Нижегородский государственный университет им. Н.И. Лобачевского Радиофизический факультет

**Зачётная работа**

По теме: “ *АЛГОРИТМ ШИФРОВАНИЯ EFFICIENT ENCRYPTION (E2)*”

Выполнил: ст-т 4 курса

радиофизического факультета группы 447 Штрауб Н.И.

Нижний Новгород

2019 г.

**Содержание**

[Введение 2](#_Toc26114072)

[1. Общие функции 2](#_Toc26114073)

[2. Генератор ключей 4](#_Toc26114074)

[3. Расширение ключа 5](#_Toc26114075)

[4. Алгоритм шифрования 7](#_Toc26114076)

[5. Часть main файла encryption 12](#_Toc26114077)

[6. Алгоритм расшифрования 13](#_Toc26114078)

[7. Часть main файла decryption 15](#_Toc26114079)

[Заключение 15](#_Toc26114080)

[Литература 16](#_Toc26114081)

# **Введение**

*Efficient Encryption (E2)* — эффективное шифрование — симметричный блочный криптоалгоритм на основе ячейки Фейстеля. E2 использует блок размером 128 бит и ключи длиной 128, 192, 256 бит. Создан в компании NTT (Nippon Telegraph and Telephone) в 1998 году и был представлен на AES конкурсе.

E2 прошел тест на криптостойкость успешно. Стойкость шифра E2 не повлияла на его быстродействие. E2 занял одну из лидирующих позиций как в соревновании на скорость шифрования/расшифрования, так и в быстроте формирования ключей. В частности, реализация шифра E2 (компилятор Borland) показала скорость шифрования/расшифрования 26 Мбит/сек. Несмотря на то, что показатели шифра менялись в зависимости от компилятора, платформы и логики, общая тенденция оставалась неизменной. Большинство авторов, писавших о конкурсе AES, утверждают, что E2 наряду с некоторыми другими шифрами успешно прошел первый круг. Однако E2 не попал в финал в пятерку лучших шифров. НИСТ было отмечено, что несмотря на хорошие показатели скорости и отсутствие уязвимостей, требования к энергонезависимой памяти слишком высоки.

# **Общие функции**

Следует начать с рассмотрения общих функций, которые используются почти в каждом блоке программы. Их всего две: *S – функция* и *P – функция*.

*S – функция* выглядит следующим образом:

*S – функция* определяется, так называемым, *S – box*[[1]](#footnote-1). Структура *S – box* следующая:

, где

Не возбраняется для более быстрых расчетов пользоваться таблицами с уже вычисленными значениями *S(x)*. То есть:

*.*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 225 | 66 | 62 | 129 | 78 | 23 | 158 | 253 | 180 | 63 | 44 | 218 | 49 | 30 | 224 | 65 |
| 204 | 243 | 130 | 125 | 124 | 18 | 142 | 187 | 228 | 88 | 21 | 213 | 111 | 233 | 76 | 75 |
| 53 | 123 | 90 | 154 | 144 | 69 | 188 | 248 | 121 | 214 | 27 | 136 | 2 | 171 | 207 | 100 |
| 9 | 12 | 240 | 1 | 164 | 176 | 246 | 147 | 67 | 99 | 134 | 220 | 17 | 165 | 131 | 139 |
| 201 | 208 | 25 | 149 | 106 | 161 | 92 | 36 | 110 | 80 | 33 | 128 | 47 | 231 | 83 | 15 |
| 145 | 34 | 4 | 237 | 166 | 72 | 73 | 103 | 236 | 247 | 192 | 57 | 206 | 242 | 45 | 190 |
| 93 | 28 | 227 | 135 | 7 | 13 | 122 | 244 | 251 | 50 | 245 | 140 | 219 | 143 | 37 | 150 |
| 168 | 234 | 205 | 51 | 101 | 84 | 6 | 141 | 137 | 10 | 94 | 217 | 22 | 14 | 113 | 108 |
| 11 | 255 | 96 | 210 | 46 | 211 | 200 | 85 | 194 | 35 | 183 | 116 | 226 | 155 | 223 | 119 |
| 43 | 185 | 60 | 98 | 19 | 229 | 148 | 52 | 177 | 39 | 132 | 159 | 215 | 81 | 0 | 97 |
| 173 | 133 | 115 | 3 | 8 | 64 | 239 | 104 | 254 | 151 | 31 | 222 | 175 | 102 | 232 | 184 |
| 174 | 189 | 179 | 235 | 198 | 107 | 71 | 169 | 216 | 167 | 114 | 238 | 29 | 126 | 170 | 182 |
| 117 | 203 | 212 | 48 | 105 | 32 | 127 | 55 | 91 | 157 | 120 | 163 | 241 | 118 | 250 | 5 |
| 61 | 58 | 68 | 87 | 59 | 202 | 199 | 138 | 24 | 70 | 156 | 191 | 186 | 56 | 86 | 26 |
| 146 | 77 | 38 | 41 | 162 | 152 | 16 | 153 | 112 | 160 | 197 | 40 | 193 | 109 | 20 | 172 |
| 249 | 95 | 79 | 196 | 195 | 209 | 252 | 221 | 178 | 89 | 230 | 181 | 54 | 82 | 74 | 42 |

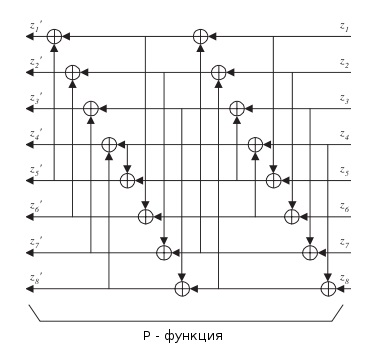
*Таблица 1. Вычисленные значения S – box.*

В данном примере реализации данного шифра была использована эта таблица.

*P – функция* обеспечивает рассеивание данных путем сложения значений (8 – битные блоки информации, речь о которых пойдет далее) по модулю 2 в определенном порядке.

*P – функция* выглядит следующим образом:

*P –* матрица преобразования, описывающая *P – функция:*



*Рис.1.1. Схема сложения по модулю два P – функции.*

# **Генератор ключей**

В *Efficient Encryption* используются ключи размером 128, 192 или 256 бит. Для генерации псевдослучайных и псевдопростых чисел такого размера использовались встроенная в ЯП Python функция *randint* из библиотеки *random* и тест простоты Ферма, написанный самостоятельно.

Тест простоты ферма в этой реализации состоит из 4 функций:

1. Функция расчёта ключа (*key\_calculation*), которая ничего не принимает на входе, а возвращает сгенерированный ключ, являющейся псевдослучайным и псевдопростым натуральным числом.
2. В функции *key\_calculation* вызывается еще две функции, одна из которых называется *generation\_odd\_key*. Эта функция не принимает никаких переменных, возвращая при этом псевдослучайный ключ, длинной, которая задается в самом начале этой функции (128, 192 или 256 бит по условию алгоритма *E2*). Также в этой функции осуществляется проверка сгенерированного ключа на нечетность, так как тест Ферма предполагает использование нечетных чисел. Поэтому ключи будут генерироваться до тех пор, пока не будут нечетными. Функция *generation\_odd\_key* вызывается в начале функции *key\_calculation* и далее будет вызываться до тех пор, пока функция *FERMA* не возвратит значение *True*.
3. Функция *gsd* предназначена для отыскания НОД. Принимает два параметра *a* и *b*. В цикле выполняется операция сравнения этих чисел. Если , то параметру *a* присваивается значение остатка от деления *a* на *b*. В противном случае (), параметру *b* присваивается значение остатка от деления *b* на *a*. Цикл выполняется до тех пор, пока один из параметров не станет равным нулю. Функция *gsd* вызывается в функции *FERMA*.
4. Непосредственно функция самого теста Ферма (*FERMA*), принимающая на входе сгенерированный ключ для дальнейшей проверки. Вероятность того, что тест Ферма подтвердит простоту числа задается в начале функции параметром *k* (чем больше *k*, тем выше вероятность). Далее в цикле от 0 до *k*. В нем сначала псевдослучайным образом выбирается параметр *a* от 2 до K-1, где K – значение ключа. Далее значения *a*, *K-1*, *K* передаются встроенной функции возведения в степень *pow* по модулю *K*. То есть . Далее идет проверка двух условий: первое – если возвращаемое значение функции , то *FERMA* возвращает значение *False* и тогда генерируется новый ключ, который снова передается в функцию *FERMA*; второе – если , то *FERMA* снова возвращает значение *False* и тогда снова генерируется новый ключ. Если же весь цикл выполнится *k* раз, то функция *FERMA* возвратит значение *True* и функция *key\_calculation* возвратит сгенерировавшийся ключ.

# **Расширение ключа**

На основе секретного ключа ( имеет размерность половины блока, то есть 64 бита и является аргументом для функций шифрования и расшифрования) генерируются подключи (битовые вектора размерности 128) с помощью G-функции и S-функции.

Процедура генерации ключей остается почти неизменной, если длина секретного ключа равна 128, 192 или 256 бит. Если заданная длина 128 бит, в качестве значений выбираются константы следующим образом:

, . Если длина ключа 192 бита, значение ключа — , где *S()* — S-функция.

было взято равным , которое в дальнейшем разбивалось по 8 бит и отправлялось в *S – функцию*, а после вычислений конкатенацией соединялось в том же порядке.

Если же длина секретного ключа равняется 256 битам, то первые 64 бита присваиваются , вторые – и так далее.

Функция, которая занимается этим преобразованием в данной реализации программы называется *splitting\_key*. Она принимает единственный параметр – секретный ключ, который сгенерирован ранее. Возвращает 4 ключа , которые далее расширяются *G – функцией*.

*G – функция* расширяет ключи таким образом, что получается 16 подключей длинной 128 бит каждый. Ее структура выглядит следующим образом:

,

где , ,

– f-функция.

Структура же *f – функции* выглядит так:

То есть *f – функция* использует *P –* и *S – функции*, описанные в первом пункте.

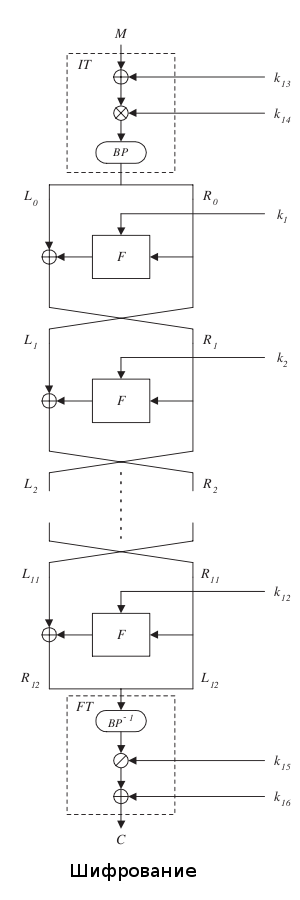
Длина параметра равна 128 битам, а сам параметр выбирался равным . Перед отправкой в S – функцию это число разбивалось по 8 бит. Затем после вычислений соединялось конкатенацией в том же порядке. В целом, все, что требовало использования *S – функции* сначала разбивалось по 8 бит и в дальнейшем соединялось конкатенацией.

*G – функция* в конечном итоге возвращает массив, состоящий из четырех 128 битных ключей.

# **Алгоритм шифрования**

После генерации и расширения ключа можно приступить к шифрованию данных (в расшифровании используется только расширение ключей, которые записываются в файл при генерации).

Работу алгоритма шифрования можно разделить на три основные части: *IT-функция*, или преобразователь начальных данных (англ. initial transformation (IT)), ячейка Фейстеля[[2]](#footnote-2) на базе F-функции, повторяющаяся 12 раз, и *FT-функция*, или преобразователь конечных данных (англ. finale transformation (FT)). Функция, выполняющая эти действия в данной реализации, называется *ENCODE*. Она принимает 128 битный блок входных данных и сгенерированный ключ, а возвращает 128 битный блок зашифрованных данных.



*Рис.4.1. Схема шифрования.*

Первое преобразование открытого текста *M* производится с помощью *IT-функции* и двух сгенерированных ключей под номерами 13 и 14 ().

Структура *IT-функции* в данной реализации программы следующая:

,

где *H* – пространство слов бинарного алфавита размерности 64 бит; *X, A, B* – бинарные слова размерности 128 бит; *BP()* – BP-функция.

В данном случае *X* – это 128 битный блок информации, поступающий в *IT-функцию*; *A* – ключ ; *B* – ключ .

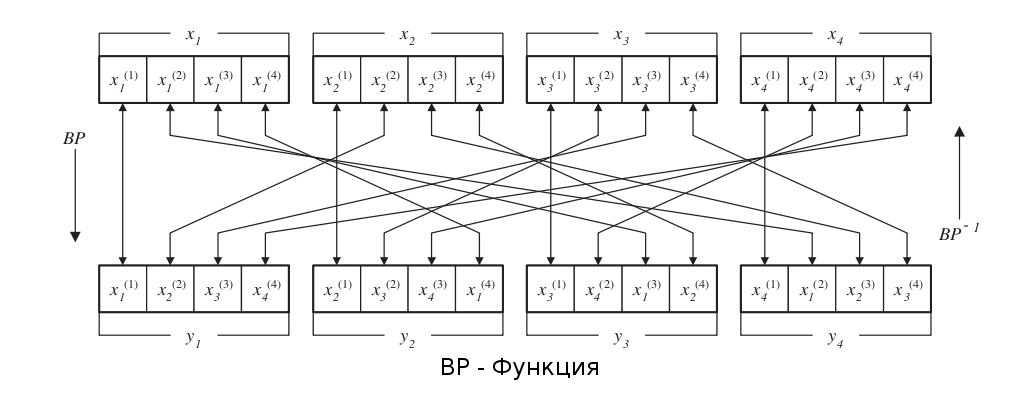
То есть *IT-функция* принимает три аргумента. Первым делом информация складывается по модулю два с ключом . Далее полученный результат разбивается на четыре блока по 32 бита и складывается по модулю с разделенным на такие же 32 битовые блоки ключом . Затем все соединяется в том же порядке конкатенацией. Завершающей операцией в *IT-функции* является *BP-функция*[[3]](#footnote-3). Она определяется следующим образом:

,

где ,

.

Приведенная ниже схема *BP-функции* иллюстрирует и обратную к ней *BP-1-функцию*, которая используется в *FT-функции* и ее описание будет продемонстрировано ниже.

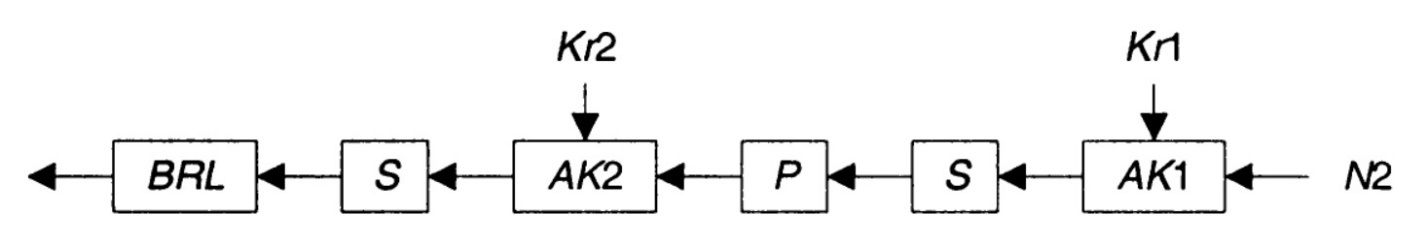


*Рис.4.2. Схема BP – функций.*

В конечном итоге *IT – функция* возвращает 128 битный блок данных (), выполнив первую стадию преобразований.

