьма трудоемкий процесс, может привести к значительному снижению скорости передачи данных.

Различают поточные и блочные шифры. Блочный шифр представляет собой разновидность симметричного шифра, оперирующего группами бит фиксированной длины (блоками), характерный размер которых меняется в пределах 64–256 бит, при этом если исходный текст (или его остаток) меньше размера блока, перед шифрованием его дополняют [2]. Поточный шифр – это симметричный шифр, в котором каждый символ открытого текста преобразуется в символ шифрованного текста в зависимости не только от используемого ключа, но и от его расположения в потоке открытого текста.

Для повышения производительности обработки информации целесообразно использовать именно поточные шифры, поскольку поточное шифрование реализует преобразование данных в реальном режиме времени с высокой скоростью, соизмеримой со скоростью поступления входной информации, вне зависимости от объема и разрядности потока преобразуемых данных. Именно этим объясняется актуальность применения методов поточного шифрования в различных открытых системах и приложениях, особенно в условиях значительного увеличения объемов трафика передачи данных [3].

СРАВНЕНИЕ АЛГОРИТМОВ

1. ***Сложность передачи ключа***. Поскольку ключ из 8–32 байт передать проще, чем гамму, длина которой равна длине самого сообщения,

блочное шифрование нередко предпочитают поточному. Однако если при реализации поточного алгоритма осуществлять передачу генератора псевдослучайных чисел (ГПЧ), данная проблема становится некритичной.

2. ***Разница алгоритмов шифрования и дешифрования***. В данном случае алгоритмы идентичны, поскольку для обоих шифрование и дешифрование производится практически одними и теми же способами.

3. ***Сложность реализации***. Алгоритм поточного шифрования реализовать проще в силу того, что в блочном алгоритме необходимо сначала делить исходное сообщение на блоки и добавлять данные при несовпадении размерностей. Тем не менее, в случае поточного шифра надо учитывать реализацию ГПЧ. Существует вариант, что генератор будет настолько сложен, что поточное шифрование получится сложнее.

4. ***Вероятность искажения текста***. В поточном шифровании отсутствует эффект размножения ошибок, как следствие, число ошибок в расшифрованной последовательности равно числу ошибок в зашифрованной. Один неверный символ в гамме равен одной ошибке в тексте, один неверный символ в ключе блочного шифрования означает одну ошибку в каждом блоке текста.

5. ***Производительность***. Если реализовывать вычислительный процесс в цифровой микросхеме, то поточный алгоритм будет производительнее. В программной реализации существуют два варианта: с нагрузкой на оперативную память и без нее. Суть варианта с нагрузкой заключается в том, что вся шифруемая последовательность хранится непосредственно в оперативной памяти, что нерационально при большом размере файла. В этом случае опять же выигрывает поточный алгоритм. Далее в статье будет рассмотрен вариант, при котором в оперативной памяти хранится 1–32 байта из последовательности, а остальные данные остаются в физической памяти. При этом считать сразу 32 байта из физической памяти и зашифровать их будет быстрее, чем считать один байт и зашифровать его. В этом случае блочный алгоритм более производителен.

МОДИФИКАЦИЯ АЛГОРИТМА: НАЛОЖЕНИЕ ПСЕВДОСЛУЧАЙНОГО ЧИСЛА СЛОЕВ ГАММЫ НА ИСХОДНЫЙ БАЙТ

Предлагаемая идея модификации алгоритма заключается в наложении псевдослучайного числа слоев гаммы (ключей) на исходный байт (блок). После прохода одного байта (блока) в генератор отправляется зашифрованный байт, в этом случае к ключу и зашифрованному блоку применятся операция XOR. В результате период генератора (длина ключа) становится не важным, поскольку генератор меняет каждый байт, при этом ключ видоизменяется для каждого блока, причем сложность вывода формулы для дешифрования значительно возрастает.

Недостаток метода заключается в объеме гаммы (количестве ключей), однако если передавать только генератор, а не саму гамму (ключи), подобная проблема не будет актуальной. Заметим, что производительность в сравнении с базовыми алгоритмами шифрования уменьшается в 4 раза (примеры сравнения производительностей алгоритмов будут приведены далее).

