

## Оглавление

<b>Введение .....</b>	<b>3</b>
<b>Глава 1. Теоретическая часть .....</b>	<b>4</b>
Историческая справка .....	4
Общие сведения о Twofish .....	5
Архитектура шифра .....	6
Отбеливание .....	8
Сеть Фейстеля .....	8
Структура раунда .....	10
Функция F .....	11
Функция g .....	12
S-boxes .....	13
MDS-матрицы .....	16
Псевдо-преобразование Адамара .....	17
Ключевое расписание .....	18
Достоинства и недостатки .....	20
Применение шифра .....	21
<b>Глава 2. Практическая часть .....</b>	<b>22</b>
Реализация алгоритма .....	22
Вычисление скоростных характеристик .....	27
<b>Заключение .....</b>	<b>28</b>
<b>Список использованных источников .....</b>	<b>29</b>

## **Введение**

Шифрование один из самых мощных и популярных способов обеспечения безопасности данных. На сегодняшний день огромному количеству организаций приходится работать с большим количеством персональных данных своих клиентов и сотрудников. Чтобы избежать юридических санкций и поддержать деловую репутацию, защита этих данных становится одной из первостепенных обязанностей. Шифрование помогает компаниям надежно хранить и передавать данные, сохраняя целостность и конфиденциальность.

Опорой всех современных технологий, которые работают с множеством взаимосвязанных объектов, лежит организация безопасности данных. Шифрование гарантирует, что конфиденциальная информация, которой обмениваются эти объекты, не подвержена утечке или раскрытию, что делает его неотъемлемой частью любой хорошей системы работы с данными.

В данной курсовой будет проведено исследование и анализ симметричного алгоритма блочного шифрования Twofish, а также реализация данного шифра на языке программирования C#.

**Объектом** курсовой работы являются алгоритм шифрования Twofish.

**Предмет** курсовой работы — обеспечение надежной безопасности информации при помощи алгоритма шифрования Twofish.

**Целью** курсовой работы является выявление преимуществ и недостатков алгоритма шифрования Twofish и его оценка.

Для достижения цели работы были поставлены следующие **задачи**:

Исследование архитектуры и принципа работы алгоритма шифрования Twofish.

Реализация алгоритма шифрования Twofish на языке программирования C#.

## **Глава 1. Теоретическая часть**

### **Историческая справка**

В 1972 и 1974 годах Национальный институт стандартов и технологий (NIST) опубликовало первый публичный запрос на алгоритм шифрования для своего нового стандарта шифрования. Таким алгоритмом стал модернизированный компанией IBM алгоритм «Люцифер», который в дальнейшем получил название DES, возможно, наиболее широко используемым и успешным алгоритм шифрования в мире на протяжении долгих лет.

Несмотря на свою популярность, DES страдает от противоречий. Некоторые криптографы возражали против закрытого процесса разработки алгоритма и задавались вопросом, добавило ли АНБ люк, позволяющий тайно взломать алгоритм. Некоторые считали 56-битный ключ слишком коротким и на то время, а к нулевым годам, с ростом производительности процессоров, методом перебора ключа алгоритм было возможно взломать на домашнем персональном компьютере.

Triple-DES появился как промежуточное решение для банковских и других консервативных систем, но для некоторых целей он слишком медленный. Что более важно, 64-битная длина блока, разделяемая DES и большинством других надежных шифров, открывает его для атак, когда большие объемы данных шифруются одним и тем же ключом.

В ответ на растущее желание заменить DES, NIST анонсировал программу Advanced Encryption Standard (AES) в январе 1997 года. Срок подачи заявок истек в июне 1998 года, и 15 участников представили свои алгоритмы миру в августе на Первой конференции кандидатов AES. NIST проведет вторую конференцию кандидатов AES в Риме в марте следующего года и будет принимать общественные комментарии по алгоритмам до 15 июня 1999 года. Он выберет пять финалистов, в число которых попали: Rijndael, Serpent, Twofish, RC6, MARS, и проведет еще один раунд общественного обсуждения, проведет третью конференцию кандидатов AES примерно в январе 2000 года, а затем выберет в качестве победителя Rijndael. NIST указал несколько критериев проектирования: более длинная длина ключа, больший размер блока, более высокая скорость и большая гибкость. Хотя ни один алгоритм не может быть оптимизирован для всех нужд, NIST сделал Rijndael стандартом симметричного шифрования следующих нескольких десятилетий.

Twofish - это блочный шифр от компании Брюса Шнайера Counterpane. Брюс Шнайер - всемирно известный специалист в области безопасности, которого журнал The Economist называет “гуру безопасности”. В Dr. Dobbs’s Journal Брюс Шнайер говорит: “Джон Келси, Крис Холл, Нильс Фергюсон, Дэвид Вагнер, Дуг Уайтинг и я разработали Twofish, чтобы он был быстрым, гибким и безопасным. Он консервативен — в нем нет

радикально новых идей безопасности или элементов дизайна. И это совершенно бесплатно — нет никаких патентных отчислений за алгоритм, авторских прав на код или лицензионных платежей за что-либо. Мы не подавали заявку на патент на Twofish и не планируем этого делать.”. Да, Twofish открыт к использованию и найдется большое количество криптографов считающих, что Twofish намного безопаснее, чем алгоритм Rijndael, который сегодня является широко используемым алгоритмом и рекомендован АНБ

За исключением названия и использования S-блоков, зависящих от ключа, он имеет мало общего со своим предком Blowfish. Twofish имеет новую архитектуру. У него есть преемник под названием Threefish, используемый в алгоритме хеширования Skein.

### **Общие сведения о Twofish**

Twofish разрабатывался специально с учетом требований и рекомендаций NIST для AES:

- 128-битный блочный симметричный шифр
- Длина ключей 128, 192 и 256 бит
- Отсутствие слабых ключей
- Эффективная программная (в первую очередь на 32-битных процессорах) и аппаратная реализация
- Гибкость (возможность использования дополнительных длин ключа, использование в поточном шифровании, хэш-функциях и т. д.)
- Простота алгоритма - для возможности его эффективного анализа

Twofish - 128-битный блочный шифр, который принимает ключ длины 128, 192, 256 бит. Twofish, являясь симметричным алгоритмом шифрования, то есть использует один ключ как для шифрования данных, так и для их расшифровки. Шифр имеет входное, выходное отбеливание и 16 раундов сети Фейстеля, каждый из которых обладает биективной функцией F, четырех зависящих от ключа S-блоков, фиксированной MDS матрицы, псевдопреобразования Адамара, побитовых поворотов, и тщательно разработанное ключевое расписание.

Отличительными особенностями Twofish являются использование предварительно вычисленных S-блоков, зависящих от ключа, и относительно сложное ключевое расписание.. Одна половина n-битного ключа используется в качестве фактического ключа шифрования, а другая половина n-битного ключа используется для модификации алгоритма шифрования (зависящие от ключа S-блоки). Twofish заимствует некоторые элементы из

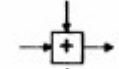

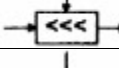
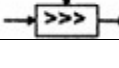
других конструкций, например, преобразование псевдо-Адамара из семейства шифров SAFER.

