УДК 004.624

Реализация шифрования методом RC5 на языке программирования Python

Кизянов Антон Олегович
Приамурский государственный университет имени Шолом-Алейхема
Студент

Глаголев Владимир Александрович
Приамурский государственный университет имени Шолом-Алейхема
к.г.н., доцент кафедры информационных систем, математики и правовой информатики

Аннотация
В данной статье приведена реализация алгоритма шифрования Рона Ривеста RC5. Для реализации был использован язык программирования Python. Итогом статьи выступает алгоритм RC5, который полностью реализован на языке Python и продемонстрирован на примере.
Ключевые слова: Python, RC5

Implementing rc5 encryption in the Python programming language

Kizyanov Anton Olegovich
Sholom-Aleichem Priamursky State University
student

Glagolev Vladimir Aleksandrovich
Sholom-Aleichem Priamursky State University
candidate of geographical Sciences, Associate Professor of the Department of Information Systems, Mathematics and Legal Informatics

Abstract
This article presents the implementation of the encryption algorithm Ron Rivest RC5. For implementation, the Python programming language was used. The result of the article is the RC5 algorithm, which is fully implemented in Python and is shown on an example.
Keywords: Python, rc5

Блочные алгоритмы шифрования, важнейшие прикладные инструменты передачи защищенной информации в локальных и глобальных сетях. В глобальной сети интернет широкое применение получили на уровне прикладных протоколов TCP/IP, включая данные гипертекстовых документов, электронной почты и месседжеров. Для реализации протоколов защищенной передачи данных требуются разработка современных

программных средств на базе серверных операционных систем, одним из решений является использование языка программирования Python, позволяющего осуществлять различную аналитическую агрегацию данных.

Цель работы является реализация алгоритма шифрования Рона Ривеста RC5 на примере языке программирования Python.

Задачами исследования являлось: изучить основные работы по данной тематике и реализацию алгоритма на языке Python.

Ранее этим вопросом интересовались Б.Я. Рябко, В.С. Стогниенко, Ю.И. Шокин развивали тему «Экспериментальные исследования эффективности генераторов псевдослучайных чисел, базирующихся на криптографических алгоритмах rc5 и rc6» [1] в которой обсуждается использование криптографических алгоритмов RC5 и RC6 как генераторов псевдослучайных чисел, приводится их краткое описание и результаты выполненного исследования. П.П. Чистяков с темой «О сравнительном анализе алгоритмов шифрования» [2], а подробнее про сравнение алгоритмов ГОСТ 28147-89 и RC5 и установлено, что более гибким в отношении приспособляемости к различным вычислительным средствам является алгоритм RC5, а более устойчивым по отношению к криптоатакам взломщика шифров следует считать алгоритм ГОСТ 28147-89. А.А. Набебин, Е.Д. Сапожков опубликовали статью «Повышающая криптостойкость оболочка над блоковым шифром ривеста» [3] рассказали про модификацию RC5-S блокового шифра Ривеста RC5, значительно повышающего криптографическую стойкость шифра RC5, в которой использован алгоритм CBC (Cipher Block Chaining) сцепления блоков шифротекста.

В отличие от многих схем, RC5 имеет переменный размер блока (32, 64 или 128 бит), размер ключа (от 0 до 2040 бит) и количество раундов (от 0 до 255). Первоначальным предлагаемым выбором параметров был размер блока 64 бит, 128-битный ключ и 12 раундов.

Тщательная простота алгоритма вместе с новизной зависящих от данных раундов сделала RC5 привлекательным объектом исследования для криптоаналитиков. RC5 в основном обозначается как RC5-w / r / b, где w = размер слова в битах, r = количество раундов, b = количество 8-битных байтов в ключе.

RC5-шифрование и дешифрование расширяют случайный ключ до 2 (r + 1) слов, которые будут использоваться последовательно (и только один раз) во время процессов шифрования и дешифрования.

В RC5 используются следующие имена переменных:

- w - длина слова в битах, обычно 16, 32 или 64. Шифрование выполняется в блоках из 2 слов.
- u = w / 8 - Длина слова в байтах.
- b - длина ключа в байтах.
- K - ключ, который рассматривается как массив байтов (с использованием индексации на основе 0).

- c - длина ключа в словах (или 1, если b = 0).
- L - временный массив, используемый во время основного планирования.
- r - количество раундов, используемых при шифровании данных.
- t = 2 (r + 1) - требуется количество раундов подразделов.
- Pw - первая магическая константа, определяемая на формуле 1

$$P_w < -Odd\big((f-1) * 2^w\big) \quad (1)$$

Для общих значений w соответствующие значения P w приведены здесь в шестнадцатеричном виде:

- Для w = 16: 0xB7E1
- Для w = 32: 0xB7E15163
- Для w = 64: 0xB7E151628AED2A6B

Qw - Вторая магическая константа, определяемая формулой на формуле 2.

$$Q_w < -Odd\big((e-2) * 2^w\big) \quad (2)$$

Для общих значений w соответствующие значения Qw приведены здесь в шестнадцатеричном виде:

- Для w = 16: 0x9E37
- Для w = 32: 0x9E3779B9
- Для w = 64: 0x9E3779B97F4A7C15

Шифрование включало несколько раундов простой функции. Рекомендуется 12 или 20 раундов, в зависимости от потребностей безопасности и соображений времени. Помимо переменных, используемых выше, в этом алгоритме используются следующие переменные:

- A, B - Два слова, составляющие блок открытого текста для шифрования.

Следующий код реализует полностью метод шифрования RC5.

```
import math
import typing


class Cipher:
    def __init__(self) -> None:
        raise NotImplementedError("Must use a subclass of generic `Cipher`")

    def encrypt_text(self, text: str) -> str:
        return self.encrypt(text.encode()).decode()

    def decrypt_text(self, text: str) -> str:
        return self.decrypt(text.encode()).decode()

    def encrypt(self, data: bytes) -> bytes:
        raise NotImplementedError("Must use a subclass of generic `Cipher`")

    def decrypt(self, data: bytes) -> bytes:
```

```
        raise NotImplementedError("Must use a subclass of generic `Cipher`")


def _rotate_left(val: int, r_bits: int, max_bits: int) -> int:
    v1 = (val << r_bits % max_bits) & (2 ** max_bits - 1)
    v2 = (val & (2 ** max_bits - 1)) >> (max_bits - (r_bits % max_bits))
    return v1 | v2


def _rotate_right(val: int, r_bits: int, max_bits: int) -> int:
    v1 = (val & (2 ** max_bits - 1)) >> r_bits % max_bits
    v2 = val << (max_bits - (r_bits % max_bits)) & (2 ** max_bits - 1)

    return v1 | v2


def _expand_key(key: bytes, wordsize: int, rounds: int) -> typing.List[int]:
    def _align_key(key: bytes, align_val: int) -> typing.List[int]:
        while len(key) % (align_val):
            key += (
                b"\x00"
            )

        L = []
        for i in range(0, len(key), align_val):
            L.append(int.from_bytes(key[i: i + align_val],
byteorder="little"))

        return L

    def _const(w: int) -> typing.Tuple[int, int]:
        if w == 16:
            return (0xB7E2, 0x9E38)
        elif w == 32:
            return (0xB7E15164, 0x9E3779B8)
        elif w == 64:
            return (0xB7E151628AED2A6A, 0x9E3779B97F4A7C14)
        raise ValueError("Bad word sie")

    def _extend_key(w: int, r: int) -> typing.List[int]:
        P, Q = _const(w)
        S = [P]
        t = 2 * (r + 1)
        for i in range(1, t):
            S.append((S[i - 1] + Q) % 2 ** w)

        return S

    def _mix(
            L: typing.List[int], S: typing.List[int], r: int, w: int, c: int
    ) -> typing.List[int]:
        t = 2 * (r + 1)
        m = max(c, t)
        A = B = i = j = 0

        for k in range(3 * m):
            A = S[i] = _rotate_left(S[i] + A + B, 3, w)
            B = L[j] = _rotate_left(L[j] + A + B, A + B, w)

            i = (i + 1) % t
            j = (j + 1) % c

        return S
```

```
    aligned = _align_key(key, wordsize // 8)
    extended = _extend_key(wordsize, rounds)

    S = _mix(aligned, extended, rounds, wordsize, len(aligned))

    return S


def _encrypt_block(
        data: bytes, expanded_key: typing.List[int], blocksize: int, rounds:
int
) -> bytes:
    w = blocksize // 2
    b = blocksize // 8
    mod = 2 ** w

    A = int.from_bytes(data[: b // 2], byteorder="little")
    B = int.from_bytes(data[b // 2:], byteorder="little")

    A = (A + expanded_key[0]) % mod
    B = (B + expanded_key[1]) % mod

    for i in range(1, rounds + 1):
        A = (_rotate_left((A ^ B), B, w) + expanded_key[2 * i]) % mod
        B = (_rotate_left((A ^ B), A, w) + expanded_key[2 * i + 1]) % mod

    res = A.to_bytes(b // 2, byteorder="little") + B.to_bytes(
        b // 2, byteorder="little"
    )
    return res


def _decrypt_block(
        data: bytes, expanded_key: typing.List[int], blocksize: int, rounds:
int
) -> bytes:
    w = blocksize // 2
    b = blocksize // 8
    mod = 2 ** w

    A = int.from_bytes(data[: b // 2], byteorder="little")
    B = int.from_bytes(data[b // 2:], byteorder="little")

    for i in range(rounds, 0, -1):
        B = _rotate_right(B - expanded_key[2 * i + 1], A, w) ^ A
        A = _rotate_right((A - expanded_key[2 * i]), B, w) ^ B

    B = (B - expanded_key[1]) % mod
    A = (A - expanded_key[0]) % mod

    res = A.to_bytes(b // 2, byteorder="little") + B.to_bytes(
        b // 2, byteorder="little"
    )
    return res


class RC5(Cipher):
    def __init__(self, key: bytes, blocksize: int, rounds: int) -> None:
        self.key = key
        self.blocksize = blocksize
        self.rounds = rounds

    def encrypt(self, data: bytes) -> bytes:
        blocksize = self.blocksize
```

```
        key = self.key
        rounds = self.rounds

        w = blocksize // 2
        b = blocksize // 8

        expanded_key = _expand_key(key, w, rounds)

        index = b
        chunk = data[:index]
        out = []
        while chunk:
            chunk = chunk.ljust(b, b"\x00")
            encrypted_chunk = _encrypt_block(chunk, expanded_key, blocksize,
rounds)
            out.append(encrypted_chunk)

            chunk = data[index: index + b]
            index += b
        return b"".join(out)

    def decrypt(self, data: bytes) -> bytes:
        blocksize = self.blocksize
        key = self.key
        rounds = self.rounds

        w = blocksize // 2
        b = blocksize // 8

        expanded_key = _expand_key(key, w, rounds)

        index = b
        chunk = data[:index]
        out = []
        while chunk:
            decrypted_chunk = _decrypt_block(chunk, expanded_key, blocksize,
rounds)
            chunk = data[index: index + b]
            if not chunk:
                decrypted_chunk = decrypted_chunk.rstrip(b"\x00")

            index += b
            out.append(decrypted_chunk)
        return b"".join(out)
```

