Тесты для криптоалгоритма «Кузнечик»

**Оглавление**

[Сравнение результата работы с тестовыми значениями 2](#__RefHeading___Toc1311_1566015580)

[Описание алгоритма: 8](#__RefHeading___Toc1313_1566015580)

[Тестирование и описание характеристик реализации 10](#__RefHeading___Toc1315_1566015580)

[1) Режим EBC 10](#__RefHeading___Toc1317_1566015580)

[2) Режим CBC 11](#__RefHeading___Toc1319_1566015580)

[3) Режим CTR-ACPKM 12](#__RefHeading___Toc1321_1566015580)

[Описание тестового стенда 14](#__RefHeading___Toc1323_1566015580)

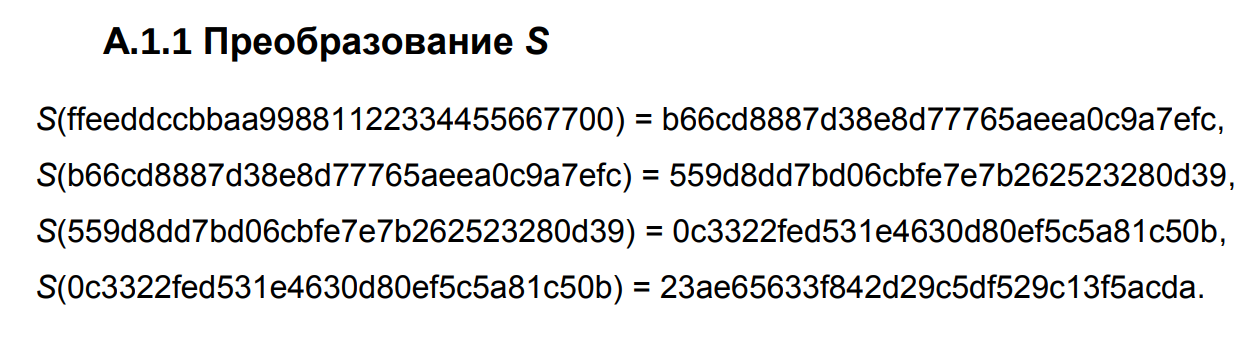
[Фотографии анализа состояния системы во время выполнения тестирования: 14](#__RefHeading___Toc1325_1566015580)

# Сравнение результата работы с тестовыми значениями

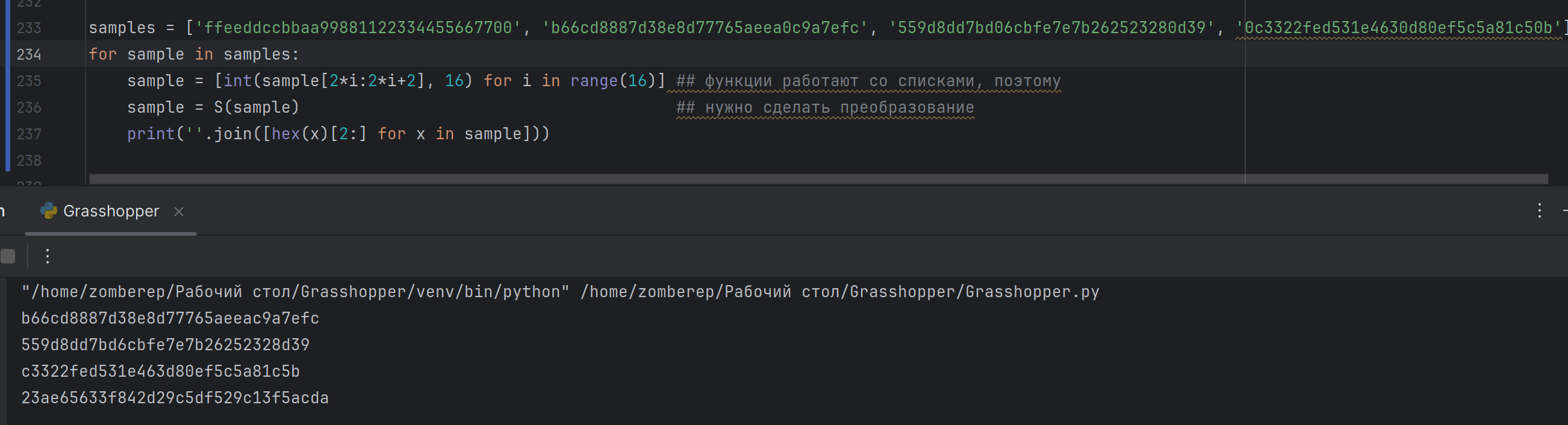
( <https://tc26.ru/standard/gost/GOST_R_3412-2015.pdf> — ссылка на спецификацию)

1) Функция S:

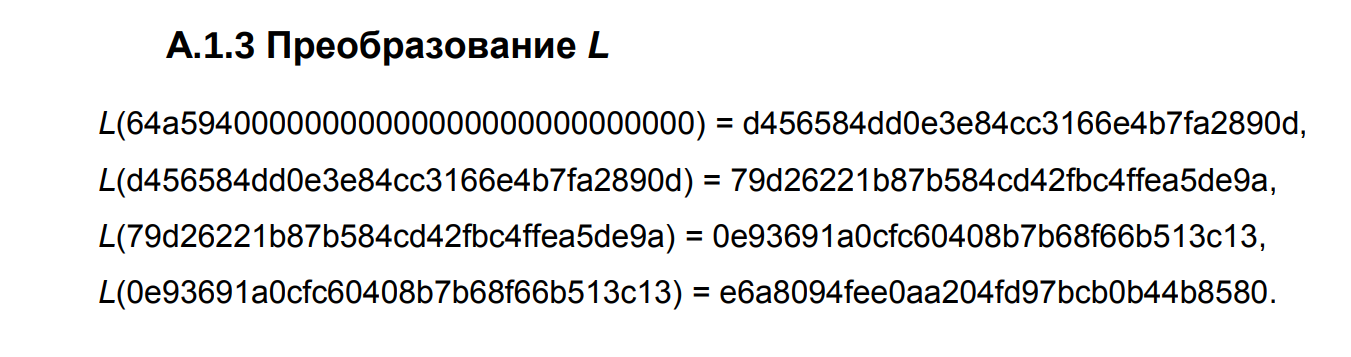
*Примеры из спецификации:*

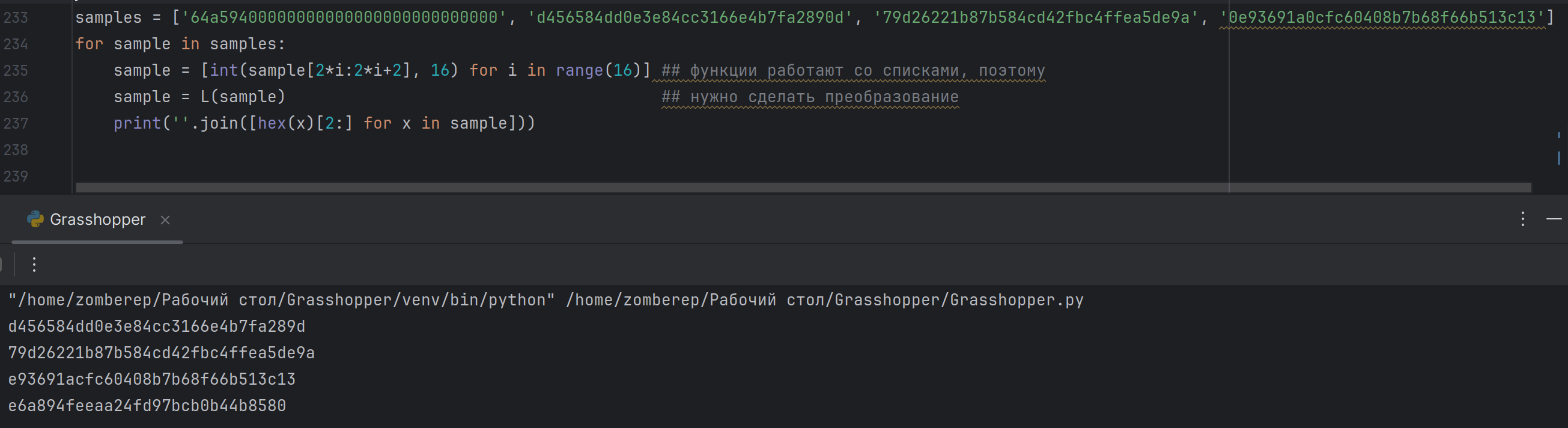


*Результат работы функции для примеров из спецификации:*



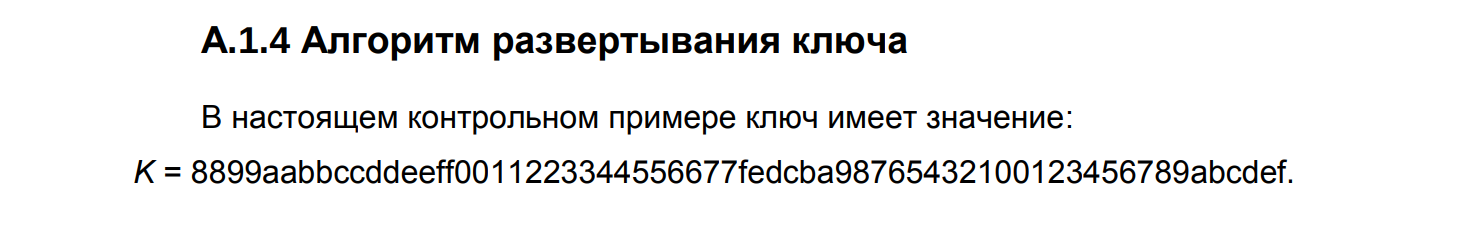
2) Функция L

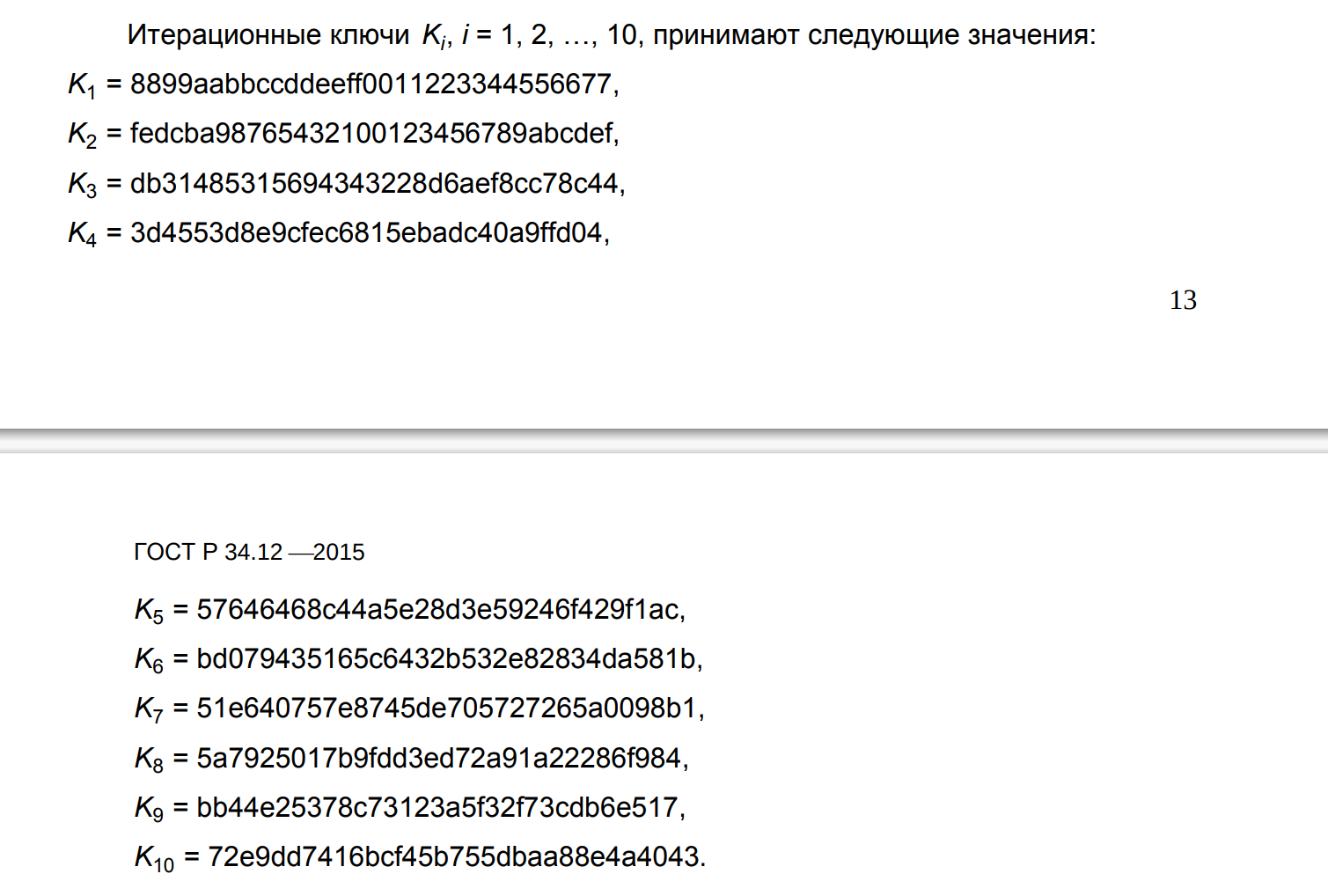


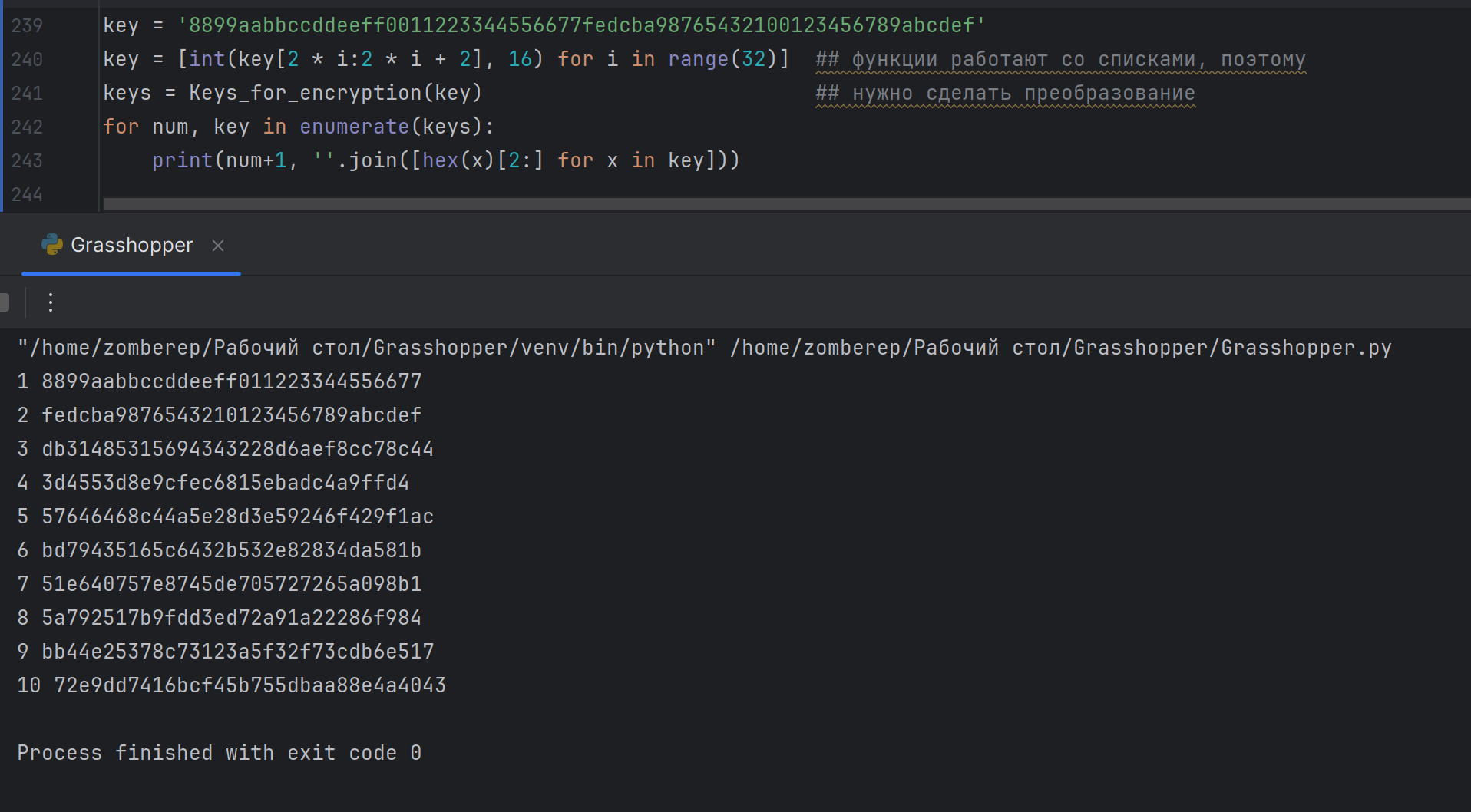


В спецификации также указывается некая функция R. Данный файл не содержит её тестов, поскольку она является подфункцией функции L. Тесты на L не могли бы быть верны, если бы функция R работала некорректно. Автор, реализующий алгоритм решил не выносить преобразование R в отдельную функцию.

