МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

Специальность Информационные системы и технологии

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №9 НА ТЕМУ:**

**Исследование криптографических хеш-функций**

Выполнил студент 3 курса 1 группы  
Кашперко Василиса Сергеевна

2023 г.

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

**Теоретические сведения**

**Математические основы асимметричных шифров**

Хеш-функция – математическая или иная функция h = H(М), которая принимает на входе строку символов М, называемую также прообразом, переменной длины n и преобразует ее в выходную строку фиксированной (обычно – меньшей) длины l.

Хеширование (или хэширование, англ. hashing) – это преобразование входного массива данных определенного типа и произвольной длины (практически) в выходную битовую строку фиксированной длины.

Хеширование – криптографический процесс, в котором большой фрагмент данных преобразуется в гораздо меньшую строку цифр и букв.

Преобразования называются хеш-функциями, или функциями свертки, а их результаты называют хешем, хеш-кодом, хеш-таблицей или дайджестом сообщения (англ. message digest).

Все существующие функции хеширования можно разделить на два больших класса:

• бесключевые хеш-функции, зависящие только от сообщения;

• хеш-функции с секретным ключом, зависящие как от сообщения, так и от секретного ключа.

Криптографическая хеш-функция – это специальный класс хеш-функций, который имеет различные свойства, необходимые для решения задач в области криптографии.

Основные задачи, решаемые с помощью хеш-функций:

• аутентификация (хранение паролей);

• проверка целостности данных;

• защита файлов;

• обнаружение зловредного ПО;

• криптовалютные технологии.

К основным свойствам хеш-функций можно отнести следующие.

Свойство 1. Детерминированность: независимо от того, сколько раз вычисляется H(M), M – const, при использовании одинакового алгоритма код хеш-преобразования h всегда должен быть одинаковым.

Свойство 2. Скорость вычисления хеша h: если процесс вычисления h недостаточно быстрый, система просто не будет эффективной.

Свойство 3. Сложность обратного вычисления: для известного H(М) невозможно (практически) определить М. Это важнейшее свойство хеш-функции для криптографических применений – свойство односторонности преобразования.

Свойство 4. Даже минимальные изменения в хешируемых данных (М ≠ М') должны изменять хеш: Н(M) ≠ Н(М').

Свойство 5. Коллизионная устойчивость (стойкость).

Зная М, трудно найти такое М' (М ≠ М'), для которого H(М) = H(М').

Если последнее равенство выполняется, то говорят о коллизии 1-го рода.

Если случайным образом выбраны два сообщения (М и М'), для которых H(М) = H(М'), говорят о коллизии 2-го рода.

Это означает, что по хеш-коду должно быть практически невозможным восстановление входной строки М.

Базовые алгоритмы обоих рассматриваемых семейств (MD и SHA) условно можно разделить на 5 стадий:

• расширение входного сообщения;

• разбивка расширенного сообщения на блоки;

• инициализация начальных констант;

• обработка сообщения поблочно (основная процедура алгоритма хеширования);

• вывод результата.

MD5 (Message-Digest Algorithm 5) – это алгоритм хеширования, разработанный Рональдом Ривестом в 1991 году. Он используется для преобразования произвольного сообщения произвольной длины в фиксированный 128-битный хеш-код.

Алгоритм MD5 состоит из четырех основных этапов:

1. Инициализация: задаются начальные значения для регистров, которые используются в процессе хеширования.
2. Предварительная обработка: сообщение дополняется битами таким образом, чтобы его длина была кратна 512 битам. Затем сообщение разбивается на блоки по 512 бит.
3. Основной цикл: каждый блок обрабатывается в цикле, состоящем из 64 итераций. В каждой итерации применяются различные преобразования к регистрам.
4. Финальная обработка: регистры объединяются в 128-битный хеш-код, который и является конечным результатом хеширования.

MD5 также считается устаревшим и небезопасным алгоритмом хеширования, так как имеет ряд уязвимостей, позволяющих произвольно изменять данные, которые приводят к одинаковым хеш-кодам. Рекомендуется использовать более современные алгоритмы хеширования, такие как SHA-256 или SHA-3.

SHA-256 (Secure Hash Algorithm 256) – это один из наиболее распространенных алгоритмов хеширования, который используется для преобразования произвольного сообщения произвольной длины в фиксированный 256-битный хеш-код.

Алгоритм SHA-256 состоит из шести основных этапов:

1. Инициализация: задаются начальные значения для регистров, которые используются в процессе хеширования.
2. Предварительная обработка: сообщение дополняется битами таким образом, чтобы его длина была кратна 512 битам. Затем сообщение разбивается на блоки по 512 бит.
3. Инициализация переменных: переменные инициализируются значениями, используя начальные значения регистров и специальные константы.
4. Основной цикл: каждый блок обрабатывается в цикле, состоящем из 64 итераций. В каждой итерации применяются различные преобразования к переменным.
5. Формирование хеш-кода: после обработки всех блоков, значения переменных объединяются в 256-битный хеш-код.
6. Завершение: переменные и регистры очищаются.

SHA-256 является криптографически стойким алгоритмом, который обеспечивает высокий уровень безопасности при хешировании данных. Он широко используется в различных областях, таких как криптография, блокчейн, безопасность информации и т.д.

SHA-256 лежит в основе биткоина. Он может взять файл любого размера, смешать его и дать данные фиксированного размера длинной 256 бит.

Когда мы пропускаем любой файл/любые данные через алгоритм SHA-256  
или другую хеш-функцию, мы никогда не узнаем, что получим на выходе. Результат будет случайный. Но несмотря на это, результат детерминистический.

То есть, если мы получаем один и тот же файл через алгоритм, то на выходе мы получаем один и тот же результат. Это означает, что имея хэш какого-либо файла, мы сможем проверить этот файл на его подлинность.

Если мы пропустим файл через SHA-256 и увидим известный нам хэш, значит это именно тот файл.

Алгоритм хеширования SHA-256 включает в себя несколько раундов, каждый из которых состоит из нескольких этапов. Один раунд SHA-256 состоит из следующих шагов:

1. Инициализация переменных. На этом этапе инициализируются переменные, которые будут использоваться в ходе раунда.
2. Подготовка расширенного сообщения. Сообщение, которое необходимо захешировать, дополняется до определенной длины и разбивается на блоки.
3. Обновление хеш-значения. На этом этапе происходит обновление текущего хеш-значения. Изначально оно инициализируется определенным значением, а затем изменяется в процессе выполнения алгоритма.
4. Обработка блоков сообщения. Каждый блок сообщения обрабатывается отдельно. Вначале происходит подготовка блока к обработке, затем блок разбивается на 16 32-битных слов, которые затем преобразуются в новые значения с помощью нескольких логических операций.
5. Обновление переменных. На этом этапе происходит обновление переменных, которые будут использоваться на следующем раунде. Они получают значения, которые были вычислены в текущем раунде.
6. Повторение. Если блоки сообщения еще остались, то происходит повторение шагов 3-5.
7. Возвращение хеш-значения. После обработки всех блоков сообщения, текущее значение хеша возвращается в качестве результата работы алгоритма.

Каждый раунд SHA-256 состоит из 64 таких шагов, которые производятся последовательно. Каждый шаг включает в себя несколько операций, таких как вращение, смещение, логические операции и сложение по модулю 2^32. Результаты этих операций объединяются в определенном порядке, что и приводит к формированию нового значения хеш-функции. Таким образом, каждый раунд алгоритма SHA-256 улучшает стойкость хеш-функции и делает ее более надежной для защиты данных.

**Практическая часть**

В ходе лабораторной работы было разработано консольное приложение, которое реализует алгоритмы хеширования SHA256 и MD5.

При этом были использована доступная готовая библиотека System.Security.Cryptography для языка программирования – С#.

Приложение обрабатывает входные сообщения, длина которых определяется спецификацией на реализуемый алгоритм. Результат выполнения шифрования представлен на рисунке 1.

Для обработки входного сообщения "Василиса" алгоритмом SHA256, необходимо выполнить следующие шаги:

1. Предварительная обработка: сообщение "Василиса" дополняется битами таким образом, чтобы его длина была кратна 512 битам. Для этого к сообщению добавляется последовательность битов "1", за которой следует некоторое количество нулей, чтобы длина сообщения стала 448 бит. Затем к сообщению добавляется 64-битное представление длины исходного сообщения (в битах).
2. Инициализация переменных: переменные инициализируются значениями, используя начальные значения регистров и специальные константы.
3. Основной цикл: каждый блок обрабатывается в цикле, состоящем из 64 итераций. В каждой итерации применяются различные преобразования к переменным.
4. Формирование хеш-кода: после обработки всех блоков, значения переменных объединяются в 256-битный хеш-код.

В итоге, для сообщения "Василиса" будет получен 256-битный хеш-код, который будет выглядеть как последовательность 64 шестнадцатеричных цифр. Значение этого хеш-кода зависит только от содержимого входного сообщения "Василиса" и будет одинаковым при каждом повторном вычислении хеш-кода для этого сообщения с помощью алгоритма SHA256.

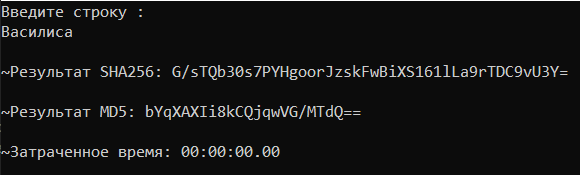


Рисунок 1 – Результат работы программы со строкой «Василиса»

Быстродействие алгоритма SHA-256 зависит от ряда факторов, включая аппаратное и программное обеспечение, на котором он выполняется, а также размер входного сообщения. В целом, SHA-256 является достаточно быстрым алгоритмом, который может обрабатывать большие объемы данных с высокой скоростью.

На современном вычислительном оборудовании, включая смартфоны, ноутбуки и серверы, скорость вычисления хеша сообщения методом SHA-256 составляет несколько миллионов операций в секунду. Это позволяет использовать алгоритм в широком спектре приложений, включая обеспечение целостности данных, аутентификацию и безопасное хранение паролей.

Оба алгоритма [MD5 и SHA256] предоставляют возможность проверки целостности, однако, хеш-функция MD5 уязвима к коллизионным атакам, в то время как хеш-функция SHA-256 являются более безопасной. Для одного и того же входного сообщения хеш-функции всегда выдают одинаковое хеш-значение, что позволяет использовать их для проверки целостности данных.

**Вывод:** мы провели лабораторную работу, в ходе которой исследовали основные свойства криптографических хеш-функций, таких как MD5 и SHA-256, с помощью реализации на языке C#. Это позволило нам узнать, что хеш-функции являются ключевым инструментом в области криптографии и применяются для защиты паролей, цифровых подписей, проверки целостности данных и других задач безопасности. Однако, некоторые хеш-функции могут быть уязвимы к атакам, таким как атака дней рождений и коллизионные атаки. В целом, наша лабораторная работа позволила нам лучше понять принципы работы криптографических хеш-функций и их важность для обеспечения безопасности в различных приложениях.

**Контрольные вопросы**

*1. Дать определение хеш-функции.*

Хеш-функция — это алгоритм, который принимает на вход произвольные данные и преобразует их в фиксированную последовательность символов фиксированной длины, называемую хеш-значением или просто хешем. Хеш-функции широко используются в компьютерных науках и криптографии для обеспечения целостности данных, проверки уникальности и ускорения поиска.

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

"Однонаправленность" хеш-функций означает, что вычисление хеш-значения от исходных данных происходит быстро и эффективно, но обратное восстановление исходных данных из хеш-значения практически невозможно.

Однонаправленные хеш-функции также используются в цифровых подписях и в алгоритмах блокчейна для обеспечения надежности и целостности данных.

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

"Коллизия" — это ситуация, когда двум разным входным данным соответствует одно и то же хеш-значение при использовании хеш-функции.

Типы коллизий хеш-функций:

Случайная коллизия: возникает, когда два разных входных значения случайным образом приводят к одинаковому хеш-значению.

Поиск коллизий: Атака на хеш-функцию, при которой злоумышленник активно ищет два разных входных значения, которые приведут к одному и тому же хеш-значению.

Постепенное формирование коллизий: Атака на хеш-функцию, при которой злоумышленник постепенно изменяет входные данные, чтобы достичь коллизии с заданным хеш-значением.

Использование криптографически стойких хеш-функций помогает минимизировать вероятность коллизий и повышает безопасность систем, которые зависят от хеш-функций.

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

Парадокс "дней рождений" заключается в том, что при достаточно большой группе людей вероятность того, что у двух людей в группе день рождения совпадает, оказывается выше, чем может показаться на первый взгляд.

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

Парадокс "дней рождений" может быть использован в криптографии для атаки на хеш-функции. Хеш-функции используются в криптографии для преобразования сообщения любой длины в фиксированную длину, которая может быть использована для проверки целостности сообщения.

Атака на хеш-функции, основанная на парадоксе "дней рождений", называется атакой дней рождений (birthday attack). Она заключается в том, чтобы найти два разных сообщения, которые дают одинаковый хеш-код. Для этого атакующий генерирует большое количество случайных сообщений и вычисляет их хеши.

При достаточно большом количестве случайных сообщений вероятность того, что какие-то два из них дадут одинаковый хеш-код, становится высокой.

Чтобы защититься от атак дней рождений, в криптографии используются хеш-функции с достаточно большой длиной хеш-кода и другие методы, например, соли или ключи, которые усложняют задачу поиска коллизий хеш-кодов.

*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?*

Как уже описывалось ранее, коллизия – это ситуация, когда два разных входных сообщения дают одинаковый хеш-код. Вероятность обнаружить коллизию зависит от длины хеша и количества входных сообщений, которые необходимо обработать.

Для вычисления необходимого количества попыток, которые 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 вероятностей 0,5, 0,7, 0,8 и 0,9 будем иметь:

***l*=64:**

P=0,5 -> N= ≈ 4,3 млрд

P=0,7 -> N=5,9 млрд

P=0,8 -> N=6,8 млрд

P=0,9 -> N=8,2 млрд

***l*=128:**

P=0,5 -> N= ≈ 1,8 \*

P=0,7 -> N=2,5 \*

P=0,8 -> N=2,9 \*

P=0,9 -> N=3,5 \*

***l*=256:**

P=0,5 -> N=2^128 ≈ 3,4 \*

P=0,7 -> N=4,7 \*

P=0,8 -> N=5,4 \*

P=0,9 -> N=6,5 \*

***l*=512:**

P=0,5 -> N=2^256 ≈ 1,2 \*

P=0,7 -> N=1,6 \*

P=0,8 -> N=1,9 \*

P=0,9 -> N=2,3 \*

Чем больше длина хеша, тем больше попыток необходимо сделать для обнаружения коллизии с заданной вероятностью.

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

Семейства хеш-функций MD (Message Digest) и SHA (Secure Hash Algorithm) являются популярными и широко используемыми алгоритмами хеширования в криптографии и информационной безопасности.

Характеристики семейств MD и SHA:

MD: Семейство хеш-функций MD включает MD5 и более безопасную версию MD - MD5 является устаревшим и не рекомендуется для использования в криптографических приложениях из-за известных уязвимостей. MD5 генерирует 128-битное хеш-значение.

SHA: Семейство хеш-функций SHA включает SHA-1, SHA-256, SHA-384, SHA-512 и другие. Они являются более безопасными и широко используются для целей, требующих хорошей стойкости к коллизиям и безопасности данных. Разные варианты SHA генерируют хеш-значения различных длин (от 160 бит до 512 бит).

Основные стадии алгоритма хеширования сообщения:

Инициализация: Начальная настройка алгоритма хеширования.

Постепенная обработка: Постепенное обрабатывание блоков входных данных для создания промежуточных хеш-значений.

Завершение: Финальная обработка данных и генерация окончательного хеш-значения.

Вывод: Возвращение окончательного хеш-значения как результата алгоритма хеширования.

Эти стадии обеспечивают преобразование исходных данных в фиксированное и уникальное хеш-значение, которое можно использовать для проверки целостности данных и других криптографических целей.

*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 MD, например, для MD5 длина хеша равна 128 бит. Для вычисления длины хешируемого сообщения после предварительной стадии в алгоритмах MD необходимо добавить биты заполнения и длину исходного сообщения (в битах).

Для вычисления длины хешируемого сообщения после предварительной стадии необходимо использовать формулу:

*L'* = (*L* + *k* + 64) mod 512, где *L* - длина исходного сообщения в битах, *k* - минимальное необходимое количество битов заполнения для выравнивания длины сообщения до 448 битов, 64 - количество битов, занимаемых записью длины сообщения.

Для блочного алгоритма MD с блочным размером 512 бит и длиной хеша 128 бит, *k* будет равно 448 - (*L* mod 512) если 448 - (*L* mod 512) ≥ 65, иначе *k* будет равно 960 - (*L* mod 512).

Для исходных сообщений длиной в 0, 484, 512, 1000, 2000 и 16000 битов длина хешируемого сообщения после предварительной стадии будет составлять соответственно: 512, 960, 1024, 2048, 4096 и 32768 битов (при условии, что используется алгоритм MD с блочным размером 512 бит и длиной хеша 128 бит).

Таким образом, для каждого из значений *L*, мы можем использовать формулу, чтобы рассчитать длину хешируемого сообщения после предварительной стадии и знание размера блока и длины хеша, можно вычислить длину хеш-значения. В данном случае для всех значений *L* длина хеша составит 128 бит.

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

«Лавинный эффект» – это свойство алгоритмов хеширования, при котором небольшие изменения во входных данных приводят к значительным изменениям в выходном хеш-коде. Это означает, что даже незначительные изменения в сообщении приводят к значительно измененному хеш-коду.

Основой «лавинного эффекта» является использование нелинейных операций и сильно нелинейных функций в алгоритмах хеширования, таких как логические операции И, ИЛИ, НЕ и XOR, а также сдвиги и замены битов. В результате, изменение одного бита во входном сообщении может привести к изменению значительного числа битов в выходном хеш-коде.

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

Здесь знак «+» в квадратной фигуре соответствует операции сложения по модулю 232. На каждом этапе 16 раз используются различные операции.

Каждая операция представляет собой нелинейную функцию над тремя из *a*, *b*, *c* и *d*.

Затем она добавляет этот результат к четвертой переменной, подблоку текста *Мj* и константе *ti*.

Далее результат циклически сдвигается вправо на переменное число *s* битов и добавляет результат к одной из переменных *a*, *b*, *c* и *d*. Наконец, результат заменяет одну из этих переменных.

Результатом хеширования *h* является конкатенация последних значений указанных переменных.

На рис. 4 показана схема выполнения одной операции в алгоритме SHA-1. Цикл состоит из **четырех** **этапов по 20 операций** в каждом (в MD5 – 4 этапа по 16 операций в каждом). Каждая операция представляет собой нелинейную функцию над тремя из пяти: *a*, *b*, *c*, *d*, *e*. Сдвиг и сложение – аналогично MD5.

В алгоритме используются следующие четыре константы:

*Kt* = 0x5a827999, при *t* = 0, …, 19,

*Kt* = 0x6ed9eba1, при *t* = 20, …, 39,

*Kt* = 0x8flbbcdc, при *t* = 40, …, 59,

*Kt* = 0xca62c1d6, при *t* = 60, …, 79.

Блок сообщения трансформируется из 16 32-битных слов (от *M0* по *M15*) в 80 32-битных слов (*W0*, …, *W79*) с помощью следующего алгоритма:

*Wt* = *Mt*, при t = 0, …, 15,

*Wt* = (*Wt* – 3⊕ *Wt* – 8⊕ *Wt* – 14 ⊕ *Wt* – 16) << 1, при *t* = 16, …, 79; здесь

*Wt* вычисляется как сумма по модулю (⊕).

После обработки всех 512-битных блоков выходом является 160-битный дайджест сообщения в виде конкатенации последних значений переменных *a*, *b*, *c*, *d*, *e*.

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

Основные структурные и функциональные особенности алгоритма хеширования SHA-3 (Keccak) следующие:

1. Конструкция с порталом: Keccak использует конструкцию с порталом (sponge construction), которая позволяет использовать алгоритм не только для хеширования, но и для аутентификации и шифрования данных.
2. Гибкость размера хеш-кода: Keccak позволяет создавать хеш-коды различных размеров (224, 256, 384 или 512 бит), что делает его универсальным для различных задач хеширования.
3. Использование буфера состояния: Keccak использует большой буфер состояния, который является основной работающей памятью алгоритма. Это позволяет алгоритму быстро и эффективно обрабатывать входные данные.
4. Простые и эффективные операции: Keccak использует простые и эффективные операции, такие как замены битов и перестановки. Это делает алгоритм быстрым и простым в реализации.
5. Высокий уровень безопасности: Keccak был выбран Национальным институтом стандартов и технологий (NIST) как стандарт для алгоритмов хеширования, и он считается одним из самых безопасных алгоритмов хеширования.
6. Высокая параллелизуемость: Keccak обладает высокой параллелизуемостью, что позволяет использовать его на многоядерных процессорах и графических процессорах (GPU).
7. Устойчивость к боковым каналам: Keccak разработан с учетом устойчивости к боковым каналам, что делает его подходящим для использования в криптографических устройствах, таких как умные карты и устройства аутентификации.1.

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

Белорусский государственный стандарт хеширования (СТБ 34.101.77–2016), также известный как «БелТСИПС», является семейством алгоритмов хеширования, который включает в себя четыре версии: БелТСИПС 2012, БелТСИПС 2015, БелТСИПС 2016 и БелТСИПС 2020.

Основные структурные и функциональные особенности БелТСИПС:

* Размер хеша составляет 256 бит.
* Базируется на комбинации преобразований Фейстеля и замены Перестановкой-Заменой.
* Размер входного сообщения не ограничен.
* Используется несколько ключей различной длины для усиления стойкости алгоритма.
* Доступен как с общим, так и с секретным ключом.