x – вход сети на текущем шаге,
y – выход сети на предыдущем шаге,
R_o – веса выхода сети для выходного фильтра,
C – состояние памяти сети,
P_o – веса состояния памяти для выходного фильтра,
b_o– сдвиг значений выходного фильтра,
φ – функция активации для значения выходного фильтра.

Итоговое значение сети определяется выходным фильтром (5) и нелинейной трансформацией над состоянием запоминающей ячейки (3):

$$y = tanh(C)g_o \quad (6)$$

гдеC– состояние памяти сети, g_o– значение выходного фильтра, $tanh$ – функция активации для выходного значения сети [2].

Повышенная невосприимчивость к продолжительности разрывов во времени является преимуществом LSTM над обычными рекуррентными нейронными сетями, марковскими моделями и другими методами обучения для последовательностей во многих областях применения.

Суть прогнозирования игры заключается в том, что на основе данных нейронной сети, собранных до текущего момента времени, она определит вероятность победы одной из команд.

На вход в нейронную сеть поступают следующие параметры: количество добытых ресурсов каждой из команд; номер команды, вероятность выигрыша которой мы хотим узнать; текущие предметы каждого из персонажей; количество убийств и использованных вспомогательных предметов, разрушенных зданий противника и оставшихся собственных зданий; опыт каждого из персонажей; распределение способностей у персонажей.

Все время одной партии игры t разбито на промежутки $t = (t_0, t_1, \dots, t_n), t > 0$. Набор игровых параметров представлен вектором $v(t) = (v_0^t, v_1^t, \dots, v_m^t)$, в конкретный момент времени игры t. Номер команды, вероятность выигрыша которой мы хотим узнать (может принимать значения 0 или 1) обозначен c. Входным набором параметров в нейронную сеть будет номер команды c и матрица:

$$X = \begin{pmatrix} v(t_0) \\ v(t_1) \\ \dots \\ v(t_n) \end{pmatrix} = \begin{pmatrix} v_0^{t_0}, v_1^{t_0}, \dots, v_m^{t_0} \\ v_0^{t_1}, v_1^{t_1}, \dots, v_m^{t_1} \\ \dots \\ v_0^{t_p}, v_1^{t_p}, \dots, v_m^{t_p} \end{pmatrix}, p < n \quad (10)$$

Выходным параметром нейронной сети является вероятность $y[0, 1]$ выигрыша команды c.

После обучения нейронной сети на большом объеме выборки, ее можно использовать не только для прогнозирования результата игры, но также в качестве помощника. Dota2 – это командная игра и в начале происходит стадия выбора героев. Данную нейронную сеть можно применить для выбора наилучшего героя в зависимости от уже выбранных. Также ее можно будет применить для выбора наилучшего предмета для героя в зависимости от стадии игры.

Нейронные сети будут эффективны при обучении аналитиков, участников соревнований, разработке более сложных и интеллектуальных стратегий, а также прогнозировании результатов соревнований.

Списокиспользованныхисточников:

1. Result Prediction by Mining Replays in Dota2. [Электронный ресурс] – Режим доступа: https://www.diva-portal.org/smash /get/diva2:829556/FULLTEXT01.pdf. – Дата доступа: 14.03.2018.

2. Рекуррентная нейронная сеть. [Электронный ресурс] – Режим доступа: http://mechanoid.kiev.ua/neural-net-lstm.html. – Дата доступа: 14.03.2018.

ПОСТРОЕНИЕ КРИПТОСТОЙКОГО ГЕНЕРАТОРА ПСЕВДОСЛУЧАЙНЫХ ЧИСЕЛ

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Молчанов И.В.

Калугина М.А.–канд. физ.-мат. наук, доцент

Источники настоящих случайных чисел найти крайне трудно. Ими могут быть, например, физические шумы, такие, как детекторы событий ионизирующей радиации, дробовой шум в резисторе, или космическое излучение. Однако реальное применение таких чисел порождает ряд проблем (время- и трудозатраты при установке и настройке, невозможность воспроизведения ранее сгенерированной последовательности и другие). Поэтому на практике чаще используются генераторы псевдослучайных чисел.

Генератор псевдослучайных чисел (ГПСЧ) — это алгоритм, порождающий последовательность чисел, элементы которой почти независимы друг от друга и подчиняются заданному распределению (обычно равномерному). Во многих задачах современной информатики качество получаемых результатов напрямую зависит от качества используемых ГПСЧ.

Вихрь Мерсенна основывается на свойствах простых чисел Мерсенна и обеспечивает быструю генерацию высококачественных псевдослучайных чисел. Вихрь Мерсенна лишён многих недостатков, присущих другим ГПСЧ, таких как малый период и предсказуемость. Тем не менее, этот генератор не является криптостойким, что ограничивает его использование в криптографии.

