МОЛДАВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет Математики и Информатики

Департамент Информатики

Криптография

Аттестация 1

Проверил:Чербу Ольга
Выполнил: Главчева Марина, IA2303

Кишинев, 2024

RIPEMD-160

RIPEMD-160 (RACE Integrity Primitives Evaluation Message Digest) — это криптографический хеш-алгоритм, созданный для формирования хешей фиксированной длины из произвольных данных. Он разработан Хансом Доббертином, Антоном Босселарсом и Бартом Пренелем в 1996 году на основе более раннего алгоритма RIPEMD и семейства MD4.

Характеристики RIPEMD-160:

- **Длина хеша:** 160 бит (20 байт), что делает его более устойчивым к атакам по сравнению с алгоритмами с более короткими хешами, например, MD5 (128 бит).

- **Безопасность:** Считается более безопасным, чем старые алгоритмы, такие как MD5 и RIPEMD. Уровень безопасности аналогичен SHA-1, но в отличие от SHA-1, серьёзных уязвимостей в RIPEMD-160 не обнаружено.

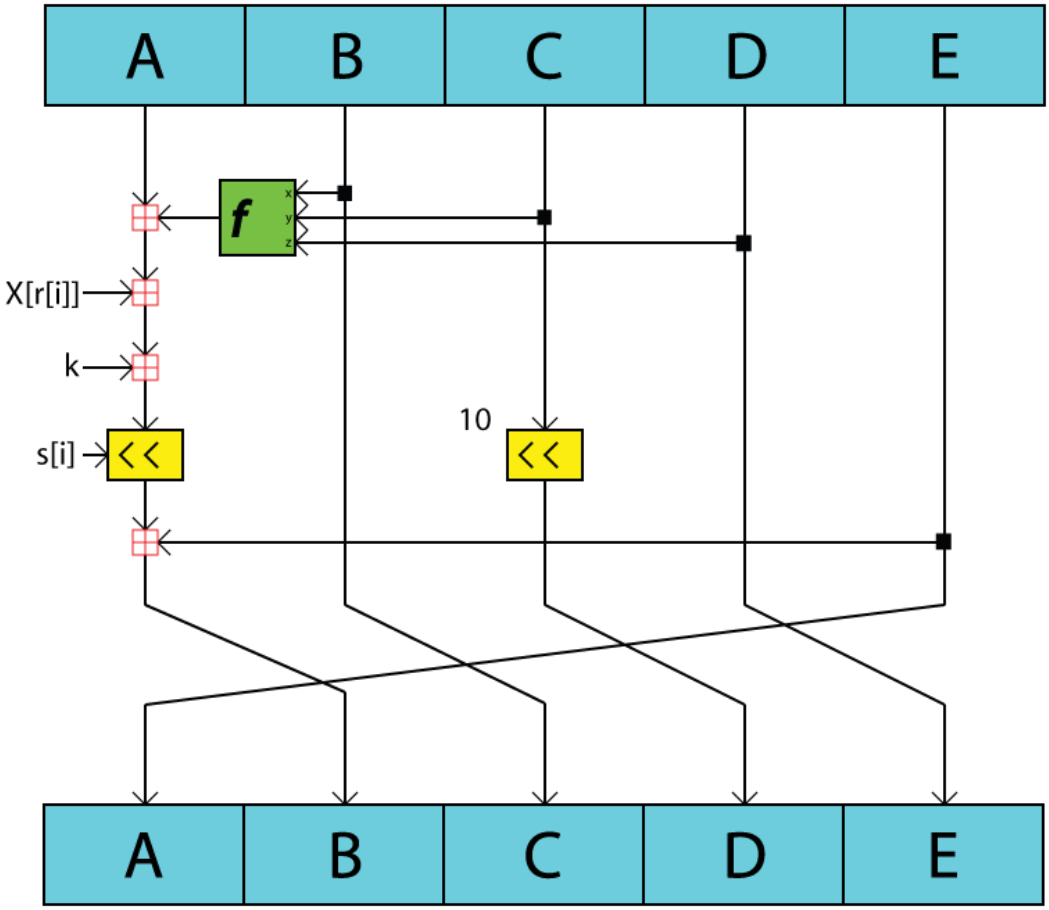
- **Структура:** Алгоритм использует две параллельные версии хеш-функции, каждая из которых работает с пятью 32-битными регистрами. Эти версии обрабатывают данные независимо и затем объединяют результаты, что повышает устойчивость к коллизиям (разные данные приводят к одинаковому хешу).

- **Скорость:** Работает немного медленнее, чем SHA-1, но более устойчив к криптоанализу.

- **Применение:**
  - Используется для цифровых подписей и проверки целостности данных.
  - Применяется в блокчейне, особенно в Bitcoin, где RIPEMD-160 комбинируется с SHA-256 для создания криптографических адресов.

- **Алгоритмическая основа**: Разработан на основе MD4, но имеет более сложную структуру для повышения криптографической стойкости. В отличие от MD5 и MD4, которые уже скомпрометированы, RIPEMD-160 остаётся надёжным.

- **Варианты:** Помимо RIPEMD-160, существуют версии RIPEMD-128, RIPEMD-256 и RIPEMD-320, но они не обеспечивают значительного улучшения безопасности по сравнению с RIPEMD-160.



Хеширование слова "algoritm" с использованием алгоритма RIPEMD-160

Шаг 1: Подготовка сообщения

Слово "algoritm" в 16-ричной системе:
a = 0x61, l = 0x6C, g = 0x67, o = 0x6F, r = 0x72, i = 0x69, t = 0x74, m = 0x6D

Включение ключа MG в сообщение

Один из простых подходов заключается в том, чтобы включить ключ непосредственно в сообщение перед его хешированием.

M = 0x4D, G = 0x47
Тогда сообщение может стать:
MG + "algoritm"

В 16-ричной системе:
MG = 0x4D47
algoritm = 0x616C676F7269746D

Сообщение становится:
0x4D47616C676F7269746D

Для хеширования сообщение сначала дополнено битами по определенным правилам:

1. Добавляем единичный бит (1) к сообщению.
2. Дополняем сообщение нулями, чтобы его длина достигла 448 бит.
3. Добавляем 64-битное представление длины исходного сообщения (8 байт = 64 бита).

Сообщение после добавления битов и дополнения:

Сообщение:
4D47616C 676F7269 746D8000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000050
Теперь сообщение занимает 512 бит (64 байта).

Шаг 2: Инициализация переменных

Начальные значения хеш-суммы (в 16-ричной системе):
h0 = 0x67452301
h1 = 0xEFCDAB89
h2 = 0x98BADCFE
h3 = 0x10325476
h4 = 0xC3D2E1F0

Шаг 3: Первый цикл

Затем процесс продолжается, как описано ранее, с функцией для каждого шага.

Для первого шага:

- W[0] = 0x4D47616C
- W[1] = 0x676F7269
- W[2] = 0x746D8000
- И так далее...
- W[15] = 0x00000050

Процесс расчета функции f(j, B, C, D) и обновления переменных происходит аналогично предыдущему примеру.

Функция для j = 0:
f(0, B, C, D) = B ⊕ C ⊕ D
Подставляем начальные значения:
B = 0xEFCDAB89
C = 0x98BADCFE
D = 0x10325476

Выполняем операцию XOR:

f(0, B, C, D) = 0xEFCDAB89 ⊕ 0x98BADCFE ⊕ 0x10325476
= 0x7F024B97

Шаг 4: Вычисляем промежуточное значение T

Теперь мы вычисляем промежуточное значение T, добавив f(0), первый блок сообщения W[0]W[0]W[0] и соответствующую константу K(0)(которая равна 0 для первого раунда):

T = A + f(0) + W[0] + K(0)
= 0x67452301 + 0x7F024B97 + 0x4D47616C + 0x00000000

T = 0x67452301 + 0x7F024B97 + 0x4D47616C
= 0x1441C0F04

Шаг 5: Остаток по модулю 2^32

Теперь берём остаток от деления суммы T на 2^32, чтобы получить 32-битное значение:

T = 0x1441C0F04 * 2^32= 0x441C0F04

Шаг 6: Циклический сдвиг

rols(11, T) = rols(11, 0x441C0F04) = 0xE079A220

Шаг 7: Обновление данных

Теперь мы обновляем значения переменных:

1. A принимает значение E (то есть A=E=0xC3D2E1F0)
2. E принимает значение D (то есть E=D=0x10325476)
3. D получает результат циклического сдвига значения C на 10 позиций:

   D = rol10(C) = rol10(0xeb73fa62) = 0xeb73fa62

4. C принимает значение B (то есть C=B=0xEFCDAB89)
5. B принимает значение T, вычисленного на шаге 6 (то есть B=T= 0x793e6b6f)

Теперь обновлённые значения переменных:

A = 0xC3D2E1F0

B = 0x793E6B6F

C = 0xEFCDAB89

D = 0xF5B5793D

E = 0x10325476

Этот процесс продолжается для всех 80 шагов. Для каждого шага подставляются следующие блоки сообщения W[j]], соответствующие функции f(j,B,C,D), и обновляются значения переменных A, B, C, D, E. После завершения всех раундов, значения переменных складываются с начальными значениями h0,h1,h2,h3,h4и результат хеширования возвращается.

Код на Python

```
def rol(value, shift, bits=32):
    """Циклический сдвиг влево на shift бит."""
    return ((value << shift) | (value >> (bits - shift))) & 0xFFFFFFFF

def ripemd160_first_cycle(message):
    # Шаг 1. Инициализация начальных значений (константы для h0-h4)
    h0 = 0x67452301
    h1 = 0xEFCDAB89
    h2 = 0x98BADCFE
    h3 = 0x10325476
    h4 = 0xC3D2E1F0

    # Шаг 2. Подготовка сообщения (слово 'algoritm' -> в 16-ричной форме)
    M = [0x616C676F, 0x7269746D, 0x80000000] + [0x00000000] * 12 + [0x00000040]

    # Константа для первого шага
    K = 0x00000000

    # Нелинейная функция для первого диапазона j (0 ≤ j ≤ 15): f(j; B; C; D) = B
⊕ C ⊕ D
```

```
    def f(j, B, C, D):
        return B ^ C ^ D

    # Начальные значения
    A = h0
    B = h1
    C = h2
    D = h3
    E = h4

    # Выполнение первой итерации (j = 0)
    j = 0
    T = rol(A + f(j, B, C, D) + M[j] + K, 11) + E
    A, E, D, C, B = E, D, rol(C, 10), B, T & 0xFFFFFFFF

    # Печать результатов после первого цикла
    print("A:", hex(A))
    print("B:", hex(B))
    print("C:", hex(C))
    print("D:", hex(D))
    print("E:", hex(E))

# Тестируем первый цикл для слова 'algoritm'
ripemd160_first_cycle('algoritm')
```

```
PS C:\Users\marin> & C:/Users/marin/AppData/Local/Microsoft/Windo
wsApps/python3.11.exe c:/Users/marin/Downloads/RIPEMD.py
A: 0xc3d2e1f0
B: 0x793e6b6f
C: 0xefcdab89
D: 0xeb73fa62
E: 0x10325476
PS C:\Users\marin>
```