### Архитектура шифра

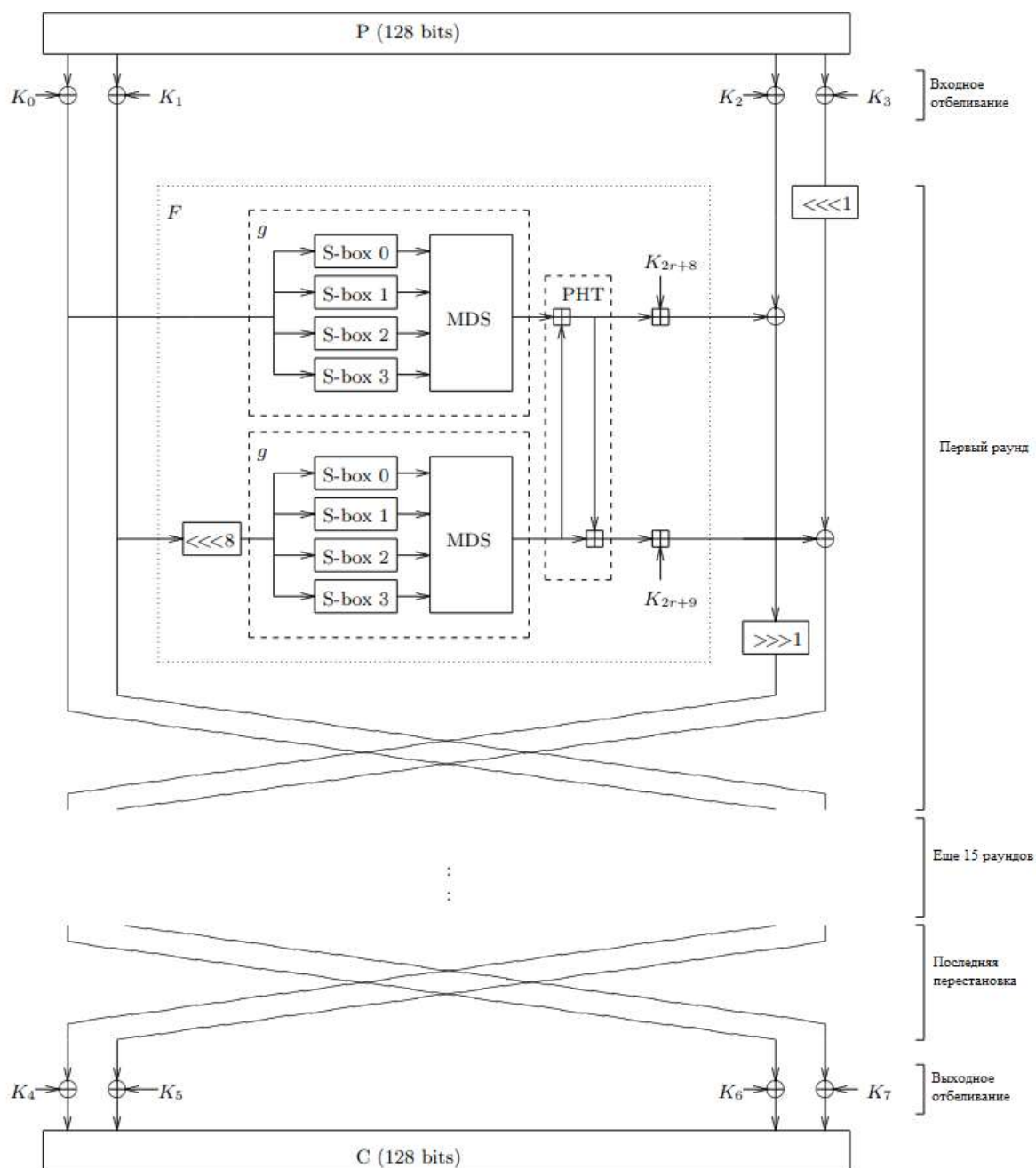
В Twofish используется достаточно сложную структуру, которая сильно усложняет его анализ, что является как плюсом, так и минусом алгоритма. В алгоритме можно выделить несколько основных элементов участвующих в процессе шифрования:

- Отбеливание. Метод до первого и после последнего раундов, при котором над частью ключевого материала и входными блоками открытого текста выполняется операция исключающего «или» (XOR).
- Сеть Фейстеля. Методов построения блочных шифров, состоящий из многократно повторяющихся раундов, на каждом из которых вычисляется функция  $F$  от части шифруемого блока, результат которой складывается с другой частью этого же блока
- S-боксы. Широко распространённый в блочных шифрах метод нелинейной замены бит. В Twofish S-блоки зависят от ключевого материала, принимают 8 бит на входе и возвращают 8 бит на выходе.
- MDS (maximum distance separable) - матрицы разделения на максимальное расстояние. В Twofish используется заранее известная MDS матрица размером 4 на 4 над полем Галуа. Она используется для перемешивания ключа, чтобы гарантировать, что небольшое изменение ключа создаст совершенно новый зашифрованный текст на выходе шифра.
- Псевдо-преобразование Адамара (PHT). Обратимое преобразование битовых строк, заключающиеся во взаимном сложении по модулю двух выходов  $g$ -функций.
- Ключевое расписание. Метод создания множества различных раундовых ключей из одного ключевого материала, которые будут использованы на каждом раунде, процессе отбеливания и определении S-блоков

Помимо всего вышеперечисленного в алгоритме используются следующие операции работы с информацией, представленной в двоичном коде:

Схема	Операция	Уравнение
	Сложение	$Z = X + Y$
	Исключающее ИЛИ	$Z = X \text{ XOR } Y$
	Циклический сдвиг влево	$Z = X \text{ ROL } Y$
	Циклический сдвиг вправо	$Z = X \text{ ROR } Y$

Ниже представлена общая схема алгоритма Twofish:

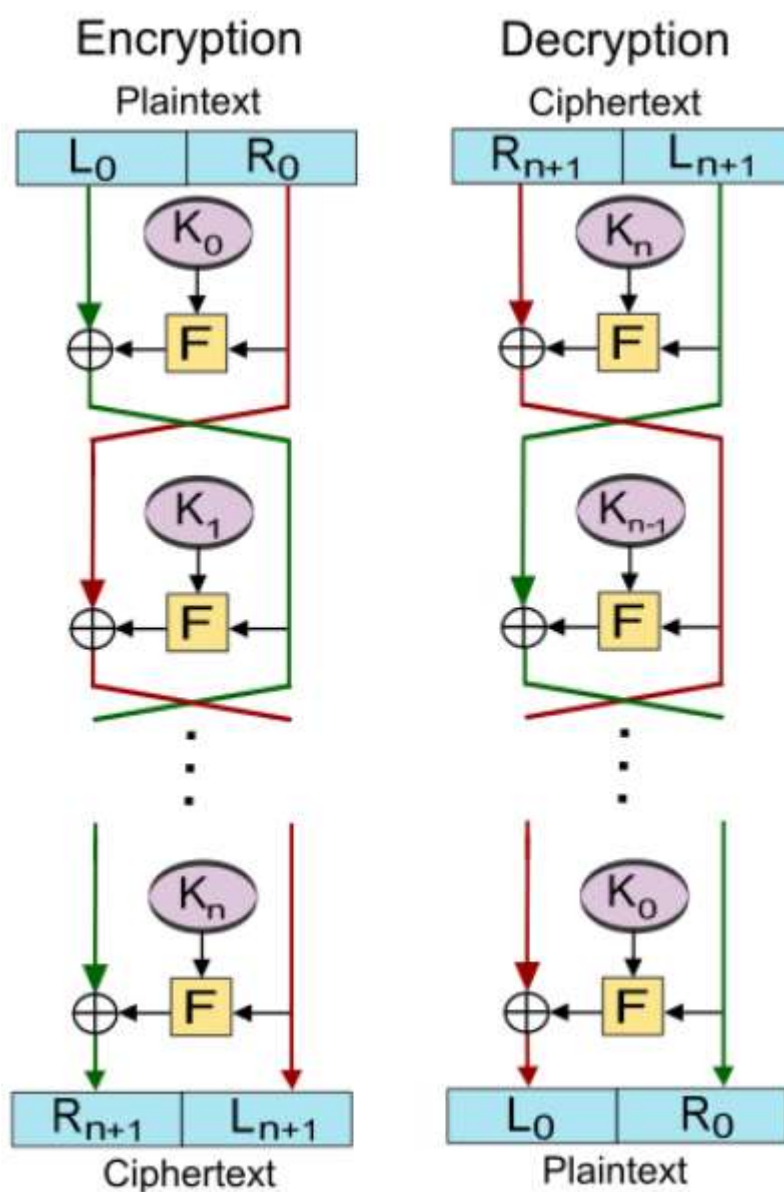


### **Отбеливание**

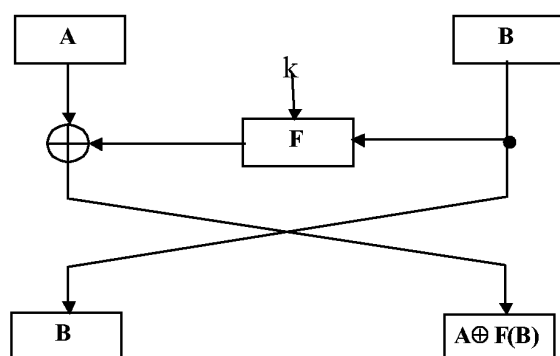
Отбеливание – метод повышения надежности алгоритма шифрования, путем объединения данных с частями ключа. В Twofish оно происходит сразу после получения нового блока открытого текста, то есть перед первым раундом и после последнего раунда сети Фейстеля. На процесс отбеливания поступает 128 бит текста, который заранее поделен на 4 блока по 32 бита. С каждым из этих блоков складываются по модулю 2 ключи  $K_0, K_1, K_2, K_3$  на входном отбеливании и ключи  $K_4, K_5, K_6, K_7$  на выходном отбеливании, которые были заранее посчитаны по тому же алгоритму что и раундовые ключи. Способ их вычисления будет описан в главе Key schedule. Отбеливание существенно увеличивает сложность атаки на шифр, скрывая от атакующего конкретные входные данные для функций  $F$  первого и последнего раундов.

### **Сеть Фейстеля**

Сеть Фейстеля это способ организовать блочный алгоритм шифрования таким образом, что криптограф не должен заботиться о том, чтобы шифр был восстановим. Так же эта методика обеспечивает выполнение требования о многократном использовании ключевого материала и блока открытого текста. В общем варианте сеть Фейстеля имеет следующий вид:



При шифровании в сети Фейстеля блок открытого текста делится на части или ветви. К правой ветви и к раундовому ключу применяется некоторая образующая функция Фейстеля, результат которой накладывается на другую ветвь, а дальше происходит обмен ветвей местами, что означает конец раунда. Структура раунда или как принято называть ячейки Фейстеля схематично в общем виде выглядит следующим образом:





В результате одного раунда сети Фейстеля меняется половина текста. Эти раунды повторяются множество раз, благодаря чему обеспечивается эффект надежного шифра. Особенностью такого способа организации алгоритма является то, что для расшифрования не требуется проходить через функцию Фейстеля в обратную сторону, то есть функция Фейстеля однонаправленно и криптографу не нужно заботиться об обратимости функции  $F$ . Обратимость функции шифрования гарантируется конструкцией ячейки Фейстеля. Для расшифрования шифртекста используется та же самая функция  $F$ . Покажем это.

