**Исследовательская работа**

**на тему:** Алгоритм шифрования на основе элементарного клеточного автомата

**Автор работы:**

**Обучающийся 8 класса**

**ГБОУ Гимназия №1538**

Чистобаев Андрей Андреевич

**Руководитель работы:**

Носкова Людмила Николаевна,

Преподаватель математики, высшая категория

**Консультант:**

Дергачёва Алевтина Николаевна

Преподаватель информатики, первая категория

Миядин Александр Андреевич

Студент 2 курса МГТУ имени Н.Э. Баумана

**Москва**

**2017 год**

**Содержание**

**Обоснование темы**……………………………………………………………….**3**

**Цели и задачи**…………………………………………………………………….**4**

**Гипотеза**…………………………………………………………………………...**5**

**Методика**………………………………………………………………………….**6**

**Тестирование**………..……………………………………………………………**7**

**Критика взлома**………………………………………………………………….**9**

**Итоги**……………………………………………………………………………..**10**

**Литература**………………………………………………………………………**11**

**Приложение 1. Словарик**……………………………………………………...**13**

**Приложение 2. О работе элементарных клеточных автоматов**………….**15**

**Приложение 3. О работе алгоритма**………………………………………….**18**

**Приложение 4. Сравнительная характеристика**…………………………...**22**

**Обоснование темы**

Сохранение *конфиденциальности* данных пользователей есть главная обязанность разработчиков современности. Для её выполнения созданы различные средства защиты информации – *криптографические алгоритмы* и *хэш-функции*. Конечно, не все из них применимы в реальных проектах. Такие бесполезные инструменты представляют собой лишь научный интерес.

При выборе метода защиты опытный *криптограф* перво-наперво обратит своё внимание на такое важное свойство, как *криптостойкость*. Уже потом он взглянет на производительность, новизну и другие качества. Криптостойкость определяется годами, поэтому не стоит рваться за последними трендами.

Для определения уровня защиты, предоставляемого тем или иным инструментом, нужно подвергнуть *шифротекст*, им созданный, *криптоанализу*. Этим я и решил заняться.

Создав собственный *блочный* алгоритм необратимого шифрования, основанный на *элементарных* *клеточных автоматах*, я поставил перед собой цель доказать его практическую ценность. В ходе исследования, разумеется, алгоритм будет модифицирован для достижения лучших результатов.

С 2008 года в открытом доступе имеется описание алгоритма, основанного на *обратимых клеточных автоматах*. Это совершенно иная идея. Сравнение её с моей возможно, но не желательно, так как перед этими инструментами стоят разные задачи.

**Цели и задачи**

Передо мной стоит единственная цель – доказать пригодность моего алгоритма.

Эту цель я собираюсь решить созданием и тестированием своей собственной реализации алгоритма шифрования.

Это решение можно разбить на несколько более мелких задач:

1. **Изучение основ криптографии и криптоанализа.** Необходимо изучить основные термины и методы работы.
2. **Изучение основ работы элементарных клеточных автоматов.** Изучением данной темы я занимаюсь уже 2 года.
3. **Изучение основ одного из языков программирования.** У меня имеются достаточные знания по C++ для написания подобной программы.
4. **Создание реализации алгоритма.** Реализация уже имеется и выдаёт верные результаты.
5. **Тестирование реализации.** Тестирование будет производиться компьютерной программой.
6. **Доработки.** Конечно, многое придётся изменить.
7. **Выкладывание реализации в свободный доступ.** При помощи этого мне удастся получить мнение других людей.
8. **Использование реализации на практике.** Возможность практического применения обсуждается в приложении 3.

План логичен и последователен, следовательно, можно начинать работу.

**Гипотеза**

У данного алгоритма, подробно описанного в приложении 3 (+ листинги с подсветкой синтаксиса), имеется существенный недочёт – открытый текст шифруется необратимо. То есть привычное равенство не выполняется:

***D( E( M ) ) ≠ M***

Формула взята из книги Брюса Шнайера «Прикладная криптография».

E – Encipher – функция, шифрующая исходное сообщение.

D – Decipher – функция, расшифровывающая шифротекст.

