

Молдавский Государственный Университет
Факультет Математики и Информатики
Департамент Информатики

**Лабораторная работа №2 и
практические задания**
по курсу “Криптография”

Выполнил: Slavov Constantin,
группа I2302
Проверила: Olga Cerbu, doctor,
conferențiar universitar

Кишинев, 2024

Лабораторная работа №2. Алгоритм хеширования RIPEMD-160, его принципы работы и код к нему.

RIPEMD-160 (от англ. RACE Integrity Primitives Evaluation Message Digest) — криптографическая хеш-функция, разработанная в Католическом университете Лувена Хансом Доббертином, Антоном Босселарсом и Бартом Пренелем.

Алгоритм появился на свет как результат европейского проекта RIPE (RACE Integrity Primitives Evaluation), который был направлен на разработку новых криптографических примитивов, включая хеш-функции. Проект был создан в ответ на растущую потребность в безопасных алгоритмах для обеспечения целостности данных и защиты от атак. RIPEMD-160 является улучшенной версией оригинальной хеш-функции RIPEMD, которая имела 128-битный вывод. Хотя RIPEMD-128 был надежным, исследователи посчитали, что увеличение длины хеша до 160 бит обеспечит более высокую степень безопасности, делая алгоритм устойчивым к более сложным атакам. Сегодня RIPEMD-160 остается одним из ключевых инструментов для использования в различных криптографических системах.

- Техническая структура

Архитектура RIPEMD-160 примечательна своей уникальной двухпоточной структурой. Это означает, что при обработке данных алгоритм использует две параллельные линии вычислений, что повышает его устойчивость к различным видам атак, включая атаки на целостность данных и коллизионные атаки. Каждая из двух линий данных проходит через пять раундов, по 16 операций в каждом раунде. Эти раунды включают такие действия, как побитовые операции, циклические сдвиги и логические комбинации данных. Основной целью каждой операции является создание уникального хеш-значения на основе введенных данных, чтобы любые малейшие изменения входных данных существенно изменяли итоговый хеш.

- Применение в области безопасности

RIPEMD-160 используется в разнообразных приложениях, от проверок целостности файлов до генерации криптовалютных адресов. Одним из самых известных примеров использования RIPEMD-160 является его интеграция в экосистему **Bitcoin**. В этой системе он играет ключевую роль в создании публичных адресов, обеспечивая их уникальность и невозможность восстановления оригинальных данных по хешу. В сочетании с алгоритмом SHA-256, RIPEMD-160 используется для повышения безопасности криптовалютных транзакций и защиты кошельков пользователей. Помимо криптовалют, алгоритм находит применение в более традиционных системах аутентификации и проверки данных, что делает его универсальным инструментом в области кибербезопасности.

- Сравнение с другими хеш-функциями

Когда речь идет о сравнении RIPEMD-160 с другими хеш-функциями, его часто сравнивают с такими стандартами, как SHA-256. Основное отличие между ними заключается в длине выходного хеша: SHA-256 генерирует 256-битный вывод, тогда как RIPEMD-160 — только 160-битный. Это означает, что SHA-256 имеет более высокую стойкость к атакам типа "коллизия", где два различных набора данных могут привести к одинаковому хешу. Однако в ряде случаев RIPEMD-160 предпочтителен из-за его меньшего размера хеша, что позволяет снизить нагрузку на систему и ускорить процесс хеширования. Тем не менее, несмотря на преимущества, многие эксперты считают SHA-256 более безопасным для долгосрочного использования, особенно в условиях роста вычислительных мощностей и развития криптоанализа.

- Потенциальные уязвимости и критика

Хотя RIPEMD-160 и остается стойким к большинству известных атак, его более короткий хеш делает его потенциально уязвимым для будущих угроз. В условиях роста вычислительных мощностей и развития методов анализа криптографических систем, существует вероятность, что атаки на коллизии могут стать более эффективными против RIPEMD-160. Это особенно важно в контексте долгосрочных приложений, где требуется высокая степень безопасности данных на протяжении десятилетий. Некоторые системы уже начали заменять RIPEMD-160 более новыми алгоритмами, такими как SHA-3, которые предлагают более высокий уровень защиты благодаря увеличенной длине хеша.

- Современное использование RIPEMD-160

Сегодня RIPEMD-160 продолжает использоваться в криптовалютах, в первую очередь в системе Bitcoin, где он применяется для создания адресов и защиты транзакций. Однако с развитием технологии блокчейна и появлением новых хеш-функций, таких как SHA-3, его использование постепенно снижается. В финансовых приложениях и в тех системах, где требуются повышенные уровни безопасности, часто предпочитают более новые и сложные хеш-функции, что снижает популярность RIPEMD-160. Тем не менее, он остается достаточно эффективным для многих задач, особенно там, где размер хеша играет ключевую роль для экономии вычислительных ресурсов.

- Заключение

RIPEMD-160 продолжает оставаться важным криптографическим инструментом для множества приложений, особенно в тех случаях, когда важен компромисс между скоростью и безопасностью. Его уникальная двухпоточная архитектура и способность генерировать относительно короткие, но надежные хеш-значения делают его предпочтительным выбором для задач с ограниченными ресурсами. Однако будущее этого алгоритма зависит от того, как быстро будут развиваться атаки на коллизии и как системы защиты данных будут адаптироваться к новым угрозам. В перспективе RIPEMD-160, вероятно, будет заменен более сложными и безопасными алгоритмами в тех областях, где требуется максимальная защита данных.

