Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Факультет безопасности информационных технологий

Дисциплина:

«Теория информационной безопасности и методология защиты информации»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №7

«Хэш-функции»

Выполнил:

студент группы N3246,

Суханкулиев Мухаммет

(подпись)

Проверила:

Коржук Виктория Михайловна

(отметка о выполнении)

(подпись)

Санкт-Петербург

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы – исследование семейства хэш-функций и детальное изучение алгоритма последней доступной версии хэш-функции.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Выбрать семейство хэш-функций (SHA, MD или российские аналоги).
2. Изучить историю появления и этапы формирования.
3. Изучить области применения на всех этапах.
4. Подробно теоретически разобрать алгоритм действия последней публично доступной версии выбранной хэш-функции (как смешиваются блоки, как побитовое сложение происходит и т. д.).

1 ХЭШ-ФУНКЦИИ

1.1 Общие сведения

Хэш-функция — это математический алгоритм, преобразующий данные произвольного размера в битовую строку фиксированной длины. Этот процесс называется **хэшированием**, а результат — **хэш-суммой** или **хэшем**.

Хэш-функции широко применяются в защите информации благодаря следующим свойствам:

- **Детерминированность**: одно и то же сообщение всегда даёт одинаковый хэш.
- **Быстрота вычисления**: хэш можно быстро получить даже для больших данных.
- **Однонаправленность**: невозможно восстановить исходное сообщение по хэшу.
- **Устойчивость к коллизиям**: крайне сложно найти два разных сообщения с одинаковым хэшем.
- **Лавинный эффект**: небольшое изменение в исходных данных сильно меняет хэш.

Основные области применения хэш-функций:

1. Проверка целостности данных (например, при передаче файлов).
2. Хранение паролей в виде хэшей для повышения безопасности.
3. Создание цифровых подписей и электронных документов.
4. Построение уникальных идентификаторов для данных.
5. Использование в криптовалютах и блокчейне.

Хэш-функции строятся на основе итеративных схем:

- Входные данные разбиваются на блоки.
- Сжимающая функция последовательно обрабатывает каждый блок, комбинируя его с результатом предыдущих шагов.
- Результатом является хэш фиксированной длины, который зависит от всех входных данных.

SHA-1 (Secure Hash Algorithm 1) – алгоритм, создающий 160-битный хэш. Разработан в 1995 году и использовался в криптографических приложениях. Несмотря на более высокий уровень безопасности по сравнению с MD5, в нем также были обнаружены уязвимости, из-за чего его использование не рекомендуется.

SHA-2 – семейство хэш-функций, включающее варианты с разными длинами хэша (SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/256 и SHA-512/224). Алгоритм обеспечивает более высокий уровень безопасности по сравнению с SHA-1 и MD5.

Например, SHA-256 генерирует 256-битное значение и широко используется для цифровых подписей и сертификатов.

SHA-3 – последняя версия семейства SHA, выпущенная в 2015 году. Она отличается от SHA-2 внутренней структурой и основана на алгоритме Keccak. SHA-3 поддерживает те же длины хэша, что и SHA-2, и предназначена для усиления безопасности на фоне растущей вычислительной мощности.

ГОСТ Р 34.11-94 – отечественный стандарт хеширования, принятый в 1994 году. Основывался на криптографических принципах ГОСТ Р 34.10-94. Он широко применялся в государственных и коммерческих организациях для формирования электронной подписи и проверки целостности данных. Обеспечивал 256-битное хэш-значение, применялся в РФ до 2013 года, пока не был заменен на ГОСТ Р 34.11-2012. Использовался в СНГ до введения нового межгосударственного стандарта ГОСТ 34.11-2018.

ГОСТ Р 34.11-2012 и ГОСТ 34.11-2018 «Стрибог» – современный стандарт хеширования, принятый в России в 2012 году и в странах СНГ в 2018 году. Разработан ФСБ России совместно с ОАО «ИнфоТеКС». Он обеспечивает хэш-значения длиной 256 или 512 бит. Основой алгоритма является функция сжатия, использующая блочный шифр с конструкцией Миягучи-Пренеля (использование нелинейных преобразований и битовых операций), что повышает стойкость к криптоанализу и коллизиям.