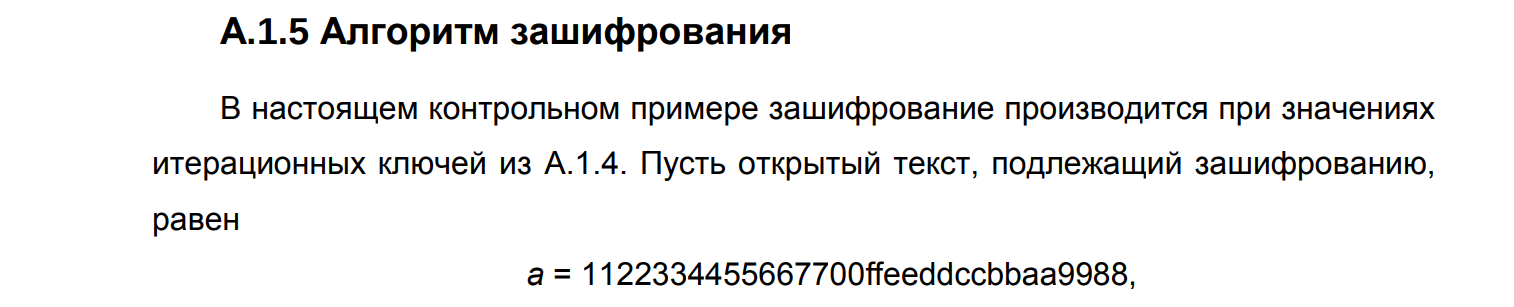
3) Генерация раундовых ключей

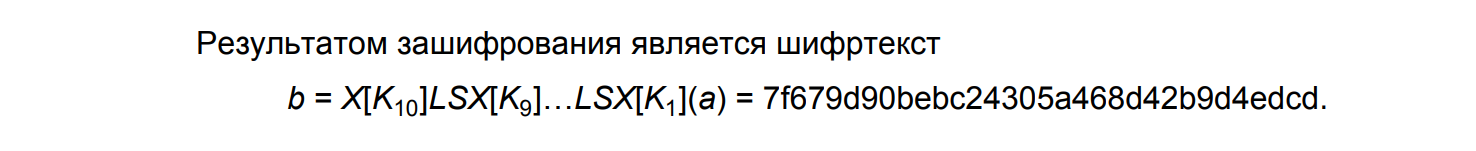


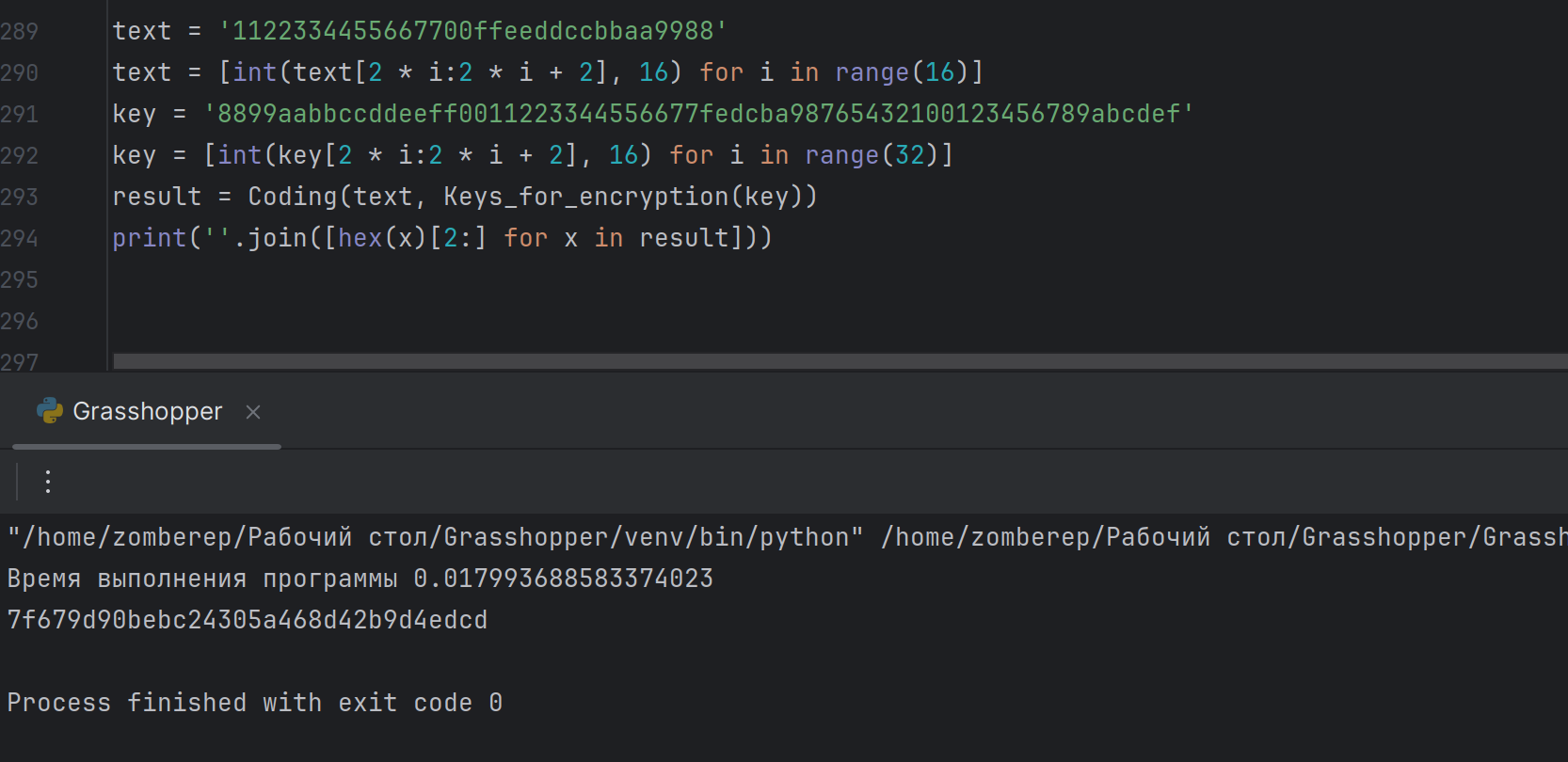




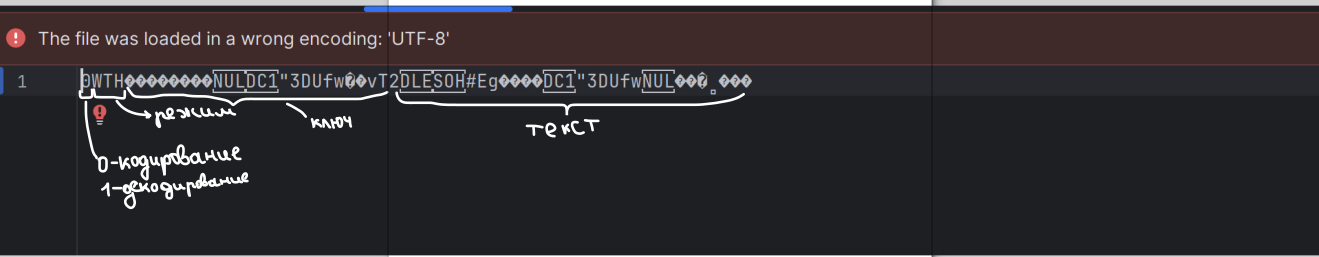
4) Алгоритм зашифрования







Если использовать работу с файлами, а именно сначала записать всё нужное в файл по шаблону ниже, а затем запустить, то:



Первая цифра файла, подающегося программе обозначает, что нужно сделать:

0 — закодировать;

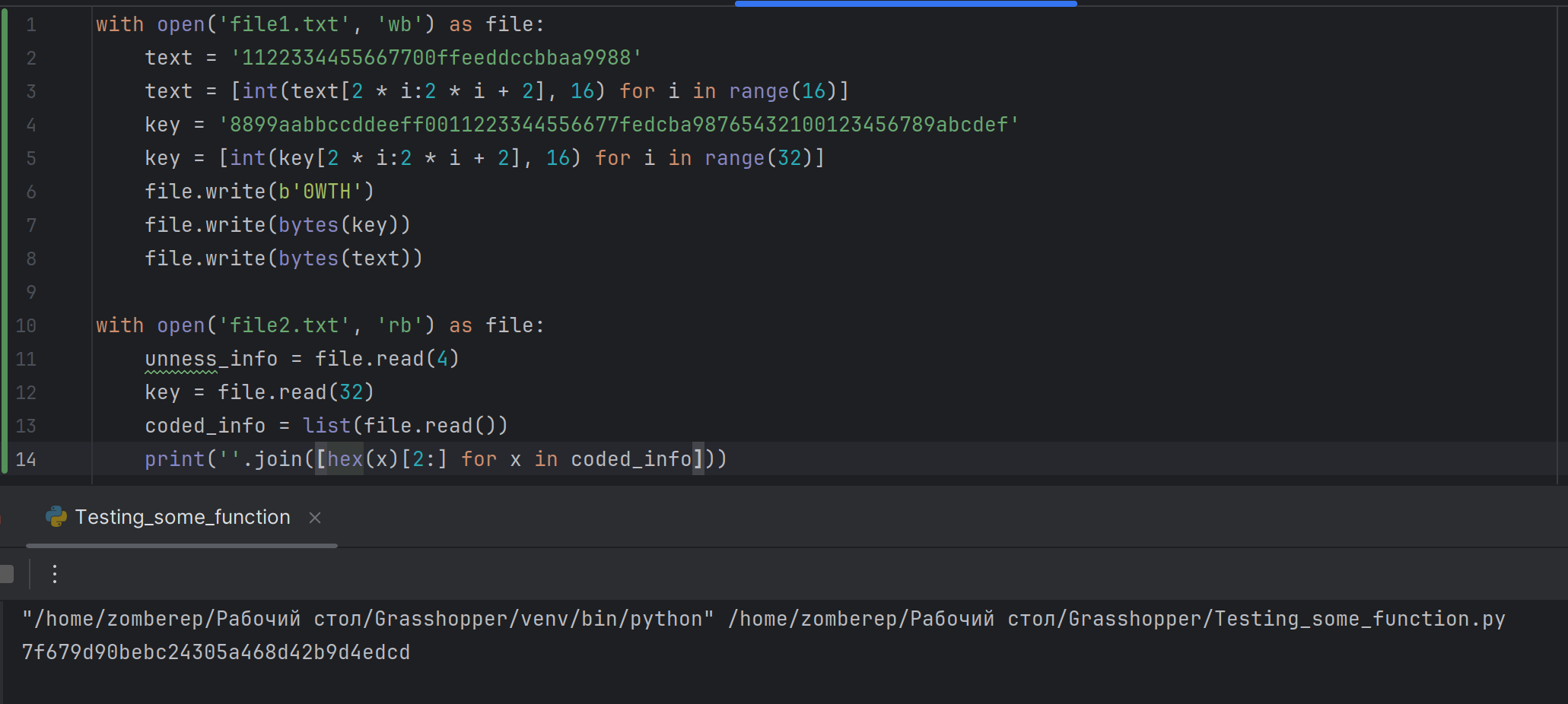
1 — декодировать;

Далее идут три буквы, которые обозначают режим, в котором нужно зашифровать текст:

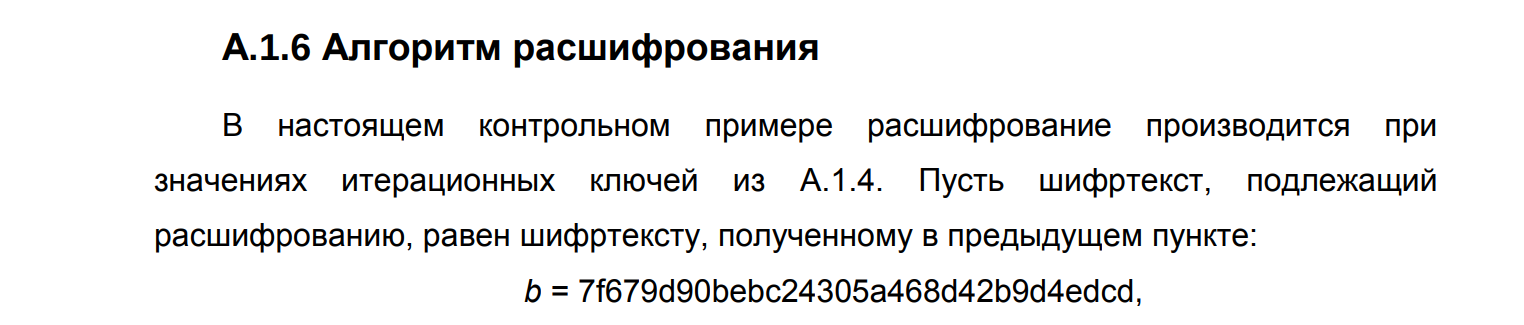
CBC — режим CBC;