Демонстрация кода на Java, который шифрует данные при помощи алгоритма RIPEMD-160 и объяснение к нему.

Код:

```
import org.bouncycastle.jce.provider.BouncyCastleProvider;

import java.security.MessageDigest;

import java.security.Security;

public class RipeMd160 {

    public static void main(String[] args) {

        Security.addProvider(new BouncyCastleProvider());

        String input = "RIPEMD160 Test Message";

        System.out.println("Исходное сообщение: " + input);

        try {

            MessageDigest md =
            MessageDigest.getInstance("RIPEMD160", "BC");

            byte[] hash = md.digest(input.getBytes());

            System.out.print("Зашифрованное сообщение (хеш): ");

            for (byte b : hash) {

                System.out.printf("%02x", b);

                }

            } catch (Exception e) {

                e.printStackTrace();

                }

        }

    }
```

Что делает данный код:

1. Добавление BouncyCastle как криптографического провайдера:

```
Security.addProvider(new BouncyCastleProvider());
```

Этот шаг добавляет BouncyCastle как дополнительный провайдер криптографических алгоритмов в Java. BouncyCastle предоставляет поддержку для алгоритмов, которые не входят в стандартную библиотеку Java, таких как RIPEMD-160.

2. Определение исходного сообщения:

```
String input = "RIPEMD160 Test Message";  
  
System.out.println("Исходное сообщение: " + input);
```

В этой строке задается сообщение, которое будет хешировано, и выводится в консоль исходный текст, чтобы пользователь мог видеть, какие данные используются для хеширования.

3. Создание объекта MessageDigest для RIPEMD-160:

```
MessageDigest md = MessageDigest.getInstance("RIPEMD160",  
"BC");
```

Здесь создается объект `MessageDigest`, который реализует алгоритм хеширования RIPEMD-160 с использованием провайдера BouncyCastle ("BC"). Этот объект выполняет основную работу по хешированию.

4. Выполнение хеширования:

```
byte[] hash = md.digest(input.getBytes());
```

Метод `digest()` хеширует строку, передавая её как массив байтов. В результате получается массив байтов длиной 20 байт (160 бит), который является хешем сообщения.

5. Вывод хеша в шестнадцатеричном формате:

```
System.out.print("Зашифрованное сообщение (хеш): ");  
  
for (byte b : hash) {  
  
    System.out.printf("%02x", b);  
  
}
```

Этот код выводит хеш в удобочитаемом шестнадцатеричном формате. Цикл `for` проходит по каждому байту массива хеша и выводит его как двузначное шестнадцатеричное число (`%02x`), чтобы результат можно было легко прочитать и сравнить.

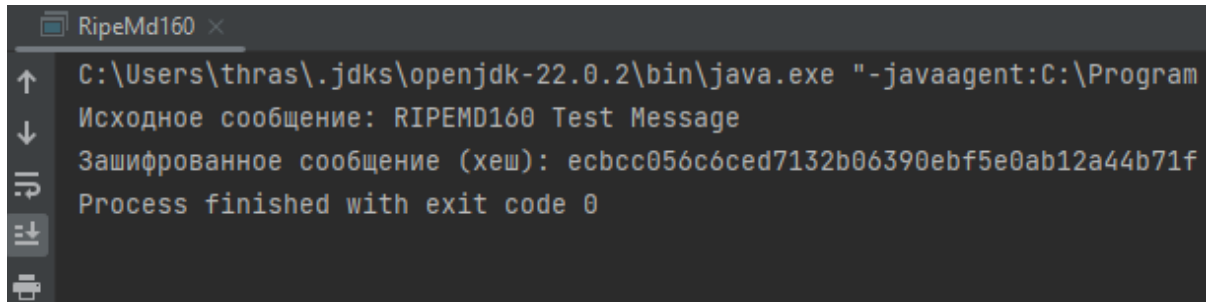
6. Обработка исключений:

```
} catch (Exception e) {  
  
    e.printStackTrace();  
  
}
```

Этот блок ловит любые исключения, которые могут возникнуть во время работы программы (например, если алгоритм не поддерживается) и выводит стек ошибок для отладки.

Если коротко, то этот код использует библиотеку BouncyCastle для вычисления хеша строки с помощью алгоритма RIPEMD-160 и выводит результат в виде шестнадцатеричной строки.

Вывод на экран результата выполнения программы:



```
RipeMd160 x
C:\Users\thras\.jdk\openjdk-22.0.2\bin\java.exe "-javaagent:C:\Program
Исходное сообщение: RIPEMD160 Test Message
Зашифрованное сообщение (хеш): ecbcc056c6ced7132b06390ebf5e0ab12a44b71f
Process finished with exit code 0
```

Выводы по лабораторной работе: В данной лабораторной работе была реализована программа для хеширования сообщений с использованием алгоритма RIPEMD-160 с помощью библиотеки BouncyCastle. Основная цель работы заключалась в изучении работы криптографических алгоритмов и их практического применения в коде на Java. В ходе выполнения лабораторной работы:

1. Было продемонстрировано добавление криптографического провайдера BouncyCastle для расширения стандартных возможностей Java.
2. Осуществлено хеширование строки с использованием алгоритма RIPEMD-160, который генерирует 160-битный хеш.
3. Программа вывела как исходное сообщение, так и зашифрованный хеш в удобочитаемом шестнадцатеричном формате.

Эта работа позволила лучше понять принципы работы хеш-функций, их применение для обеспечения целостности данных, а также познакомиться с библиотеками для криптографических задач.