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

Выполнила:

Студентка 3 курса 1 группы ФИТ

Шимчёнок Елизавета Константиновна

**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**

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.

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

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

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.

**Ход работы**

Данный алгоритм имеет 64 итерации в одном цикле, принимает на вход сообщения предельно допустимого размера до 264 бит, и выдает на выходе дайджест размером 32 байт. *SHA*256 представляет собой однонаправленную функцию.

Программная реализация алгоритма в разработанном ПС осуществляется с помощью встроенной *C*# библиотеки класса *System*.*Security*.*Cryptography*. Рассмотрим подробнее алгоритм хеширования.

Во-первых, необходимо сгенерировать так называемую «соль» указанной длины – строка данных, которая позже будет передана хэш-функции вместе с входным массивом данных (прообразом) для вычисления хэша. Программная реализация генерации соли продемонстрирована на рисунке 1.

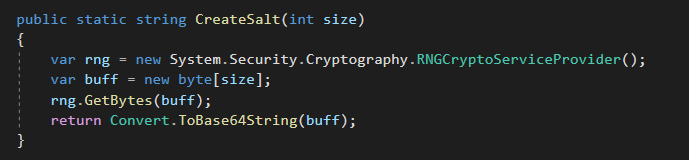


Рис. 1 – Генерация соли

*RNG* (*random number generator*) возвращает случайную последовательность чисел, которую мы конвертируем в строку *base*64. Соль используется нами для усложнения определения прообраза хэш-функции методом перебора по словарю, таким образом максимально увеличивает криптостойкость и уменьшает возможность коллизии. В данном случае использована динамическая соль, что является еще более эффективным с точки зрения профилактики атак.

Далее вызывается функция генерации хэша с помощью алгоритма *SHA*256, встроенная в библиотеку. Входными параметрами данной функции являются вычисленная на предыдущем шаге «соль» и входное сообщение. Программная реализация данной функции продемонстрирована на рисунке 2.

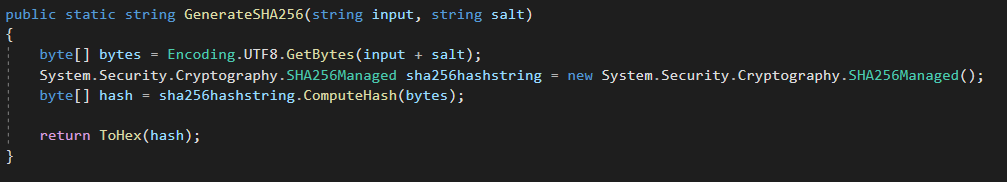


Рис. 2 – Реализация функции хэширования

В данном случае использована встроенная функция *SHA*256*Managed*(), позволяющая вычислить хэш. После получения результата функции, преобразуем его в 16-ричное представление с помощью разработанной функции *ToHex*().

Рассмотрим и проанализируем результаты выполнения выше указанных функций. Результат запуска консольного приложения представлен на рисунке 3.

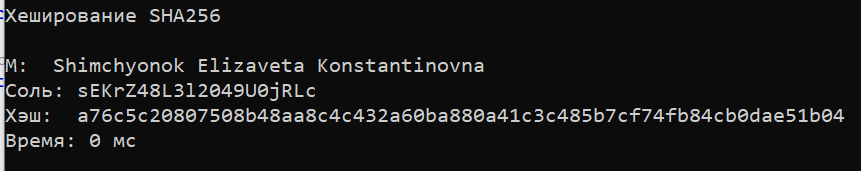


Рис. 3 – Результат работы приложения

Проанализировав время выполнения операции хеширования, можно предположить, что алгоритм *SHA*256 достаточно производительный. Это подтверждает то, что в протоколе указана максимально достигаемая скорость примерно 140 МБ/с. Ведь, если процесс вычисления хеша недостаточно быстрый, система просто не будет эффективной.

**Вывод:** в ходе лабораторной работы были изучены алгоритмы хеширования и приобретены практические навыки их реализации и использования в криптографии. Также была оценена скорость вычисления кодов хеш-функции *SHA*256.

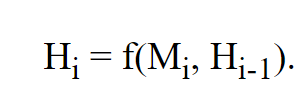
**Ответы на вопросы**

1. Дать определение хеш-функции.

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.

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

Большинство хэш-функций строится на основе однонаправленной функции *f*(•), которая образует выходное значение длиной n при задании двух входных значений длиной п. Этими входами являются блок исходного текста М, и хэш-значение *Ні*-1 предыдущего блока текста (рис. 4):



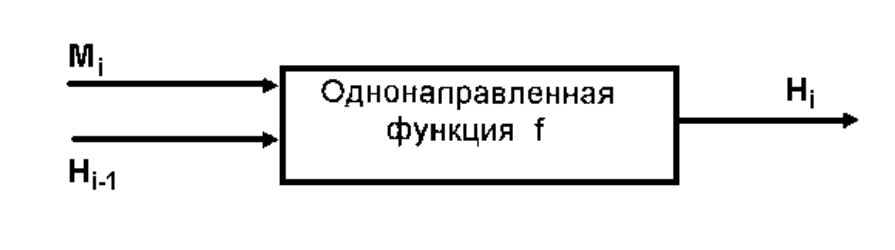


Рис. 4 – Построение однонаправленной хеш-функции

В результате однонаправленная хэш-функция всегда формирует выход фиксированной длины n (независимо от длины входного текста).

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

Коллизия возникает, когда более одного значения, которое должно быть хешировано конкретной хеш-функцией, хешируется в один и тот же слот в таблице или структуре данных (хеш-таблице), сгенерированной хеш-функцией.

Хеш-функции могут сталкиваться с различными типами коллизий, которые определяются в основном двумя факторами: структурой хеш-функции и свойствами входных данных. Вот некоторые из типов коллизий, с которыми можно столкнуться при использовании хеш-функций:

Случайная коллизия – тип коллизии, когда два разных входных значения создают одинаковый хеш-код случайным образом. Такие коллизии неизбежны в случае конечного пространства возможных хеш-кодов и бесконечного пространства входных данных.

Постепенная коллизия (англ. *gradual collision*) – тип коллизии, который возникает при изменении одного или нескольких битов входного значения. Постепенные коллизии могут быть проблемой в некоторых криптографических приложениях, где даже незначительное изменение входных данных должно приводить к существенному изменению хеш-кода.

Деловая коллизия (англ. *birthday collision*) – тип коллизии возникает, когда два различных входных значения создают одинаковый хеш-код. Название "деловая коллизия" происходит от парадокса дней рождения: вероятность того, что у двух людей в группе есть одинаковая дата рождения, оказывается выше, чем может показаться на первый взгляд. В хеш-функциях это связано с парадоксальной вероятностью совпадения хеш-кодов.

Частичная коллизия – тип коллизии, при котором два различных входных значения дают одинаковый фрагмент хеш-кода, но остальная часть хеш-кода различна. Частичные коллизии могут возникать из-за особенностей структуры хеш-функции или из-за свойств входных данных.

Алгебраическая коллизия – тип коллизии, при котором два различных входных значения дают одинаковый результат операций, выполняемых хеш-функцией. Например, если хеш-функция основана на полиномиальных операциях, алгебраическая коллизия может возникнуть, когда два различных полинома дадут одинаковый результат.

1. Сформулировать в общем виде парадокс «дней рождений».

Парадокс "дней рождений" заключается в том, что вероятность такого события оказывается выше, чем может показаться на первый взгляд. Несмотря на то, что количество возможных дат рождения составляет 365 (или 366 в случае високосного года), вероятность наличия совпадений становится значительной уже при небольшом числе людей.

Формула для вычисления вероятности совпадения дат рождений в группе из *n* человек может быть сформулирована следующим образом:

*P*(*n*) = 1 - (365/365) \* (364/365) \* (363/365) \* ... \* ((365 - *n* + 1)/365),

где *P*(*n*) - вероятность наличия совпадения дат рождений при *n* людях.

Этот парадокс иллюстрирует неожиданную зависимость между числом участников и вероятностью совпадения, которая объясняется принципом Дирихле и связана с комбинаторными свойствами вероятности.

1. Как парадокс «дней рождений» используется в криптографии?

Парадокс "дней рождений" имеет важное применение в криптографии в контексте атаки на хеш-функции, известной как атака дней рождений (*birthday* *attack*).

Атака дней рождений используется для нахождения коллизий в хеш-функциях, то есть поиска двух различных входных значений, которые дают одинаковый хеш-код. Эта атака основана на том, что вероятность наличия коллизии в хеш-функции растет с увеличением числа входных значений.

Используя парадокс "дней рождений", атака дней рождений позволяет эффективно искать коллизии в хеш-функциях, даже если пространство хеш-кодов очень велико. Вместо того чтобы искать коллизии напрямую путем сравнения всех возможных пар входных значений, атака дней рождений основывается на вероятностном подходе и использовании статистических свойств парадокса "дней рождений".

Например, предположим, что хеш-функция возвращает хеш-коды длиной 128 бит. Парадокс "дней рождений" показывает, что вероятность того, что два разных входных значения дают одинаковый хеш-код, становится значительной, когда количество входных значений приближается к 264, что значительно меньше, чем 2128 - общее количество возможных хеш-кодов. Таким образом, используя атаку дней рождений, злоумышленник может эффективно искать коллизии в хеш-функции, выполняя относительно небольшое количество операций.

Для защиты от атаки дней рождений криптографические хеш-функции обычно имеют достаточно большую длину хеш-кода и другие безопасностные меры, такие как соли (*salt*) и рандомизация, чтобы усложнить поиск коллизий методом атаки дней рождений.

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

Для определения количества попыток, необходимых для обнаружения коллизии с заданной вероятностью, можно использовать формулу, основанную на парадоксе "дней рождений". Формула выглядит следующим образом:

*N* = *sqrt*(2 \* *M* \* *ln*(1 / (1 - *P*))),

где: *N* - количество попыток, *M* - количество возможных хеш-кодов (2l), *P* - заданная вероятность обнаружения коллизии.

Давайте рассчитаем количество попыток для разных значений длины хеша (l) и заданных вероятностей (0,5; 0,7; 0,8; 0,9):

Длина хеша l = 64 бита:

Для вероятности 0,5:

*N* = *sqrt*(2 \* 2^64 \* *ln*(1 / (1 - 0,5))) ≈ 2^32.5 ≈ 67 108 864;

Для вероятности 0,7:

*N* = *sqrt*(2 \* 2^64 \* *ln*(1 / (1 - 0,7))) ≈ 2^34.2 ≈ 475 920 315;

Для вероятности 0,8:

*N* = *sqrt*(2 \* 2^64 \* *ln*(1 / (1 - 0,8))) ≈ 2^35.1 ≈ 1 214 104 137;

Для вероятности 0,9:

*N* = *sqrt*(2 \* 2^64 \* *ln*(1 / (1 - 0,9))) ≈ 2^36 ≈ 3 289 868 133.

Длина хеша l = 128 бит:

Для вероятности 0,5:

*N* = *sqrt*(2 \* 2^128 \* *ln*(1 / (1 - 0,5))) ≈ 2^64 ≈ 18 446 744 073 709 551 616;

Для вероятности 0,7:

*N* = *sqrt*(2 \* 2^128 \* *ln*(1 / (1 - 0,7))) ≈ 2^66.8 ≈ 2 193 655 399 452 037 808 151;

Для вероятности 0,8:

*N* = *sqrt*(2 \* 2^128 \* *ln*(1 / (1 - 0,8))) ≈ 2^67.8 ≈ 5 570 708 733 983 669 282 067;

Для вероятности 0,9:

*N* = *sqrt*(2 \* 2^128 \* *ln*(1 / (1 - 0,9))) ≈ 2^68.7 ≈ 15 036 724 010 740 843 393 814.

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

Семейства хеш-функций *MD* и *SHA* являются широко используемыми алгоритмами хеширования в криптографии. Вот общая характеристика каждого из этих семейств:

Семейство *MD* (*Message Digest*): *MD* является семейством хеш-функций, разработанных Рональдом Ривестом. В настоящее время наиболее известной и широко применяемой вариацией является *MD*5. *MD*-функции обеспечивают довольно быструю обработку и обладают низкими вычислительными требованиями. Однако изначальный *MD*5 стал уязвимым к коллизиям, и его использование в криптографических целях не рекомендуется.

Семейство *SHA* (*Secure Hash Algorithm*): *SHA* является семейством хеш-функций, разработанных Национальным институтом стандартов и технологии (*NIST*) в США. Наиболее широко используемыми версиями *SHA* являются *SHA*-1, *SHA*-256, *SHA*-384 и *SHA*-512. Они обеспечивают более надежное хеширование и рекомендуются для использования в криптографических приложениях. Варианты *SHA*, такие как *SHA*-256, используют более длинные хеш-коды и обеспечивают более высокий уровень безопасности.

Относительно основных стадий алгоритма хеширования сообщения, они обычно включают следующие шаги:

* Инициализируется начальное состояние хеш-функции.
* Сообщение разбивается на блоки фиксированного размера, которые последовательно обрабатываются хеш-функцией. Каждый блок обрабатывается вместе с предыдущим состоянием хеш-функции.
* Если сообщение не является кратным размеру блока, оно дополняется для обеспечения полной обработки. Обычно используются различные схемы дополнения, чтобы обеспечить целостность данных.
* После обработки всех блоков выполняется окончательное вычисление хеш-кода на основе конечного состояния хеш-функции.
* Получение хеш-кода.

1. Рассчитать общую длину (*L*') хешируемого сообщения после предварительной стадии на основе алгоритма *MD*, если объем (*L*) исходного сообщения составлял 512 битов. Какова 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).

1. Входное сообщение состоит из вашего имени. Используя представление сообщения в кодах *ASCII*, представить в табличной форме содержание каждого 32-битного подблока расширенного входного сообщения.

*M* = «*Elizaveta*».

*L* =9 \* 8 = 72.

Первый блок – само сообщение в двоичном виде.

Одну единицу запишем во второй блок и 512 – 64 – 72 – 1 = 375 нулей запишем во второй блок.

Третий блок – 64 – 8 = 56 нулей и число 72 в двоичном виде.

|  |  |  |
| --- | --- | --- |
| 100010111011001101001111101011000011110110110010111101001100001 | 1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 | 000000000000000000000000000000000000000000000000000000001001000 |

1. Представить и охарактеризовать структурную схему одного раунда алгоритмов хеширования на основе *MD*5.

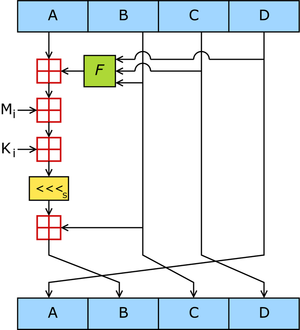


Рис. 5 – Раунд хэширования на основе *MD*5

Инициализируются четыре 32-битных регистра, обозначаемых *A*, *B*, *C* и *D*, начальными значениями.

Входной блок данных размером 512 бит разбивается на 16 слов, каждое из которых состоит из 32 бит.

Выполняются последовательные операции для каждого из 16 слов:

– выполняется логическое ИЛИ (*XOR*) между словом и соответствующей константой;

– выполняется циклический сдвиг влево (циклический сдвиг битов) на определенное количество битов;

– прибавляется значение регистра *A* к слову.

Циклические операции и обновление состояния:

– выполняется четыре циклические операции, обновляющие значения регистров *A*, *B*, *C* и *D*: *A = D*; *D = C*; *C = B*; *B = B* + ((*A* + *F*(*B*, *C*, *D*) + слово + константа) <<< сдвиг).

<<< обозначает циклический сдвиг влево.

*F*(*B*, *C*, *D*) - функция, которая зависит от текущих значений *B*, *C* и *D* и выполняет определенные логические операции.

После выполнения всех раундов алгоритма, значения регистров *A*, *B*, *C* и *D* объединяются для формирования итогового хеш-кода.

Структурная схема одного раунда в алгоритме *MD*5 (рис. 5) показывает основные шаги обработки входных данных и обновления состояния регистров. Циклические операции обеспечивают диффузию и сложность вычислений, а константы и функции вносят нелинейность в алгоритм, чтобы обеспечить стойкость к атакам.

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

"Лавинный эффект" в алгоритмах хеширования означает, что даже небольшие изменения во входных данных должны приводить к значительным изменениям в выходных хеш-кодах. Это свойство обеспечивает диффузию (распространение) изменений по всему хешу и является важным свойством хеш-функций в контексте криптографии.

Основная цель реализации лавинного эффекта в алгоритмах хеширования состоит в обеспечении стойкости к различным атакам. Если малейшие изменения во входных данных несут с собой значительные изменения в хеш-кодах, то это делает атаки на хеш-функцию более сложными.

