УДК 004:003.26

ПРИЕМЫ ПОВЫШЕНИЯ СКОРОСТИ ШИФРОВАНИЯ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ АЛГОРИТМА ШИФРОВАНИЯ PRESENT

Ищукова Е.А., Салманов В.Д., Шамильян О.П., Половко И.Ю.

Южный федеральный университет, Институт компьютерных технологий и информационной безопасности, Таганрог, e-mail: uaishukova@sfedu.ru

В статье рассмотрены подходы, направленные на повышение скорости шифрования данных для алгоритма малоресурсной криптографии Present. Вопрос производительности шифра стоит очень остро тогда, когда необходимо обрабатывать большие объемы информации. Обычно это связано с решением задач по проверке надежности шифра, связанных с выполнением большого количества однообразных действий по зашифровыванию или расшифровыванию блоков данных. Для ускорения этого процесса можно использовать ряд средств и приемов, таких как использование распределенных многопроцессорных вычислений, оптимизация кода, разработка скоростных алгоритмов. В настоящей работе авторы предлагают рассмотреть несколько приемов, использование которых позволяет повысить скорость выполняемых преобразований в 100 раз. Такое большое ускорение достигается за счет замены обращений к массивам данных на использование операций конъюнкции, дизъюнкции, исключающее ИЛИ и т.д. В работе рассматриваются варианты программных реализаций для языков программирования Python и Си. Экспериментально было установлено, что реализация на языке Си с использованием разработанного скоростного алгоритма преобразует информацию в 30 раз быстрее, чем при использовании классической реализации на том же языке программирования. Реализация на языке Python работает еще медленнее: в 3 раза медленнее классической реализации на языке программирования Си и в 100 раз медленнее реализации на языке Си на основе использования разработанного скоростного алгоритма преобразования данных.

Ключевые слова: алгоритм шифрования, секретный ключ, раундовый подключ, циклический сдвиг, оптимизация, скорость шифрования

METHODS FOR INCREASING DATA ENCRYPTION SPEED USING PRESENT ENCRYPTION ALGORITHM

Ishchukova E.A., Salmanov V.D., Shamilyan O.P., Polovko I.Yu.

Southern Federal University, Institute of Computer and Information Security, Taganrog, e-mail: uaishukova@sfedu.ru

The article discusses approaches aimed at increasing the speed of data encryption for the algorithm of low-resource cryptography Present. The issue of cipher performance is very acute when it is necessary to process large amounts of information. This is usually associated with the solution of tasks to verify the reliability of the cipher associated with the implementation of a large number of uniform actions to encrypt or decrypt data blocks. To speed up this process, you can use a number of tools and techniques, such as the use of distributed multiprocessor computing, code optimization, development of high-speed algorithms. In this work, the authors propose to consider several techniques, the use of which allows to increase the speed of the performed transformations by 100 times. Such a great acceleration is achieved by replacing calls to data arrays with the use of conjunction, disjunction, exclusive OR, etc. The paper discusses options for software implementations for the programming languages Python and C. It was experimentally established that an implementation in C using the developed speed algorithm converts information 30 times faster than when using a classic implementation in the same programming language. A Python implementation is even slower: 3 times slower than a classical implementation in the C programming language and 100 times slower than a C implementation based on the use of the developed high-speed data transformation algorithm.

Keywords: encryption algorithm, secret key, round subkey, cyclic shift, optimization, encryption speed

В настоящей статье рассматривается блочный симметричный алгоритм шифрования Present [1], который представляет собой шифр так называемой малоресурсной криптографии. Малоресурсная криптография используется преимущественно в области Интернета вещей [2–4]. Используется там, где предъявляются особые требования к соотношению качества используемого шифра и объемов потребляемых им ресурсов (памяти, энергии и т.д.) [5].

Цель работы: исследовать возможность повышения скорости шифрования данных для алгоритма Present.

Материалы и методы исследования: алгоритм шифрования Present, ПЭВМ Intel Core i5-7300HQ 2.50GHz, языки программирования Python и C.