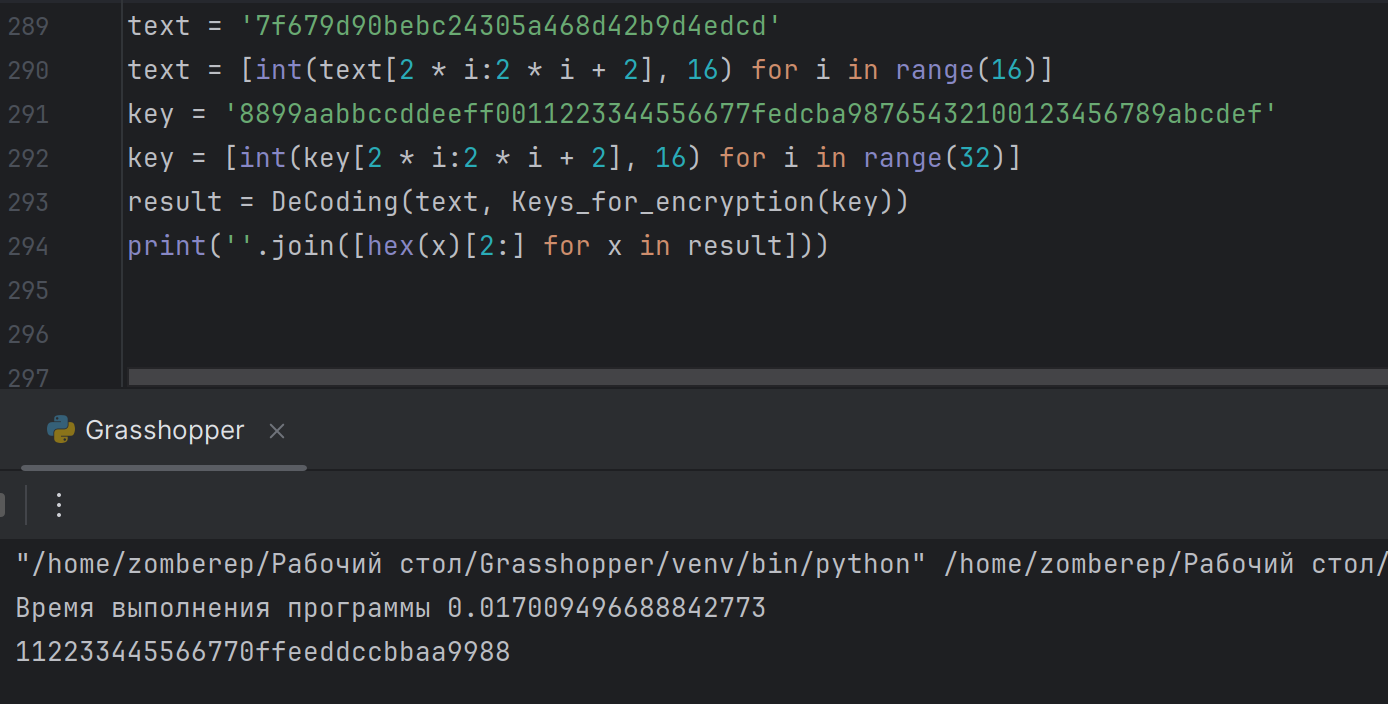
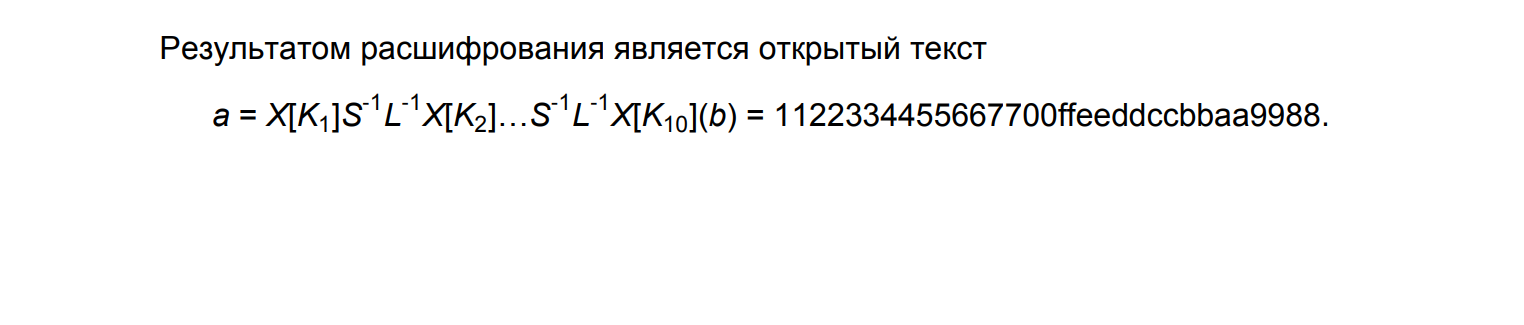
ACP — режим CTR-ACPKM;

WTH — режим ECB;



5) Алгоритм расшифрования



# Описание алгоритма:

1) «Кузнечик» — симметричный алгоритм блочного шифрования с размером блока 128 бит и длиной ключа 256 бит. Основу алгоритма составляет так называемая SP сеть — подстановочно-перестановочная сеть (Substitution-Permutationnetwork). Шифр на основе SP-сети получает на вход блок и ключ и совершает несколько чередующихся раундов, состоящих из стадий подстановки и стадий перестановки. В «Кузнечике» выполняется девять полных раундов, каждый из которых включает в себя три последовательные операции:

1. Операция наложения раундового ключа или побитовый XOR ключа и входного блока данных;

2. Нелинейное преобразование (функция S), которое представляет собой простую замену одного байта на другой в соответствии с таблицей;

