**Министерство образования и науки Российской Федерации**

**Российский государственный университет нефти и газа (национальный исследовательский университет) имени И.М. Губкина**

**Кафедра Автоматизированных систем управления**

Домашнее задание

дисциплины ***Алгоритмические языки***

***Хэш-функции***

**Выполнил:**

студент группы АС-22-05

Зимин Дмитрий Сергеевич

**Проверила:**

Папилина Т.М.

г. Москва, 2023

Оглавление

[Описание хэш-функций 3](#_Toc152250648)

[Описание и реализация хэш-функций 5](#_Toc152250649)

[MD5 5](#_Toc152250650)

[SHA-1 6](#_Toc152250651)

[SHA-2(512) 7](#_Toc152250652)

[SHA-2(256) 9](#_Toc152250653)

[SHA-3(256) 10](#_Toc152250654)

[ГОСТ-34.11-94 13](#_Toc152250655)

[CRC-32 15](#_Toc152250656)

[Проверка хэш-функций 19](#_Toc152250657)

[Список использованной литературы 19](#_Toc152250658)

# Описание хэш-функций

Криптографическая хэш-функция — это математический алгоритм, который отображает данные произвольного размера в битовый массив фиксированного размера.

Результат, производимый хеш-функцией, называется «хеш-суммой» или же просто «хэшем», а входные данные называют «сообщением».

Для идеальной хэш-функции выполняются следующие условия:

**1. Детерминированность**

Хэш-функция является детерминированной, то есть одно и то же сообщение приводит к одному и тому же хеш-значению. **2. Необратимость**

Невозможно найти сообщение, которое дает заданное хэш-значение. **3. Уникальность**

Невозможно найти два разных сообщения с одинаковым хэш-значением. **4. Непрогнозируемость**

Небольшое изменение в сообщении изменяет хэш настолько сильно, что новое и старое значения кажутся некоррелирующими.

**5. Скорость**

Значение хэш-функции быстро вычисляется для любого сообщения.

Однако идеальную хэш-функцию создать не получится из-за одного условия: **3. Уникальность**

Это условие невозможно выполнить, т.к. количество входных данных, которые мы можем занести в хэш-функцию – бесконечно, а вот размер хэша-ограниченный. Из-за этого может произойти явление коллизии: когда два различных слова дают одинаковый хэш.

**Применение хэша в повседневности**

1. **Проверка целостности файлов:** хэш-функции используют для проверки того, были ли файлы подделаны или изменены путем сравнения сгенерированного хэша с исходным хэшем.

2. **Цифровые подписи:** хэш-функции используются в цифровых подписях для обеспечения аутентификации и неотказуемости, гарантируя, что подписанные данные остаются защищенными от несанкционированного доступа.

3. **Хранение паролей:** хэш-функции надежно хранят пароли, затрудняя злоумышленникам извлечение исходного пароля из значения хэша.

4. **Технология блокчейна:** хэш-функции играют важную роль в защите транзакций блокчейна и обеспечении целостности данных, хранящихся в блоках.

# Описание и реализация хэш-функций

## MD5

**MD5 (Message Digest Algorithm 5)** - это широко используемая криптографическая хеш-функция, которая создает хеш-значение длиной 128 бит, обычно представляемое в виде 32-значного шестнадцатеричного числа. Ее разработал Рональд Ривест в 1991 году, и она часто используется для проверки целостности данных и хранения паролей.

**Основные принципы:**

**Хеш-функция:**

*Детерминированность:* Для одного и того же ввода MD5 всегда будет производить одинаковый вывод.

*Быстродействие:* MD5 создан для скорости, что делает его эффективным для хеширования больших объемов данных.

*Фиксированный размер вывода:* MD5 всегда создает хеш-значение длиной 128 бит, независимо от размера или содержания ввода.

*Стойкость к коллизиям:* Идеально различные входные данные не должны создавать одинаковый хеш MD5 (коллизия). Однако MD5 считается слабым в этом отношении из-за уязвимостей, позволяющих атакам с коллизиями.

**Шаги алгоритма:**

*Инициализация:* Инициализация четырех 32-битных переменных (A, B, C, D) определенными константными значениями.

*Обработка:* Входные данные разделяются на блоки по 512 бит.