MD5 и **SHA-1** были подвержены коллизиям первого рода. В 2004 году был найден пример коллизии для MD5, а в 2017 году для SHA-1. Эти хэш-функции теперь считаются небезопасными для использования в криптографических приложениях.

Коллизия первого рода: Для конкретного хэш-значения можно найти два разных сообщения.

Коллизия второго рода: Для любого сообщения существует другое сообщение с таким же хэш-значением.

1.2 История появления и этапы формирования семейства Message Digest

MD2 (Message Digest 2) – разработан Рональдом Ривестом в 1989 году для стандарта защищённой электронной почты PEM. MD2 использовалась для 128-битного хэширования сообщений фиксированной длины и была популярна в своё время, однако считается устаревшей из-за низкой стойкости. Её использование ограничено старыми системами.

MD4 – создан в 1990 году как улучшение MD2. Генерирует 128-битные хэши, применялся для аутентификации и проверки данных. Однако недостатки в устойчивости к атакам сделали его неподходящим для современных задач.

MD5 – наиболее известный представитель семейства MD, разработанный в 1991 году. MD5 создаёт 128-битный хэш и широко применялся для проверки целостности данных и хэширования паролей. Однако из-за обнаруженных уязвимостей, таких как возможность создания коллизий (недостаток нелинейных функций, так же время атаки существенно снизилось с помощью метода туннелирования и других подходов) и уязвимость к birthday-атакам (как и SHA-1) она больше не считается безопасной для криптографических целей.

Для полного перебора или перебора по словарю можно использовать программы PasswordsPro, MD5BFCPF, John the Ripper, Hashcat. Для перебора по словарю существуют готовые словари. Основным недостатком такого типа атак является высокая вычислительная сложность.

RainbowCrack — ещё один метод нахождения прообраза хеша из заданного множества. Он основан на генерации цепочек хешей, чтобы по получившейся базе вести поиск заданного хеша. Хотя создание радужных таблиц занимает много времени и памяти, последующий взлом производится очень быстро. Основная идея данного метода — достижение компромисса между временем поиска по таблице и занимаемой памятью.

В 2004 году начались активные атаки на MD5, которые в 2008 году привели к его официальному признанию небезопасным. Сегодня MD5 используется лишь в некриптографических задачах (контрольные суммы).

Одна из наиболее распространенных причин успешных атак заключается в том, что компании не используют добавление соли к исходному паролю. **Соль** – это фрагмент информации, состоящий из строки символов, которые добавляются к открытому тексту (исходному паролю пользователя), а затем хешируется. Соление делает пароли более устойчивыми к атакам типа радужных таблиц, так как подобный пароль будет иметь более высокую информационную энтропию и, следовательно, менее вероятное существование в предварительно вычисленных радужных таблицах. Как правило, соль должна быть не менее 48 бит.

MD6 – представлен в 2008 году как попытка создать новый стандарт хэширования. Алгоритм продемонстрировал современные подходы к построению устойчивых хэш-функций. Однако он не оправдал ожиданий из-за своей сложности и недостаточной производительности, уступив место более эффективным решениям.

1.3 Области применения

- **Генерация контрольных сумм для проверки целостности файлов.**

MD5 использовался для вычисления контрольных сумм файлов, которые затем публиковались вместе с файлами (например, на сайтах загрузки ПО). После загрузки файла пользователь мог сравнить хэш-сумму, чтобы убедиться в отсутствии изменений.

Коллизии в MD5 позволили злоумышленникам подменять файл, сохраняя тот же хэш.

- **Хеширование паролей.**

Пароли хранились в виде хэш-значений вместо явных текстовых строк. Это повышало безопасность, так как хэш-значение нельзя было легко преобразовать обратно в исходный пароль (в то время). Например, в старых версиях протоколов аутентификации (таких как MS-CHAP) использовались MD4 и MD5 для хеширования паролей.

- **Цифровые подписи и аутентификация.**

Алгоритмы MD применялись в цифровых подписях и аутентификации для обеспечения того, чтобы документ или сообщение не были изменены:

В системе цифровых подписей хэш сообщения подписывается с использованием закрытого ключа, что позволяет получателю проверить подлинность сообщения с помощью открытого ключа.