Число уровней для каждого байта генерируется по формуле

$$f(key) = (abs(10000\cos(generate(key))) + 1) \bmod N .$$

Для оптимизации производительности алгоритма значение параметра N устанавливается экспериментально и зависит от размера файла (табл. 1).

Таблица 1

Зависимость уровней от размера файла

Размер файла (байт)	N
0–512000	999
512000–10485760	333
более 10485760	33

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМОВ

Для программной реализации алгоритмов использован язык программирования C++, при этом для генерации гаммы и ключа использован ГПЧ rand. Функция srand выполняет инициализацию генератора случайных чисел rand. Генератор псевдослучайных чисел инициализируется с помощью аргумента seed, который играет роль зерна. Прототип функции:

void srand (unsigned int seed).

Зерно seed в данном случае является ключом шифрования.

Для наложения гаммы или ключа на исходную последовательность применяется операция XOR, которая выполняет логическое исключение с двумя выражениями типа Boolean или побитовое исключение с двумя числовыми выражениями. Для реализации потоков шифрования используется библиотека Process.h. Для доступа к файлам в физической памяти используется библиотека Fstream.h. Построение пользовательского интерфейса было осуществлено на базе платформы Microsoft. Net Framework.

Далее подробно изложено описание методов шифрования. Поскольку реализация алгоритмов дешифрования выглядит аналогично, их описание представляется излишним.

Условные обозначения: $A_1, A_2, ..., A_n$ – исходный текст, $G_1, G_2, ..., G_n$ – гамма, K – ключ, $B_1, B_2, ..., B_n$ – зашифрованные данные.

1. Шифрование гаммированием

1) пользователь вводит ключ *K*;

2) передаем ключ в ГПЧ: $K \rightarrow seed$;

3) считываем первый байт исходных данных из физической памяти в оперативную: физическая память $\rightarrow A_1$;

4) ГПЧ на основе ключа генерирует гамму для первого байта: $rand \rightarrow G_1$;

5) накладываем гамму на первый байт исходных данных:

$$B_1 = A_1 \oplus G_1;$$

6) записываем шифрованный байт в физическую память:

$$B_1 \rightarrow \text{физическая память};$$

7) считываем следующий байт данных из физической памяти в оперативную: физическая память $\rightarrow A_2$;

8) ГПЧ на основе ключа генерирует гамму для следующего байта:

$$rand \rightarrow G_2;$$

9) накладываем гамму на следующий байт данных: $B_2 = A_2 \oplus G_2$.

10) записываем шифрованный байт в физическую память:

$$B_2 \rightarrow \text{физическая память};$$

11) п. 7-10 повторяются до окончания последовательности данных.

2. Шифрование гаммированием с обратной связью

При реализации этого алгоритма повторяются п. 1-8 предыдущего способа шифрования. Принципиальное отличие касается п. 9: накладываем гамму и предыдущий шифрованный байт на следующий байт исходных данных:

$$B_2 = A_2 \oplus B_2 \oplus G_2.$$

Далее п. 9-11 повторяются.

3. Шифрование многоуровневым гаммированием

1) пользователь вводит ключ *K*;

2) передаем ключ в ГПЧ: $K \rightarrow seed$;

3) на основе ключа генерируем число слоев для первого байта:

$$lvlgen(K) \rightarrow lvl_1;$$

4) считываем первый байт исходных данных из

физической памяти в оперативную:

$$\text{физическая память} \rightarrow A_1;$$

5) ГПЧ на основе ключа генерирует слои гаммы для первого байта:

$$rand \rightarrow \{G_1, G_2, \dots, G_{lvl_1}\};$$

6) накладываем слои гаммы на первый байт исходных данных:

$$B_1 = \{A_1 = A_1 \oplus G_1, \dots, A_1 = A_1 \oplus G_{lvl_1}\};$$

7) записываем шифрованный байт в физическую память:

$$B_1 \rightarrow \text{физическая память};$$

8) передаем предыдущий шифрованный байт в ГПЧ: $B_1 \rightarrow seed$;