M – Message – исходное сообщение, открытый текст.

Но в современном мире уже существуют и активно используются подобные алгоритмы.

Данные алгоритмы применимы при авторизации пользователя в компьютерной сети. То есть пароль, введённый пользователем, сохраняется в базу данных в зашифрованном виде. Так как шифрование необратимое, *открытый текст* (пароль) будет известен только вводившему его пользователю. При авторизации, пароль, введённый в форму, необратимо шифруется и сравнивается с шифротекстом из базы данных, соответствующим введённому логину. При их совпадении доступ к сети открывается.

Но не всё так просто. Я считаю, что, несмотря на необратимость элементарных клеточных автоматов, при определённом *ключе* k равенство ***Dk( Ek( M ) ) = M*** становится верным, то есть расшифровка возможна.

**Методика**

Криптоанализ, его основы, изучался мною по книге Брюса Шнайера «Прикладная криптография». В ней автор выделил 4 основных метода криптоанализа:

1. **Вскрытие с использованием только шифротекста.**
2. **Вскрытие с использованием открытых текстов и соответствующих им шифротекстов.**
3. **Вскрытие с использованием выбранного открытого текста.**
4. **Адаптивное вскрытие с использованием выбранного открытого текста.**

Очевидно, что название метода содержит доступные криптоаналитику данные.

Во всех 4 случаях целью вскрытия является нахождение ключа шифрования, так как при взломе допустимо использование моей реализации алгоритма, что принято в мировой практике (по предположению Керкхофса). То есть определение алгоритма защиты не может быть целью атаки.

Доступ к реализации позволяет подбирать ключ «*грубой силой*».

Примерная структура теста:

1. Сохранение текущего процессорного времени.
2. Попытка взлома «грубой силой» (перебор всех возможных ключей).
3. Сохранение текущего процессорного времени.
4. Вывод времени работы.
5. Вывод результатов.

Программа работает не наглядно, но выдаёт информативный и полноценный результат.

**Тестирование**

Перед запуском теста стоит определиться с целью атаки и инструментом её проведения.

Почему целью выбран ключ?

Во-первых, алгоритм шифрования доступен с самого начала и активно используется во время вскрытия.

Во-вторых, ключ обеспечивает криптоаналитика новыми важными данными:

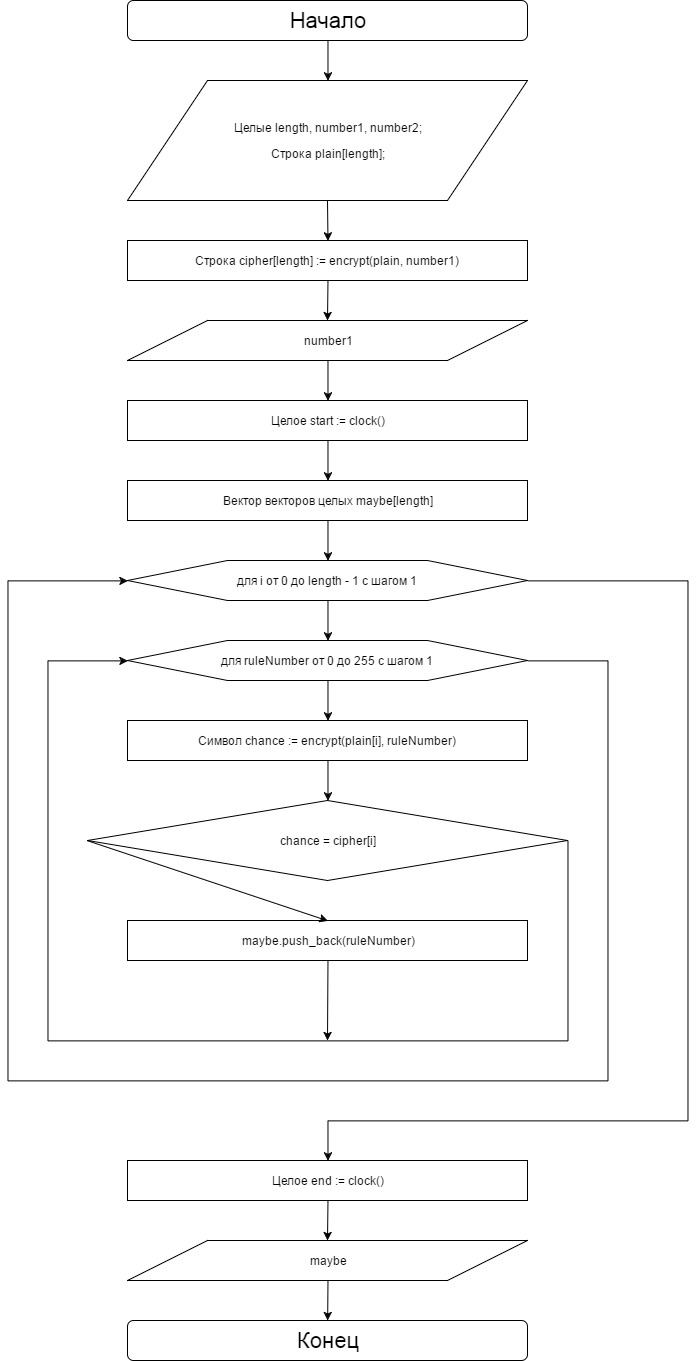
* **Список *столкновений*.** Список столкновений даёт взломщику возможность оптимизировать словарь (исходный символ -> шифрованный символ) для подбора открытого текста, исключив одинаково шифрующиеся символы.
* **Возможность подмены сообщения.** Так как, возможно, существуют обратимые элементарные клеточные автоматы, алгоритм может использоваться для передачи сообщений. Узнав ключ, взломщик сможет зашифровать своё сообщение и подменить им пойманное.