Для каждого блока выполняется серия операций (раундов), включая побитовые операции, логические функции и модульное сложение.

*Трансформация:* Каждый блок проходит через несколько раундов (всего 64 для MD5), где значения A, B, C и D обновляются на основе текущего блока и результатов предыдущих операций.

*Вывод:* Окончательные значения A, B, C и D конкатенируются для создания хеш-значения длиной 128 бит.

**Уязвимости:**

*Атаки с коллизиями:* MD5 уязвим к атакам с коллизиями, где различные входные данные создают одинаковое хеш-значение. Эта слабость делает его непригодным для приложений, требующих высокой степени безопасности.

*Криптографическая слабость:* из-за прогресса в области криптоанализа MD5 считается небезопасным для криптографических целей. Рекомендуется использовать более безопасные хеш-функции, такие как SHA-256 или SHA-3.

**Обычные применения:**

*Целостность данных:* Проверка целостности файлов или данных сравнением хеш-значений MD5 до и после передачи.

*Хранение паролей:* MD5 был исторически использован для хеширования паролей, но его уязвимости привели к замене более безопасными алгоритмами, такими как bcrypt и Argon2.

Хотя MD5 был широко использован, его уязвимости и подверженность атакам с коллизиями делают его непригодным для приложений, требующих высокой степени безопасности. Рекомендуется использовать более безопасные хеш-функции для криптографических целей.

## SHA-1

Алгоритм SHA-1 (Secure Hash Algorithm

криптографическая хеш-Функция, разработанная агентством национальной безопасности США (NSA) для использования в цифровой подписи и других приложениях.

SHA-1 генерирует 160-битное (20 байт) криптографически стойкое значение хеша для любого входного сообщения.

Алгоритм работает путем итеративного преобразования блоков данных размером 512 бит в 160-битное значение хеша с использованием логических операций и сдвигов регистров.

Основная идея SHA-1 заключается в том, чтобы сделать невозможным восстановление исходного сообщения по его хешу, а также найти два разных сообщения с одинаковым хешем. Это достигается благодаря криптографической стойкости алгоритма и использованию однонаправленной функции, которая легко вычисляет хеш, но практически невозможно обратить.

SHA-1 был разработан для обеспечения целостности данных и подтверждения их происхождения. Он широко используется в цифровой подписи, протоколах TLS/SSL

и многих других приложениях, где требуется проверка подлинности сообщений.

Алгоритм SHA-1 основан на комбинации различных логических операций, таких как побитовое И (AND), побитовое ИЛИ (OR), побитовое исключающее ИЛИ (XOR), а также циклических сдвигов и вращений битов. Эти операции применяются к блокам данных и внутренним состояниям, что обеспечивает нелинейность и диффузию изменений по всему хешу.

Основной процесс работы алгоритма

SHA-1 включает следующие этапы:

1. ﻿﻿﻿Инициализация: Задаются начальные значения внутренних состояний алгоритма.
2. ﻿﻿﻿Дополнение сообщения: Входное сообщение дополняется дополнительными битами, чтобы его длина стала кратной 512 битам.  
   Дополнение включает бит "1", за которым следуют нулевые биты, а затем добавляется длина исходного сообщения в битах.
3. ﻿﻿﻿Разбиение на блоки: Дополненное сообщение разбивается на блоки размером 512 бит.
4. ﻿﻿﻿Итерационное преобразование:  
   Каждый блок данных проходит через серию итераций, в которых

выполняются операции смешивания, сдвиги, логические функции и обновление внутренних состояний алгоритма.

5. Формирование хеш-значения:

После завершения всех итераций, внутренние состояния алгоритма объединяются в окончательное 160-битное хеш-значение.

SHA-1 обеспечивает хорошую криптографическую стойкость, но с течением времени стали известны некоторые уязвимости, которые могут быть использованы для подбора коллизий (ситуация, когда два разных сообщения дают одинаковый хеш). В связи с этим, алгоритм SHA-1 считается устаревшим для некоторых криптографических задач, и рекомендуется использовать более прочные хеш-функции, такие как SHA-256 или SHA-3.

Однако, SHA-1 все еще широко применяется во многих системах и приложениях, и может быть безопасным для некритических задач, где высокая степень стойкости не требуется.

## SHA-2(512)

SHA-512 расшифровывается как 512-битный алгоритм безопасного хэширования.

‘512’ в SHA-512 означает, что длина создаваемого им хэша составляет 512 бит. Это целых 128 шестнадцатеричных символов!

Алгоритм SHA-512, как и другие члены семейства SHA-2, основан на структуре Меркле-Дамгарда и включает в себя несколько раундов вычислений для обеспечения свойства односторонней функции.

**Шаги алгоритма SHA-512**

1. Заполнение сообщения: Первым шагом в SHA-512 является подготовка входного сообщения к обработке. Алгоритм начинается с добавления бита 1 в конец сообщения. Далее добавляется k битов 0, где k - наименьшее неотрицательное решение уравнения l + 1 + k ≡ 896 mod 1024 (l - длина исходного сообщения в битах). Наконец, к сообщению добавляется 128-битное представление l в формате big-endian. Этот процесс гарантирует, что общая длина сообщения кратна 1024 битам, размеру блока для SHA-512.

2. Синтаксический анализ сообщения: дополненное сообщение затем разбивается на N 1024-битных блоков. Эти блоки обрабатываются один за другим.

3. Настройка начальных значений хэша: перед началом обработки SHA-512 инициализирует восемь 64-битных слов (значений хэша). Они получены из дробных частей квадратных корней первых восьми простых чисел.

4. Обработка блоков сообщений: Каждый блок сообщения обрабатывается в том порядке, в каком они отображаются в дополненном сообщении. Обработка каждого блока включает в себя серию из 80 шагов с использованием расписания сообщений из 80 64-битных слов, восьми значений хэша и 80 постоянных 64-битных слов, полученных из дробных частей кубических корней первых восьмидесяти простых чисел. Вот упрощенное описание шагов, выполняемых в каждом раунде:

* Инициализация первые 16 слов в расписании сообщений словами блока сообщений.
* Остальная часть расписания сообщений генерируется с использованием некоторых логических функций.
* Инициализируйте восемь рабочих переменных, от a до h, текущим значением хэша.
* Для каждого раунда от 0 до 79 выполняются некоторые логические функции, побитовые операции и модульные дополнения к рабочим переменным и словам расписания сообщений.
* Значение хэша вычисляется путем добавления сжатого фрагмента к текущему значению хэша.

5. Создайте окончательное хэш-значение: после обработки всех блоков сообщений результирующие хэш-значения объединяются, создавая 512-битный дайджест сообщения.

Помните, что каждая операция в рамках алгоритма разработана таким образом, чтобы гарантировать, что даже небольшое изменение входных данных приведет к значительному изменению выходных данных, повышая безопасность функции.

**Плюсы SHA-512:**

1. Безопасность: Являясь членом семейства SHA-2, SHA-512 обеспечивает надежную защиту. Длина хэша в 512 бит делает его устойчивым к атакам методом перебора и коллизиям.

2. Скорость: при запуске на 64-разрядных платформах SHA-512 может работать быстрее, чем его аналог SHA-256. Это связано с тем, что он обрабатывает данные большими 64-разрядными блоками.

3. Широкое использование и принятие: SHA-512 — это общепринятый в отрасли стандарт, используемый в самых разных приложениях, от SSL и TLS до IPSec для защиты сетевого трафика. Такое широкое признание указывает на его надежность.

**Минусы SHA-512:**

1. Скорость в 32-разрядных системах: В 32-разрядной системе производительность SHA-512 не так эффективна, как SHA-256. Это медленнее, потому что ему приходится обрабатывать большие 64-битные блоки данных по частям.

2. Использование памяти: SHA-512 использует больше памяти, чем SHA-256 или SHA-1. Это может быть ограничивающим фактором в средах с ограниченными ресурсами или во встроенных системах.

3. Излишество для определенных приложений: Для многих общих целей SHA-256 или даже SHA-1 обеспечивает достаточную безопасность. Использование SHA-512 может оказаться ненужным, поскольку потребляет дополнительные ресурсы без дополнительной пользы.

4. Угрозы квантовых вычислений: несмотря на то, что в настоящее время SHA-512 безопасен, как и все хэш-функции, потенциально может стать уязвимым для атак будущих квантовых компьютеров.

**В заключение**, является ли SHA-512 правильным выбором или нет, зависит от вашего конкретного варианта использования. Необходимо учитывать такие факторы, как требуемый уровень безопасности, системные ресурсы и вычислительная эффективность.

## SHA-2(256)

Хэш функция SHA-256 очень похожа концептуально на SHA-512.Она основана на тех же принципах, что и 512. Основные отличия:

1. Размер блока 512 бит
2. 32-битные значение слов хэша
3. 64 раунда операций

*Безопасность*

Поскольку в настоящее время ни одна технология не способна взломать шифрование SHA-2 и коллизий обнаружено не было, безопасность,

обеспечиваемая SHA-256 и SHA-512, практически идентична. Теоретически

возможно, что SHA-512 будет иметь небольшое преимущество в

безопасности, учитывая появление квантовых компьютеров. Тем не менее,

гораздо более вероятно, что все алгоритмы SHA-2 потребуется заменить

более новыми поколениями квантово-устойчивых алгоритмов, чтобы

эффективно смягчить потенциальные коллизионные атаки.

*Вычислительная мощность*

SHA-256 выдает меньшие выходные данные, чем SHA-512. Это означает, что для хранения и передачи данных обычно требуется меньшая пропускная способность. Использование SHA-256 обычно требует меньше памяти, а в некоторых случаях и вычислительной мощности. Как показывает этот тест, SHA-256 работает на быстрее, чем SHA-512.

## SHA-3(256)

SHA-3 (Keccak) – алгоритм хеширования переменной разрядности, разработанный группой во главе с Йоаном Дайменом в 2012 году. 5 августа 2015 года алгоритм утверждён и опубликован в качестве стандарта FIPS 202. Keccak был выбран в качестве официального алгоритма для SHA-3 в 2012 году. Keccak основан на конструкции Sponge (Губка), которая является новым способом проектирования хеш-функций. Губка — это итеративная конструкция для создания функции с произвольной длиной на входе и произвольной длиной на выходе на основе преобразований перестановки.

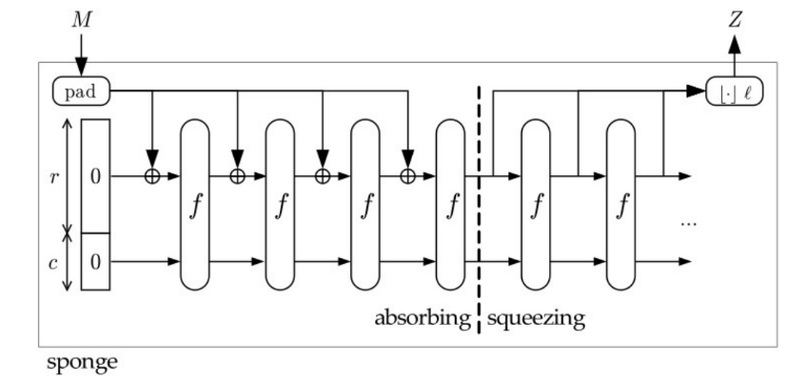


Рисунок (схема губки)

Алгоритм состоит из 2 этапов – Впитывание (англ. Absorbing) и отжимание (англ. Squeezing). Впитывание – входящее сообщение подвергается многораундовым перестановкам f. Отжимание – вывод получившегося значения. А что такое r и c? Это части входной строки, сумма которых должны быть равна функции Keccak. Разработчики рекомендуют использовать Keccak – f{1600} – следовательно r+c=1600. Для алгоритма SHA3-256 r=1088, а c=512. Количество функций перестановок для нашего случая равно 24. (размер части состояния, который записывается и считывается, называется «скоростью» (англ. rate) и обозначается r, а размер части, которая нетронута вводом / выводом, называется «емкостью» (англ. capacity) и обозначается c.)

Основные шаги алгоритма:

1. **Дополнение нашего сообщения.**

Мы должны это сделать таким образом, чтобы сообщение было кратно r. Первый и последний бит заполнения будут ‘1", а все биты между ними "0". После заполнения они делятся на " n " частей, где n \* r эквивалентно длине дополненного сообщения.

1. **Впитывание**

Каждый блок Pi дополняется нулями до строки длины b бит (b=r+c) и суммируется по модулю 2 со строкой состояния S аналогичной длины b. Перед началом работы функции все элементы S равны нулю. Для каждого следующего блока состояние — строка, полученная применением функции перестановок к результату предыдущего шага.

Рассмотрим более подробно функции перестановок. Функция перестановок, используемая в SHA-3, включает в себя исключающее «ИЛИ» (XOR), побитовое «И» (AND) и побитовое отрицание (NOT). Функция определена для строк длины-степени 2. w=2^l(l=6) →w=64 Состояние S можно представить в виде трёхмерного массива A размером 5×5×w.

Тогда элемент массива A[x][y][z] будет размером (5x+y)×w+z строки состояния S. Внутри функции мы выполняем один и тот же набор из пяти операций {θ, ρ, π, χ, ι} в течение двадцати четырех раз. На каждом шаге обозначим входной массив A выходной массив A':

**Шаг θ**: для всех x и z, где 0≤x<5 и 0≤z<ω следует

C(x,z)=A[x, 0, z] XOR A[x, 1, z] XOR A[x, 2, z] XOR A[x, 3, z] XOR A[x, 4, z]

D(x,z) = C[(x-1) mod 5, z] XOR C[(x+1) mod 5, (x-1) mod ω]

Затем A’[x,y,z]= A[x,y,z] XOR D[x,z].

**Шаг ρ**: для всех z, 0≤z<ω, A’[0,0,z]=A[0,0,z]. Для всех t от 0 до 23 и с начальными (x,y)=(1, 0)

1)Для всех k A’[x,y,z]=A[x,y,(z-(t+1)(t+2)/2)mod ω]

2)(x,y)=(y,(2x+3y)mod 5)

**Шаг π**: A’[x,y,z]=A[(x+3y)mod 5, y,z].

**Шаг χ:** при 0≤x<5, 0≤y<5.

A’[x,y,z]=A[x,y,z] XOR((NOT A[(x+1)mod 5,y,z])AND A[(i+2) mod 5, y, z])

**Шаг ι** - На данном шаге вводится функция rc(t)

Если t mod 255 =0, то возвращается 1

Пусть R=[10000000]

Для x от 1 до t mod 255

R=0||R, R[0]=R[0]XOR R[8], R[4]=R[4]XOR R[8], R[5]=R[5]XOR R[8], R[6]=R[6]XOR R[8]

Затем берется целая часть R и возвращается R[0].

Перейдем к алгоритму шага, xr – номер раунда перестановок, RC – нулевой массив длины ω, для x от 0 до l следует RC[2x-1]=rc(x+7xr), затем для всех z A’[0,0, z]=A[0,0,z] XOR RC[z].

Каждый раунд перестановки добавляет асимметрию в код, чем больше раундов, тем асимметричнее результат. Асимметрия необходима, потому что без неё все раунды были бы друг другу эквивалентны, что позволило бы без особых проблем найти исходное сообщение. В данном случае, с каждым раундом значения с каждым разом все сильнее отличаются друг от друга.

Данный шаг состоит из сложения с раундовыми константами, таблица которых приведена ниже

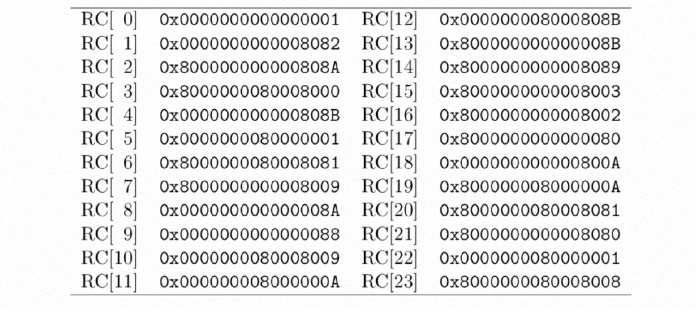


Рисунок (константы)

1. **Отжимание**

Пока длина меньше d (d— количество бит в результате хеш-функции), к длине добавляется r первых бит состояния S после каждого прибавления к S применяется функция перестановок. Затем происходит обрезается до длины d бит. Строка длины d бит возвращается в качестве результата.

## ГОСТ-34.11-94

ГОСТ Р 34.11-94 «Информационная технология. Криптографическая защита информации. Функция хэширования» — устаревший российский криптографический стандарт вычисления хеш-функции, основанный на ГОСТ Р 34.10-94. В странах СНГ переиздан и используется как межгосударственный стандарт ГОСТ 34.311-95.