Числа Мерсенна в широком смысле — числа вида $2^n - 1$, где n — натуральное число. Иногда числами Мерсенна называют только числа вида $2^p - 1$ с простым показателем p. Вне зависимости от выбранного определения, простое число Мерсенна – число вида $2^n - 1$, являющееся простым (для всех простых чисел вида $2^n - 1$ показатель степени n также является простым числом). С использованием простых чисел Мерсенна связана длина цикла Вихря Мерсенна: $2^{19937} - 1$ (24-е простое число Мерсенна), что позволяет говорить о практической невозможности зациклить данный генератор.

В данной работе алгоритм Вихря Мерсенна реализован в классе Generator. Несмотря на то, что в среде .NET существует несколько генераторов псевдослучайных чисел и, возможно, их использование было бы логичным, подобная работа проведена по следующим причинам:

+ чтобы показать отсутствие зависимостей и возможность перевести код на любой язык;
+ для дальнейшей разработки может понадобиться прямое обращение к коду;
+ чтобы убедиться в возможности получить последовательность,
  имея некоторое количество сгенерированных чисел.

Чтобы показать некриптостойкость Вихря Мерсенна, был также реализован класс Extractor, который по 624 последовательно сгенерированным числам получает исходную последовательность. Алгоритм получения исходной последовательности основан на инвертировании процесса «закалки» в Вихре Мерсенна.

Так как цель данной работы состоит в построении криптостойкого генератора псевдослучайных чисел, то рассмотрим несколько вариантов решения:

Алгоритм Blum-Blum-Shub (генератор псевдослучайных чисел, основанный на вычислительной сложности задачи факторизации больших чисел);

Алгоритм Blum-Micali (генератор псевдослучайных чисел, основанный на вычислительной сложности задачи дискретного логарифмирования);

Композиция любого генератора псевдослучайных чисел и стойкого блочного шифра либо криптографического алгоритма хеширования (сведение задачи о взломе генератора к задаче о взломе стойкого шифра).

Первые два варианта хороши, если речь идет о задачах криптографии, выполняемых на машинах с большой вычислительной мощностью. Оба алгоритма основаны на вычислительной сложности некоторых задач, и, как следствие, имеют низкую скорость работы, что порождает большое количество проблем (например, при генерации ключей – длинных простых чисел). Именно из-за этих возможных проблем был выбран последний вариант построения такого генератора. Учитывая, что мы имеем уже реализованный алгоритм Вихря Мерсенна (класс Generator), то остается только реализовать шифрование его регистра сдвига.

Построим на основе Вихря Мерсенна криптостойкий генератор с помощью стандартных средств шифрования C#. Например, создадим класс, инкапсулирующий обьект класса Generator и использующий класс AesManaged для шифрования регистра. Заметим, что при реализации такого алгоритма мы не ограничены языком программирования C# и шифром AES и сможем без труда перевести код на любой язык с C-подобным синтаксисом, возможно, используя любой другой блочный шифр. В то же время генерация действительно случайных чисел требует источника энтропии, обращение к которому различается в зависимости от языка программирования, что порождает зависимости генератора.

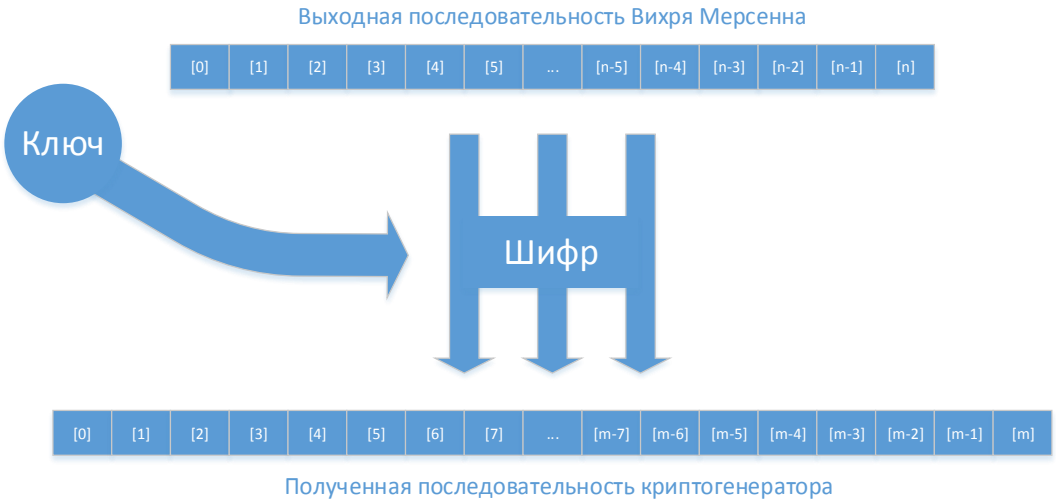
Реализация выбранного решения – это класс CryptoGenerator, инкапсулирующий объект Generator и объект, реализующий интерфейс ICryptoTransform, предназначенный для шифрования регистра сдвига. Для простоты возьмем размер регистра сдвига равным размеру массива uint[260], что представляет собой длину uint[256] + 32 бит (превышение является следствием наложения шифра AES). Стоит заметить, что длина может не меняться или изменяться по-другому вследствие использования другого блочного шифра, что не влияет на работу генератора.

