Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Отчет по лабораторной работе №9**

**«Исследование криптографических хеш-функций»**

Выполнил:

Cтудент 3 курса 1 группы

Парибок И. А.

Вариант 5

Минск 2023

**Цель:** изучение алгоритмов хеширования и приобретение практических навыков их реализации и использования в криптографии.

Задачи:

1. Закрепить теоретические знания по алгебраическому описанию и алгоритмам реализации операций вычисления однонаправленных хеш- функций.

2. Освоить методику оценки криптостойкости хеш-преобразований на основе «парадокса дня рождения».

3. Разработать приложение для реализации заданного алгоритма хеширования (из семейств MD и SHA).

4. Оценить скорость вычисления кодов хеш-функций.

5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

**Теоретические сведения**

Хеширование — это преобразование информации с помощью особых математических формул. В результате возникает хеш — отображение данных в виде короткой строки, в идеале — уникальной для каждого набора информации. Размер строки может быть одинаковым для информации разного объема.

**Что такое хеш?**

Хеш — это не закодированная исходная информация. Это скорее уникальная метка, которая генерируется для каждого набора данных индивидуально. Если захешировать большую книгу и одно слово, получатся хеши одинаковой длины. А если изменить в слове одну букву и снова захешировать полученную строку, новый хеш будет совершенно другим, там не окажется участков, которые повторяли бы предыдущий.

Математические алгоритмы, по которым хешируется информация, называются хеш-функциями. Их название тоже иногда сокращают как «хеш». Хеш-функций существует очень много, они различаются методами вычислений, назначением, надежностью и другими параметрами.

**Для чего нужно хеширование?**

Основное назначение хеширования — проверка информации. Эта задача важна в огромном количестве случаев: от проверки паролей на сайте до сложных вычислений в блокчейне. Так как хеш — это уникальный код определенного набора данных, по нему можно понять, соответствует ли информация ожидаемой. Поэтому программа может хранить хеши вместо образца данных для сравнения. Это может быть нужно для защиты чувствительных сведений или экономии места.

**Как работает хеш-функция?**

Возможных преобразований для получения хеша бесконечное количество. Это могут быть формулы на основе умножения, деления и других операций, алгоритмы разного уровня сложности. Но если хеш применяется для защиты данных, его функция должна быть криптографической — у таких хеш-функций есть определенные свойства. Именно криптографические хеш-функции используются, например, при хранении паролей.

Если говорить о криптографической хеш-функции, то она чаще всего работает в несколько шагов. Данные разбиваются на части и проходят через сжимающую функцию, которая преобразовывает информацию в меньшее количество бит. Функция должна быть криптостойкой — такой, результат которой практически невозможно вскрыть.

**Свойства криптографических хеш-функций**

1. Необратимость. Из хеша нельзя получить исходные данные даже теоретически. Слишком много информации отбрасывается в процессе; это не зашифровка информации.
2. Детерминированность. Если подать хеш-функции одинаковые данные, то и хеш у них будет одинаковым. Именно это свойство позволяет использовать хеши для проверки подлинности информации.
3. Разнообразие. Даже если два набора информации различаются одним-двумя символами, их хеши будут кардинально разными. У них не будет общих блоков, по ним невозможно будет понять, что исходные данные схожи.
4. Высокая скорость генерации. Это в целом свойство любых хешей: в отличие от зашифрованных версий файлов, они генерируются быстро, даже если входной массив данных большой.

**Безопасность криптографических хеш-функций.**

Цель использования хешей — обеспечить безопасность пользователей. Идентификация или проверка подлинности данных нужны, чтобы никто не мог воспользоваться чувствительной информацией в своих целях. Поэтому специалисты пользуются именно криптографическими хеш-функциями. Они должны быть безопасными — так, чтобы никто не мог взломать их.

Идеальная криптографическая хеш-функция полностью отвечает перечисленным ниже требованиям. Реальные не могут ответить им на 100%, поэтому задача их создателей — максимально приблизиться к нужным свойствам.

**Стойкость к коллизиям**. Выше мы писали, что коллизия — явление, когда у двух разных наборов данных получается одинаковый хеш. Это небезопасно, потому что так злоумышленник сможет подменить верную информацию неверной. Поэтому коллизий стремятся максимально избегать. Современные криптографические хеш-функции не полностью устойчивы к коллизиям. Но так как они очень сложные, для поиска коллизии нужно огромное количество вычислений и много времени — годы или даже столетия. Задача такого поиска становится практически невыполнимой.

**Стойкость к восстановлению данных.** Частично это означает все ту же необратимость. Но восстановить данные в теории можно не только с помощью обратной функции — еще есть метод подбора. Стойкость к восстановлению данных подразумевает, что, даже если злоумышленник будет очень долго подбирать возможные комбинации, он никогда не сможет получить исходный массив информации.

**Устойчивость к поиску первого и второго прообраза.** Первый прообраз — как раз возможность найти обратную функцию. Такой возможности нет, ведь криптографическая хеш-функция необратима. Этот пункт пересекается с требованием стойкости к восстановлению данных.

Второй прообраз — почти то же самое, что нахождение коллизии. Разница только в том, что в случае со вторым прообразом ищущий знает и хеш, и исходные данные, а при поиске коллизии — только хеш. Хеш-функция, неустойчивая к поиску второго прообраза, уязвима: если злоумышленник будет знать исходные данные, он сможет подменить информацию.

Криптографические хеш-функции устойчивы к поиску второго прообраза потому же, почему они считаются стойкими к коллизиям. Вычисления для нахождения таких данных слишком сложные и длительные, чтобы задача была реальной.

**Практическая часть**

1. Разработать оконное приложение, реализующее один из алгоритмов хеширования из указанного преподавателем семейства (MD или SHA; или иного). При этом можно воспользоваться доступными готовыми библиотеками. Язык программирования – на свой выбор. Приложение должно обрабатывать входные сообщения, длина которых определяется спецификацией на реализуемый алгоритм.
2. Оценить быстродействие выбранного алгоритма хеширования.

Для работы был выбран алгоритм SHA-256.

SHA-256 (Secure Hash Algorithm 256) - это криптографическая функция хеширования, которая принимает на вход сообщение произвольной длины и вычисляет для него фиксированный 256-битный хеш. SHA-256 был разработан Национальным институтом стандартов и технологий США (NIST) в 2001 году и стал одним из наиболее распространенных алгоритмов хеширования.

SHA-256 является безопасным алгоритмом хеширования благодаря свойствам, называемым "криптографической стойкостью". Эти свойства обеспечивают, что изменение даже небольшой части входного сообщения приведет к существенным изменениям в выходном хеше, что делает практически невозможным подделку или подмену данных.

SHA-256 использует блочный алгоритм хеширования, который делит входное сообщение на блоки фиксированного размера (512 бит) и применяет к каждому блоку серию математических операций для вычисления промежуточных хешей. Затем эти промежуточные хеши объединяются для получения окончательного 256-битного хеша.

SHA-256 имеет несколько преимуществ перед своими предшественниками, такими как MD5 и SHA-1. В частности, SHA-256 обеспечивает более высокий уровень стойкости к коллизиям (когда два разных входных сообщения дают одинаковый хеш) и предоставляет большую длину выходного хеша, что уменьшает вероятность возникновения коллизий.

SHA-256 широко используется в различных криптографических протоколах и приложениях, таких как цифровые подписи, аутентификация, защита паролей, хранение паролей в хэш-таблицах и т.д.

Программа выполнена на языке Python с использованием пакета «hashlib». Листинг кода программы представлен ниже.

import hashlib

input\_data = input('Введите сообщение: ')

hash\_object = hashlib.sha256(input\_data.encode())

hex\_dig = hash\_object.hexdigest()

print(hex\_dig)

Листинг 1 – Реализация алгоритма SHA-256 с использованием библиотеки hashlib

Для измерения скорости выполнения производится 10000 итераций выполнения шифрования с использованием алгоритма SHA-256. Листинг представлен ниже.

import hashlib

import timeit

input\_data = "Hello, world!"

def sha256():

    hash\_object = hashlib.sha256(input\_data.encode())

    hex\_dig = hash\_object.hexdigest()

    return hex\_dig

print(timeit.timeit(sha256, number=10000))

Листинг 2 – Измерения скорости выполнения 10000 итераций

Результат выполнения програмного средства представлен на рисунке 1.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1 – Результат выполнения программы

Вывод: В рамках выполнения данной лабораторной работы были усвоены основные концепции и принципы работы с алгоритмами хеширования, а также получены практические навыки использования SHA-256 для создания криптографически стойких хешей.

Были исследованы свойства SHA-256, такие как криптографическая стойкость и устойчивость к коллизиям, а также изучены методы вычисления хешей с использованием библиотеки hashlib в Python.

Полученные знания и навыки могут быть применены в широком спектре областей, связанных с безопасностью информации, таких как создание цифровых подписей, защита паролей, аутентификация и др.

**Контрольные вопросы**

1. **Дать определение хеш-функции.**

Хеш-функция — это функция, которая принимает на вход некоторые данные произвольной длины и преобразует их в фиксированный набор битов (хеш-код), который обычно имеет гораздо меньшую длину, чем исходные данные. Цель хеш-функции - создать уникальный "отпечаток" (хеш) для каждого набора данных, чтобы обеспечить быстрый доступ к этим данным и ускорить поиск, сортировку и сравнение данных. Хеш-функции широко применяются в криптографии, базах данных, компьютерной безопасности, а также в других областях, где требуется эффективное хранение и поиск больших объемов данных.

**2. Что такое «однонаправленность» хеш-функций и какова роль этого свойства хеш-функций в криптографии?**

Однонаправленность (или необратимость) является одним из основных свойств хеш-функций в криптографии. Она означает, что из хеш-значения нельзя восстановить исходные данные, которые были использованы для его вычисления. То есть, если мы имеем некоторый хеш-код H для данных D, мы не можем найти какие-то другие данные D', которые также дают хеш-код H.

Это свойство очень важно в криптографии, так как позволяет использовать хеш-функции для создания цифровых подписей, хранения паролей, проверки целостности данных и других задач, которые требуют сохранения конфиденциальности и защиты от взлома.

**3. Что такое «коллизия»? Типы коллизий хеш-функций.**

Коллизия — это ситуация, когда два разных набора данных дают одинаковый хеш-код (то есть, их хеш-значения совпадают). Таким образом, коллизия означает, что хеш-функция не может гарантировать уникальность хеш-значений для всех возможных входных данных.

Существуют два типа коллизий:

Случайная коллизия - когда два разных набора данных случайным образом дают одинаковый хеш-код. Такие коллизии могут возникать даже в идеальной хеш-функции, поскольку количество возможных наборов данных намного больше, чем количество возможных хеш-значений.

Целенаправленная коллизия - когда злоумышленник специально создает два набора данных, которые дают одинаковый хеш-код. Этот тип коллизий может использоваться для атак на системы, которые используют хеш-функции для проверки целостности данных или аутентификации.

В криптографических приложениях очень важно минимизировать вероятность появления коллизий. Для этого используются криптографические хеш-функции, которые обладают дополнительными свойствами, такими как однонаправленность и устойчивость к целенаправленным атакам на создание коллизий. Однако, даже самые криптографические хеш-функции могут иметь некоторую вероятность коллизий, поэтому приложения, использующие хеш-функции, должны быть спроектированы с учетом возможности коллизий.

**4. Сформулировать в общем виде парадокс «дней рождений».**

Парадокс «дней рождений» заключается в том, что при наличии группы из n случайных людей вероятность того, что среди них найдутся хотя бы два человека с одинаковой датой рождения, оказывается значительно выше, чем может показаться на первый взгляд.

Более точно, вероятность появления хотя бы одной пары людей с одинаковыми днями рождения растет с увеличением числа людей в группе. Например, если в группе 23 человека, то вероятность появления хотя бы одной пары с одинаковыми днями рождения уже превышает 50%, а при 70 человеках вероятность становится уже выше 99%.

Этот парадокс возникает из-за того, что количество возможных комбинаций дат рождений в группе растет гораздо быстрее, чем само число людей в группе. Таким образом, вероятность появления хотя бы одной пары с одинаковыми днями рождения растет с увеличением числа комбинаций дат рождений, а не просто с увеличением числа людей в группе.

**5. Как парадокс «дней рождений» используется в криптографии?**

Парадокс «дней рождений» может использоваться в криптографии для анализа криптографических хеш-функций и вычисления вероятности возникновения коллизий.

Для этого можно представить все возможные значения хеш-функции как дни рождения, а входные данные (например, сообщения) как людей, имеющих соответствующие дни рождения. Тогда вероятность возникновения коллизии в хеш-функции можно оценить, опираясь на парадокс «дней рождений».

Например, если для криптографической хеш-функции используется 128-битный хеш-код, то количество возможных значений этого хеш-кода составляет 2 в степени 128. Используя формулу парадокса «дней рождений», можно оценить вероятность коллизии при заданном количестве случайных сообщений.

Таким образом, парадокс «дней рождений» может помочь оценить безопасность криптографических хеш-функций и выбрать такую функцию, которая обеспечивает достаточный уровень защиты от возможных коллизий.

**6. Сколько попыток нужно сделать, чтобы с вероятностью более 0,5 (0,7; 0,8; 0,9) обнаружить коллизию при длине хеша (l) 64 (128; 256; 512) битов?**

Для решения этой задачи мы можем использовать формулу для оценки количества попыток, необходимых для обнаружения коллизии с заданной вероятностью.

Формула имеет вид: N = sqrt(2 \* (2 ^ l) \* (-ln(1-p))) где: N - количество попыток l - длина хеша в битах p - заданная вероятность обнаружения коллизии (0,5; 0,7; 0,8; 0,9)

Для удобства расчетов, можно использовать приближенное значение числа e: 2,718.

Для решения задачи, подставим значения l и p в формулу и рассчитаем N:

При l=64 битах:

* Для вероятности 0,5: N = sqrt(2 \* (2^64) \* (-ln(1-0.5))) ≈ 5.84 \* 10^8 попыток
* Для вероятности 0,7: N = sqrt(2 \* (2^64) \* (-ln(1-0.7))) ≈ 9.71 \* 10^8 попыток
* Для вероятности 0,8: N = sqrt(2 \* (2^64) \* (-ln(1-0.8))) ≈ 1.40 \* 10^9 попыток
* Для вероятности 0,9: N = sqrt(2 \* (2^64) \* (-ln(1-0.9))) ≈ 2.46 \* 10^9 попыток

При l=128 битах:

* Для вероятности 0,5: N = sqrt(2 \* (2^128) \* (-ln(1-0.5))) ≈ 6.87 \* 10^19 попыток
* Для вероятности 0,7: N = sqrt(2 \* (2^128) \* (-ln(1-0.7))) ≈ 1.15 \* 10^20 попыток
* Для вероятности 0,8: N = sqrt(2 \* (2^128) \* (-ln(1-0.8))) ≈ 1.65 \* 10^20 попыток
* Для вероятности 0,9: N = sqrt(2 \* (2^128) \* (-ln(1-0.9))) ≈ 2.90 \* 10^20 попыток

При l=256 битах:

* Для вероятности 0,5: N = sqrt(2 \* (2^256) \* (-ln(1-0.5))) ≈ 8.86 \* 10^38 попыток
* Для вероятности 0,7: N = sqrt(2 \* (2^256) \* (-ln(1-0.7))) ≈ 1.48 \* 10^39 попыток
* Для вероятности 0,8: N = sqrt(2 \* (2^256) \* (-ln(1-0.8 ))) ≈ 2.12 \* 10^39 попыток
* Для вероятности 0,9: N = sqrt(2 \* (2^256) \* (-ln(1-0.9))) ≈ 3.73 \* 10^39 попыток

При l=512 битах:

* Для вероятности 0,5: N = sqrt(2 \* (2^512) \* (-ln(1-0.5))) ≈ 1.15 \* 10^77 попыток
* Для вероятности 0,7: N = sqrt(2 \* (2^512) \* (-ln(1-0.7))) ≈ 1.93 \* 10^77 попыток
* Для вероятности 0,8: N = sqrt(2 \* (2^512) \* (-ln(1-0.8))) ≈ 2.77 \* 10^77 попыток
* Для вероятности 0,9: N = sqrt(2 \* (2^512) \* (-ln(1-0.9))) ≈ 4.88 \* 10^77 попыток

Таким образом, для обнаружения коллизии в хеш-функции с вероятностью более 0,5, 0,7, 0,8 и 0,9 при длине хеша 64, 128, 256 и 512 битов, требуется разное количество попыток, причем это количество растет очень быстро с увеличением длины хеша. Это подчеркивает важность использования хеш-функций с достаточно длинной длиной хеша, чтобы обеспечить безопасность в криптографии.

**7. Дать общую характеристику алгоритмам хеширования семейств MD и SHA. Из каких основных стадий состоит алгоритм хеширования сообщения?**

Семейства MD (Message Digest) и SHA (Secure Hash Algorithm) представляют собой широко используемые криптографические хеш-функции. Хеш-функции семейства MD были разработаны компанией RSA Security, а функции SHA были разработаны Национальным институтом стандартов и технологий (NIST) США.

Алгоритмы хеширования сообщений семейств MD и SHA обладают хорошей однонаправленной свойством, необратимостью и стойкостью к коллизиям. Они используются для обеспечения безопасности во многих криптографических протоколах и системах.

Алгоритм хеширования сообщения состоит из нескольких основных стадий:

Инициализация: установка начальных параметров для алгоритма, включая выбор соответствующей хеш-функции и инициализационных значений.

Добавление сообщения: сообщение, которое нужно захешировать, разбивается на блоки фиксированной длины, которые обрабатываются по очереди.

Преобразование блоков сообщения: каждый блок сообщения обрабатывается с помощью функций сжатия, которые смешивают блок сообщения с текущим значением хеш-функции.

Генерация хеш-кода: после того как все блоки сообщения были обработаны, алгоритм генерирует окончательный хеш-код, который является фиксированной длиной и является уникальным для каждого сообщения.

Для обеспечения стойкости к атакам, хеш-функции семейств MD и SHA используют различные методы, такие как изменение порядка обработки блоков, добавление случайного значения (соли) в сообщение и другие.

**8. Рассчитать общую длину (L') хешируемого сообщения после  
предварительной стадии на основе алгоритма MD, если объем (L) исходного сообщения составлял: 0; 484; 512; 1000; 2000; 16 000 битов. Какова в каждом случае будет длина хеша?**

Длина хеша зависит от конкретного алгоритма хеширования, поэтому необходимо уточнить, какой именно алгоритм MD рассматривается. Для примера возьмем алгоритм MD5.

Для алгоритма MD5 на предварительной стадии исходное сообщение дополняется битом "1" и далее дополняется нулями до того объема, который удовлетворяет условию:

L' = L + 1 + K ≡ 448 (mod 512),

где K - некоторое целое число. Далее добавляется 64-битовое представление длины исходного сообщения L. Таким образом, общая длина хешируемого сообщения для алгоритма MD5 будет кратна 512 и составит:

Если L = 0, то L' = 512 битов.

Если L = 484 бита, то K = 447 - (484 + 1) ≡ -38 (mod 512), L' = 960 битов.

Если L = 512 бит, то L' = 1024 бита.

Если L = 1000 бит, то K = 447 - (1000 + 1) ≡ -554 (mod 512), L' = 1472 бита.

Если L = 2000 бит, то K = 447 - (2000 + 1) ≡ -504 (mod 512), L' = 2496 битов.

Если L = 16000 бит, то K = 447 - (16000 + 1) ≡ -496 (mod 512), L' = 16128 битов.

Длина хеша для алгоритма MD5 всегда составляет 128 бит.

**10. Представить и охарактеризовать структурную схему одного раунда алгоритмов хеширования на основе MD4; MD5; SHA-1.**

MD4, MD5 и SHA-1 - это все алгоритмы хеширования, которые имеют схожую структурную схему одного раунда. Они состоят из нескольких основных шагов.

Структурная схема одного раунда алгоритма хеширования на основе MD4 выглядит следующим образом:

Инициализация вектора состояния (A, B, C, D).

Разбиение входного сообщения на 16 32-битных слов.

Выполнение 3 раундов, каждый из которых состоит из 16 шагов:

a. Побитовая операция ИЛИ между словами B и C, результат сохраняется в переменную F.

b. Побитовая операция ИЛИ между словами B и C, результат сохраняется в переменную F.

c. Побитовая операция ИСКЛЮЧАЮЩЕЕ ИЛИ между словами B, C и D, результат сохраняется в переменную F.

d. Добавление текущего слова в переменную F.

e. Сдвиг влево на определенную величину (2, 4 или 6 бит) значения переменной F.

f. Прибавление значения A к переменной F.

g. Сдвиг влево на определенную величину (3, 7, 11 или 19 бит) значений переменных A, B, C и D.

h. Прибавление значения F к переменной D.

Обновление вектора состояния, прибавление к каждой из переменных A, B, C и D соответствующего слова из входного сообщения.

**11. На чем основан «лавинный эффект» в алгоритмах хеширования? В чем состоит цель его реализации?**

"Лавинный эффект" — это свойство хеш-функции, которое заключается в том, что при небольшом изменении входных данных результат хеширования должен измениться значительно. Это означает, что при изменении даже одного бита входных данных должно произойти изменение значительной части битов хеш-значения.

Это свойство основано на диффузии, т.е. распространении изменений входных данных на все биты хеш-значения. В алгоритмах хеширования, которые реализуют "лавинный эффект", каждый бит хеш-значения должен быть зависим от всех битов входных данных, чтобы даже маленькое изменение входных данных привело к изменению большого числа битов хеш-значения.

Цель реализации "лавинного эффекта" в алгоритмах хеширования состоит в увеличении криптографической стойкости функции, т.е. сложности обнаружения коллизий. Если алгоритм хеширования реализует "лавинный эффект", то любая небольшая изменение входных данных должно привести к значительному изменению хеш-значения, что делает поиск коллизий более сложным.

**12. В чем состоят основные структурные и функциональные особенности алгоритма хеширования SHA-3?**

Алгоритм хеширования SHA-3 (Secure Hash Algorithm 3) - это семейство алгоритмов хеширования, который был выбран в качестве нового стандарта NIST (National Institute of Standards and Technology) в 2015 году. Он является преемником алгоритмов SHA-1, SHA-2 и SHA-0 и был разработан Криптографическим форумом KECCAK.

Основные структурные и функциональные особенности алгоритма хеширования SHA-3 включают:

KECCAK-перестановки: SHA-3 использует KECCAK-перестановки, которые являются уникальными в сравнении с другими алгоритмами хеширования. KECCAK-перестановки являются высокопроизводительными, что позволяет быстро обрабатывать большие объемы данных.

Конструкция "sponge": SHA-3 использует конструкцию "sponge", которая позволяет преобразовывать произвольный входной поток данных в выходной поток фиксированной длины. Конструкция "sponge" также обеспечивает свойство "ложной случайности", что делает алгоритм более стойким к атакам.

Динамические параметры: SHA-3 имеет динамические параметры, которые позволяют настраивать алгоритм на определенные требования без изменения его основной структуры.

Размер хеш-значения: SHA-3 позволяет генерировать хеш-значения различной длины, включая 224, 256, 384 и 512 бит.

Безопасность: SHA-3 является стойким к различным атакам, включая дифференциальный и линейный криптоанализ.

В целом, основные структурные и функциональные особенности алгоритма хеширования SHA-3 включают его уникальные KECCAK-перестановки, конструкцию "sponge", динамические параметры, возможность генерировать хеш-значения различной длины и высокий уровень безопасности.

**13. Охарактеризовать структурные, функциональные особенности и криптостойкость белорусского государственного стандарта хеширования (СТБ 34.101.77–2016)**

Белорусский государственный стандарт хеширования (СТБ 34.101.77–2016) является алгоритмом хеширования, разработанным для использования в криптографических приложениях, и поддерживается государственными органами Беларуси.

Основные структурные и функциональные особенности СТБ 34.101.77-2016 включают:

Размер хеш-значения: алгоритм позволяет генерировать хеш-значения длиной 256 бит.

Структура Merkle-Damgård: СТБ 34.101.77-2016 использует структуру Merkle-Damgård для генерации хеш-значений. Эта структура основана на поэтапном преобразовании блоков данных и обеспечивает высокую скорость обработки данных.

Применение итеративного цикла: алгоритм СТБ 34.101.77-2016 использует итеративный цикл для обработки блоков данных и генерации хеш-значений. Итеративный цикл состоит из нескольких этапов, включая сжатие данных, обновление вектора инициализации и расширение блока данных.

Использование нелинейных S-блоков: СТБ 34.101.77-2016 использует нелинейные S-блоки для обеспечения стойкости алгоритма к различным атакам.

Проверка четности: алгоритм СТБ 34.101.77-2016 включает проверку четности блоков данных для обеспечения целостности хеш-значения.

Криптостойкость алгоритма СТБ 34.101.77-2016 соответствует требованиям, предъявляемым к современным алгоритмам хеширования. Он обеспечивает стойкость к различным атакам, включая дифференциальный и линейный криптоанализ. Однако, как и в случае с любым алгоритмом хеширования, возможны атаки на основе коллизий, которые могут уменьшить стойкость алгоритма.