Стандарт определяет алгоритм и процедуру вычисления хеш-функции для последовательности символов. Этот стандарт является обязательным для применения в качестве алгоритма хеширования в государственных организациях РФ и ряде коммерческих организаций.

ГОСТ Р 34.11-94 определяет алгоритм и процедуру вычисления хэш-функции для любой последовательности двоичных символов, которая применяется в криптографических методах обработки и защиты информации, в том числе для реализации процедур электронной цифровой подписи при передаче, обработке и хранении информации в автоматизированных системах.

Алгоритм хэширования ГОСТ 34.11-94 состоит из нескольких шагов:

**1. Инициализация начального вектора (IV):**

В данной реализации начальный вектор h инициализируется нулями. Этот вектор будет обновляться по мере обработки блоков сообщения.

**2. Дополнение сообщения:**

Длина сообщения выравнивается до кратного 64 битам (8 байтам) путем добавления нулей. Если длина сообщения уже кратна 64 битам, не добавляются дополнительные байты.

**3. Обработка блоков:**

Сообщение разбивается на блоки по 64 бита (8 байт). Каждый блок подается на вход функции шифрования ГОСТ 34.11-94.

**4. Функция шифрования ГОСТ 34.11-94 (gostBlockEncrypt):**

Каждый блок шифруется отдельно. В функции шифрования выполняется нелинейное преобразование с использованием S-блока, функции подстановки P и повторяющихся циклов.

**5. Преобразование хеша в строку:**

Конечный результат представляется в виде строки шестнадцатеричных символов.

**Минусы алгоритма:**

**Устаревший стандарт:**

ГОСТ 34.11-94 является устаревшим стандартом, и сейчас предпочтение отдается более современным хеш-функциям, таким как SHA-256.

**Отсутствие открытых результатов:**

Безопасность ГОСТ 34.11-94 основывается на секретности некоторых параметров S-блока, и открытых доказательств его стойкости не существует.

**Отсутствие подтвержденной стойкости:**

Несмотря на предполагаемую стойкость, не проведены обширные криптографические анализы, которые могли бы подтвердить его надежность на текущий момент.

**Отсутствие широкой международной поддержки:**

ГОСТ 34.11-94 не получил широкого распространения за пределами стран, использующих стандарты ГОСТ.

**Ограниченный размер хеша:**

Размер хеша составляет всего 256 бит (32 байта), что считается недостаточным для некоторых современных приложений, требующих более длинные хеши для обеспечения стойкости.

## CRC-32

Контрольная сумма — это число, которое вычисляется из некоторых данных с помощью специального алгоритма. Её главная задача — проверка целостности данных и обнаружение ошибок. Один из самых распространенных алгоритмов для расчета контрольной суммы — CRC32 (Cyclic Redundancy Check 32). Он используется во множестве протоколов и форматов данных, включая Ethernet, ZIP, PNG и другие.

Размер CRC32 составляет 32 бита и он представляет собой 8-значное шестнадцатеричное число. Для вычисления CRC32 необходимо сначала инициализировать его начальным значением, а затем последовательно обрабатывать каждый байт данных с помощью определенных операций. Конечный результат будет представлять собой контрольную сумму данных.

**Что такое контрольная сумма?**

Контрольная сумма — это числовое значение, которое вычисляется на основе определенного алгоритма и используется для проверки целостности данных. Она представляет собой компактное представление данных, которое позволяет быстро и легко проверить, были ли данные повреждены или изменены.

Для вычисления контрольной суммы обычно используются различные алгоритмы, такие как CRC32 (Cyclic Redundancy Check), которые получают на вход некоторые данные и возвращают контрольную сумму. Алгоритмы CRC32 широко используются в различных областях, таких как сетевые протоколы, хранение данных и др.

Контрольная сумма может быть использована для различных целей, например:

1. Проверка целостности данных при передаче. При передаче файлов по сети или записи на носитель информации данные могут быть повреждены. Контрольная сумма позволяет быстро определить, были ли данные повреждены в процессе передачи.
2. Проверка целостности файлов. По контрольной сумме можно проверить, не были ли данные файла повреждены или изменены после его создания.
3. Поиск ошибок. Контрольная сумма может использоваться для поиска ошибок в данных, например, в тексте сообщения или других форматах данных.