Почему для взлома выбран язык программирования C++?

C++ очень практичный и универсальный инструмент. Поддерживая множество стилей программирования, он не принуждает пользователя следовать одному из них, предоставляя некоторую свободу действий. C++ быстр и выразителен. В этом можно убедиться, взглянув на листинги в приложении 3.

Цель и инструмент вскрытия определены, можно начинать. На следующей странице будет описан этап взлома с известными открытыми текстами и соответствующими им шифротекстами. Исходный код теста доступен в моём репозитории [github](https://github.com/Andrew15381/Cryptographic-algorithm-based-on-linear-Cellular-Automata).

**Атака на основе открытых текстов и соответствующих им шифротекстов.**

****

Версия программы, изображённая в виде блок-схемы, немного упрощена для удобства пояснения, поэтому переменная number2, хранящая число просчитываемых поколений, не использована.

1. На вход поступает строка символов (plain) и ключ шифрования (number1).
2. Исходная строка шифруется с помощью введённого ключа, результат сохраняется в новую строковую переменную (cipher) той же длины.
3. Выводится исходный ключ, для сверки результата.
4. Засекается время начала взлома.
5. Создаётся массив maybe на length элементов-массивов переменной длины, так как изначально неизвестно, каково количество возможных значений ключа на каждый символ.
6. Начинается цикл подбора возможных значений ключа для каждого символа, так как алгоритм отдельно шифрует каждый символ исходной строки с помощью данного ключа.
   1. Начинается цикл подбора возможных значений ключа для отдельно взятого символа исходной строки.
      1. Возможным ключом шифруется отдельно взятый символ исходной строки, результат сохраняется в символьную переменную (chance)
      2. Если значение переменной совпало с символом шифрованной строки, соответствующим отдельно взятому символу исходной строки, то сохраняется возможное значение ключа. Иначе осуществляется переход к следующему возможному значению ключа.
7. Засекается время окончания взлома.
8. Выводятся возможные значения ключа для каждого символа исходной строки.

**Критика взлома**

Так как шифр является блочным (см. приложение 3), можно считать, что дано сразу несколько пар текст–шифротекст. Но по отдельности эти пары бесполезны, потому что для шифрования одного символа в другой пригодны сразу несколько различных ключей. При взломе предполагается, что все символы строки зашифрованы при помощи одного ключа. Поэтому, просчитав все возможные ключи для каждого отдельного символа, возможно найти ключ, общий для всей строки.

