**<http://metod.vt.tpu.ru/lab/huffman/theory.html>**

**https://neerc.ifmo.ru/wiki/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC\_%D0%A5%D0%B0%D1%84%D1%84%D0%BC%D0%B0%D0%BD%D0%B0**

**Лекция 6. Сжатие данных. Принципы сжатия данных и их применимость. Типы скремблеров.**

Методы сжатия данных имеют достаточно длинную историю развития, которая началась задолго до появления первого компьютера. Все алгоритмы сжатия оперируют входным потоком информации, минимальной единицей которой является бит, а максимальной - несколько бит, байт или несколько байт. Целью процесса сжатия, как правило, есть получение более компактного выходного потока информационных единиц из некоторого изначально некомпактного входного потока при помощи некоторого их преобразования. Основными техническими характеристиками процессов сжатия и результатов их работы являются:

- степень сжатия (compress rating) или отношение (ratio) объемов исходного и результирующего потоков;

- скорость сжатия - время, затрачиваемое на сжатие некоторого объема информации входного потока, до получения из него эквивалентного выходного потока;

- качество сжатия - величина, показывающая на сколько сильно упакован выходной поток, при помощи применения к нему повторного сжатия по этому же или иному алгоритму.

Существует несколько различных подходов к проблеме сжатия информации. Одни имеют весьма сложную теоретическую математическую базу, другие основаны на свойствах информационного потока и алгоритмически достаточно просты. Любой способ подход и алгоритм, реализующий сжатие или компрессию данных, предназначен для снижения объема выходного потока информации в битах при помощи ее обратимого или необратимого преобразования. Поэтому, прежде всего, по критерию, связанному с характером или форматом данных, все способы сжатия можно разделить на две категории: обратимое и необратимое сжатие.

Под необратимым сжатием подразумевают такое преобразование входного потока данных, при котором выходной поток, основанный на определенном формате информации, представляет, с некоторой точки зрения, достаточно похожий по внешним характеристикам на входной поток объект, однако отличается от него объемом. Степень сходства входного и выходного потоков определяется степенью соответствия некоторых свойств объекта (т.е. сжатой и несжатой информации, в соответствии с некоторым определенным форматом данных), представляемого данным потоком информации. Такие подходы и алгоритмы используются для сжатия, например, данных растровых графических файлов с низкой степенью повторяемости байтов в потоке. При таком подходе используется свойство структуры формата графического файла и возможность представить графическую картинку приблизительно схожую по качеству отображения (для восприятия человеческим глазом) несколькими (а точнее n) способами. Поэтому, кроме степени или величины сжатия, в таких алгоритмах возникает понятие качества, т.к. исходное изображение в процессе сжатия изменяется, то под качеством можно понимать степень соответствия исходного и результирующего изображения, оцениваемая субъективно, исходя из формата информации. Для графических файлов такое соответствие определяется визуально, хотя имеются и соответствующие интеллектуальные алгоритмы и программы. Необратимое сжатие невозможно применять в областях, в которых необходимо иметь точное соответствие информационной структуры входного и выходного потоков. Данный подход реализован в популярных форматах представления видео и фото информации, известных как [JPEG](http://www.compression.ru/arctest/descript/rastr-comp.htm) и JFIF алгоритмы и JPG и JIF форматы файлов.

Обратимое сжатие всегда приводит к снижению объема выходного потока информации без изменения его информативности, т.е. - без потери информационной структуры. Более того, из выходного потока, при помощи восстанавливающего или декомпрессирующего алгоритма, можно получить входной, а процесс восстановления называется декомпрессией или распаковкой, и только после процесса распаковки данные пригодны для обработки в соответствии с их внутренним форматом.

В обратимых алгоритмах кодирование как процесс можно рассматривать со статистической точки зрения, что еще более полезно, не только для построения алгоритмов сжатия, но и для оценки их эффективности. Для всех обратимых алгоритмов существует понятие стоимости кодирования. Под стоимостью кодирования понимается средняя длина кодового слова в битах. Избыточность кодирования равна разности между стоимостью и энтропией кодирования, а хороший алгоритм сжатия всегда должен минимизировать избыточность (напомним, что под энтропией информации понимают меру ее неупорядоченности.). Фундаментальная теорема Шеннона о кодировании информации говорит о том, что "стоимость кодирования всегда не меньше энтропии источника, хотя может быть сколь угодно близка к ней". Поэтому, для любого алгоритма, всегда имеется некоторый предел степени сжатия, определяемый энтропией входного потока.