9) генерируем число слоев для следующего байта на основе предыдущего значения:

$$lvlgen(lvl_1) \rightarrow lvl_2;$$

10) считываем следующий байт исходных данных из физической памяти в оперативную:

$$\text{физическая память} \rightarrow A_2;$$

11) ГПЧ на основе предыдущего шифрованного байта генерирует слои гаммы для следующего байта: $rand \rightarrow \{G_1, \dots, G_{lvl_2}\}$;

12) накладываем слои гаммы на следующий байт исходных данных:

$$\{A_2 = A_2 \oplus G_1, \dots, A_2 = A_2 \oplus G_{lvl_2}\} \rightarrow B_2;$$

13) записываем шифрованный байт в физическую память:

$$B_2 \rightarrow \text{физическая память};$$

14) п. 8-13 повторяются до окончания последовательности данных.

Замечание. В связи с тем, что дополнение последнего блока исходных данных до длины ключа шифрования в некоторых случаях может выдать значение этого параметра, в данной реализации алгоритма указанная процедура исключена.

4. Шифрование ECB (*Electronic code book*)

1) пользователь вводит ключ *K*;

2) передаем ключ в ГПЧ: $K \rightarrow seed$;

3) генерируем длину блока данных на основе ключа: $rand \rightarrow$ KeyLength;

4) ГПЧ генерирует ключ шифрования:

$$rand \rightarrow \{K_1, \dots, K_{\text{KeyLength}}\};$$

5) считываем первый блок исходных данных из физической памяти в оперативную:

$$\text{физическая память} \rightarrow \{A_1, \dots, A_{\text{KeyLength}}\};$$

6) накладываем ключ шифрования на первый блок исходных данных:

$$\{B_1, \dots, B_{\text{KeyLength}}\} = \{A_1, \dots, A_{\text{KeyLength}}\} \oplus \{K_1, \dots, K_{\text{KeyLength}}\};$$

7) записываем шифрованный блок в физическую память:

$\{B_1, \dots, B_{\text{KeyLength}}\} \rightarrow$ физическая память;

8) считываем следующий блок исходных данных из физической памяти в оперативную:

физическая память $\rightarrow$

$\{A_{\text{KeyLength}+1}, \dots, A_{\text{KeyLength}+\text{KeyLength}}\}$;

9) накладываем ключ шифрования на следующий блок исходных данных.

$$\{B_{\text{KeyLength}+1}, \dots, B_{\text{KeyLength}+\text{KeyLength}}\} = \{A_{\text{KeyLength}+1}, \dots, A_{\text{KeyLength}+\text{KeyLength}}\} \oplus \{K_1, K_2, \dots, K_{\text{KeyLength}}\};$$

10) записываем шифрованный блок в физическую память:

$\{B_{\text{KeyLength}+1}, \dots, B_{\text{KeyLength}+\text{KeyLength}}\} \rightarrow$ физическая память;

11) п. 8-10 повторяются до окончания последовательности данных.

Если последний блок данных меньше длины ключа шифрования, ключ урезается до его длины.

5. Шифрование ECB с обратной связью

При реализации этого способа повторяются п. 1-11 предыдущего алгоритма ECB. Принципиальное отличие касается только п. 9: накладываем ключ шифрования и предыдущий зашифрованный блок на следующий блок исходных данных:

$$\{B_{\text{KeyLength}+1}, \dots, B_{\text{KeyLength}+\text{KeyLength}}\} = \{A_{\text{KeyLength}+1}, \dots, A_{\text{KeyLength}+\text{KeyLength}}\} \oplus \{B_1, \dots, B_{\text{KeyLength}}\} \oplus \{K_1, \dots, K_{\text{KeyLength}}\}.$$

6. Шифрование многоуровневым ECB

1) пользователь вводит ключ K;

2) передаем ключ в генератор псевдослучайных чисел: $K \rightarrow seed$;

3) генерируем длину блока данных на основе ключа: $rand \rightarrow$ KeyLength;

4) считываем первый блок исходных данных из физической памяти в оперативную:

физическая память $\rightarrow$

$\{A\_1, A\_2, \dots, A\_\text{KeyLength}\}$;