Существуют два типа атак, которые могут быть затруднены благодаря лавинному эффекту:

– коллизионные атаки;

– предобразования.

Целью коллизионных атак является нахождение двух разных входных сообщений, которые приводят к одному и тому же хеш-коду. Лавинный эффект усложняет поиск коллизий, так как небольшие изменения во входных данных должны приводить к существенным изменениям в хеш-коде.

Предобразования направлены на поиск входных данных, которые после хеширования приводят к желаемому хеш-коду. Лавинный эффект затрудняет эти атаки, поскольку изменения во входных данных равномерно распространяются по всему хешу, делая предсказание желаемого хеш-кода сложным.

Таким образом, реализация лавинного эффекта в алгоритмах хеширования повышает их безопасность и стойкость к различным атакам, обеспечивая равномерное распространение изменений и усложняя обратные вычисления, необходимые для поиска коллизий или предобразований.

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

Конструкция с порабощенной схемой спонсирования (sponge construction): *SHA*-3 основан на концепции спонж-конструкции, которая позволяет гибко использовать алгоритм не только для хеширования, но и для других задач, таких как генерация случайных чисел или аутентифицированное шифрование. Это делает *SHA*-3 универсальным и гибким инструментом.

Использование криптографической сетки (cryptographic sponge construction): SHA-3 использует криптографическую сетку, которая состоит из состояния и функции поглощения (absorbing function) и функции выжимания (squeezing function). Функция поглощения принимает входные данные и поглощает их в состояние сетки, а функция выжимания извлекает хеш-коды из состояния сетки.

Гибкий выбор длины хеша: *SHA*-3 позволяет выбирать длину хеша в зависимости от потребностей приложения. Длина хеша может быть 224, 256, 384 или 512 битов.

*SHA*-3 разработан с учетом современных криптографических требований и имеет высокую стойкость к различным атакам, включая коллизионные атаки и предобразования.

*SHA*-3 обладает высокой скоростью обработки данных и хорошо адаптирован к современным процессорам и аппаратной реализации. Это делает его эффективным для широкого спектра приложений, включая криптографическую защиту данных, цифровые подписи, аутентификацию и другие.

*SHA*-3 является одним из криптографических стандартов, рекомендованных *NIST*, и получил широкое признание в криптографическом сообществе.

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

Белорусский государственный стандарт хеширования (СТБ 34.101.77–2016) описывает алгоритмы хеширования, разработанные для использования в информационной безопасности. Вот основные характеристики этого стандарта:

* Структурные особенности:

СТБ 34.101.77–2016 включает в себя несколько алгоритмов хеширования, включая стандарты, такие как SHA-256, SHA-384 и SHA-512.

Алгоритмы хеширования, описанные в стандарте, основаны на конструкции Меркла-Дамгора, которая обеспечивает эффективное и безопасное хеширование данных.

Стандарт также определяет параметры и рекомендации для использования алгоритмов хеширования в различных сценариях и приложениях.

* Функциональные особенности:

Алгоритмы хеширования, описанные в СТБ 34.101.77–2016, обеспечивают высокую стойкость к криптографическим атакам, включая коллизионные атаки и предобразования.

Они предоставляют возможность генерации фиксированной длины хеш-кода для произвольных входных данных.

СТБ 34.101.77–2016 также обеспечивает поддержку верификации хеш-кодов и проверку целостности данных.

* Криптостойкость:

СТБ 34.101.77–2016 основывается на криптографических алгоритмах, таких как SHA-256, SHA-384 и SHA-512, которые являются широко принятыми и изученными в криптографическом сообществе.

Криптостойкость этих алгоритмов основана на сложности обратных вычислений, то есть на то, что вычисление входного сообщения по заданному хеш-коду является вычислительно неосуществимой задачей.

Однако следует отметить, что с течением времени появляются новые атаки и методы взлома, поэтому необходимо постоянно обновлять алгоритмы хеширования, чтобы они соответствовали современным требованиям к безопасности.

**Вывод:** в ходе лабораторной работы были приобретены практические навыки разработки и использования приложений для реализации асимметричных шифров RSA и Эль-Гамаля. Было разработано приложение для реализации методов генерации ключевой информации и ее использования. Также была оценена скорость зашифрования/расшифрования.