В  $i$ -ом раунде на вход попадают два блока данных –  $X_{R1}$  и  $X_{L1}$ . В результате работы ячейки Фейстеля на выходе мы получаем

$$X_{L2} = X_{R1}$$

$$X_{R2} = X_{L1} \text{ XOR } F_k(X_{R1})$$

Для восстановления  $X_{R1}$  и  $X_{L1}$  из  $X_{L2}$  и  $X_{R2}$  нужно применить ту же функцию  $F$  с тем же ключом  $k$

$$X_{R1} = X_{L2}$$

$$X_{L1} = X_{R2} \text{ XOR } F_k(X_{L2})$$

В Twofish используется 16 раундовая сеть Фейстеля.

### Структура раунда

На вход раунда поступает блок открытого текста длиной 128 бит, который в последствии делится на 4 ветви  $R_{r,0}$ ,  $R_{r,1}$ ,  $R_{r,2}$ ,  $R_{r,3}$  по 32 бита каждая. Ветви  $R_{r,0}$  и  $R_{r,1}$  идут на вход функции Фейстеля. Результатом является 2 блока по 32 бита, один из которых складывается по модулю 2 с  $R_{r,2}$  и выполняется циклический сдвиг вправо, а второй блок складывается по модулю 2 с  $R_{r,3}$  к которому заранее было применен циклический сдвиг влево.

Алгоритм раунда описывается следующим образом

$$(F_{r,0}, F_{r,1}) = F(R_{r,0}, R_{r,1}, r)$$

$$R_{r+1,0} = \text{ROR}(R_{r,2} \text{ XOR } F_{r,0}, 1)$$

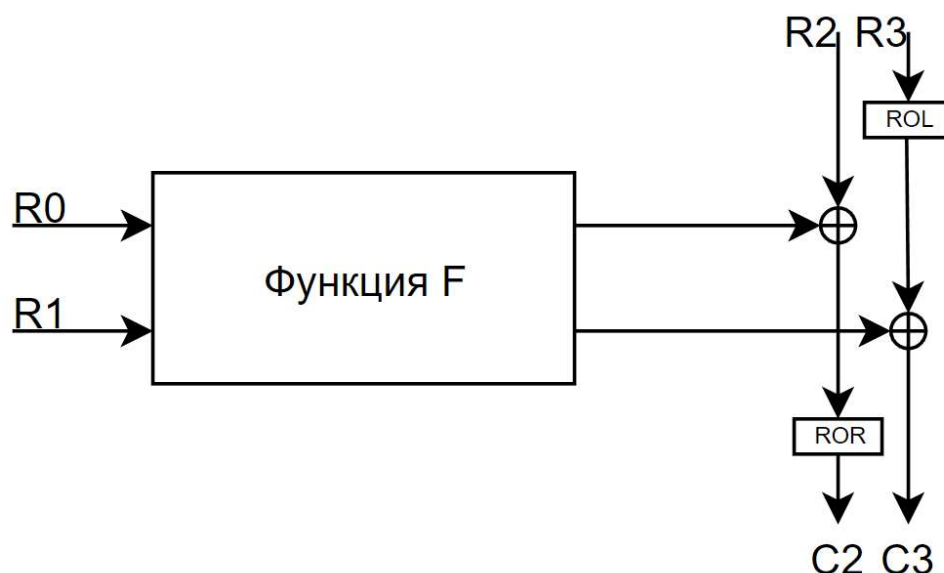
$$R_{r+1,1} = \text{ROL}(R_{r,3}, 1) \text{ XOR } F_{r,1}$$

$$R_{r+1,2} = R_{r,0}$$

$$R_{r+1,3} = R_{r,1}$$

Где  $r = 0, \dots, 15$  – номер раунда.

Схематично алгоритм можно изобразить как показано на схеме ниже



Блоки C2, C3, R0, R1 являются результатом раунда.

### Функция F

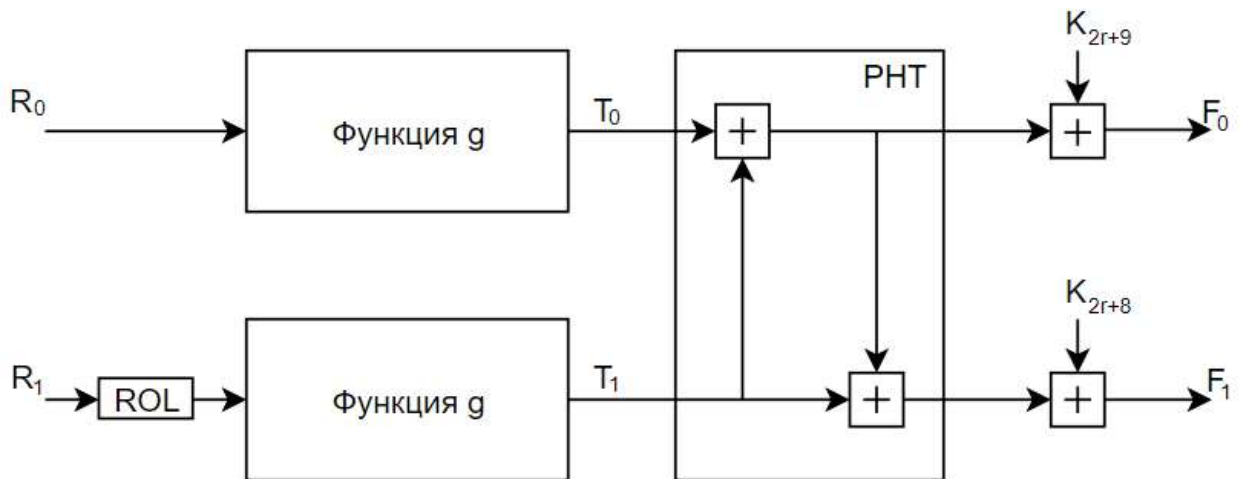
Функция F есть функция Фейстеля. Она получает на вход две ветви по 32 бита и номер раунда  $r$ , а её результатом являются две последовательности по 32 бита, которые складываются с двумя другими ветвями. R0 передается в функцию  $g$  и на выходе получается  $T_0$ , а к R1 сначала применяется операция циклического сдвига влево и затем он уже поступает на вход функции  $g$  для получения  $T_1$ . Затем  $T_0$  и  $T_1$  попадают на вход РНТ(Псевдо-преобразование Адамара) и полученные две последовательности складываются с раундовыми ключами  $K_{2r+8}$  и  $K_{2r+9}$ .

$$T_0 = g(R_0)$$

$$T_1 = g(\text{ROL}(R_1, 8))$$

$$F_0 = (T_0 + T_1 + K_{2r+8}) \bmod 2^{32}$$

$$F_1 = (T_0 + 2T_1 + K_{2r+9}) \bmod 2^{32}$$



### Функция g

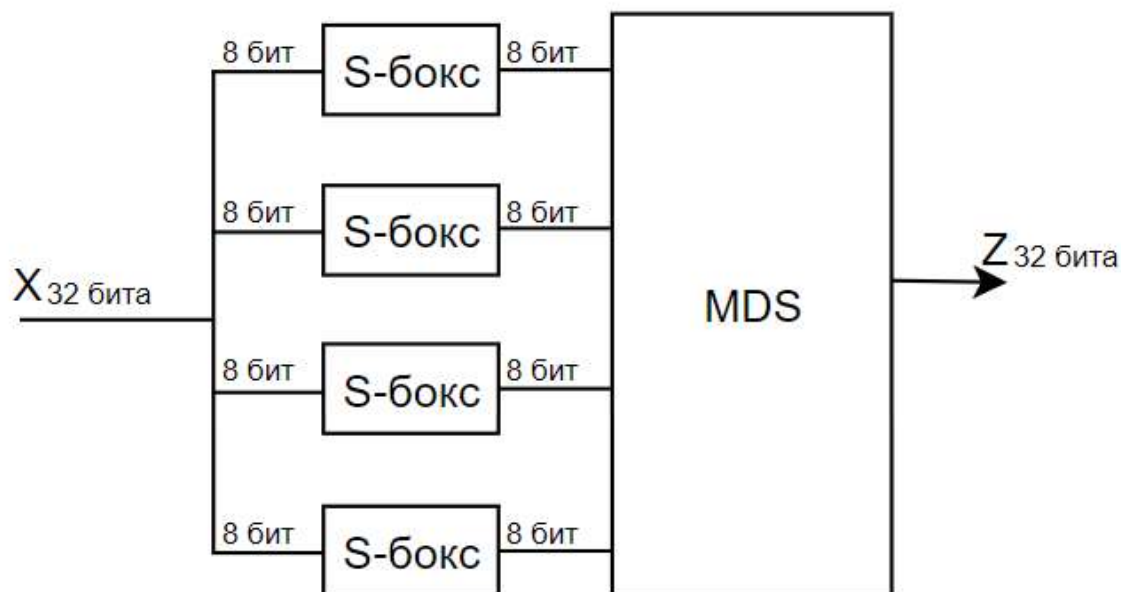
Функция g это сердце алгоритма Twofish. Данная функция применяется дважды в функции F, а также участвует в создании раундовых ключей. На вход функция принимает 32 бита и возвращает 32 бита. Входное значение разбивается на 4 части по 8 бит каждая. Каждая из этих 8-ми битных частей идет в S-бокс. S-бокс принимает и выдает на выходе 8 бит. 4 выхода S-боксов интерпретируются как вектор длины 4 над  $GF(2^8)$  и умножается на матрицу MDS. Результат интерпретируется как 32-ухбитная последовательность и именно она является результатом функции g.

$$x_i = [X/2^{8i}] \bmod 2^8 \quad i = 0, 1, 2, 3$$

$$y_i = s_i[x_i] \quad i = 0, 1, 2, 3$$

$$\begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} \cdot & \cdots & \cdot \\ \cdot & MDS & \cdot \\ \cdot & \dots & \cdot \end{pmatrix} \times \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

$$Z = \sum_{i=0}^3 z_i \times 2^{8i}$$

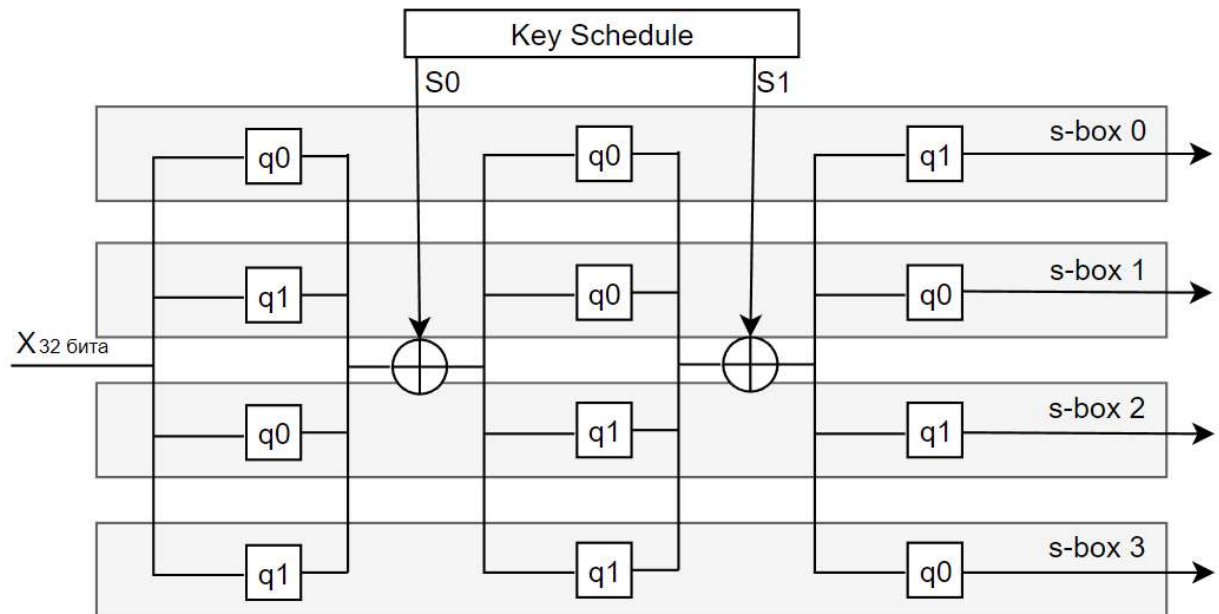


### S-boxes

Блок подстановки или S-box является одним из основных компонентов большинства блочных шифров. S-блок принимает на вход  $n$  бит, преобразует их по определённому алгоритму и возвращает на выходе  $m$  бит. Функция заложенная в S-блок из себя представляет операцию нелинейной замены, управляемую таблицей поиска. Данные таблицы могут генерироваться динамически на основе ключа или заранее быть тщательно подобранными так, чтобы противостоять линейному и дифференциальному криптоанализу.

Twofish использует четыре разных, биективных, зависящих от ключа S-блока размером 8 на 8 бит. Эти S-боксы построены с использованием двух фиксированных 8-на-8-битных перестановок  $q_0$  и  $q_1$  и ключевого материала.

Каждый S-блок имеет набор из трех перестановок составленного из двух функций  $q_0$  и  $q_1$ . Между этими тремя перестановками есть операции XOR, выполняющие сложение по модулю два с ключевым материалом.



Каждая из перестановок  $q_0$  и  $q_1$  имеет одинаковую структуру и отличаются только используемыми таблицами поиска  $t_0 \dots t_3$ .

Перестановки  $q_0$  и  $q_1$ :

$$a_0, b_0 = \left\lfloor \frac{x}{16} \right\rfloor, x \bmod 16$$

$$a_1 = a_0 \text{ XOR } b_0$$

$$b_1 = a_0 \text{ XOR } \text{ROR}_4(b_0, 1) \text{ XOR } 8a_0 \bmod 16$$

$$a_2, b_2 = t_0[a_1], t_1[b_1]$$

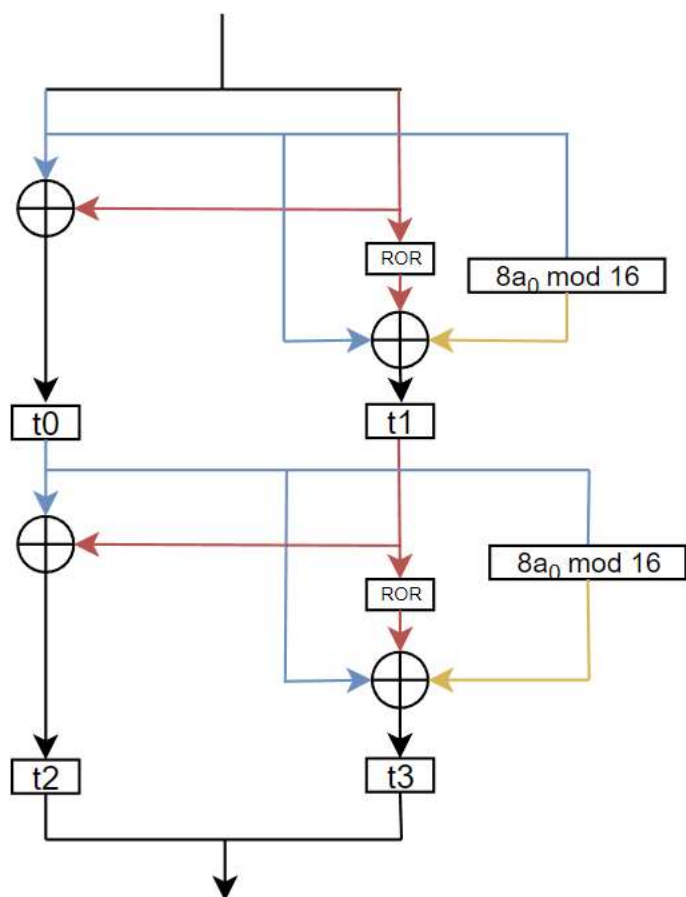
$$a_3 = a_2 \text{ XOR } b_2$$

$$b_3 = a_2 \text{ XOR } \text{ROR}_4(b_2, 1) \text{ XOR } 8a_2 \bmod 16$$

$$a_4, b_4 = t_2[a_3], t_3[b_3]$$

$$y = 16 b_4 + a_4$$

Схематично их можно изобразить следующим образом:



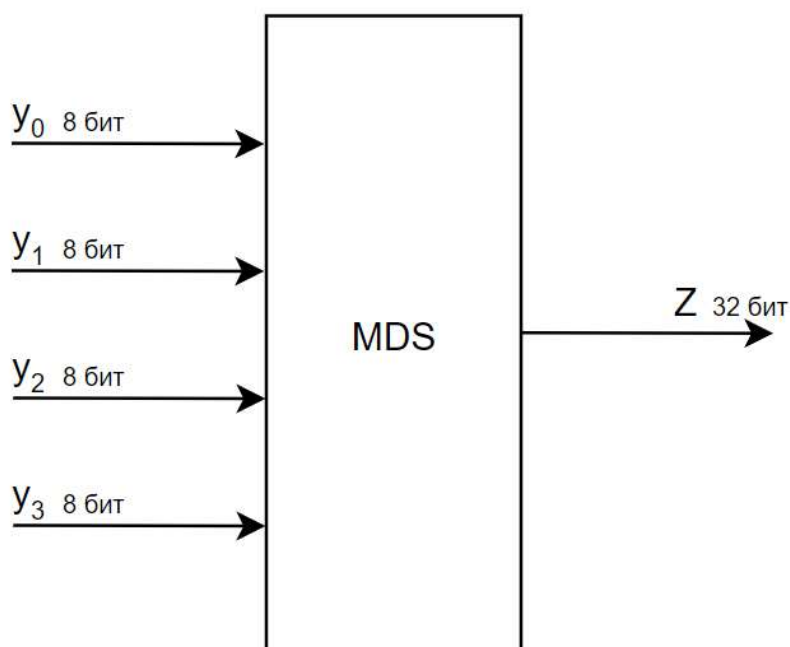
Таблицы поиска для  $q$  – перестановок представлены ниже.