При таком подходе результат неоднозначен. В различных элементарных клеточных автоматах при определённом начальном состоянии могут возникать одинаковые поколения. То есть, как описано выше, одному зашифрованному символу соответствуют сразу несколько различных ключей. Поэтому, если ошибочный ключ встретится у каждого символа исходной строки, то он попадёт в ответ. В итоге получится 2 или более равновероятных ключа. Исключить какой-либо из них по известным данным невозможно.

**Итоги**

Подводя итоги, можно сказать, что они лишь промежуточные. Я провёл полноценное исследование своего алгоритма и его реализации. Но множество свойств ещё не изучено до конца. Хотелось бы довести действо до логического завершения.

Передо мной уже стоят некоторые планы на будущее. Зачем нужна реализация, нигде не используемая? Надеюсь, вскоре ей найдётся практическое применение как в моих, так и в других проектах. Для этого нужно сделать множество доработок и новых тестов. Исходный код совершенно открытый, поэтому любой желающий может модифицировать его под свои нужды.

Что касается моей гипотезы, то она подтвердилась. При помощи дополнительного теста мне удалось найти ключи, при использовании которых не возникает столкновений. Углубившись в тему, я узнал, как при помощи этих ключей шифруется символ. Закономерность была найдена во время просмотра кодов исходных и зашифрованных символов.

Клеточный автомат 204, к примеру, совершенно никак не изменяет исходное состояние. При использовании ключа 170 код символа увеличивается вдвое. А 240-ой, наоборот, уменьшает в 2 раза. То есть, зашифровав символ ключом 170, возможно вернуть его, используя ключ 240. Если не обращать внимания на надёжность такого метода, то можно сказать, что алгоритм готов к работе.

Любой комментарий, связанный с перспективами развития, принимается по моему адресу электронной почты – [odmen@itgnews.ga](mailto:odmen@itgnews.ga).

**Литература**

* «C++ для чайников» 2003, Стефан Рэнди Дэвис
* «Программирование. Принципы и практика с использованием C++» 2016, Бьярне Страуструп
* «Прикладная криптография» 2002, Брюс Шнайер
* «Криптосистемы клеточных автоматов» 2008, Росошек, Боровиков, Евсютин – <http://cyberleninka.ru/article/n/kriptosistemy-kletochnyh-avtomatov>

**Приложение 1. Словарик**

**Конфиденциальность** – обеспечение доступа к информации только авторизированных пользователей, *ГОСТ Р ИСО/МЭК 17799-2005*.

**Криптография** – наука о способах преобразования информации с целью её защиты от незаконных пользователей, *В. В. Ященко и другие «Введение в криптографию»*.

**Криптоанализ** – наука получения открытого текста, не имея ключа, *Брюс Шнайер «Прикладная криптография».*

**Криптограф** – специалист в области криптографии.

**Криптоаналитик** – специалист в области криптоанализа.

**Криптографический алгоритм** – математическая функция, изменяющая вид сообщения так, чтобы спрятать его суть, *Брюс Шнайер «Прикладная криптография»*.

**Криптостойкость** – способность шифра противостоять всевозможным атакам на него, *В.В Ященко и другие «Введение в криптографию».*

**Хэш-функция** – функция, получающая на вход строку переменной длины и преобразующая её в выходную строку фиксированной длины, *Брюс Шнайер «Прикладная криптография»*.

**Столкновение** – два прообраза с равными значениями хэш-функции, *Брюс Шнайер «прикладная криптография»*. Но в тексте имеется ввиду два открытых текста с равными шифротекстами. Решение проблемы ищется в приложении 3.

**Открытый текст** – исходное сообщение.

**Шифротекст** – зашифрованное сообщение.

**Ключ шифрования** – секретная информация, используемая криптографическим алгоритмом при шифровании/расшифровке сообщений, постановке и проверке цифровой подписи, вычислении кодов аутентичности, *securitylab.ru*.

**Грубой силой** – вскрытие перебором возможных ключей, *Брюс Шнайер «Прикладная криптография».*

**Блочный шифр** — разновидность симметричного шифра. Особенностью блочного шифра является обработка блока нескольких байт за одну итерацию. Блочные криптосистемы разбивают текст сообщения на отдельные блоки и затем осуществляют преобразование этих блоков с использованием ключа, *Национальная библиотека им. Н. Э. Баумана*.

**Клеточный автомат** – дискретная модель, представляющая собой сетку произвольной размерности, каждая клетка её в каждый момент времени может принимать одно из конечного множества состояний, определено правило перехода клеток из одного состояния в другое, *Андрей Озорнин «Простейшие клеточные автоматы и их практическое применение».*

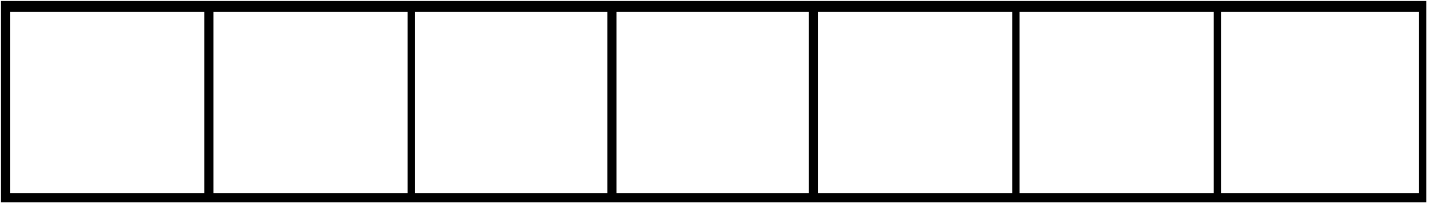
**Обратимый клеточный автомат** – клеточный автомат, в процессе функционирования которого не происходит потери информации, *И.В. Кучеренко «Обратимые клеточные автоматы».*

**Элементарный клеточный автомат** – одномерный бинарный клеточный автомат, где состояние клетки в каждый момент времени зависит только от её собственного состояния и состояний смежных с ней клеток в предыдущий момент времени, *Андрей Озорнин «Простейшие клеточные автоматы и их практическое применение»*.

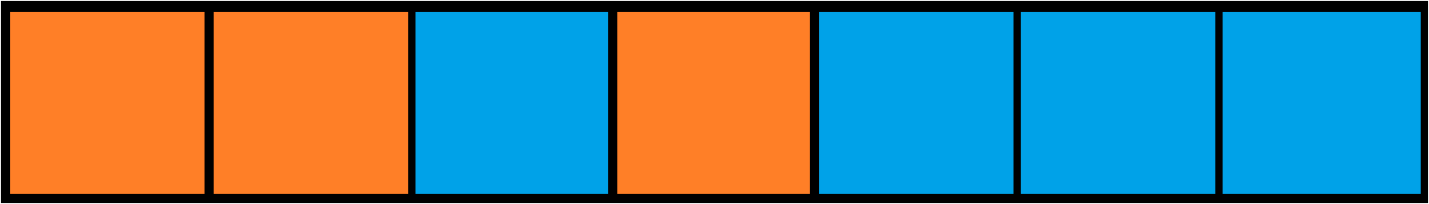
**Приложение 2. О работе элементарных клеточных автоматов**

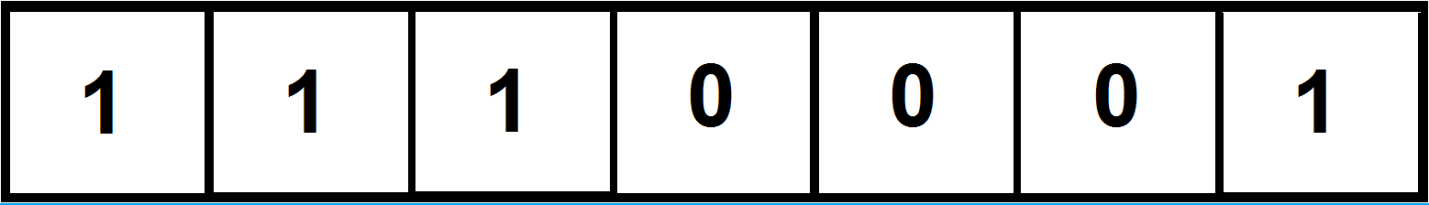
Для полноценного понимания алгоритма стоит узнать основы работы элементарных клеточных автоматов.

Поле элементарного клеточного автомата представляет собой бесконечную одномерную сетку:



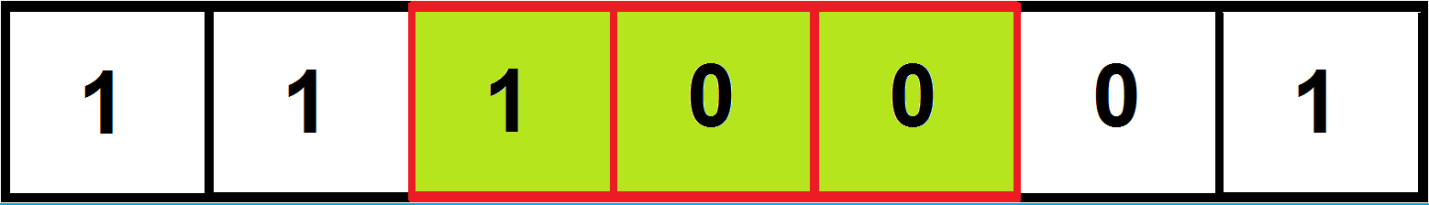
Каждая клетка поля в каждый момент времени имеет одно из двух возможных состояний. Состояние можно обозначить как цветом клетки (чёрный-белый), так и значком (1-0):



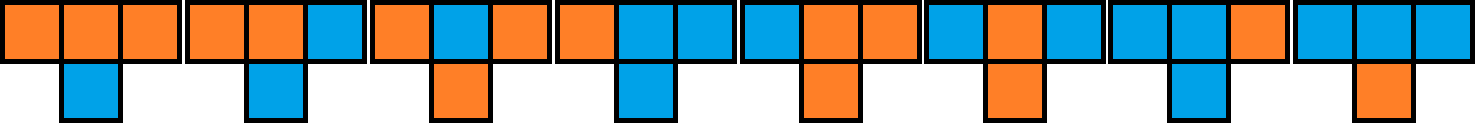


Для начала работы клеточного автомата нужно каждой клетке поля задать её начальное состояние, как и было сделано выше.

Каждая клетка поля имеет свою окрестность – данная клетка и две клетки смежные с ней:



Всего возможно 8 случаев окрестности, каждому из которых правило перехода состояний сопоставляет состояние, получаемое клеткой посередине в будущем поколении:



Это правило будет действовать по всему полю клеточного автомата, причём все клетки обрабатываются одинаково.

Всего существует 256 элементарных клеточных автоматов (28). Каждому из них Стивен Вольфрам причислил номер от 0 до 255 включительно. Для определения номера необходимо переписать представленную выше таблицу, сопоставив каждому цвету 0 либо 1. Пусть оранжевый = 1, а синий = 0.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Окрестность | 111 | 110 | 101 | 100 | 011 | 010 | 001 | 000 |
| Состояние | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |

Представив, что окрестность есть двоичное число, можно заметить, что все случаи записаны в порядке убывания. Это обязательное требование.

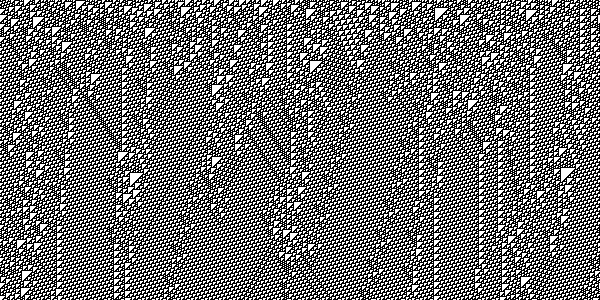
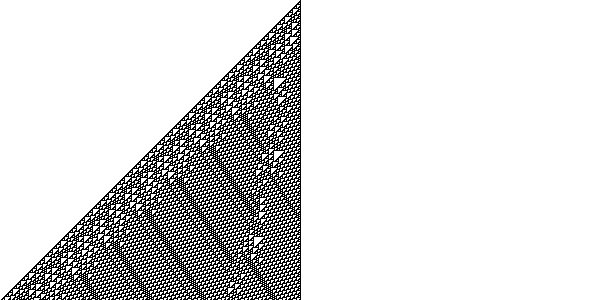
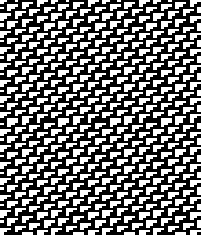
После составления таблицы из неё выносится строка «Состояние». В примере результатом будет строка 00101101. Очевидно, запись представляет собой двоичную запись числа 4510, старший бит слева.