Алгоритм шифрования Present построен по принципу сети на основе подстановок и перестановок (SP-сеть). Алгоритм является блочным, за один раз обрабатывается блок данных размерностью 64 бита. Работа алгоритма осуществляется в течение 31 раунда шифрования. Считается, что такого большого количества раундов шифрования достаточно для того, чтобы обеспечить хороший запас надежности шифра. При

этом для алгоритма шифрования Present предусмотрена возможность использования секретных ключей шифрования различной длины, секретный ключ может содержать 80 или 128 бит.

Каждый раунд алгоритма состоит из трех базовых операций, а именно: поразрядное сложение данных с секретным раундовым подключом по модулю два, замена данных с помощью S-блока замены и перестановка битов по таблице перестановки P. Подробнее с работой алгоритма шифрования Present можно ознакомиться в работе [6]. В настоящей статье приведены только те данные, которые используются при применении приемов для повышения скорости реализации шифра.

Блок замены S работает аналогично тому, как это происходит в алгоритме шифрования «Магма» (бывший ГОСТ 28147-89) [7]. Перестановка P является линейной операцией и переставляет биты преобразуемого блока в соответствии с табл. 1, которая показывает, в каком порядке будут переставлены биты. Важно помнить, что при описании шифров обычно нумерация битов идет слева направо от 1 до n, где n – размерность блока.

Выше уже упоминалось, что для алгоритма шифрования Present предусмотрено использование двух вариантов секретного ключа: длиной 80 и 128 бит. Выработка раундовых подключей для каждого раундового преобразования происходит по следующему принципу.

Будем считать, что биты секретного ключа k нумеруются справа налево от 0 до n-1. Для секретного 80-битного ключа выполняются следующие действия:

Шаг I. В регистре ключа производится циклический сдвиг влево на 61 позицию.

Шаг II. Четыре старших бита заменяются с использованием S-блока замены.

Шаг III. К битам секретного ключа k_{19}, k_{18}, k_{17}, k_{16}, k_{15} добавляются с использованием операции XOR младшие значащие биты раундового счетчика.

Для секретного 128-битного ключа выполняются следующие действия:

Шаг I. В регистре ключа производится циклический сдвиг влево на 61 позицию.

Шаг II. Восемь старших битов заменяются с использованием S-блока замены.

Шаг III. К битам секретного ключа k_{66}, k_{65}, k_{64}, k_{63}, k_{62} добавляются с использованием операции XOR младшие значащие биты раундового счетчика.

Алгоритм шифрования Present преобразует данные в течение 31 раунда шифрования. При этом после 31 раунда выполняется дополнительное сложение данных с секретным раундовым подключом. Таким образом, для полного зашифровывания требуется иметь 32 раундовых подключа [1].

При исследовании любого шифра на предмет его стойкости всегда очень остро встает вопрос о том, с какой скоростью могут выполняться преобразования для данного шифра. В данном исследовании для выявления быстродействия преобразований было решено использовать программные реализации для двух языков программирования, а именно: Python и Си. Первая задача, которая ставилась, – сравнить скорость преобразования данных при использовании различных подходов (с оптимизацией и без оптимизации) к шифрованию данных, а также при использовании различных языков программирования. Так, сейчас большую популярность набирает язык программирования Python, что влечет за собой появление большого количества библиотечных функций, в том числе и криптографических. Кроме того, программная реализация на языке Python легко может быть преобразована в приложение, предназначенное для надежного и удобного шифрования файлов с любым разрешением. Следует отметить, что в случае шифрования файлов скорость преобразования информации не сильно влияет на общее время ожидания. Например, если файл имеет объем 1 Гб, то для его преобразования необходимо обработать всего 2^{27} блоков информации (1 Гб = 2^{10} Мб = 2^{20} Кб = 2^{30} байт, в одном обрабатываемом блоке содержится 8 байт = 2^3 байт = 1 блок). Обработка 2^{27} блоков информации выполняется, и преобразование может быть выполнено за приемлемое время. При рассмотрении различных техник по выявлению уязвимостей в алгоритме шифрования очень часто необходимо зашифровывать большие объемы данных. В этом случае вопрос скорости преобразований становится очень остро.