5) на основе ключа генерируем число ключей шифрования для первого блока:

$$lvlgen(K) \rightarrow lvl_1;$$

6) генерируем ключ шифрования с помощью ГПЧ:

$$rand \rightarrow \{K_1, \dots, K_{\text{KeyLength}}\};$$

7) накладываем ключ на блок данных:

$$\{A_1, \dots, A_{\text{KeyLength}}\} = \{A_1, \dots, A_{\text{KeyLength}}\} \oplus \{K_1, \dots, K_{\text{KeyLength}}\};$$

8) п. 6-7 повторяются до тех пор, пока не будет наложено lvl_1 ключей шифрования.

9) результат работы п. 6–8 записываем в новый блок:

$$\{B_1, \dots, B_{\text{KeyLength}}\} = \{A_1, \dots, A_{\text{KeyLength}}\};$$

10) записываем шифрованный блок в физическую память:

$\{B_1, B_2, \dots, B_{\text{KeyLength}}\} \rightarrow$ физическая память;

11) считываем следующий блок исходных данных из физической памяти в оперативную:

физическая память $\rightarrow$

$\{A_{\text{KeyLength}+1}, \dots, A_{\text{KeyLength}+\text{KeyLength}}\}$;

12) передаем в ГПЧ значение ключа, применив к нему и результату работы ГПЧ после генерации последнего ключа операцию XOR:

$$K \oplus rand \rightarrow seed;$$

13) генерируем число ключей шифрования для следующего блока данных на основе предыдущего числа слоев: $lvlgen(lvl_1) \rightarrow lvl_2$;

14) генерируем ключ шифрования:

$$rand \rightarrow \{K_1, K_2, \dots, K_{\text{KeyLength}}\};$$

15) накладываем ключ на блок данных:

$$\{A_1, \dots, A_{\text{KeyLength}}\} = \{A_1, \dots, A_{\text{KeyLength}}\} \oplus \{K_1, \dots, K_{\text{KeyLength}}\};$$

16) п. 14-15 повторяем, пока не будет наложено lvl_2 ключей шифрования;

17) результат работы пунктов 14-16 записываем в новый блок:

$$\{B_1, B_2, \dots, B_{\text{KeyLength}}\} = \{A_1, A_2, \dots, A_{\text{KeyLength}}\};$$

18) записываем зашифрованный блок в физическую память:

$\{B_1, B_2, \dots, B_{\text{KeyLength}}\} \rightarrow$ физическая память;

19) п. 11-18 повторяются до окончания последовательности данных.

Если последний блок данных меньше длины ключей, ключи шифрования урезаются.

РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТАЛЬНОГО ИССЛЕДОВАНИЯ

В разработанном программном комплексе реализованы все инструменты для проведения эксперимента по сравнению описанных выше алгоритмов шифрования, остается лишь последовательно зашифровать файлы разной размерности и, проанализировав графики, таблицы и скорость работы алгоритмов, сделать соответствующий вывод о влиянии сложности алгоритма на его производительность. Далее, принимая во внимание, например, криптостойкость алгоритмов, остается решить, какой алгоритм будет оптимальным для конкретной ситуации.

На рис. 1, 2 в качестве примера приведены результаты численной реализации некоторых алгоритмов шифрования.