- **Хэширование в криптографических протоколах.**

В старых версиях протоколов SSL и TLS для создания хэш-значений и обеспечения целостности данных использовался MD5. В некоторых криптографических схемах VPN, включая протоколы IPsec, также использовались хэш-функции MD.

- **Генерация уникальных идентификаторов.**

Хэши могли быть использованы для создания идентификаторов сообщений или объектов, чтобы они могли быть быстро и уникально идентифицированы в системах или базах данных.

- **Интеграция в старые системы.**

Электронные документы: Для подписания и проверки целостности документов часто использовались хэш-функции MD.

Файловые системы: MD5 использовался для создания хэш-значений файлов, что позволяло быстро сравнивать файлы для поиска дубликатов или изменений.

Несмотря на устаревание, MD5 всё ещё используется в некриптографических задачах:

- Проверка целостности данных.
- Индексирование и быстрый поиск данных.
- Проверка данных в сетях (обнаружение ошибок передачи данных, проверка целостности).
- Некритичная аутентификация (В приложениях, не связанных с конфиденциальными данными, MD5 иногда используется для создания хэшей пользователей или сессий).

Для криптографических целей рекомендованы более надёжные алгоритмы, такие как SHA-2 и SHA-3.

1.4 Теоретический анализ MD5

На вход алгоритма поступает входной поток данных, хеш которого необходимо найти. Длина сообщения измеряется в битах и может быть любой (в том числе нулевой). Запишем длину сообщения в L. Это число целое и неотрицательное. Кратность каким-либо числам необязательна. После поступления данных идёт процесс подготовки потока к вычислениям.

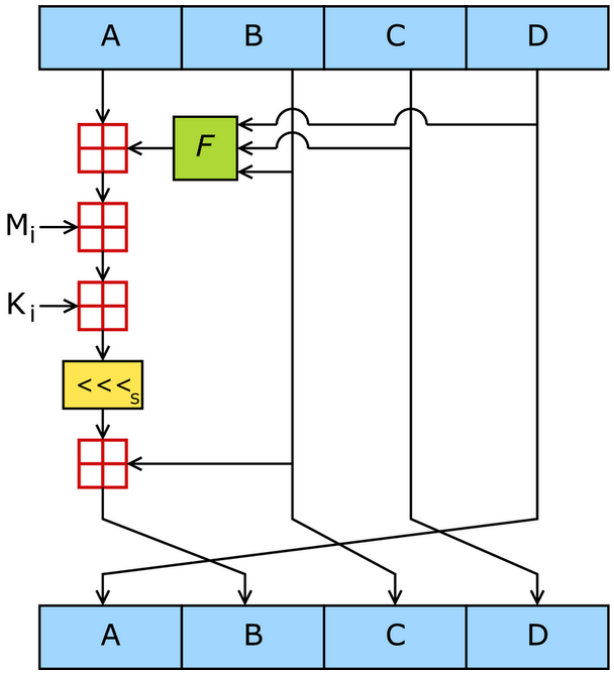

Рисунок 1 – Схема работы алгоритма MD5

F — нелинейная функция. M_i обозначает 32-битный блок входного сообщения, а K_i — 32-битную константу. $<<<_s$ обозначает циклический сдвиг влево на s бит. $\boxplus$ обозначает сложение по модулю 2^{32}. F зависит от раунда, K_i и s меняются каждую операцию.

MD5 состоит из пяти основных этапов:

1. **Выравнивание потока**

Сначала к концу потока дописывается единичный бит.

Затем добавляется некоторое число нулевых бит такое, чтобы новая длина потока L' стала сравнима с 448 (из 512 бит последний блок сообщения резервирует 64 бита для записи длины исходного сообщения) по модулю 512, ($L' = 512 \times N + 448$). Выравнивание происходит в любом случае, даже если длина исходного потока уже сравнима с 448.

Таким образом, до полного блока длиной 512 бит остаётся место для хранения длины исходного сообщения.

2. Добавление длины сообщения

В это место дописывают длину исходного сообщения в битах (до выравнивания). Сначала записывают младшие 4 байта, затем старшие. Если длина превосходит $2^{64} - 1$, то дописывают только младшие биты (эквивалентно взятию по модулю 2^{64}) (Выравнивание потока необходимо для того, чтобы сообщение можно было разбить на блоки длиной 512 бит а также гарантировать, что длина сообщения не превышает максимально возможную длину, которую можно записать в 64 бита). После этого длина потока станет кратной 512. Вычисления будут основываться на представлении этого потока данных в виде массива слов по 512 бит.