На рисунке 1 представлен рабочий пример алгоритма, который использует секретный ключ, размер блока и количество раундов. Было зашифровано сообщение Secret massages. В первом случае вывели только зашифрованное сообщение, а во втором зашифрованное сообщение и расшифровали обратно.

Рис. 1 Пример шифровки и расшифровки текста на RC5

Вывод

С ростом вычислительных мощностей растет и потребность в более сложных алгоритмах шифрования информации, менее поддающихся расшифровке. На данный момент алгоритм RC5 входит в состав новых и более совершенных методов шифрования. Что говорит о его надежности и оптимизированности.

Библиографический список

1. Рябко Б.Я., Стогниенко В.С., Шокин Ю.И. Экспериментальные исследования эффективности генераторов псевдослучайных чисел, базирующихся на криптографических алгоритмах rc5 и rc6 // Вычислительные технологии 2000. № 6 С. 70-79. URL https://elibrary.ru/item.asp?id=13026345 (Дата обращения: 4.10.2018)
2. Чистяков П.П. О сравнительном анализе алгоритмов шифрования // Список-2012 2012. С. 183-186. URL https://elibrary.ru/item.asp?id=26648928 (Дата обращения: 4.10.2018)
3. Набебин А.А., Сапожков Е.Д. Повышающая криптостойкость оболочка над блоковым шифром ривеста // Вестник московского энергетического института 2016. № 1 С. 15-17. URL https://elibrary.ru/item.asp?id=25808195 (Дата обращения: 4.10.2018)