1. Цель работы

Реализовать алгоритм хеширования MD4.
Провести эксперименты:
1). Нахождение второго прообраза.
2). Нахождение коллизий.

Построить графики зависимости среднего значения сложности второго прообраза и коллизии от количества взятых бит. Оценить полученные графики.

2. Алгоритм

Алгоритм состоит из пяти шагов:

2.1. Добавление недостающих битов
Необходимо, чтобы количество бит соответствовало N, где N mod 512 = 448.

2.2. Добавление длины сообщения
Длина добавляется в 64-битном представлении так, что N mod 512 = 0.

2.3. Инициализация MD-буфера
word A: 01 23 45 67
word B: 89 ab cd ef
word C: fe dc ba 98
word D: 76 54 32 10

2.4. Обработка сообщений блоками по 16 слов

2.4.1. Определим вспомогательные функции
f(X,Y,Z) = XY v not(X)Z
g(X,Y,Z) = XY v XZ v YZ
h(X,Y,Z) = X xor Y xor Z

2.4.2. Теперь обозначим [A B C D i s] как операцию
A = (A + f(B,C,D) + X[i]) <<< s, где X[i] – слово из обрабатываемого блока.

2.4.3. Первый раунд:
[A B C D 0 3]
[D A B C 1 7]
[C D A B 2 11]
[B C D A 3 19]
[A B C D 4 3]
[D A B C 5 7]
[C D A B 6 11]
[B C D A 7 19]
[A B C D 8 3]
[D A B C 9 7]
[C D A B 10 11]
[B C D A 11 19]
[A B C D 12 3]

[D A B C 13 7]
[C D A B 14 11]
[B C D A 15 19]

2.4.4. Теперь обозначим [A B C D i s] как операцию
A = (A + g(B,C,D) + X[i] + 5A827999) <<< s, где X[i] – слово из обрабатываемого блока.

2.4.5. Второй раунд:
[A B C D 0 3]
[D A B C 4 5]
[C D A B 8 9]
[B C D A 12 13]
[A B C D 1 3]
[D A B C 5 5]
[C D A B 9 9]
[B C D A 13 13]
[A B C D 2 3]
[D A B C 6 5]
[C D A B 10 9]
[B C D A 14 13]
[A B C D 3 3]
[D A B C 7 5]
[C D A B 11 9]
[B C D A 15 13]

2.4.6. Теперь обозначим [A B C D i s] как операцию
A = (A + g(B,C,D) + X[i] + 6ED9EBA1) <<< s, где X[i] – слово из обрабатываемого блока.
[A B C D 0 3]
[D A B C 8 9]
[C D A B 4 11]
[B C D A 12 15]
[A B C D 2 3]
[D A B C 10 9]
[C D A B 6 11]
[B C D A 14 15]
[A B C D 1 3]
[D A B C 9 9]
[C D A B 5 11]
[B C D A 13 15]
[A B C D 3 3]
[D A B C 11 9]
[C D A B 7 11]
[B C D A 15 15]

2.4.7. К значениям A B C D прибавить их состояния в начале блока.

2.5. Формирование кэша ABCD

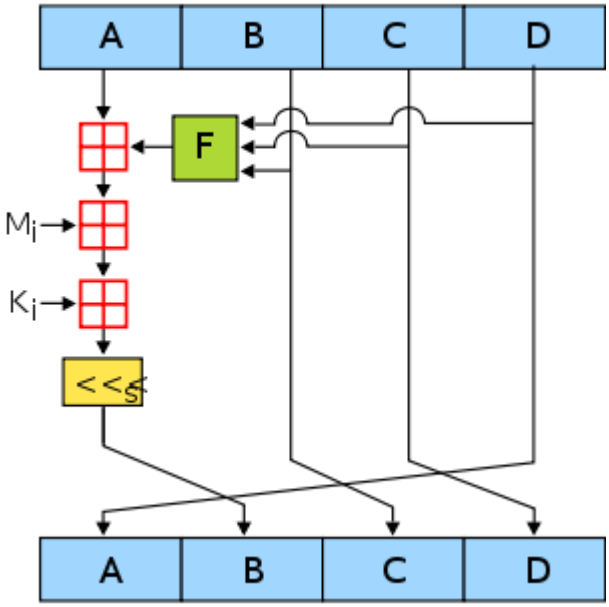

Рис. 1 - Одна операция MD4

3. Пример:

3.1. Исходная строка: ""
Результат работы программы: 31d6cfe0d16ae931b73c59d7e0c089c0
Полученный хэш совпадает с теоретическим значением.

3.2. Исходная строка: " The quick brown fox jumps over the lazy dog"
Результат работы программы: 1bee69a46ba811185c194762abaeae90
Полученный хэш совпадает с теоретическим значением.

3.3. Исходная строка: " abc"
Результат работы программы: a448017aaf21d8525fc10ae87aa6729d
Полученный хэш совпадает с теоретическим значением.