3. Инициализация буфера

Для хранения промежуточных результатов вычислений используются четыре 32-битных регистра. Эти значения фиксированы в стандарте MD5 и не меняются. Они инициализируются следующими значениями (в шестнадцатеричной форме, порядок байтов little-endian):

```
# Начальные значения (инициализирующие векторы)
A = 0x67452301
B = 0xEFCDAB89
C = 0x98BADCFE
D = 0x10325476
```

Начальное состояние $ABCD$ называется инициализирующим вектором.

4. Основной цикл вычислений

Определим функции и константы, которые понадобятся нам для вычислений.

Для каждого раунда потребуется своя функция. Введём функции от трёх параметров — слов, результатом также будет слово:

1-й этап: $FunF(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z)$,

2-й этап: $FunG(X, Y, Z) = (X \wedge Z) \vee (\neg Z \wedge Y)$,

3-й этап: $FunH(X, Y, Z) = X \oplus Y \oplus Z$,

4-й этап: $FunI(X, Y, Z) = Y \oplus (\neg Z \vee X)$,

где $\oplus, \wedge, \vee, \neg$ побитовые логические операции XOR, AND, OR и NOT соответственно.

Каждая функция реализует определенную логику для вычисления на разных этапах, что помогает избежать регулярности в конечном хеше и делает его более стойким к атакам.

Определим таблицу констант $T[1 \ldots 64]$ – 64-элементная таблица данных, где $T[n] = int(2^{32} \cdot |sinn|)$. Эти значения формируют дополнительные "случайные" элементы в процессе вычислений, что также помогает усилить стойкость алгоритма.

Каждый 512-битный блок проходит 4 этапа вычислений по 16 раундов. Для этого блок представляется в виде массива X из 16 слов по 32 бита. Все раунды однотипны и имеют вид: $[abcd\ k\ s\ i]$, определяемый как $a = b + ((a + Fun(b, c, d) + X[k] + T[i] \lll s)$, где k – номер 32-битного слова из текущего 512-битного блока сообщения, и $\ldots \lll s$ – циклический сдвиг влево на s бит полученного 32-битного аргумента. Число s задается отдельно для каждого раунда.

Заносим в блок данных элемент n из массива 512-битных блоков. Сохраняются значения A, B, C и D, оставшиеся после операций над предыдущими блоками (или их начальные значения, если блок первый).

```
# Сохраняем старые значения A, B, C, D
A_, B_, C_, D_ = A, B, C, D
```

В процессе обработки каждого блока хеш обновляется с учетом предыдущих значений регистров. Это позволяет последовательно изменять состояние хеша, что делает результаты каждого шага зависимыми от всех предшествующих.

После обработки каждого 512-битного блока промежуточные результаты регистров (A, B, C, D) суммируются с их начальными значениями.

```
# Обновляем значения A, B, C, D
A = (A + A_) & 0xFFFFFFFF
B = (B + B_) & 0xFFFFFFFF
C = (C + C_) & 0xFFFFFFFF
D = (D + D_) & 0xFFFFFFFF
```

После окончания цикла необходимо проверить, есть ли ещё блоки для вычислений. Если да, то переходим к следующему элементу массива $(n + 1)$ и повторяем цикл.

5. Вывод результата

Результат вычислений находится в буфере ABCD, это и есть хеш. Если выводить побайтово, начиная с младшего байта A и заканчивая старшим байтом D, то мы получим MD5-хеш.

```
md5_hash = (A.to_bytes(4, 'little') + B.to_bytes(4, 'little') + C.to_bytes(4,
'little') + D.to_bytes(4, 'little')).hex()
```

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы были рассмотрены хэш-функции. Рассмотрены основные алгоритмы, такие как MD5, SHA-1, SHA-2 и их российские аналоги. Особое внимание было уделено алгоритму MD5, его уязвимостям, таким как возможность создания коллизий, что делает его непригодным для криптографических целей. В работе также подробно разобран алгоритм действия MD5.

