МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

Институт информационных технологий

Кафедра «Информационная безопасность»

**Пояснительная записка**

к расчетно-графической работе по дисциплине

«Защита программ и данных»

на тему: разработка программной модели вычисления хеш-суммы на основе «функции губки»

Выполнил: студент 2 курса, группы ИБ/б-21-2-о

направления подготовки 10.03.01 «Информационная безопасность» профиль Организация и технология защиты информации

|  |
| --- |
| Толчин Максим Владимирович |

(фамилия, имя, отчество студента)

Руководитель Лихолоб П.Г.,к.т.н.,доцент каф. ИБ

(фамилия, инициалы, степень, звание, должность) (подпись)

Дата допуска к защите « » 20 г.

Зав. кафедрой ИБ М.И. Ожиганова

(подпись) (инициалы, фамилия)

г. Севастополь

2022 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРИТЕТ»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Институт | | информационных технологий | | |
| Кафедра | «Информационная безопасность» | | | |
| Направление подготовки/специальность | | | | 10.03.01 – «Информационная безопасность» |
|  | | | | (код и наименование) |
| Профиль/специализация | | | Организация и технология защиты информации | |
|  | | |  | |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **УТВЕРЖДАЮ** | | |
|  | Заведующий кафедрой «Информационная безопасность» | | |
|  |  |  | М.И. Ожиганова |
|  | (подпись) |  | (инициалы, фамилия) |
|  | «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_202\_года | | |

**З А Д А Н И Е**

**на РГР**

по дисциплине: «Защита программ и данных»

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| студенту | Толчин Максим Владимирович | | | | | | | | | |
|  | (фамилия, имя, отчество) | | | | | | | | | |
| 1. Тема работы (проекта) | | | Разработка программной модели вычисления хеш-суммы на | | | | | | | |
| основе «функции губки» | | | | | | | | | | |
|  | | | | | | | | | | |
| руководитель работы (проекта) | | | | Лихолоб П.Г., к.т.н., доцент кафедры ИБ | | | | | | |
|  | | | | (фамилия, имя, отчество, степень, звание, должность) | | | | | | |
| 2. Срок подачи студентом работы (проекта) | | | | | | 01.12.2022 г. | | | | |
| 3. Входные данные к работе (проекту) | | | | |  | | | | | |
| Предмет: модель хеширования на основе «функции губки» | | | | | | | | | | |
| Объект: методы хеширования текста | | | | | | | | | | |
| Задача: 1. Обзор технологии хеширования 2. Разработка структуры программной модели | | | | | | | | | | |
| хеширования на основе «функции губки» 3. Разработка программной модели хеширования | | | | | | | | | | |
| 4. Тестирование работоспособности программной модели хеширования на основе «функции | | | | | | | | | | |
| губки» | | | | | | | | | | |
| 4. Содержание пояснительной записки (перечень вопросов, которые нужно разработать) | | | | | | | | | |  |
| Введение | | | | | | | | | | |
| 1 Технологии вычисления хеш-суммы | | | | | | | | | | |
| 2 Разработка программного прототипа вычисления хеш-суммы на основе «функции губки» | | | | | | | | | | |
| 3 Проверка работоспособности программной модели хеширования на основе «функции губки» | | | | | | | | | | |
| Заключение | | | | | | | | | | |
| Список использованных источников | | | | | | | | | | |
| 5. Перечень иллюстративного (графического) материала | | | | | | | Презентация в PowerPoint, | | | |
| метод вычисления хеша на основе «функции губки», алгоритм вычисления хеша на основе | | | | | | | | | | |
| «функции губки», результаты выполнения программы, коды программ на Python3 и | | | | | | | | | | |
| JavaScript, коды программ на Python3 и JavaScript, иллюстрации к тексту, результаты | | | | | | | | | | |
| тестирования | | | | | | | | | | |
| 6. Консультанты разделов работы (проекта) | | | | | |  | | | | |
| Раздел | Фамилия, инициалы и должность  консультанта | | | | | | | Подпись, дата | | |
| задание  выдал | задание принял | |
| 1 | Лихолоб П., доцент кафедры ИБ | | | | | | |  |  | |
| 2 | Лихолоб П., доцент кафедры ИБ | | | | | | |  |  | |
| 3 | Лихолоб П., доцент кафедры ИБ | | | | | | |  |  | |
|  |  | | | | | | |  |  | |
|  |  | | | | | | |  |  | |
| 7. Дата выдачи задания | | « 03 » октября 2022 г. | | | | | | | | |

**КАЛЕНДАРНЫЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Название этапов  работы (проекта) | Срок выполнения этапов работы | Примечание |
| 1 | Сбор и анализ материала | 03.10 – 14.10 |  |
| 2 | 3й раздел курсового проекта | 02.10 – 20.10 | описание программ, параметров запуска, входных данных, полученного результата и подключаемых модулей, тестирование разработанной программной модели |
| 3 | 2й раздел курсового проекта | 14.10 – 10.11 | описание программ в виде структурной схемы; описание программ в виде блок-схемы алгоритма, математической модели |
| 4 | 1й раздел курсового проекта: | 08.10 – 17.11 | описание правового поля для использования исследуемых программных прототипов и данных;  описание применяемой в программе терминологии и математических понятий |
| 5 | Оформление пояснительной записки | 08.10 – 01.12 | Согласно ГОСТ 2.105 |
| 6 | Оформление графического материала | 08.10 – 07.12 |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Студент |  |  |  |
|  |  | (подпись) |  | (фамилия и инициалы) |
|  |  |  |  |  |
|  | Руководитель работы (проекта) |  |  | Лихолоб П. |
|  |  | (подпись) |  | (фамилия и инициалы) |

