МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
Вычислительной техники	Вычислительная техника

Направление подготовки 09.03.01 «Информатика и вычислительная техника»

Профиль Вычислительные машины, комплексы, системы и сети

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

на тему

Блок сжатия криптографической хеш-функции на базе полносвязной нейронной сети

Студент _____ (подпись, дата) Петкилев Александр Алексеевич

Руководитель _____ (подпись, дата) Дубравин А.В.

Нормоконтролёр _____ (подпись, дата) Кучин А.В. 18.06.19

Работа допущена к защите (протокол зас. кафедры от 31.05.2019 г. № 38-09-14)

Заведующий кафедрой _____ (подпись) Митрохин М.А.

Работа защищена с отметкой отлично (протокол заседания ГЭК от 19.06.19 № 2)

Секретарь ГЭК _____ (подпись) Гурин Е.И.

Пенза, 2019

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
Вычислительной техники	Вычислительная техника

«Утверждаю»
Заведующий кафедрой
_____М.А. Митрохин
«13» апреля 2019 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Петкилев Александр Алексеевич гр. 15ВВ1 факультета вычислительной техники.

2.Тема работы: «Разработка криптографического блока сжатия хеш-функции».

Тема утверждена приказом ПГУ № 554/о от "29" апреля 2019 г.

3. Руководитель работы к.т.н., доцент каф. ВТ ПГУ Дубравин А.В..

4. Задание на работу (назначение разработки, исходные данные и т.п.).

Разработать блок сжатия криптографической хеш-функции и отображение в виде графиков результатов статистических проверок. Разработать и отладить взаимодействие модулей приложения. Система должна быть представлена в виде динамической библиотеки классов, с целью обеспечения возможности его применения в качестве компонента для решения конкретных практических задач.

5. Перечень подлежащих разработке вопросов.

Требуется произвести анализ предметных областей: изучить типы искусственных нейронных сетей, виды и свойства хеш-функций, рассмотреть функции хаотического отображения. Необходимо изучить методологии проектирования нейронных сетей и хеш-функций, а также изучить виды атак на хеш-функции.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	7	29.04.2019 - 06.05.2019	
2.	Анализ требований	3	07.05.2019 - 10.05.2019	
3.	Выбор технологий, применяемых при проектировании и разработке	3	11.05.2019 - 14.05.2019	
4.	Разработка блока сжатия хеш-функции	10	15.05.2019 - 25.05.2019	
5.	Оформление пояснительной записки	5	26.05.2019 - 31.05.2019	

Дата выдачи задания "29" апреля 2019 г.

Руководитель бакалаврской работы _____ Дубравин А.В.
(подпись, дата)

Задание к исполнению принял студент 29.04.2019 Петкилев А.А.
(подпись, дата)

Работу к защите допустить
Декан факультета 18.6.19 Фионова Л.Р.
(подпись, дата)

РЕФЕРАТ

Пояснительная записка содержит 17 рисунков, 30 источников, 2 приложения.

ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ, ХЕШ-ФУНКЦИЯ, КРИПТОГРАФИЯ, ХАОТИЧЕСКОЕ ОТОБРАЖЕНИЕ, ARTIFICAL NEURON NETWORK, HASHFUNCTION, CRYPTOGRAPHY, CHAOTICMAP.

В рамках ВКР было исследовано применение искусственных нейронных сетей в задачах обеспечения информационной безопасности вычислительных систем. Разработан алгоритм хеширования входной последовательности на основе полносвязной нейронной сети с использованием хаотических отображений. Описана математическая модель блока сжатия криптографической хеш-функции.

Целью работы является разработка блока сжатия криптографической хеш-функции. Метод основан на использовании свойств нейронных сетей и хаотических отображений.

В процессе работы проводился анализ, сравнение и выбор структуры нейронной сети, наиболее подходящей для данного типа задачи.

В результате выполнения работы была создана библиотека для работы с предложенным алгоритмом хеширования. В дальнейшем планируется улучшить структуру алгоритма для повышения безопасности и оптимизации скорости его работы.

СОДЕРЖАНИЕ

РЕФЕРАТ 4
СОДЕРЖАНИЕ 5
ВВЕДЕНИЕ 7
1 ПОСТАНОВКА ЗАДАЧИ 9
2 ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ 10
2.1 Хеш-функция 10
2.1.1 Виды хеш-функций 10
2.1.2 Свойства хеш-функций 11
2.1.3 Криптографическая хеш-функция 11
2.1.4 Виды атак на хеш-функции 13
2.1.5 Базовые алгоритмы хеш-функций 15
2.1.6 Обзор современных алгоритмов 17
2.2 Нейронные сети 27
2.2.1 Искусственный нейрон 27
2.2.2 Активационная функция 27
2.2.3 Перцептрон 28
2.2.4 Сигмоидальный нейрон 29
2.2.5 Архитектуры нейронных сетей 30
2.2.6 Методы обучения 32
2.2.7 Функция хаотического отображения 35
3 ПРОЕКТИРОВАНИЕ БЛОКА СЖАТИЯ 39
3.1 Архитектура блока сжатия хеш-функции 39
3.2 Анализ свойств разработанного блока сжатия хеш-функции 41
4 РАЗРАБОТКА СИСТЕМЫ 44
4.1 Выбор инструментов реализации 44
4.1.1 Язык программирования R 44
4.1.2 Язык программирования Python 44
4.1.3 Язык программирования C# 45

4.2 Проектирование системы 45
4.3 Программная реализация системы 46
ЗАКЛЮЧЕНИЕ 49
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 50
ПРИЛОЖЕНИЕ А. Листинги программы 53

ВВЕДЕНИЕ

Одной из важнейших задач информационных систем является обеспечение безопасности самой системы, а также хранимой и обрабатываемой информации. Подсистема безопасности должна обеспечивать не только защиту системы от несанкционированного доступа, но и безопасность аутентификационных данных пользователей даже в случае взлома системы. Очевидно, что хранение базы паролей пользователей в открытом виде является небезопасным. Исходя из этого, используются специальные системы для их хранения на основе криптографических хеш-функций. Исходя из этого, актуальной становится задача поиска наиболее криптографически стойкого и надежного алгоритма хеширования.

Криптоанализ хеш-функий MD4 [1,2] и SHA-1, MD5 [3] приводит к тому, что возникает необходимость разработки новых более криптографически стойких и при этом эффективных алгоритмов хеширования, что в дальнейшем приведет к постепенному отказу от данных алгоритмов в информационных системах с повышенными требованиями к надежности.

Искусственные нейронные сети благодаря своей архитектуре в действительности представляют собой эффективный способ параллельной обработки данных. Данное свойство искусственных нейронных сетей позволяет в перспективе применять их для широкого круга задач.

Системы цифровых водяных знаков, построенные с использованием искусственных нейронных сетей, позволяют обеспечивать защиту авторского права [4]. Архитектура искусственных нейронных сетей позволяет эффективно выполнять работы по распознаванию образов и классификации множества объектов по какому-либо признаку. Кроме того, благодаря правильно спроектированным алгоритмам обучения нейронные сети могут достигать крайне высоких показателей точности. Приведенные выше преимущества позволяют эффективно использовать искусственные нейронные сети в системах биологической аутентификации. В частности, существует ряд работ, посвященных распознаванию подписи человека с использованием

искусственных нейронных сетей [5], распознаванию лица человека [6]. В России существует стандарт ГОСТ Р 52633-2006, определяющий требования к средствам высоконадёжной биометрической аутентификации. При биометрической аутентификации в соответствии с данным стандартом используется многослойная нейронная сеть [7].

Исходя из этого, в рамках данной работы предлагается разработка и анализ алгоритма хеширования, альтернативного традиционным алгоритмам, основанного на использовании свойств нейронных сетей и хаотических отображений.

1 ПОСТАНОВКА ЗАДАЧИ

В рамках работы необходимо разработать блок сжатия криптографической хеш-функции, основанного на искусственной нейронной сети, который должен преобразовывать данные входной последовательности произвольной длины в последовательность фиксированной длины. Алгоритм должен обладать следующими свойствами:

1. Однонаправленности, достигаемой высокой алгоритмической сложностью построения прообраза хеш-функции по её значению;
2. Высокой чувствительности к входным данным – зависимости от всех бит входной последовательности и от их взаимного расположения.

Используя возможности языка программирования C# необходимо разработать искусственную нейронную сеть, функцию хаотического отображения, функции проверки статистических характеристик алгоритма, а также реализовать отображение в виде графиков результатов статистических проверок: побитных различий между выходными последовательностями и расстояния Хемминга между ними.

Реализация алгоритма блока сжатия хеш-функции должна быть представлена в виде динамической библиотеки классов, с целью обеспечения возможности его применения в качестве компонента для решения конкретных практических задач.

2 ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

2.1 Хеш-функция

Хеш-функция h(x) – это функция, принимающая на входе в качестве аргумента информационную последовательность (строку) M произвольной длины и дающая на выходе в качестве результата информационную последовательность (строку) фиксированной длины. Результат хеширования информационной последовательности M называется хеш-образом h(M). Соотношение между длинами M и h(M) может быть произвольным, иначе говоря, возможны любые соотношения

$$|M| > |h(M)|, |M| < |h(M)|, |M| = |h(M)|\ , (2.1)$$

где $|M|$ – длина информационной последовательности M.

2.1.1 Виды хеш-функций

По методам применения хеш-функции делятся за следующие виды:

1. Контрольные суммы - алгоритмы, используемые для защиты данных от непреднамеренных искажений, в том числе — от ошибок аппаратуры. Преимуществом контрольных сумм является быстрая скорость исполнения и простота реализации. К недостаткам стоит отнести отсутствие стойкости к атакам и коллизиям.

2. Геометрическое хеширование – алгоритмы, необходимые для решения задач в двумерном и трехмерном пространствах. Хеш-функция в данном методе обычно получает на вход какое-либо метрическое пространство и разделяет его, создавая сетку из клеток. Геометрическое хеширование применяется в телекоммуникациях при работе с многомерными сигналами[8].

3. Хеш-таблица - структура данных, реализующая ассоциативный массив, позволяющая хранить ключ и значение, выполнять следующие операции: добавление нового ключа и значения, поиска и удаления пары по ключу. Хеш-таблицы применяются с целью ускорения поиска в базах данных и языковых процессорах для обслуживания таблиц идентификаторов.

4. Криптографическая хеш-функция – особый класс хеш-функций, алгоритмы которого удовлетворяют дополнительным требованиям криптостойкости что позволяет использовать функции этого класса в задачах криптографии.

2.1.2 Свойства хеш-функций

Любая хеш-функция должна удовлетворять следующим условиям:

1. должна уметь выполнять преобразования данных произвольной длины в фиксированную;
2. должна иметь открытый алгоритм для возможности исследования её криптостойкости;
3. должна быть односторонней, то есть не должно быть математической возможности по результату определить исходные данные;
4. должна быть стойкой к коллизиям, то есть не должна выдавать одинаковых значений при разных входных данных;
5. не должна требовать больших вычислительных ресурсов;
6. должна обладать лавинным эффектом, то есть при малейшем изменении входных данных результат должен существенно изменяться.

2.1.3 Криптографическая хеш-функция

Любая криптографическая хеш-функция h(x) должна удовлетворять следующим требованиям [9-12]:

1. Хеш-образ должен зависеть от всех бит прообраза и от их взаимного расположения;
2. При любом изменении входной информации, хеш-образ должен изменяться непредсказуемо, иначе говоря, в среднем должна измениться половина бит хеш-образа (каждый бит может измениться с вероятностью 1/2);
3. При заданном значении прообраза задача нахождения хеш-образа должна быть вычислительно разрешима, иначе говоря, по заданному значению M легко вычисляется значение h(M);

4. При заданном значении хеш-образа задача нахождения прообраза должна быть вычислительно неразрешимой, иначе говоря, по заданному значению h(M) трудно вычислить значение M (рис. 2.1, а) (Pre-Image Resistance); формально говоря, хеш-функция h является односторонней (One-Way), если для произвольной n-разрядной строки $y \in \{0,1\}^n$ вычислительно сложно найти $x \in \{0,1\}^*$, такое, что h(x) = y;

5. При заданных значениях хеш-образа h(M) и первого прообраза M задача нахождения второго прообраза $M' \neq M$, такого, что h(M') = h(M), должна быть вычислительно неразрешимой (рис. 2.1,в) (Second Pre-Image Resistance); формально говоря, хеш-функция h является стойкой в смысле нахождения второго прообраза, если для заданной m-разрядной строки $x \in \{0,1\}^m$ вычислительно сложно найти произвольную строку $x' \in \{0,1\}^*$, $x' \neq x$, такую, что $h(x') = h(x)$;

6. Задача нахождения коллизии хеш-функции, т.е. нахождение двух произвольных сообщений M1 и M2, таких, что $M1 \neq M2$, а h(M1) = h(M2), должна быть вычислительно неразрешимой (рис. 2.1,б) (Collision Resistance); формально говоря, хеш-функция h является стойкой в смысле нахождения коллизий, если вычислительно сложно найти две произвольные строки $x', x'' \in \{0,1\}^*, x' \neq x''$, такие, что $h(x') \neq h(x'')$.

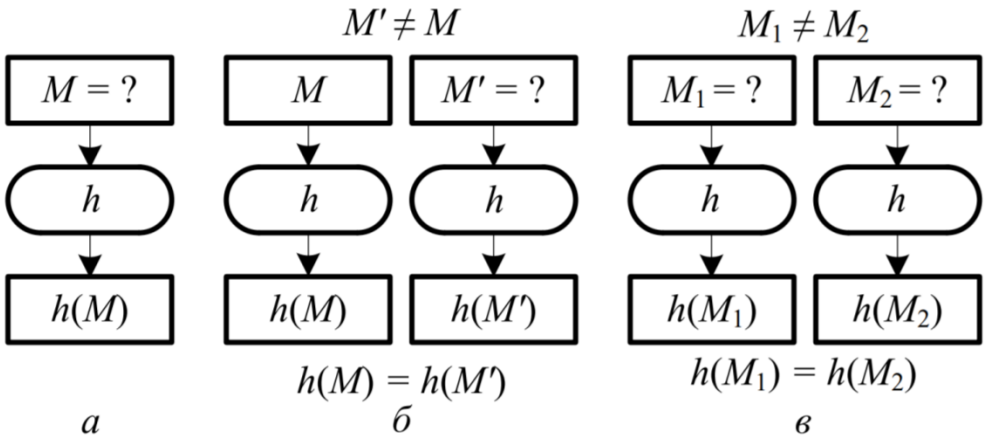


Рисунок 2.1 – Задачи: а – нахождение прообраза; б – нахождение второго прообраза; в – нахождение коллизии.

2.1.4 Виды атак на хеш-функции

Атака дней рождения

Идея этой атаки основана на том, что для группы из 23 человек вероятность того, что у двоих из них день рождения будет в один день, превышает $\frac{1}{2}$. Т.к. число 23 намного меньше числа, которое предполагает большинство людей, это называется «парадокс дней рождения». Другой пример: если есть две группы по 17 человек в каждой, то вероятность того, что у двух людей в разных группах совпадут дни рождения, также превышает $\frac{1}{2}$. Здесь предполагается, что дни рождений случайно и равномерно распределены по всем дням в году, иначе вероятность совпадений будет выше. Это можно обобщить следующим образом: если взять два объекта размерами r_1 и r_2 из набора с n объектами и если $r_1 r_2 = n$ при $r_1, r_2 = O(\sqrt{n})$, то вероятность совпадения равна $1 - \frac{1}{e} = 63\%$.

Первая атака, основанная на этом свойстве, была предложена Г. Ювалем в 1979 году. Он показал, что проще найти коллизию для односторонней хеш-функции, чем пытаться найти прообраз для элемента из заданного диапазона. Коллизия может быть найдена следующим образом:

1. Противник генерирует r_1 поддельных сообщений и r_2 подлинных. Если $r_1 = r_2 = \sqrt{n}$, то вероятность нахождения коллизии равна 63%.
2. Поиск совпадений не требует r^2 операций: после сортировки данных, которая потребует $\mathrm{O(r \log r)}$ операций, сравнение будет простым.

Алгоритмическое улучшение для подобного поиска коллизий было предложено Дж. Дж. Квисквотером в 1990 году: оно аннулирует требования к дисковому пространству атакующего, если у него есть возможность применять требуемую хеш-функцию. Идея состоит в том, что происходит итерация по случайному отображению и выходные данные используются в качестве входных. Рано или поздно такая итерация после λ шагов будет повторяться в

цикле с периодом μ. В момент, когда итерация начинает повторяться, будет найдено два значения x' и x'' такие, что $f(x') = f(x'')$. Ожидаемое количество вычислений хеш-функции равно $\rho = \lambda + \mu = \sqrt{\pi n/8} + \sqrt{\pi n/8} = \sqrt{\pi n/2}$.

Атака «встреча посередине»

Эта атака является разновидностью атаки дней рождения, но вместо хеша в ней сравниваются промежуточные переменные. В отличие от атаки дней рождения, такая атака позволяет создать сообщение с заранее известным хешом. Атакующий создает r_1 вариаций первой части сообщения и r_2 вариаций последней. Начиная с начального значения *IV* и двигаясь «назад» от хеша, вероятность совпадения значения промежуточной переменной такая же, как и в атаке дней рождения. Единственное ограничение состоит в том, что точка «встречи» не должна быть на первом или последнем блоке сообщения. Алгоритм Квисквотера может быть легко изменен для подобного класса атаки, при этом требования к дисковому пространству атакующего также останется несущественным.

Атака корректирования блока

Эта атака заключается в замене всех блоков сообщения, кроме некоторого X_j. Затем этот блок генерируется таким образом, чтобы хеш принял нужное значение. Хеш-функции, основанные на модульной арифметике, особенно уязвимы этому типу атак.

Атака корректирования блока может быть также применена для получения коллизии. Атакующий начинает с двух произвольных сообщений X' и X'' и добавляет к ним один или более корректирующих блоков Y' и Y'' таких, что сообщения $X' \,||\, Y'$ и $X'' \,||\, Y''$ имеют одинаковый хеш.

Атака фиксированной точки

Идея этой атаки заключается в поиске X_i и H_{i-1} таких, что $f(X_i, H_{i-1}) = H_{i-1}$. Если промежуточная переменная равна H_{i-1}, то возможно вставить в

сообщение произвольное число произвольных блоков без изменения результирующего хеша. Нахождение коллизии или второго прообраза в результате этой атаке возможно, если промежуточную переменную удастся сделать равной H_{i-1}. Однако, предотвратить эту атаку довольно просто: достаточно добавлять к сообщению его длину.

Дифференциальные атаки

Дифференциальные атаки основаны на отношении разностей входных и выходных данных и применим как к блочным алгоритмам шифрования, так и к хеш-функциям. При выполнении этих атак находится зависимость между различиями входных и выходных данных. Эти атаки применимы только к алгоритмам шифрования, которые удовлетворяют некоторым условиям, т.н. марковским шифрам. Им подвержены, например, DES, FEAL, LOKI, IDEA и PES. Для таких шифров в результате применения этой атаке будет найден ключ на основании большого количества открытых текстов и соответствующих им шифротекстов. Проще всего провести эту атаку на MDC, где вся информация доступна публично. К таким атакам относятся обнаруженные относительно недавно т.н. туннели, появляющиеся, если слова состояний подходят под определенный шаблон.

2.1.5 Базовые алгоритмы хеш-функций

Итеративная последовательная схема

В общем случае в основе построения хеш-функции лежит итеративная последовательная схема. Ядром алгоритма является сжимающая функция — преобразование k входных в n выходных бит, где n — разрядность хеш-функции, а k — произвольное число, большее n. При этом сжимающая функция должна удовлетворять всем условиям криптостойкости.

Входной поток разбивается на блоки по (k − n) бит. Алгоритм использует временную переменную размером в n бит, в качестве начального значения которой берется некое общеизвестное число. Каждый следующий блок данных

объединяется с выходным значением сжимающей функции на предыдущей итерации. Значением хеш-функции являются выходные n бит последней итерации. Каждый бит выходного значения хеш-функции зависит от всего входного потока данных и начального значения. Таким образом достигается лавинный эффект.

При проектировании хеш-функций на основе итеративной схемы возникает проблема с размером входного потока данных. Размер входного потока данных должен быть кратен (k − n). Как правило, перед началом алгоритма данные расширяются неким, заранее известным, способом.

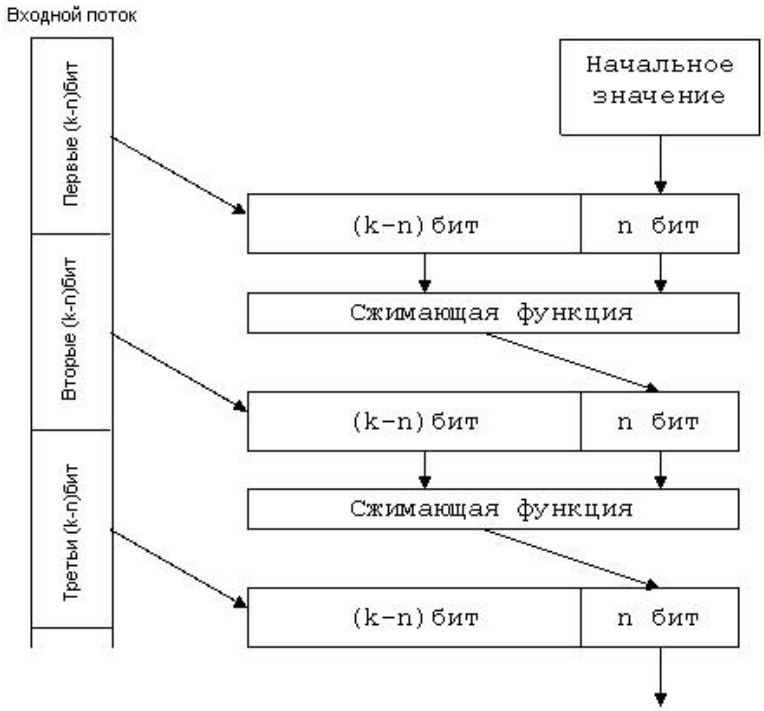


Рисунок 2.2 – Итеративная последовательная схема.

Сжимающая функция на основе симметричного блочного алгоритма

В качестве сжимающей функции можно использовать симметричный блочный алгоритм шифрования. Для обеспечения большей безопасности можно использовать в качестве ключа блок данных, предназначенный к хешированию на данной итерации, а результат предыдущей сжимающей функции — в качестве входа. Тогда результатом последней итерации будет выход алгоритма.

В таком случае безопасность хеш-функции базируется на безопасности используемого алгоритма.

Обычно при построении хеш-функции используют более сложную систему. Обобщённая схема симметричного блочного алгоритма шифрования изображена на рис. 2.3.

Рисунок 2.3 – Обобщенная схема формирования хеш-функции.

Основным недостатком хеш-функций, спроектированных на основе блочных алгоритмов, является низкая скорость работы. Необходимую криптостойкость можно обеспечить и за меньшее количество операций над входными данными. Существуют более быстрые алгоритмы хеширования, спроектированные исходя из требований криптостойкости. Наиболее распространенные из них — MD5, SHA-1, SHA-2.

2.1.6 Обзор современных алгоритмов

2.1.6.1 Конкурс SHA -3

Алгоритм SHA является частью стандарта SHS (Secure Hash Standard), разработанного в 1993 г. Национальным институтом стандартов и технологий США (FIPS 180) и Агенством национальной безопасности США. SHA использует принципы, предложенные ранее Р. Ривестом при разработке своих алгоритмов MD4 и MD5. В 1995 г. стандарт был пересмотрен (FIPS 180-1) в пользу версии SHA-1.Позднее стандарт был пересмотрен вновь и были определены 4 новые версии: SHA-224, SHA-256, SHA-384, SHA-512. Все версии имеют одинаковую структуру, поэтому часто их называют общим именем SHA-2.

Многие популярные хеш-функции используют так называемую ARX конструкцию, предполагающую использование алгебраически несовместимых операций: сложения по модулю 2^n, где n = 32 или n = 64 (Add); сдвига (Rotate) и поразрядного сложения по модулю два (XOR). ARX очень сложна для анализа и по этой причине эффективные атаки на нее появляются спустя годы.

Начиная с 2005 года, положение в области криптографических хеш-функций стало напоминать кризисное, и по этой причине в 2007 году НИСТ объявил открытый международный конкурс по выбору нового стандарта хеширования SHA-3. Конкурс был ответом НИСТ на последние достижения в области криптоанализа алгоритмов хеширования.

Конкурс SHA-3 вызвал появление новых подходов к построению хеш-функций:

1. конструкции HAIFA, суть которой заключается в двух нововведениях, а именно, в добавлении счетчика итераций и в полной интеграции соли (salt) в структуру хеш-функции [13-15];
2. конструкции Wide-pipe, идея которой заключается в увеличении (по сравнению с традиционной конструкцией Narrow-pipe) размера внутреннего состояния [16];
3. конструкции Sponge, основанной на использовании качественного генератора псевдослучайных чисел [17-19].

Основываясь на результатах криптоанализа, особенностей аппаратной и программной реализации, в октябре 2012 года НИСТ объявил алгоритм Keccak (Sponge) победителем, завершив конкурс.

2.1.6.2 Конструкция Sponge.

В качестве основы для нового стандарта SHA-3 была выбрана хеш-функция Keccak, построенная с использованием конструкции под названием Sponge (губка), которая показана на рис. 2.4.

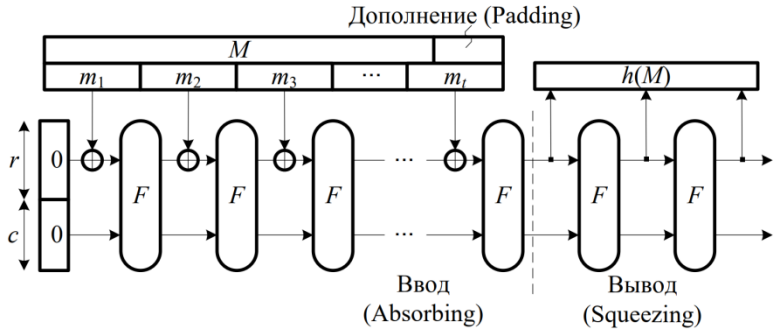


Рисунок 2.4 – Схема Sponge.

Схема Sponge имеет внутреннее состояние S разрядностью b, b = r + c, значение r называется rate (определяет скорость преобразования), а значение c (параметр безопасности) – capacity. Иначе говоря, состояние делится на две части: S_1 разрядностью r и S_2 разрядностью c. Пусть RL (Resistance Level) – уровень стойкости, тогда по утверждению авторов конструкции справедливо соотношение c 2RL (иначе говоря, Sponge гарантирует нижнюю границу сложности $2^{c/2}$для любой атаки на хеш-функцию).

Процесс хеширования состоит из двух многораундовых этапов: ввод (Absorbing, впитывание) и вывод (Squeezing, выжимание).

Этап ввода информации:

1. Состояние S инициализируется нулями. Сообщение M дополняется до длины кратной r, а затем разбивается на блоки длины r;
2. Первый блок сообщения M суммируется по модулю 2 (операция XOR) с S_1;

результат операции XOR и S_2 передаются на вход раундовой функции F;

3. Второй блок сообщения M складывается по модулю 2 с первыми r битами выхода функции F; результат операции XOR и последние c бит результата действия функции f вновь передаются на вход функции F второго раунда;
4. Этап продолжается до тех пор, пока не будут обработаны все блоки сообщения M.

Особенностью этапа является в том, что на каждом раунде блок сообщения складывается по модулю 2 только с частью состояния, а функция F производит преобразование всего состояния и делает его зависимым от всего сообщения M.

Результат длины b, полученный на этапе впитывания, вновь подаётся на вход функции F . При этом с выхода этой функции считываются первые r бит. Этот процесс повторяется конечное число раз, при этом над r битами, считанными на каждом этапе, производится операция конкатенации (сцепления блоков). Повторение происходит до тех пор, пока не получим результат нужной длины.

2.1.6.3 Алгоритм Keccak

Хеш-функция Keccak[20] определяет целое семейство функций KeccakF[b] с разрядностью состояния

$$b \in \{25, 50, 100, 200, 400, 800, 1600\}$$

Однако, хотя пользователь в принципе может выбирать для своей реализации любую из предложенных авторами функций, следует иметь в виду, что в качестве стандарта SHA-3 принята только функция Keccak-1600. В этом случае авторы рекомендуют использовать следующие параметры:

SHA-224: r = 1152, c = 448 (вернуть первые 28 байт результата);

SHA-256: r = 1088, c = 512 (вернуть первые 32 байт результата);

SHA-384: r = 832, c = 768 (вернуть первые 48 байт результата);

SHA-512: r = 576, c = 1024 (вернуть первые 64 байт результата).

Последний вариант будет использован в стандарте.

Внутреннее состояние. Алгоритм работает со словами длиной $w = 2^l$, $0 < l \leq 6$. Разрядность состояния равна $b = 5 \times 5 \times 2^l$ бит. Состояние (state) рассматривается как трехмерный массив S [x] [y] [z], в составе которого выделяются столбцы (columns), строки (rows), линии (lanes) длиной 2^l бит и срезы (slices), как показано на рис. 2.5. Срез представляет собой квадратный массив 5 × 5 бит.

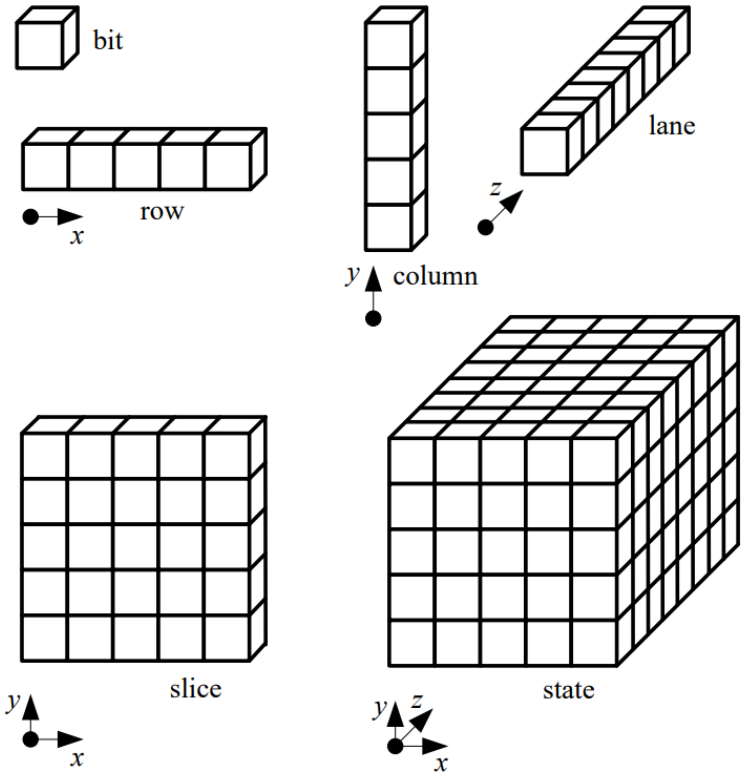


Рисунок 2.5 – Состояние Keccak (на примере b = 200).

Длина слова w определяет соответствующую функцию Keccak-F[b]. В SHA-3 используется функция Keccak-F[1600].

Инициализация и дополнение.

Вектор инициализации IV для конструкции Sponge:

$S[x][y][z] \leftarrow 0\ forall\ 0 \leq x, y \leq 4, 0 \leq z \leq 2^l - 1$ (2.2)

Дополнение последнего блока:

1. Добавляется единичный бит;
2. Добавляется минимальное число нулевых бит для достижения промежуточной длины последнего блока (r - 1);
3. Добавляется финальный единичный бит.

Раундовая функция.

Число раундов равно 12 + 2l. В SHA-3 для 64-разрядных процессоров l = 6, поэтому число раундов равно 24. Каждый раунд суть композиция 5 раундовых преобразований

$$S \leftarrow \tau \circ \chi \circ \pi \circ \rho \circ \theta(S) \text{ (2.3)}$$

Каждое преобразование оперирует либо только строками, либо только столбцами, либо только линиями, либо является перестановкой на плоскости (x, y).

Формальное описание theta-функции θ.

Суть преобразования – замена каждого бита суммой по модулю 2 данного бита и битов четности (parity_1 и parity_2) соседних столбцов:

$$\text{parity}_1 \leftarrow \sum_{\mu=0}^{4} S[x-1][\mu][z] mod 2 \text{ (2.4)}$$

$$\text{parity}_2 \leftarrow \sum_{\mu=0}^{4} S[x+1][\mu][z] mod 2 \text{ (2.5)}$$

$$S[x][y][z] \leftarrow S[x][y][z] + \text{parity}_1 + \text{parity}_2 \text{ (2.6)}$$

Строки и столбцы индексируются в GF(5), линии – по модулю 2^l, т.е. равному длине слова.

Формальное описание rho-функции ρ.

Суть преобразования – сдвиг бит линии (lane), при этом величина сдвига равна $\frac{(t+1)(t+2)}{2} mod\ 2^l$ для $-1 \leq t \leq 23$:

1. Вычислить (все операции осуществляются в GF(5)) для $0 \leq t \leq 23$ координаты строки и столбца

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} 1 \\ 0 \end{pmatrix}; \text{ (2.7)}$$

2. Дополнительно установить t = –1 для x = y = 0;

$$\rho: S[x][y][z] \leftarrow S[x][y][z - \frac{(t+1)(t+2)}{2}] \text{ (2.8)}$$

Суть преобразования – перестановка линий, иначе говоря, перестановка в плоскости (x, y):

$$\begin{pmatrix} x \\ y \end{pmatrix} \leftarrow \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} y \\ 2x + 3y \end{pmatrix} \quad (2.9)$$

Все вычисления осуществляются по модулю 5. Следует иметь в виду, что на рис. 2.6 точка (0, 0) находится в центре.

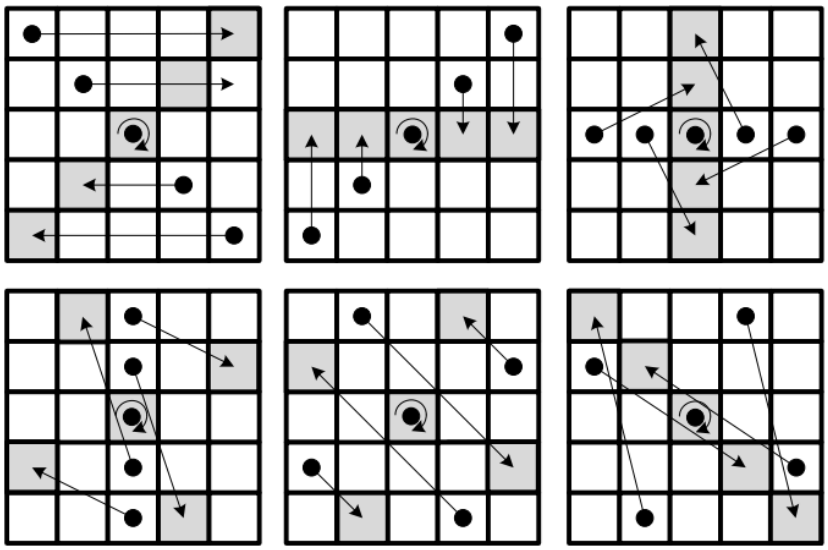


Рисунок 2.6 – Преобразование π.

Таким образом, получаем:

$$\pi: S[x][y][z] \leftarrow S[y][2x + 3y][z] \quad (2.10)$$

Формальное описание chi-функции χ (рис. 2.7).

Суть преобразования – замена бита путем комбинирования его с двумя последующими битами строки, индексы x и y при этом не меняются:

$$\chi: S[x][y][z] \leftarrow S[x][y][z] + (\neg S[x + 1][y][z] \text{^} S[x + 2][y][z]) \quad (2.11)$$

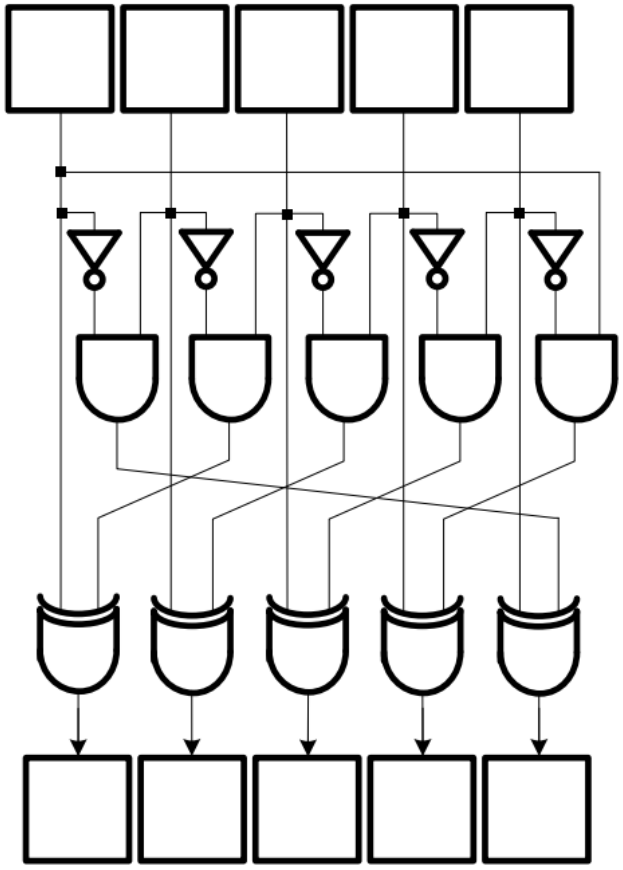


Рисунок 2.7 – Преобразование χ.

2.1.6.4 Алгоритм «Стрибог»

В конце 2012 года как ответ на принятие стандарта SHA-3 был принят новый российский стандарт хеширования ГОСТ 34.11 – 2012 [21]. Новый алгоритм хеширования получил название Стрибог. Стандарт определяет две функции хеширования – «Стрибог-256» и «Стрибог-512» с длинами хеш-образа 256 и 512 бит соответственно. Эти функции отличаются начальным внутренним состоянием и его частью, принимаемой за результат вычислений.

Хеш-функция построена с использованием архитектуры Квадрат, так как внутреннее состояние функции сжатия можно представить в виде квадратного массива байтов 8 × 8. Хеш-функция построена на основе блочного шифра, в котором произведены некоторые принципиальные изменения по сравнению с AES.

В функции сжатия используются только преобразование LPSX. Преобразование LPSX суть композиция четырех преобразований S, P, L и X,

описанных ниже. Вместе со сложением по модулю 2^{512} они составляют полный набор операций, использующихся в стандарте.

Используемые преобразования

X-преобразование. Суть преобразования – поразрядное сложение по модулю два (XOR) двух последовательностей длиной 512 бит каждая.

S-преобразование. Суть преобразования – замена байтов, выполняемая независимо с каждым байтом состояния в соответствии с фиксированной таблицей подстановок размером 8 × 256, каждый байт входной последовательности заменяется соответствующим байтом из таблицы подстановок.

P-преобразование. Суть преобразования – транспонирование матрицы состояний, иначе говоря, перестановка – для каждой пары байт из входной последовательности выполняется замена одного байта другим.

L-преобразование. Линейное преобразование векторов (строк матрицы состояния) задается умножением справа на матрицу A над GF(2) размером 64 × 64.

Функция сжатия

На каждой итерации используется функция сжатия

$$G_N: \{0,1\}^{512} \times \{0,1\}^{512} \to \{0,1\}^{512}, N \in \{0,1\}^{512}, \quad (2.12)$$

значение которой вычисляется по формуле

$$G_N(h,m) = E(LPS(h \oplus \mathrm{N}), \mathrm{m}) \oplus \mathrm{h} \oplus \mathrm{m}, \quad (2.13)$$

где

$$E(k,m) = X[K_{13}] \circ LPSX[K_{12}] \circ \dots \circ LPSX[K_2] \circ LPSX[K_1](m). \quad (2.14)$$

Значения $K_i \in \{0,1\}^{512}, i = 1, \dots, 13$ вычисляются следующим образом $K_i = LPS(K_{i-1} \oplus C_{i-1}), i = 2, \dots, 13$.

Схема хеширования показана на рис. 2.8, схема функции сжатия – на рис. 2.9.

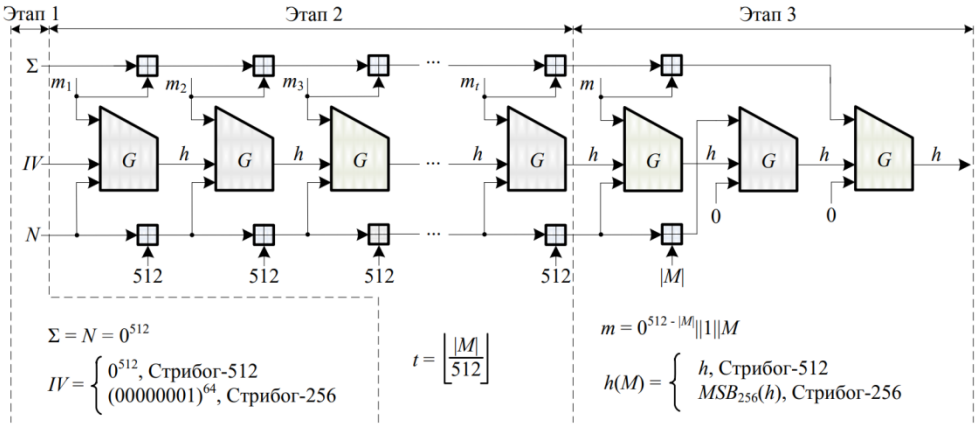


Рисунок 2.8 – Стрибог: процесс вычисления хеш-функции.

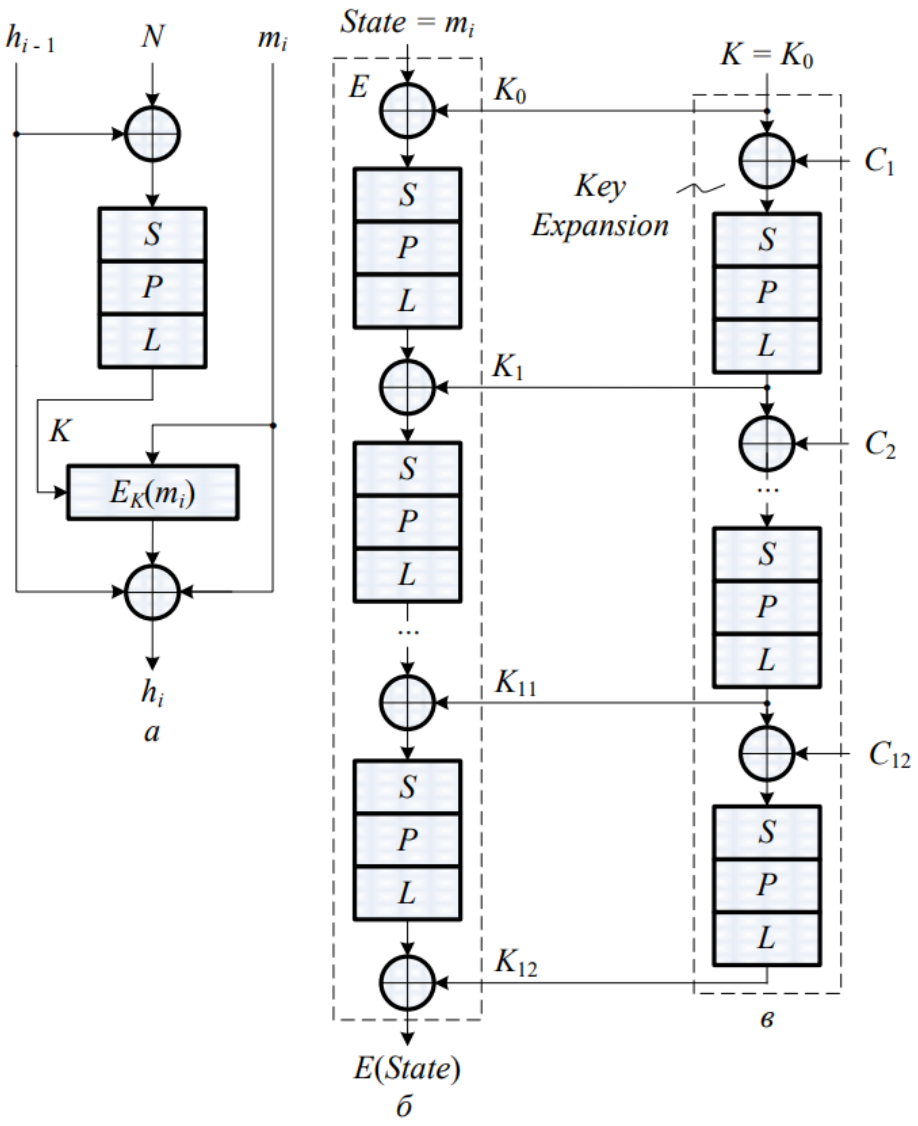


Рисунок 2.9 – Стрибог: а – схема функции сжатия; б и в – схема блочного шифра, лежащего в основе алгоритма (соответственно структура функции шифрования E и последовательность «разворачивания» ключа).

2.2 Нейронные сети

Искусственные нейронные сети – технология, которая объединяет в себе обширное множество дисциплин, таких как нейрофизиология, статистика, математика, физика, а также различные области информационных технологий. Применение искусственных нейронных сетей возможно в следующих направлениях: распознавание образов, анализ временных рядов, моделирование, обработка сигналов и др. [22].

2.2.1 Искусственный нейрон

Искусственные нейронные сети были построены по принципу биологических нейронных сетей, которые представляют собой сети нервных клеток, выполняющие определенные физиологические функции. Составным элементом нейронных сетей являются нейроны.

У нейрона есть несколько функций:

1. Приёмная функция: синапсы получают информацию;
2. Интегративная функция: на выходе нейрона сигнал, который несёт информацию обо всех суммированых в нейроне сигналах;
3. Проводниковая функция: по аксону проходит информация к синапсу;
4. Передающая функция: импульс, достигший окончания аксона, заставляет медиатор передавать возбуждение следующему нейрону.

В искусственной нейронной сети искусственный нейрон — это некоторая нелинейная функция, аргументом которой является линейная комбинация всех входных сигналов. Такая функция называется активационной. Затем результат активационной функции посылается на выход нейрона. Объединяя такие нейроны с другими, получают искусственную нейронную сеть.

2.2.2 Активационная функция

Функция активации нейрона характеризует зависимость сигнала на выходе нейрона от суммы сигналов на его входах. Обычно функция является

монотонно возрастающей и находится в области значений [-1,1] (гиперболический тангенс) и [0,1] (сигмоида). Для некоторых алгоритмов обучения необходимо, чтобы активационная функция была непрерывно дифференцируемой на всей числовой оси. Искусственный нейрон характеризуется своей активационной функцией (например, название "сигмоидальный нейрон"). Основными активационными функциями являются:

1. Пороговая активационная функция (функция Хевисайда).

$$f(x) = \begin{cases} 1 \text{ if } x \geq -w_0 x_0 \\ 0 \text{ else} \end{cases} \quad (2.15)$$

2. Сигмоидальная активационная функция;

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad (2.16)$$

3. Гиперболический тангенс.

$$\tanh(Ax) = \frac{e^{Ax}-e^{-Ax}}{e^{Ax}+e^{-Ax}} \quad (2.17)$$

2.2.3 Перцептрон

Перцептрон — тип искусственного нейрона, разрабатываемый Фрэнком Розенблаттом в 1950-ых и 1960-ых годах. В современных работах чаще всего используют другую модель искуственного нейрона — сигмоидальный нейрон. Чтобы понять, как работает сигмоидальный нейрон, необходимо рассмотреть структуру и принцип работы перцептрона. Перцептрон принимает на вход значения x_1, x_2... и выдаёт бинарный результат (см. рис. 2.10).

Розенблатт предложил использовать веса — числа, выражающие важность вклада каждого входа в конечный результат. Взвешенная сумма (или веса) сравнивается с пороговым значением (threshold), и по результатам определяется, будет ли выдан 0 или 1. Пороговое значение также является параметром нейрона.

$$\begin{cases} 0 \; if \; \sum_j w_j x_j \leq threshold \\ 1 \; if \; \sum_j w_j x_j > threshold \end{cases} \quad (2.18)$$

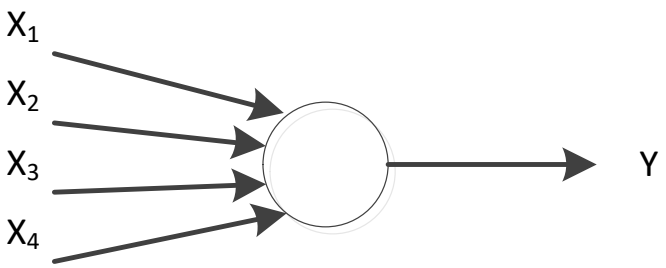


Рисунок 2.10 — Схема перцептрона

Перцептроны могут быть классифицированы как искусственные нейронные сети:

1. с одним скрытым слоем;
2. с пороговой активационной функцией;
3. с прямым распространением сигнала.

2.2.4 Сигмоидальный нейрон

Сигмоидальные нейроны похожи на перцептроны, однако небольшие изменения в их весах и смещениях незначительно изменяют выход нейрона. Этот факт позволяет сети из сигмоидальных нейронов обучаться. На вход сигмоидального нейрона подаются любые значения между 0 и 1. На выходе также выдаётся значение между 0 и 1, так как в качестве активационной функции используется сигмоида, являющаяся нелинейной.

$$f(x) = \frac{1}{1+e^{-\beta x}} \qquad (2.19)$$

Чем больше β (параметр наклона сигмоидальной функции активации), тем сильнее крутизна графика. При $\beta \to \infty$ сигмоида стремится к функции Хевисайда.

Нейроны можно разделить на группы в зависимости от их положения в сети:

1. входные нейроны принимают исходный вектор данных;
2. в промежуточных нейронах происходят основные вычислительные операции — обучение;

3. выходные нейроны — результат работы сети.

2.2.5 Архитектуры нейронных сетей

Полносвязная нейронная сеть прямого распространения

Нейронные сети определяют нелинейную гипотезу $h_{W,b}(x)$ с параметрами W и b. Нейронная сеть составлена из множества простых нейронов так, что выход одного из нейронов будет входом другого (см. рис. 2.11).

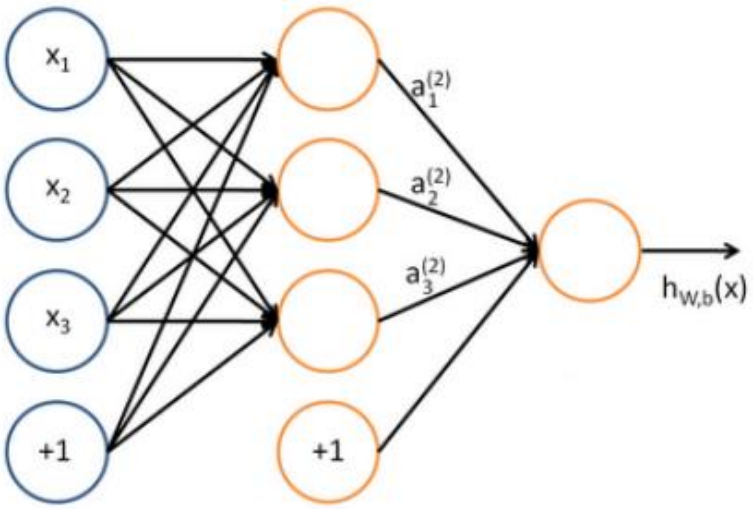


Рисунок 2.11 — Схема простейшей сети прямого распространения

Крайний левый слой называется входным, а крайний правый слой — выходным. Слой посередине является скрытым и называется так из-за того что его значения не наблюдаются в тренировочных примерах. Таким образом в данной сети три элемента входа, три скрытых элемента и один выходной элемент. Пусть n_l — количество слоёв в сети (в данном случае 3). Параметры сети (W, b) = ($W^{(1)}$, $b^{(1)}$, $W^{(2)}$, $b^{(2)}$). Результат применения функции активации (выхода) обозначается a_i для i-ого элемента. Получаем такую систему:

$$\begin{cases} a_1^{(2)} = f\left(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_{(1)}^1\right), \\ a_2^{(2)} = f\left(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_{(1)}^2\right), \\ a_3^{(2)} = f\left(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_{(1)}^3\right), \\ h = f(W_{11}^{(2)}a_1 + W_{12}^{(2)}a_2 + W_{13}^{(2)}x_3 + b_{(2)}^1) \end{cases} \quad (2.20)$$

Обозначив функцию суммирования через z, получим в векторной форме:

$$\begin{cases} z^2 = W^{(1)}x + b^{(1)} \\ a^{(2)} = f(z^{(2)}) \\ z^{(3)} = W^{(2)}a^{(2)} + b^{(2)} \\ h = f(z^{(3)}) \end{cases} \quad (2.21)$$

Общая формула будет выглядеть следующим образом:

$$\begin{cases} z^{(l+1)} = W^{(l)}a^{(l)} + b^{(l)} \\ a^{(l+1)} = f(z^{(l+1)}) \end{cases} \quad (2.22)$$

Сетью прямого распространения называются нейронные сети, которые используют выход одного слоя в качестве входных данных для следующего слоя.

Сверточные нейронные сети

Сверточная нейронная сеть была создана для эффективного распознавания изображений. Ключевой особенностью данной сети является чередование сверточного и объединяющего слоя. Фрагменты анализируемоего изображения умножаются на матрицу свертки, попиксельно, а результаты складываясь записываются в поля выходного изображения.

Структура сверточной сети схожа с структурой персептрона – обратных связей в ней нет, так что она тоже считается однонаправленной. Но если персептрон считается полносвязной нейронной сетью, где каждый элемент сети связывается с всеми элементами предыдущего слоя, то в сверточной сети, благодаря операции свертки, используется матрица весов небольшого размера, которая при движении по анализируемому изображению активирует только те нейроны следующего слоя, которые находятся в той же позиции. На выходе мы получаем матрицу весов, в которую записывается даже такая информация как наличие наклонной линии под определенным углом и ее координаты. Учитывая, что таких наборов много, сеть кодирует всевозможные линии и дуги под разными углами. Объединяющий слой в свою очередь уменьшает размерность сформированных карт признаков. После прохождения данных

слоев карта признаков, чаще всего преобразуется в вектор. Далее, для работы с ним на выходе дополнительно устанавливают несколько слоев полносвязного персептрона, который заканчивает анализ изображения.

Рекуррентные нейронные сети

В сетях прямого распространения используется единственный вход, который полностью определяет активации всех нейронов в оставшихся слоях. Такую сеть невозможно обучить предсказывать события, например, в сюжете фильма — неясно, как бы могла быть использована информация о предыдущих событиях в фильме. Рекуррентные нейронные сети призваны решить эту проблему. Имея внутри циклы, RNN позволяет информации сохраняться: поведение скрытых нейронов будет определяться не только активацией в других скрытых слоях, но и полученными ранее активациями самих нейронов.

RNN может быть представлена в качестве множества копий одной и той же нейронной сети, где каждая копия передает сообщение следующей копии. То есть, имея цепеобразную структуру, как последовательсти или списки, RNN является естественной архитектурой нейронной сети, используемой для таких данных.

2.2.6 Методы обучения

Алгоритм обратного распространения ошибки определяет стратегию подбора весов многослойной сети с применением градиентных методов оптимизации. Поскольку целевая функция, обычно определяемая как квадратичная разность суммы между фактическими и ожидаемыми выходными значениями, является непрерывной, градиентные методы оптимизации являются эффективными при обучении сети. При обучении многослойной нейронной сети необходимо вычислить вектор градиента относительно параметров всех слоёв сети. Пусть имеется конечный набор тренировочных данных (m примеров). Для обучения нейронной сети применяют пакетный градиентный спуск (batch gradient descent). Квадратичная ошибка целевой

функции (squared-error cost function) для одного примера будет вычислена по формуле:

$$J(W,b;x,y) = \frac{1}{2}||h_{W,b}(x) - y||^2 \text{ (2.23)}$$

Тогда целевая функция для m примеров будет выглядеть так:

$$J(W,b) = [\frac{1}{m}\sum_{i=1}^{m} J\left(W,b;x^{(i)},y^{(i)}\right)] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}(W_{ji}^{(l)})^2 \text{ (2.24)}$$

Первый член выражения J(W,b) это сумма квадратов ошибок, второй — член регуляризации, позволяющий уменьшить значения весов и предотвратить переобучение. Параметр регуляризации весов λ используют для проверки относительной значимости частей данного выражения. В задачах бинарной классификации y представлен 0 или 1. Задача — минимизировать J(W,b). Для обучения нейронной сети необходимо инициализировать каждый параметр $W_{ij}^{(l)}$ и $b_i^{(l)}$ малыми случайными величинами, близкими к нулю, а затем применить алгоритм оптимизации.

Так как J(W, b) не является выпуклой функцией, то градиентный спуск восприимчив к локальным оптимумам. Каждая итерация градиентного спуска обновляет параметры таким образом:

$$W_{ji}^{(l)} = W_{ji}^{(l)} - \propto \frac{\partial}{\partial W_{ji}^{(l)}} J(W,b) \text{ (2.25)}$$

$$b_i^{(l)} = b_i^{(l)} - \propto \frac{\partial}{\partial b_i^{(l)}} J(W,b) \text{ (2.26)}$$

где α — скорость обучения (learning rate).

$$\frac{\partial}{\partial W_{ji}^{(l)}} J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial W_{ji}^{(l)}} J\left(W,b;x^{(i)},y^{(i)}\right)\right] + \lambda W_{ji}^{(l)} \text{ (2.27)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W,b) = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial b_i^{(l)}} J\left(W,b;x^{(i)},y^{(i)}\right) \text{ (2.28)}$$

Шаги алгоритма обратного распространения ошибки:

1. Осуществляется прямой проход по сети, вычисляются активации слоёв L2, L3 и так далее до выходного слоя L_{nl} ;

2. Для каждого выходного элемента i в выходном слое n_l (the output layer) рассчитывается ошибка

$$\delta_i^{(n_i)} = \frac{\partial}{\partial z_i^{(n_i)}} \frac{1}{2} ||y - h_{W,b}(x)||^2 = -(y_i - a_i^{(n_i)}) \cdot f'(z_i^{(n_i)}) \quad (2.29)$$

3. Для каждого элемента в слое l, рассчитывается

$$\delta_i^{(l)} = (\sum_{j=1}^{s_l+1} W_{ji}^{(l)} \delta_i^{(l+1)}) f'(z_i^{(l)}) \quad (2.30)$$

4. Вычисляются частные производные

$$\frac{\partial}{\partial W_{ji}^{(l)}} J(W, b; x, y) = a_i^{(l)} \delta_i^{(l+1)} \quad (2.31)$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y) = \delta_i^{(l+1)} \quad (2.32)$$

Для обучения нейронной сети, нужно повторно выполнить шаги градиентного спуска, чтобы уменьшить значение целевой функции J(W, b).

Недостатки градиентного спуска

Основная трудность обучения нейронных сетей состоит в методах выхода из локальных минимумов. Недостатками градиентного спуска при обучении сети являются:

1. Паралич сети: значения весов сети в результате коррекции могут стать очень большими величинами. Поскольку ошибка, посылаемая обратно в процессе обучения, пропорциональна производной сжимающей функции, то процесс обучения может почти остановиться. Это можно предотвратить, уменьшая шаг η, однако процесс обучения будет происходить дольше.

2. Размер шага: если значение шага не изменяется и оно довольно мало, то метод сходится слишком медленно. Если же шаг слишком велик, то может возникнуть паралич сети. Необходимо изменять значение шага: увеличивать до тех пор, пока не прекратится улучшение оценки в направление антиградиента и уменьшать, если оценка не улучшается.

2.2.7 Функция хаотического отображения

Хаотическое отображение – разностное уравнение, описывающее дискретную динамическую систему, которая является хаотичной.

Рассмотрим следующие хаотические отображения.

Система Лоренца

Аттрактор Лоренца описывается следующей системой уравнений:

$$\begin{cases} \acute{x} = \sigma(y - x) \\ \acute{y} = x(r - z) - y \\ \acute{z} = xy - bz \end{cases} \quad (2.33)$$

где параметры σ и r связаны соответственно с числами Прандтля и Рэлея, а третий параметр b описывает геометрию системы. Параметры, при которых система демонстрирует хаотическое поведение: $\sigma = 10$, $r = 28$, $b = \frac{8}{3}$.

Система Чуа

Математическая цепь Чуа имеет вид:

$$\begin{cases} \frac{dx}{d\tau} = \alpha\big(y - x - h(x)\big), \\ \frac{dy}{d\tau} = x - y + z, \\ \frac{dz}{d\tau} = -\beta y, \end{cases} \quad (2.34)$$

где x, y, z – фазовые переменные, а

$$h(x) = m_1 x + \frac{1}{2}(m_0 - m_1)(|x + 1| - |x - 1|) \quad (2.35)$$

При значениях параметров α= 9, $\beta = \frac{100}{7}$, $m_0 = -\frac{8}{7}$, $m_1 = -\frac{5}{7}$ траектория системы демонстрирует хаотическое поведение.

Система Рёсслера

Хаотический аттрактор, которым обладает нелинейная система дифференциальных уравнений Рёсслера:

$$\begin{cases} \frac{dx}{dt} = -y - z \\ \frac{dy}{dt} = x + ay \\ \frac{dz}{dt} = b + z(x - c) \end{cases} \qquad (2.36)$$

где a, b, c – положительные постоянные.

Поведение аттрактора Ресслера в значительной степени зависит от значений постоянных параметров a, b, c. Изменение каждого параметра дает определенный эффект, в результате чего система может сойтись к периодической орбите, к неподвижной точке или устремиться в бесконечность. Количество периодов системы Ресслера определяется числом его витков вокруг центральной точки, возникающих перед серией петель.

Перемешивающее свойство отображений

Одним из положительных свойств хаотических отображений является хорошее перемешивание входных данных, то есть зависимость каждой координатной функции отображения от возможно большего числа переменных. В противном случае элементы ключа могут быть найдены из решения соответствующей системы уравнений методами типа «разделяй и вскрывай» [23]. «Динамическая система $\dot{x} = F(x)$, $x(0) = x0$ называется перемешивающей, а фазовый поток – перемешивающим, если при $t \to \infty$ существует предел отношения меры (объема) $\mu(Et \cap B)$ области $Et \cap B$ к мере (объему) $\mu(B)$ области B, равный

$$\lim_{t\to\infty} \frac{\mu(E_t \cap B)}{\mu(B)} = \frac{\mu(E)}{\mu(A)}, \qquad (2.37)$$

где E,$B-$ области A, $\mu(E)$, $\mu(B)$, $\mu(A)$ $-$ меры (объемы)соответствующих областей, область$B-$ неподвижна, областьE $-$ эволюционирует с течением времени под действием непрерывного отображения φ^t ,т.е. $E_t = \varphi^t(E)$.» [24]

Определение перемешивающей динамической системы имеет следующий смысл. Если рассматриваемая динамическая система консервативна, то мера области E сохраняется $\mu(Et) = \mu(E)$. Отношение $\mu(E)/\mu(A)$ есть доля объема,

занимаемого областью E в области A. Отношение $\mu(Et \cap B)/\mu(B)$ есть доля объема, занимаемая в области B попавшими внутрь нее в момент t частями области t E . Определение (2.37) утверждает, что в пределе при $t \to \infty$ эти два отношения совпадают независимо от размера, формы и взаимного расположения областей E и B.

Из определения перемешивающих систем можно сделать ряд выводов о характере поведения таких систем. Во-первых, область B можно сделать сколь угодно малой и поместить в любое место области A. Если выполнено соотношение (2.37), то это означает, что с ростом t части области E можно найти с одинаковым успехом в любой точке фазового пространства, доступного для движения системы. Следовательно, область E с течением времени превращается в сколь угодно тонкую паутинку, пронизывающую всю область A фазового пространства. Объем этой паутинки равен объему исходной области E. Паутинка равномерно и однородно пронизывает фазовое пространство. Внутри любой, наугад взятой области, нити паутинки занимают одну и ту же относительную долю объема.

Исходную область E можно выбрать сколь угодно малой и расположить в любой доступной области фазового пространства. Тем не менее, как следует из (2.37), при $t \to \infty$ эта область превратится в пронизывающую все паутинку. Это означает, что в динамической системе с перемешиванием фазовые траектории всегда неустойчивы по отношению к малым возмущениям и разбегаются с течением времени.

Во-вторых, поведение системы с перемешиванием непредсказуемо. Так как малая область фазового пространства с течением времени трансформируется во все пронизывающую паутинку, то малая область неопределенности начальных условий при достаточно больших временах наблюдения заполнит всю доступную область E фазового пространства. При достаточно большом времени наблюдения для любой точки области A фазового пространства найдется сколь угодно малая окрестность, содержащая фазовую траекторию, начинающуюся в малой окрестности начальной точки. Таким

образом, если в начальный момент времени положение фазовой точки было известно с конечной точностью, т. е. известна принадлежность точки некоторой области с характерным размером ε, то сказать, где она окажется через достаточно длительный промежуток времени, невозможно.

В-третьих, перемешивание приводит к необратимости. Любая неопределенность начальных условий через некоторое время заполняет всю область А. Зная, что в момент времени t система находится в некоторой малой окрестности ε точки $x(t)$ фазового пространства, мы не можем указать начальное состояние системы, так как к моменту времени t в окрестность ε может попасть траектория, начинающаяся в малой окрестности любой начальной точки.

Поскольку математическое моделирование систем можно производить лишь с конечной точностью, то для систем с перемешиванием детерминированное описание при больших временах оказывается лишенным смысла, более адекватным будет стохастическое описание.

При эволюции диссипативной динамической системы фазовый объем с течением времени сокращается. Фазовые траектории притягиваются к некоторому множеству (аттрактору). В случае хаотического режима фазовая траектория на самом аттракторе выглядит как клубок перепутанных нитей, и малое возмущение траектории может существенно изменить направление движения. Поэтому диссипативная система со странным аттрактором характеризуется глобальным сжатием фазового объема и локальной неустойчивостью фазовых траекторий.

Перемешивающий поток имеет место, если в фазовом пространстве близкие в начальный момент времени точки будут двигаться по сильно расходящимся траекториям.

3 ПРОЕКТИРОВАНИЕ БЛОКА СЖАТИЯ

3.1 Архитектура блока сжатия хеш-функции

На сегодняшний день существуют множество исследований, посвященных компрессионным и диффузионным свойствам нейронных сетей, по разработке потоковых шифров [25,26], блок-шифров [27,28] и хеш-функции [29,30].

Предлагается использовать полносвязную нейронную сеть с хаотическим отображением на выходе. Определим полносвязную нейронную сеть как:

$$Y = f^{T}(W_2(W_1(W_0 * X + B_0) + B_1) + B_2) \qquad (3.1)$$

где i=0,1,2 – номер каждого слоя, Wi и Bi – веса и биас нейронов слоя, представленных в виде матриц, T – количество повторений функции f(Z).

Определим операцию логического сдвига в виде символа >>, тогда функция хаотического отображения f(Z) будет иметь следующий вид:

$$f(Z) = Z * (maxVal - Z) \gg 2 \qquad (3.2)$$

где Z - выход с нейронной сети, maxValue – максимально возможное значение Z + 1.

Данное хаотическое отображение имеет свойство чувствительности к входным параметрам. Если просчитать функцию хаотического отображения T раз (T>10), то небольшие различия во входных параметрах приведут к значительным отличиям в итерируемых значениях. Как правило, при итерации функции хаотического отображения T>50 раз результатом является случайный выход, каждая повторная итерация улучшает частотные характеристики.

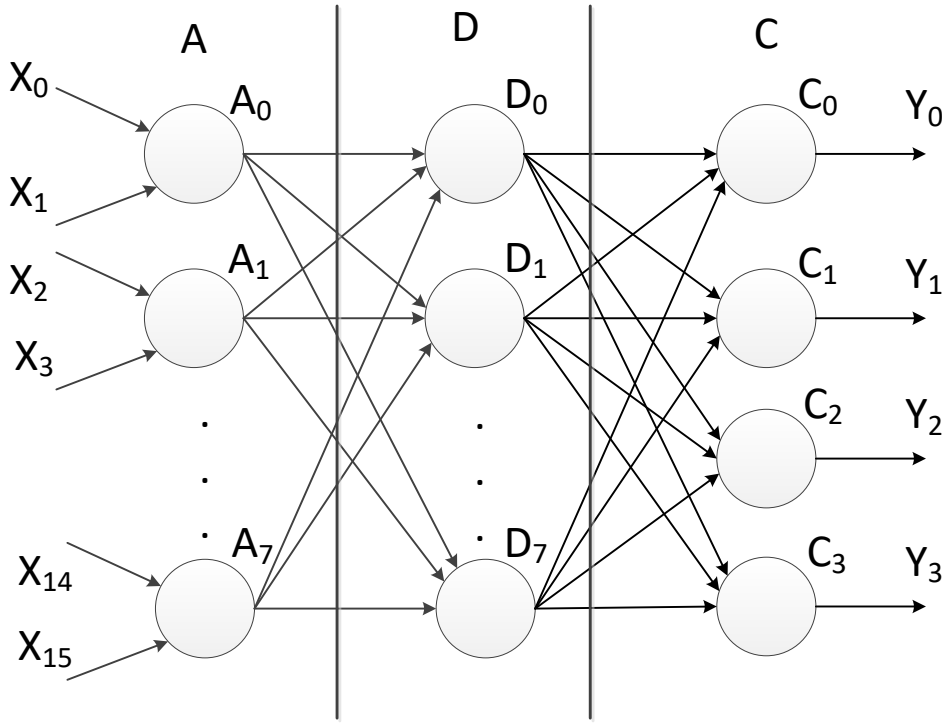


Рисунок 3.1. Блок сжатия хеш-функции

Предлагаемая структура блока сжатия хеш-функции, представленная на рисунке 3.1, является полносвязной нейронной сетью с функцией активации на выходе. Данная структура нейронной сети предполагает эффективную свертку изначальных данных. В основе топологии нейронной сети заложена возможность изменения её структуры в зависимости от требований к хеш-функции. Для улучшения частотных характеристик существует возможность увеличить количество скрытых слоев, использовать хаотическое отображение на скрытых слоях и увеличить входной и выходной вектора за счет пропорционального увеличения структуры нейронной сети. Данные подходы позволят экспоненциально увеличить сложность поиска прообраза хеш-функции при линейном увеличении скорости итерации, что позволяет использовать блок сжатия для проектирования хеш-функции с необходимыми параметрами.

Входной вектор X = [X0X1..X15] состоит из 16 целых чисел без знака размером 64 бит каждое, тем самым образуя размер входного блока 1024 бит.

Слой A = [A0A1..A7] – слой из 8 нейронов без функции активации, содержащий значения весов и биас W0, B0. Размерность матрицы весов и биаса слоя A составляет [8, 1].

Слой D = [D0D1..D7]– слой из 8 нейронов без функции активации, содержащий значения весов и биас W1, B1. Слои A и D являются полносвязными относительно друг друга. Размерность матрицы весов слоя D составляет [8, 8], а биаса – [8,1].

Слой C = [C0C1C2C3] – слой из 4 нейронов с функцией хаотического отображения f(Z) на выходе, содержащий значения весов и биас W2, B2. Слои D и C являются полносвязными относительно друг друга. Размерность матрицы весов слоя C составляет [8, 4], а биаса – [4,1].

Выходной вектор Y = [Y0Y1Y2Y3] состоит из 4 значений без знаковых целых чисел размеров 64 бит, тем самым образуя дайджест сообщения размером 256 бит.

В общем случае при проектировании хеш-функции матрицы W_1, W_2, W_3, B_1, B_2, B_3 могут быть сгенерированы при помощи генератора случайных чисел.

Основным преимуществом данной структуры является высокий уровень параллелизма, что позволяет увеличить скорость итерации алгоритма по сравнению с классическими алгоритмами.

3.2 Анализ свойств разработанного блока сжатия хеш-функции

Свойство однонаправленности

Однонаправленность – это высокая алгоритмическая сложность построения прообраза хеш-функции по её значению. Согласно формуле (3.1) в предложенной функции сжатия выходной вектор является легко вычисляемым значением при известном значении входного вектора и значений параметров нейронной сети, но выполнение обратной функции при известном Y, является вычислительно сложной задачей. Данное свойство обусловлено тем, что задача нахождения множителей при знании только результата произведения не является тривиальной и, также, на последнем слое нейронном слое функцией

активации является итерация T раз функции хаотического отображения (3.1.2). Обратное решение функции хаотического отображения при знании только выходного значения представляет трудность, так как каждая итерация данной функции увеличивает количество ее решений и даже после полного обратного решения верный результат входного параметра не будет известен.

Выход Y представляет из себя:

$$Y_i = f^T(C_i) = \ f^T\left(\sum_{i=0}^{7}\sum_{j=0}^{7} W_{2,i} * \ D_{i*j} + B_{2,i}\right) \qquad (3.3)$$

Используя метод полного перебора для того, чтобы получить X, необходимо вычислить функцию 2^{256} раз для дайджеста сообщения равным 256 бит, что при существующем уровне развития вычислительной техники потребует продолжительного времени.

Высокая чувствительность

Разработанный блок сжатия хеш-функции должен удовлетворять требованию чувствительности, т.е. для входных данных, которые отличаются только 1 битом, вычисляются сильно различные дайджесты. На рисунке 3.2 отображено частотное свойство алгоритма между входными данными, различными в 1 бит, где ось X – входное значение, а ось Y – расстояние Хемминга между соседними значениями.

Также важной характеристикой является частота изменения каждого бита выходной последовательности. На рисунке 3.3 приведено частотное свойство алгоритма по значению каждого бита, где ось X – номер выходного бита, а ось Y – расстояние Хемминга.

Как видно из результатов все значения расстояния Хемминга лежат около 50%, то можно говорить, что предложенный блок сжатия удовлетворяет требованию чувствительности.

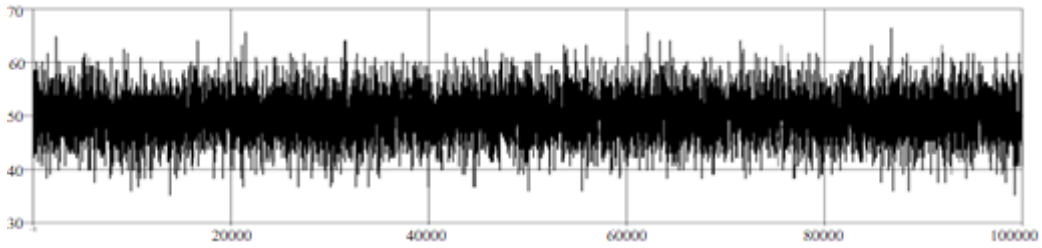


Рисунок 3.2 Частотное свойство функции сжатия между схожими входными данными.

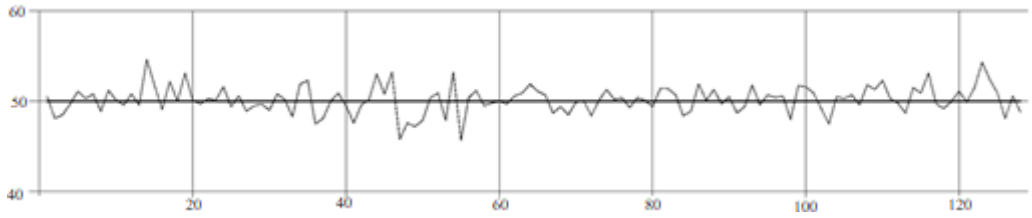


Рисунок 3.3 Частотное свойство функции сжатия по значению каждого бита.

Несмотря на то, что хеш-функции являются достаточно надежным средством обеспечения безопасности, они подвержены ряду атак, которые позволяют обходить свойство необратимости. Рассмотрим самые распространенные их них.

Атака дней рождения

Атака на хеш-функции на основе парадокса дней рождений является одной из самых распространённых. Предлагаемый хеш имеет выходную длину 256 бит, т.е. для поиска коллизии сложность атаки дней рождения будет составлять 2^{128}, а не 2^{256}. Для повышения сложности проведения атаки предлагаемую функцию сжатия можно увеличить в N количество раз, что повысит криптостойкость алгоритма в 2^N.

Встреча по середине

В связи с тем, что нейронная сеть является полносвязной и каждое значение выходного вектора непосредственно зависит от каждого значения входного, невозможно разбить результат на блоки и применять алгоритмы поблочного анализа.

4 РАЗРАБОТКА СИСТЕМЫ

4.1 Выбор инструментов реализации

Благодаря широкому развитию концепции «больших данных», распространение получил такой раздел информатики, как наука о данных (data science). Наука о данных изучает проблемы анализа, обработки и представления данных больших объемов. Наиболее распространенными инструментами для работы с анализом больших данных на сегодняшний день являются C#, Python и R.

4.1.1 Язык программирования R

R — язык программирования, предназначенный для статистической обработки данных и работы с графикой, но в то же время это свободная программная среда с открытым исходным кодом, развиваемая в рамках проекта GNU. R получил широкое распространение в сферах, где проводится работа с данными. Основная вычислительная мощь R заключается в статистическом анализе, однако он также обладает обширным функционалом для первичного анализа данных (построение графиков и таблиц сопряженности) и математического моделирования.

4.1.2 Язык программирования Python

Альтернативным инструментом анализа данных является использование языка Python совместно с библиотеками, предназначенными для решения различных вычислительных задач. Помимо этого, свой популярностью в области анализа данных язык обязан широкому набору библиотек для научных и статистических расчетов, среди которых можно выделить самые широко распространенные: NumPy, pandas, matplotlib и SciPy.

4.1.3 Язык программирования C#

C# - современный объекто-ориентированный и типобезопасный язык программирования, использующий промежуточный байт-код, что позволяет использовать программы, написанные на нем, на различных платформах.

C# имеет следующие функции, обеспечивающие надежность и устойчивость приложений:

1. Сборщик мусора – автоматически освобождает память, занятую неиспользуемыми объектами.
2. Обработка исключений – структурированный и расширяемый способ выявлять и обрабатывать ошибки.
3. Строгая типизация языка – не позволяет обращаться к неинициализированным переменным, выходить за пределы массива и выполнять неконтролируемое приведение типов.

Для работы с машинным обучением и нейронными сетями на платформе .Net Framework используется библиотека Accord.Net, которая позволяет оперировать матрицами, различными статистическими функциями, содержит в себе реализации генетического алгоритма, нейронных сетей, обработку звука и изображений.

Исходя из вышесказанного по совокупности своих преимуществ: скорости исполнения, надежности и безопасности для реализации ВКР был выбран язык C#. В связи с недостаточной гибкостью библиотеки Accord.Net было принято решение самостоятельно реализовать алгоритм работы нейронной сети.

4.2 Проектирование системы

Выделим следующие компоненты, которые необходимо реализовать в рамках данной работы:

1. Компонент блока сжатия хеш-функции, содержащий в себе весь необходимый для его работы функционал.

2. Компонент загрузки весов искусственной нейронной сети из файла, позволяющий считывать значения матриц для их дальнейшего применения в нейронной сети.

3. Компонент статистических тестов должен реализовать функционал проверки на побитные различия между входными последовательностями и производить расчет расстояния Хемминга заданных последовательностей

4. Компонент графического приложения необходим для визуализации результатов функционала пункта 3 и возможности вычисления блока сжатия хеш-функции определенного в пункте 1.

4.3 Программная реализация системы

Разработанная программа состоит из следующих модулей:

1. HashNN.cs
2. StaticTests.cs
3. Form.cs

Компоненты блока сжатия хеш-функции, загрузки весов искусственной нейронной сети из файла и статистических тестов были реализованы в виде динамически подключаемой библиотеки классов.

Компонент графического приложения реализован в виде формы, которая содержит компоненты управления для вычисления хеш-значения и проведения статистических тестов.

Основным модулем приложения является файл HashNN.cs, в котором реализованы методы LogMap, производящий вычисление функции хаотического отображения, NNHash, вычисляющий хеш значение из входной последовательности, и Deserialize, при помощи которого можно проинициализировать веса нейронной сети из файла.

Модуль StaticTests.cs содержит методы анализа статистических свойств блока сжатия хеш-функции: DiffByteArr, считающий процент различий бит дайджеста при входных данных, которые различаются только 1 битом, и StatisticBitTest, который реализует вычисление расстояния Хемминга.

Графическое приложение, реализованное в Form.cs, позволяет производить вычисление хеша заданной последовательности и визуализировать результат работы функции DiffByteArr. Результаты работы приложения представлены на рисунках 4.1 и 4.2.

Диаграмма компонентов представлена на рисунке 4.3.

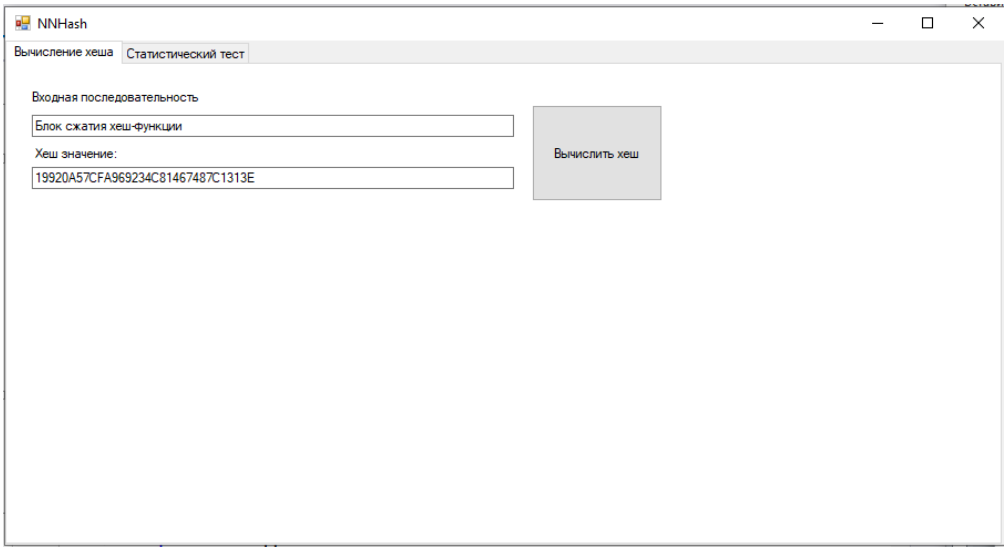


Рисунок 4.1 – Результат работы графического приложения: вычисление хеша

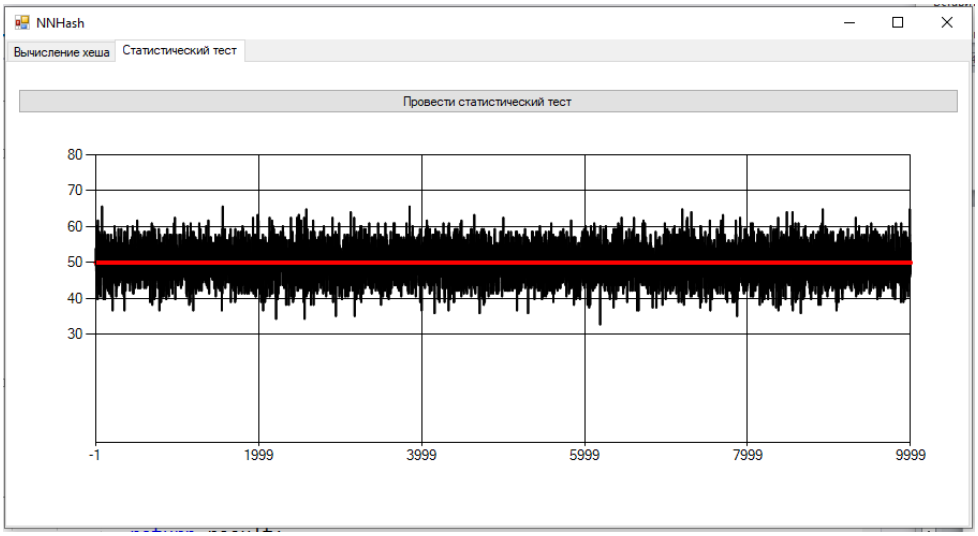


Рисунок 4.2 – Результат работы графического приложения: выполнение статистического теста

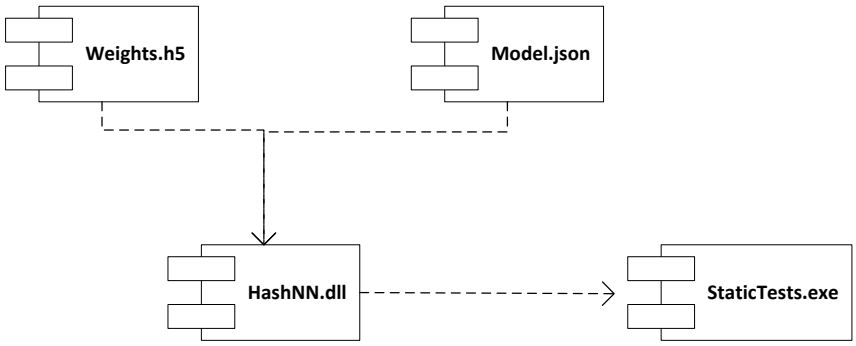


Рисунок 4.3 – Диаграмма компонентов

ЗАКЛЮЧЕНИЕ

В рамках ВКР разработан и проанализирован криптографический блок сжатия для хеш-функции, основанный на нейронной сети.

Предложенный алгоритм реализует свойство односторонности нейронной сети, свойство диффузии и путаницы и удовлетворяет требованиям безопасности. Была разработана архитектура искусственной нейронной сети, функция хаотического отображения и программа проверки статистических характеристик предложенного алгоритма. Реализованную динамическую библиотеку классов можно использовать в качестве компонента для решения конкретных практических задач.

В дальнейшем планируется улучшить структуру алгоритма для повышения безопасности и оптимизации скорости его работы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сяоюн Ванг, Ксуеджия Лай, Дэньгго Фэн, Хой Чэнь, Сююань Юй Криптоанализ функций мешанины MD4 и RIPEMD. // Евросклеп 2005, с. 1–18.
2. Ю Сасаки, Лэй Ван, Казуо Охта, Noboru Kunihiro Новое различие в сообщении для MD4. // Быстрое шифрование программного обеспечения, 2007, с. 329–348.
3. А. А. Кузнецов Параллельный алгоритм подбора одноблочной MD5-коллизии. // ПРОГРАММНЫЕ СИСТЕМЫ №3(26), 2015, с. 61–72.
4. Bansal, A., Singh Bhadauria, S. Watermarking using Neural Network and Hiding the Trained Network within the cover Image. // Journal of Theoretical and Applied Information Technology, 2008, p. 663-670.
5. Jarrod Trevathan, Wayne Read, Alan McCabe. Neural network-based handwritten signature verification. // J Comput, 3 (8), 2008, p. 9–22.
6. Henry Rowley, Baluja S., Kanade T. “Neural Network-Based Face Detection, Computer Vision and Pattern Recognition”, Neural Network-Based Face Detection, Pitts-burgh, Carnegie Mellon University, PhD thesis, 1999
7. Marshalko, “On the security of a neural network-based biometric authentication scheme” // Матем. вопр. криптогр. 5, 2014, с. 87–98.
8. Wolfson H.J., Rigoutsos I. Geometric Hashing: An Overview // IEEE Computational Science and Engineering 4(4), 1997, p. 10-21.
9. G. Brassard Advances in Cryptology // CRYPTO '89, 9th Annual International Cryptology. Conference, Santa Barbara, California, USA, August 20-24, 1989.
10. B. Denton, R. Adhami. Modern Hash Function Construction. [Электронный ресурс]. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.218.1717
11. B. Preneel The First 30 Years of Cryptographic Hash Functions and the NIST SHA-3 Competition.
12. A. Rimoldi. An introduction to Hash functions. [Электронный ресурс]. URL: http://www.science.unitn.it/~sala/BunnyTN/rimoldi.pdf.

13. E. Biham, O. Dunkelman. A framework for iterative hash functions – HAIFA // Cryptology ePrint Archive, Report 2007/278, 2007.

14. B. Burr. NIST Hash Competition: Where we are and what we're learning.

15. T. Peyrin. State-of-the-art of Hash Functions. [Электронный ресурс]. URL: http://www1.spms.ntu.edu.sg/~ccrg/WAC2010/slides/session_4/4_3_Peyrin_hash.pdf .

16. S. Lucks. Design Principles for Iterated Hash Functions. [Электронный ресурс]. URL: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

17. G. Bertoni, J. Daemen, M. Peeters, G. Van Assche. Permutation-based encryption, authentication and authenticated encryption. [Электронный ресурс]. URL: http://keccak.noekeon.org/KeccakDIAC2012.pdf

18. G. Bertoni, J. Daemen, M. Peeters, G. Van Assche. Sponge functions // Ecrypt Hash Workshop, Barcelona, Spain, 2007.

19. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. On the indifferentiability of the sponge construction. In Advances in Cryptology // EUROCRYPT 2008, volume 4965 of Lecture Notes in Computer Science, pages 181–197. Springer, 2008.

20. G. Bertoni, J. Daemen, M. Peeters, G. Van Assche. Keccak specifications [Электронный ресурс]. URL: http://keccak.noekeon.org/Keccak-specifications-2.pdf

21. Информационная технология. Криптографическая защита информации. Функция хеширования. ГОСТ Р 34.11-2012. – Москва, Стандартинформ, 2012.

22. Хайкин С. Нейронные сети: полный курс Neural Networks: A Comprehensive Foundation 2 - е изд. — М.: Вильямс, 2006. – 1104 с.

23. Фомичев В.М. Курс лекций «Дискретная математика и криптология».

24. Шашихин В. Н. Хаос и нелинейная динамика. Регулярная и хаотическая динамика: учебное пособие / В.Н. Шашихин; -СПб.: Изд-во Политехн. ун-та, 2010. – 211 с.

25. C.-K. Chan and L.M. Cheng. The convergence properties of a clipped Hopfield network and its application in the design of keystream generator // IEEE Transactions on Neural Networks, Vol. 12, No. 2, pp. 340-348, March 2001.

26. D.A. Karras and V. Zorkadis. On neural network techniques in the secure management of communication systems through improving and quality assessing pseudorandom stream generators // Neural Networks, Vol. 16, No. 5-6, p899-905, June - July, 2003.

27. S.G. Lian, G.R. Chen, A. Cheung, Z.Q. Wang. A Chaotic-Neural-Network-Based Encryption Algorithm for JPEG2000 Encoded Images // In: Processing of 2004 IEEE Symposium on Neural Networks (ISNN2004), Dalian, China, Springer LNCS, 3174 (2004) 627-632.

28. Liew Pol Yee and L.C. De Silva. Application of multilayer perception networks in symmetric block ciphers // Proceedings of the 2002 International Joint Conference on Neural Networks, Honolulu, HI, USA, Vol. 2, p.1455 – 1458, 12-17 May 2002.

29. Yantao Li Parallel chaotic Hash function construction based on cellular neural network // Neural Comput & Applic., p. 1563–1573, 2011.

30. Dubravin A.V., Petkilev A.A., Balaev K.A., Mezhgorin D.S. CRYPTOGRAPHIC HASH FUNCTION'S COMPRESSION BLOCK CREATION METHOD BASED ON FULLY CONNECTED NEURAL NETWORK / Resumes of the Past and Challenges of the Present plus. 2018, T. 7. № 4 (44). P. 119-121

ПРИЛОЖЕНИЕ А. Листинги программы

Файл «HashNN.cs»

```
using System;
using System.IO;
namespace HashNN
{
    public static class HashNN
    {
        const UInt32 maxV = UInt32.MaxValue;
        public static UInt64 LogMap(UInt64 x, int T)
        {
            if (T != 1){x = LogMap(x, T - 1);}
            return x * (maxV - x) >> 2;
        }
        public static UInt32[] NNHash(UInt32[] inX)
        {
            if (inX.Length != 8){return new UInt32[8];}
            UInt32[]  w1  =  new  UInt32[8]  {  0x41545194,
0x15235174,  0x23448923,  0x29473195,  0x29607547,  0x15219345,
0x27734801, 0x25989939 };
            UInt32[]  b1  =  new  UInt32[8]  {  0x33178731,
0x42362963,  0x29489987,  0x10551244,  0x26759090,  0x11287336,
0x34449337, 0x36385934 };
            int lastLayer = 2;
            int[] OutputRel = new int[2] { 8, 4 };
            int[] InputRel = new int[2] { 8, 8 };
            UInt32[][,] Weight = new UInt32[2][,];
            UInt32[][] Bias = new UInt32[2][];
            UInt32[][] LayersOutput = new UInt32[3][];
            Weight[0]  =  new  UInt32[8,  8]  {  {  0x36240037,
0x29356511,  0x27418565,  0x42447386,  0x97201725,  0x26806368,
0x18588270,  0x13645398  },{  0x30590602,  0x22671984,  0x21540070,
0x29680630,  0x41436244,  0x23202673,  0x26959232,  0x37986624  },{
0x35729170,  0x39855770,  0x37686324,  0x37425916,  0x25912102,
0x54595953,  0x14340172,  0x33721054  },{  0x39281728,  0x33524754,
0x32496764,  0x82363140,  0x85170214,  0x18837667,  0x29518873,
0x13840565  },{  0x21530408,  0x10516664,  0x56019840,  0x40816256,
0x14738826,  0x39596576,  0x98043524,  0x25189513  },{  0x13872538,
0x61805082,  0x39297418,  0x48326124,  0x16028970,  0x25752736,
0x67525398,  0x27914889  },{  0x37815298,  0x42238412,  0x41645114,
0x19973206,  0x21921202,  0x28107426,  0x18944396,  0x30472370  },{
0x28011598,  0x24114396,  0x39667332,  0x37806842,  0x26322540,
0x34960000, 0x15007250, 0x34669008 } };
            Weight[1]  =  new  UInt32[8,  4]  {  {  0x10670950,
0x16244878,  0x41146598,  0x90671148  },{  0x35437060,  0x80157120,
0x37415302,  0x34448256  },{  0x34658360,  0x14559422,  0x85770496,
0x15509902 },{ 0x31742756, 0x20447808, 0x21941860, 0x27287588 },{
0x28609372,  0x29142292,  0x31654584,  0x29635292  },{  0x14153706,
0x23410168,  0x13514410,  0x17489020  },{  0x29236652,  0x85558094,
```

```
0x27017824, 0x40477924 },{ 0x37818224, 0x14940562, 0x29492528,
0x20615047 } };
                Bias[0] = new UInt32[8] { 0x29435336, 0x15304306,
0x24486088,  0x97704060,  0x64553774,  0x37113162,  0x37517390,
0x14628074 };
                Bias[1] = new UInt32[4] { 0x25998175, 0x32174506,
0x12883578, 0x25633915 };
                LayersOutput[0] = new UInt32[8];
                LayersOutput[1] = new UInt32[8];
                LayersOutput[2] = new UInt32[4];
                UInt32[] Y = new UInt32[4];
                for (int i = 0; i < 8; i++){LayersOutput[0][i] =
((inX[i] * w1[i]) % maxV + b1[i]) % maxV;
                    Console.Write(inX[i].ToString("X2") + " ");}
                Console.WriteLine();
                for (int i = 0; i < lastLayer; i++){for (int j =
0; j < OutputRel[i]; j++){UInt64 result = 0;
                        for (int k = 0; k < InputRel[i];
k++){result = result + Weight[i][k, j] * LayersOutput[i][k];}
                        result = result + Bias[i][j];
                        if (i == lastLayer - 1){result =
LogMap(result, 10);}
                        LayersOutput[i + 1][j] = (UInt32)result %
maxV;}}
                Y = LayersOutput[2];
                return Y;}

            public static UInt32[][] Deserialize(Stream stream)
            {
                var reader = new BinaryReader(stream);
                var dimCount = reader.ReadInt32();
                var sizes = new long[dimCount];
                var result = new UInt32[dimCount][];
                for (int i = 0; i < dimCount; ++i){sizes[i] =
reader.ReadInt64();
                    result[i] = new UInt32[sizes[i]];
                }
                return result;
            }
            }
    }
```

Файл «StaticTests.cs»

```
    using System;
    namespace HashNN
    {
        public static class StaticTests
        {
            public static int DiffByteArr(byte[] a, byte[] b)
            {byte[] magicValues = new byte[] { 0,1,1,2,1,2,2,3,
1,2,2,3,2,3,3,4,         1,2,2,3,2,3,3,4,         2,3,3,4,3,4,4,5,
1,2,2,3,2,3,3,4,         2,3,3,4,3,4,4,5,         2,3,3,4,3,4,4,5,
```

```
3,4,4,5,4,5,5,6,          1,2,2,3,2,3,3,4,          2,3,3,4,3,4,4,5,
2,3,3,4,3,4,4,5,   3,4,4,5,4,5,5,6,2,3,3,4,3,4,4,5,3,4,4,5,4,5,5,6,
3,4,4,5,4,5,5,6,4,5,5,6,5,6,6,7,1,2,2,3,2,3,3,4,
2,3,3,4,3,4,4,5,2,3,3,4,3,4,4,5,   3,4,4,5,4,5,5,6,2,3,3,4,3,4,4,5,
3,4,4,5,4,5,5,6,3,4,4,5,4,5,5,6,  4,5,5,6,5,6,6,7,  2,3,3,4,3,4,4,5,
3,4,4,5,4,5,5,6,                    3,4,4,5,4,5,5,6,4,5,5,6,5,6,6,7,
3,4,4,5,4,5,5,6,4,5,5,6,5,6,6,7,  4,5,5,6,5,6,6,7,  5,6,6,7,6,7,7,8,
};
                if (a.Length != b.Length){return -1;}
                byte temp = 0;
                int countOfDiff = 0;
                for (int i = 0; i < a.Length; i++){temp =
(byte)(a[i] ^ b[i]);
                    countOfDiff += magicValues[temp];}
                return countOfDiff;
            }

            public static int[] StatisticBitTest(UInt32[][] a)
            {int[] AllOnes = new int[a[0].Length * 32];
                for (int j = 0; j < a.Length; j++){
                    for (int i = 0; i < a[0].Length; i++){
                        for (int m = 0; m < sizeof(UInt32) * 8;
m++){
                            UInt32 b = a[j][i] & (UInt32.MaxValue
>> m);
                            if (b >= ((UInt32.MaxValue >> m) &
(0x80000000 >> m))){AllOnes[i * 32 + m]++;}}}
                }return AllOnes;}
        }
    }
```

Файл «Form.cs»

```
    using System;
    using System.Drawing;
    using System.Text;
    using System.Windows.Forms;
    using System.Windows.Forms.DataVisualization.Charting;
    namespace HashGUI
    {
        public partial class Form : System.Windows.Forms.Form
        {public Form(){InitializeComponent();}
            private void button2_Click(object sender, EventArgs
e){Chart myChart = new Chart();
                myChart.Parent = panel1;
                myChart.Dock = DockStyle.Fill;
                myChart.ChartAreas.Add(new ChartArea("Math"));
                myChart.ChartAreas[0].AxisY.Interval = 10;
                myChart.ChartAreas[0].AxisY.IntervalOffset = 30;
                Series mySeriesOfPoint = new Series("Func");
                mySeriesOfPoint.ChartType = SeriesChartType.Line;
                mySeriesOfPoint.ChartArea = "Math";
                mySeriesOfPoint.Color = Color.Black;
```

```
                mySeriesOfPoint.BorderWidth = 2;
                Series mySeriesOfPoint2 = new Series("Func2");
                mySeriesOfPoint2.ChartType = SeriesChartType.Line;
                mySeriesOfPoint2.ChartArea = "Math";
                mySeriesOfPoint2.Color = Color.Red;
                mySeriesOfPoint2.BorderWidth = 4;
                UInt32[] y2 = new UInt32[] { 0, 0, 0, 0, 0, 0, 0,
0 };
                for (UInt32 qq7 = 0; qq7 < 10000; qq7++) {
UInt32[] y1 = HashNN.HashNN.NNHash(new UInt32[] { 0, 0, 0, 0, 0,
0, 0, qq7 });
                    int tmp = 0;
                    for (int i = 0; i < y1.Length; i++){tmp +=
HashNN.StaticTests.DiffByteArr(BitConverter.GetBytes(y1[i]),
BitConverter.GetBytes(y2[i]));
                    }
                    int len = y1.Length * sizeof(UInt32) * 8;
                    y2 = y1;
                    mySeriesOfPoint.Points.AddXY(qq7, (float)tmp /
len * 100);}
                for (int i = 0; i < 10000;
i++){mySeriesOfPoint2.Points.AddXY(i, 50);}
                myChart.Series.Add(mySeriesOfPoint);
                myChart.Series.Add(mySeriesOfPoint2);
            }
            private void button1_Click_1(object sender, EventArgs
e)
            {
                var inputStr = textBox1.Text;
                var byteInput = Encoding.ASCII.GetBytes(inputStr);
                UInt32[] input = new UInt32[8];
                for (int i = 0; i < byteInput.Length; i=i+
sizeof(UInt32)){if (i + sizeof(UInt32) > byteInput.Length){break;}

input[i/sizeof(UInt32)]=BitConverter.ToUInt32(byteInput, i);
                }
                UInt32[] y = HashNN.HashNN.NNHash(input);
                string output = "";
                for (int i = 0; i < y.Length; i++){output +=
y[i].ToString("X");}
                textBox2.Text = output;}
        }
    }
```

ПРИЛОЖЕНИЕ В. Презентация

Блок сжатия криптографической хеш-функции на базе полносвязной нейронной сети

Автор работы: студент группы 15ВВ1 Петкилев А.А.
Научный руководитель: к.т.н., доцент Дубравин А.В.

Цель и объект работы

Цель работы: разработка блока сжатия криптографической хеш-функции.

Объектом разработки является алгоритм, который должен обладать следующими свойствами:

- Однонаправленности, достигаемой высокой алгоритмической сложностью построения прообраза хеш-функции по её значению;
- Высокой чувствительности к входным данным – зависимости от всех бит входной последовательности и от их взаимного расположения.

2

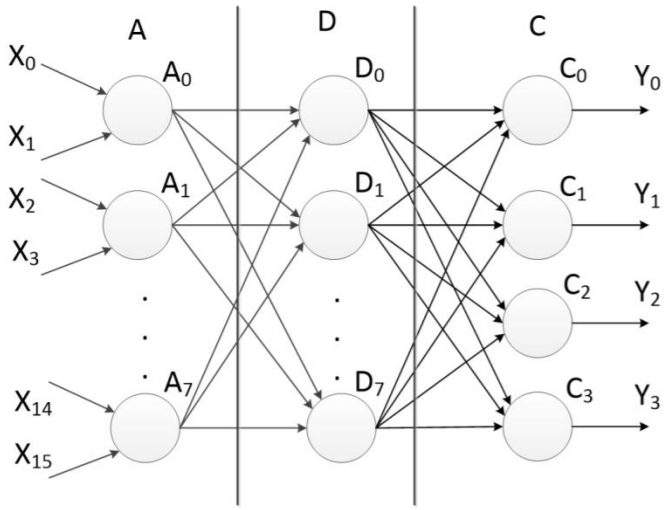
Разработка алгоритма

$$Y = f^{T}(W_2(W_1(W_0 * X + B_0) + B_1) + B_2) \ (1)$$

$$f(Z) = Z * (maxVal - Z) \gg 2 \ \ (2)$$

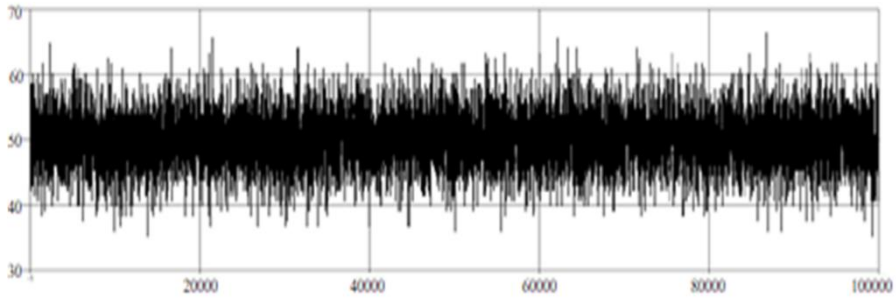
3

Разработка алгоритма. ИНС



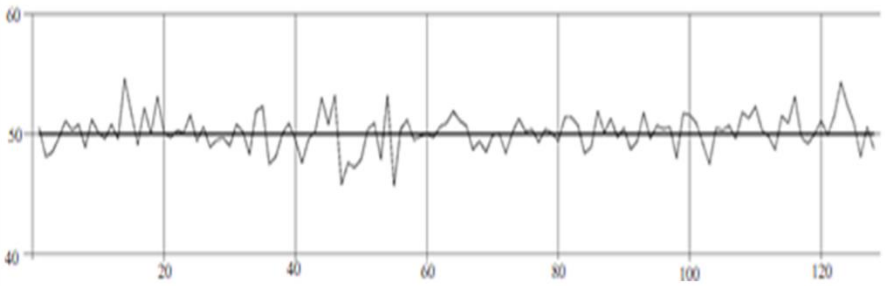
4

Частотное свойство функции сжатия между схожими входными данными



5

Частотное свойство функции сжатия по значению каждого бита



6

Свойство однонаправленности

$$Y = f^{T}(W_2(W_1(W_0 * X + B_0) + B_1) + B_2) \ (1)$$

$$f(Z) = Z * (maxVal - Z) \gg 2 \ (2)$$

$$Y_i = f^T(C_i) = \ f^T\left(\sum_{i=0}^{7} \sum_{j=0}^{7} W_{2,i} * \ D_{i*j} + B_{2,i}\right) (3)$$

7

Результаты работы

- разработан блок сжатия для хеш-функции
- спроектирована архитектура искусственной нейронной сети
- разработана функция хаотического отображения
- реализована программа проверки статистических характеристик предложенного алгоритма

Реализованную динамическую библиотеку классов можно использовать в качестве компонента для решения конкретных практических задач.

8

Благодарю за внимание!