$q_0$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$t_0$	8	1	7	D	6	F	3	2	0	B	5	9	E	C	A	4
$t_1$	E	C	B	8	1	2	3	5	F	4	A	6	7	0	9	D
$t_2$	B	A	5	E	6	D	9	0	C	8	F	3	2	4	7	1
$t_3$	D	7	F	4	1	2	6	E	9	B	3	0	8	5	C	A

$q_1$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$t_0$	2	8	B	D	F	7	6	E	3	1	9	4	0	A	C	5
$t_1$	1	E	2	B	4	C	3	7	6	D	A	5	F	9	0	8
$t_2$	4	C	7	5	1	6	9	A	0	E	D	8	2	B	3	F
$t_3$	B	9	5	1	C	3	D	E	6	4	7	F	2	0	8	A

### MDS-матрицы

Для того, чтобы шифр был стойким к атакам, в процессе шифрования должен создаваться лавинный эффект - свойство схемы, при котором изменение одного бита на входе значительно влияет на все биты на выходе. Лавинный эффект осуществляется за счёт перемешивания и рассеивания. Перемешивание обеспечивает как можно более сложную зависимость ключа и шифрованных данных и используется для того, чтобы усложнить задачу получения информации о ключе. Рассеивание - свойство алгоритма, при котором влияние одного бита открытого текста распространяется на большое количество битов шифротекста, скрывая статистические характеристики текста. MDS матрицы, умножение на которые используется в качестве линейного преобразования в блочных шифрах, является механизмом диффузии.



Это умножение матрицы в Twofish обеспечивают свойство рассеивания для противодействия атакам на основе статистических характеристик. Свойство MDS здесь гарантирует, что количество измененных входных байтов плюс количество измененных выходных байтов составляет не менее пяти. Другими словами, любое изменение в одном входном байте гарантированно изменит все четыре выходных байта, любое изменение в любых двух входных байтах гарантированно изменит по крайней мере три выходных байта и т.д. Существует более  $2^{127}$  таких матриц MDS, но матрица MDS Twofish также тщательно выбирается со свойством сохранения количества измененных байтов, даже после преобразований в функции F.

В Twofish четыре байта, выводимые из четырех S-блоков, представляются как вектор и умножаются на матрицу MDS 4 на 4. Умножение выполняется в поле Галуа  $GF(2^8)$ .

$$\begin{bmatrix} 01 & EF & 5B & 5B \\ 5B & EF & EF & 01 \\ EF & 5B & 01 & EF \\ EF & 01 & EF & 5B \end{bmatrix} \times \begin{bmatrix} y0 \\ y1 \\ y2 \\ y3 \end{bmatrix} = \begin{bmatrix} z0 \\ z1 \\ z2 \\ z3 \end{bmatrix} \parallel \xrightarrow{\quad} Z_{32 \text{ бита}}$$

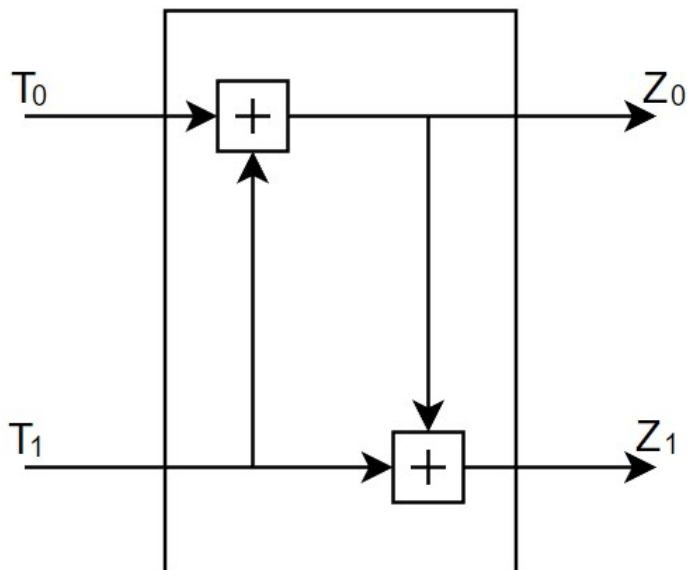
### Псевдо-преобразование Адамара

Псевдо-преобразование Адамара (PHT) - обратимое преобразование битовых строк, используемое в криптографии для обеспечения свойства рассеивания для противодействия атакам на основе статистических характеристик. Получая два входа,  $T_0$  и  $T_1$  в виде 32-разрядных строк, в результате действия псевдо-преобразования Адамара получим две строки  $Z_0$  и  $Z_1$ , значения которых вычисляются по следующим формулам:

$$Z_0 = T_0 + T_1 \bmod 2^{32}$$

$$Z_1 = T_0 + 2T_1 \bmod 2^{32}$$

Схематично псевдо-преобразование Адамара изображается следующим образом



Обратное псевдо-преобразование Адамара получается очень просто

$$T_1 = Z_1 - Z_0 \bmod 2^{32}$$

$$T_0 = 2Z_0 - Z_1 \bmod 2^{32}$$

Псевдо-преобразование Адамара может быть представлено в матричной форме.

Представим  $T$  и  $Z$  в векторной форме

$$T = \begin{bmatrix} T_0 \\ T_1 \end{bmatrix}, Z = \begin{bmatrix} Z_0 \\ Z_1 \end{bmatrix}$$



Псевдо-преобразование Адамара будет равносильно умножению на матрицу

$$H = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} \div \begin{bmatrix} Z_0 \\ Z_1 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} T_0 \\ T_1 \end{bmatrix}$$

по модулю  $2^{32}$ .

### Ключевое расписание

Ключевое расписание – средство, с помощью, которого из битов ключа получают раундовые ключи, которое уже используются на протяжении всего алгоритма шифрования Twofish в различных местах. Благодаря этому на вход шифру требуется меньшее количество ключевых битов, чем требуется для шифра, но при этом на каждом раунде используются уникальные раундовые ключи.

Входной ключ  $M$  имеет длину  $N$  равную 128, 192, 256 бит. Из него строится два набора ключей  $K$  и  $S$ . Набор ключей  $S$  состоит из двух ключей  $S_0$  и  $S_1$ , которые используются в  $S$ -блоках. Набор ключей  $K$  состоит из 40 ключей  $K_0, K_1, \dots, K_{39}$ , каждый из которых имеет длину 32 бита:

- 4 ключа для входного отбеливания
- 4 ключа для выходного отбеливания
- 2 ключа на каждом раунде, что в сумме дает  $2 \cdot 16 = 32$  ключа

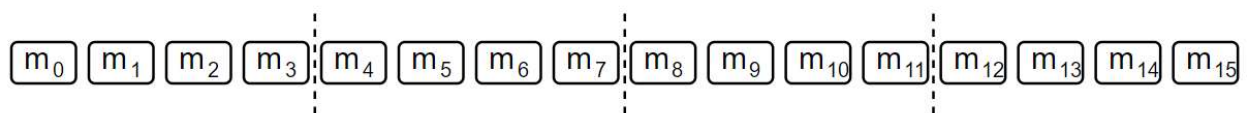
В начале алгоритм находит  $k = N/64$ . Получается что  $M$  состоит из  $8k$  байтов  $m_0, \dots, m_{8k-1}$ . Ключ  $M$  делится на  $2k$  частей по 32 бита следующим образом

$$M_i = \sum_{j=0}^3 m_{(4i+j)} * 2^{8j}, \quad i = 0, \dots, 2k - 1$$

И эти части распределяются по двум векторам

$$M_e = (M_0, M_2, \dots, M_{2k-2})$$

$$M_o = (M_1, M_3, \dots, M_{2k-1})$$



$$M_e = [m_0 \ m_1 \ m_2 \ m_3 \ m_8 \ m_9 \ m_{10} \ m_{11}]$$

$$M_o = [m_4 \ m_5 \ m_6 \ m_7 \ m_{12} \ m_{13} \ m_{14} \ m_{15}]$$

Так же еще нужно найти третий вектор длиной k. Это делается путем взятия байтов ключа в группах по 8, интерпретируя их как вектор на  $GF(2^8)$  и умножая на RS матрицу размера 4 на 8. Результат умножения интерпретируется как 32-битные слова, которые и составляют 3-ий вектор S.

$$RS = \begin{pmatrix} 01 & A4 & 55 & 87 & 5A & 58 & DB & 9E \\ A4 & 56 & 82 & F3 & 1E & C6 & 68 & E5 \\ 02 & A1 & DC & C1 & 47 & AE & 3D & 19 \\ A4 & 55 & 87 & 5A & 58 & DB & 9E & 03 \end{pmatrix}$$

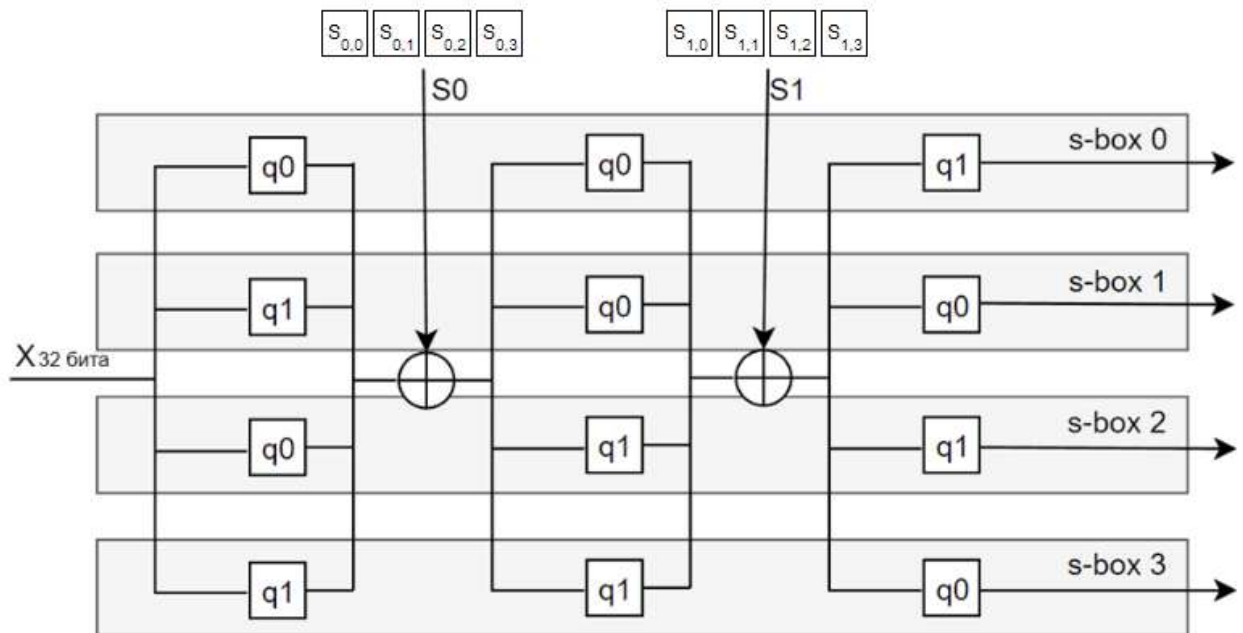
$$\begin{pmatrix} s_{i,0} \\ s_{i,1} \\ s_{i,2} \\ s_{i,3} \end{pmatrix} = \begin{pmatrix} \vdots & \dots & \vdots \\ & RS & \\ \vdots & \dots & \vdots \end{pmatrix} * \begin{pmatrix} m_{8i} \\ m_{8i+1} \\ m_{8i+2} \\ m_{8i+3} \\ m_{8i+4} \\ m_{8i+5} \\ m_{8i+6} \\ m_{8i+7} \end{pmatrix}$$

$$S_i = \sum_{j=0}^3 s_{i,j} \times 2^{8j} \quad \text{для } i = 0, \dots, k-1$$

$$S = (S_{k-1}, S_{k-2}, \dots, S_0)$$

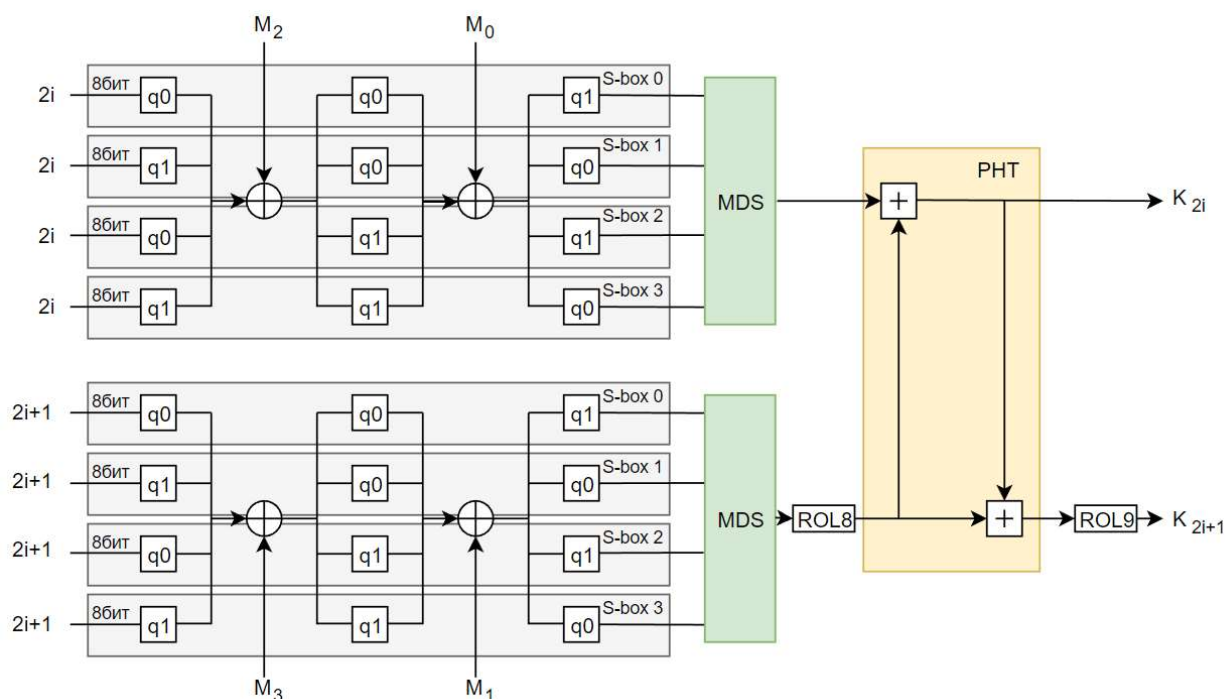
Эти три вектора  $M_e$ ,  $M_o$  и S составляют основу для расписания ключей.

Ключи вектора S используются в функции g



Вектора  $M_e$ ,  $M_o$  поступают на вход функции h, которая генерирует ключи  $K_0, K_1, \dots, K_{39}$ .

Функция  $h$  для  $k = 4$  схематично изображена ниже.



### Достоинства и недостатки

Согласно анализу IEEE, алгоритм Twofish отстает по скорости шифрования текста и изображений по сравнению с сегодняшним стандартом шифрования AES. Но при достаточном увеличении объема оперативной памяти алгоритм Twofish был быстрее для шифрования текста и на одном уровне с AES для шифрования изображений. Это указывает на требовательность к ресурсам у Twofish. Алгоритм Twofish по своей конструкции является сложным и использует 16 раундов независимо от используемой длины ключа. Все это приводит к тому, что в большинстве случаев он медленный и неэффективный. Преимущество AES также в том, что высокопроизводительные современные процессоры x86 и ARM включают аппаратное ускорение для него, что делает его несправедливым по отношению к практически любому другому шифрованию.

На Twofish нет практических атак, даже для сокращенных вариантов. Единственные атаки являются теоретическими, но вычислительно невозможными из-за их большой временной сложности.

Для большинства приложений алгоритм AES, вероятно, является лучшим вариантом, поскольку он достаточно быстрый и безопасный. Но если у вас есть конфиденциальная информация, которую вы хотите защитить, и производительность не является проблемой, алгоритм Twofish – хороший вариант.

## Применение шифра

Любой может использовать Twofish без ограничений, поскольку его создатели не запатентовали его и сделали доступным для общественности. Фактически, это один из немногих шифров, включенных в стандарт OpenPGP (RFC 4880), наиболее широко используемый на сегодняшний день стандарт шифрования электронной почты.

Алгоритм Twofish может использоваться в различных направлениях и широко применяться в системах с разными уровнями требований к безопасности, так как стойкость данного алгоритма достаточно высока, а сам алгоритм не столь популярен, что увеличивает надежность его применения.

Twofish помимо своего прямого назначени – сокрытия информации от неавторизованных лиц с предоставлением в это же время авторизованным пользователям доступа к ней, может быть использован иначе:

- в построении односторонних хеш-функций, диапазон применения которых очень широк: проверка целостности сообщений и файлов, верификация пароля, аутентификационные сообщения, цифровая подпись...
- лежать в основе генератора псевдослучайных чисел.

Продукты, использующие шифрование Twofish:

- PGP Pretty Good Privacy (PGP): компьютерная программа, также библиотека функций, позволяющая выполнять операции шифрования и цифровой подписи сообщений, файлов и другой информации. Эта программа держит в своем арсенале алгоритм Twofish.
- GnuPG: GnuPG - реализация стандарта IETF OpenPGP, позволяет вам шифровать и подписывать ваши данные и сообщения; он оснащен универсальной системой управления ключами, а также модулями доступа ко всем видам каталогов открытых ключей. GnuPG, также известный как GPG, представляет собой инструмент командной строки с функциями для легкой интеграции с другими приложениями. Доступно множество интерфейсных приложений и библиотек.
- VeraCrypt: VeraCrypt - бесплатное программное обеспечение с открытым исходным кодом для создания и поддержки на лету зашифрованного тома (устройства хранения данных). Шифрование на лету означает, что данные автоматически шифруются непосредственно перед сохранением и расшифровываются сразу после загрузки без какого-либо вмешательства пользователя. Никакие данные, хранящиеся на зашифрованном томе, не

могут быть прочитаны (расшифрованы) без использования правильных паролей / файлов ключей или правильных ключей шифрования. Вся файловая система зашифрована

- KeePass: KeePass - это программное обеспечение с открытым исходным кодом для хранения и управления паролями, которое использует Twofish для шифрования паролей своих баз данных.. Он также использует Twofish для создания паролей для своих пользователей.