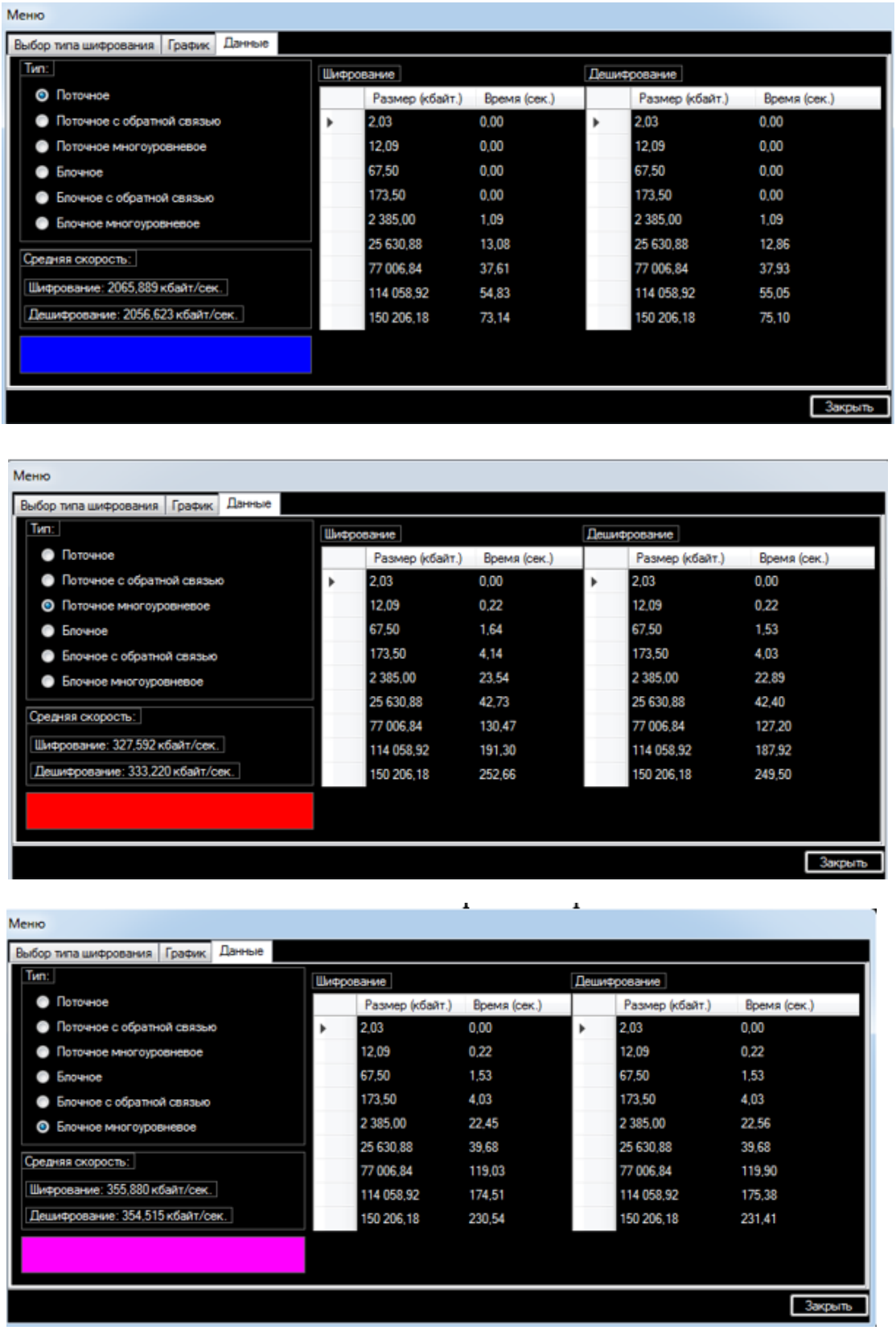

Рис. 1. Численные характеристики реализации алгоритмов поточного, многоуровневого поточного и блочного шифрования и расшифрования данных