Таблица 1

Принцип работы преобразования P для шифра Present

1	5	9	13	17	21	25	29	33	37	41	45	49	53	57	61
2	6	10	14	18	22	26	30	34	38	42	46	50	54	58	62
3	7	11	15	19	23	27	31	35	39	43	47	51	55	59	63
4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64

Если говорить о преимуществах реализации на языке программирования Си, то следует отметить возможность использования библиотеки MPI (от англ. Message Passing Interface – библиотека для выполнения распределенных многопроцессорных вычислений) с целью сокращения общего времени анализа.

Ранее было отмечено, что для шифра Present можно использовать два варианта секретного ключа шифрования: длиной 80 или 128. При использовании языка программирования Си мы ориентируемся на применение максимально большой беззнаковой переменной типа unsigned long long, которая имеет размерность 64 бита. Получается, что для хранения секретного ключа шифрования (и размерностью 80 бит, и размерностью 128 бит) необходимо использовать как минимум две такие переменные. Обозначим их как Key_Left и Key_Right. В левой части ключа (переменная Key_Left) будут находиться старшие биты секретного ключа, а в правой части ключа (переменная Key_Right) соответственно будут находиться младшие биты. В случае, когда будет использоваться 128-битовый секретный ключ, обе переменные (Key_Left и Key_Right) будут использоваться целиком. При использовании 80-битного секретного ключа переменная Key_Right будет задействована полностью, а в переменной Key_Left будут использоваться только младшие 16 битов (старшие 48 битов переменной Key_Left будут являться незначащими).

Рассмотрим, как можно оптимизировать программную реализацию для выработки раундовых подключей шифрования. Для этого необходимо выполнять три действия так, как было описано выше: выполнить циклический сдвиг, заменить по таблице старший байт, добавить значение счетчика.

Будем рассматривать вариант с 80-битным секретным ключом. Перемещение битов секретного ключа в переменных Key_Left и Key_Right при выполнении циклического сдвига влево на 61 позицию будет выполнено так, как показано на рис. 1.

Пронумеруем биты секретного ключа справа налево, начиная от 0: $К_{79}$ до $К_0$. Тогда после циклического сдвига в переменной Key_Left будут находиться биты с $К_{18}$ по $К_3$, а в переменной Key_Right будут располагаться биты сначала с $К_2$ по $К_0$, потом с $К_{79}$ по $К_{17}$.

Введем три константы, с помощью которых мы сможем выделить необходимые биты: C1 = 0xFFFF; C2 = 0xE000000000000000; C3 = 0x1FFFFFFFFFF. Кроме того, введем две временные переменные temp_Left и temp_Right, инициализировав их значением 0 (temp_Left = 0, temp_Right = 0). В этом случае необходимо будет выполнить следующий набор действий для выполнения операции циклического сдвига:

Шаг 1 (сдвиг вправо на три позиции): temp_Left=(Key_Right>>3);

Шаг 2 (выделили биты с К18 по К3): temp_Left = temp_Left &C1;

Шаг 3 (выделили биты с К2 по К0): temp_Right =(Key_Right<<61)&C2;

Шаг 4 (соединили биты с К2 по К0 с битами с К79 по К64) temp_Right = temp_Right^ (Key_Left<<45);

Шаг 5 (выделили биты с К63 по К17) temp_Right = temp_Right ^((Key_Right>>19)&C3);

Шаг 6 (записали новое значение битов ключа) Key_Left=temp_Left; Key_Right=temp_Right;

Следующим шагом является замена по таблице для 4 старших битов. Для этого их необходимо выделить из общего блока, преобразовать по таблице и записать в исходные позиции. Старшие биты ключа находятся в переменной Key_Left, поэтому нам необходимо использовать константу C4=0xFFF для отделения тех 12 битов, которые менять не нужно. Также будем использовать две временные переменные temp1 и temp2:

Шаг 7 (выделили 4 старших бита): temp1=(Key_Left>>12)&15;

Шаг 8 (выделили 12 младших битов): temp2=Key_Left&C4;

Шаг 9 (заменили по таблице): temp1=S-box[temp1];

Шаг 10 (сформировали новое значение): Key_Left=(temp1<<12)^temp2.