## Глава 2. Практическая часть

### Реализация алгоритма

В ходе практической работы мною была написана программная реализация алгоритм Twofish на языке C#. Основная функциональная часть представлена ниже.

Поля класса

```
static private readonly byte[,] RS = new byte[,]  
{ {0x01, 0xA4, 0x55, 0x87, 0x5A, 0x58, 0xDB, 0x9E},  
  {0xA4, 0x56, 0x82, 0xF3, 0x1E, 0xC6, 0x68, 0xE5},  
  {0x02, 0xA1, 0xFC, 0xC1, 0x47, 0xAE, 0x3D, 0x19},  
  {0xA4, 0x55, 0x87, 0x5A, 0x58, 0xDB, 0x9E, 0x03} };  
  
static private readonly byte[,] tq0 = new byte[,]  
{ {0x8, 0x1, 0x7, 0xD, 0x6, 0xF, 0x3, 0x2, 0x0, 0xB, 0x5, 0x9, 0xE, 0xC, 0xA, 0x4},  
  {0xE, 0xC, 0xB, 0x8, 0x1, 0x2, 0x3, 0x5, 0xF, 0x4, 0xA, 0x6, 0x7, 0x0, 0x9, 0xD},  
  {0xB, 0xA, 0x5, 0xE, 0x6, 0xD, 0x9, 0x0, 0xC, 0x8, 0xF, 0x3, 0x2, 0x4, 0x7, 0x1},  
  {0xD, 0x7, 0xF, 0x4, 0x1, 0x2, 0x6, 0xE, 0x9, 0xB, 0x3, 0x0, 0x8, 0x5, 0xC, 0xA} };  
  
static private readonly byte[,] tq1 = new byte[,]  
{ {0x2, 0x8, 0xB, 0xD, 0xF, 0x7, 0x6, 0xE, 0x3, 0x1, 0x9, 0x4, 0x0, 0xA, 0xC, 0x5},  
  {0x1, 0xE, 0x2, 0xB, 0x4, 0xC, 0x3, 0x7, 0x6, 0xD, 0xA, 0x5, 0xF, 0x9, 0x0, 0x8},  
  {0x4, 0xC, 0x7, 0x5, 0x1, 0x6, 0x9, 0xA, 0x0, 0xE, 0xD, 0x8, 0x2, 0xB, 0x3, 0xF},  
  {0xB, 0x9, 0x5, 0x1, 0xC, 0x3, 0xD, 0xE, 0x6, 0x4, 0x7, 0xF, 0x2, 0x0, 0x8, 0xA} };  
  
static private readonly List<byte[,]> tq = new List<byte[,]> { tq0, tq1 };  
  
static private readonly byte[,] MDS = new byte[,]  
{ {0x01, 0xEF, 0x5B, 0x5B},  
  {0x5B, 0xEF, 0xEF, 0x01},  
  {0xEF, 0x5B, 0x01, 0xEF},  
  {0xEF, 0x01, 0xEF, 0x5B} };
```

Метод Encrypt принимает на вход открытый текст, делит его на блоки по 16 байт, производит вычисление раундовых ключей, шифрует каждый блок и возвращает на выходе закрытый текст.

```
public byte[] Encrypt(byte[] plaintext)  
{  
    // KeyShedule  
    KeyShedule(out k_keys, out s_keys, ref key);
```

```

List<byte> plaintext_list = plaintext.ToList();
List<byte> ciphertext_list = new List<byte>();
while (plaintext_list.Count%16 != 0)
    plaintext_list.Add(0);
for(int i = 0; i < plaintext_list.Count; i += 16)
{
    ciphertext_list.AddRange(
        Encrypt_block(plaintext_list.GetRange(i,16).ToArray()));
}
return ciphertext_list.ToArray();
}

```

Метод Encrypt\_block принимает на вход блок открытого текста длиной 128 бит, производит входное и выходное отбеливание, шифрование 16 раундовой сетью Фейстеля и выдает на выход блок закрытого текста длиной 128 бит.

```

public byte[] Encrypt_block(byte[] plaintext)
{
    if (plaintext.Length != 16)
        throw new Exception("Something go wrong! " +
            "The block size is not equal to 128");
    if (key == null)
        throw new Exception("Key is null. Use SetKey method.");

    // Разделение открытого текста на 4 ветви
    byte[][] text_branches = new byte[4][];
    for (int i = 0; i < 4; i++)
    {
        text_branches[i] = new byte[4];
        for (int j = 0; j < 4; j++)
        {
            text_branches[i][j] = plaintext[i * 4 + j];
        }
    }

    // Входное отбеливание
    byte[][] keys_input_whitening = new byte[][] { k_keys[0],
                                                    k_keys[1],
                                                    k_keys[2],
                                                    k_keys[3] };
    Whitening(ref text_branches, ref keys_input_whitening);

    // 16 раундов сети Фейстеля
    for (int round = 0; round < 16; round++)
    {
        byte[][] f_result = f_function( text_branches[0],
            text_branches[1], round);

        byte[] c2 = XOR(f_result[0], text_branches[2]);
        c2 = ROR(c2, 1);
        byte[] c3 = text_branches[3];
        c3 = ROL(c3, 1);
        c3 = XOR(f_result[1], c3);

        text_branches = new byte[][] { c2, c3,
            text_branches[0], text_branches[1] };
    }

    //Выходное отбеливание
    byte[][] keys_output_whitening = new byte[][] { k_keys[4],
                                                    k_keys[5],
                                                    k_keys[6],
                                                    k_keys[7] };
    Whitening(ref text_branches, ref keys_output_whitening);
}

```

```

        // Слияние 4 ветвей в конечный закрытый текст
        byte[] ciphertext = new byte[16];
        for (int i = 0; i < 4; i++)
            for (int j = 0; j < 4; j++)
                ciphertext[i * 4 + j] = text_branches[i][j];

        return ciphertext;
    }

```

Метод KeyShedule производит построение ключевого расписания.

```

private void KeyShedule(out byte[][] k_keys, out byte[][] s_keys, ref byte[]
main_key)
{
    int k = main_key.Length / 8; // 2

    // Построение M_even и M_odd
    byte[] m_even = new byte[main_key.Length / 2];
    byte[] m_odd = new byte[main_key.Length / 2];

    for (int i = 0; i < k; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            m_even[i * 4 + j] = main_key[i * 8 + j];
            m_odd[i * 4 + j] = main_key[i * 8 + 4 + j];
        }
    }

    // Построение S ключей
    s_keys = new byte[k][];
    for (int i = 0; i < k; i++)
    {
        byte[,] tmp_matrix = new byte[8, 1];
        for (int j = 0; j < 8; j++)
        {
            tmp_matrix[j, 0] = main_key[i * 8 + j];
        }

        byte[,] rez = MatrixMultiplication(RS, tmp_matrix, 0x4D);
        // x^8 + x^6 + x^3 + x^2 + 1

        s_keys[i] = new byte[4];
        for (int j = 0; j < 4; j++)
        {
            s_keys[i][j] = rez[j, 0];
        }
    }

    // Построение K ключей
    k_keys = h_function(m_even, m_odd);
}

```

Метод h\_function реализует построение 40 K-ключей.

```

private byte[][] h_function(byte[] m_even, byte[] m_odd)
{
    byte[] M0 = m_even.Take(4).ToArray();
    byte[] M1 = m_odd.Take(4).ToArray();
    byte[] M2 = m_even.Skip(4).ToArray();
    byte[] M3 = m_odd.Skip(4).ToArray();

    byte[][] k_keys = new byte[40][];
}

```

```

for(byte i = 0; i < 40; i+=2)
{
    byte[,] key_1_matrix = MatrixMultiplication(
        MDS, h_function_Sboxs(i,M2, M0), 0x69);
    byte[,] key_2_matrix = MatrixMultiplication(
        MDS, h_function_Sboxs((byte)(i+1), M3, M1), 0x69);

    byte[] key_1 = {    key_1_matrix[0, 0],
                        key_1_matrix[1, 0],
                        key_1_matrix[2, 0],
                        key_1_matrix[3, 0] };
    byte[] key_2 = {    key_2_matrix[0, 0],
                        key_2_matrix[1, 0],
                        key_2_matrix[2, 0],
                        key_2_matrix[3, 0] };

    key_2 = ROL(key_2, 8);
    PHT(ref key_1, ref key_2);
    key_2 = ROL(key_2,9);

    k_keys[i] = key_1;
    k_keys[i+1] = key_2;
}

return k_keys;
}

```

Метод `h_function_Sboxs` реализует S-блоки для функции `h`.

```

private byte[,] h_function_Sboxs(byte input, byte[] M1, byte[] M2)
{
    byte[,] output = new byte[4,1];

    output[0, 0] = q1((byte)(q0((byte)(q0(input) ^ M1[0])) ^ M2[0]));
    output[1, 0] = q0((byte)(q0((byte)(q1(input) ^ M1[1])) ^ M2[1]));
    output[2, 0] = q1((byte)(q1((byte)(q0(input) ^ M1[2])) ^ M2[2]));
    output[3, 0] = q0((byte)(q1((byte)(q1(input) ^ M1[3])) ^ M2[3]));

    return output;
}

```

Метод `Whitening` принимает на вход 4 ветви текста по 32 бита и 4 К-ключа по 32 бита и производит побитовую операцию XOR.

```

private void Whitening(ref byte[][] text, ref byte[][] keys)
{
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            text[i][j] = (byte)(text[i][j] ^ keys[i][j]);
}

```

Метод `f_function` реализует функцию Фейстеля, то есть производит вычисление `g`-функций от двух ветвей, РНТ и сложение с раундовыми ключами.

```

private byte[][] f_function(byte[] R1, byte[] R2, int round)

```



```

{
    R2 = ROL(R2,8);

    byte[] Z1 = g_function(R1);
    byte[] Z2 = g_function(R2);

    PHT(ref Z1, ref Z2);

    byte[] F1 = ADD(Z1, k_keys[round * 2 + 8]);
    byte[] F2 = ADD(Z2, k_keys[round * 2 + 9]);

    byte[][] result = new byte[][] {F1,F2};

    return result;
}

```

Метод `g_function` реализует g-функцию, умножая MDS-матрицу на результат S-боксов.

```

private byte[] g_function(byte[] R)
{
    byte[,] Z_matrix = MatrixMultiplication(MDS, Sboxes(R), 0x69);
    byte[] Z = { Z_matrix[0, 0],
                 Z_matrix[1, 0],
                 Z_matrix[2, 0],
                 Z_matrix[3, 0] };

    return Z;
}

```

Метод `Sboxes` реализует производит вычисление S-боксов.

```

private byte[,] Sboxes(byte[] input)
{
    byte[,] output = new byte[4, 1];

    output[0, 0] = q1((byte)(q0((byte)(q0(input[0]) ^ s_keys[0][0])) ^
s_keys[1][0]));
    output[1, 0] = q0((byte)(q0((byte)(q1(input[1]) ^ s_keys[0][1])) ^
s_keys[1][1]));
    output[2, 0] = q1((byte)(q1((byte)(q0(input[2]) ^ s_keys[0][2])) ^
s_keys[1][2]));
    output[3, 0] = q0((byte)(q1((byte)(q1(input[3]) ^ s_keys[0][3])) ^
s_keys[1][3]));

    return output;
}

```

Методы `q0` и `q1` реализует  $q_0$  и  $q_1$  перестановки используя таблицы поиска `tq0` и `tq1`.

```

private byte q0(byte input)
{
    byte a0 = (byte)(input & 15);
    byte b0 = (byte)(input / 16);

    byte a1 = (byte)(a0 ^ b0);
    byte b1 = (byte)(a0 ^ ROR(b0, 1, 4) ^ ((8 * a0) % 16));

    byte a2 = tq0[0, a1];
    byte b2 = tq0[1, b1];
}

```

```

    byte a3 = (byte)(a2 ^ b2);
    byte b3 = (byte)(a2 ^ ROR(b2, 1, 4) ^ ((8 * a2) % 16));

    byte a4 = tq0[2, a3];
    byte b4 = tq0[3, b3];

    return (byte)(16 * b4 + a4);
}

private byte q1(byte input)
{
    byte a0 = (byte)(input & 15);
    byte b0 = (byte)(input / 16);

    byte a1 = (byte)(a0 ^ b0);
    byte b1 = (byte)(a0 ^ ROR(b0, 1, 4) ^ ((8 * a0) % 16));

    byte a2 = tq1[0, a1];
    byte b2 = tq1[1, b1];

    byte a3 = (byte)(a2 ^ b2);
    byte b3 = (byte)(a2 ^ ROR(b2, 1, 4) ^ ((8 * a2) % 16));

    byte a4 = tq1[2, a3];
    byte b4 = tq1[3, b3];

    return (byte)(16 * b4 + a4);
}

```

### Вычисление скоростных характеристик

В данном разделе мной произведены вычисления скоростных характеристик алгоритма Twofish. Для этого были созданы файлы различных размеров: 1 килобайт, 100 килобайт, 10 мегабайт, 1 гигабайт. Данные файлы были зашифрованы однопоточный режим выполнения без сцепления блоков с ключом 128 бит на процессоре AMD Ryzen 5 3600 6-Core Processor с частотой 3.95 GHz. Результаты представлены в таблице ниже.

Размер файла (байт)	Затрачено времени (миллисекунд)	Скорость шифрования (Кбайт/секунда)
1024	3	350
102400	13	7648
10485760	1 066	9 830
1073741824	109 013	9849

Из тестов видно, что скорость шифрования больших файлов, которую можно соотнести со скоростью шифрования потока данных, равна 10 Мбайт/с на одном потоке. В данном процессоре имеется 6 ядер и 12 потоков, что в теории дает то, что скорость на 12 потоках может достичь 120 Мбайт/с.

## **Заключение**

В данной курсовой работе было произведено исследование архитектуры и принципа работы алгоритма шифрования Twofish. В ходе работы было выяснено, что алгоритм имеет сложную структуру с множеством механизмов, которые противостоят линейному и дифференциальному криптоанализу. Алгоритм Twofish удовлетворяет требования к современным системам шифрования и может использоваться, как надежный способ организации безопасности данных. Практические атаки с полным перебором вычислительно невозможны, а на сегодняшний день нет известных слабостей (уязвимостей) алгоритма, которые могли бы повлиять на его вычислительную сложность.

В практической части курсовой работы была написана программная реализация алгоритма на языке C#, а также были выяснены скоростные способности алгоритма. В результате была получена достаточно высокая скорость, которая говорит о том, что алгоритм может быть использован для большинства нужд, но не подходит для высоконагруженных систем, так как алгоритм сложен и был разработан для работы на 32-разрядных процессорах, которые на сегодняшний день не широко используются, а его конкурент AES имеет аппаратную поддержку во многих современных процессорах.

Задачи курсовой работы были выполнены и цель курсовой работы достигнута.

### Список использованных источников

1. Н.Р.Спиричева Алгоритмы блочной криптографии: учебно-методическое пособие / Н. Р. Спиричева. – Екатеринбург : Изд-во Урал. Ун-та, 2013. – 78, [2] с.
2. Twofish: A 128-Bit Block Cipher / Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson – Minneapolis : [б. и.], 2015. – 68 p.
3. P. Preneel, V. Rijmen, and A. Bosselaers: “Principles and Performance of Cryptographic Algorithms” Dr.Dobb’s Journal, v. 23, n. 12, 1998, pp 126 – 131.
4. «Announcing Request for Candidate Algorithm Nominations for the Advanced Encryption Standard (AES)» (англ.). Department of Commerce -- National Institute of Standards and Technology -- Federal Register: September 12, 1997
5. Nechvatal J., Barker E., Bassham L., Burr W., Dworkin M., Foti J., Roback E. «Report on the Development of the Advanced Encryption Standard (AES)» (англ.). -- National Institute of Standards and Technology.
6. N. Ferguson, J. Kelsey, B. Schneier, D. Whiting «A Twofish Retreat: Related-Key Attacks Against Reduced-Round Twofish» (англ.) Twofish Technical Report #6, February 14, 2000
7. Fauzan Mirza, Sean Murphy «An Observation on the Key Schedule of Twofish» (англ.) -- Information Security Group, Royal Holloway, University of London -- January 26, 1999
8. ГОСТ 34.13-2018 «Информационная технология. Криптографическая защита информации. Режимы работы блочных шифров»
9. Гатченко Н.А., Исаев А.С., Яковлев А.Д. «Криптографическая защита информации» – СПб: НИУ ИТМО, 2012 – 142 с.
10. Основы криптографии: учебное пособие / В. И. Коржик, В. А. Яковлев. – СПб., ИЦ Интермедия, 2016 – 296 с. : илл.
11. Шнайер, Б. Прикладная криптография / Б. Шнайер. – М. : Триумф, 2002.
12. Защита информации. Блочные шифры. — URL:  
<https://www.youtube.com/watch?v=PbDOmI1iM64&t=725s>

13. Pseudo-Hadamard transform. — URL: [https://en.wikipedia.org/wiki/Pseudo-Hadamard\\_transform](https://en.wikipedia.org/wiki/Pseudo-Hadamard_transform)
14. Feistel cipher. — URL: [https://en.wikipedia.org/wiki/Feistel\\_cipher](https://en.wikipedia.org/wiki/Feistel_cipher)
15. Twofish vs AES Encryption. — URL: <https://cloudstorageinfo.org/twofish-vs-aes-encryption>
16. Twofish. — URL: <https://www.techtarget.com/searchsecurity/definition/Twofish>