Перейдем теперь непосредственно к алгоритмическим особенностям обратимых алгоритмов и рассмотрим важнейшие теоретические подходы к сжатию данных, связанные с реализацией кодирующих систем и способы сжатия информации.

##### **Процесс кодирования и его методы**

Под кодированием обычно понимают обработку потока символов (в нашем случае байтов или полубайтов) в некотором алфавите, причем частоты появления символов в потоке различны. Целью кодирования является преобразование этого потока в поток бит минимальной длины, что достигается уменьшением энтропии входного потока путем учета частот символов. Длина кода, представляющего символы из алфавита потока должна быть пропорциональна объему информации входного потока, а длина символов потока в битах может быть не кратна 8 и даже переменной. Если распределение вероятностей частот появления символов из алфавита входного потока известно, то можно построить модель оптимального кодирования. Однако, ввиду существования огромного числа различных форматов файлов задача значительно усложняется т.к. распределение частот символов данных заранее неизвестно. В таком случае, в общем виде, используются два подхода.

Первый заключается в просмотре входного потока и построении кодирования на основании собранной статистики (при этом требуется два прохода по файлу - один для просмотра и сбора статистической информации, второй - для кодирования, что несколько ограничивает сферу применения таких алгоритмов, т.к., таким образом, исключается возможность однопроходного кодирования "на лету", применяемого в телекоммуникационных системах, где и объем данных, подчас, не известен, а их повторная передача или разбор может занять неоправданно много времени). В таком случае, в выходной поток записывается статистическая схема использованного кодирования. Данный метод известен как статическое кодирование **Хаффмена** [Huffman].

Второй метод - [метод адаптивного кодирования](http://www.compression.ru/arctest/descript/superadapt.htm) (adaptive coder method). Его общий принцип состоит в том, чтобы менять схему кодирования в зависимости от характера изменений входного потока. Такой подход имеет однопроходный алгоритм и не требует сохранения информации об использованном кодировании в явном виде. Адаптивное кодирование может дать большую степень сжатия, по сравнению со статическим, поскольку более полно учитываются изменения частот входного потока. Данный метод известен как динамическое кодирование Хаффмена [Huffman], [Gallager], [Knuth], [Vitter].

В статическом кодировании Хаффмена входным символам (цепочкам битов различной длины) ставятся в соответствие цепочки битов, также, переменной длины - их коды. Длина кода каждого символа берется пропорциональной двоичному логарифму его частоты, взятому с обратным знаком. А общий набор всех встретившихся различных символов составляет алфавит потока. Это кодирование является префиксным, что позволяет легко его декодировать результативный поток, т.к., при префиксном кодировании, код любого символа не является префиксом кода никакого другого символа - алфавит уникален.

**Пример:** Пусть входной алфавит состоит из четырех символов: a, b, c, d, частоты которых в входном потоке равны, соответственно, 1/2, 1/4, 1/8, 1/8. Кодирование Хаффмена для этого алфавита задается следующей таблицей:

Например, кодом цепочки abaaacb, представленной на входе как 00 01 00 00 00 10 01, будет 0 10 0 0 0 110 10, соответственно - 14 бит на входе дали 11 бит на выходе. Кодирование по Хаффмену обычно строится и хранится в виде двоичного дерева, в "листьях" которого находятся символы, а на "ветвях" - цифры 0 или 1. Тогда уникальным кодом символа является путь от корня дерева к этому символу, по которому все 0 и 1 "собираются" в одну уникальную последовательность.

При использовании адаптивного кодирования Хаффмена усложнение алгоритма состоит в необходимости постоянной корректировки дерева и кодов символов основного алфавита в соответствии с изменяющейся статистикой входного потока.

Методы Хаффмена дают достаточно высокую скорость и умеренно хорошее качество сжатия. Эти алгоритмы давно известны и широко применяется как в программных (всевозможные компрессоры, архиваторы и программы резервного копирования файлов и дисков), так и в аппаратных (системы сжатия "прошитые" в модемы и факсы, сканеры) реализациях.

Однако, кодирование Хаффмена имеет минимальную избыточность при условии, что каждый символ кодируется в алфавите кода символа отдельной цепочкой из двух бит - {0, 1}. Основным же недостатком данного метода является зависимость степени сжатия от близости вероятностей символов к 2 в некоторой отрицательной степени, что связано с тем, что каждый символ кодируется целым числом бит. Так при кодировании потока с двухсимвольным алфавитом сжатие всегда отсутствует, т.к. несмотря на различные вероятности появления символов во входном потоке алгоритм фактически сводит их до 1/2.

Данная проблема, как правило, решается путем введения в алфавит входного потока новых символов вида "ab", "abc",. . . и т.п., где a, b, c - символы первичного исходного алфавита. Такой процесс называется сегментацией или блокировкой входного потока. Однако, сегментация не позволяет полностью избавиться от потерь в сжатии (они лишь уменьшаются пропорционально размеру блока), но приводит к резкому росту размеров дерева кодирования, и, соответственно, длине кода символов вторичных алфавитов. Так, если, например, символами входного алфавита являются байты со значениями от 0 до 255, то при блокировании по два символа мы получаем 65536 символов (различных комбинаций) и столько же листьев дерева кодирования, а при блокировании по три - 16777216! Конечно, при таком усложнении, соответственно возрастут требования и к памяти и ко времени построения дерева, а при адаптивном кодировании - и ко времени обновления дерева, что приведет к резкому увеличению времени сжатия. Напротив, в среднем, потери составят 1/2 бита на символ при отсутствии сегментации, и 1/4 или 1/6 бита соответственно при ее наличии, для блоков длиной 2 и 3 бита.

##### **Арифметическое кодирование**

##### Совершенно иное решение предлагает т.н. [арифметическое кодирование](http://www.compression.ru/arctest/descript/arithm.htm) [Witten]. Арифметическое кодирование является методом, позволяющим упаковывать символы входного алфавита без потерь при условии, что известно распределение частот этих символов и является наиболее оптимальным, т.к. достигается теоретическая граница степени сжатия.

Предполагаемая требуемая последовательность символов, при сжатии методом арифметического кодирования, рассматривается как некоторая двоичная дробь из интервала [0, 1). Результат сжатия представляется, как последовательность двоичных цифр из записи этой дроби. Идея метода состоит в следующем: исходный текст рассматривается как запись этой дроби, где каждый входной символ является "цифрой" с весом, пропорциональным вероятности его появления. Этим объясняется интервал, соответствующий минимальной и максимальной вероятностям появления символа в потоке. Поясним работу метода на примере:

Пусть алфавит состоит из двух символов: a и b с вероятностями соответственно 3/4 и 1/4. К ак уже говорилось выше, кодирование Хаффмена не может упаковывать слова в данном алфавите, т.к. не справляется без сегментации с двухсимвольным алфавитом.

Рассмотрим наш интервал вероятностей [0, 1). Разобьем его на части, длина которых пропорциональна вероятностям символов. В нашем случае это [0, 3/4) и [3/4,1). Суть алгоритма в следующем: каждому слову во входном алфавите соответствует некоторый подинтервал из интервала [0, 1) а пустому слову соответствует весь интервал [0, 1). После получения каждого следующего символа интервал уменьшается с выбором той его части, которая соответствует новому символу. Кодом цепочки является интервал, выделенный после обработки всех ее символов, точнее, двоичная запись любой точки из этого интервала, а длина полученного интервала пропорциональна вероятности появления кодируемой цепочки.

Применим данный алгоритм для цепочки "aaba":

В качестве кода можно взять любое число из интервала, полученного на шаге 4, например, 0.1.

Алгоритм декодирования работает синхронно с кодирующим: начав с интервала [0, 1), он последовательно определяет символы входной цепочки. В частности, в нашем случае он вначале разделит (пропорционально частотам символов) интервал [0, 1) на [0, 0.11) и [0.11, 1). Поскольку число 0.0111 (код цепочки "aaba") находится в первом из них, можно получить первый символ: "a". Затем делим первый подинтервал [0, 0.11) на [0, 0.1001) и [0.1001, 0.1100) (пропорционально частотам символов). Опять выбираем первый, так как 0 " 0.0111 " 0.1001. Продолжая этот процесс, мы однозначно декодируем все четыре символа. Для того, чтобы декодирующий алгоритм мог определить конец цепочки, мы можем либо передавать ее длину отдельно, либо добавить к алфавиту дополнительный уникальный символ - "конец цепочки".

При разработке этого метода возникают две проблемы: во-первых, необходима арифметика с плавающей точкой, теоретически, неограниченной точности, и, во-вторых, - результат кодирования становится известен лишь при окончании входного потока. Однако, дальнейшие исследования показывают [Rubin], что можно практически без потерь обойтись целочисленной арифметикой небольшой точности (16-32 разряда), а также добиться инкрементальной работы алгоритма: цифры кода могут выдаваться последовательно по мере чтения входного потока при ограничении числа символов входной цепочки каким либо разумным числом.

##### **Модели входного потока**

Кодирование представляет собой лишь часть процесса упаковки. Как было показано, арифметическое кодирование имеет минимальную избыточность при заданном распределении символов входного потока. Но какой алфавит выбрать и каким соответствующим распределением воспользоваться? Ответы на эти вопросы дает построение модели входного потока, представляющей собой некоторый способ определения возможного распределения вероятностей появления каждого очередного символа в потоке. Каждого, поскольку статические модели (в которых распределение принимается неизменным), в большинстве случаев, не дают максимального качества сжатия. Гораздо больший интерес представляют так называемые адаптивные модели, учитывающие текущий контекст потока. Такие модели позволяют строить быстрые однопроходные алгоритмы сжатия, не требующие априорных знаний о входном потоке данных и строящие распределение "на лету". В отдельную группу выделяют также класс "локально адаптивных" алгоритмов, отдающих при построении распределения предпочтение некоторым особенным, например, последним поступившим символам.

Возможны различные подходы к этой проблеме: простейший из них - сбор статистики появления каждого символа независимо от других (моделирование источником Бернулли, при котором вероятность появления последующего символа не зависит от того, какие символы встретились перед ним). Возможно, также и использование марковских моделей: сбор статистики появления каждого символа в которых производится с учетом некоторого количества предыдущих появлявшихся символов (в марковском источнике первого порядка вероятность появления символа зависит только от одного последнего символа, второго - от двух и т. д.). Марковские модели могут давать более точную картину источника, однако число состояний в них больше, соответственно большим будет объем хранимых таблиц частот. Кроме того, при использовании кодирования Хаффмена они могут даже ухудшить качество сжатия, поскольку порождаемые ими вероятности обычно хуже приближаются степенями 1/2.

##### **Кодирование сортировкой**

Здесь нельзя не упомянуть простой и достаточно эффективный метод кодирования источника с неизвестным распределением частот, известный как сжатие при помощи "стопки книг" или как сжатие сортировкой или хешированием. Метод был впервые открыт и исследован Рябко в 1980г., а затем переоткрыт Бентли, Слейтером, Тарьяном и Веи в 1986г. Идея метода состоит в следующем: пусть алфавит источника состоит из N символов с номерами 1, 2,..., N. Кодирующий алгоритм сохраняет последовательность символов, представляющую собой некоторую перестановку символов в последовательности первичного входного алфавита. При поступлении на вход некоторого символа c, имеющего в этой переставленной последовательности номер i, кодирующий алгоритм записывает код этого символа (например, монотонный код). Затем поступивший символ переставляется в начало последовательности и номера всех символов, стоящих перед c, увеличиваются на 1. Таким образом, наиболее часто встречающиеся символы будут переходить в начало списка и иметь более короткие коды, что в свою очередь снизит объем выходного потока при их записи в качестве символов выходного потока.

##### **Двухступенчатое кодирование. Алгоритм Лемпеля-Зива**

Все рассмотренные выше методы и модели кодирования предполагали в качестве входных данных цепочки символов (тексты) в некотором конечном алфавите. При этом оставался открытым вопрос о связи этого входного алфавита кодирующего алгоритма с данными, подлежащими упаковке (обычно также представленными в виде цепочек в алфавите (при байтовой организации обычно состоящем из 256 различных символов - значений байт).

В простейшем случае для кодирования в качестве входного алфавита можно использовать именно эти символы (байты) входного потока. Именно так работает метод squashing программы [PKPAK](http://www.compression.ru/arctest/self/pkpak.htm) (использовано статическое кодирование Хаффмена и двухпроходный алгоритм). Степень сжатия при этом относительно невелика - для текстовых файлов порядка 50%. Гораздо большей степени сжатия можно добиться при выделении из входного потока повторяющихся цепочек - блоков и кодирования ссылок на эти цепочки с построением хеш таблиц от первого до n-го уровня.

Метод, о котором и пойдет речь, принадлежит Лемпелю и Зиву, и обычно называется [LZ](http://www.compression.ru/arctest/descript/lzwcomp.htm)-compression. Суть его состоит в следующем: упаковщик постоянно хранит некоторое количество последних обработанных символов в буфере. По мере обработки входного потока вновь поступившие символы попадают в конец буфера, сдвигая предшествующие символы и вытесняя самые старые. Размеры этого буфера, называемого также скользящим словарем (sliding dictionary), варьируются в разных реализациях кодирующих систем. Экспериментальным путем установлено, что программа [LHArc](http://www.compression.ru/arctest/self/lharc.htm) использует 4-килобайтный буфер, [LHA](http://www.compression.ru/arctest/self/lha.htm) и [PkZip](http://www.compression.ru/arctest/self/pkzip.htm) - 8-ми, а [ARJ](http://www.compression.ru/arctest/self/arj.htm) - 16-килобайтный.

Затем, после построения хеш таблиц алгоритм выделяет (путем поиска в словаре) самую длинную начальную подстроку входного потока, совпадающую с одной из подстрок в словаре, и выдает на выход пару (length, distance), где length - длина найденной в словаре подстроки, а distance - расстояние от нее до входной подстроки (то есть фактически индекс подстроки в буфере, вычтенный из его размера). В случае, если такая подстрока не найдена, в выходной поток просто копируется очередной символ входного потока.

В первоначальной версии алгоритма предлагалось использовать простейший поиск по всему словарю. Время сжатия при такой реализации было пропорционально произведению длины входного потока на размер буфера, что совсем непригодно для практического использования. Однако, в дальнейшем, было предложено использовать двоичное дерево и хеширование для быстрого поиска в словаре, что позволило на порядок поднять скорость работы алгоритма.

Таким образом, алгоритм Лемпеля-Зива преобразует один поток исходных символов в два параллельных потока длин и индексов в таблице (length + distance). Очевидно, что эти потоки являются потоками символов с двумя новыми алфавитами, и к ним можно применить один из упоминавшихся выше методов (RLE, кодирование Хаффмена или арифметическое кодирование).

Так мы приходим к схеме двухступенчатого кодирования - наиболее эффективной из практически используемых в настоящее время. При реализации этого метода необходимо добиться согласованного вывода обоих потоков в один файл. Эта проблема обычно решается путем поочередной записи кодов символов из обоих потоков.

**Типы скремблеров**

## **Цели скремблирования**

Применительно к телекоммуникационным системам скремблирование повышает надежность синхронизации устройств, подключенных к линии связи (обеспечивает надежное выделение тактовой частоты непосредственно из принимаемого сигнала), и уменьшает уровень помех, излучаемых на соседние линии многожильного кабеля. Другая область применения скремблеров — защита передаваемой информации от несанкционированного доступа.

Для алгоритмов скремблирования исключительно важны скорость работы и случайный характер последовательности, чтобы его нельзя было восстановить в случае перехвата противником. Процесс скремблирования может включать в себя добавление определенных компонент к исходному сигналу либо изменение важных частей сигнала для того, чтобы усложнить восстановление вида исходного сигнала либо для придания сигналу определенных статистических свойств.

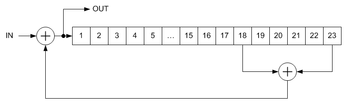
Скремблеры применяются в [телефонных сетях общего пользования](http://yavix.ru/%7B+%28%3E%D1%81%D0%B0%D0%B9%D1%82%3C=%3E%5b%3Ehttps:%E2%95%B1%E2%95%B1ru.wikipedia.org%20%E2%95%B1wiki%E2%95%B1%D0%A2%D0%B5%D0%BB%D0%B5%D1%84%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C_%D0%BE%D0%B1%D1%89%D0%B5%D0%B3%D0%BE_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F%3C%5d%3C%29+%7D) , [спутниковой](http://yavix.ru/%7B+%28%3E%D1%81%D0%B0%D0%B9%D1%82%3C=%3E%5b%3Ehttps:%E2%95%B1%E2%95%B1ru.wikipedia.org%20%E2%95%B1wiki%E2%95%B1%D0%A1%D0%BF%D1%83%D1%82%D0%BD%D0%B8%D0%BA%D0%BE%D0%B2%D0%B0%D1%8F_%D1%81%D0%B2%D1%8F%D0%B7%D1%8C%3C%5d%3C%29+%7D) и [радиорелейной связи](http://yavix.ru/%7B+%28%3E%D1%81%D0%B0%D0%B9%D1%82%3C=%3E%5b%3Ehttps:%E2%95%B1%E2%95%B1ru.wikipedia.org%20%E2%95%B1wiki%E2%95%B1%D0%A0%D0%B0%D0%B4%D0%B8%D0%BE%D1%80%D0%B5%D0%BB%D0%B5%D0%B9%D0%BD%D0%B0%D1%8F_%D1%81%D0%B2%D1%8F%D0%B7%D1%8C%3C%5d%3C%29+%7D) , [цифровом телевидении](http://yavix.ru/%7B+%28%3E%D1%81%D0%B0%D0%B9%D1%82%3C=%3E%5b%3Ehttps:%E2%95%B1%E2%95%B1ru.wikipedia.org%20%E2%95%B1wiki%E2%95%B1%D0%A6%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%BE%D0%B5_%D1%82%D0%B5%D0%BB%D0%B5%D0%B2%D0%B8%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5%3C%5d%3C%29+%7D) , а также для защиты лазерных дисков от копирования.

Обычно скремблирование осуществляется на последнем этапе цифровой обработки непосредственно перед модуляцией.

## Типы скремблеров

* Самосинхронизирующиеся скремблеры (СС)
* Аддитивные скремблеры (с установкой)

### Самосинхронизирующиеся скремблеры

[](http://yavix.ru/%7b+(%3e%D1%81%D0%B0%D0%B9%D1%82%3c=%3e%5b%3ehttps:%E2%95%B1%E2%95%B1ru.wikipedia.org%20%E2%95%B1wiki%E2%95%B1%D0%A4%D0%B0%D0%B9%D0%BB:Scrambler_randomizer_multiplicative_scrambler.png%3c%5d%3c)+%7d)

Скремблер самосинхронизирующийся

Основной частью скремблера является [генератор псевдослучайной последовательности (ПСП)](http://yavix.ru/%7B+%28%3E%D1%81%D0%B0%D0%B9%D1%82%3C=%3E%5b%3Ehttps:%E2%95%B1%E2%95%B1ru.wikipedia.org%20%E2%95%B1wiki%E2%95%B1%D0%93%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80_%D0%BF%D1%81%D0%B5%D0%B2%D0%B4%D0%BE%D1%81%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D1%8B%D1%85_%D1%87%D0%B8%D1%81%D0%B5%D0%BB%3C%5d%3C%29+%7D) в виде [линейного n-каскадного регистра с обратными связями](http://yavix.ru/%7B+%28%3E%D1%81%D0%B0%D0%B9%D1%82%3C=%3E%5b%3Ehttps:%E2%95%B1%E2%95%B1ru.wikipedia.org%20%E2%95%B1wiki%E2%95%B1%D0%A0%D0%B5%D0%B3%D0%B8%D1%81%D1%82%D1%80_%D1%81%D0%B4%D0%B2%D0%B8%D0%B3%D0%B0_%D1%81_%D0%BB%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D0%BE%D0%B9_%D0%BE%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D0%BE%D0%B9_%D1%81%D0%B2%D1%8F%D0%B7%D1%8C%D1%8E%3C%5d%3C%29+%7D), формирующий последовательность максимальной длины .