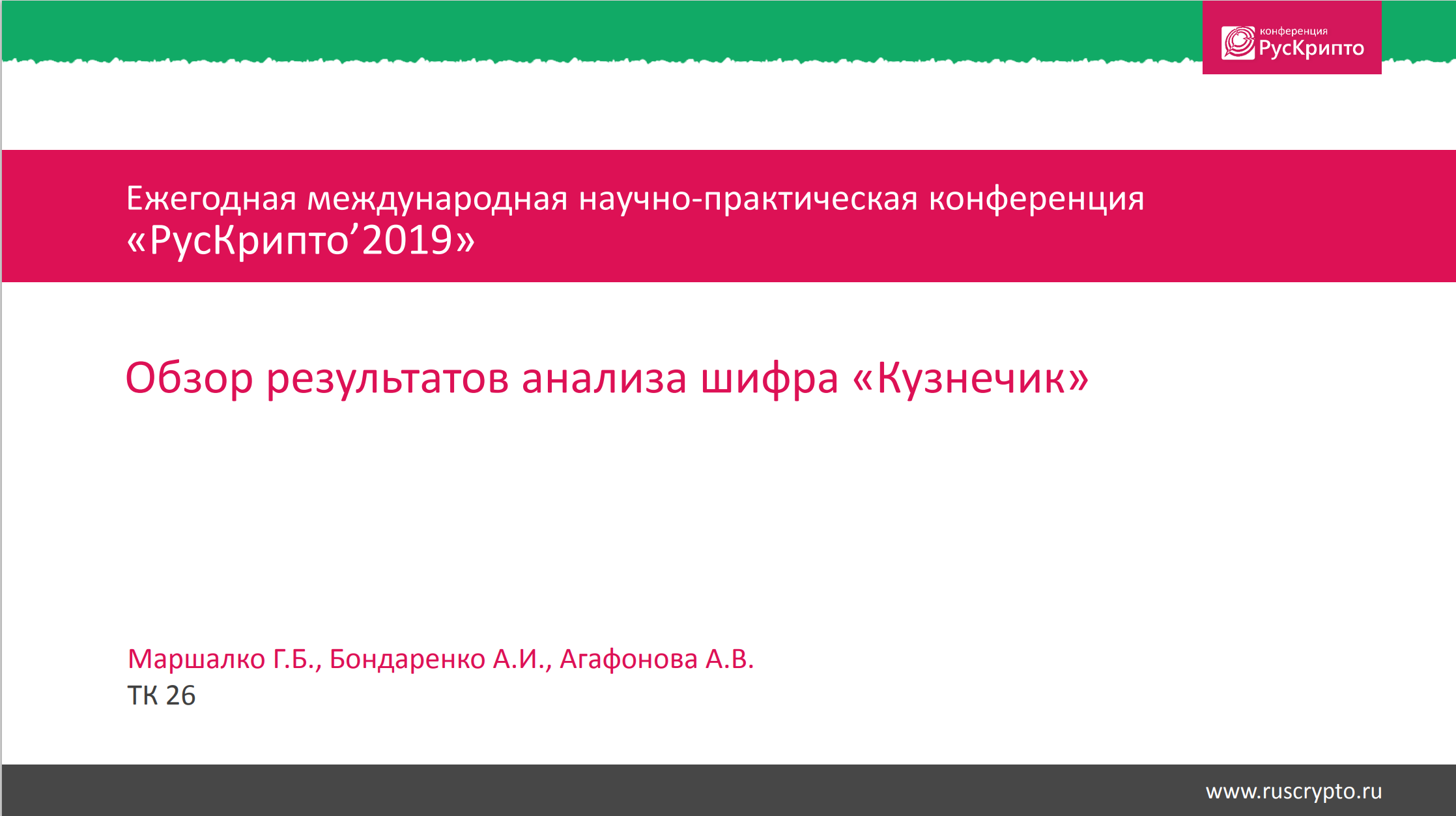
3. Линейное преобразование функция (функция L). Каждый байт из блока умножается в поле Галуа на один из коэффициентов ряда (148, 32, 133, 16, 194, 192, 1, 251, 1, 192, 194, 16, 133, 32, 148, 1) в зависимости от порядкового номера байта. Байты складываются между собой по модулю 2, и все 16 байт блока сдвигаются в сторону младшего разряда, а полученное число записывается на место считанного байта.

Последний десятый раунд *не полный*, он включает в себя только первую операцию XOR.

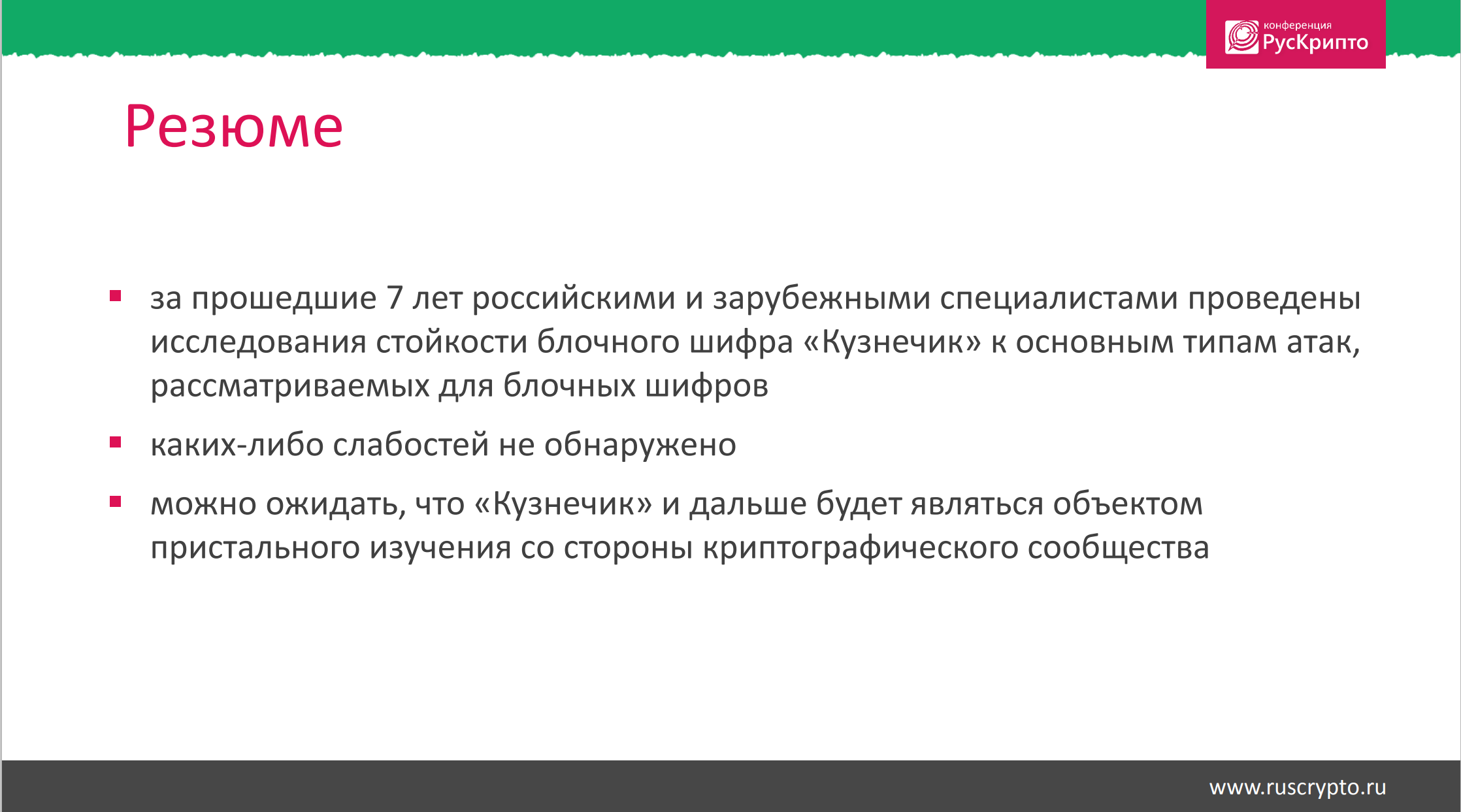
Кузнечик — блочный алгоритм, он работает с блоками данных длинной 128 бит или 16 байт. Длина ключа составляет 256 бит (32 байта)

*Максимальная теоретическая стойкость* алгоритма составляет 2 в степени 256 (115792089237316195423570985008687907853269984665640564039457584007913129639936 вариантов). Количество комбинаций бит, из которых может быть составлен ключ.

*Результаты открытых исследований* стойкости Кузнечика были продемонстрированы в 2019 году:



Ссылка на полную версию презентации: <https://www.ruscrypto.ru/resource/archive/rc2019/files/02_Marshalko_Bondarenko_Agafonova.pdf>



# Тестирование и описание характеристик реализации

## 1) **Режим EBC**

Реализованы функции в соответствии с ГОСТ Р 34.12 2015. А именно *функция L* (линейное преобразование), *функция S* (нелинейное преобразование). Обратные им *функции opp\_S* и *opp\_L* (нужные для реализации расшифровывания). Данные функции принимают на вход список байт.

Далее реализована *функция Keys\_for\_encryption*, которая принимает на вход 32 элементный список и на основе ГОСТ формирует 10 ключей, необходимых для как для шифрования, так и для расшифровывания.

Следующими реализованы функции Coding и DeCoding, которые принимают на вход блок данных и список заранее сформированных ключей. Первая функцию выполняет шифрование текущего блока данных, вторая — расшифровывание.