Чтобы получить данные об эффективности построенного генератора, необходимо сравнить его со схожим стандартным средством. В сборке System.Security.Cryptography платформы .NET находится класс RNGCryptoServiceProvider, который предоставляет криптостойкие случайные числа. Стоит заметить, что информации о работе этого генератора крайне мало, что препятствует детальному сравнению с разработанным генератором. Сравнение будем строить на сопоставлении скорости работы и распределении значений генераторов.

При сравнении скорости работы генераторов была замечена следующая особенность: при многочисленных обращениях к стандартному генератору RNGCryptoServiceProvider (1 число – 1 обращение) скорость работы была значительно меньшей, чем при более редких обращениях с заполнением большего по объему буфера. Таким образом, для дальнейших исследований можно поставить дополнительную проблему – исследовать зависимость скорости работы RNGCryptoServiceProvider от внешних факторов, а также исследовать возможный принцип его работы. В общем, можно сказать, что скорость работы обоих генераторов удовлетворяет требованиям задач, в которых они используются (генерация 10^6 чисел происходит за 10 - 500мс).

Что касается исследования распределения, стоит опираться на следующее утверждение: значения, полученные с помощью Вихря Мерсенна без доработок, равномерно распределены. Таким образом, остается проверить, повлияет ли использование шифрования на распределение конечного генератора. Чтобы сравнить распределения генераторов, построим таблицу на основе теоретических расчетов для

равномерного распределения и данных, полученных нашей программой (в качестве параметров распределения возьмем математическое ожидание и дисперсию).



Отрезок	Теоретические		CryptoGenerator		RNGCryptoServiceProvider	
	Матожидание	Дисперсия	Матожидание	Дисперсия	Матожидание	Дисперсия
0..99	49	816	49	833	49	833
0..999	499	83166	499	83326	499	83303
0..9999	4999	8331666	4999	8330739	4999	8333683
0..99999	49999	833316666	49996	833291139	50007	833501899
0..999999	499999	83333166666	499997	83328154282	499961	83330635124

Таким образом, распределение обоих генераторов можно считать равномерным (на основании схожести данных, полученных опытным путем, и теоретических значений).

Кроме криптостойкости и равномерного распределения генерируемых значений, написанный нами генератор имеет и некоторые другие преимущества:

+ функциональность для генерации стандартных типов данных;
+ возможность использовать различные шифры;
+ высокая скорость работы;
+ открытый код и возможность доработки.

Планируется исследовать возможности оптимизации построенного генератора, а также исследовать проблемы, поставленные в ходе данной работы: зависимость характеристик работы генератора от вида шифрования, скорость обращения к объекту RNGCryptoServiceProvider, реализовать возможность генерации различных типов данных. Также планируется построить генераторы с распределением, отличным от равномерного (моделировать заданное распределение различными методами).

Список использованных источников:
1. https://habrahabr.ru/post/196442/
2. https://habrahabr.ru/company/mailru/blog/274253/
3. https://ru.wikipedia.org/wiki/Числа_Мерсенна
4. https://ru.wikipedia.org/wiki/Вихрь_Мерсенна
5. https://ru.wikipedia.org/wiki/Генератор_псевдослучайных_чисел

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ ДЛЯ ДИСТАНЦИОННОГО УПРАВЛЕНИЯ ПЕРСОНАЛЬНЫМ КОМПЬЮТЕРОМ

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Неведомский Д.А.

Алексеев Ю.И., м.т.н.

Системы дистанционного управления используются человеком повсеместно, начиная от домашней бытовой техники и заканчивая сложными системами, такими как беспилотные самолёты. Решений же для дистанционного управления персональным компьютером существует немного, и все они имеют ограниченный функционал.

Данное мобильное приложение для платформы Android позволяет управлять некоторыми возможностями персонального компьютера. В такой функционал входит: