**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc162132915)

[1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 6](#_Toc162132916)

[1.1 Древовидные машины четности 6](#_Toc162132917)

[1.2 Архитектура ДМЧ 8](#_Toc162132918)

[1.3 Виды атак и надежность 12](#_Toc162132919)

[2 ПРАКТИЧЕСКАЯ ЧАСТЬ 18](#_Toc162132920)

[2.1 Реализация класса ДМЧ 18](#_Toc162132921)

[2.2 Синхронизация двух ДМЧ 20](#_Toc162132922)

[2.3 Обучение ДМЧ для атаки 23](#_Toc162132923)

[ЗАКЛЮЧЕНИЕ 26](#_Toc162132924)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 27](#_Toc162132925)

[ПРИЛОЖЕНИЯ 28](#_Toc162132926)

[Приложение А 29](#_Toc162132927)

# ВВЕДЕНИЕ

В современном информационном обществе безопасность передачи и хранения данных является одним из ключевых аспектов, требующих постоянного совершенствования и инноваций. Вместе с тем, рост вычислительной мощности и развитие криптографических алгоритмов подталкивают к необходимости создания новых методов обеспечения конфиденциальности информации.

Одним из важных направлений в рамках нейрокриптографии является исследование древовидных машин четности (ДМЧ). ДМЧ представляют собой особый класс неполносвязных искусственных нейронных сетей, обладающих уникальными свойствами, способными быть примененными в области криптографии.

Целью данной работы является изучение применения древовидных машин четности в нейрокриптографии с целью обеспечения безопасности передачи данных. Конкретно, мы сосредоточимся на исследовании синхронизации двух древовидных машин четности и ее применении в задачах криптографии.

Таким образом, тема данной работы является актуальной для современной криптографии и информационной безопасности.

Задачи, решаемые в данной работе:

1. Изучение принципов работы древовидных машин четности и их применимости в области криптографии.
2. Разработка алгоритма синхронизации двух древовидных машин четности для обеспечения безопасной передачи данных.
3. Оценка потенциальных угроз и сценариев атак на систему с использованием синхронизированных древовидных машин четности.
4. Предложение методов защиты от выявленных угроз и атак.
5. Исследование методов остановки обучения.

# ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## Древовидные машины четности

Древовидные машины четности представляют собой особый класс нейронных сетей, который привлекает внимание в области нейрокриптографии за свои потенциальные возможности в защите информации. Эти машины используются для исследования пространства решений и создания новых типов атак на существующие алгоритмы шифрования. Нейронные сети, включая ДМЧ, могут представлять любую функцию, что делает их мощным инструментом в области криптографии. Их свойства, такие как взаимное обучение, самообучение и стохастическое поведение, а также низкая чувствительность к шуму и неточностям, делают их подходящими для решения различных задач криптографии, включая обмен открытыми ключами, распределение ключей, хеширование и генерацию псевдослучайных чисел.

Идеи нейрокриптографии были впервые представлены Себастьяном Дорленсом в 1995 году. Несмотря на то, что эта область относительно нова и пока не имеет широкого практического применения, уже проведены эксперименты, демонстрирующие потенциал нейрокриптографии. Например, в работе Дорленса была показана возможность использования нейронного криптоанализа для инвертирования S-перестановок в DES, что позволило находить значительную часть ключа за короткое время. Другим примером является протокол шифрования с открытым ключом, разработанный Халилом Шибаба. В данном протоколе процесс расшифрования основан на многоуровневой нейронной сети, обучающейся по алгоритму обратного распространения. Этот метод обладает преимуществами в малых затратах времени и памяти, однако на больших массивах данных нейронная сеть обучается длительное время, что ограничивает его применимость на практике.

Современные информационные системы, ориентированные на повышенный уровень безопасности, в основном, базируются на асимметричной криптографии и хеш-преобразованиях, обеспечивающих необратимость данных. Однако, главным недостатком таких систем является высокая сложность математических операций, что замедляет процесс вычислений. В связи с этим сохраняется актуальность проблемы разработки быстрых протоколов согласования ключей.

Одним интересным направлением стало применение искусственных нейронных сетей (ИНС) для решения задач защиты информации. Эта концепция была впервые предложена И. Кантером и В. Кинцелем. Развитием этой идеи стала архитектура ТРМ (Tree Parity Machine, древовидная машина четности), которая изначально использовала целые действительные числа для описания и анализа процессов в сети.

Для обмена ключами между абонентами часто применяется алгоритм Диффи-Хеллмана или, в более безопасной альтернативе, синхронизация ДМЧ. Этот процесс напоминает синхронизацию двух хаотических осцилляторов в теории хаотических связей. Суть его заключается в том, чтобы достичь полной синхронизации двух ДМЧ и использовать их весовые коэффициенты в качестве общего ключа шифрования. Процесс включает инициализацию весовых векторов обеих сетей, обмен выходными значениями и секретность начальных состояний векторов весов.

## Архитектура ДМЧ

Древовидные машины четности — это особый вид многоуровневой нейронной сети прямого распространения, рисунок 1.

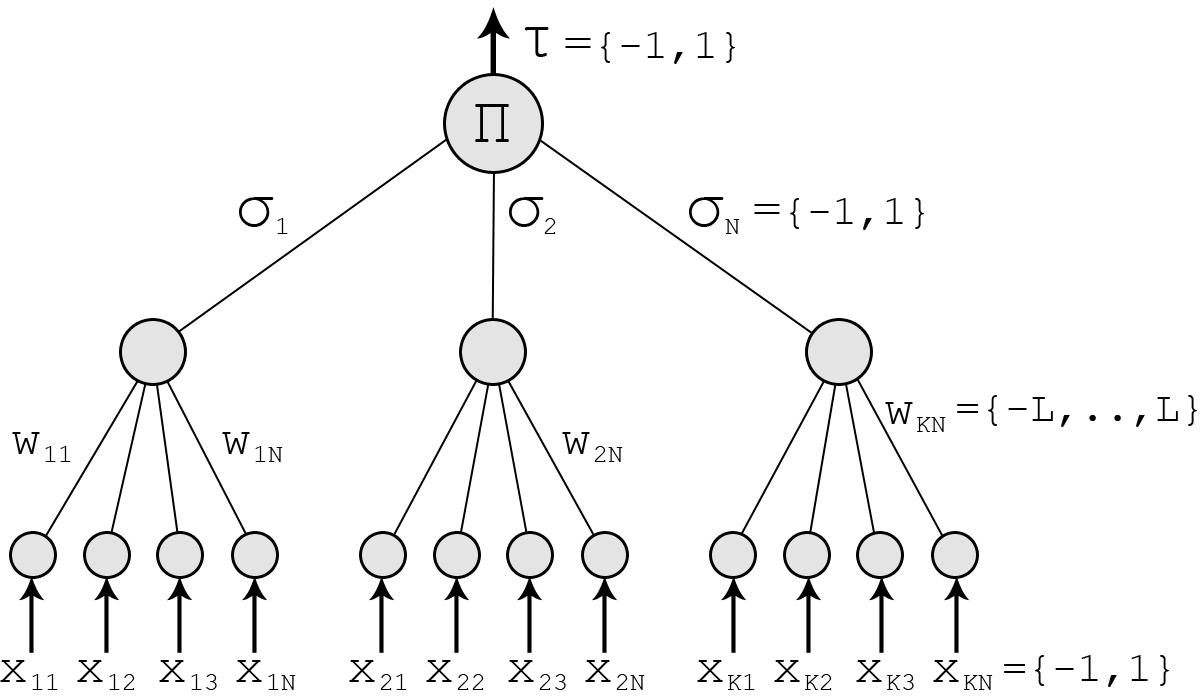


Рисунок 1 — Схема ДМЧ

Она состоит из одного выходного нейрона, K скрытых нейронов и K×N входных нейронов. Входные нейроны принимают двоичные значения:

Веса между входными и скрытыми нейронами принимают значения:

Значение каждого скрытого нейрона есть сумма произведений входного значения и весового коэффициента:

Значение выходного нейрона есть произведение всех скрытых нейронов:

Выходное значение также двоичное.

Для обновления весовых коэффициентов могут использоваться следующие правила:

Правило Хебба:

Анти-правило Хебба:

Случайное блуждание:

Как говорилось выше, для шифрования и расшифрования требуется ключ шифрования, рассмотрим принцип работы системы синхронизации ДМЧ для получения ключа шифрования методом двунаправленного обучения.

У каждого абонента есть своя ДМЧ, рисунок 2. Их синхронизация происходит следующим образом:

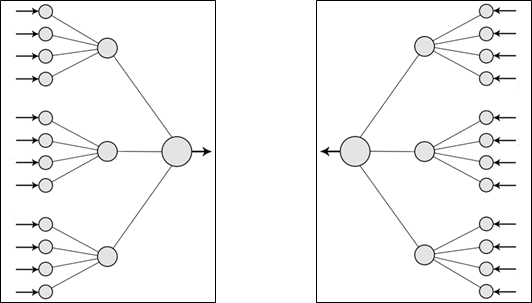


Рисунок 2 — ДМЧ абонентов

1. Задаём случайные значения весовых коэффициентов, рисунок 3.

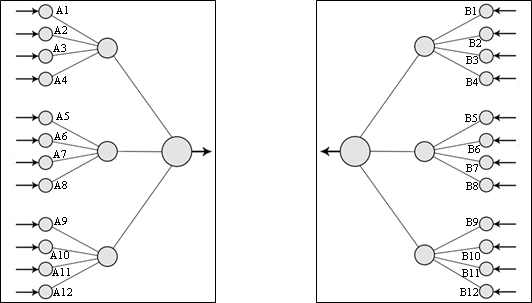


Рисунок 3 — Задание случайных весовых коэффициентов

2. Выполняем следующие шаги, пока не наступит синхронизация.

3. Генерируем случайный входной вектор X, рисунок 4.

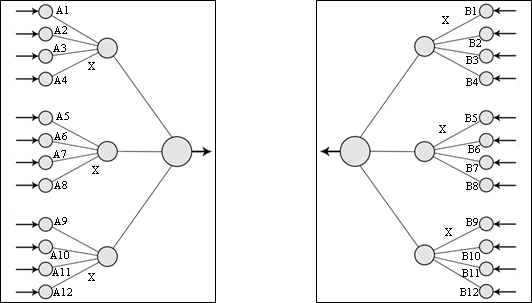


Рисунок 4 — Входной вектор

4. Вычисляем значения скрытых нейронов **N**, рисунок 5.

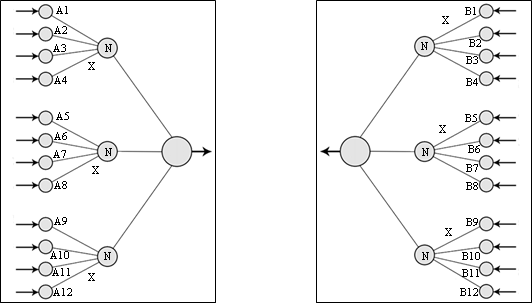


Рисунок 5 — Скрытые нейроны

5. Вычисляем значение выходного нейрона **T**, рисунок 6.

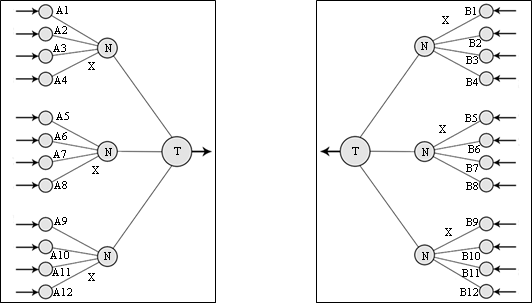


Рисунок 6 — Выходной нейрон

1. Сравниваем выходы двух ДМЧ, рисунок 7.

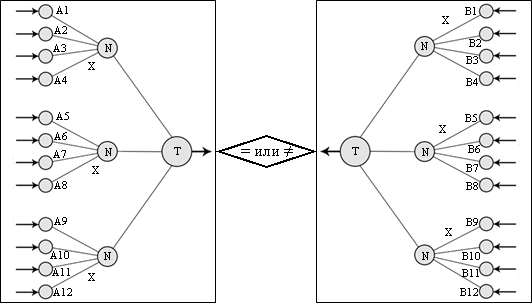


Рисунок 7 — Сравнение выходов

Если выходы разные, то переходим к шагу 2 и генерируем случайный выходной вектор, а если выходы одинаковые, применяем выбранное правило к весовым коэффициентам.

После полной синхронизации, веса обоих ДМЧ одинаковые, их можно использовать в качестве ключа. Данный метод вполне оправдывает себя, стоит помнить, что любой метод может быть подвержен воздействию, направленному на искажение структуры работы и несанкционированному получению результата работы системы.

Итак, при рассмотрении процесса обучения двух ДМЧ возникает вопрос о том, когда следует прекращать его выполнение. Иначе говоря, требуется найти критерии синхронизации, удовлетворив который, можно будет остановить алгоритм обучения. Возможны следующие подходы:

Полный перебор:

Обеим ДМЧ подаются на вход все возможные входные векторы, а их вычисленные выходные значения сравниваются. При достижении синхронизации при всех выходных векторах все соответствующие выходы должны совпадать. Такой подход является самым неэффективным ввиду слишком большого количества эпох двунаправленного обучения. В случае увеличения числа сравнений результатов работы ДМЧ двух абонентов криптоаналитик с высокой вероятностью успеет обучить свою ДМЧ нужной структуры, и тем самым получить генерируемый секретный ключ шифрования.

Итеративный подход:

Заключается в эмпирическом оценивании необходимого числа итераций. Также является не самым лучшим вариантом. В зависимости от начальных состояний ДМЧ абонентов требуемое количество итераций может сильно варьироваться. Результатом может стать как недостаточное количество эпох обучения ДМЧ, так и превышение необходимого их количества, что повысит шансы криптоаналитика на получение секретного ключа.

И тд. (ДОПИСАТЬ)

Явное сравнение:

## Виды атак и надежность

Перед рассмотрением различных видов атак и их степени надежности важно отметить, что в каждом сценарии криптоаналитик (Е) предполагается имеющим возможность подслушивать сообщения между А и Б, но не имеющим способности изменять их содержание.

Метод грубой силы:

Подход к атаке заключается в том, чтобы проверить все возможные варианты ключей или весовых коэффициентов. Для этого криптоаналитику необходимо рассмотреть огромное количество комбинаций, что на практике делает такую атаку невозможной. Например, для сетей с K скрытыми нейронами, K×N входными нейронами и максимальным весом L, количество возможных комбинаций составляет что при больших значениях K, N и L приводит к колоссальной сложности, это делает такую атаку невозможной.

Обучение собственной ДМЧ:

Допустим, криптоаналитик обладает нейронной сетью того же типа, что и у абонентов, и стремится синхронизировать её с двумя другими нейронными сетями.

Круговая синхронизация, рисунок 8.

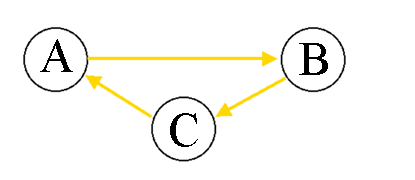


Рисунок 8 — Круговая синхронизация

Этот метод атаки описывается как процесс нелегальной синхронизации трех ДМЧ, маскируемый под обычную синхронизацию двух ДМЧ. На канал передачи информации встраивается атакующая сторона, воздействующая только в одном направлении.

Алгоритм синхронизации работает следующим образом:

1. ДМЧ-А вычисляет функцию и передает результат на ДМЧ-В;
2. ДМЧ-В корректирует веса, вычисляет функцию и передает результат на ДМЧ-С;
3. ДМЧ-С корректирует веса, вычисляет функцию и передает результат на ДМЧ-А;
4. ДМЧ-А корректирует веса, вычисляет функцию и передает результат на ДМЧ-В;

Этот процесс повторяется до достижения полной синхронизации всех трех ДМЧ.

Хотя для этого требуется больше времени и большее количество итераций без дополнительных вычислений, невозможно использовать полученные данные для атаки на ДМЧ. Это связано с тем, что синхронизация двух ДМЧ может занимать разное количество времени: иногда она завершается за несколько итераций, а иногда может затянуться на сотни итераций, и точно предсказать необходимое время невозможно.

Маятник рисунок 9.

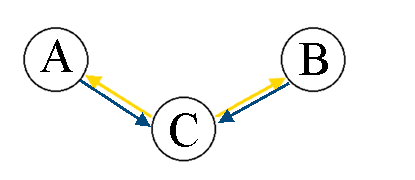


Рисунок 9 — Маятник

Синхронизация третьей ДМЧ с двумя другими ДМЧ контрагентами чередуется и представляет собой сложную атаку, которая легко обнаруживается по результатам вычислений. Однако стоит отметить, что данный метод ускоряет синхронизацию ДМЧ за счет того, что весь диалог проходит через атакующего.

Шаги алгоритма синхронизации следующие:

1. ДМЧ-А вычисляет функцию и отправляет результат на ДМЧ-С;
2. ДМЧ-С корректирует веса, вычисляет функцию и передает результат на ДМЧ-В;
3. ДМЧ-В корректирует веса, вычисляет функцию и передает результат на ДМЧ-С;
4. ДМЧ-С снова корректирует веса, вычисляет функцию и передает результат на ДМЧ-А;

Этот процесс коррекции и вычисления функции повторяется до полной синхронизации всех трех ДМЧ.

Этот метод синхронизации требует большего числа операций при учете всех трех ДМЧ, однако, если учесть, что ДМЧ-А и ДМЧ-В не знают о существовании ДМЧ-С, то количество итераций останется прежним. Тем не менее, время на выполнение этих итераций заметно увеличится, что делает данный метод атаки легко обнаружимым. Более того, по окончании синхронизации можно сравнить отправленные данные одной стороны с полученными данными другой стороны, что позволит сразу выявить факт атаки и принять меры по обеспечению безопасности системы шифрования, например, пересоздать ключ, используя альтернативный канал связи.

Синхронизация суммированием рисунок 10.

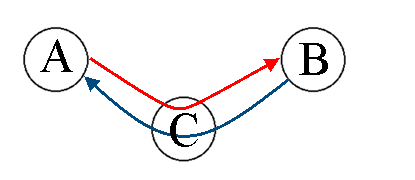


Рисунок 10 — Синхронизация суммированием

Алгоритм синхронизации следующий:

1. ДМЧ-А вычисляет функцию и передает результат на ДМЧ-С;
2. ДМЧ-С корректирует веса, вычисляет функцию, суммирует результаты и передает их на ДМЧ-В;
3. ДМЧ-В корректирует веса, вычисляет функцию и передает результат на ДМЧ-С;
4. ДМЧ-С корректирует веса, вычисляет функцию, суммирует результаты и передает их на ДМЧ-А;

Этот процесс повторяется до полной синхронизации всех трех ДМЧ.

Этот метод атаки достаточно быстрый и при этом малозаметный, однако его легко вычислить, когда результаты вычислений проверяются на промежуточном этапе. Его единственным плюсом является скорость. Факт атаки легко обнаружить, проверив результаты вычислений в процессе синхронизации: любые расхождения указывают на атаку, и можно принять меры для обеспечения безопасности данных.

Для понимания атаки наиболее удобным методом является использование обученной нейронной сети. При выполнении синхронизации через надежный канал следует регистрировать время, количество итераций и разницу в вычислении функции, чтобы потом использовать эти данные для анализа синхронизации ДМЧ в случае возможной атаки. Система анализа применяется на стороне каждого участника, и если обнаруживается значительное расхождение в каких-либо параметрах, то принимаются меры по предотвращению атаки.

Другие виды атак:

Увеличение уровня защищенности традиционных криптографических систем обычно достигается путем увеличения длины ключа, что делает их более устойчивыми к атакам. Однако в нейрокриптографии используется другой подход: вместо этого увеличивается синаптическая длина, что приводит к значительному усложнению атак на систему. Этот подход позволяет сделать взлом нейрокриптографической системы NP-сложной задачей, в которой затраты на дешифрацию увеличиваются не так быстро, как сложность самой системы.

Исследователи, такие как Александр Климов, Антон Митягин и Ади Шамир, обнаружили, что исходный алгоритм нейросинхронизации уязвим к нескольким видам атак, включая геометрические, вероятностные и генетические алгоритмы. И хотя эти уязвимости делают данную реализацию несовершенной, концепция случайной синхронизации может послужить основой для разработки абсолютно надежной криптографической схемы.

Генетическая атака:

Атака основана на создании большого количества нейронных шифровальных устройств (НШУ), которые представляют собой нейронные сети с абсолютно идентичной структурой, как у абонентов А и Б. В процессе обмена информацией между абонентами либо происходит отбрасывание НШУ, которые не приносят пользы, либо наращивание числа НШУ, которые могут быть перспективными для взлома. Формализованный алгоритм представляет собой следующее:

Взломщик инициализует свою машину только с одной НШУ. Веса для него выбираются случайно. Также устанавливается некоторое число М — порог популяции НШУ, которое может себе позволить взломщик.

Затем при обмене информацией между А и Б могут возникнуть три ситуации:

1. Если выходные значения А и Б не равны между собой: Output(A) ≠ Output(B), обновление весов не происходит. Взломщик не трогает свою популяцию.

2. Если Output(A) = Output(B) и число шифровальных устройств у взломщика не превышает порог М, то все НШУ заменяются на репрезентацию из F новых НШУ, каждая из которых получается альтернативной заменой скрытого нейрона на противоположное значение, собственно для этого подбираются новые веса. Затем происходит обучение по правилам Хеббиана.

3. Если Output(A) = Output(B) и число шифровальных устройств у взломщика превышает порог М, то удаляются все те нейромашины, выходное значение которых Output(E) ≠ Output(A).

Этот метод работает только с небольшими нейронными сетями (N, K <= 3) и становится неэффективным даже на мощных компьютерах при K >= 6. Фактически, все предложенные Александром Климовым и Антоном Митягиным атаки основаны на предположении, что нейронные шифровальные устройства имеют небольшой размер. Однако на практике редко используются шифровальные нейронные сети с параметрами N < 100, K < 100, L < 10.

Квантовые компьютеры:

В квантовых компьютерах информация хранится в кубитах, что позволяет решать более сложные задачи, такие как дискретный логарифм и факторизация, значительно быстрее. В связи с этим становится крайне важным разработать алгоритмы, не зависящие от этих численных проблем.

Нейронный протокол обмена ключами не опирается на теорию чисел, а основан на различии между однонаправленной и двунаправленной синхронизацией нейронных сетей. Именно поэтому такие протоколы могут значительно ускорить процесс обмена информацией.

# ПРАКТИЧЕСКАЯ ЧАСТЬ

## Реализация класса ДМЧ

Класс Tree Parity Machine (TPM) представляет собой модель нейронной сети, которая используется для решения задач обучения с учителем. Приведем описание его полей и методов:

Поля:

1. num\_inputs: Количество входов (размерность входного массива).
2. num\_neurons: Количество нейронов.
3. max\_weight: Максимальное значение веса.
4. weights: Матрица весов размера [num\_neurons, num\_inputs].
5. inputs: Входные данные.
6. neuron\_outputs: Выходы нейронов.
7. global\_output: Глобальный выход.

Методы:

\_\_init\_\_(self, num\_inputs, num\_neurons, max\_weight): Конструктор класса, инициализирующий поля объекта.

predict(self, inputs): Метод для предсказания выхода на основе входных данных.

Алгоритм:

1. Умножить каждый входной сигнал на соответствующий вес нейрона.
2. Применить функцию активации (в данном случае - знак).
3. Если выход нейрона равен 0, установить его на 1.
4. Вычислить глобальный выход как произведение выходов нейронов.

update\_weights(self, target\_output): Метод для обновления весов на основе целевого выхода.

Алгоритм:

1. Если целевой выход совпадает с текущим глобальным выходом:
2. Получить индексы нейронов с совпадающим выходом.
3. Обновить веса только для этих нейронов путем добавления произведения входных данных и целевого выхода.
4. Ограничить значения весов максимальным и минимальным значением.

encrypt\_text(self, text): Метод для шифрования текста с использованием текущих весов.

Алгоритм:

1. Кодировать текст в последовательность чисел Unicode.
2. Умножить закодированный текст на транспонированную матрицу весов.
3. Вернуть зашифрованное сообщение.

decrypt\_text(self, encrypted\_message): Метод для дешифрования зашифрованного сообщения.

Алгоритм:

1. Умножить зашифрованное сообщение на псевдообратную матрицу весов.
2. Декодировать числа обратно в символы Unicode.
3. Вернуть расшифрованный текст.

Опишем функцию, которую подаем на вход ДМЧ:

generate\_random\_input(num\_neurons, input\_size) генерирует случайный вход для Tree Parity Machine. Она принимает количество нейронов num\_neurons и размер входного массива input\_size. Возвращает случайный вход, который представляет собой матрицу размером [num\_neurons, input\_size], состоящую из случайных значений -1 и 1.

Алгоритм:

1. Начать с получения двух входных параметров: num\_neurons (количество нейронов) и input\_size (размер входного массива).
2. Создать случайный входной массив, используя функцию np.random.choice.
3. Для каждого элемента массива:
   1. Выбрать случайным образом значение из списка [-1, 1], представляющее собой возможные выходные значения нейрона.
4. Вернуть полученный массив как случайный вход для Tree Parity Machine.

## Синхронизация двух ДМЧ

Процесс синхронизации был подробно рассмотрен в п. 1.2, перейдем к описанию алгоритма синхронизации.

1. Настройка параметров:
   1. num\_neurons: количество нейронов в TPM.
   2. input\_size: размер входных данных.
   3. max\_weight: максимальный вес для связей между нейронами.
   4. sync\_iterations: список для хранения номеров итераций синхронизации.
   5. alice\_weights\_diff: список для хранения разницы весов между Алисой и Бобом.
   6. count\_syns: счетчик для отслеживания числа последовательных синхронизаций.
2. Создание объектов TPM для Алисы и Боба.
3. Основной цикл for, который выполняет итерации для синхронизации весов:
   1. Генерируются случайные входные данные input\_data.
   2. Прогнозируются выходы для Алисы и Боба (alice\_output и bob\_output).
   3. Если выходы различаются, счетчик сбрасывается, и происходит переход к следующей итерации.
   4. Если выходы совпадают, веса обновляются для обоих TPM.
   5. Записываются данные для построения графика.
   6. Если количество последовательных синхронизаций превышает заданное значение (заранее просчитанное эмпирическим путем), цикл прерывается.

Синхронизация прошла успешно, и цикл завершился после 468 итерации. Посмотрим на график синхронизации от количества итераций, где по оси “Y” будет разность весов Алисы и Боба, а по оси “X” номер итерации, рисунок 11.

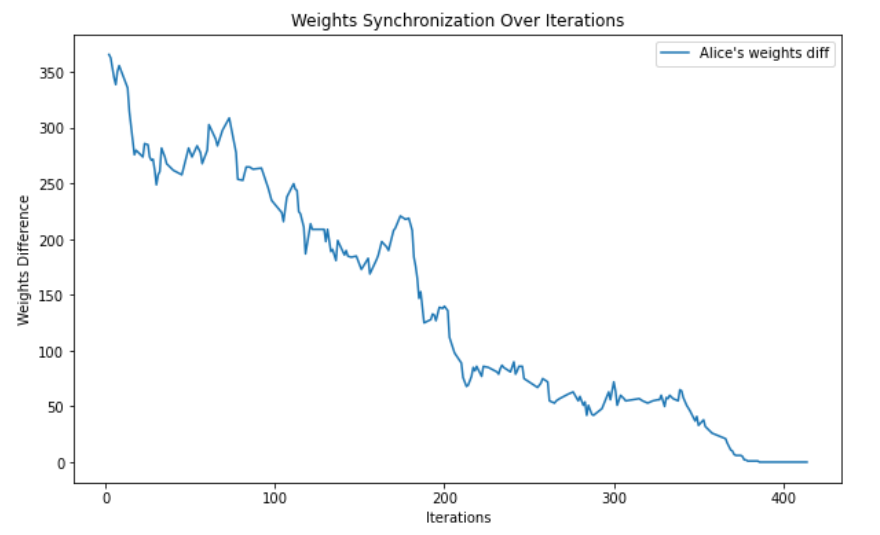


Рисунок 11 — Синхронизация

Приведем матрицы Алисы и Боба и заметим, что они полностью совпадают рисунок 12, 13.

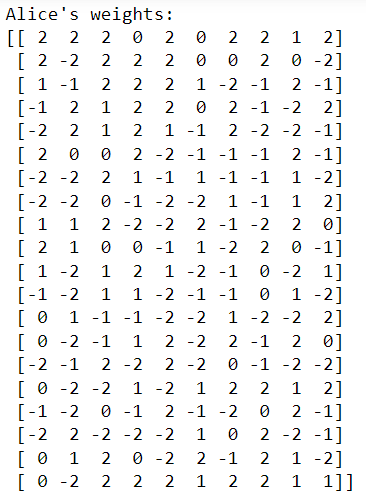


Рисунок 12 — Веса Алисы

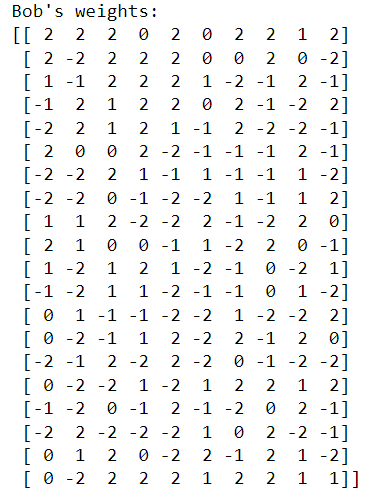


Рисунок 13 — Веса Боба

Далее мы можем использовать получившиеся веса для шифрования сообщения. Попробуем передать наше зашифрованное сообщение, сообщение выглядит, как массив чисел, рисунок 14.



Рисунок 14 — Зашифрованное сообщение

После передачи и дешифровки сообщения мы получили исходное сообщение, которое и передавали.

## Обучение ДМЧ для атаки

По алгоритму, описанному в пункте 1.3. сымитируем атаку на наш алгоритм. Выберем алгоритм круговой синхронизации, для синхронизации весов трех ДМЧ потребовалось в 3 раза больше итераций 1465. Посмотрим на то, какие значения весовых коэффициентов получились у наших ДМЧ после синхронизации Алисы и Боба, рисунки 15-17.

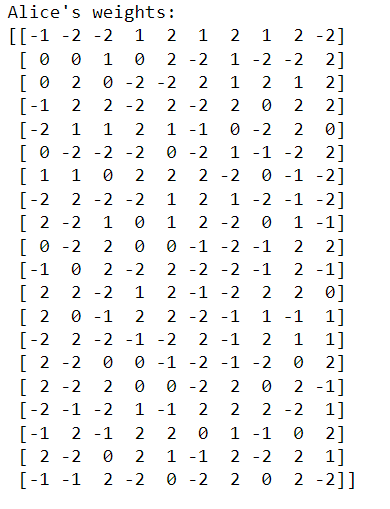


Рисунок 15 - Веса Алисы

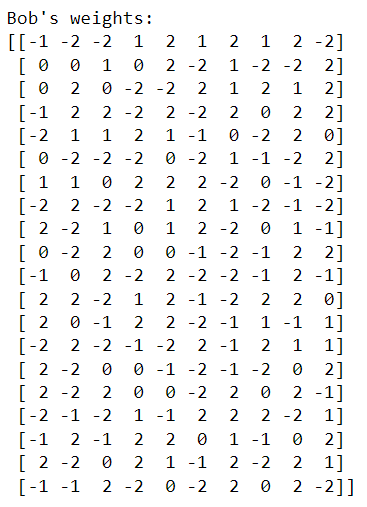


Рисунок 16 – Веса Боба

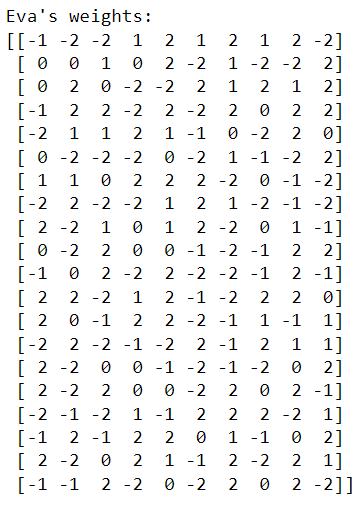


Рисунок 17 – Веса Евы

Заметим, что веса всех трех ДМЧ одинаковы, а значит атака прошла успешно. Проверив и передав сообщение, ДМЧ Евы так же смогла его расшифровать.

# ЗАКЛЮЧЕНИЕ

В ходе данного исследования мы рассмотрели применение древовидных машин четности в нейрокриптографии с целью обеспечения безопасности передачи данных. Основываясь на изученных принципах работы таких систем, разработали алгоритм синхронизации двух древовидных машин четности для обеспечения безопасной передачи информации. Проведенный анализ потенциальных угроз и сценариев атак позволил выявить слабые места в системе и предложить методы их защиты.

Наше исследование продемонстрировало, что древовидные машины четности обладают потенциалом быть эффективным средством защиты информации в рамках нейрокриптографии. Однако, несмотря на их преимущества, необходимо учитывать возможные атаки и угрозы, такие как методы грубой силы, генетические алгоритмы и другие.

Для повышения безопасности передачи данных рекомендуется применение дополнительных методов защиты, таких как мониторинг сетевой активности, шифрование данных и использование аутентификации. Кроме того, важно проводить регулярное обновление системы с учетом новых угроз и атакующих методов.

Таким образом, результаты нашего исследования подтверждают актуальность и значимость применения древовидных машин четности в нейрокриптографии для обеспечения безопасности информации в современном информационном обществе.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

* 1. The first definition of the Neuro-Cryptography (AI Neural-Cryptography) applied to DES cryptanalysis by Sebastien Dourlens, France.

* 1. [Применение искусственных нейронных сетей и системы остаточных классов в криптографии](https://search.rsl.ru/ru/record/02000010936) / [Червяков Н. И. и др.].— Москва: Физматлит, 2012.
  2. Нейрокриптографическая защита информации / Научная статья Никишова А. В., Умницын М. Ю.

# ПРИЛОЖЕНИЯ

Приложение А — Листинги кода программы.

## Приложение А

Код класса TPM представлен на листинге 1.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78 | **import** **numpy** **as** **np** # Для работы с массивами  **from** **matplotlib** **import** pyplot **as** plt # Для рисования    # Класс ДМЧ  **class** **TPM**:  **def** **\_\_init\_\_**(self, num\_inputs, num\_neurons, max\_weight):  """  Инициализация класса TPM.    :param num\_inputs: Количество входов (размерность входного массива)  :param num\_neurons: Количество нейронов  :param max\_weight: Максимальное значение веса  """  self.num\_inputs = num\_inputs  self.num\_neurons = num\_neurons  self.max\_weight = max\_weight  self.weights = np.random.randint(-max\_weight, max\_weight + **1**, [num\_neurons, num\_inputs])  self.inputs = []  self.neuron\_outputs = np.zeros([num\_neurons])  self.global\_output = **0**    **def** **predict**(self, inputs):  """  Предсказание выхода на основе входных данных.    :param inputs: Входные данные  :return: Результат предсказания  """  **if** inputs.shape[**0**] != self.num\_neurons:  inputs = inputs.reshape([self.num\_neurons, self.num\_inputs])    neuron\_outputs = np.sign(np.sum(inputs \* self.weights, axis=**1**))  neuron\_outputs[neuron\_outputs == **0**] = **1**    global\_output = np.prod(neuron\_outputs)    self.inputs = inputs  self.neuron\_outputs = neuron\_outputs  self.global\_output = global\_output    **return** global\_output    **def** **update\_weights**(self, target\_output):  """  Обновление весов на основе целевого выхода.    :param target\_output: Целевой выход  """  weights = self.weights  **if** target\_output == self.global\_output:  indices = self.neuron\_outputs == target\_output  weights[indices, :] = weights[indices, :] + self.inputs[indices, :] \* target\_output  weights[weights > self.max\_weight] = self.max\_weight  weights[weights < -self.max\_weight] = -self.max\_weight    self.weights = weights    **def** **encrypt\_text**(self, text):  """  Шифрование текста с использованием текущих весов.    :param text: Входной текст  :return: Зашифрованное сообщение  """  encoded\_text = np.array([ord(char) **for** char **in** text])  encrypted\_message = np.dot(encoded\_text, self.weights.T)  **return** encrypted\_message    **def** **decrypt\_text**(self, encrypted\_message):  """  Дешифрование зашифрованного сообщения.    :param encrypted\_message: Зашифрованное сообщение  :return: Расшифрованный текст  """  decoded\_text = np.dot(encrypted\_message, np.linalg.pinv(self.weights.T))  decoded\_text = ''.join([chr(int(round(char))) **for** char **in** decoded\_text **if** **0** <= round(char) <= **0x10FFFF**])  **return** decoded\_text |

Листинг 1 — Класс TPM

Код синхронизации ДМЧ Алисы и Боба представлен на листинге 2.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66 | # Параметры ДМЧ  num\_neurons = **10**  input\_size = **20**  max\_weight = **2**  # Списки для хранения данных  sync\_iterations = []  alice\_weights\_diff = []  count\_syns = **1**  # Создание объектов TPM для Алисы и Боба  alice\_tpm = TPM(num\_neurons, input\_size, max\_weight)  bob\_tpm = TPM(num\_neurons, input\_size, max\_weight)    # Число итераций синхронизации  num\_iterations = **2000**    **for** iteration **in** range(num\_iterations):  # Генерация случайного входа  input\_data\_a = generate\_random\_input(num\_neurons, input\_size)  # input\_data\_b = generate\_random\_input(num\_neurons, input\_size)    # Предсказание выходов для Алисы и Боба  alice\_output = alice\_tpm.predict(input\_data\_a)  bob\_output = bob\_tpm.predict(input\_data\_a)    **if** alice\_output != bob\_output:  count\_syns = **1**  # Если выходы разные, переходим к следующей итерации  **continue**    count\_syns += **1**  # Если выходы равны, обновляем коэффициенты  alice\_tpm.update\_weights(bob\_output)  bob\_tpm.update\_weights(alice\_output)    # Записываем данные для графиков  sync\_iterations.append(iteration)  alice\_weights\_diff.append(np.sum(np.abs(alice\_tpm.weights - bob\_tpm.weights))) # Разница весов Алисы и Боба      **if** count\_syns > **30**:  **print**("Synchronization achieved at iteration:", iteration)  **print**("Alice's weights:")  **print**(alice\_tpm.weights)  **print**("**\n**Bob's weights:")  **print**(bob\_tpm.weights)  **print**("Alice's output:", alice\_output)  **print**("Bob's output:", bob\_output)  **break**    # Шифрование и дешифрование текста  encrypted\_message = alice\_tpm.encrypt\_text("Hello Bob!")  decrypted\_message = bob\_tpm.decrypt\_text(encrypted\_message)  **print**(encrypted\_message)  **print**(decrypted\_message)  # Построение графиков  plt.figure(figsize=(**10**, **6**))  plt.plot(sync\_iterations, alice\_weights\_diff, label="Alice's weights diff")  plt.xlabel("Iterations")  plt.ylabel("Weights Difference")  plt.title("Weights Synchronization Over Iterations")  plt.legend()  plt.show() |

Листинг 2 - Синхронизация