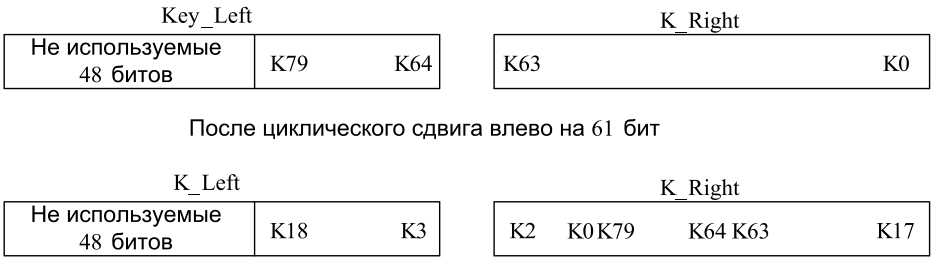

Рис. 1. Преобразование битов секретного ключа для случая, когда ключ содержит 80 битов

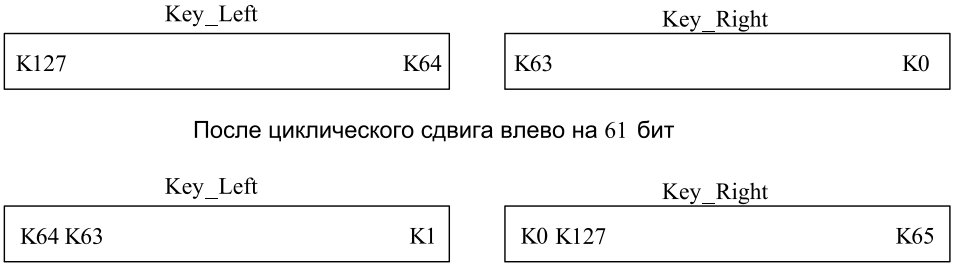

Рис. 2. Преобразование битов секретного ключа для случая, когда ключ содержит 128 битов

Остается последний этап при выработке раундового подключа: добавление значения счетчика к определенным битам (с K_{19} по K_{15}). Для того чтобы это сделать, нужно сместить переменную счетчика (amount) на 15 позиций влево:

Шаг 11 (сложение со счетчиком): Key_Right = Key_Right^(amount<<15).

В случае использования 128-битного секретного ключа обе переменные (Key_Left и Key_Right) будут задействованы полностью так, как показано на рис. 2.

В этом случае для выработки раундовых подключей необходимо будет использовать константу C5 = 0xFFFFFFFFFFFFFFFE, а также две временные переменные temp_Left и temp_Right. Алгоритм будет состоять из следующих шагов:

Шаг 1 (сделали бит K_{64} самым старшим, остальные обнулили): temp_Left=(Key_Left&1)<<63;

Шаг 2 (добавили в левую часть биты с 63 по 1) temp_Left=temp_Left^((Key_Right&C5)>>1);

Шаг 3 (сделали бит K_0 старшим, остальные обнулили): temp_Right=(Key_Right&1)<<63;

Шаг 4 (добавили биты с 127 по 65): temp_Right=temp_Right^((Key_Right&C5)>>1);

Шаг 5 (получили новое состояние): Key_Left=temp_Left; Key_Right=temp_Right;