Далее идут преобразования, основанные на *ячейке Фейстеля на базе F-функции*. разбивается на два блока и равной длины, каждый из элементов и является битовым вектором размерностью 64 бита. Затем выполняются 12 циклов преобразований в ячейке Фейстеля, в которой правый блок на текущей итерации цикла определяется сложением по модулю два левой части предыдущей итерации цикла и результата функции *F*, аргументами которой являются правая часть предыдущей итерации и ключ , а левому блоку на *r* шаге цикла присваивается значение правого блока на *r-1* шаге. Цикл повторяется 12 раз, то есть *r* изменяется от 1 до 12.

*F – функция* состоит из нескольких последовательно выполняемых операций, показанных на рисунке 4.3. Все указанные операции выполняются над отдельными байтами субблока (через обозначена 64 – битная последовательность данных, поступающая на вход *F – функции*), то есть можно считать, что 64 – битный субблок разбивается на 8 подблоков , каждый из которых имеет размер 8 битов.



*Рис.4.3. Схема F – функции.*

Операции и выполняют наложение соответственно левой и правой частей ключа раунда на входной субблок данных. Наложение выполняется побитовым сложением по модулю два каждого байта субблока и соответствующего ему ключа раунда. То есть операции и выполняются так ( и обозначают, соответственно, текущее значение *x-го* байта субблока перед конкретной операцией и после нее):

;

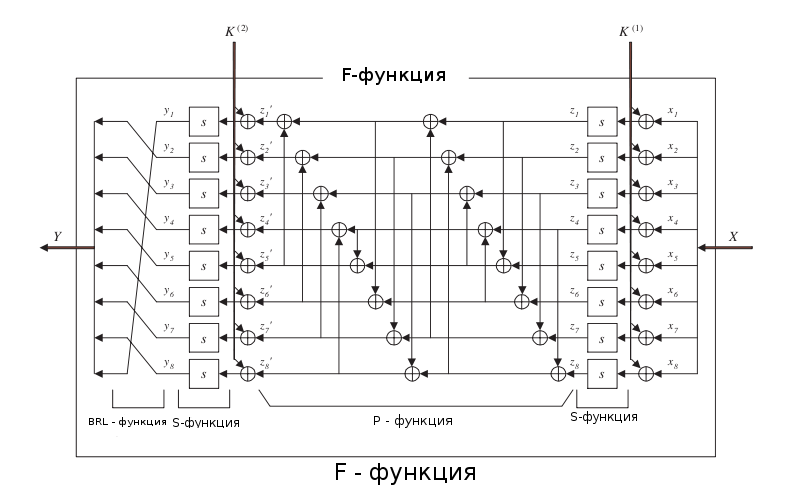
.

*S – функция* и *P – функция* выполняют преобразования, описанные в первом пункте.

*BRL – функция* (англ. byte rotate left function), или циклический сдвиг влево:

,

где – бинарное слово размерности 8 бит.



*Рис.4.4. Расширенная схема F – функции.*

Финальный этап шифрования — выполнение *FT-функции*. Результат *FT-функции*, аргументами которой являются конкатенация правой и левой частей на выходе 12 итерации ячейки Фейстеля и ключи :

Структура *FT-функции*:

.

*H* – пространство слов бинарного алфавита размерности 64 бит; *X* (соответствует входным данным)*, A* (соответствует ключу )*, B* (соответствует ключу ) – бинарные слова размерности 128 бит.

Операция выполняется следующим образом:

сначала ключ и данные делятся на блоки по 32 бита каждый, и из блока данных вычитается соответствующий ей блок ключа; затем, если получившееся значение отрицательное, то к нему прибавляется ; если значение положительное, то выполняется следующая операция *FT – функции*.

Функция – обратная *BP – функции*. Ее структура выглядит так:

,

где ,

.

Схема *BP-1-функции* представлена на рисунке 4.2.

*FT-функция* – это функция, обратная *IT-функции*:

# **Часть main файла encryption**

Перед выполнением программы необходимо создать текстовый файл с названием *“input\_file.txt”*, в котором будет содержаться информация, подлежащая шифрованию.

Первым делом в программе открываются три текстовых файла, один из которых, как сказано ранее, должен быть создан до выполнения программы и содержать информация для шифрования. Два других файла создавать необязательно, так как программа, в случае их отсутствия создаст их сама. Эти два файла будут содержать зашифрованную информацию, записанную в двоичном виде (в виде “0” и “1”), и ключи, записанные построчно. Их названия *“encoded\_file.txt”* и *“keys.txt”* соответственно.

После открытия файлов создаются две переменных, которые используются в цикле для посимвольного считывания входной информации. Условие выхода из цикла – конец файла. Каждые 16 итераций цикла генерируется ключ шифрования, и считанная информация (128 битная последовательность) отправляется в функцию *ENCODE* вместе с ключом. Затем ключ и зашифрованная информация записываются в соответствующие файлы. Запись информации происходит после преобразования функцией *FIX\_BIT*, которая представляет зашифрованную последовательность в виде 128 битных блоков двоичной последовательности.

По выходу из цикла происходит проверка переменной, с помощью которой производилось считывания информации, на предмет наличия в ней информации. Если она не равна нулю, то происходит единожды генерация секретного ключа, отправка информации в *ENCODE* и запись всего этого в соответствующие файлы.

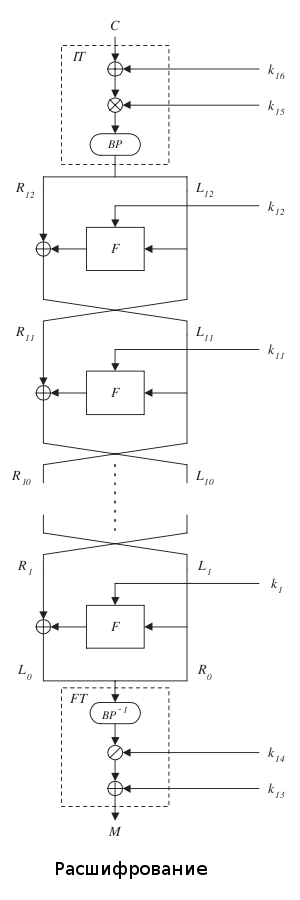
После выполнения все файлы закрываются, и программа завершается.

# **Алгоритм расшифрования**

Расшифрование происходит по схеме, аналогичной шифрованию. Работу алгоритма расшифрования, можно разделить на три основные части: *IT-функция* (начальное преобразование — англ. initial information (IT)), 12 циклов ячейки Фейстеля с *F-функцией* и в конце *FT-функция* (англ. finale transformation (FT)). Блок алгоритма, отвечающий за расширение ключей, из секретного ключа, который считывается из файла, непосредственно перед шифрованием генерирует 16 подключей , которые являются битовыми векторами размерностью 128. На первом этапе происходит выполнение IT-функции, аргументами которой являются криптограмма и два подключа .

Результат *IT-функции* разбивается на 2 равные части по 64 бита (половина блока): правую и левую . Далее выполняются 12 циклов ячейки Фейстеля на базе *F-Функции* ( меняется от 12 до 1).

По завершении последнего цикла ячейки Фейстеля осуществляется конкатенация половинок блока . И в конце — финальное преобразование: выполняется *FT-функция*, аргументами которой являются результат конкатенации и два ключа . Результатом выполнения *FT-функции* является открытый текст .



*Рис.6.1. Схема расшифрования.*

Как видно из рисунка 6.1, алгоритм расшифрования отличается лишь последовательностью применения ключей.

# **Часть main файла decryption**

*Decryption* запускается после отработки программы *encryption*, то есть когда существует файл с зашифрованными данными. Этот файл называется *“encoded\_file.txt”*.