В завершение добавлена реализованы функции *Standart\_Coding* и *Standart\_DeCoding*, которые принимают на вход считанные данные, формируют из них блоки и выполняют шифрование и расшифровывание соответственно.

*Скорость выполнения тестовых заданий*

* Для одного блока:

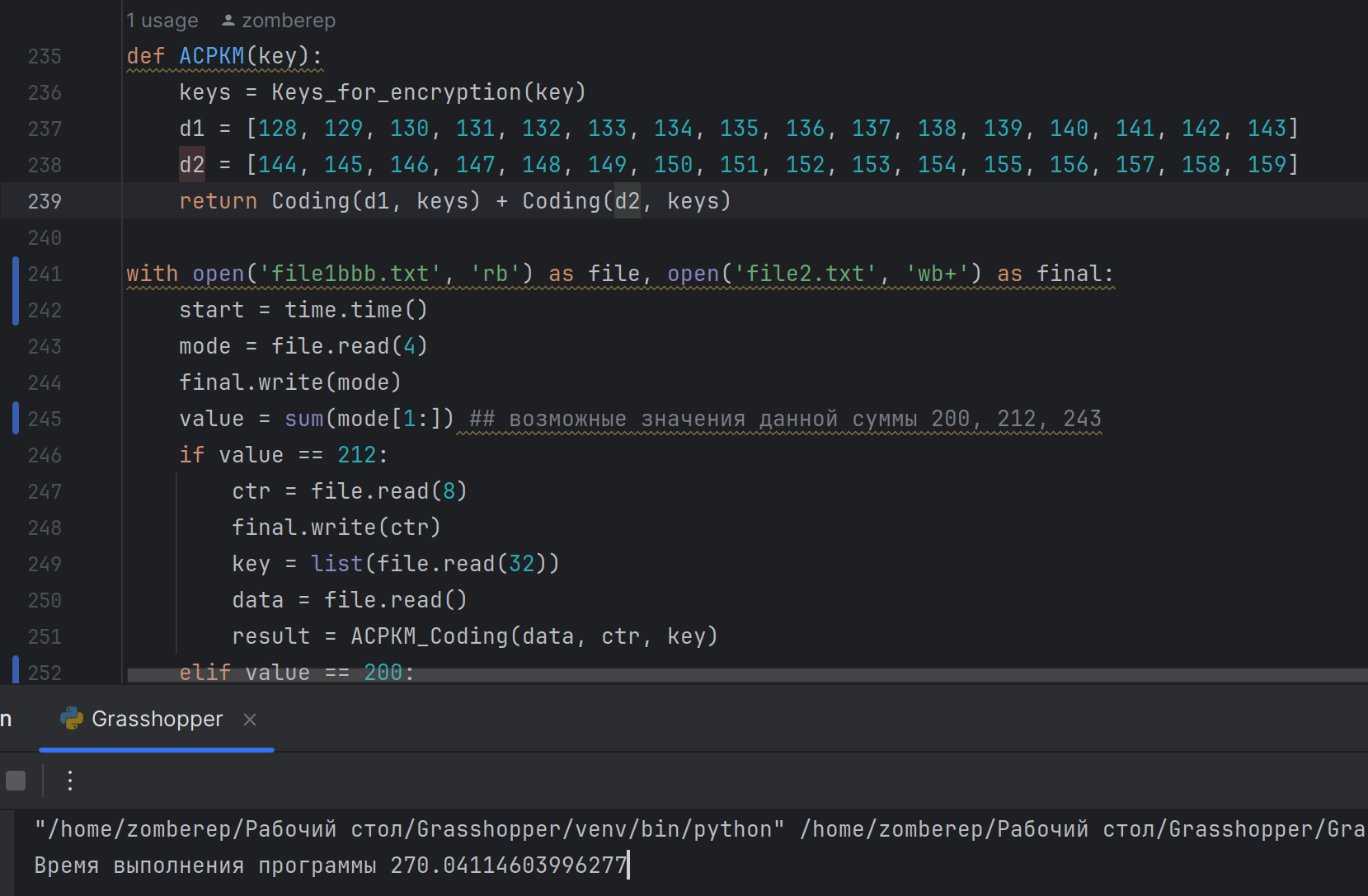
Время выполнения программы **0.012689590454101562 сек.**

* Для 1000 блоков:

Время выполнения программы **1.6874587535858154 сек.**

* Для 1000000 блоков:

Время выполнения программы **270.04114603996277 сек.**



* Для 1 Mb:

Время выполнения программы **110.6569664478302 сек.**

* Для 100 Mb:

*Расчётное* время выполнения программы **3,06 часов**

* Для 1000 Mb:

*Расчётное* время выполнения программы **30,56 часов**

## 2) **Режим CBC**

( [https://ru.wikipedia.org/wiki/Режим\_шифрования#Cipher\_Block\_Chaining\_(CBC](https://ru.wikipedia.org/wiki/Режим_шифрования" \l "Cipher_Block_Chaining_(CBC)) — ссылка на описание )

Для данного режима реализованы две дополнительные функции, а именно *CBC\_Coding* и *CBC\_Decoding*. Внутри них реализуется принцип использования предыдущего блока шифротекста для шифрования следующего. Поскольку для первого блока предыдущего нет, то на вход также требуется вектор инициализации, который будет выполнять роль последнего зашифрованного блока. Функции принимают на вход массив байт и ключ, разбивают его на блоки нужной длины и выполняют шифрования и расшифровывание соответственно.

*Скорость выполнения тестовых заданий*

* Для одного блока:

Время выполнения программы **0.010090827941894531 сек.**

* Для 1000 блоков:

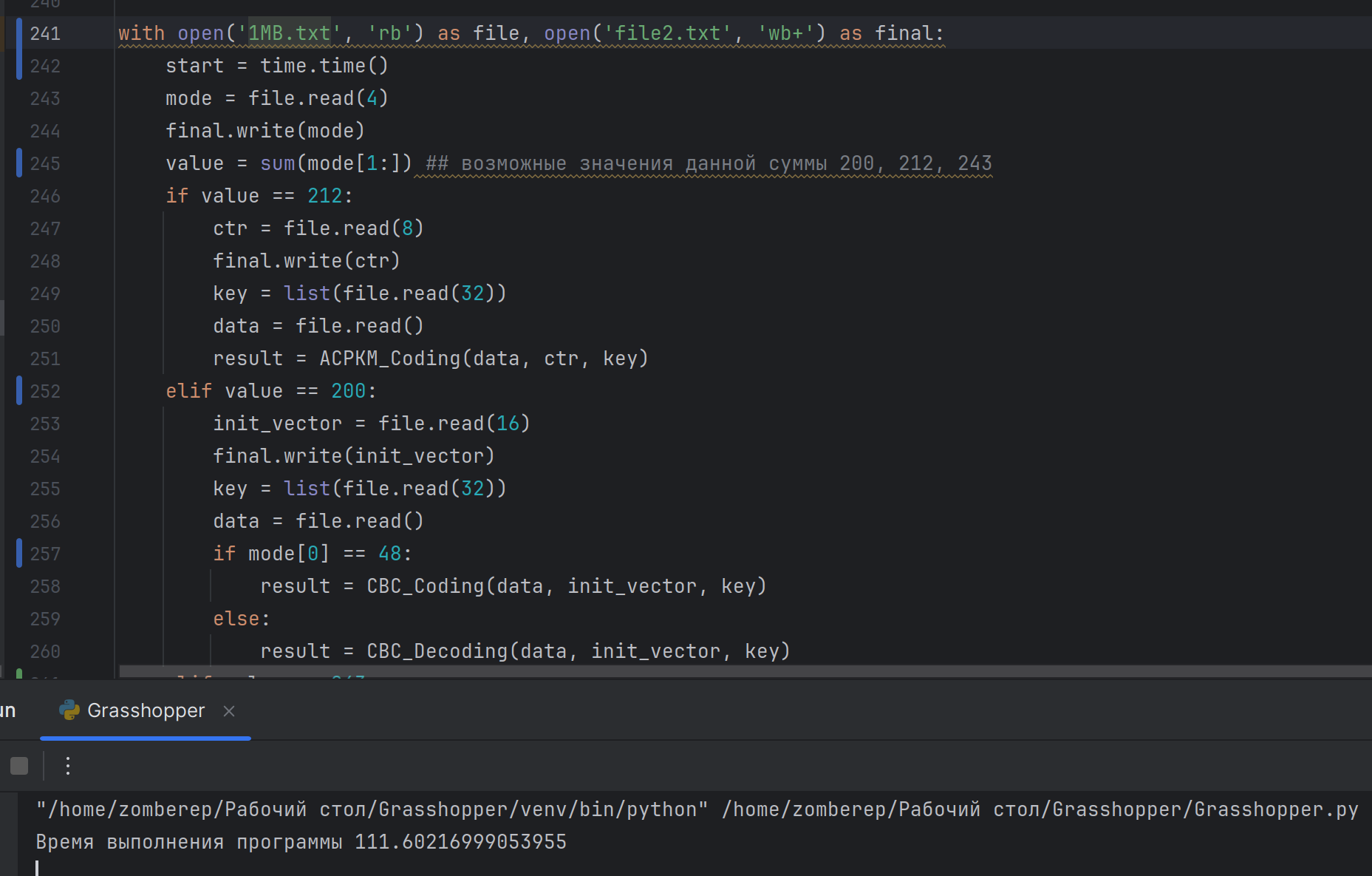
Время выполнения программы **1.6932549476623535 сек.**