Следующим шагом является замена по таблице для 8 старших битов (две группы по 4 бита). Для этого их необходимо выделить из общего блока, преобразовать по таблице и записать в исходные позиции. Старшие биты ключа находятся в переменной Key_Left, поэтому нам необходимо использовать константу C6=0xFFFFFFFFFFFFFFF. Также будем использовать три временные переменные temp1, temp2, temp3:

Шаг 6 (выделили первые 4 старших бита): temp1=(Key_Left>>60)&15;

Шаг 7 (выделили вторые 4 старших бита): temp2=(Key_Left>>56)&15;

Шаг 8 (выделили младшие 56 битов): temp3=Key_Left&C6;

Шаг 9 (заменили обе группы битов по таблице): temp1=S-box[temp1]; temp2=S-box[temp2];

Шаг 10 (сформировали новое значение): Key_Left=(temp1<<60)^(temp2<<56)^temp3.

Остается последний этап при выработке раундового подключа: добавление значения счетчика к битам с K_{66} по K_{62}. Эти биты затрагивают обе переменные (Key_Left и Key_Right), поэтому необходимо выполнить преобразование обеих переменных:

Шаг 11 (добавили старшие 3 бита счетчика) Key_Left=Key_Left^((amount >>2)&7);

Шаг 12 (добавили младшие 2 бита счетчика) Key_Right=Key_Right^((amount&3)<<62;

Помимо процедуры выработки раундовых подключей, в самом алгоритме Present есть преобразование, которое требует постоянного обращения к массиву – это операция перестановки битов. В силу того что операции простой логики работают на порядок быстрее операций чтения/записи данных в/из массива, было принято решение разработать алгоритм, который бы позволил выполнять преобразование перестановки битов без обращения к массиву. Если внимательно изучить таблицу перестановки (табл. 1), можно заметить, что биты 1, 22, 43 и 64 после перестановки все равно остаются на своих местах. Поэтому можно зафиксировать данные биты, умножив переменную на значение 0x8000040000200001. Дальнейшее изучение таблицы 1 покажет нам, что 5-й бит переместится в позицию 2. Также при циклическом сдвиге влево на 5 позиций биты 26 и 47 также попадут в нужные позиции. Данные биты можно зафиксировать, умножив переменную на константу 0x4000020000100000. Действуя по той же схеме, можно будет выполнить полную перестановку битов, всегда пользуясь только одной формулой temp=temp^(((Block<<sdv_left)|(text>>sdv_right))&Const), где Block – это 64-битная переменная, для которой необходимо выполнить перестановку битов, а значения sdv_left, sdv_right и Const изменяются в соответствии с табл. 2.

Таблица 2

Параметры для выполнения перестановки с помощью операций простейшей логики

Получаемые биты	sdv_left	sdv_right	Const		Получаемые биты	sdv_left	sdv_right	Const
1, 22, 43, 6	0	0	0x8000040000200001		2, 23, 44	49	15	0x0000800004000020
5, 26, 47	3	61	0x4000020000100000		6, 27, 48	52	12	0x0000400002000010
9, 30, 51	6	58	0x2000010000080000		10, 31, 52	55	9	0x0000200001000008
13, 34, 55	9	55	0x1000008000040000		14, 35, 56	58	6	0x0000100000800004
17, 38, 59	12	52	0x0800004000020000		2,23,44	61	3	0x0000080000400002
21, 42, 63	15	49	0x0400002000010000		3, 24	34	30	0x0000000080000400
25, 46	18	46	0x0200001000000000		7, 28	37	27	0x0000000040000200
29, 50	21	43	0x0100000800000000		11, 32	40	24	0x0000000020000100
33, 54	24	40	0x0080000400000000		15,36	43	21	0x0000000010000080
37, 58	27	37	0x0040000200000000		19, 40	46	18	0x0000000008000040
41, 62	30	34	0x0020000100000000		4	19	45	0x0000000000008000
45	33	31	0x0010000000000000		8	22	42	0x0000000000004000
49	36	28	0x0008000000000000		12	25	39	0x0000000000002000
53	39	25	0x0004000000000000		16	28	36	0x0000000000001000
57	42	22	0x0002000000000000		20	31	33	0x0000000000000800
61	45	19	0x0001000000000000					