3.4. Исходная строка: " message digest"
Результат работы программы: d9130a8164549fe818874806e1c7014b
Полученный хэш совпадает с теоретическим значением.

3.5. Исходная строка: " abcdefghijklmnopqrstuvwxyz"
Результат работы программы: d79e1c308aa5bbcdeea8ed63df412da9
Полученный хэш совпадает с теоретическим значением.

3.6. Исходная строка: "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijk
lmnopqrstuvwxyz0123456789"
Результат работы программы: 043f8582f241db351ce627e153e7f0e4
Полученный хэш совпадает с теоретическим значением.

3.7. Исходная строка: "123456789012345678901234567890123456
5678901234567890123456789012345678901234567890"
Результат работы программы: e33b4ddc9c38f2199c3e7b164fcc0536
Полученный хэш совпадает с теоретическим значением.

4. Нахождение второго прообраза и коллизий

Возьмем слово-пароль из 3х слов, одно из них состоит из нулей.
X = «hello 000 word»
Необходимо найти от него хеш h(**x**), взять от полученного хеша первые n бит, обозначив последовательность $\mathbf{y}_0$. Далее случайным образом сгенерировать N (N зависит от двух условий, описанных ниже) сообщений, найти от каждого хеш и взять от каждого хеша первые n бит, получив последовательность $\mathbf{y}_1$, $\mathbf{y}_2$,..., $\mathbf{y}_{N-1}$.

1) Для нахождения сложности второго прообраза необходимо найти такой $\mathbf{y}_i$, что $\mathbf{y}_i = \mathbf{y}_0$. Посчитать количество шагов, которое потребовалось, чтобы найти $\mathbf{y}_i$.
2) Для нахождения коллизии необходимо найти в полученной последовательности такие $\mathbf{y}_i$ и $\mathbf{y}_j$, что $\mathbf{y}_i = \mathbf{y}_j$. Посчитать количество шагов, которое потребовалось, чтобы найти эту пару.

Проделав эксперимент 1000 раз, получим средние значения сложности второго прообраза и сложности коллизии.

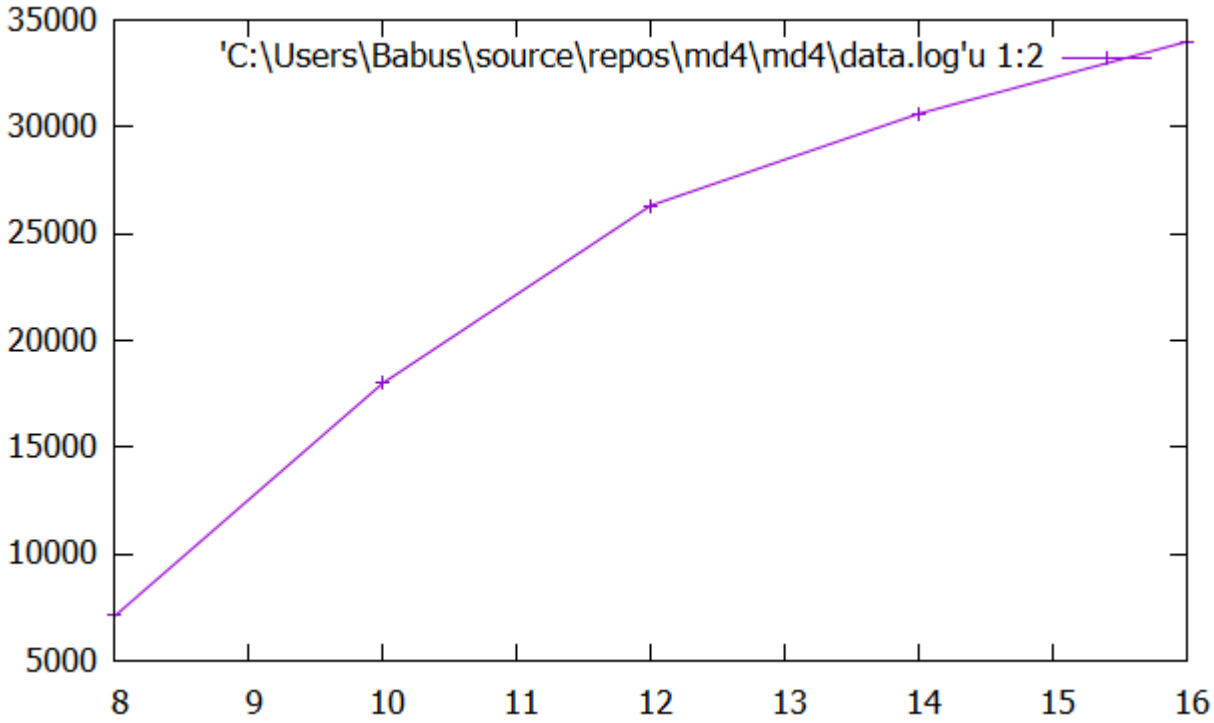

Рис.2 - график зависимости среднего значения сложности второго прообраза от количества взятых бит.