Также, для расшифрования необходим файл с ключами, который также открывается при выполнении программы (*“keys.txt”*).

Файл же с расшифрованными данными создавать необязательно – он создастся автоматически.

Как уже стало понятно, сначала программа открывает три файла. Файл *“encoded\_file.txt”* сразу считывается в переменную и затем закрывается. Далее создаются переменные, необходимые для обработки и расшифровки входной информации.

Так как зашифрованная информация хранится в файле в виде подряд идущих 128 битных последовательностей, то внешний цикл выполняется до тех пор, пока не считаются все последовательности.

Далее идет внутренний цикл, в котором происходит считывание 128 битной последовательности. То есть, как только считалось 128 бит данных из созданной в начале переменной, считывается строка из файла с ключами, всё это переводится в значение int, и они сразу отправляются на обработку в *DECODE*.

После того, как *DECODE* расшифровал данные, они посимвольно переводятся в char и записываются в файл *“decoded\_file.txt”*.

После выполнения основного цикла файлы *“decoded\_file.txt”* и *“keys.txt”* закрываются. На этом выполнение программы *decryption* завершается.

# **Заключение**

Сеть Фейстеля “по определению” дает алгоритму ряд преимуществ: алгоритмы, основанные на ней, применялись в течение десятилетий, их свойства хорошо изучены, что дает некоторую гарантию отсутствия как скрытых уязвимостей, так и незадекларированных возможностей. Известны также сильные и слабые стороны таких алгоритмов по отношению к различным методам криптоанализа. Кроме того, неоспоримым достоинством такой структуры является “симметричность” процедур зашифрования и расшифрования, для реализации которых можно использовать практически один и тот же программный код или аппаратный модуль.

Алгоритм *E2* является очередным подтверждением надежности алгоритмов шифрования, основанных на сети Фейстеля, – в процессе анализа данного алгоритма не было выявлено каких-либо слабостей (за исключением “усеченных” версий алгоритма – с уменьшенным количеством раундов) и серьезных недостатков.

Недостатками же *E2* являются:

1. Алгоритм предъявляет очень высокие требования к энергонезависимой памяти, что делает весьма затрудненными его применение в таких устройствах, как смарт-карты;
2. Алгоритм показывает относительно невысокую скорость шифрования;
3. Дополнительные ресурсы требуются для функции расширения ключа, что также не способствует реализации *E2* в смарт-картах.

Таким образом, разработчики алгоритма *E2* создали действительно сильный алгоритм шифрования, не подверженный каким-либо известным или прогнозируемым атакам. Однако высокая криптостойкость алгоритма достигается за счет невысокой скорости шифрования и активного использования вычислительных ресурсов и памяти.

# **Литература**

[1] Панасенко, Сергей Петрович. Алгоритмы шифрования. Специальный справочник. С. П. Панасенко. – СПб.: БХВ-Петербург, 2009. – 576 с.: ил. – (справочник). – ISBN 978-5-9775-0319-8. С. 201 – 206.

[2] https://ru.wikipedia.org/wiki/E2\_(шифр)

1. *S-блок* (англ. s-box от substitution-box), «блок подстановок» — функция в коде программы или аппаратная система, принимающая на входе n бит, преобразующая их по определённому алгоритму и возвращающая на выходе m бит. n и m не обязательно равны [↑](#footnote-ref-1)
2. *Сеть Фе́йстеля*, или конструкция Фейстеля (англ. Feistel network, Feistel cipher), — один из методов построения блочных шифров. Сеть состоит из ячеек, называемых ячейками Фейстеля. На вход каждой ячейки поступают данные и ключ. На выходе каждой ячейки получают изменённые данные. Все ячейки однотипны, и сеть представляет собой определённую многократно повторяющуюся (итерированную) структуру. При шифровании и расшифровании выполняются одни и те же операции; отличается только порядок ключей. [↑](#footnote-ref-2)
3. *BP – функция*, или функция перестановки байтов (англ. byte permutation), является частью IT-функции и FT — функции. [↑](#footnote-ref-3)