Такая запись получила название «код Вольфрама».

Записанные в хронологическом порядке поколения образуют замысловатый двумерный узор, называемый эволюцией элементарного клеточного автомата. По получаемому узору элементарные клеточные автоматы были разбиты Стивеном Вольфрамом на 4 класса:

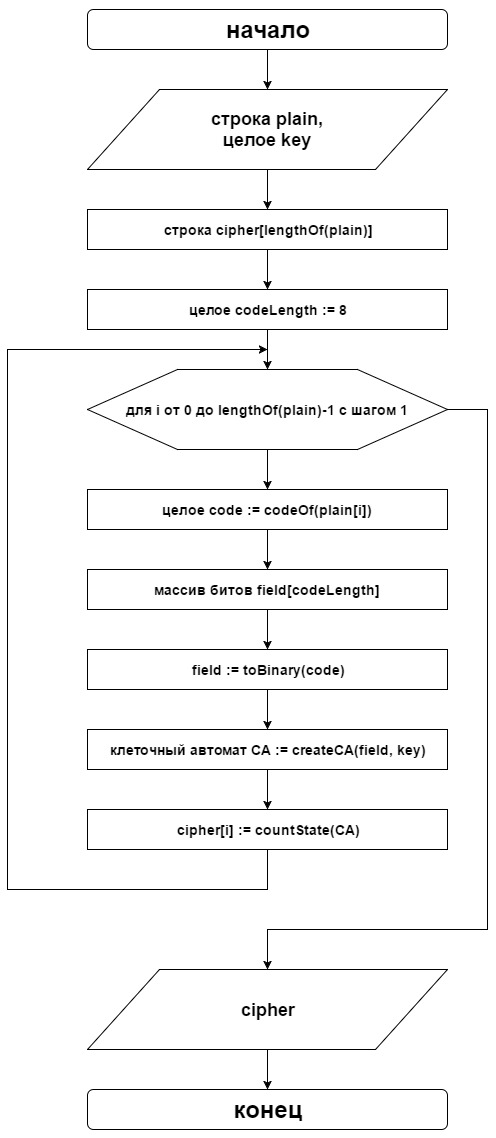
1. Все клетки быстро принимают одинаковое состояние, не меняющееся со временем.
2. Состояния клеток периодически сменяют друг друга со временем.
3. На эволюции отчётливо просматриваются хаотичные непериодические структуры. Небольшие изменения исходного состояния значительно изменяют узор.
4. Автомат порождает сложные устойчивые структуры, взаимодействующие между собой. Не достигает стабильности.

Стоит отметить, что эволюция полностью зависит от начального состояния. К примеру, элементарный клеточный автомат 110 в зависимости от начального состояния может относиться ко всем четырём классам:



**Приложение 3. О работе алгоритма**

В данном приложении не может быть описана конечная версия алгоритма, так как он находится в активной разработке и модифицируется мною достаточно часто.



**Проблемы**

Было выявлено 4 основных проблемы данного алгоритма:

1. **Посимвольное шифрование.** Узнав открытый текст, соответствующий известному шифротексту, криптоаналитик получает сразу несколько пар открытый текст-шифротекст.
2. **Единый ключ для всей строки.** Узнав возможные значения ключа для каждого отдельного символа, быстро определяется ключ, общий для всей строки.
3. **Маленький диапазон значений ключа.** Ключ быстро определяется перебором.
4. **Коллизии.** В алгоритме шифрования это недопустимо.

Все они негативно повлияли на криптостойкость.

**Решения**

Решение так же очевидно, как и сами проблемы. Необходимо перестать разбивать строку на блоки и подавать её на вход целиком, тем самым избавившись от пунктов 1-3. Шифр должен стать потоковым.

Коллизии невозможны при использовании особых ключей:

* 15, обратный – 85.
* 51, обратный – 51 (следующее поколение).
* 85, обратный – 15.
* 105, необратим.
* 150, необратим.
* 170, обратный – 240.
* 204, открытый текст равен шифротексту.
* 240, обратный – 170.

Их список был выяснен опытным путём.

Использование этих пар ключей при шифровании и дешифровании полностью избавляет криптографа от проблемы коллизий.

**Реализация**

На данный момент существует 3 реализации алгоритма, различающиеся лишь производительностью. В приложении я приведу только первый вариант.

**Файл Field.hpp**

#ifndef LIB\_FIELD\_H

#define LIB\_FIELD\_H

#include <cstddef>

#include <climits>

#include <vector>

#include "Rule.hpp"

**namespace** Crypto

{

**class** **Field**

{

**const** Rule rule;

std::vector < **bool** > current;

std::vector < **bool** > next;

**const** **size\_t** length = **sizeof**(**unsigned** **short**) \* CHAR\_BIT;

**public:**

Field(**const** Rule & rule, **const** **unsigned** **short** code):rule(rule)

{

current.resize(length);

next.resize(length);

**for** (**size\_t** i = **0**; i < length; ++i)

current[length - **1** - i] = code & (**1** << i);

}

**unsigned** **short** encrypt()

{

**for** (**size\_t** i = **1**; i < length - **1**; ++i)

next[i] = rule[current[i - **1**] \* **4** + current[i] \* **2** + current[i + **1**]];

next[**0**] = rule[current[length - **1**] \* **4** + current[**0**] \* **2** + current[**1**]];

next[length - **1**] = rule[current[length - **2**] \* **4** + current[length - **1**] \* **2** + current[**0**]];

**for** (**size\_t** i = **0**; i < length; ++i)

current[i] = next[i];

**unsigned** **short** code = **0**;

**for** (**size\_t** i = **0**; i < length; ++i)

code += current[length - **1** - i] \* (**1** << i);

**return** code;

}

**unsigned** **short** encrypt(**unsigned** **short** stateNumber)

{

**for** (**size\_t** i=**1**; i<stateNumber; ++i)

encrypt();

**return** **encrypt**();

}

};

}

#endif

**Файл Rule.hpp**

#ifndef LIB\_RULE\_H

#define LIB\_RULE\_H

#include <vector>

#include <cstddef>

**namespace** Crypto

{

**class** **Rule**

{

std::vector < **bool** > state;

**public:**

Rule(**const** **unsigned** **short** number)

{

state.resize(**8**);

**for** (**size\_t** i = **0**; i < **8**; ++i)

state[i] = number & (**1** << i);

}

**bool** **operator**[] (**const** **unsigned** **short** neighbourhood) **const** noexcept

{

**return** state[neighbourhood];

}

};

}

#endif

Контейнер std::vector<> используется из-за его специализации для типа bool, которая заключается в том, что для каждого элемента выделяется лишь 1 бит вместо 1 байта в массиве.

Замечательно, что в реализации использованы лишь стандартные средства языка C++. Это делает её переносимой и производительной.

Для поддержки многобайтовых кодировок символьный тип заменён целочисленным. В скором времени шифр станет потоковым, и подобные «трюки» будут исключены из программы.

**Приложение 4. Сравнительная характеристика**

Данные таблицы получены посредством вычисления и округления среднего значения для 10 запусков.

**Сравнение вариантов реализации**

|  |  |  |  |
| --- | --- | --- | --- |
| Номер версии | Время на 1 символ с созданием Field, с | Время на 1 символ без создания Field, с | Изменения |
| №1 | 0,00008 | 0,00001 | **----------------** |
| **№2** | **0,00001** | **0,000006** | Тип std::vector<bool> заменён на массив bool, активно используется арифметика указателей |
| №3 | 0,00006 | 0,000015 | Объект rule типа Rule заменён внутренним экземпляром std::vector<bool> |

Примечательно, что проиграв в 8 раз в памяти, 2 версия в 8 раз быстрее.