Изучив алгоритм, можно сделать вывод, что несмотря на свою популярность в прошлом, MD5 больше не является безопасным для применения в криптографии. Сегодня для защиты данных рекомендуется использовать более стойкие алгоритмы, такие как SHA-2 и SHA-3, которые обеспечивают более высокий уровень безопасности и устойчивость к атакам.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Hash function - Wikipedia
2. Хеш-функция, что это такое? / Хабр
3. Хеш-функция — Википедия
4. Что такое хеш-функция и как работают алгоритмы хеширования / Skillbox Media
5. Криптографическая хеш-функция — Википедия
6. SHA256 Algorithm
7. Реализация алгоритма SHA-256 / Хабр
8. SHA-1 — Википедия
9. SHA-2 — Википедия
10. SHA-3 - Wikipedia
11. MD2 — Википедия
12. MD4 — Википедия
13. MD5 — Википедия
14. MD6 — Википедия
15. Закат эпохи алгоритма MD5? / Хабр
16. MD5, SHA-1 и SHA-2. Какой алгоритм хэширования самый безопасный и как их проверить
17. ГОСТ Р 34.11-94 — Википедия
18. gost_r_34.11-94.pdf
19. Стрибог (хеш-функция) — Википедия
20. Хеш-функция Стрибог или в городе новый шериф / Хабр
21. Импортозамещенное шифрование глазами программиста. Хешируем по ГОСТ 34.11—2012 — Хакер

ПРИЛОЖЕНИЕ А

Листинг А.1 – Реализация MD5 на python

```
import hashlib
import math

def F(X, Y, Z):
    return (X & Y) | (~X & Z)

def G(X, Y, Z):
    return (X & Z) | (~Z & Y)

def H(X, Y, Z):
    return X ^ Y ^ Z

def I(X, Y, Z):
    return Y ^ (~Z | X)

# Начальные значения (инициализирующие векторы)
A = 0x67452301
B = 0xEFCDAB89
C = 0x98BADCFE
D = 0x10325476

# Преобразуем строку в байты
message = "Hello World!".encode()

print("MD5 хэш (hashlib):", hashlib.md5(message).hexdigest())

# Выравнивание потока
message_length = len(message) * 8
message = bytearray(message)
message.append(0x80)  # Дописываем 1 бит

# Дополнение длины сообщения
while len(message) % 64 != 56:
    message.append(0x00)  # Дополняем нулями
message.extend(message_length.to_bytes(8, byteorder='little'))

# Таблица констант
T = [int(abs(math.sin(i+1)) * 2**32) & 0xFFFFFFFF for i in range(64)]

# Функция циклического сдвига
def left_rotate(n, b):
    return ((n << b) | (n >> (32 - b))) & 0xFFFFFFFF

# Основной цикл
for i in range(0, len(message), 64):
    # Разбиваем сообщение на 16 32-битных слов
    X = [int.from_bytes(message[i + j:i + j + 4], byteorder='little') for j
in range(0, 64, 4)]

    # Сохраняем старые значения A, B, C, D
    A_, B_, C_, D_ = A, B, C, D

    # Выполняем 64 раунда
    for j in range(64):
        if j < 16:
            F_val = F(B, C, D)
```

```
            k = j
            s = [7, 12, 17, 22]
        elif j < 32:
            F_val = G(B, C, D)
            k = (5 * j + 1) % 16
            s = [5, 9, 14, 20]
        elif j < 48:
            F_val = H(B, C, D)
            k = (3 * j + 5) % 16
            s = [4, 11, 16, 23]
        else:
            F_val = I(B, C, D)
            k = (7 * j) % 16
            s = [6, 10, 15, 21]

        # Основная операция
        temp = D
        D = C
        C = B
        B = (B + left_rotate((A + F_val + X[k] + T[j]) & 0xFFFFFFFF, s[j %
4])) & 0xFFFFFFFF
        A = temp

    # Обновляем значения A, B, C, D
    A = (A + A_) & 0xFFFFFFFF
    B = (B + B_) & 0xFFFFFFFF
    C = (C + C_) & 0xFFFFFFFF
    D = (D + D_) & 0xFFFFFFFF

# Вывод результата
md5_hash = (A.to_bytes(4, 'little') + B.to_bytes(4, 'little') + C.to_bytes(4,
'little') + D.to_bytes(4, 'little')).hex()
print("MD5 хэш (самостоятельная реализация):", md5_hash)
```