**СОДЕРЖАНИЕ**

Введение 5

1. Технологии вычисления хеша 6

1.1. Правовое поле разработки, производства и распространения средств хеширования 6

1.2. Основные понятия и определения 7

2. Разработка программного прототипа вычисления хеш-суммы на основе «функции губки» 12

2.1. Функция губки 12

2.2. Структура программной модели хеширования на основе «функции губки» 13

2.3. Реализация программной модели вычисления хеша на основе «функции губки» 16

3. Проверка работоспособности программной модели хеширования на основе «функции губки» 19

3.1. Анализ программной модели вычисления хеша на основе «функции губки» 19

3.2. Создание алгоритма для проверки созданной программной модели вычисления хеш-суммы 19

3.3. Алгоритм для тестирования созданной на Python3 программной модели вычисления хеша на основе «функции губки» 20

Заключение 23

Список источников 24

Приложение А 26

Приложение Б 29

**ВВЕДЕНИЕ**

Хеши и хеш-функции являются надежными средствами создания образа данных, из которого невозможно восстановить изначальную информацию. Благодаря хешу происходит защита пароля, защищается цепочка транзакций в финансовых блокчейн сетях, проверяется внесение изменений в системах контроля версий, проверяется целостность информации и так далее [3, 4, 5].

Проблемы безопасности хеш-функций влекут за собой серьезные проблемы в безопасности в критических инфраструктурах. Все время существования хеш-функций существует неустранимая проблема их безопасности, которая носит название коллизии. С помощью этой уязвимости злоумышленник может в короткие сроки подобрать из хеша изначальную информацию и воспользоваться ею [3, 5]. Из-за этого остро стоит проблема совершенствования современных хеш-функций, чтобы уменьшить возможность их «расшифровки», поэтому темой расчетно-графической работы была выбрана разработка своей программной модели вычисления хеша.

Цель расчетно-графической работы: разработать программную модель хеш-функции на основе «функции губки».

Задачи расчетно-графической работы:

1. обзор технологии хеширования;
2. разработка структуры программной модели хеширования на основе «функции губки»;
3. разработка программной модели хеширования;
4. тестирование работоспособности программной модели хеширования на основе «функции губки».

Предмет исследования: модель хеширования на основе «функции губки».

Объект исследования: методы хеширования текста.

На 25 страницах расчетно-графической работы: использовано 7 рисунков, указано 7 формул, 3 листинга и были задействованы 8 источников.

**1 ТЕХНОЛОГИИ ВЫЧИСЛЕНИЯ ХЕША**

**1.1 Правовое поле разработки, производства и распространения средств хеширования**

Регулятором в сфере криптографии (в том числе и хеш-функций) в Российской Федерации выступает Федеральная служба безопасности. Она вправе [7, 8]:

* регулировать разработку, производство, реализацию и эксплуатацию криптографических средств на территории Российской Федерации, а также регулировать услуги в области шифрования и телекоммуникационные системы, расположенные на территории Российской Федерации и использующие криптографические средства;
* разрабатывать, создавать, эксплуатировать информационные системы, системы передачи данных, системы связи, средства защиты информации, в том числе и криптографические средства защиты;
* контролировать функционирование и организацию криптографической и инженерно-технической безопасности сетей связи, которые обеспечивают передачу информации с использованием шифрования;
* контролировать соблюдения режима секретности при обращении шифрованной информации в организациях и подразделениях государственных органов, занимающихся шифрованием информации, а также их учреждениях.

Ввоз и вывоз шифровальных средств осуществляется только при включении соответствующей нотификации или при наличии лицензии.

В соответствии с постановлением правительства Российской Федерации от 16 апреля 2012 г. №313 г. Москва «Об утверждении Положения о лицензировании деятельности по разработке, производству, распространению шифровальных (криптографических) средств, информационных систем и телекоммуникационных систем, защищенных с использованием шифровальных (криптографических) средств, выполнению работ, оказанию услуг в области шифрования информации, техническому обслуживанию шифровальных (криптографических) средств, информационных систем и телекоммуникационных систем, защищенных с использованием шифровальных (криптографических) средств (за исключением случая, если техническое обслуживание шифровальных (криптографических) средств, информационных систем и телекоммуникационных систем, защищенных с использованием шифровальных (криптографических) средств, осуществляется для обеспечения собственных нужд юридического лица или индивидуального предпринимателя)» разработка, производство и распространение криптографических средств подлежит лицензированию [7, 8].

Хеш-функции входят в имитозащитные средства безопасности информации, так как с помощью свойства хеша, согласно которому небольшое изменение входного массива данных приводит к серьезному изменению всей хеш-суммы, информацию проверяют на наличие изменений посредством сравнения исходной хеш-суммы с текущей, следовательно хеш-функции относятся к криптографическим средствам, на которые распространяется лицензия.

Лицензированием юридических лиц и индивидуальных предпринимателей занимается Федеральная Служба Безопасности

Таким образом, создание средств хеширования облагается лицензией Федеральной Службы Безопасности, но в данной расчетно-графической работе создается не криптографическое средство, а программная модель вычисления хеш-функции на основе функции «губки», поэтому данная разработка не попадает под лицензию.

**1.2 Основные понятия и определения**

Далее будут приведены основные определения и понятия необходимые для разработки программной модели вычисления хеш-суммы.

*Хеш (хеш-сумма)* – строка установленной длины, получаемая в ходе преобразования определенным алгоритмом входной последовательности байт.

*Хеширование* – процесс преобразования хеш-функцией входной строки байт в хеш.

*Хеш-функция* – функция, которая преобразует исходную строку байт в хеш.

*Коллизия* – явление, при котором две разные входные строки генерируют равные хеш-суммы [3, 4, 5].

На рисунке 1 представлено множество всех входных строк хеш-функции *X* и связанное с ним множество выходных хеш-сумм *Y*. На данном рисунке видно, как входные строки 3 и 4 сходятся в одну точку *C* в области *Y*, что и называется коллизией.

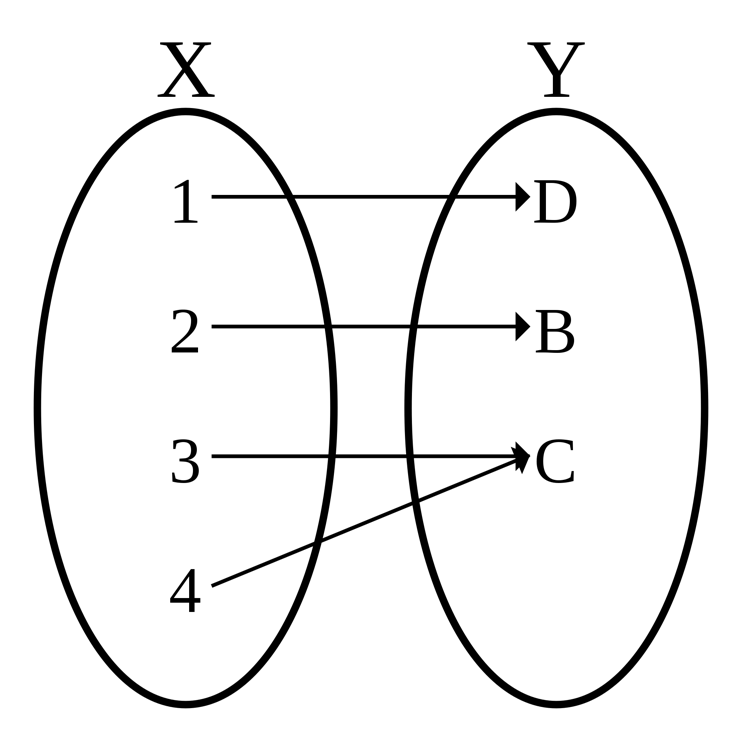


Рисунок 1 – Визуальное представление коллизии

Коллизии являются серьезной угрозой для многих систем, которые используют хеш. С помощью алгоритмов реализации коллизий злоумышленник может подбирать нужную последовательность символов для обхождения системы аутентификации или подделки сообщений о проведении валютной операции [3].

**Хеш-функция должна обладать следующими свойствами:**

* вычисленный хеш должен быть уникален для разных входных массивов байт, но при этом допускается минимальное количество коллизий;
* хеш-функция должна быть необратимой, то есть из хеша нельзя вычислить исходную информацию;
* хеш должен быть установленной длины для любого размера исходной последовательности байт;
* незначительное изменение исходной информации приводит к значительному изменению получаемой хеш-суммы [4, 5].

Хеш-функция должна обладать стойкостью к коллизиям и вычислительной сложность, которая позволяет вычислять хеш-сумму за адекватный промежуток времени. Устранение коллизий реализуется различными методами, например, методом добавления к входному массиву данных случайной последовательность символов, которую называют «солью», конкатенацией различных хэш-функций (склеивание хеш-сумм, которые были вычислены различными хэш-функциями для одного и того же массива данных) [3].

В алгоритмах для вычисления хеш-суммы используются различные побитовые операции для преобразования битовых строк. В создаваемой программной моедли будут использоваться *XOR*, отрицание, дизъюнкция и побитовое смещение.

*Исключающее «или»* – функция, которая возвращает «истинное» значение при сравнении двух битов при условии, что только один из аргументов является истинным, а другой ложным. Обозначается символом . Графическое представление исключающего «или» представлено на рисунке 2.

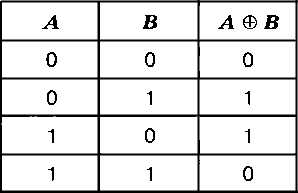


Рисунок 2 – Таблица истинности для исключающего «или»

*Отрицание* – булевая функция, которая инвертирует истинное значение в ложное, а ложное в истинное. Обозначается, как .



Рисунок 3 – Таблица истинности для отрицания

*Дизъюнкция (логическое «или»)* – логическая функция, которая возвращает истинное значение, если хотя бы одно из входных значений истинное, иначе функция возвращает отрицательное значение. Обозначается символами *.* Диаграмма, отражающая суть дизъюнкции, представлена на рисунке 3.

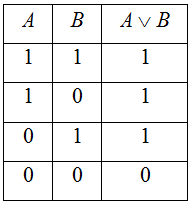


Рисунок 4 – Таблица истинности для дизъюнкции

*Побитовое смещение влево* – функция, которая сдвигает последовательность бит на заданное число позиций. Обозначается:

где ***x*** – битовая строка; – символ операции смещения влево; *N* – количество позиций, на которое нужно сдвинуть строку.