* Для 1000000 блоков:

Время выполнения программы **280.9698886871338 сек.**

* Для 1 Mb:

Время выполнения программы **111.60216999053955 сек.**



* Для 100 Mb:

*Расчётное* время выполнения программы **3,1 часа**

* Для 1000 Mb:

*Расчётное* время выполнения программы **31 час**

## 3) **Режим CTR-ACPKM**

( <https://files.stroyinf.ru/Data2/1/4293736/4293736760.pdf> — ссылка на его спецификацию)

Были реализованы функция ACPKM, CTR и ACPKM-Coding. Первая из них принимает на вход ключ и с помощью заданных значений шифрует ключ в режиме Кузнечик, чтобы получить ряд ключей, необходимых для реализации данного режима. Функция CTR работает с одним списком, последнее значение которого на каждом шаге нужно увеличивать на 1. Как только последний элемент будет равен 256, нужно перенести единицу в следующий элемент списка, а последний обнулить. Функция ACPKM\_Coding реализует шифрование. Поскольку расшифровывание не отличается от последнего, то функцию, выполняющую декодинг реализовывать не потребовалось.

*Скорость выполнения тестовых заданий*

* Для одного блока:

Время выполнения программы **0.025417804718017578 сек.**

* Для 1000 блоков:

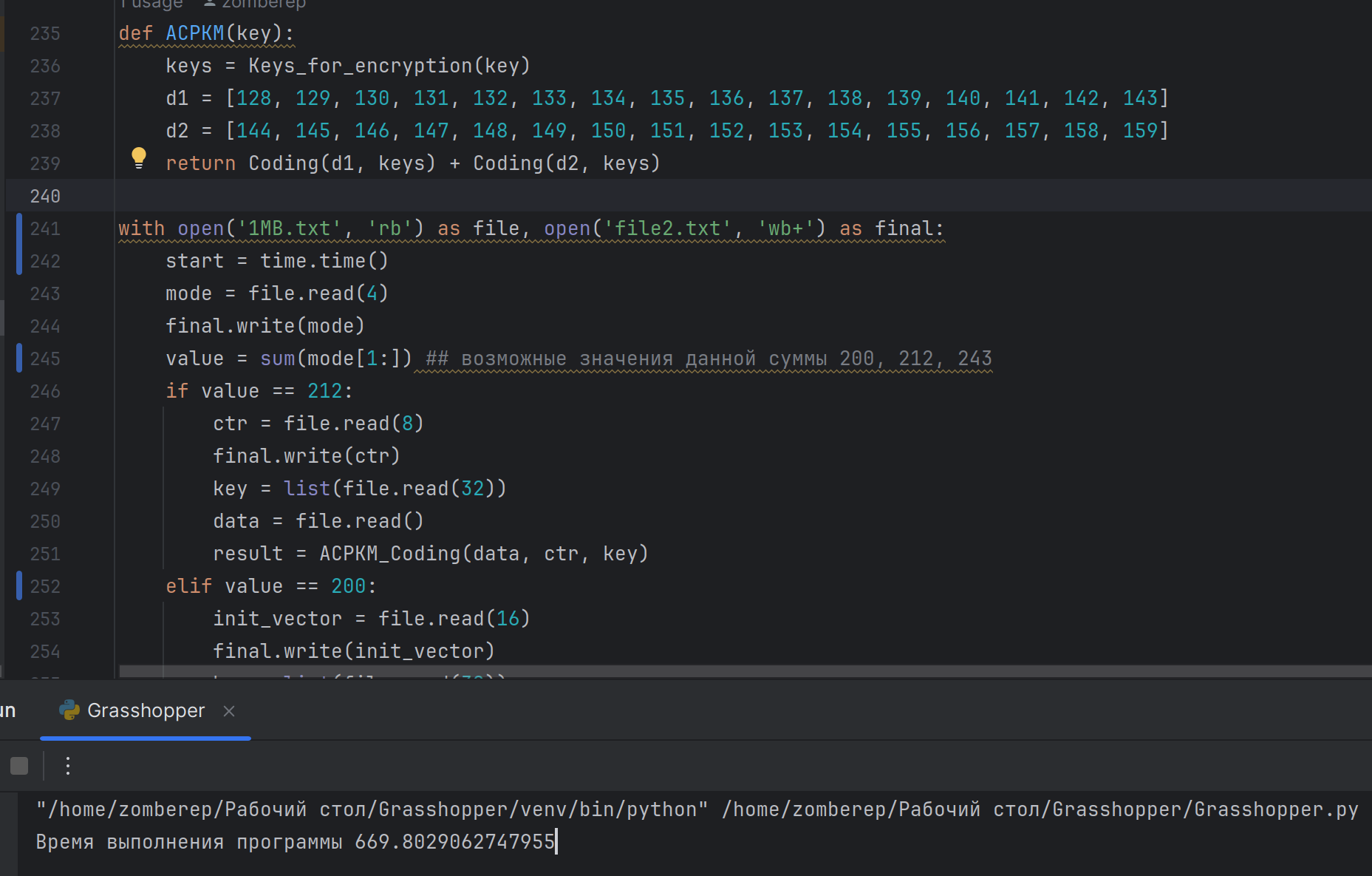
Время выполнения программы **10.001661777496338 сек.**

* Для 1000000 блоков:

*Расчётное* время выполнения программы **27.9 минут**

* Для 1 Mb:

Время выполнения программы **669.8029062747955 сек.**



* Для 100 Mb:

*Расчётное* время выполнения программы **18,61 часов**

* Для 1000 Mb:

*Расчётное* время выполнения программы **7,752 дней**

# Описание тестового стенда

1) Запуск программы производится в интегрированной среде разработки PyCharm;

2) Процессор аппарата, на котором происходит тестирование:

AMD Ryzen 9 7940HS

3) Оперативная память: LDDR5, 4800 МГц

4) Операционная система Linux (дистрибутив Ubuntu)

# Фотографии анализа состояния системы во время выполнения тестирования:

