Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

Дисциплина «Информационная безопасность»

**Лабораторная работа №3**

«Потоковые шифры, функции хеширования»

Выполнил студент

группы ИВТАПбд-41:

Матюнин И.С.

Проверил:

Мартынов А. И.

Ульяновск, 2024

**Задание на лабораторную работу**

1. Шифровать и дешифровать текстовые и двоичные файлы с помощью потокового шифрования и генератора псевдослучайных чисел, разработанного в предыдущей лабораторной работе
2. Сохранять зашифрованные/дешифрованные данные в файл
3. Хешировать текстовый пароль, который используется при шифровании для инициализации генератора псевдослучайных чисел с помощью функции хеширования, указанной в варианте

**Дополнительные требования к приложению:**

Программа должна быть оформлена в виде удобной утилиты

Программа должна обрабатывать файлы любого размера и содержания

Должна быть предусмотрена возможность просмотра, сгенерированного хешзначения по введенному паролю

* Текст программы оформляется прилично (удобочитаемо, с описанием ВСЕХ функций, переменных и критических мест).
* В процессе работы программа ОБЯЗАТЕЛЬНО выдает информацию о состоянии процесса шифрования / дешифрования.
* Интерфейс программы может быть произвольным, но удобным и понятным (разрешается использование библиотек GUI)
* Среда разработки и язык программирования могут быть произвольными.

**MaHash4**

Здесь используются классические сдвиги и обычный циклический сдвиг. Только вместо взаимозамены опять “сборка” значений. Эта функция, как и прошлая, стабильно показывает очень хорошие результаты, не имеет “провалов”.

*Листинг 1. Функция MaHash4*

|  |
| --- |
| # Функция MaHash4 def mahash4(password):  # Инициализируем 4 регистра, каждый по 32 бита  h0, h1, h2, h3 = 0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476  password\_bytes = password.encode('utf-8')   for byte in password\_bytes:  x = sTable[byte]  h0 = (h0 + ((x ^ h1) << 7 | (x ^ h1) >> (32 - 7))) & 0xFFFFFFFF  h1 = (h1 + ((x ^ h2) << 11 | (x ^ h2) >> (32 - 11))) & 0xFFFFFFFF  h2 = (h2 + ((x ^ h3) << 17 | (x ^ h3) >> (32 - 17))) & 0xFFFFFFFF  h3 = (h3 + ((x ^ h0) << 19 | (x ^ h0) >> (32 - 19))) & 0xFFFFFFFF  return ((h0 << 96) | (h1 << 64) | (h2 << 32) | h3).to\_bytes(16, byteorder='big') |

Работа функции основывается на побайтовой обработке входной строки и обновлении четырех регистров (h0, h1, h2, h3) с помощью битовых операций, таких как XOR, циклический сдвиг и сложение.

На первом этапе инициализируются четыре 32-битных регистра (h0, h1, h2, h3) фиксированными значениями, которые служат начальной точкой для вычислений.

Далее, входной пароль преобразуется в массив байтов в кодировке UTF-8, чтобы обеспечить поддержку различных символов. Затем каждый байт обрабатывается по очереди: его значение используется в качестве индекса для обращения к таблице подстановки sTable, которая должна содержать 256 значений. Таблица играет роль источника дополнительной энтропии, повышая стойкость функции.

Для каждого байта происходит обновление регистров. Значение байта из sTable комбинируется с текущими значениями регистров с помощью операций XOR и циклического сдвига на разные величины (7, 11, 17, 19 бит). После каждой операции результат складывается с текущим значением регистра, и итог ограничивается 32 битами с помощью побитового И с маской 0xFFFFFFFF.

После обработки всех байтов входной строки регистры объединяются в один 128-битный результат. Это достигается путем сдвига значений h0, h1, h2, h3 на 96, 64 и 32 бита соответственно и их побитового ИЛИ. Затем полученное число преобразуется в массив байтов длиной 16 (128 бит) в порядке "старший байт первым".

*Листинг 2. Функция шифрования-дешифрования*

|  |
| --- |
| #Функция шифрования/дешифрования def encrypt\_decrypt\_file(input\_file, output\_file, seed):  with open(input\_file, 'rb') as f:  data = f.read()  key\_stream = linear\_congruent\_generator(len(data), seed)  transformed\_data = bytes([byte ^ key\_stream[i] for i, byte in enumerate(data)])  with open(output\_file, 'wb') as f:  f.write(transformed\_data) |

Функция выполняет шифрование и дешифрование файла с помощью XOR и псевдослучайного потока ключей, сгенерированного линейным конгруэнтным генератором (LCG).

1. Читает данные из входного файла в бинарном режиме.
2. Генерирует поток ключей такой же длины, как данные, на основе начального значения seed.
3. Для каждого байта данных выполняет XOR с соответствующим байтом ключа, преобразуя данные.
4. Записывает преобразованные данные в выходной файл.

Метод, используемый в данной функции, является симметричным: повторное использование того же ключа дешифрует зашифрованный файл.