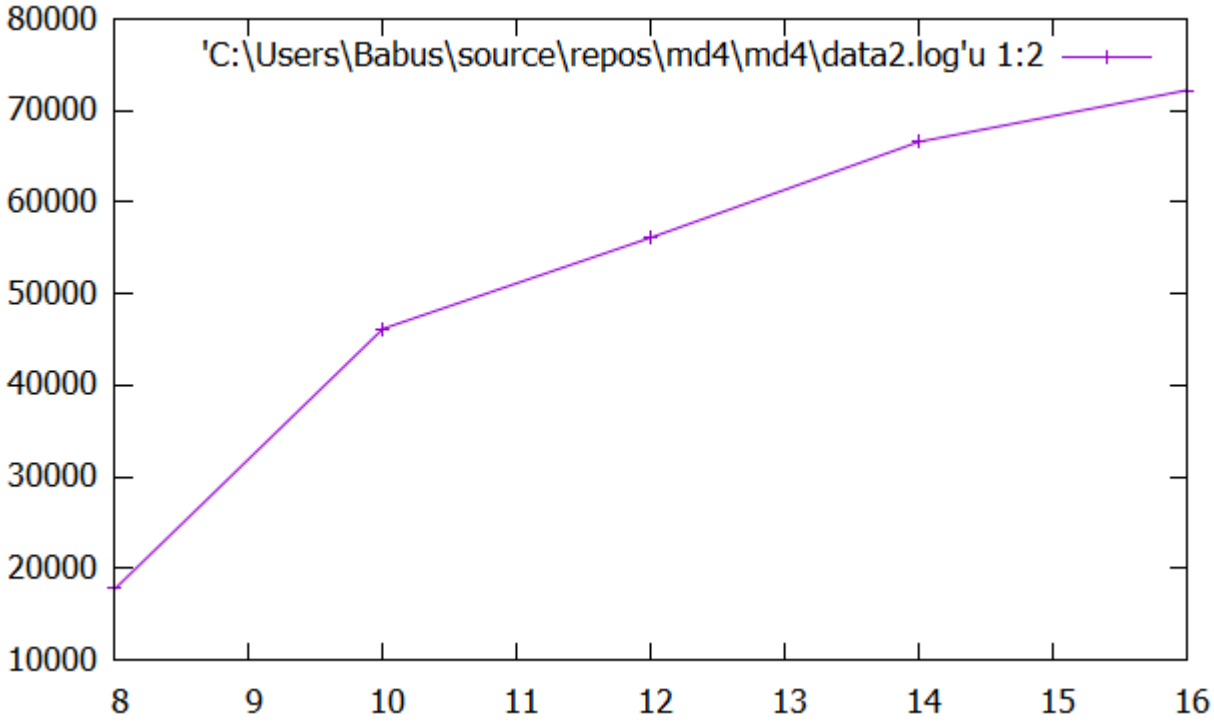

Рис.3 - график зависимости среднего значения сложности коллизии от количества взятых бит.

Анализируя данные графики, можно увидеть прямую зависимость среднего значения сложности второго прообраза и коллизии от количества бит. При увеличении с 8 до 16 бит, сложность второго прообраза возросла в 6 раз, а сложность коллизии – в 8. Это связано с тем, что для нахождении коллизии, нужно использовать количество шагов, необходимое для сравнения каждого элемента последовательности с каждым ($O(n^2)$), а для второго прообраза – сравнить последовательность с одним ее элементом ($O(n)$). Так же на формирование сложности влияют внутренние шаги программного вычисления. Поэтому средняя сложность коллизии превышает сложность второго прообраза и функция растет быстрее.

Эксперимент был проведен 1000 раз при количестве сообщений равном 100. Так как сообщения генерировались случайным образом, при заданных параметрах наблюдалась стойкость алгоритма ко второму прообразу для 16 бит. При количестве сообщений 500 и более, стойкость не наблюдалась.

5. Вывод

В ходе данной лабораторной работы был программно воспроизведен алгоритм хэширования MD4, а так же проведен анализ устойчивости данного алгоритма к нахождению второго прообраза и коллизий. Можно сделать следующие выводы о проделанной работе:

Так как результат программы совпадает с теоретическим значением, можно сказать, что алгоритм реализован верно. В качестве аналитического эксперимента, была исследована зависимость сложности второго прообраза и

коллизий от количества бит. По данным графиков была найдена прямая зависимость от данных величин. Функция сложности коллизии растет быстрее функции сложности второго прообраза.
Самая быстрая хэш-функция, оптимизирована для 32-битных машин среди семейства MD-функций, содержит три цикла по 16 шагов каждый.
В 1993 году был описан алгоритм взлома MD4, поэтому на сегодняшний день данная функция не рекомендована для использования с реальными приложениями.

6. Список источников

1. А.Л. Чмора "Современная прикладная криптография"
2. Черчхаус. Коды и шифры