**Как работает алгоритм CRC32?**

CRC32 (Cyclic Redundancy Check) – это алгоритм контрольной суммы, который используется для проверки целостности данных. Он особенно полезен в передаче данных по ненадежным каналам связи или при хранении данных на ненадежных носителях.

Алгоритм CRC32 основан на полиномиальной арифметике, где биты входных данных рассматриваются как коэффициенты полинома. Основная идея заключается в генерации контрольной суммы на основе деления полинома на заранее заданный образующий полином.

В случае алгоритма CRC32 используется образующий полином степени 32 (32 бита). Этот полином представляется в виде 32-битного числа, где коэффициентами являются биты полинома. Примером такого полинома является 0x04C11DB7.

**Подготовка данных для расчёта контрольной суммы**

**Шаги подготовки данных для расчета контрольной суммы:**

1. Определите формат данных, для которых необходимо посчитать контрольную сумму. Это может быть, например, текстовая строка, бинарный файл или пакет сетевых данных.
2. Преобразуйте данные в последовательность битов. Для этого, в зависимости от формата данных, может потребоваться выполнить соответствующие операции. Например, для текстовой строки нужно провести кодирование символов в байты.
3. Определите параметры алгоритма CRC32, которые будут использоваться при расчете контрольной суммы. Обычно это представляет собой набор параметров (например, полинома) для инициализации и выполнения алгоритма CRC32.

**Приведем пример подготовки данных для расчета контрольной суммы:**

1. Предположим, что у нас есть текстовая строка «Hello, world!» для которой требуется посчитать контрольную сумму.
2. Преобразуем строку в байты. В данном случае, используется ASCII кодировка для преобразования каждого символа в соответствующий байт.
3. Устанавливаем параметры алгоритма CRC32, такие как полином, начальное значение и порядок обработки битов.

**Как правильно представить данные?**

Для правильного вычисления контрольной суммы с использованием алгоритма CRC32 необходимо корректно представить данные, с которыми будет выполняться расчет.

Важно понимать, что каждая порция данных должна быть представлена в определенном формате, чтобы гарантировать точность результатов. Ошибки в представлении данных могут привести к некорректному расчету контрольной суммы.

Ниже приведены некоторые рекомендации по представлению данных перед вычислением контрольной суммы с использованием алгоритма CRC32:

* Битовое представление: Данные должны быть представлены в виде последовательности битов, где каждый бит имеет значение 0 или 1.
* Порядок битов: Важно указать порядок следования битов в байте. Обычно используется порядок Big Endian (старший бит первый) или Little Endian (младший бит первый).
* Выравнивание байтов: Если данные представлены в виде последовательности байтов, важно учесть правильное выравнивание байтов. Некорректное выравнивание может привести к неправильному расчету контрольной суммы.
* Дополнительная информация: Если необходимо добавить дополнительные данные, такие как заголовки или метаданные, важно учитывать их в представлении данных.

**Преимущества использования CRC32:**

1. Быстрый и эффективный алгоритм
2. Высокий уровень обнаружения ошибок
3. Простота реализации
4. Широко используется в различных протоколах связи и архиваторах

**Недостатки использования CRC32:**

1. Низкий уровень коррекции ошибок. CRC32 предназначен прежде всего для обнаружения ошибок, а не их исправления.
2. Возможность ложноположительных срабатываний. В редких случаях CRC32 может ошибочно указывать на наличие ошибки.

# Проверка хэш-функций

Для исследования изученных хэш-функций мы создали быстрый тест, который проверяет хэш-функции на основные её постулаты, т.е. на детерминируемость, непрогонозируемость и т.д

В ходе проверки мы удостоверились, что все функции выполняли свою задачу и подходят по большенству параметров.

Отмечу, что

# Список использованной литературы

1. [Что такое хэш-функция?](https://habr.com/ru/articles/534596/) Статья Habr (д. о. 22.11.2023)
2. [Как работает хэширование?](https://habr.com/ru/companies/ruvds/articles/747084/) Статья Habr (д. о. 22.11.2023)
3. [Видео о работе хэш-функции](https://www.youtube.com/watch?v=xV8USnjKGCU) (д. о. 11.11.2023)