*Побитовое смещение вправо* – аналогично смещению влево, но осуществляет смещение битовой строки на заданное количество позиций в обратном направлении. Обозначается:

где – символ операции смещения вправо [6].

Также часто при реализации алгоритмов хеширования используется вычисление остатка по делению. Данная операция имеет обозначение

Еще важно отметить, что в качестве входных данных будут выступать строки состоящие из символов, которые входят в кодировку *UTF-8*. В UTF-8 представлено множество символов, состоящее из латиницы, кириллицы, иероглифов, цифр, математических символов, эмоджи и прочих символов.

Таким образом, создаваемая программная модель будет опираться на перечисленные выше понятия, математические функции и требования к хеш-функциям.

**2. РАЗРАБОТКА ПРОГРАММНОГО ПРОТОТИПА ВЫЧИСЛЕНИЯ ХЕШ-СУММЫ НА ОСНОВЕ «ФУНКЦИИ ГУБКИ»**

**2.1 Функция губки**

Программный прототип вычисления хеша в расчетно-графической работе будет основываться на «функции губки».

*Функция губки* – функции с установленной длиной внутреннего состояния, которые принимают на вход двоичную строку любой длины и выдают на выход тоже строку произвольной длины.

Битовая скорость (*r*) и мощность (*с*) для реализуемой модели равны 576 и 1024 соответственно. Данные значения были выбраны самостоятельно по подобию алгоритму SHA3(Keccak).

Функция губки имеет две стадии: впитывание и выжимание. На стадии впитывания части входной строки проходят операцию *XOR* со строкой внутреннего состояния, которая в свою очередь каждый шаг проходит через ряд преобразований в функции *f*.

На стадии «выжимания» на выход подается внутреннее состояние, которое тоже проходит ряд тех же преобразований.

Хеш-функция на основе «функции губки» будет работать следующим образом:

1. на вход подается строка произвольной длины (*P*);
2. она переводится в байтовое представление и дополняется до нужной длины;
3. строка байтов делится на блоки ();
4. строк, поделенная на блоки, проходит стадию «впитывания»;
5. состояние проходит стадию выжимания, доводя выход (*Z*) до установленной длины, результатом которой будет хеш-сумма [2].

На рисунке 5 представлена структурная схема робы функции губки.

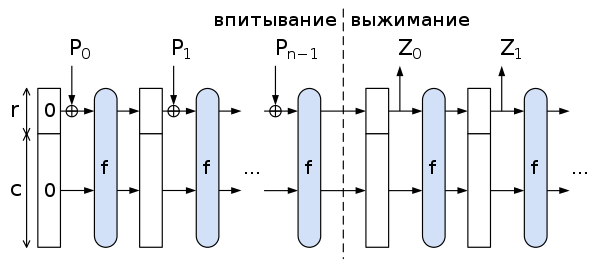


Рисунок 5 – Структурная схема работы функция губки

В итоге «функция губки» преобразует входной массив данных любой длины (P) в выходной массив данных установленной длинны (Z).

Таким образом, программная модель хеш-функции будет основываться на «функции губки», так как она позволяет реализовать основные требования к хеш-функции.

**2.2 Структура программной модели хеширования на основе  
«функции губки»**

На первом этапе в функцию подается текстовая строка, над которой будут происходить дальнейшие преобразования. Далее эта строка представляется в виде последовательности байтов согласно кодировки «*Utf-8*».

*Кодировка* – способ представления символов в виде числовых идентификаторов.

Далее строка байтов дополняется следующим образом:

1. первый байт, которым дополняется байтовая строка, равняется первому байту исходной строки (при получении пустой строки приравнивается 0);
2. все байты между первым и последним байтами дополнения приравниваются следующему значению: первые 4 бита байта дополнения – число равное первой цифре длины исходной строки, последние 4 бита байта дополнения – число равное первой цифре суммы байтов, представляющих исходную строку, например, если исходная строка имеет длину 65 и сумму байтов 134, то строка будет дополнятся битами 0x61;
3. последний байт приравнивается последнему байту исходной байтовой строки (при получении пустой строки приравнивается 0).

Дополненная строка делится на *n* частей длинной 576 бит, то есть произведение 576 на *n* будет равно длине дополненной строки.

Далее происходит впитывание каждого блока впитанной строки. Как можно увидеть из рисунка 6, каждый блок поделенной строки проходит операцию *XOR* со строкой состояния, над которой потом проводятся перестановки.



Рисунок 6 – Блок-схема впитывания входной строки во внутреннюю строку состояния

Функция перестановок представляет из себя пять последовательных шагов, в которых над элементами строки состояния происходят различные побитовые операции. Представим строку состояния как трехмерную матрицу *S* размером , где каждый элемент является соответствующим битом строки состояния. Эта матрица передается как аргумент функции перестановок.

Введем следующие обозначения:

* ;
* – идентификатор страницы трехмерной матрицы;
* – идентификатор строки трехмерной матрицы;
* – идентификатор столбца трехмерной матрицы;
* – переменная для смещения строки бит на различные значения.

Операции, проводимые со строкой состояния в функции перестановок:

Далее происходит «выжимание» функции состояния по схеме, представленной на рисунке 7.