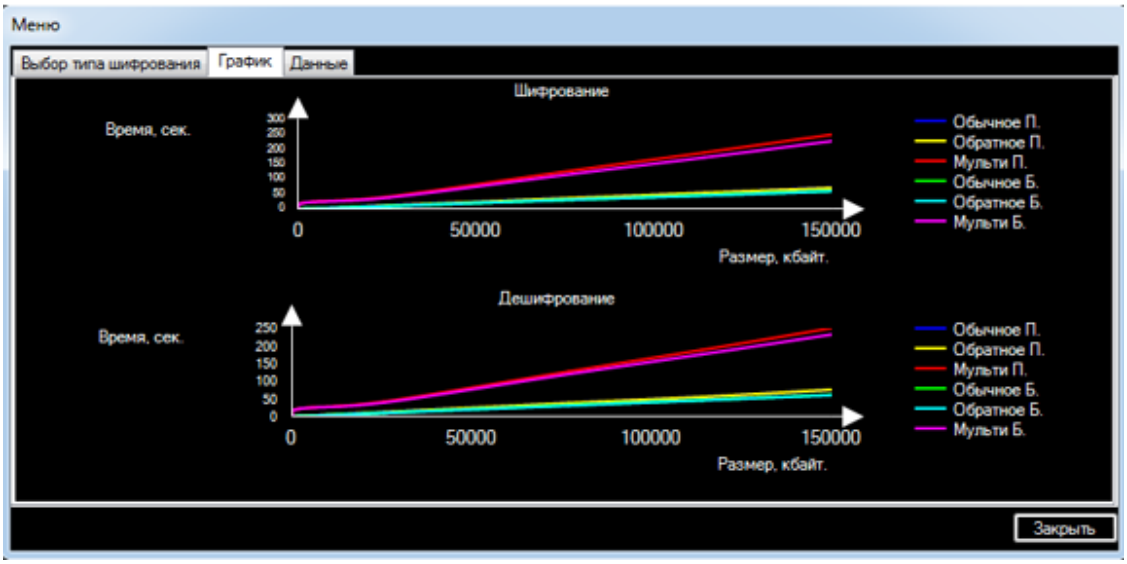

Рис. 2. Показатели временных затрат алгоритмов

Результаты экспериментального тестирования, приведенные на рис.1, демонстрируют скорость и время шифрования и расшифрования для некоторых алгоритмов относительно данных разной размерности (от 2,03 до 150206,18 Кбайт).

В частности, при численной реализации поточного многоуровневого алгоритма шифрования средняя скорость шифрования данных составила 327,592 Кбайт/с, а средняя скорость расшифрования – 333,220 Кбайт/с. Соответствующие показатели для блочного многоуровневого алгоритма
составили 355,880 Кбайт/с и 354,515 Кбайт/с. Заметим, что при этом средняя скорость работы обычного поточного алгоритма значительно выше: 2065,899 Кбайт/с и 2056,623 Кбайт/с соответственно.

На рис. 2 приведены графики, иллюстрирующие временные затраты каждого из алгоритмов при шифровании и расшифровании данных, выделенные для наглядности различным цветом в зависимости от выбранного алгоритма. Очевидно, что в случае объема шифрования данных порядка 30000 Кбайт и выше наиболее эффективными оказываются обычные версии блочного и поточного алгоритмов шифрования.

Общие показатели тестирования для определения временных затрат при шифровании приведены в табл. 2, 3.

Таблица 2

Временные затраты при шифровании данных, секунды

Объем данных (Кбайт)	Поточный алгоритм	Поточный алгоритм с обратной связью	Многоуровневый поточный алгоритм	Блочный алгоритм	Блочный алгоритм с обратной связью	Многоуровневый блочный алгоритм
2.03	0.00	0.00	0.00	0.00	0.00	0.00
12.09	0.00	0.00	0.22	0.00	0.00	0.22
67.50	0.00	0.00	1.64	0.00	0.00	1.53
173.50	0.00	0.00	4.14	0.00	0.00	4.03
2385.00	1.09	1.20	23.54	0.87	0.87	22.45
25630.88	13.08	12.32	42.73	10.36	10.25	39.68
77006.84	37.61	37.61	130.47	31.83	31.94	119.03
114058.92	54.83	55.05	191.30	47.42	45.78	174.51
150206.18	73.14	71.83	252.66	61.48	60.71	230.54

Отметим, что при расшифровании такого же объема данных значения, приведенные в табл. 2, практически не отличаются.