Вывод:

```
D:\...\ТИБиМЗИ\лр7>md5.py
MD5 хэш (hashlib): ed076287532e86365e841e92bfc50d8c
MD5 хэш (самостоятельная реализация): ed076287532e86365e841e92bfc50d8c
```

ПРИЛОЖЕНИЕ Б

Листинг Б.1 – Реализация MD2 на python

```
from Crypto.Hash import MD2

message = b"Hello, MD2!"
h = MD2.new()
h.update(message)
md2_hash = h.hexdigest()
print("MD2 Hash (pycryptodome):", md2_hash)

S = [
    41, 46, 67, 201, 162, 216, 124, 1, 61, 54, 84, 161, 236, 240, 6, 19,
    98, 167, 5, 243, 192, 199, 115, 140, 152, 147, 43, 217, 188, 76, 130,
202,
    30, 155, 87, 60, 253, 212, 224, 22, 103, 66, 111, 24, 138, 23, 229, 18,
    190, 78, 196, 214, 218, 158, 222, 73, 160, 251, 245, 142, 187, 47, 238,
122,
    169, 104, 121, 145, 21, 178, 7, 63, 148, 194, 16, 137, 11, 34, 95, 33,
    128, 127, 93, 154, 90, 144, 50, 39, 53, 62, 204, 231, 191, 247, 151, 3,
    255, 25, 48, 179, 72, 165, 181, 209, 215, 94, 146, 42, 172, 86, 170, 198,
    79, 184, 56, 210, 150, 164, 125, 182, 118, 252, 107, 226, 156, 116, 4,
241,
    69, 157, 112, 89, 100, 113, 135, 32, 134, 91, 207, 101, 230, 45, 168, 2,
    27, 96, 37, 173, 174, 176, 185, 246, 28, 70, 97, 105, 52, 64, 126, 15,
    85, 71, 163, 35, 221, 81, 175, 58, 195, 92, 249, 206, 186, 197, 234, 38,
    44, 83, 13, 110, 133, 40, 132, 9, 211, 223, 205, 244, 65, 129, 77, 82,
    106, 220, 55, 200, 108, 193, 171, 250, 36, 225, 123, 8, 12, 189, 177, 74,
    120, 136, 149, 139, 227, 99, 232, 109, 233, 203, 213, 254, 59, 0, 29, 57,
    242, 239, 183, 14, 102, 88, 208, 228, 166, 119, 114, 248, 235, 117, 75,
10,
    49, 68, 80, 180, 143, 237, 31, 26, 219, 153, 141, 51, 159, 17, 131, 20
]

def md2(message: bytes) -> str:
    # Шаг 1: Добавление недостающих бит
    padding_length = 16 - (len(message) % 16)
    padding = bytes([padding_length] * padding_length)
    padded_message = message + padding

    # Шаг 2: Добавление контрольной суммы
    checksum = [0] * 16
    L = 0

    for i in range(0, len(padded_message), 16):
        block = padded_message[i:i + 16]
        for j in range(16):
            c = block[j]
            checksum[j] ^= S[c ^ L]
            L = checksum[j]

    # Добавляем контрольную сумму к сообщению
    final_message = padded_message + bytes(checksum)

    # Шаг 3: Инициализация MD-буфера
    X = [0] * 48

    # Шаг 4: Обработка сообщения блоками по 16 байт
    for i in range(0, len(final_message), 16):
```

```
        block = final_message[i:i + 16]

        # Копирование блока в буфер X
        for j in range(16):
            X[16 + j] = block[j]
            X[32 + j] = X[16 + j] ^ X[j]

        # 18 раундов обработки
        t = 0
        for j in range(18):
            for k in range(48):
                t = X[k] ^ S[t]
                X[k] = t
            t = (t + j) % 256

    # Шаг 5: Формирование хеша
    return ''.join(f'{x:02x}' for x in X[:16])

md2_hash = md2(message)
print("MD2 Hash (самостоятельная реализация):", md2_hash)
```

Вывод:

```
D:\...\ТИБиМЗИ\лр7>md2.py
MD2 Hash (pycryptodome): 3a95b463147c002aa2852880578a3006
MD2 Hash (самостоятельная реализация): 3a95b463147c002aa2852880578a3006
```