Рисунок 7 – Блок-схема «выжимания» внутренней строки состояния

Результатом работы программы является преобразование входной строки в последовательность бит установленной длины в 512 символов.

Таким образом программная модель вычисления хеша на основе «функции губки» будет создаваться согласно структуре, описанной в этом подразделе.

**2.3 Реализация программной модели вычисления хеша на основе «функции губки»**

В соответствии со структурной моделью из рисунка 2.1 была создана программная модель на языке *Python3*, которую можно разделить на три части:

1. функция дополнения исходной строки;
2. функция перестановок;
3. функция вычисления хеш-суммы.

В приложении А представлен код программной модели, где представлена функция преобразования и дополнения исходной строки до нужной длины. В листинге 1 представлен основной код функции, который дополняет исходную строку необходимыми байтами. Сначала вычисляется длина исходной строка. Далее она конвертируется в последовательность байт функцией *bytearray()* и переводится из последовательности байт в последовательность шестнадцатеричных представлений этих байтов. Далее эта последовательность байтов дополняется в соответствии со структурной моделью программной модели, делится на блоки и переводится в двоичное представление.

Листинг 1 – Код дополнения исходной строки

def padding(M):

lengM = len(M)

g = bytearray(M.encode('utf-8'))

temp = list(map(int, g))

g = list(map(hex, g))

lengG = len(g)

temp = sum(temp)

for i in range(len(g)):

g[i] = g[i][2:]

if len(g[i])!=2:

g[i] = '0'+g[i]

try:

g.append(g[0])

except:

g.append('01')

while((len(g)\*8)%576!=576-8):

g.append(str(lengM)[0]+str(temp)[0])

g.append(g[lengG-1])

В приложении А также представлена функция перестановок, которая проводит ряд побитовых математических операций над строкой состояния в соответствии с формулами (3) - (7).

Также в приложении А представлена основная функция – функция вычисления хеша на основе «функции губки». В листинге 2 представлен код функции вычисления хеша. Функция принимает на вход символьную строку. Далее она проходит функцию дополнения, в ходе которой получается разбитая на n блоков дополненная исходная строка – *P*. Каждый блок проходит операцию *XOR* со строкой состояния, которая проходит 24 раунда перестановок с помощью описанной выше функции перестановок. Далее состояние, параллельно проходя по 24 раунда перестановок, подается на выход до длины в 512 бит.

Листинг 2 – Функция вычисления хеш-суммы

def GetHash(M):

S =[[[0 for k in range(64)] for i in range(5)] for j in range(5)]

P = padding(M)

r=[]

for Pi in P:

for i in range(5):

for j in range(5):

for k in range(64):

if i+j\*5<576/64:

S[i][j][k] = S[i][j][k] ^ Pi[i+j\*5][k]

for i in range(24):

S = peres(S)

Z = ""

while True:

for i in range(5):

for j in range(5):

if 5\*i+j < 576/64:

Z = Z + hex(int('0b'+''.join(map(str, S[j][i])),2))[2:]

for i in range(24):

S = peres(S)

if len(Z)>128:

break

return Z[0:128]

Таким образом, была создана программная модель хеш-функции на основе «функции губки» согласно структуре, описанной в разделе 2.2.

**3. ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОЙ МОДЕЛИ ХЕШИРОВАНИЯ НА ОСНОВЕ «ФУНКЦИИ ГУБКИ»**

**3.1 Анализ программной модели вычисления хеша на основе «функции губки»**

Программная модель создавалась с упором на универсальность, то есть реализованный алгоритм может принимать на вход разные последовательности символов, которые могут состоять из символов латиницы, кириллицы, иероглифов, математических символов и так далее. Данную универсальность входных данных реализует кодировка *Utf-8*.

В основном на длительность работы программы влияет количество впитываемых блоков. С каждым увеличением входной строки байтов на 576 бит будет добавляться один блок, для которого должна будет выполняться процесс впитывания в строку состояния и функция перестановок. Количество операций выполняемых при впитывании исходной строки линейно возрастает с увеличением количества блоков, на которые делится исходная строка, следовательно вычислительная сложность хеш-функции суммы равна *O(N).*

Написанная программная модель способна принимать различные входные значения, а также эффективно справляется вычислением хеш-суммы, так как она имеет линейную вычислительную сложность.

**3.2 Создание алгоритма для проверки созданной программной модели вычисления хеш-суммы**

На основе структурной схемы из рисунка 5 был создан аналогичный программной модели на *Python3* алгоритм на языке *JavaScript*, который представлен в приложении Б.

Главные отличия алгоритма на *JavaScript* от алгоритма на языке программирования *Python3*:

1. были реализованы переводы из шестнадцатеричной системы исчисления и двоичной системы исчисления в десятичную с типом данных *BigInt*, так как встроенные методы не могут работать с большими числовыми значениями;
2. более высокая скорость работы из-за более эффективной работы языка *JavaScript* с типами данных, чем у *Python3*;
3. более объемный код программы, чем у кода на *Python3.*

Таким образом, благодаря алгоритму на другом языке программирования, созданного по той же математической модели, что и реализация на языке Python3, будет протестирована работоспособность созданной программной модели.

**3.3 Алгоритм для тестирования созданной на Python3 программной модели вычисления хеша на основе «функции губки»**

Был реализован алгоритм на языке *Python3*, который сравнивает результаты работы двух алгоритмов (на *JavaScript* и на *Python3*), на вход которых поступают строки из датасета, состоящего из строк, написанных на разных языках, с пробелами и без, с цифрами и без, одинаковых строк с небольшими изменениями, строк псевдослучайных последовательностей символов.

Алгоритм, представленный из листинга 3., запускает из терминала программы вычисления хеша на *JavaScript* и *Python3* с предустановленными параметрами из датасета, которые в свою очередь записывают результат работы для данной строки из датасета в отдельные файлы. Далее записанные в файлы значения сравниваются между собой, и алгоритм запоминает ошибки и соответствующие им значения входной строки и выводит их, если сравниваемые хеши различаются.

Листинг 3 – Код программы тестирования алгоритмов вычисления хеш-суммы на основе «функции губки»

import os

f = open("Blog/test.txt")

b=True

errors = dict()

i = 0

hash = []

val = []

for line in f.readlines():

py = os.system("node /Users/lrnby/YourBlog/Blog/hash.js "+line)

js = os.system("python3 Blog/hash\_sha3.py "+line)

f\_p = open("Blog/test\_res\_py.txt", "r")

f\_j = open("Blog/test\_res\_js.txt", "r")

fp = f\_p.readlines()

fj = f\_j.readlines()

if fp in hash:

print("Обнаружена коллизия:",)

print(line)

print(val[hash.index(fp)])

with open("Blog/coll.txt",'a') as fc:

print("Обнаружена коллизия:", file=fc)

print(line, file=fc)

print(val[hash.index(fp)], file=fc)

val.append(line)

hash.append(fp)

with open("Blog/log.txt","a") as fl:

if fp[0][0:len(fp[0])-1].replace('\n', '') != fj[0]:

errors[i]=line

b = False

if len(line)>10:

print("№ "+ str(i) + " " + line[:10] +"... FALSE", file=fl)

else:

print("№ "+ str(i) + " " + line.replace('\n','') +" FALSE",file=fl)

else:

if len(line)>10:

print("№ "+ str(i) + " " + line[:10] +"... TRUE",file=fl)

else:

Продолжение листинга 3

print("№ "+ str(i) + " " + line.replace('\n','') +" TRUE",file=fl)

i+=1

if b:

print("Во время тестирования не возникло ошибок")

else:

print("Ошибки возникли во время тестов: "+', '.join(map(str, list(errors.keys()))))

a = input("Если хотите получить значения входных строк, при которых возникли ошибки, введите Y: ")

if a == "Y":

for i in list(errors.values()):

print()

print(i)

В ходе тестирования было установлено, что разработанная программная модель правильно вычисляет хеш суммы для различного вида входных данных. С результатами проведенного тестирования можно ознакомиться по ссылке [1].

**ЗАКЛЮЧЕНИЕ**

В ходе расчетно-графической работы была создана программная модель вычисления хеш-суммы на основе «функции губки», которая обеспечивает выполнение всех требования к созданной хеш-функции. «Функция губки» принимает на вход данные любой длины и преобразует их в выходной массив данных установленной длины. Также функция преобразований внутри «функции губки» и многочисленное проведение этих преобразований с помощью побитовых операций позволяют сделать функцию необратимой и выдавать на выход сильно различающиеся хеш-суммы при незначительном изменении входных данных.

Программная модель была реализована на языках программирования *Python3* и *JavaScript*. Наибольшая скорость работы алгоритма получилась на языке программирования *JavaScript*. Для увеличения скорости работы алгоритма на *Python3* можно прибегнуть к таким библиотекам, как *Numpy* и *Cython*, или же реализовать его как библиотеку, написанную на языке *C/C++.*

Также было проведено тестирование созданной программной модели, в ходе которого было выяснено, что она может обрабатывать входные массивы данных, которые имеют различную длину и представлены различными наборами символов, например, латиницей, кириллицей, иероглифами, математическими символами и так далее. Так же было установлено, что созданный алгоритм обладает линейной вычислительной сложностью. Это означает, что созданная программная модель будет вычислять хеш-сумму примерно на одно и то же количество операций больше с увеличением объема входного массива данных.

**СПИСОК ИСТОЧНИКОВ**

1. Google диск : сайт. – URL: https://drive.google.com/file/d/1FQff6YHxRzvIkrbA33UKOMls2x0aEwGF/view?usp=share\_link
2. Keccak sponge function family main document / G. Bertoni, J. Daemen, M. Peeters, G.V. Assche // ResearchGate : электронный научный журнал. – URL: https://www.researchgate.net/profile/Michael-Peeters-3/publication/265099836\_Keccak\_sponge\_function\_family\_main\_document/links/54745a970cf29afed60f79e2/Keccak-sponge-function-family-main-document.pdf. – Дата публикации: 19.06.2010.
3. Владстон Ф.Ф. Теоретический минимум по Computer Science / Ф.Ф. Владстон, М. Пиктет. – СПб : Питер, 2022. – 129-137 с.
4. Данилов Н.Н. Криптовалюта. Необходимый минимум знаний / Н.Н. Данилов. – Любляна : Николай Данилов, 2022. – 74-79 с.
5. Комарова Ю.А. ХЕШ-ФУНКЦИИ И ИХ ПРИМЕНЕНИЕ / Ю.А. Комарова // Материалы V Международной научно-практической конференции (школы-семинара) молодых ученых . – Кострома : Издатель Качалин Александр Васильевич, 2019. – С. 47-49.
6. О битовых операциях // Tproger : сайт. – URL: https://tproger.ru/translations/bitwise-operations/?ysclid=lb3xhlnjmz958256068 (дата обращения: 11.12.2022)
7. Об утверждении Положения о лицензировании деятельности по разработке, производству, распространению шифровальных (криптографических) средств, информационных систем и телекоммуникационных систем, защищенных с использованием шифровальных (криптографических) средств, выполнению работ, оказанию услуг в : Постановление Правительства Российской Федерации // Собрание законодательства Российской Федерации. – 2012. – № 313.
8. Об утверждении Положения о разработке, производстве, реализации и эксплуатации шифровальных (криптографических) средств защиты информации (Положение ПКЗ-2005) : Приказ ФСБ РФ от 9 февраля 2005 г. // ы. – 2005. – № 66.

**ПРИЛОЖЕНИЕ А**

**Листинг – Реализации программной модели вычисления хеша на основе «функции губки» на языке *Python3***

#функция дополнения и разбиения на блоки

def padding(M):

lengM = len(M)

g = bytearray(M.encode('utf-8'))

temp = list(map(int, g))

g = list(map(hex, g))

lengG = len(g)

temp = sum(temp)

for i in range(len(g)):

g[i] = g[i][2:]

if len(g[i])!=2:

g[i] = '0'+g[i]

try:

g.append(g[0])

except:

g.append('01')

while((len(g)\*8)%576!=576-8):

g.append(str(lengM)[0]+str(temp)[0])

g.append(g[lengG-1])

size = len(g)\*8/576

P = [[] for i in range(int(size))]

count = 0

j = 0

i = 0

for ch in g:

if j > 576/64-1:

j=0

i+=1

count+=1

if count\*8%64 == 0:

temp = g[int(count-64/8):count]

st = temp[0]

for c in range(1,8):

st+=temp[c]

st = bin(int(st,16))

st = st[2:]

if len(st)<64:

while len(st)<64:

st = "0"+st

row = list(map(int, st))

P[i].append(row)

j+=1

return P

#функция перестановок

def peres(A):

A1 =[[[0 for k in range(64)] for i in range(5)] for j in range(5)]

**Продолжение листинга**

# 1 шаг

for i in range(5):

for j in range(5):

for k in range(32):

for t in range(28):

A1[i][j][k] = (A[(i+2)%5][(j+1)%5][(k+(t\*3\*t))%64]^1)^A[i][j][k]

A = A1

A1 =[[[0 for k in range(64)] for i in range(5)] for j in range(5)]

# 2 шаг

for i in range(5):

for j in range(5):

for k in range(64):

A1[i][j][k]=A[i][j][k]^A[5-1-i][(j+1)%5][k]

A = A1

A1 =[[[0 for k in range(64)] for i in range(5)] for j in range(5)]

# 3 шаг

for i in range(5):

for j in range(5):

for k in range(64):

A1[i][j][k] = (A[i][j][(k+1)%64]^1)^(A[i][(j+1)%5][k]^1)

A = A1

A1 =[[[0 for k in range(64)] for i in range(5)] for j in range(5)]

# 4 шаг

for i in range(5):

for j in range(5):

for k in range(64):

A1[i][j][k] = A[(i-2)%5][j][k]|A[j][i][(k+32)%64]

A = A1

A1 =[[[0 for k in range(64)] for i in range(5)] for j in range(5)]

# 5 шаг

for i in range(5):

for j in range(5):

for k in range(64):

A1[i][j][k] = (A[(i+4)%5][j][k]^1)^(A[j][i][(k-44)%64]^A[j][i][(k-43)%64])

A = A1

return A

# получение хэш-суммы

def GetHash(M):

S =[[[0 for k in range(64)] for i in range(5)] for j in range(5)] #первоначальное состояние

**Продолжение листинга**

P = padding(M) #дополнение и разбиение на блоки исходного сообщения

r=[]

for Pi in P: #впитывание блока в состояниие

for i in range(5):

for j in range(5):

for k in range(64):

if i+j\*5<576/64:

S[i][j][k] = S[i][j][k] ^ Pi[i+j\*5][k]

for i in range(24): #выполнение 24 раундов перестановок

S = peres(S)

Z = ""

while True: #генерация вывода длиной 512 бит

for i in range(5): #добавление к выводу последовательностей битов из состояния

for j in range(5):

if 5\*i+j < 576/64:

Z = Z + hex(int('0b'+''.join(map(str, S[j][i])),2))[2:]

for i in range(24): #выполнение 24 раундов перестановок для состояния

S = peres(S)

if len(Z)>128:

break

return Z[0:128] #возвращение функцией хэш-суммы

print(GetHash(""))

**ПРИЛОЖЕНИЕ Б**

**Листинг – Реализации программной модели вычисления хеша на основе «функции губки» на языке *JavaScript***

function padding(string){

var te = new TextEncoder("utf-8");

var temp = te.encode(string);

var g = new Array();

for (var i=0; i < temp.length; i++){

g[i] = temp[i].toString(16)

if (g[i].length <2){

g[i]="0"+g[i];

}

}

lengM = string.length;

lengG = g.length;

temp = temp.reduce(function(sum, elem) {

return sum + elem;

}, 0);

i = lengG

if (g[0]==undefined){

g[0] = "00"

}

g[g.length]=g[0]

i = lengG+1;

while (((g.length\*8)%576)!=576-8){

g[i]= String(lengM)[0]+String(temp)[0]

i++

}

if(lengG<1){

lengG+=1

}

g[i]=g[lengG-1]

size = g.length\*8/576

P = [[]]

count = 0

j=0

i=0

for (var ch = 0; ch<g.length; ch++){

if (j>((576/64)-1)){

j=0

i+=1

P[i]=[]

}

count+=1

if (count\*8%64 == 0){

temp = g.slice(count-64/8, count)

st = temp[0]

for (var c = 1; c<8; c++){

st+=temp[c]

}

var num\_st = 0n

**Продолжение листинга**

for (var k =0; k<st.length;k++){

var tp = st[k];

if(st[k]=='a'){

tp = 10n

}

if(st[k]=='b'){

tp = 11n

}

if(st[k]=='c'){

tp = 12n

}

if(st[k]=='d'){

tp = 13n

}

if(st[k]=='e'){

tp = 14n

}

if(st[k]=='f'){

tp = 15n

}

tp = BigInt(Number(tp))

num\_st += BigInt(Math.pow(16, 15-k)) \* tp

}

st = num\_st.toString(2)

if (st.length < 64){

while(st.length<64){

st = "0"+st

}

}

row = []

for (var k = 0; k < st.length; k++){

row[k] = Number(st[k])

}

P[i][j]=row

j+=1

}

}

return P

}

function peres(A){

A1 =[[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

**Продолжение листинга**

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]],[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]],[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]],[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]],[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]]]

for (var i = 0; i<5;i++){

for (var j = 0; j<5; j++){

for (var k = 0; k<32; k++){

for (var t = 0; t<28; t++){

A1[i][j][k] = (A[(i+2)%5][(j+1)%5][(k+(t\*3\*t)%64)]^1)^A[i][j][k]

}

}

}

}

**Продолжение листинга**

A = A1

A1 =[[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]],[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]],[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]],[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]],[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

**Продолжение листинга**

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]]]

for (var i = 0; i<5; i++){

for (var j = 0; j<5; j++){

for (var k = 0; k<64;k++){

A1[i][j][k]=A[i][j][k]^A[5-1-i][(j+1)%5][k]

}

}

}

A = A1

A1 =[[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]],[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]],[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]],[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]],[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,

**Продолжение листинга**

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]]]

for (var i = 0; i<5; i++){

for (var j = 0; j<5; j++){

for (var k = 0; k<64;k++){

A1[i][j][k]=(A[i][j][(k+1)%64]^1)^(A[i][(j+1)%5][k]^1)

}

}

}

A=A1

A1 =[[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]],[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]],[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]],[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

**Продолжение листинга**

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]],[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]]]

for (var i = 0; i<5; i++){

for (var j = 0; j<5; j++){

for (var k = 0; k<64;k++){

if ((i-2)%5<0){

A1[i][j][k]=A[5+(i-2)%5][j][k]|A[j][i][(k+32)%64]

} else {

A1[i][j][k]=A[(i-2)%5][j][k]|A[j][i][(k+32)%64]

}

}

}

}

A=A1

A1 =[[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]],[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,

**Продолжение листинга**

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]],[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]],[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]],[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]]]

for (var i = 0; i<5; i++){

for (var j = 0; j<5; j++){

for (var k = 0; k<64;k++){

if ((k-44)%64<0){

if((k-43)%64<0){

A1[i][j][k]=(A[(i+4)%5][j][k]^1)^(A[j][i][64+(k-44)%64]^A[j][i][64+(k-43)%64])

} else{

A1[i][j][k]=(A[(i+4)%5][j][k]^1)^(A[j][i][64+(k-44)%64]^A[j][i][(k-43)%64])

}

} else{

if((k-43)%64<0){

**Продолжение листинга**

A1[i][j][k]=(A[(i+4)%5][j][k]^1)^(A[j][i][(k-44)%64]^A[j][i][64+(k-43)%64])

} else{

A1[i][j][k]=(A[(i+4)%5][j][k]^1)^(A[j][i][(k-44)%64]^A[j][i][(k-43)%64])

}

}

}

}

}

A=A1

return A

}

function GetHash(M){

S =[[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]],[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]],[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]],[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

**Продолжение листинга**

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]],[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]]]

P = padding(M)

for (var p = 0; p < P.length; p++){

for (var i =0; i<5; i++){

for (var j = 0; j<5; j++){

for (var k = 0; k<64; k++){

if (i+j\*5<576/64){

S[i][j][k] = S[i][j][k] ^ P[p][i+j\*5][k]

}

}

}

}

for (var i = 0; i<24; i++){

S = peres(S)

}

}

Z = ""

while (true){

for (var i = 0; i<5; i++){

for (var j = 0; j<5; j++){

if (5\*i+j<576/64){

temp = ""

temp\_sum = 0n

for (var c = 0; c<64;c++){

temp+=String(S[j][i][c])

temp\_sum += BigInt(Math.pow(2, (63-c)))\*BigInt(S[j][i][c])

}

Z += temp\_sum.toString(16)

}

}

}

for (var i = 0; i<24;i++){

S = peres(S)

}

if (Z.length>128){

break

**Продолжение листинга**

}

}

return Z.slice(0,128)

}

let test\_st = process.argv[2]

h = GetHash(test\_st)

fs=require("fs")

fs.writeFileSync("Blog/test\_res\_js.txt", h, "utf-8")