Таблица 3

Сравнение скоростных показателей для разных реализаций

Реализация	Язык программирования	Время обработки одного блока, секунд
Классическая реализация	Python	0,000929
Классическая реализация	C	0,000288
Скоростная реализация	C	0,000009

Результаты исследования и их обсуждение

В результате работы над проектом было получено несколько программных реализаций для алгоритма шифрования Present. Первая программная реализация была сделана на основе кода в открытом доступе [8]. Вторая реализация была получена на основе данных из [9] и представляет собой классическую реализацию шифра на языке программирования Си. Третья реализация на языке программирования Си была получена на основе предложенного алгоритма оптимизации. Все эксперименты выполнялись для варианта шифра с 80-битным секретным ключом на ПЭВМ с характеристикой Intel Core i5-7300HQ 2.50GHz. В эксперименте определялась средняя скорость зашифровывания одного блока данных, включая процесс выработки раундовых подключей. Средняя скорость шифрования определялась путем деления общего времени, затраченного на шифрование n блоков, на значение n. Результаты экспериментов представлены в табл. 3.

Заключение

Экспериментально было установлено, что реализация на языке Си с использованием разработанного скоростного алгоритма преобразует информацию в 30 раз быстрее, чем при использовании классической реализации на том же языке программирования. Реализация на языке Python работает еще медленнее: в 3 раза медленнее классической реализации на языке программирования Си и в 100 раз медленнее реализации на языке Си на основе использования разработанного скоростного алгоритма преобразования данных.

Работа выполнена при поддержке гранта РФФИ № 17-07-00654 «Разработка и исследование последовательных и параллельных алгоритмов анализа современных симметричных шифров с использованием технологий MPI, NVIDIA CUDA, SageMath».

Список литературы

1. Bogdanov A., Knudsen L.R., Leander G., Paar C., Poschmann A., Robshaw M.J.B., Seurin Y., Vikkelsoe C. PRESENT: An Ultra-Lightweight Block Cipher. [Electronic resource]. URL: https://web.archive.org/web/20101217063636/http://www.ist-ubisecsens.org/publications/present_ches2007.pdf (date of access: 20.03.2020).

2. ECRYPT Network of Excellence. The Stream Cipher Project: eSTREAM. [Electronic resource]. URL: www.ecrypt.eu.org/stream (date of access: 20.03.2020).

3. Toru Akishita, Harunaga Hiwatari Very Compact Hardware Implementations of the Blockcipher CLEFIA. [Electronic resource]. URL: http://www.sony.co.jp/Products/cryptography/clefia/download/data/clefia-hw-compact-20110615.pdf (date of access: 20.03.2020).

4. Christophe De Canni`ere and Bart Preneel Trivium Specifications. [Electronic resource]. URL: http://www.ecrypt.eu.org/stream/ciphers/trivium/trivium.pdf (date of access: 20.03.2020).

5. Ищукова Е.А., Толоманенко Е.А. Анализ алгоритмов шифрования малоресурсной криптографиив контексте интернета вещей // Современные наукоемкие технологии. 2019. № 3–2. С. 182–186.

6. Панасенко С.П. Алгоритмы шифрования. Специальный справочник. СПб.: БХВ-Петербург, 2009. 576 с.

7. Бабенко Л.К. Ищукова Е.А. Современные алгоритмы блочного шифрования и методы их анализа. М.: «Гелиос АРВ», 2006. 376 с.

8. Test vectors for PRESENT algorithm? [Electronic resource]. URL: https://crypto.stackexchange.com/questions/26037/test-vectors-for-present-algorithm (date of access: 22.03.2020).

9. Present-C [Electronic resource]. URL: https://github.com/bozhu/PRESENT-C/blob/master/present.h (date of access: 22.03.2020).