Таблица 3

Средняя скорость обработки данных

Алгоритм	Средняя скорость (Кбайт/с)	
	Шифрование данных	Расшифрование данных
Поточный алгоритм	2065,889	2056.623
Поточный алгоритм с обратной связью	2056.215	2015.509
Многоуровневый поточный алгоритм	327.592	333.220
Блочный алгоритм	2495.729	2491.320
Блочный алгоритм с обратной связью	2522.668	2552.220
Многоуровневый блочный алгоритм	355.880	354.515

ЗАКЛЮЧЕНИЕ

Анализ экспериментальных данных производительности алгоритмов позволяет сделать вывод о том, что у всех методов время обработки данных растет прямо пропорционально их объему. Скорость шифрования и расшифрования блочных алгоритмов немного превосходит скорость работы поточных. Алгоритмы с обратной связью, в свою очередь, незначительно превосходят свои стандартные версии, несмотря на то, что в версии алгоритма с обратной связью совершается большее количество преобразований и проверяется больше условий. Многоуровневые версии алгоритмов в среднем отстают от остальных в 3-4 раза, что обусловлено значительно большим количеством итераций над одним байтом (блоком).

В целом, нельзя сделать однозначный вывод о предпочтении какого-либо определенного алгоритма, стандартные реализации методов шифрования дают более низкую криптостойкость. В то же время, реализация алгоритмов с обратной связью при повреждении одной части текста не позволяет расшифровать оставшийся текст, а многоуровневый алгоритм оказывается достаточно слабым по своей производительности. Таким образом, перечисленные факторы не препятствуют широкому применению указанных алгоритмов шифрования. Каждый потенциальный пользователь криптографии имеет возможность выбора между системами шифрования, ограничив свое предпочтение такими решающими факторами, как функции криптографической защиты, тип шифруемых данных, характеристики канала связи и т.п., делая необходимый выбор между производительностью или криптостойкостью.

СПИСОК ЛИТЕРАТУРЫ

1. **Фергюсон Н.**, **Шнайер Б.** Практическая криптография. М.: Изд-во «Вильямс», 2005. 424 с. **[** N. Ferguson and B. Schneier. Practical Cryptography, (in Russian). Moscow: Publishing house «Williams», 2005. **]**

2. **Панасенко С**. Алгоритмы шифрования. Специальный справочник. СПб: БХВ-Петербург, 2009. 576 с. **[** S. Panasenko. Encryption algorithm. Special reference, (in Russian). St. Petersburg: BHV-Petersburg, 2009. **]**

3. **Рябко Б. Я.**, **Фионов А. Н**. Криптографические методы защиты информации: Учебное пособие для вузов. М.: Горячая линия-Телеком, 2005. 229 с. **[** B. Ya. Ryabko and A. N. Fionov. Cryptographic methods of information protection: Textbook for universities, (in Russian). Moscow: Hot line Telecom, 2005. **]**

ОБ АВТОРАХ

ЗМЫЗГОВА Татьяна Рудольфовна, зав. каф. информатики. Дипл. механика, прикладная математика (НГУ, 1991). Канд. техн. наук по матем. модел., числ. методам и комплексам программ (ТюмГУ, 2007). Иссл. в обл. компьютерного моделирования, обработки цифровых изображений.

МАРКОВ Антон Константинович, студ. 3-го курса по направлению Информационная безопасность автоматизированных систем. Иссл. в обл. технологий программирования, безопасности информационных систем.

METADATA

Title: The specifics of numerical implementation of algorithms in stream and block encryption.

Authors: T. R. Zmyzgova[1], A. A. Markov[2]

Affiliation: Kurgan State University (KGU), Russia.

Email: [1]tanja21.zm@gmail.com, [2] informatika@kgsu.ru.

Language: Russian.

Source: Vestnik UGATU (scientific journal of Ufa State Aviation Technical University), vol. 20, no. 1 (71), pp. 162–167, 2016. ISSN 2225-2789 (Online), ISSN 1992-6502 (Print).

Abstract: The article deals with the problem of ensuring information security. The author considers special features of numerical implementation in connection with various stream and block encryption algorithms and conducts a comparative analysis. Moreover, the author points out the peculiarities of using pseudorandom number generators.

Key words: information security, encryption, cryptography.

About authors:

ZMYZGOVA, Tatyana Rudolfovna, head. DEP. "Computer science". Dipl. mechanics, applied mathematics (NSU, 1991). Candidate. tech. of Sciences math. model., num. methods and complexes of programs (TSU, 2007).

MARKOV, Anton Konstantinovich, student of the 3rd course in the direction 10.05.03 – Information security of automated systems.