Особенностью *самосинхронизирующегося скремблера* ( *СС скремблера* ) является то, что он управляется скремблированной последовательностью, то есть той, которая передаётся в канал. Поэтому при данном виде скремблирования не требуется специальной установки состояний скремблера и дескремблера: скремблированная последовательность записывается в регистры сдвига скремблера и дескремблера, устанавливая их в идентичное состояние. При потере синхронизма между скремблером и дескремблером время восстановления синхронизма не превышает числа тактов, равного числу ячеек регистра скремблера.

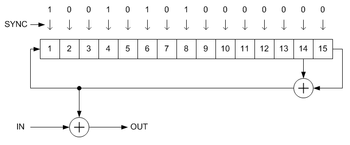
На приёмной стороне выделение исходной последовательности происходит путём [сложения по модулю 2](http://yavix.ru/%7B+%28%3E%D1%81%D0%B0%D0%B9%D1%82%3C=%3E%5b%3Ehttps:%E2%95%B1%E2%95%B1ru.wikipedia.org%20%E2%95%B1wiki%E2%95%B1%D0%A1%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E_2%3C%5d%3C%29+%7D) принятой скремблированной последовательности с последовательностью на выходе сдвигового регистра. Например, для схемы, приведенной на рисунке, входная последовательность IN с помощью скремблера в соответствии с соотношением  преобразуется в посылаемую двоичную последовательность OUT. В приёмнике из этой последовательности таким же регистром сдвига, как на приёме, формируется последовательность

.

Как следует из принципа действия схемы, при одной ошибке в последовательности IN ошибочными получаются также последующие восемнадцатый и двадцать третий символы (в данном примере). В общем случае влияние ошибочно принятого бита будет сказываться *a* раз, где *a* - число обратных связей в регистре сдвига. Таким образом, СС скремблер-дескремблер обладает свойством размножения ошибок. Данный недостаток СС скремблера-дескремблера ограничивает число обратных связей в регистре сдвига; практически это число не превышает *a* = 2.

Второй недостаток СС скремблера связан с возможностью появления на его выходе при определенных условиях так называемых «критических ситуаций», когда выходная последовательность приобретает периодический характер с периодом, меньшим длины ПСП. Чтобы предотвратить это, в скремблере и дескремблере предусматриваются специальные дополнительные схемы контроля, которые выявляют наличие периодичности элементов на входе и нарушают её.

### Аддитивные скремблеры

[](http://yavix.ru/%7b+(%3e%D1%81%D0%B0%D0%B9%D1%82%3c=%3e%5b%3ehttps:%E2%95%B1%E2%95%B1ru.wikipedia.org%20%E2%95%B1wiki%E2%95%B1%D0%A4%D0%B0%D0%B9%D0%BB:Scrambler_randomizer_additive.png%3c%5d%3c)+%7d)

Скремблер аддитивный

При *аддитивном скремблировании* требуется предварительная идентичная установка состояний регистров скремблера и дескремблера. В *скремблере с установкой (АД-скремблере)* , как и в СС скремблере, производится суммирование входного сигнала и ПСП, но результирующий сигнал не поступает на вход регистра. В дескремблере скремблированный сигнал также не проходит через регистр сдвига, поэтому размножения ошибок не происходит.

Суммируемые в скремблере последовательности независимы, поэтому их период всегда равен наименьшему общему кратному величин периодов входной последовательности и ПСП и критическое состояние отсутствует. Отсутствие эффекта размножения ошибок и необходимости в специальной логике защиты от нежелательных ситуаций делают способ аддитивного скремблирования предпочтительнее, если не учитывать затрат на решение задачи синхронизации скремблера и дескремблера. В качестве сигнала установки в цифровых системах передачи данных используют сигнал цикловой синхронизации.

## **Защита телефонных переговоров**

Скремблеры активно применяются для защиты телефонных переговоров. При скремблировании возможно преобразование речевого сигнала по трем параметрам: *амплитуде* , *частоте* и *времени* . Однако в системах подвижной радиосвязи практическое применение нашли в основном частотные и временные преобразования сигнала, а также их комбинации. Возможные помехи в радиоканале существенно затрудняют точное восстановление амплитуды речевого сигнала, в связи с чем амплитудные преобразования при скремблировании практически не применяются.

Основные методы преобразования речевого сигнала:

* Частотные преобразования
  + Частотная инверсия сигнала (преобразование спектра сигнала с помощью гетеродина и фильтра)
  + Разбиение полосы частот речевого сигнала на несколько поддиапазонов и частотная инверсия спектра в каждом относительно средней частоты поддиапазоне
  + Разбиение полосы частоты речевого сигнала на несколько поддиапазонов и их частотные перестановки
* Временные преобразования
  + Инверсия по времени сегментов речи
  + Временные перестановки сегментов речевого сигнала
* Комбинированные методы

### Частотные преобразования

При *частотной инверсии* преобразование спектра речевого сигнала эквивалентно повороту частотной полосы сигнала вокруг некоторой средней частоты F и  — частоты инверсии.

Несколько более сложный по сравнению с частотной инверсией способ преобразования сигнала обеспечивает скремблер с *разбиением полосы речевого сигнала на поддиапазоны* с частотной инверсией сигнала в каждом поддиапазоне (полосно-сдвиговый инвертор). Обычно используется разбиение полосы на 2 поддиапазона.

*Полосовые скремблеры* используют способ разбиения полосы речевого сигнала на несколько поддиапазонов с частотными перестановками этих поддиапазонов. Полосовой скремблер может быть реализован на основе [быстрого преобразования Фурье (БПФ)](http://yavix.ru/%7B+%28%3E%D1%81%D0%B0%D0%B9%D1%82%3C=%3E%5b%3Ehttps:%E2%95%B1%E2%95%B1ru.wikipedia.org%20%E2%95%B1wiki%E2%95%B1%D0%91%D1%8B%D1%81%D1%82%D1%80%D0%BE%D0%B5_%D0%BF%D1%80%D0%B5%D0%BE%D0%B1%D1%80%D0%B0%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%A4%D1%83%D1%80%D1%8C%D0%B5%3C%5d%3C%29+%7D) . В таком скремблере на передающей стороне производится прямое БПФ, частотная перестановка полос, а затем — обратное БПФ. На приёмной стороне осуществляются аналогичные преобразования с обратной частотной перестановкой полос. В скремблерах с БПФ возможно достичь высокой степени защиты информации за счёт увеличения количества перемешиваемых полос, однако на практике этот метод скремблирования в подвижной радиосвязи применяется редко в связи со сложностями технической реализации. Кроме этого, скремблеры с БПФ вносят в канал связи временную задержку.

### Временные преобразования

Простейшим видом временного преобразования является *временная инверсия* , при которой исходный сигнал делится на последовательность временных сегментов и каждый из них передается инверсно во времени — с конца к началу.

В скремблере с *временными перестановками* речевой сигнал делится на временные кадры, каждый из которых в свою очередь подразделяется на сегменты, а затем сегменты речевого сигнала подвергаются перестановке.

### Комбинированные преобразования

Для дальнейшего повышения степени закрытия речи используется комбинация временного и частотного скремблирования. В таком скремблере после аналого-цифрового преобразования спектр оцифрованного речевого сигнала разбивается на частотно — временные элементы, которые затем перемешиваются на частотно — временной плоскости в соответствии с одним из криптографических элементов и суммируются, не выходя за пределы частотного диапазона исходного сигнала.